Titre: Title:	Évaluation expérimentale d'environnements informatiques pour le prototypage rapide de systèmes de commande
Auteur: Author:	Cédric Demers-Roy
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Demers-Roy, C. (2004). Évaluation expérimentale d'environnements informatiques pour le prototypage rapide de systèmes de commande [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7474/

NOTE TO USERS

UNIVERSITÉ DE MONTRÉAL

Évaluation expérimentale d'environnements informatiques pour le prototypage rapide de systèmes de commande

CÉDRIC DEMERS-ROY
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

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

Your file Votre référence
ISBN: 0-612-97939-3
Our file Notre référence
ISBN: 0-612-97939-3

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

Évaluation expérimentale d'environnements informatiques pour le prototypage rapide de systèmes de commande

présenté par: DEMERS-ROY Cédric
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. GOURDEAU Richard, Ph.D., président
M. HURTEAU Richard, Ph.D., membre et directeur de recherche
M. DESANTIS Romano, Ph.D., membre et codirecteur de recherche
M. FORTIN Clément, Ph.D., membre

REMERCIEMENTS

J'aimerais d'abord et avant tout remercier ma famille pour m'avoir poussé à poursuivre mes projets même lorsque la motivation faisait défaut. Merci à ma mère, Jocelyne, pour sa présence, son réconfort. Merci à mon père, Jean, pour sa présence, ses conseils. Un grand merci aussi à celle que j'aime, Chantale, pour m'avoir écouté parler de mon projet, pour m'avoir supporté avec amour même dans les moments plus difficiles. Un grand merci aussi à mes directeurs, messieurs Richard Hurteau et Romano M. DeSantis, sans qui ce projet n'aurait même pas existé et surtout sans qui il n'aurait certainement pas abouti dans sa réalisation actuelle. Merci aussi au personnel technique du département, particulièrement à madame Marie-Lyne Brisson et à monsieur Richard Grenier, pour avoir fourni le support technique nécessaire au projet. Je remercie finalement mes collègues de travail et mes amis pour leur soutien et leur aide lorsque nécessaire. Un projet de génie est plus une collaboration qu'une réalisation personnelle, je remercie donc tous ceux qui m'ont aidé ou qui y ont contribué.

RÉSUMÉ

Ce projet consiste principalement à explorer les possibilités de xPC Target, un environnement informatique inclus dans la suite Matlab, dans le cadre de l'implantation de prototypes de laboratoire de systèmes de commande en temps réel. L'analyse de l'environnement implique l'exploration des: modalités d'utilisation; possibilités offertes; caractéristiques d'opération; paramètres comme le coût, le temps d'implantation et l'expertise requise; éventuelles limites. L'approche utilisée est par voie d'étude de cas en simulation et en expérimentation sur deux systèmes: un banc d'essai composé d'un moteur à courant continu et de volants d'inertie liés par un lien élastique pour un asservissement en position et le guidage en téléopération d'un véhicule électrique. Le premier cas illustre une boucle de contrôle à un niveau et permet une étude comparative avec un autre environnement bien connu dans le domaine du contrôle: LabVIEW. Le deuxième cas est un cas plus complexe d'un contrôle à deux niveaux qui permet la téléopération du véhicule via un lien réseau sans fil. L'étude comparative ne s'étend pas à ce cas, mais une brève correspondance avec la librairie MICROB de l'IREQ est effectuée dans le cadre d'un rapport technique sur le sujet. Les résultats obtenus sont multiples. D'abord, le projet fournit un ensemble de considérations générales sur les aspects à explorer lors de l'étude d'un environnement pour la conception et l'utilisation d'un contrôleur en temps réel. Ces considérations sont appliquées et approfondies pour l'analyse de xPC Target et de LabVIEW dans le contexte d'utilisation du projet. Ensuite, le projet permet une diffusion de xPC Target dans les laboratoires de la section Automation et Systèmes de l'École Polytechnique de Montréal, ce qui fournit un nouvel environnement informatique de contrôle pour la formation des ingénieurs. D'ailleurs, cette diffusion a déjà permis de compléter plusieurs projets: contrôle en température d'un four électrique, contrôle d'un pendule inversé, manoeuvres assistées de stationnement pour un véhicule électrique, contrôle d'un palan électrique. Le travail réalisé spécifiquement dans le cadre de l'atteinte de l'objectif de cette recherche est présenté dans ce mémoire qui inclut de nombreuses références à des rapports techniques écrits sur le sujet (certains sont encore en cours de réalisation).

ABSTRACT

This project is about analysing a commercial software environment used to create control prototypes in control system laboratories xPC Target. This product is part of the well-known Matlab suite which is used by many engineers. xPC Target is used as an operating environment as well as a development environment. To analyse the product implies exploring: usage modalities; offered possibilities; operating caracteristics; parameters like cost, realization time and required expertise; potential limits. This will be done using case studies in simulation and real control for two systems: first is position control on a test bench composed of a DC motor and two inertia wheels with an elastic link; second is distant operation for an electric vehicle. The first case shows a single level control loop that serves for a comparative study with another well-known product: LabVIEW. The second case is more complex using a two levels control loop which allows distant operation of the vehicle on a wireless network. The comparative study doesn't extend to this case but a short correspondance with the MICROB library from IREQ is done in a technical report on this subject. Results in this project are abundant. First, it presents a set of general considerations about what to explore while studying a software environment for the design and usage of a real time controller. These considerations are applied and deepened while analysing xPC Target and LabVIEW in the working context of this project. Secondly, this work allows the usage of xPC Target in the laboratories of the section (Automation et Systèmes, École Polytechnique de Montréal), which gives a new computer environment for control which can be used in the training of new engineers. The implementation already lead to the completion of many projects: temperature control of an electrical oven, position control for an inverted pendulum, automatic parking for an electrical vehicle, position control for an electrical crane. The work done to attain the goal of this project is presented in this thesis which include many references to completed and uncompleted technical reports on related subjects.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTES DES SIGLES ET ABRÉVIATIONS

A	Unité (Ampère)
CC	Courant continu
DOS	*Disk Operating System*
E/S	Entrée(s) / Sortie(s)
Hz	Unité (Hertz)
IEEE	*Institute of Electrical and Electronics Engineers*
kg	Unité (kilogramme)
kHz	Unité (kiloHertz)
Mbps	Unité (Mégabits par seconde)
MHz	Unité (MégaHertz)
mm	Unité (millimètre)
ms	Unité (milliseconde)
OS	*Operating System*
PC	*Personal Computer* (ordinateur personnel)
RTW	*Real Time Workshop*
SE	Système d'exploitation
us (µs)	Unité (microseconde)
VDC	*Voltage Direct Current* (tension en courant continu)

Introduction

Dans le domaine de l'ingénierie, plus particulièrement en recherche sur les algorithmes de contrôle, le choix d'un environnement informatique pour les essais en simulation et en expérimentation est particulièrement important. Les milieux académique et industriel comportent déjà une multitude d'environnements disponibles et leur nombre augmente constamment. Comment choisir celui qui convient aux objectifs d'un projet donné? Lors d'un sondage mené en 1999 par la firme ITRON [1,2], on a demandé à plusieurs chercheurs de spécifier le critère qui était le plus important lors de la sélection de l'environnement informatique de leur projet. La figure 0.1 est directement tirée des résultats de ce sondage (en langue originale anglaise). Notez que ce sondage limite les réponses à la partie *exploitation* de l'environnement de travail complet tel qu'il sera présenté plus loin dans ce mémoire.

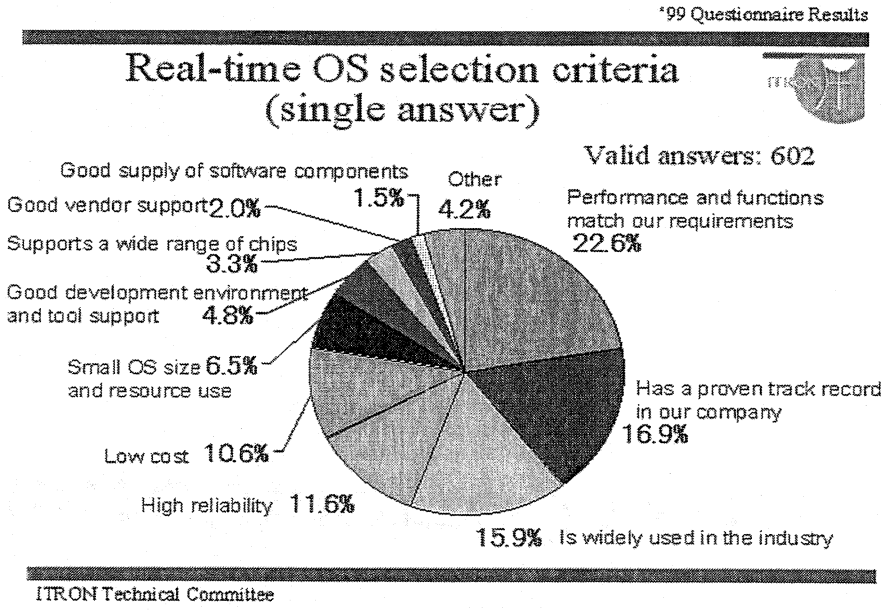


Figure 0.1: Critère de sélection pour un système d'exploitation (sondage 1999)

Une analyse rapide de cette figure montre que trois choix totalisent plus de 50% des réponses comme premier critère de sélection d'un environnement d'exploitation:

1) utiliser un produit qui remplit les besoins en termes de performances et de fonctionnalités (23 %)

2) utiliser un produit avec lequel les usagers sont déjà familiers ce qui permet de réduire le temps d'apprentissage. (17 %)
3) utiliser un produit couramment employé dans l'industrie afin de tirer profit de l'expertise ainsi développée (16 %)

Le premier critère en ordre d'importance est logique puisque les considérations techniques sont nécessaires pour déterminer la convenance aux besoins; du moins c'est ce que croient 23 % des chercheurs visés par le sondage. Si l'environnement fournit effectivement les performances attendues dans le cadre du projet en cours (ce qui peut ne pas être le cas selon les particularités du projet), le choix est effectif. Concernant les critères 2 et 3 en ordre d'importance (pour 33 % en tout), on peut émettre le commentaire suivant: ce n'est pas parce que l'environnement a déjà été utilisé par le groupe de recherche ou parce que d'autres l'utilisent que ce dernier convient au projet actuel.

Ce qui est certain c'est qu'il arrive fréquemment que l'environnement informatique en question ne convienne pas à la tâche spécifique et que le travail doive être recommencé en partie ou en totalité en cours de projet. Bien que généralement l'équipe arrive à atteindre ses objectifs avec l'environnement informatique retenu, est-ce qu'une économie de temps et d'efforts n'auraient pas pu être faite? Souvent la question du choix de l'environnement est esquivée pour concentrer l'attention sur le développement des algorithmes reliés directement au projet et pour obtenir le plus rapidement possible des simulations concluantes. À savoir si ces simulations seront pertinentes dans le cas pratique, le questionnement arrive en fin de projet et parfois le capital temps / efforts est presque épuisé et cette dernière étape est réalisée en toute hâte.

Objectif

L'objectif principal de cette recherche est d'explorer les possibilités offertes par xPC Target, un environnement informatique temps réel pour le prototypage expérimental rapide d'algorithmes de contrôle à partir de modèles Simulink. En particulier, on explore en détails les possibilités offertes par cet environnement en simulation et en expérimentation de systèmes de commande. Cette exploration consiste à analyser ses

qualités au niveau: des modalités d'opération; des possibilités offertes; des caractéristiques d'opération (performance, interface, etc.); des paramètres généraux comme le coût, le temps d'implantation, l'expertise requise; ainsi que l'exploration de ses éventuelles limites. L'environnement xPC Target est choisi, car il appartient à la suite Matlab qui est couramment utilisée dans le milieu académique pour les simulations de contrôleurs et qu'il est profitable pour tous de l'étudier. Comme point de comparaison, LabVIEW sera utilisé. LabVIEW est un logiciel utilisé surtout en industrie pour les mesures de tests et leur visualisation. La comparaison permettra de mieux départager les performances de chaque environnement informatique au niveau des questions posées.

L'objectif secondaire est de fournir des bancs d'essai complets dont pourront profiter les laboratoires de la section Automation et Systèmes de l'École Polytechnique de Montréal. D'ailleurs, l'aspect comparatif de l'étude demande l'implantation du premier banc d'essai dans les deux environnements de travail, ce qui permet de fournir deux implantations équivalentes complètes d'un même système.

Méthodologie

Afin d'atteindre l'objectif, la voie directe est employée soit l'implantation expérimentale de systèmes de commande représentant le plus possible des cas usuels retrouvés dans le milieu de l'automatisation de procédés. Ce système sera en premier lieu un banc d'essai en asservissement de position, puis en second lieu un véhicule mobile offrant une possibilité de téléopération. Chaque cas est différent: le premier utilise une boucle de contrôle simple (à un seul niveau) alors que le deuxième nécessite l'emploi d'une boucle plus complexe à deux niveaux. L'étude comparative entre xPC Target et LabVIEW se fera dans le premier cas. La boucle à deux niveaux demande beaucoup plus de travail de conception et d'implantation, un travail qui ne sera réalisé que dans l'environnement qui offre les meilleures possibilités selon les résultats obtenus lors de la première étude de cas.

Ainsi, en fin de projet, trois implantations expérimentales complètes seront disponibles: deux pour le banc d'essai et une pour le véhicule. Chacune de ces

implantations doit regrouper la simulation et l'expérimentation afin de fournir un environnement d'essai complet pour l'analyse des algorithmes de contrôle. L'analyse de chaque implantation doit se faire selon plusieurs critères qualitatifs et quantitatifs afin de définir dans la mesure du possible lequel des environnements informatiques correspond le mieux à la tâche.

L'avantage d'une étude expérimentale est que le milieu immédiat profite largement des développements reliés à ce projet. Puisque l'exploration réalisée concerne l'environnement plutôt que le système contrôlé, les environnements étudiés et appliqués aux systèmes à l'étude sont utilisables non seulement dans le cadre de la recherche actuelle, mais aussi pour la formation académique et pour les travaux d'autres chercheurs. Les systèmes implantés lors des études de cas proposées dans cette recherche peuvent être utilisés soit pour le prototypage d'algorithmes de commande (utilisation directe des systèmes développés), soit comme exemples complets de conception d'une boucle de contrôle incluant la simulation et l'expérimentation dans un seul et même environnement informatique.

La problématique de sélection d'un environnement informatique a aussi l'avantage de concerner la quasi totalité des projets en contrôle de procédé lorsque vient le temps de faire des études expérimentales: trouver un environnement aux performances éprouvées et avoir les outils et / ou les résultats d'analyse pour déterminer si cet environnement convient au projet actuel.

L'atteinte de l'objectif n'est pas simple. Tout d'abord, cette atteinte implique de se familiariser avec deux environnements informatiques afin d'en extraire des éléments de réponse en terme d'utilisation générale et de possibilités offertes. Elle implique aussi de bien assimiler les possibilités de ces environnements pour créer un environnement de test d'algorithmes robuste et fiable. De plus, il faut traiter avec le système de commande à l'étude en comprenant / concevant sa structure mécanique et électronique et en comprenant / développant les modèles correspondants pour leur simulation et pour leurs entrées / sorties (E/S) réelles. Ces systèmes étant composés de plusieurs constituants de natures variées (structure mécanique, éléments électroniques, éléments informatiques,

éléments d'interface), leur utilisation implique elle aussi une bonne connaissance de leurs caractéristiques. Tous ces éléments complexifient la tâche, mais en bout de ligne l'expérience acquise dans le domaine de l'analyse expérimentale de systèmes de commande et en sélection d'un environnement informatique est profitable.

Structure du mémoire

Ce mémoire est divisé en cinq chapitres. Le premier concerne la présentation de la problématique reliée à l'implantation d'une boucle de contrôle pour un système de commande usuel et présente les points d'analyse utilisés pour comparer les environnements informatiques retenus. Le deuxième chapitre présente les deux environnements utilisés pour la première étude de cas qui, elle, est présentée au chapitre trois. Cette première étude de cas concerne l'implantation d'une boucle de contrôle à un niveau pour la commande en position d'un banc d'essai en asservissement de position. Le quatrième chapitre traite de la deuxième étude de cas: un véhicule mobile dont le contrôle s'effectue en téléopération, d'où l'utilisation d'une boucle de contrôle à deux niveaux. Le cinquième et dernier chapitre présente les points d'intérêts futurs associés à ce projet, notamment l'utilisation de xPC Target dans le cadre de plusieurs autres projets de recherche et l'analyse de ses limites d'utilisation.

Chapitre 1 : Problématique

Afin de mieux diriger l'analyse, il faut d'abord poser les bases de comparaison pour les environnements qui seront étudiés. Pour ce faire, la notion de système de commande est présentée, puis les critères d'analyse des environnements informatiques sont définis.

1.1 Structure générale d'un système de commande

De manière générale, un système de commande est composé d'un système électromécanique à asservir et du matériel électronique et informatique nécessaire pour réaliser l'asservissement en question. La figure 1.1 montre un exemple général de boucle d'asservissement. On y retrouve les éléments généraux constituant la boucle, ainsi que l'interaction entre la boucle et l'extérieur via une interface opérateur et une base externe de données. Un ordinateur est utilisé dans la boucle comme contrôleur afin de pouvoir implanter facilement différents algorithmes de contrôle. Cela implique la présence des interfaces nécessaires entre l'ordinateur et le procédé. Le contrôle illustré à la figure 1.1 est direct, c'est-à-dire que l'opérateur communique directement le point d'opération à la boucle de contrôle.

Si des capteurs externes se greffent ou si le problème nécessite une structure décisionnelle complexe, la boucle à un niveau ne suffit plus à la tâche car plusieurs contrôleurs ayant chacun leur tâche vont se partager la commande. La structure de la figure 1.2 illustre le cas d'une boucle à deux niveaux où le point de consigne ne vient plus de l'opérateur, mais plutôt d'un deuxième niveau de contrôle qui lui se base non seulement sur l'opérateur, mais aussi sur des capteurs externes. Ce deuxième type de boucle est de plus en plus répandu, notamment pour la téléopération, et ce cas sera considéré dans l'analyse de l'environnement informatique. La présence d'un deuxième ordinateur implique de nouveaux besoins en termes de communication et d'échange de données.

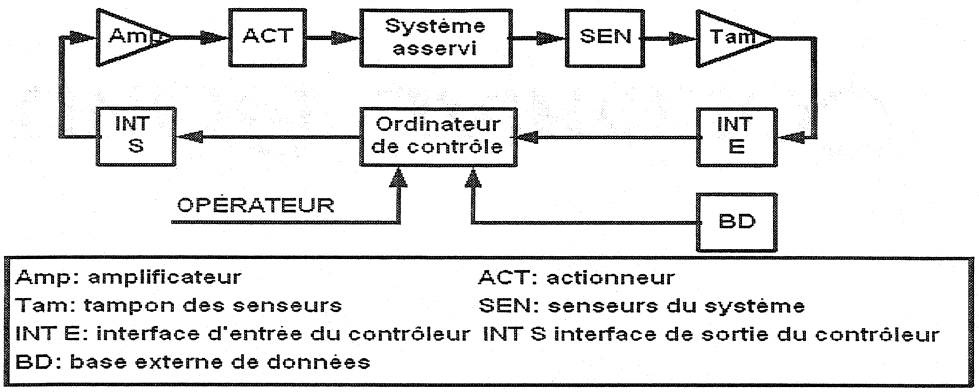


Figure 1.1: Boucle générale d'asservissement à un niveau

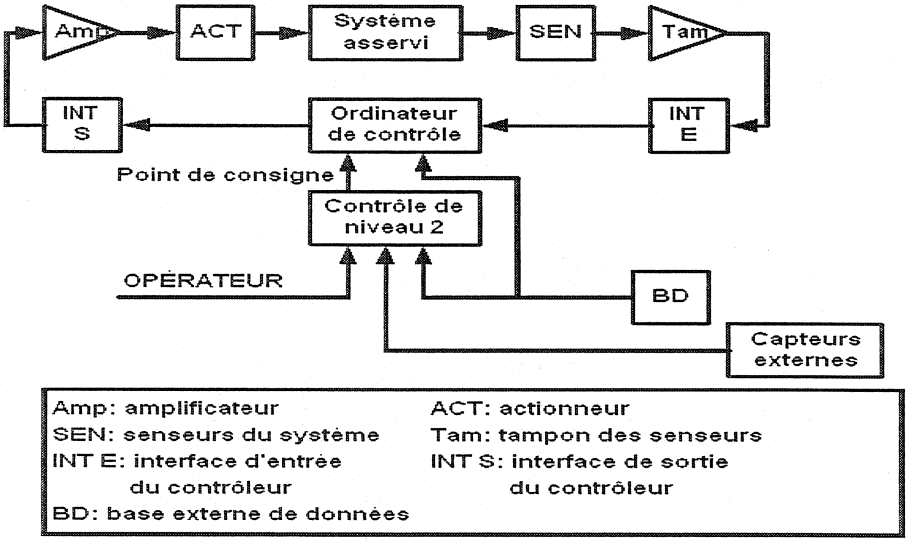


Figure 1.2: Boucle générale d'asservissement à deux niveaux

L'intérêt d'inclure un deuxième, voir même un troisième, niveau de contrôle est d'augmenter les possibilités de la boucle de contrôle. La figure 1.2 illustre l'introduction de capteurs externes qui informent le contrôleur de l'état du système. Un exemple de ce type de capteur est l'implantation d'un système de vision pour des procédés impliquant des déplacements ou des manipulations dans un environnement dit « dynamique ».

Que ce soit l'une ou l'autre des boucles de commande qui est à implanter, l'environnement retenu doit pouvoir permettre l'implantation à la fois en simulation et en expérimentation du bloc *système asservi* présent sur chaque figure. Cette mention

implique aussi que le passage d'un mode à l'autre doit pouvoir se faire sans recompilation du code et sans devoir recommencer la conception d'aucun élément de la boucle. Plus le système contrôlé est complexe, plus ce point est important car il y a toujours des divergences entre la démonstration théorique (simulation) et pratique (expérimentation) d'un algorithme donné. Si le passage entre les deux types de démonstration peut se faire rapidement, alors l'exploration d'un algorithme donné sera à la fois rapide et efficace et l'objectif de prototypage rapide sera atteint.

1.2 Cycle de développement d'un algorithme de contrôle

Le processus de conception en génie n'est pas simple, il s'échelonne sur plusieurs étapes parfois même réalisées au sein de plusieurs équipes qui travaillent en parallèle. La figure 1.3 (traduction, figure originale provenant du site de *The Mathworks* [3]) illustre une boucle typique de conception: à gauche le problème à résoudre, à droite le produit fini à commercialiser. L'environnement informatique complet à retenir est celui qui couvre le mieux possible la totalité des étapes prévues par ce diagramme allant de la modélisation du système à la production du code embarqué pour le produit final. Remarquez les liaisons orientées: certaines étapes nécessitent un retour en arrière lors de la conception. Si l'environnement est suffisamment complet pour mener la conception du début à la fin, alors les possibilités de travail en parallèle sont accrues et le cycle de développement est accéléré, tout en simplifiant la tâche d'intégration des différentes sections de codes participant à la réalisation complète. Dans le cas de ce projet, seul le prototypage d'algorithmes de commande est traité; la production de masse du code pour un contrôleur à commercialiser ne fait pas partie de l'objectif d'exploration de l'environnement informatique. Ainsi, la flèche complètement à droite du diagramme qui concerne la production ne s'applique pas dans les considérations de choix bien que la case SE *(système embarqué)* y participe pour les tests dans l'environnement d'exploitation fonctionnant en temps réel.

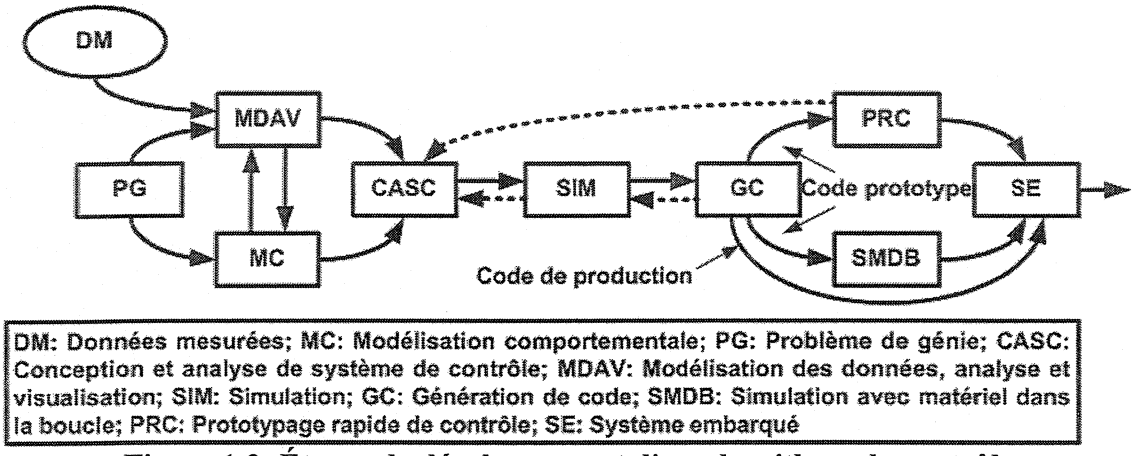


Figure 1.3: Étapes de développement d'un algorithme de contrôle

1.3 Considérations pour le choix d'un environnement informatique

Le domaine de l'automatisation et du contrôle de procédé regroupe peu de références (articles, conférences, livres) traitant de la comparaison entre différents environnements informatiques complets[1] pour le contrôle en temps réel de systèmes de commande. Plusieurs articles comme [4] et [5] vantent les mérites respectifs d'un environnement temps réel en particulier, mais ce dernier est alors dédié au contrôle du procédé sans prévoir à la base les outils de simulation et d'analyse de données nécessaires aux procédures de test d'un algorithme de commande. Il semble que le problème de fournir un environnement informatique complet n'ait pas reçu l'attention souhaitable, d'où l'objectif actuel de recherche de valider un de ces environnements.

Pour effectuer une sélection efficace de l'environnement informatique qui prenne en considération le problème complet dès les premières étapes de l'étude, l'analyse est divisée en cinq points importants:

1) compatibilité des éléments de la boucle;
2) protocole d'échange de données;
3) environnement de conception;

1Un environnement informatique complet regroupe les environnements de conception, d'exploitation et d'opération définis plus loin dans ce mémoire; et ce autant en simulation qu'en expérimentation.

4) environnement d'exploitation;

5) environnement d'opération.

Ces points peuvent être analysés un à un, mais il est clair que tous doivent être pris en compte lors de la sélection finale de l'environnement de travail. Par exemple, un environnement différent peut être sélectionné pour chacune des étapes du développement expérimental d'un algorithme (conception / exploitation / opération). Il est cependant préférable d'utiliser le même environnement pour toutes les étapes de façon à diminuer les transpositions entre les différents environnements et réduire les efforts nécessaires pour passer d'une étape à l'autre du cycle de prototypage de l'algorithme de commande à l'étude.

1.3.1 Compatibilité des éléments de la boucle

Par « compatibilité des éléments de la boucle », on entend que la boucle doit pouvoir être fonctionnelle. Ceci nécessite que tous les éléments constituant la boucle peuvent être insérés avec la possibilité d'en modifier les paramètres de fonctionnement sans programmation supplémentaire.

Par exemple, si le montage utilise un encodeur optique dont les impulsions sont comptées par une carte d'interface qui conserve les données dans une mémoire tampon de 64 bits, il faut pouvoir effectuer la lecture de ce tampon dans l'environnement d'exploitation retenu. Dans le même ordre d'idée, si la carte communique via un bus de données PC-104 [6, 7], l'environnement informatique doit supporter ce type de bus. Il s'agit d'un point d'analyse trivial qu'il faut absolument considérer lors de la sélection initiale, car un oubli à ce niveau implique une nouvelle sélection matérielle et un nouvel effort de conception.

Si le pilote d'un élément nécessaire à la boucle n'est pas disponible, on détermine à ce point si la programmation maison est envisageable ou non. La possibilité de le faire dépend de la disponibilité du code source et des conventions de programmation pour l'environnement informatique en question. Si l'environnement est de type entièrement fermé, il est difficile de programmer un pilote maison et de le rendre compatible.

1.3.2 Protocole d'échange de données

L'analyse des protocoles d'échange de données implique l'étude des possibilités de communication entre les éléments de la boucle (contrôleur / système par exemple) et entre la boucle et l'extérieur (contrôleur / opérateur par exemple). Plusieurs types d'échanges existent: lien série RS-232, FireWire ou USB, lien parallèle IBM, lien réseau TCP/IP, lien sans fil avec Technologie BlueTooth ou 802.11b, média amovible (disque dur amovible, disquette), etc.

Les protocoles d'échange de données sont très importants à considérer, d'abord parce que les différents éléments de la boucle ont chacun leur protocole qu'il faut supporter, mais surtout car il faut pouvoir communiquer avec la boucle de contrôle lors des essais. Sachant qu'un ordinateur est utilisé comme contrôleur et que cet ordinateur n'est pas nécessairement le poste de développement, la communication avec la boucle est inévitable: d'abord pour transférer l'algorithme, mais aussi lors de son exécution pour pouvoir connaître les états du système et du contrôle.

Dans l'un ou l'autre des cas, il est possible de procéder par transfert direct des fichiers, il suffit d'avoir une interface de transfert comme un disque dur ou un média mobile comme une disquette ou un disque compact (CD-R). Cependant, ceci implique une interaction supplémentaire avec le système et les interfaces ne peuvent pas toujours être présentes sur un système mobile comme un véhicule.

Il est préférable de passer par un lien direct de données pour procéder aux échanges. Habituellement, trois choix se présentent: lien série, lien parallèle et lien réseau. Chaque type de lien regroupe plusieurs protocoles disponibles mais deux sont particulièrement connus pour l'échange d'informations: RS-232 (lien série) et TCP/IP (lien réseau). Dans la majorité des cas, le lien TCP/IP est retenu car il regroupe la rapidité (typiquement 10 ou 100 Mbps), la fiabilité (gestion fiable des paquets) et la simplicité (permet le branchement en réseau; pas de connexion directe point à point nécessaire).

1.3.3 Environnement de conception

L'environnement de conception est celui où la programmation de la boucle de contrôle est effectuée. Son étude implique entre autres l'analyse des possibilités suivantes: implantation des modules de simulation, implantation des modules d'expérimentation, qualité de l'interface de programmation, possibilités de déverminage, clarté des programmes résultants.

Le but premier est d'avoir un environnement de conception qui permet un passage le plus direct possible entre les essais en simulation sur un modèle mathématique et en expérimentation sur le système réel. Cette possibilité permet de rapidement analyser la correspondance théorie / pratique dans les systèmes de commande.

De manière générale, l'environnement de conception doit fournir tous les outils mathématiques et analytiques nécessaires à l'implantation des algorithmes de contrôle pertinents. Plus cet environnement sera facile à utiliser, meilleures seront les possibilités d'utilisation par le plus grand nombre de concepteurs. Typiquement, un environnement graphique de conception offre beaucoup plus d'attrait qu'un environnement purement textuel en présentant des librairies de blocs fonctionnels, des liaisons de données orientées et une illustration graphique de la boucle réalisée. Cependant, ce type d'environnement très graphique rend l'usager dépendant de la disponibilités des blocs. Ce désavantage est éliminé s'il y a possibilité de créer facilement ses propres blocs. Un environnement textuel de conception offre une mauvaise vue globale de la boucle et est typiquement beaucoup plus difficile à apprendre et à utiliser, bien que ceux qui le maîtrisent le préfère habituellement à un environnement graphique. L'objectif sur ce point est de valider la faisabilité de l'implantation tout en analysant la simplicité d'utilisation en général.

1.3.4 Environnement d'exploitation

L'environnement d'exploitation est celui où la boucle de contrôle est exécutée en temps réel et dans lequel l'algorithme de contrôle doit remplir la tâche qui lui est assignée. Son étude implique entre autres l'analyse des possibilités suivantes: stabilité

temporelle, rapidité d'exécution et fiabilité générale de fonctionnement de l'environnement. Les données sont recueillies dans cet environnement lors de l'exécution de la boucle de contrôle.

Il est difficile d'atteindre de bonnes performances en contrôle si l'environnement d'exploitation n'a pas lui-même de bonnes performances en termes de stabilité temporelle, de rapidité d'exécution et de fiabilité d'exécution. Ces trois paramètres fixent directement les possibilités de contrôle et il est très important de les mesurer pour définir si l'environnement d'exploitation fournit les performances nécessaires. Peu de logiciels offrent directement leur propre environnement d'exploitation et le choix de ce dernier se fait habituellement en parallèle en s'assurant qu'il est compatible avec le logiciel retenu pour les environnements de conception et d'opération. Il faut faire ce choix dès le départ puisque la conception des modèles en dépend largement.

1.3.4.1 Stabilité temporelle

La stabilité temporelle est la propriété principale à rechercher pour l'obtention d'un contrôle en temps réel. Un fonctionnement en temps réel implique que l'instant d'échantillonnage est exact: ni plus rapide ni plus lent que la période demandée.

Tous les algorithmes de contrôle en boucle fermée dépendent des états actuels du système et de la mesure de leur évolution dans le temps. Souvent, la base de calcul est précisément le moment auquel chaque mesure a été prise. Par exemple, le calcul de premier ordre de la vitesse à partir d'une mesure de position: $v_k = \frac{p_k - p_{k-1}}{T}$ avec v_k la vitesse à l'instant k; p_k la position à l'instant k; p_{k-1} la position à l'instant k-1 et T la période d'échantillonnage:

1) si la valeur de T n'est pas stable, alors le calcul de vitesse sera erroné;
2) si le moment de la mesure de la position ne correspond pas à l'instant exact d'échantillonnage, la mesure n'est pas représentative de l'état actuel du système.

Le même raisonnement s'applique pour tout calcul impliquant le temps: un filtre, l'intégration d'un signal, le calcul d'une durée, etc.

Contrairement à la majorité des critères présentés dans cette section, la stabilité est mesurable quantitativement. Les modèles d'évaluation incluront dans la boucle un dispositif de mesure simple implanté à cet effet: un oscillateur unitaire qui change d'état à chaque itération de la boucle de commande et qui est relié à une sortie du système physique. L'analyse du signal ainsi produit avec un oscilloscope va permettre l'évaluation précise de la stabilité de la période d'échantillonnage.

1.3.4.2 Rapidité d'exécution

La rapidité d'exécution est représentée par la plus petite période d'échantillonnage que l'environnement d'exploitation peut utiliser pour un contrôleur donné. Certaines applications demandent une rapidité minimale qui devient alors un critère pour valider ou non le choix de l'environnement d'exploitation.

Physiquement, le fait d'inclure un ordinateur comme contrôleur implique nécessairement l'échantillonnage à instant précis (discrétisation temporelle), la conversion d'analogique à numérique des signaux d'entrée et la conversion de numérique à analogique des signaux de sorties. Normalement, il faut alors utiliser des algorithmes développés dans le contexte de la théorie de la commande discrète. Or, les études d'algorithmes de contrôle utilisent souvent l'hypothèse que la boucle de commande est de type continue plutôt que discrète. Pour que cette hypothèse de travail soit valide il faut vérifier deux paramètres de la boucle:

1) le contrôleur doit pouvoir reconstruire avec fidélité tous les signaux d'entrée;
2) il faut pouvoir agir suffisamment rapidement pour que la dynamique du système ne réagisse pas au caractère échantillonné de la commande.

Pour le premier point, cela implique l'utilisation d'une période d'échantillonnage au moins de cinq à dix fois plus petite que la plus petite constante de temps du système. L'exemple d'un moteur à courant continu illustre l'importance du deuxième point: la période d'échantillonnage doit être suffisamment petite pour ne pas obtenir une vitesse saccadée en sortie lors d'une rampe de tension. Si la rapidité d'exécution permet de respecter ces contraintes et que la stabilité mentionnée au paragraphe précédent est elle

aussi assurée, alors l'hypothèse de commande en continu est valable et les méthodes de développement peuvent être choisies en conséquence. L'importance du critère de rapidité pour l'environnement d'exploitation s'étend au fait que la tendance en contrôle se dirige vers des systèmes de plus en plus rapides avec des tâches de plus en plus complexes.

1.3.4.3 Fiabilité générale de l'environnement

La fiabilité générale se mesure par l'aptitude de l'environnement d'exploitation à garder un niveau de performance constant, indépendamment des perturbations extérieures.

La stabilité temporelle et la rapidité sont prioritaires, mais encore faut-il que le système puisse supporter une exécution prolongée et sans interruption au maximum de ses capacités sans dégradation de performance et surtout sans devenir instable. Un environnement d'exploitation multitâches comme Windows est un bon exemple inverse de cette propriété. Le travail multitâches demande de la mémoire, or lorsqu'elle est pleine l'ordinateur adopte un comportement erratique pendant qu'elle est nettoyée (principe de la mémoire cache sur le disque). Cet épisode occasionne un ralentissement imprévu des opérations et l'apparition de délais supplémentaires de traitement pour les processus en cours d'exécution. Lors du contrôle en temps réel de procédés, cet aspect imprévisible est très critique, car il peut survenir à tout moment et modifier substantiellement le comportement du système asservi. Si par malheur l'environnement d'exploitation fige (interruption complète des calculs et des échanges mémoires), alors le système n'est plus asservi et la situation peut même devenir dangereuse selon le système en question.

Un environnement d'exploitation fiable doit même prévoir l'inclusion de modules *chien de garde* (*watchdog*) pour assurer la sécurité. Ces modules assurent la gestion des entrées / sorties en prenant en considération des paramètres de sécurité comme la fonctionnalité du lien de communication et / ou le respect de contraintes sur des paramètres de fonctionnement comme la température ou la position des éléments

mécaniques du système asservi. Bien que laissé à la responsabilité du concepteur lors de la programmation des algorithmes, l'ajout des modules de garde doit être possible dans l'environnement d'exploitation et il est même préférable que certaines fonctions de sécurité soient assurées directement par l'environnement plutôt que par les algorithmes eux-mêmes.

En résumé, l'environnement d'exploitation doit être: temporellement stable, rapide en terme de fréquence d'échantillonnage, et globalement fiable au niveau de ses performances. Ces trois critères permettent de choisir un environnement capable de respecter les hypothèses de performances. D'autres critères qualitatifs d'analyse de l'environnement d'exploitation sont aussi utilisés lors de l'analyse:

- permet-il un fonctionnement en téléopération?;
- peut-on consulter les données pendant l'exécution de la boucle?;
- les paramètres d'exécution sont-ils modifiables pendant l'exécution de la boucle?
- l'environnement informe-t-il l'usager de son état de fonctionnement?

Ces questions décrivent le degré d'interaction entre l'opérateur et le système via les outils fournis par l'environnement informatique d'exploitation.

1.3.5 Environnement d'opération

L'environnement d'opération regroupe toutes les fonctions d'exécution de la boucle de contrôle, ainsi que les éléments disponibles pour la visualisation et l'analyse des données en cours d'exécution et / ou suite à cette dernière.

La mise à l'essai d'un algorithme implique nécessairement l'envoi de commandes au système et la récupération des données d'exécution. Lors des essais, le déroulement des tests est sous la responsabilité d'un opérateur qui communique des consignes à la boucle, qui gère son exécution et qui analyse les résultats. L'environnement d'opération est le produit final de la conception de la boucle d'un système asservi; il fournit l'interface entre l'opérateur et la boucle. Ce sont au départ les possibilités offertes par l'environnement de conception pour la création de l'interface qui vont fixer les qualités

de l'environnement d'opération.

L'interface la plus simple et directe est sans aucun doute celle de type *ligne de commande* qui est bien illustrée par l'utilisation du système d'exploitation commercial DOS, qui ne fournit que l'invite de commande pour permettre à l'usager d'interagir via des commandes textuelles comme *dir *.**, *copy file.txt file2.txt*, *move file.txt a:*, *rename file1.txt file2.txt*, etc.. Actuellement, des environnements d'opération de ce type sont de moins en moins appréciés dû à leur manque d'interactivité et à l'absence d'un aspect intuitif d'utilisation. L'opérateur usuel préfère une interface de type graphique avec des boutons de commandes identifiés, des fenêtres de navigation et des graphiques interactifs montrant l'évolution des données. L'opérateur peut alors se limiter à connaître l'effet des boutons et néanmoins faire fonctionner le système de manière efficace. Si en plus cet environnement d'opération inclut des protections pour éviter les commandes erronées, alors la sécurité s'en trouve accrue sans diminuer la simplicité d'utilisation.

Ce type d'interface est beaucoup plus facile à utiliser et demande beaucoup moins de temps d'apprentissage. Toutefois, sa conception demande beaucoup d'efforts et d'outils disponibles. Il faut se rappeler que sur une interface graphique, l'absence d'une commande la rend inutilisable dans le cas précis où cette commande est nécessaire. Ainsi, la simplification de l'utilisation peut entraîner un problème critique: la diminution des possibilités offertes.

Habituellement, une interface graphique offre moins de flexibilité (interaction limitée aux commandes prévues sur l'interface) qu'une interface de commande textuelle qui demande plus de connaissances au départ et est plus difficile à utiliser, mais dont l'opérateur contrôle mieux l'appel aux fonctions. Il s'agit de trouver un environnement d'opération qui offre le juste milieu entre ces trois éléments: simplicité, possibilités offertes et temps d'apprentissage. La figure 1.4 illustre cette

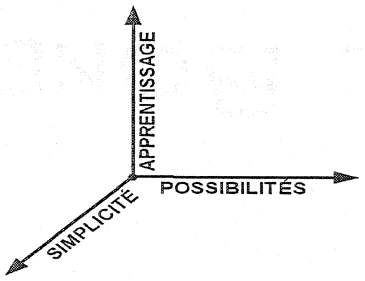


Figure 1.4: Paramètres de choix d'environnement

interaction sous la forme d'un système d'axes qui permet de mieux situer l'environnement d'opération à l'étude selon ces trois aspects.

De plus les mêmes points d'évaluation qualitatifs que pour l'environnement d'exploitation se posent pour l'environnement d'opération:

- permet-il un fonctionnement en téléopération?;
- peut-on consulter les données pendant l'exécution de la boucle?;
- les paramètres d'exécution sont-ils modifiables pendant l'exécution de la boucle?
- l'environnement informe-t-il l'usager de son état de fonctionnement?

Les mêmes points d'évaluations sont utilisés car en fait les deux environnements doivent pouvoir communiquer entre eux. Si l'environnement d'opération ne permet pas de profiter des avantages de l'environnement d'exploitation, alors quel est l'intérêt?

1.4 Résultats attendus

Dans ce travail, il n'existe pas de réponse unique aux problématiques étudiées. L'objectif est d'explorer les possibilités d'un environnement informatique complet (allant de la conception à l'opération) pour le prototypage rapide d'algorithmes de commande. Cette exploration doit se faire tout en se fixant une méthode pour évaluer la performance de l'environnement informatique à l'étude. Les résultats prennent la forme d'une liste de critères à analyser; ces critères sont appliqués à la classification de xPC Target par rapport à LabVIEW qui est utilisé comme environnement de comparaison. Les résultats proviennent de deux études de cas d'implantation. Le tableau[2] 1.1 résume les points d'analyse présentés dans ce chapitre.

Comme résultats secondaires directs, deux implantations physiques complètes de systèmes de commande seront disponibles suite aux études de cas: un banc d'essai en asservissement de position et un véhicule mobile téléopéré. Ces implantations pourront servir dans des recherches futures sur le contrôle de ces systèmes, mais aussi et surtout comme base d'apprentissage pour la conception de nombreux autres systèmes utiles à la

2 La taille des caractères dans le tableau est diminuée par rapport au texte pour permettre d'insérer le tableau dans le sens normal de la page sans occuper trop d'espace.

formation des ingénieurs et à la complétion d'autres projets de recherche.

Tableau 1.1: Résumé des critères d'analyse pour un environnement informatique

Élément d'analyse	Choix / Description
COMPATIBILITÉ DES ÉLÉMENTS DE LA BOUCLE	
Tous les éléments de la boucle sont disponibles	OUI / NON
Ajout d'élément possible par programmation	OUI / NON (degré de difficulté)
PROTOCOLE D'ÉCHANGE DE DONNÉES	
Types supportés	Série / parallèle / réseau / média mobile (disquette, CD, etc.)
ENVIRONNEMENT DE CONCEPTION	
Implantation combinée possible (simulation et expérimentation)	OUI / NON (possibilité d'implanter à la fois les simulations et l'expérimentation dans un même modèle sans devoir faire des modifications importantes)
Présence des outils d'analyse	OUI / NON
Facilité d'utilisation	Simple / moyen / difficile
Possibilité d'interface graphique	OUI / NON (simplicité d'intégration)
Type d'environnement	Graphique / textuel / combiné
Couverture du cycle de développement	Degré de couverture des étapes du diagramme présenté à la figure 1.3.
ENVIRONNEMENT D'EXPLOITATION	
Stabilité de la période d'échantillonnage	OUI / NON Critère quantitatif (mesure à l'oscilloscope de la stabilité d'un oscillateur qui change d'état à chaque itération de la boucle de commande).
Rapidité d'exécution	Critère quantitatif (mesure de la plus petite période d'échantillonnage applicable au système courant).
Fiabilité générale	Évaluation de la fiabilité générale de l'environnement d'exploitation: répétabilité, déterminisme, robustesse générale.
ENVIRONNEMENT D'OPÉRATION	
Type	Graphique / textuel / combiné
Possibilités	Commandes offertes, outils en ligne disponibles, analyse des données.

Élément d'analyse	Choix / Description
Temps d'apprentissage	Évaluation qualitative du temps d'apprentissage nécessaire pour faire fonctionner une interface d'opération typique développée dans l'environnement à l'étude.
Simplicité de création	Évaluation qualitative du temps nécessaire pour concevoir une interface d'opération complète dans l'environnement à l'étude.
Possibilités offertes	Téléopération (oui / non); consultation de données en cours d'exécution (oui / non); modifications des paramètres en cours d'exécution (oui / non); consultation des états de fonctionnement de l'environnement d'exploitation (oui / non).

Chapitre 2 : Présentation des environnements informatiques

Le choix définitif d'un environnement informatique est complexe, car plusieurs paramètres sont à étudier et certains demandent une analyse détaillée qui peut prendre beaucoup de temps. L'objectif présent est d'explorer deux de ces environnements dans un contexte expérimental qui doit à la fois supporter l'étude en simulation et en expérimentation de deux systèmes de commande. Pour cette recherche, xPC Target et LabVIEW ont été retenus.

2.1 Raison du choix de xPC Target et LabVIEW

Ces deux progiciels ont été retenus, car ils représentent deux options couramment utilisées dans le domaine de l'automatisation: Matlab est surtout utilisé en simulation et en analyse et LabVIEW est surtout utilisé en contrôle et en mesure dans un contexte industriel. Selon leurs spécifications techniques, chacun peut remplir l'objectif d'effectuer les simulations et les expérimentations dans le même environnement informatique.

Toutefois, chaque environnement est bien différent dans son contexte d'utilisation. LabVIEW fonctionne entièrement sous le système d'exploitation Windows, alors que xPC Target possède son propre système d'exploitation pour l'exécution des algorithmes de contrôle. Pourquoi les comparer entre eux alors? D'abord pour la raison mentionnée plus haut concernant leur utilisation répandue, mais aussi pour voir à quel point la nécessité d'un environnement temps réel est absolue. Tout environnement possède ses avantages et ses inconvénients qu'il est important d'explorer. Dans ce cas, l'on verra plus loin que le désavantage de xPC Target est qu'il est de type hôte / cible ce qui nécessite l'utilisation de deux ordinateurs alors que LabVIEW fonctionne entièrement sur un seul ordinateur. Si les performances des deux environnements sont équivalentes, alors ce sera sur ces avantages et inconvénients que le partage se fera.

2.2 Environnement Matlab / Simulink / xPC Target

Matlab est un logiciel de la société *The Mathworks* qui fait maintenant partie du bagage technique de nombreux chercheurs, ingénieurs et praticiens travaillant dans le domaine de la simulation des modèles dynamiques. Il regroupe plusieurs outils mathématiques pour l'analyse de signaux et pour la conception d'algorithmes numériques très avancés. Conséquence directe, la connaissance de ce logiciel entraîne le désir de vouloir l'utiliser pour la commande expérimentale lorsque les simulations sont complétées, ce qui est la raison même du choix initial de cet environnement pour cette recherche. La base de Matlab est un environnement de programmation dans un langage mathématique dédié à la conception d'algorithmes. Cet environnement a beaucoup évolué avec les années et il comporte maintenant plusieurs nouveaux outils qui peuvent simplifier le travail en commande de système asservis et en prototypage rapide.

Le progiciel Matlab se compose principalement de quatre sections : l'environnement principal permettant la programmation, M-Editor; l'extension graphique permettant la programmation par bloc, Simulink; l'extension de compilation pour l'exécution en temps réel, Real Time Workshop (RTW) et la cible de compilation en temps réel, xPC Target.

2.2.1 Fenêtre principale Matlab

La section Matlab du site Internet de *The Mathworks* [3] présente en détails ce produit avec ses nombreuses options et la littérature scientifique contient de multiples références sur ce logiciel. L'objectif de la recherche actuelle n'est pas la formation à l'utilisation générale de Matlab, donc seul un bref résumé est présenté.

À la base, Matlab est une interface intuitive de programmation comportant un langage principalement constitué de fonctions mathématiques, mais aussi de fonctions graphiques intégrées (contrairement à des langages comme le Fortran ou le C). Il utilise la notion de *boîte à outils* pour regrouper les fonctions propres à un domaine en particulier. On trouve par exemple la boîte *control* pour les fonctions reliées au contrôle de procédés; la boîte *graphics* pour la visualisation de données; la boîte *stateflow* pour la

création de machines à états finis; la boîte *matfun* pour les opérations sur les matrices; la boîte *uitools* pour la gestion des entrées / sorties, la boîte *ident* pour les fonctions d'identification de systèmes, etc. Il y a plusieurs dizaines de boîtes disponibles sur le site de la compagnie et Matlab offre même la possibilité d'intégrer des modules programmés dans un autre langage (C, C++, Java, Fortran) directement dans le code en langage M[3].

2.2.2 Simulink

La section Simulink du site Internet de *The Mathworks* [8] présente en détails les caractéristiques de Simulink; seules les principales sont retrouvées dans ce texte.

La fenêtre principale de Matlab demeure un environnement de programmation: le concepteur doit connaître les fonctions et leur syntaxe et la performance du code dépend grandement de ses connaissances du langage M. Afin de faciliter et d'accélérer le développement d'algorithmes, tout particulièrement en automatique, Matlab a inclus une extension graphique nommée Simulink qui permet la programmation par blocs de schémas de traitement. Ce type de programmation est beaucoup plus intuitif et permet un développement accéléré sans nécessairement connaître le détails des fonctions Matlab utilisées par l'extension graphique.

Simulink permet de construire un schéma-bloc, de simuler son comportement, d'évaluer ses performances et d'affiner sa conception. L'environnement s'intègre de manière transparente dans Matlab et fournit un accès immédiat à une vaste gamme d'outils d'analyse et de conception. Les outils sont regroupés à la manière de Matlab dans des boîtes à outils qu'il est possible d'acquérir selon ses besoins. La bibliothèque de fonctions peut ainsi regrouper un vaste choix de modules pour effectuer de nombreuses tâches distinctes: systèmes continus ou discrets, systèmes linéaires ou non linéaires, interfaces d'échange de données, routage et traitement de signaux, statistiques sur les données, source de données, interface de visualisation des données, etc. Encore une fois, plusieurs dizaines de boîtes sont disponibles selon les besoins de l'usager et si malgré cette grande disponibilité un bloc nécessaire n'existe pas dans la bibliothèque de

3 Le terme *langage M* est utilisé pour définir le langage de programmation propre à Matlab bien que cet épellation ne soit pas définie clairement dans la littérature.

fonctions, Simulink permet aussi la création de nouveaux blocs à l'aide de code en C/C++, Fortran, Java, etc. Ces blocs dynamiques peuvent essentiellement être liés à n'importe quel code suivant le respect d'un certain gabarit pour les données et pour les fonctions d'interface avec Simulink.

Simulink offre aussi la possibilité de regrouper des blocs en systèmes indépendants dans le schéma principal pour conserver une grande lisibilité et une bonne clarté du résultat final. Le logiciel supporte aussi les signaux de type BUS afin de regrouper plusieurs signaux sur une même ligne afin de clarifier encore plus le diagramme. Une utilisation efficace des fonctions de routage et d'échange de données permet de créer des algorithmes très complexes en très peu de temps, tout en conservant un maximum de lisibilité sur le résultat final d'implantation.

Puisque Simulink communique directement avec Matlab, l'analyse post-exécution est grandement simplifiée car le langage M permet l'analyse mathématique avant, pendant ou après l'exécution de la boucle. Cette facilité d'analyse fait de Matlab / Simulink un premier choix en matière de modélisation et de simulation dans le domaine de l'automatique. Dans la section Automation et Systèmes du département de génie électrique de l'École Polytechnique de Montréal, ce logiciel fait partie de la formation de tous les ingénieurs.

2.2.3 Real Time Workshop

Pour modéliser et simuler des algorithmes dans un contexte purement mathématique, Matlab et Simulink fournissent tous les outils dont un concepteur a besoin. Cependant, lorsque vient le temps d'utiliser les algorithmes dans un contexte physique de contrôle, la notion d'exécution en temps réel devient critique. Puisque Matlab est un logiciel de haut niveau conçu pour fonctionner sous un système d'exploitation complexe comme Windows, l'exécution en temps réel directement dans l'interface du logiciel devient impossible[4]. Il est important alors de fournir un outil qui

4 L'environnement d'exploitation Windows est multitâches, ce qui implique une mémoire et un temps de calcul partagés entre tous les programmes. Ce partage occasionne des délais imprévisibles qui empêchent l'exécution en temps réel.

permet d'exécuter les programmes selon des exigences temporelles rigoureuses, c'est-à-dire avec une période d'échantillonnage fixe et stable.

L'extension Real Time Workshop (RTW) est un outil de compilation des modèles Simulink et des fonctions Matlab pour permettre l'exécution indépendante du code dans un environnement dit *déterministe*[5]. Le détail de cette extension est présentée dans la section correspondante du site Internet de *The Mathworks* [9].

À la base, RTW est un générateur de code pour les modèles Simulink, ce qui implique qu'il est directement relié avec la conception d'un modèle dans Simulink en premier lieu. Lorsqu'il est invoqué, l'outil RTW permet une compilation du modèle pour en faire un code efficace qui peut respecter des contraintes de temps réel et qui peut s'exécuter seul dans un environnement indépendant, sans devoir utiliser l'environnement propre à Simulink. Le code généré est documenté directement par le logiciel afin de favoriser sa compréhension et permettre d'éventuelles modifications par le concepteur du modèle Simulink. Le code est au format C ANSI et il est optimisé, portable et ajustable à partir des modèles Simulink. La compilation peut se faire pour un système entier ou pour des sous-systèmes particuliers, à des fins de simulations temps réel ou pour des simulations avec matériel dans la boucle (*hardware-in-the-loop simulations*).

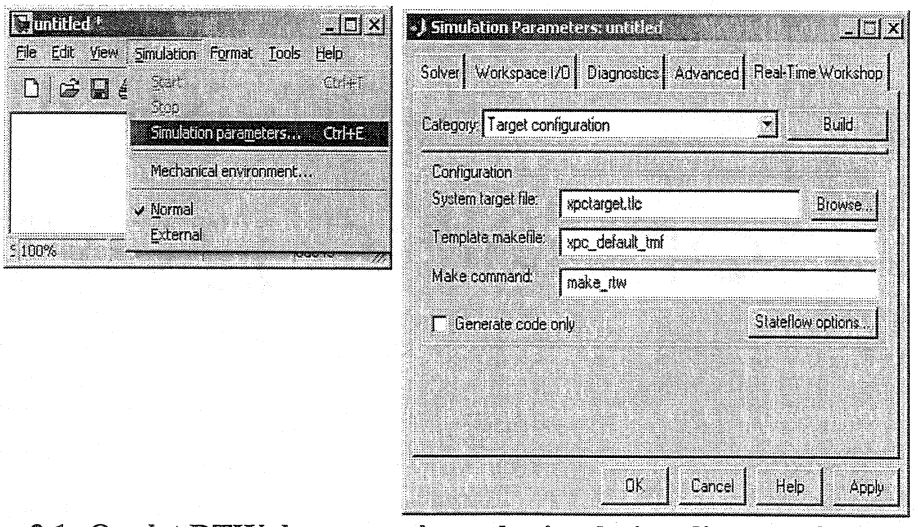


Figure 2.1: Onglet RTW des paramètres de simulation d'un modèle Simulink

5 Un environnement dit déterministe fonctionne nécessairement en temps réel.

Cet outil est accessible à partir des propriétés de tout diagramme Simulink comme l'illustre la figure 2.1. Sa configuration nécessite la spécification d'une cible qui sera utilisée lors de la compilation du modèle actif. Le résultat de compilation est alors compatible en particulier pour l'environnement spécifié dans la case *System target file*. L'option *Browse...* de la fenêtre permet d'afficher les cibles supportées par l'installation de RTW afin d'en choisir une. Il y a plusieurs cibles disponibles comme le montre la figure 2.2.

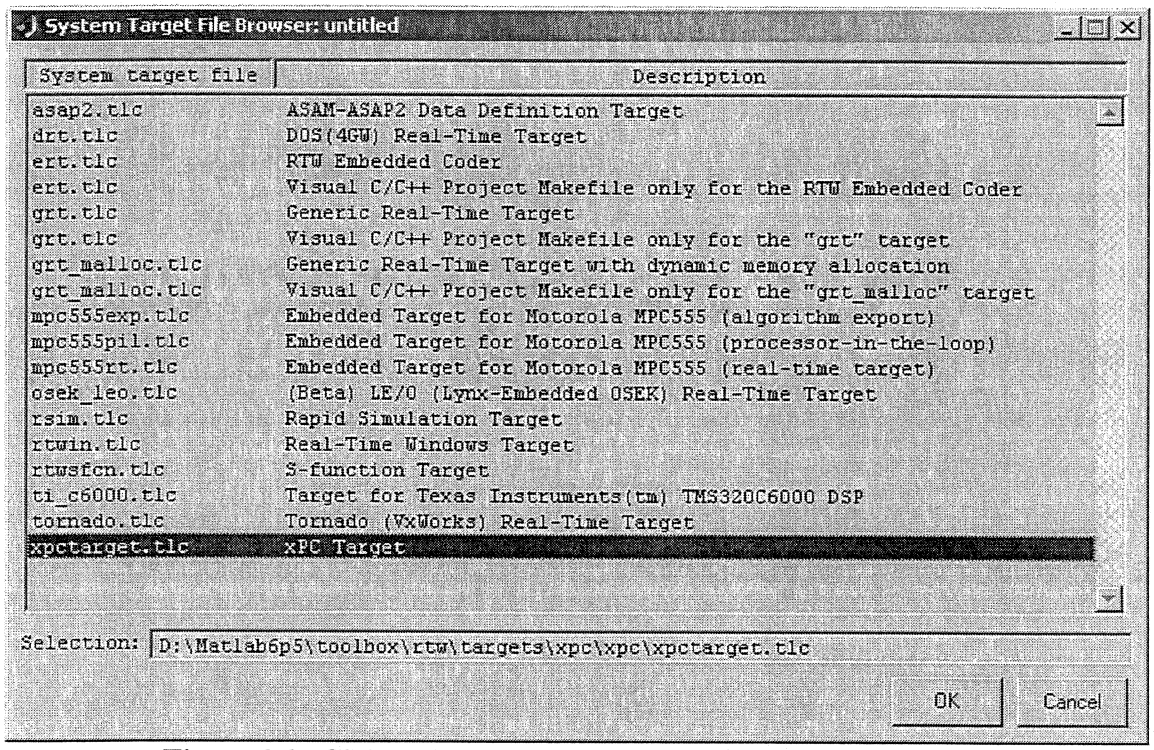


Figure 2.2: Cibles disponibles dans la configuration de RTW

La majorité de ces cibles concernent la production de code pouvant être utilisé comme base pour la programmation d'une application complète. Les cibles générales comme *ert*, *grt*, *grt*, *grt_malloc* et *rwtsfcn* servent à produire un code qui sera intégré dans une application complète et compilé pour créer le code de cette application. En prototypage rapide, on désire obtenir un code compilé par RTW qui peut être utilisé directement pour effectuer les tests désirés sur l'algorithme de commande. Ainsi, on

s'intéresse à des cibles qui utilisent un environnement spécifique comme *DOS(4GW) Real-Time Target*, *Real-Time Windows Target* [10], *Tornado (VxWorks) Real-Time Target* [11] et *xPC Target* [12]. Chacun de ces environnements peut offrir les possibilités recherchées dans ce projet. De manière plus spécifique, les possibilités offertes par *xPC Target*, *Real-Time Windows Target* et *Tornado (VxWorks) Real-Time Target* ont été étudiées et sont regroupées dans le tableau qui suit.

Tableau 2.1: Analyse de différentes cibles de RTW

Paramètre	**Cible**		
	xPC Target	**Real-Time Windows**	**Tornado**
Intégration dans Matlab directement	**Oui**	Oui	Non
Support de plusieurs types de processeur	**Non (Intel / AMD)**	Non (Intel / AMD)	Oui
Grande librairie de blocs E/S	**Oui**	Oui	Oui (achat un par un des pilotes)
Type hôte / cible	**Oui**	Non	Oui
Coût d'acquisition	**Éducation: 50$ (Mathworks)**	Éducation: 50$ (Mathworks)	Coûteux (> 1000$) (environnement dédié de WindRiver)
Nécessité d'un compilateur	**Oui (Visual C++)**	Oui (fourni)	Oui (fourni)
Possibilité d'utiliser n'importe quel PC comme cible	**Oui (disquette de démarrage)**	Non (le PC de développement)	Non (installation nécessaire)

Il faut noter que la liste de cibles de RTW peut s'allonger avec des produits spécialisés comme *RT-LAB Solo* [13] de la compagnie Opal-RT. Ce progiciel fonctionne sur des Pentium II à 400 Mhz (minimum) avec l'environnement QNX, il

coûte 800 $ pour une licence éducationnelle et il regroupe plus de 80 pilotes de cartes E/S performantes. Ses spécifications indiquent qu'il s'intègre totalement à Matlab lors de son installation et qu'il fournit toutes les fonctions de communication pour permettre un suivi de l'exécution directement dans Simulink. *RT-LAB Solo* n'est pas inclus dans le tableau 2.1 car il n'est pas dans la liste de la figure 2.2 et il ressemble beaucoup à l'environnement *Tornado* pour les fins de comparaison.

Dans le cas de cette recherche, la cible xPC Target est retenue car elle s'intègre directement à Matlab, elle est peut coûteuse dans un contexte éducationnel et cet environnement possède la propriété intéressante de pouvoir utiliser n'importe quel ordinateur comme cible à partir d'une disquette de démarrage sans devoir installer un environnement d'exploitation spécifique. Dans un objectif de prototypage rapide, cette particularité est intéressante.

Il faut noter que le fait de configurer RTW dans un modèle ne modifie pas son comportement en simulation et il peut être utilisé indépendamment car RTW est invoqué exclusivement lors d'une commande de compilation provenant de la commande *Build* de la fenêtre RTW. Cependant, si des blocs inclus dans le modèle dépendent de la cible, comme par exemple les blocs pilotes pour les cartes d'entrées / sorties réelles, alors le modèle ne peut pas fonctionner directement dans Simulink et doit absolument être compilé et expédié dans l'environnement d'exploitation de la cible.

2.2.4 xPC Target

La cible xPC Target est un environnement d'exploitation conçu directement par *The Mathworks* [12] et qui est inclus dans les choix proposés par RTW. Voici les points importants concernant cette cible.

xPC Target est un environnement de hautes performances de type hôte-cible[6] pour le prototypage rapide de modèles Simulink. Le prototypage dans cet environnement se déroule en quelques étapes simples:

- production sur le poste hôte d'un modèle Simulink pour l'algorithme à tester;

6 L'appellation « hôte-cible » indique l'utilisation de deux ordinateurs, l'un comme hôte de programmation et de contrôle et l'autre comme cible pour l'exécution en temps réel.

- compilation sur le poste hôte du modèle avec RTW;
- transfert du résultat de compilation vers un matériel compatible PC (cible) relié au système à asservir via des interfaces d'entrées / sortie (E/S) compatibles;
- exécution du modèle dans l'environnement temps réel de la cible;
- récupération des données sur la cible et analyse des résultats sur le poste hôte.

Cette procédure rapide fait de xPC Target un produit idéal pour le prototypage expérimental rapide et pour des simulations avec matériel dans la boucle (*hardware-in-the-loop simulation*). Le matériel compatible PC peut être un ordinateur de bureau classique (processeur Intel ou AMD), un ordinateur industriel comme le xPCTargetBox [14], une carte mère PC/104 ou PC/104+, un carte CompactPCI, etc.

Au niveau de l'interaction avec un système physique, xPC Target propose une vaste librairie de blocs E/S présentant des pilotes pour des cartes standards couramment utilisées. En fait, il supporte au-delà de 150 cartes E/S avec leurs multiples fonctions et il fournit des fichiers gabarit (*templates*) pour permettre la création de pilotes pour des cartes non supportées (code en C++). L'ajout d'E/S dans les modèles revient donc à inclure un bloc, comme pour n'importe quelle fonction Simulink.

Au niveau des performances, le noyau xPC Target permet d'atteindre facilement une fréquence d'échantillonnage de l'ordre de 1 kHz et ce, en assurant la communication quasi temps réel avec le poste hôte pour les commandes et les données. Toutefois, une demande excessive pour l'affichage ou le transfert des données peut évidemment grandement diminuer les performances du système.

2.2.5 Utilisation de xPC Target: avantages et limitations

Puisque xPC Target fait partie de la suite Matlab / Simulink, il est clair que c'est un environnement de conception et d'exploitation idéal pour un usager qui a déjà fait toute l'étude en simulation dans cet environnement. En utilisant l'extension xPC Target, l'usager s'assure de devoir compléter seulement un minimum d'étapes avant de pouvoir

passer aux tests expérimentaux. Puisque l'environnement Matlab comporte beaucoup d'outils d'analyse et d'échange de données et qu'il comporte un éditeur qui supporte la programmation textuelle en plus de l'implantation graphique, il ne limite pas la liberté d'intégration de nouveaux éléments dans la boucle.

Par contre, l'usager ou le concepteur est entièrement dépendant de l'environnement Matlab et chaque ajout à cet environnement implique l'acquisition de nouvelles librairies de fonctions. Il est ainsi quasi impensable d'utiliser cet environnement pour l'insérer dans un produit commercial, puisque des droits doivent alors être payés pour chaque unité vendue avec le noyau de xPC Target comme système d'exploitation. Cette limitation ne nuit pas au projet actuel puisque son objectif est d'explorer l'environnement en terme de prototypage d'algorithme de commande dans un contexte de recherche. La commercialisation amène des critères de sélection de l'environnement qui intègrent des éléments quantitatifs comme le coût d'achat et le coût de distribution, des critères qui débordent de l'objectif de ce travail.

La figure 2.3 résume la structure de l'environnement Matlab, elle est tirée directement (sans traduction ni modification) du site Internet de *The Mathworks* [12] et représente très bien les interactions entre les différentes sections du logiciel avec la division hôte / cible typique de xPC Target. On peut noter la présence de Matlab comme élément clef en haut du diagramme, puis viennent Simulink et RTW conjointement avec le module de machine à états (*Stateflow*). Vient ensuite la section *hôte* de xPC

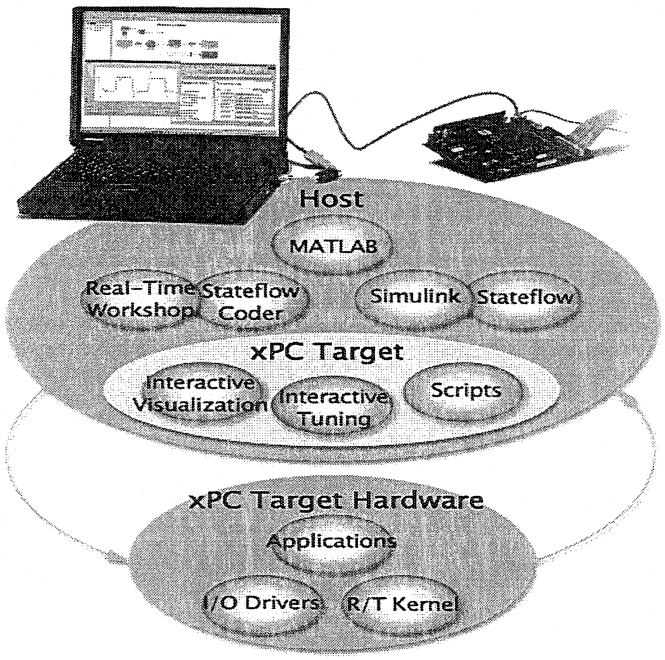


Figure 2.3: Schéma relationnel de xPC Target

Target qui assure la communication avec la cible et la visualisation interactive. La section du bas montre la cible avec le matériel E/S, le noyau temps réel et l'application à exécuter.

2.3 Environnement LabVIEW

Le logiciel LabVIEW de la société National Instruments est le deuxième environnement retenu. Ce choix est justifié par le fait que ce logiciel est fréquemment utilisé en milieu industriel pour la mesure de signaux et le contrôle de procédés via les cartes d'acquisition et de contrôle fournies par la même société. Le lien direct du logiciel avec les cartes d'acquisition en fait un produit de choix lorsque le développeur ne veut pas avoir à programmer les interfaces entre le matériel et le logiciel. Bien qu'il soit utilisé couramment en génie mécanique et chimique à l'École Polytechnique de Montréal, LabVIEW est moins répandu que Matlab / Simulink dans les laboratoires de la section Automation et Systèmes du département de génie électrique. Il est cependant intégré, à échelle réduite, dans la formation des étudiants dans les laboratoires académiques de la section. Son potentiel pour l'atteinte de l'objectif de fournir un environnement informatique de simulation et d'expérimentation est analysé dans ce travail au même titre que l'environnement xPC Target.

L'environnement de LabVIEW se divise en deux sections distinctes lors de la programmation: le panneau qui regroupe les contrôles et le diagramme qui permet les calculs et l'utilisation des contrôles. Une présentation du logiciel par la compagnie se trouve sur leur site Internet [15]. Couvrir le détail de son utilisation dans ce travail est inutile, car il s'agit d'un logiciel complexe pour lequel la formation est le sujet de livres entiers et de formations interactives fournies par la maison mère du logiciel. Seuls les concepts de base concernant ce logiciel sont présentés.

2.3.1 Section panneau

La section panneau contient tous les interfaces usagers fournis par l'application lors de sa conception. Le nom de panneau est donné car le but de LabVIEW est de créer

des instruments virtuels[7] qui rappellent un panneau de contrôle avec des boutons, des jauges, des indicateurs, des sélecteurs, etc.

Aucune programmation algorithmique n'est réalisée dans cette section, elle permet seulement de définir l'interface que l'utilisateur aura pour communiquer avec le contrôleur implanté derrière le panneau. Si des éléments d'interface sont absents , alors il est impossible d'utiliser le contrôleur implanté puisque le logiciel ne fournit aucune interface de type *ligne de commande* pour communiquer avec la section diagramme de l'implantation. L'avantage de fonctionner ainsi est que l'usager ne peut pas effectuer une opération qui n'est pas prévue lors de la conception et que les contrôles de type *panneau de commande* fournissent une bonne visualisation et sont très intuitifs à utiliser.

2.3.2 Section diagramme

Seul, le panneau d'un programme LabVIEW ne fournit aucune fonctionnalité; il doit être relié aux différents blocs de traitement par des lignes de flux de données. Encore selon le contexte d'instrument virtuel, les concepteurs du logiciel ont choisi de fonctionner avec la notion de circuit électrique pour les relations entre les différents éléments du panneau de contrôle et les blocs de traitement. La création d'un programme consiste donc à concevoir une interface, puis ensuite à la relier à un circuit de traitement qui assure les fonctions désirées en termes de calcul, d'entrées / sorties et d'analyse de signaux. LabVIEW est conçu autour de ces fonctions de traitement qui regroupent une multitude de blocs d'analyse et de traitement de signaux utiles à la mesure et à l'échange d'information avec un procédé physique.

Pour les traitements mathématiques nécessaires à la simulation numérique (comme la dérivée et l'intégration numérique), la structure de diagramme électrique offre moins de potentiel qu'une structure algorithmique comme le langage M, mais il demeure possible d'implanter tous les traitements nécessaires à une application dans LabVIEW si le concepteur a une bonne compréhension de son fonctionnement et une maîtrise des librairies de blocs.

---

7 L'extension des programmes LabVIEW est *VI*, soit l'abréviation de *Virtual Instrument*.

2.3.3 Extension LabVIEW RT

Afin de fournir de meilleures performances temporelles, LabVIEW a développé une extension fonctionnant en temps réel: LabVIEW RT [16]. Avec cette extension et une conception attentive des différents blocs dans le diagramme, le modèle développé est beaucoup plus performant qu'avec la version de base de LabVIEW qui fonctionne dans l'environnement d'exploitation Windows. Cette extension nécessite cependant l'achat d'une carte spécifique contenant un processeur dédié et une carte d'échange de données particulière à LabVIEW RT.

Puisque la planification du projet de recherche n'a pas inclus l'utilisation concrète de cette extension et qu'elle implique l'achat du logiciel et du matériel qui lui est spécifique, l'extension n'est pas explorée en détails dans ce travail. Elle ne sera pas prise en compte lors des essais expérimentaux. Un document sur la programmation d'une application précise est fourni sur le site de la compagnie [17] et indique comment LabVIEW RT permet d'améliorer les performances d'un module VI. Ces résultats n'ont pu être vérifiés lors de ce travail de recherche.

2.3.4 Utilisation de LabVIEW: avantages et limitations

Le contexte principal d'utilisation de LabVIEW est sans aucun doute le milieu industriel, plus particulièrement celui du test. LabVIEW est conçu pour effectuer l'acquisition et le traitement de signaux provenant de cartes E/S spécialisées dont plusieurs sont produites par la même société que le logiciel. Au niveau des fonctionnalités et des performances dans un tel contexte d'utilisation, il est difficile de concurrencer ce produit.

Quant à la question de savoir si LabVIEW est oui ou non un environnement général et performant pour la programmation d'algorithmes comme ceux associés au contrôle de procédés; ce travail tente d'y répondre. Plusieurs exemples d'applications complexes se trouvent sur le site du produit et vantent son mérite. Des articles comme [18] et [19] semblent confirmer ces affirmations. Toutefois, aucune mesure de la complexité relative de l'implantation dans LabVIEW versus l'implantation dans un autre

environnement n'a été trouvée lors des recherches préliminaires de ce projet. La première étude de cas présentée dans ce travail vise à répondre à cette question.

2.4 Environnement hybride simulation / expérimentation

Deux environnements ont été présentés et chacun possède ses forces et ses faiblesses comme tout produit destiné à l'utilisation par un grand nombre d'usagers dans une vaste gamme de domaines d'application. Lors d'un projet donné, une équipe vient souvent à la conclusion qu'il est préférable de concevoir son propre environnement et ainsi avoir le contrôle complet sur tous les aspects de son fonctionnement. Dans un tel système on peut trouver des éléments d'un logiciel commercial pour une partie de la conception (par exemple le modèle mathématique de simulation), puis ensuite un autre logiciel pour une autre partie du problème (par exemple l'interface graphique) et ainsi de suite pour tout le développement. La liaison entre ces produits est soit manuelle via un transfert des données et du code d'une section à l'autre ou automatique via la programmation des interfaces nécessaires.

C'est ce qui arrive dans de nombreux projets: les simulations sont faites dans un environnement donné, puis lorsque les résultats théoriques sont intéressants, les algorithmes sont transférés vers un environnement qui permet le contrôle en temps réel du système physique à asservir. Lors de ce transfert, l'équipe doit typiquement reprogrammer tous les éléments de la boucle (contrôleurs, fonctions d'analyse des données, fonctions de conditionnement des signaux, etc.) dans le nouvel environnement. Pour chaque modification majeure dans une section, le travail doit être transféré à l'autre afin d'assurer un maximum de concordance entre les simulations et les expérimentations.

2.4.1 Utilisation d'un environnement hybride: avantages et limitations

Un environnement hybride ne convient définitivement pas aux objectifs de prototypage rapide propres à ce projet. Cependant, il demeure un choix très efficace

pour certains projets.

La principale raison d'y recourir est lors du désir de commercialiser un produit qui utilise un algorithme récemment développé. Avec des environnements commerciaux comme Matlab ou LabVIEW, une redevance est associée à la vente d'un produit qui utilise la technologie du logiciel et cette redevance augmente le coût du produit, ce qui le rend inévitablement moins concurrentiel. L'équipe qui souhaite commercialiser une solution technologique dans un tel cadre a tout avantage à développer un système hybride basé sur des environnements gratuits comme Linux [20] (environnement d'exploitation), le langage C++ (environnement de programmation et de compilation gratuit disponible) et OpenGL [21] ou WxWidgets [22] (interface graphique et / ou textuelle constituant l'environnement d'opération).

Un autre cas d'emploi d'un environnement hybride est illustré par l'implantation de systèmes très complexes qui utilisent plusieurs processeurs ou qui demandent un niveau élevé de communications interprocessus. Ces systèmes nécessitent l'emploi d'algorithmes bien précis que les environnements commerciaux ne supportent pas nécessairement, d'où la nécessité de passer vers un environnement hybride.

2.5 Conclusions sur les environnements informatiques

Chaque environnement possède ses caractéristiques propres et de nombreux cas d'applications pour vanter leurs avantages malgré certains inconvénients. Dans ce travail de recherche, l'objectif est la maîtrise de ces environnements dans un contexte de prototypage rapide d'algorithmes de commande dans un environnement temps réel. Ceci passe par l'analyse détaillée des performances de chacun dans un contexte d'utilisation avant de pouvoir les départager. Pour le moment, il est clair que l'environnement Matlab / Simulink est propice aux simulations, que l'environnement LabVIEW est adapté au contrôle réel de procédé et qu'un environnement hybride ne remplit pas du tout les objectifs de prototypage rapide. Les études de cas des chapitres suivants vont permettre une meilleure analyse des performances de xPC Target et LabVIEW dans un contexte expérimental d'utilisation.

Chapitre 3 : Étude expérimentale: asservissement en position

Depuis plusieurs années déjà, les laboratoires d'enseignements de la section Automation et Systèmes de l'École Polytechnique de Montréal utilisent un banc d'essais en asservissement de position de conception maison qui a été le sujet d'un article de l'*Institute of Electrical and Electronics Engineers* (IEEE) il y a vingt ans [23] et dont l'évolution technique et technologique se poursuit encore aujourd'hui. Il est d'ailleurs le sujet d'un laboratoire académique [24] à l'École Polytechnique de Montréal. Ce banc d'essai a été reproduit par plusieurs autres universités pour leurs cours du baccalauréat. Il constitue donc un bon exemple pour comparer les deux environnements informatiques qui viennent d'être présentés soit: LabVIEW et xPC Target. L'implantation complète de la boucle fermée de contrôle de ce système est réalisée dans chacun des environnements et une analyse qualitative et quantitative des performances obtenues est développée. Le contrôleur implanté utilise l'algorithme PID-DL développé par DeSantis [25].

3.1 Présentation du banc d'essais et du matériel utilisé

La conception de ce système offre un banc d'essais mécaniquement et structurellement simple avec un potentiel intéressant pour l'exploration de différents algorithmes de commande de position en boucle fermée. Avec les années, sa structure a évoluée pour accueillir des capteurs plus performants, mais sa base mécanique est demeurée la même. La figure 3.1 illustre la composition de la boucle de commande incluant le banc d'essais.

Afin de réaliser la boucle de commande, il faut posséder les informations techniques sur chacun de ses éléments. Le tableau 3.1 indique la compagnie d'origine et les caractéristiques importantes de chaque composante. Il faut noter que certaines composantes datent de plusieurs années et qu'il n'est pas facile de retrouver leurs spécifications complètes aujourd'hui. Tout matériel équivalent peut être utilisé pour constituer le banc. Le contrôleur n'est pas mentionné dans le tableau descriptif puisqu'il

est constitué d'un module de nature logicielle plutôt que matérielle. La figure 3.2 illustre à l'aide d'une photographie annotée la structure mécanique du banc d'essais.

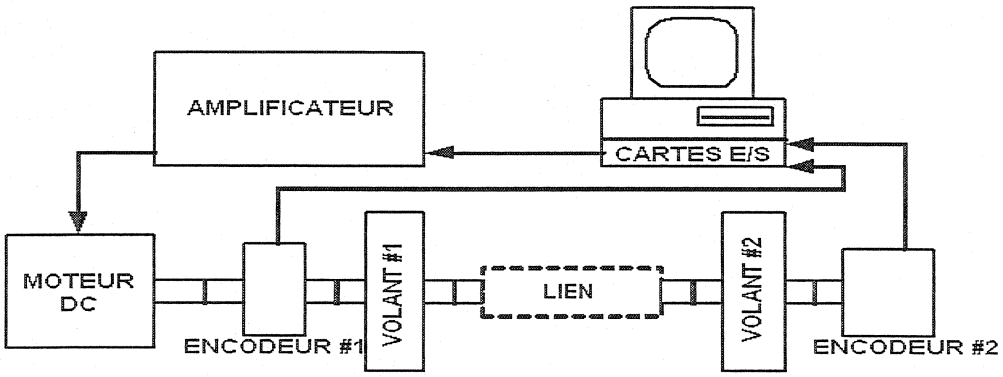


Figure 3.1:Composition mécanique et électronique du banc d'essai en asservissement de position

Tableau 3.1: Éléments constituant le banc d'essai en asservissement de position

Élément	Compagnie	Modèle / Description
Moteur	Electrocraft	Moteur à courant continu ±100 VDC; couple maximal T_M=0.565 Nm; constante de couple K_T=0.325 Nm/A; résistance de l'armature R_A=10.5 Ohms)
Encodeur 1	Hohner	Modèle *Hohner Shaft Encoder* - Encodeur de milieu de chaîne 200 impulsions par tour (sur 2 canaux)
Encodeur 2	Sandtron	Modèle *SunX Optical Encoder* - Encodeur de fin de ligne 200 impulsions par tour (sur 2 canaux)
Amplificateur	Kepco	Modèle BOP-36M; (± 36 Volts; 250 Watts) Amplificateur inverseur linéaire (gain de -9.9 max).
Interface 1	Computer Boards	Modèle CIO-QUAD02 (BUS ISA; 2 entrées) Carte de lecture d'encodeur optique à deux canaux (permet la lecture en quadrature).
Interface 2	National Instruments	PCI-6025E (BUS PCI, 16 entrées; 2 sorties). Carte d'entrées / sorties analogiques (±10 Volts)

Élément	Compagnie	Modèle / Description
Lien	*Construction maison*	**Élastique**: ressort de constante d'élasticité indéfinie. **Rigide**: couplage mécanique direct
Volant	*Construction Maison*	Volant d'inertie (rayon R = 50 mm; masse M = 1 kg)

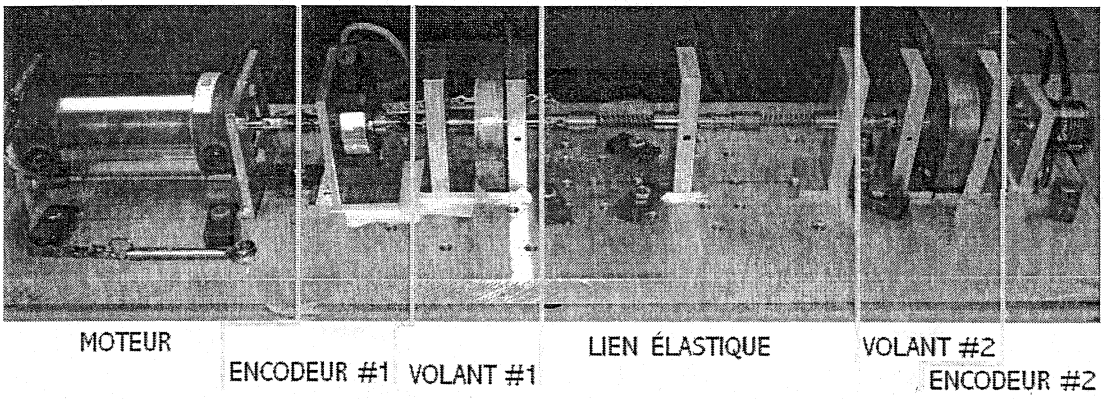


Figure 3.2: Implantation physique du banc d'essai en asservissement de position

Pour les simulations du banc d'essais, la dynamique du système est représentée par une fonction de transfert qui représente chaque cas d'utilisation (lien rigide ou élastique):

$$F(S)=\frac{\Phi(s)}{U(s)}=\frac{G}{s(s+c)(s^2+as+b)} \text{ (élastique) ou } F(S)=\frac{\Phi(s)}{U(s)}=\frac{K_m}{s(s\tau_m+1)} \text{ (rigide)}$$

La provenance théorique et les différents paramètres de ce modèle ont été identifiés dans le cadre d'expériences précédentes [24] et les valeurs qui ont été validées dans le passé sont utilisées pour le simulateur. L'expérience montre aussi que ces paramètres varient beaucoup lorsque l'assemblage mécanique du banc d'essai est repris d'une fois à l'autre. Un rapport sur l'implantation complète du système, sur de nouvelles techniques d'identification des paramètres et sur une analyse en profondeur du contrôle de son asservissement avec un contrôleur PID-DL fait l'objet d'un rapport technique qui est en cours de préparation [26].

Tableau 3.2 : Valeurs des paramètres pour les simulations du banc d'essai en asservissement de position

Paramètre	Provenance théorique	Valeur
K_m	Gain statique du système d'ordre 2 (lien rigide)	10.9336
τ_m	Constante de temps du système d'ordre 2 (lien rigide)	0.2202
a	Paramètre temporel du système d'ordre 4 (lien élastique)	1.6848
b	Paramètre temporel du système d'ordre 4 (lien élastique)	82.8845
c	Paramètre temporel du système d'ordre 4 (lien élastique)	2.9176
G	Gain statique du système d'ordre 4 (lien élastique)	3563.1257

3.2 Implantation du banc d'essai dans LabVIEW

Cette implantation utilise au départ une application programmée pour les laboratoires académiques utilisant le banc d'essai. Elle a été modifiée pour en améliorer les performances et ajouter de nouveaux outils comme la possibilité d'effectuer les simulations et les expérimentations sans changer d'application.

La figure 3.3 présente l'essentiel de l'implantation réalisée dans le fichier *CTRL_PID.VI*. Le détail de tous les blocs principaux se retrouve à l'annexe I afin d'alléger le texte. Le diagramme présenté sur la figure ne montre que la boucle principale de commande. L'initialisation des paramètres, la gestion du fichier contenant les paramètres globaux ainsi que la mise en place des vecteurs de données pour la sauvegarde des résultats ne sont pas présentés sur la figure 3.3 car elles occupent trop d'espace graphiquement sur la figure. Bien qu'elles augmentent la taille du diagramme, ces fonctionnalités sont nécessaires dans un modèle qui sert à faire le prototypage d'algorithmes de commande.

Un fichier de paramètres globaux, *GLOBAL_PARAM.VI*, permet de simplifier les échanges de données dans le diagramme, il est présent pour initialiser les variables du modèle utilisé en simulation et pour transférer les paramètres de fonctionnement de différents blocs comme le contrôleur. Cet ajout permet aussi de simplifier l'interface usager présentée sur le panneau principal.

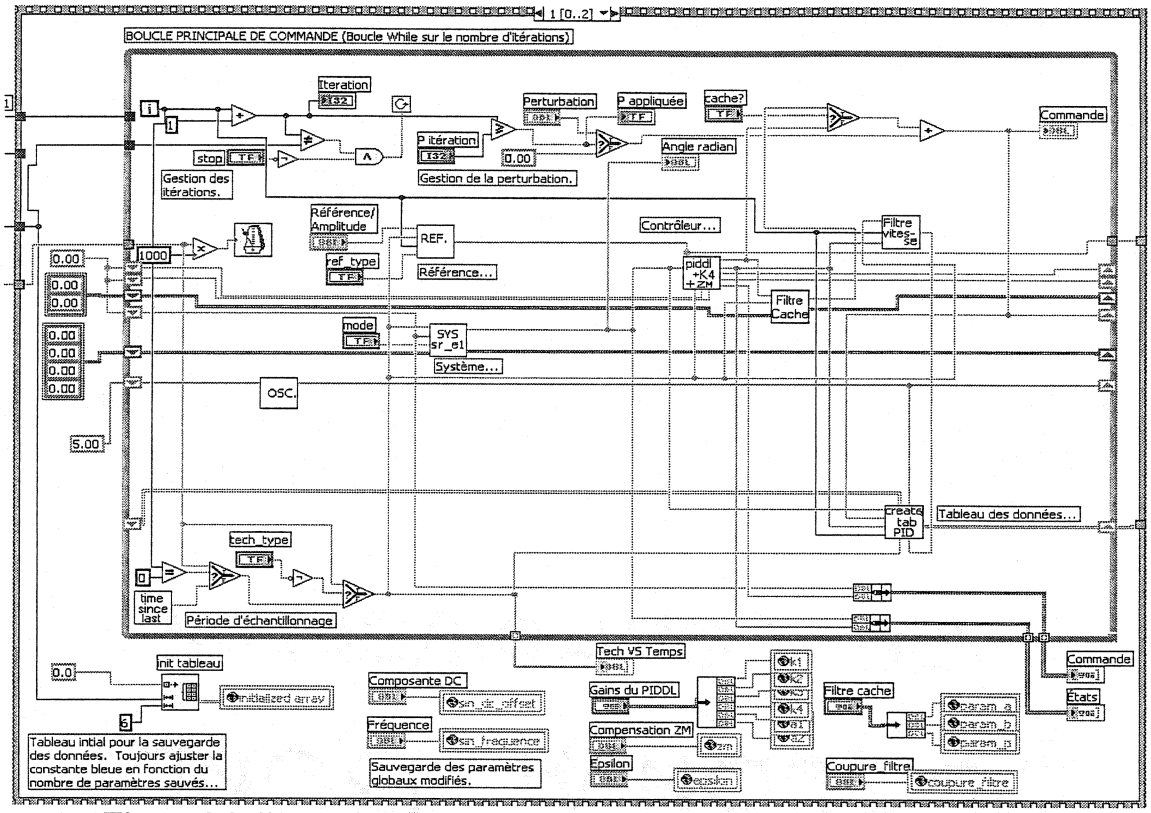


Figure 3.3: Implantation du contrôleur dans LabVIEW (diagramme)

Le panneau principal du fichier *CTRL_PID.VI* est le résultat final pour l'usager. C'est l'écran principal que LabVIEW présente lors de l'ouverture du fichier. La figure 3.4 offre un aperçu complet de cette interface. Les boîtes de texte permettent de modifier les paramètres des essais et les trois graphes donnent un aperçu de la réponse à la fin de l'essai. Évidemment, la gestion de ces éléments d'affichage et de contrôle ajoute plusieurs blocs au diagramme, cependant ils permettent de communiquer les paramètres des essais au diagramme et d'observer les résultats avant de les conserver dans des fichiers de données.

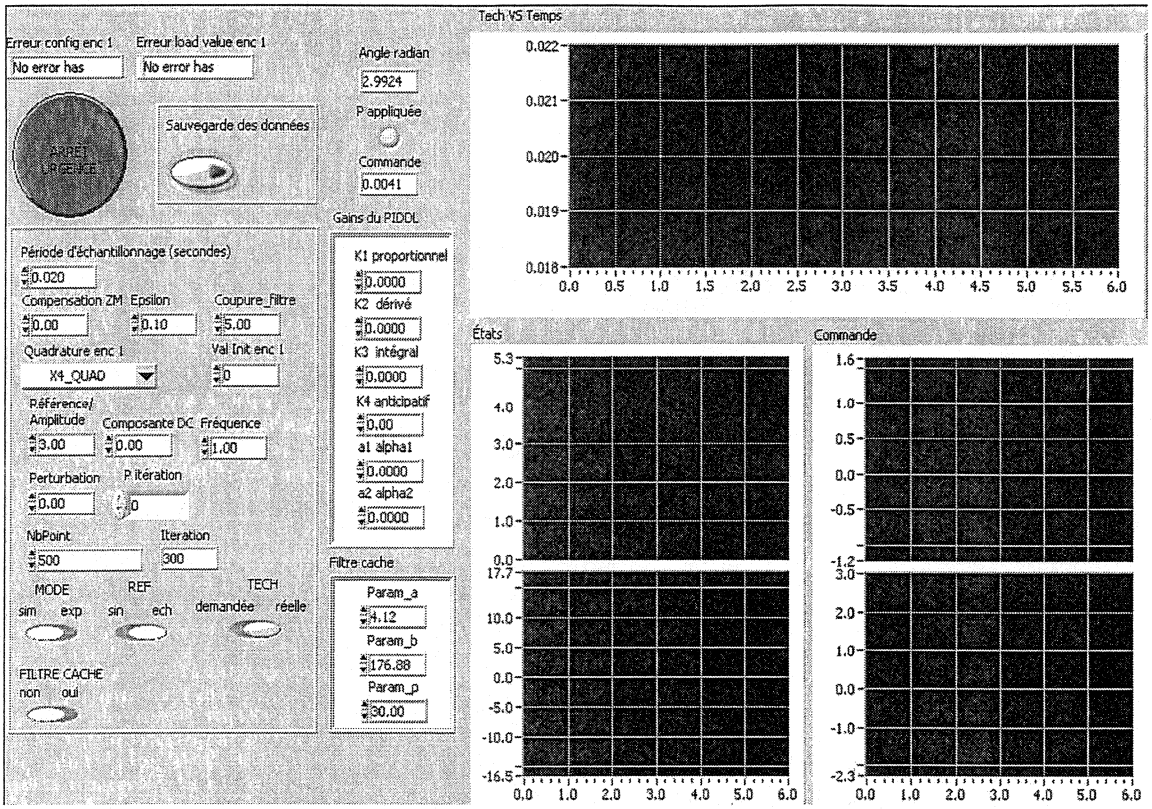


Figure 3.4: Implantation du contrôleur dans LabVIEW (diagramme)

3.2.1 Évaluation initiale de l'implantation dans LabVIEW

Voici l'analyse de l'environnement LabVIEW par rapport aux spécifications recherchées dans le cadre de ce projet.

3.2.1.1 Compatibilité des éléments de la boucle

La totalité des éléments de la boucle du banc d'essais sont implantés directement sous forme de diagrammes avec les blocs fournis dans les librairies de LabVIEW. Pour les sorties analogiques (moteur, oscillateur), le logiciel fourni un pilote matériel pour la carte PCI-6025E utilisée par le banc d'essai. Pour la carte d'acquisition des encodeurs optiques, LabVIEW n'a pas de pilote particulier. L'implantation d'un pilote a demandé une analyse approfondie de la carte en question à partir du document du manufacturier [27]. Le code en C a été relié au diagramme principal via des blocs *Code Interface Node*

dans LabVIEW (librairie de fonctions *Advanced*). De plus, il a fallu implanter un bloc réalisant la conversion en complément à 2 [28] pour assurer la compatibilité des données entre la carte d'acquisition et LabVIEW. Le reste des éléments ont été codés sans grande difficulté via les blocs de calculs standards provenant directement des librairies fournies.

3.2.1.2 Protocole d'échange de données

Dans cette implantation, les seuls échanges de données avec le système physique passent par les cartes E/S, soit la carte de sortie analogique et la carte d'acquisition des encodeurs optiques. Comme discuté précédemment, l'implantation des pilotes de ces cartes s'est bien déroulée et ces pilotes sont fonctionnels.

LabVIEW est conçu pour l'analyse en ligne des données durant l'exécution et il ne fournit pas de fonctions pour l'analyse post-exécution. Cette analyse est réalisée dans Matlab, ce qui implique qu'un moyen pour transférer les données vers Matlab est nécessaire. La solution retenue est la création d'un fichier texte pour recueillir les données. Au niveau des échanges, cela implique l'indexation des données dans un tableau et l'utilisation d'un bloc d'écriture dans un fichier texte pour le sauvegarder. Un en-tête complet contenant les paramètres d'exécution est aussi créé et conservé au début du fichier de données.

Puisque le contrôle en temps réel demande de la puissance de calcul et de la mémoire disponible sur le poste de contrôle, la solution de conserver les données est jugée optimale. Des outils sont disponibles dans LabVIEW pour créer une passerelle directe vers Matlab à l'aide de blocs de communication, toutefois l'analyse post-exécution ne justifie pas l'emploi d'une telle passerelle. D'ailleurs, l'ouverture de Matlab en même temps que LabVIEW monopolise inutilement les ressources du système d'exploitation. En conservant les données des essais dans des fichiers textes, il est possible de les charger en différé dans Matlab sans dégrader la performance du contrôleur.

3.2.1.3 Environnement de conception

Tout le développement se fait dans les deux sections principales du programme, soit le panneau et le diagramme avec une brève excursion dans le fichier d'implantation des paramètres globaux. Le panneau permet de disposer les éléments pour fournir une interface agréable et pour optimiser le contrôle qu'aura l'opérateur sur le système final. Dans la section diagramme, toutes les fonctions sont assurées par une suite de blocs représentant des fonctions de base comme la somme, la multiplication, les boucles, mais aussi par des blocs plus évolués qui effectuent des traitements comme le filtrage des signaux ou qui regroupent plusieurs autres blocs qui permettent l'implantation d'un algorithme complexe. Certains blocs permettent même l'implantation dans un langage textuel ressemblant au C. Tous ces blocs permettent une représentation graphique avec des liaisons orientées qui fixent le flux des données dans le diagramme. La compréhension de base des diagrammes est très simple, mais le nombre de liaisons devient rapidement important si la notion de sous-système n'est pas utilisée. Puisque les paramètres présents sur le panneau de commande sont traités comme des signaux de données, il faut donc des liaisons entre chaque paramètre et tous les blocs auxquels il est destiné.

La possibilité d'utiliser des sous-systèmes pour regrouper plusieurs blocs dans un seul simplifie le schéma. Cependant, chaque sous-système possède alors son propre panneau et son propre diagramme et se retrouve du même coup sauvegardé dans un fichier indépendant qu'il faudra toujours conserver avec le fichier original. Cette division facilite d'éventuelles erreurs lors du transfert des fichiers.

Un fichier séparé contenant les paramètres globaux de fonctionnement à partager avec tous les blocs d'un diagramme principal est aussi implanté. Or, la gestion de ce type de fichier prend quand même beaucoup d'espace graphique dans le diagramme puisque le panneau principal doit pouvoir communiquer tous les nouveaux paramètres à ce fichier avant le début de la boucle de commande (voir même pendant l'exécution selon le type de paramètre). Ne pas avoir à passer tous les paramètres via des liaisons séparées vers tous les blocs qui les utilisent demeure un avantage important justifiant

l'emploi des paramètres globaux. Le meilleur exemple de simplification du diagramme par cette méthode est l'utilisation de la période d'échantillonnage. Cette dernière est utilisée par plusieurs sous-systèmes, ce qui demande plusieurs liaisons. En passant par le fichier de paramètres globaux, une seule liaison conserve la valeur, puis tous les blocs peuvent ensuite la lire directement à partir de leur diagramme, éliminant de nombreuses liaisons dans le diagramme principal.

En résumé, la méthode de conception offerte par LabVIEW est très efficace si l'usager organise son code et tire profit des signaux spéciaux comme les BUS de données et les paramètres globaux. Évidemment, une maîtrise de LabVIEW est nécessaire.

3.2.1.4 Environnement d'exploitation

Sachant que seule la partie principale de LabVIEW est utilisée, sans son extension temps réel, l'environnement d'exploitation est le même que l'environnement de conception. Dans ce projet, LabVIEW est utilisé dans l'environnement Windows 2000 pour la conception et l'exécution des algorithmes de commande. Afin de permettre les meilleures performances possibles, le processus LabVIEW peut être placé à un niveau de haute priorité dans le gestionnaire de tâches de Windows. L'option *temps réel* proposée par Windows 2000 comme niveau de priorité ne sera pas retenue. Ceci est justifié par le fait qu'en priorité *temps réel*, le gestionnaire de tâches bloque complètement l'accès à la souris et au clavier pendant l'exécution de l'algorithme de commande, ce qui empêche toute interaction de la part de l'opérateur. Au niveau de la sécurité, ceci est inacceptable.

Stabilité temporelle

Au niveau de la stabilité temporelle, un bloc de temporisation dans la boucle itérative de commande permet de fixer une période d'échantillonnage (paramètre Tech). Il faut se rappeler que dans le système d'exploitation Windows, selon la tâche en cours dans le gestionnaire du système d'exploitation, il est possible que cette période d'attente soit légèrement augmentée ou même diminuée si le gestionnaire modifie d'un coup la tranche de calcul attribuée au processus LabVIEW.

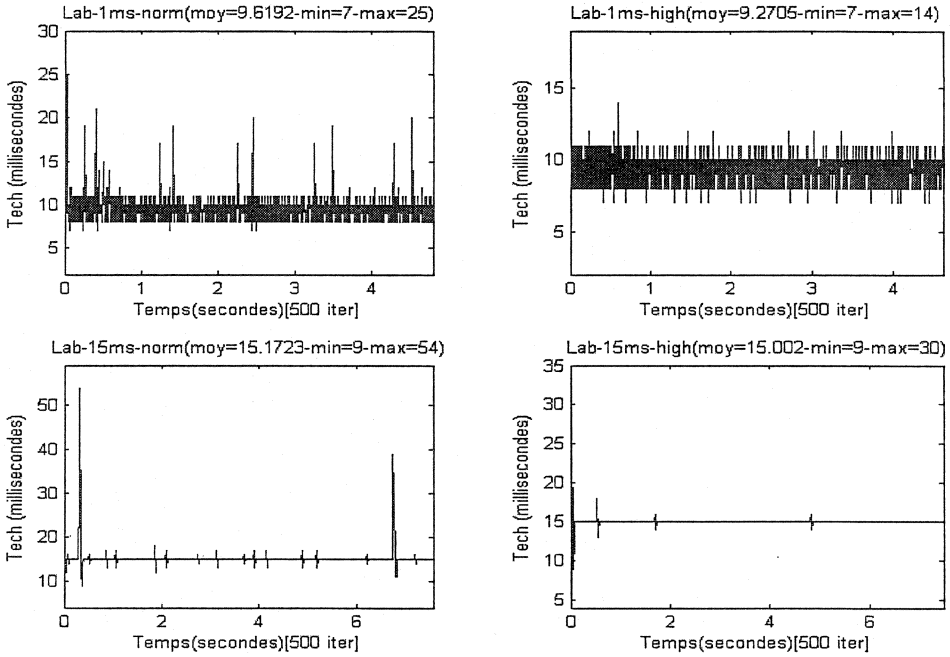


Figure 3.5 : Stabilité de la période d'échantillonnage réelle dans LabVIEW (500 itérations)

Quantitativement, la figure 3.5 présente les résultats illustrant la période d'échantillonnage au fil des itérations pour 500 itérations de la boucle de contrôle selon les deux niveaux de priorité attribués par le gestionnaire de tâches de Windows (*normale* et *haute*). Sur la figure, les titres ont la signification suivante: *Lab* indique un essai dans LabVIEW, *1ms* indique une période d'échantillonnage de 1 ms (*15ms* indique une période de 15 ms), *norm* indique un niveau de priorité *normale* (*high* indique un niveau de priorité *haute*). Les titres incluent aussi la moyenne, le minimum et le maximum de la période d'échantillonnage appliquée lors de l'essai. L'analyse de ces résultats permet de faire remarquer que plus la période d'échantillonnage est petite, plus l'incertitude sur sa valeur appliquée devient importante. Le tableau 3.3 met en évidence cette affirmation.

Pour une période d'échantillonnage de 1 ms, la période appliquée est de 9.6 ms en moyenne, ce qui représente un écart moyen de 8.6 ms avec la valeur demandée. Cette différence est importante et elle nuit au bon fonctionnement de la boucle de commande. Le fait d'attribuer la priorité *haute* au processus a diminué le jeu (écart entre les valeurs

maximale et minimale de la période d'échantillonnage appliquée), mais l'écart avec la valeur demandée demeure quasi inchangé (8.2 ms).

Pour une période d'échantillonnage de 15 ms, l'écart entre la valeur désirée et la valeur réelle est presque nul bien que le jeu soit plus important quantitativement. L'analyse de la figure 3.5 montre cependant que ce jeu n'est pas temporellement fréquent, malgré des valeurs numériques un peu plus grandes, ce qui est moins nuisible que l'oscillation continue observée dans les deux premiers cas.

Tableau 3.3: Effet de la priorité système et de la période demandée sur les variations de la période d'échantillonnage appliquée

Essai	Moyenne (ms)	Minimum (ms)	Maximum (ms)	Jeu (ms)	Écart (ms)
1ms;norm	9,6	7	25	18	8,6
1ms high	9,3	7	14	7	8,2
15ms;norm	15,2	9	54	45	0,2
15ms;high	15	9	30	21	0

Afin de mieux analyser la stabilité du système, un deuxième test consistant à augmenter significativement le nombre d'itérations dans la boucle de commande a été réalisé. Ce nombre a été arbitrairement fixé à 5000 et le même type de graphique qu'à l'essai précédent est refait (les titres ont la même signification). La différence de cet essai par rapport au précédent consiste au fait que puisque les données d'exécution sont indexées et conservées jusqu'à la fin de la boucle, l'augmentation des besoins en mémoire pour LabVIEW demande constamment à Windows de refaire le partage mémoire. Cela demande du temps et des ressources comme le démontre les résultats de la figure 3.6.

On voit clairement en comparant le tableau 3.4 avec le tableau 3.3 que l'incertitude est plus grande pour une longue exécution de la boucle, ce qui confirme que LabVIEW et Windows ont plus de difficultés à gérer la mémoire pour les données lors d'un long essai.

Tableau 3.4: Effet de la longueur des essais sur les variations de la période d'échantillonnage appliquée

Essai	Moyenne (ms)	Minimum (ms)	Maximum (ms)	Jeu (ms)	Écart (ms)
1ms;norm	16,5	15	51	36	15,5
1ms high	16,2	14	54	40	15,2
15ms;norm	16,6	13	42	29	1,6
15ms;high	16,3	13	48	35	1,3

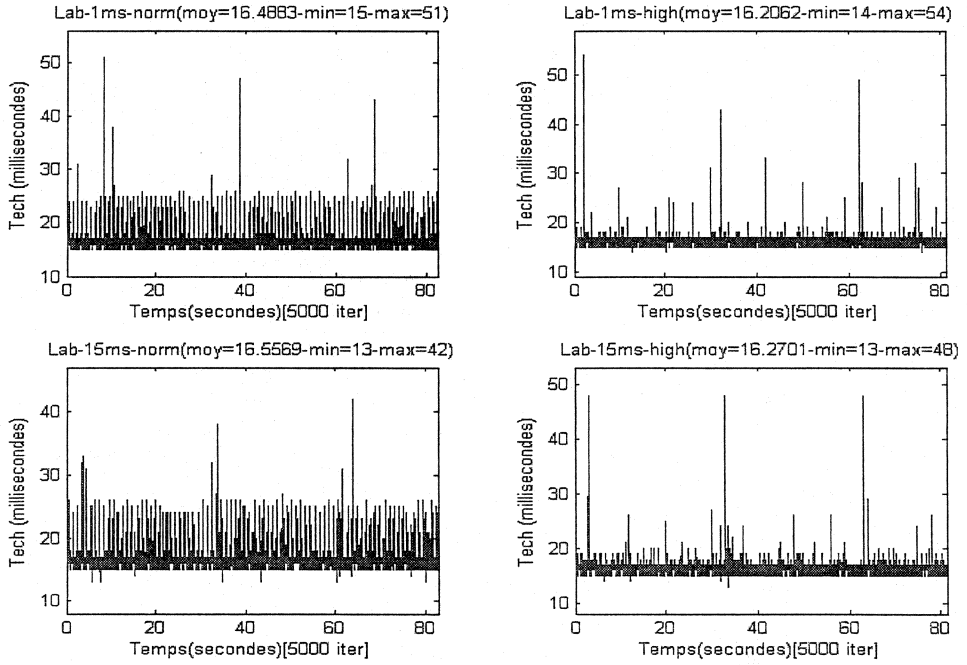


Figure 3.6: Stabilité de la période d'échantillonnage réelle dans LabVIEW (5000 itérations)

En résumé au sujet de la stabilité dans LabVIEW, il est clair qu'il faut utiliser le niveau de haute priorité et utiliser une période d'échantillonnage suffisamment élevée (plus de 10 ms). De plus, les long essais sont à proscrire pour ne pas dégrader les performances en calcul de la boucle de commande. Ces considérations entraînent la remarque que la stabilité temporelle de l'environnement d'exploitation est relativement précaire.

Rapidité d'exécution

Un premier moyen de fixer les attentes en termes de rapidité d'exécution est de consulter une table comparative entre LabVIEW et la programmation C++, qui est une option couramment utilisée en contrôle de procédés. National Instruments fournit une figure comparative à cet effet qui inclut aussi une comparaison entre l'ancienne version de LabVIEW et la nouvelle version (celle utilisée dans ce projet). La figure 3.7 provient d'un document promotionnel de la société et peut être retrouvée sur Internet [29]. Le détail des tests effectués pour obtenir cette figure est aussi donné à l'adresse en question. Ce qui ressort comme résultat principal de cette figure est que LabVIEW est plus rapide que le C pour des opérations comme la lecture de fichier, l'analyse spectrale, le filtrage et les manipulation de tableaux. Ces opérations sont utiles à la réalisation d'un contrôleur, ce qui donne à LabVIEW un avantage en vitesse par rapport à la programmation en C directement.

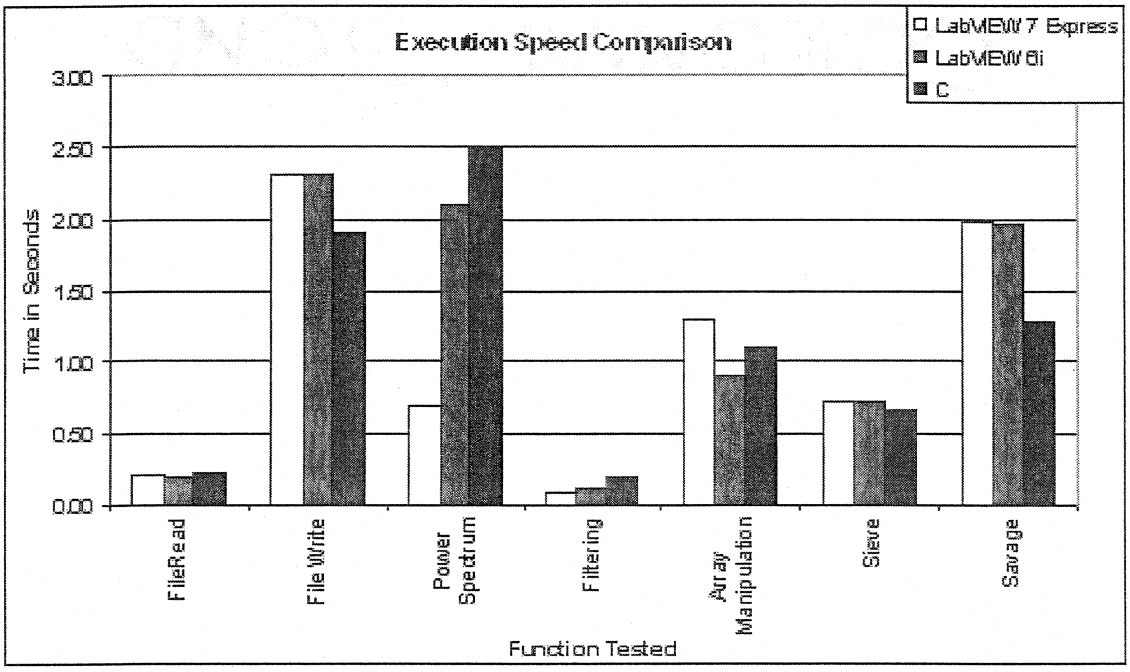


Figure 3.7: Banc d'essais en performance de LabVIEW

Le meilleur moyen de vérifier ces résultats dans le cadre de notre implantation est de trouver la plus petite période d'échantillonnage que le système peut assurer avec le

contrôleur implanté. Pour obtenir cette valeur, une période d'échantillonnage de 1 ms est demandée et le temps actuellement écoulé entre deux itérations est mesuré dans la boucle. Ces résultats ont été présentés indirectement aux figures 3.5 et 3.6 lors des essais précédents. Pour le test actuel, seules les courbes dont le titre comporte la mention *1 ms* sont analysées et résumées dans le tableau 3.5. Selon ces résultats, le niveau de priorité accordé à LabVIEW a une influence sur la rapidité d'exécution, mais celle-ci demeure faible avec une fréquence d'échantillonnage (Fech) maximale de 100 Hz environ. De plus, si les essais nécessitent plus de temps (plus d'itérations de la boucle de commande), la fréquence maximale diminue aux environs de 60 Hz.

Tableau 3.5: Analyse de la plus petite période d'échantillonnage atteignable par le contrôleur du banc d'essai en asservissement de position

Essai	**Tech moyenne (ms)**	**Fech moyenne (Hz)**
norm;500	9,6	104
high;500	9,3	108
norm;5000	16,6	60
high;5000	16,3	61

En résumé au sujet de la rapidité de l'environnement d'exploitation de LabVIEW, il est clair que la plus grande fréquence d'échantillonnage atteignable dans cette implantation demeure faible (environ 100 Hz). De plus, comme pour la stabilité, les long essais sont à proscrire pour ne pas dégrader les performances en calcul de la boucle de commande.

Fiabilité générale de l'environnement

Deux éléments doivent être analysés pour fixer la fiabilité de l'environnement d'exécution: la fiabilité du système d'exploitation Windows et la fiabilité de l'environnement d'exploitation de LabVIEW.

Si LabVIEW effectue une bonne gestion de sa mémoire et que l'interaction avec Windows est optimisée par le fait que le processus est placé en haute priorité dans le gestionnaire de tâches, alors la stabilité est assurée. Cependant, les expériences passées avec LabVIEW lors d'essais menés dans le cadre des laboratoires académiques montrent que ce dernier devient vite instable (diminution de performances, logiciel qui fige) lorsque la mémoire disponible diminue (voir les essais précédents avec 5000 itérations de la boucle de commande). Il faut donc minimiser la longueur des essais pour ne pas occasionner de débordement en mémoire et assurer sa disponibilité maximale.

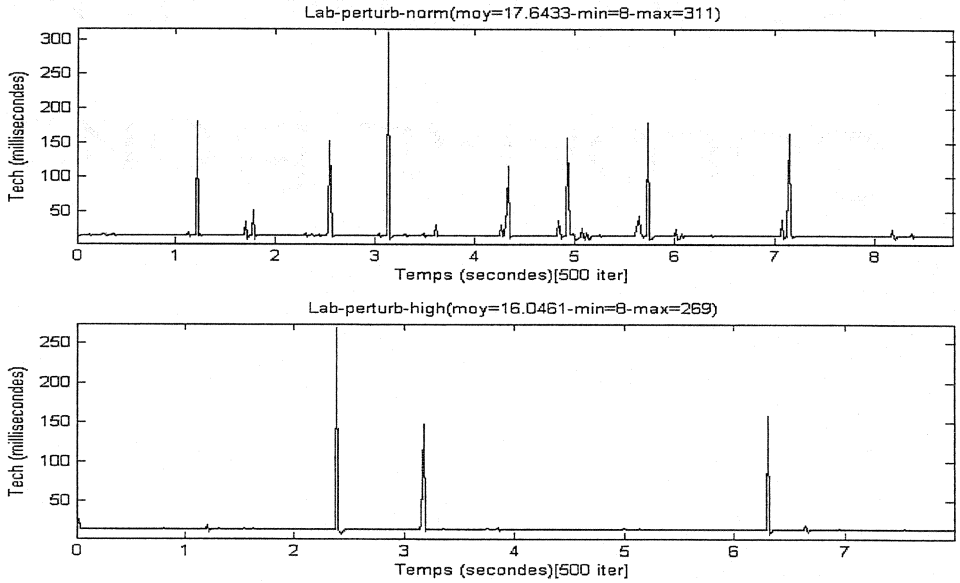


Figure 3.8: Fiabilité de la période d'échantillonnage dans LabVIEW en présence de perturbations

Autre point important, Windows est un système d'exploitation multitâches, ce qui implique que l'usager peut effectuer d'autres opérations alors que la boucle de contrôle est en exécution. Les perturbations alors engendrées par ces opérations diminuent la fiabilité de l'environnement d'exploitation de LabVIEW comme le montre la figure 3.8. Sur cette figure, on retrouve des essais montrant la période d'échantillonnage appliquée durant une exécution de la boucle avec LabVIEW en priorité *normale* (en haut) ou *haute* (en bas). Les essais comportent 500 itérations de la boucle de commande avec une période d'échantillonnage de 15 ms, pour une durée totale de 7.5 secondes (si la période

est respectée). Les perturbations lors des essais sont les suivantes:

- changements de fenêtre entre l'écran panneau et le diagramme du contrôleur;
- exécution d'un script de simulation dans l'environnement Matlab (déjà ouvert en arrière plan avant l'exécution de la boucle).

Le résultat de ces perturbations s'observe sous forme de pointes pouvant aller jusqu'à quelques centaines de millisecondes pour une période d'échantillonnage demandée de 15 ms. Ces excursions de la valeur de la période d'échantillonnage appliquée sont très dommageables pour les calculs du contrôleur. La haute priorité limite l'effet du basculement des fenêtres comme le montre le bas de la figure 3.8 avec moins de pointes et une erreur moyenne légèrement moindre (16.0 ms comparativement à 17.6 ms) pour sensiblement les mêmes perturbations. La haute priorité donne aussi un dépassement maximal légèrement inférieur avec 269 ms plutôt que 311 ms.

Au niveau de la fiabilité générale de Windows, si un programme comme un anti-virus (processus typiquement de très haute priorité) entre en fonction pendant l'exécution de l'algorithme, la dégradation des performances est importante. L'opérateur doit toujours assurer lui-même un certain niveau de sécurité à l'intérieur du système d'exploitation en limitant les opérations en parallèle et en libérant le plus possible la mémoire des programmes inutiles au contrôle comme l'antivirus.

En résumé au sujet de la fiabilité générale de l'environnement d'exploitation de LabVIEW, il est évident que l'interaction avec un système d'exploitation comme Windows pose problème, mais il est possible d'obtenir des performances acceptables en suivant quelques considérations d'optimisation comme le choix d'une période d'échantillonnage suffisamment grande et la gestion efficace de l'espace mémoire.

3.2.1.5 Environnement d'opération

Tous les essais du contrôleur se font strictement via le panneau de contrôle. Si ce dernier est bien conçu, l'usager a toutes les options pour assurer une opération efficace. Cependant, l'environnement conçu dans LabVIEW étant purement graphique, l'opérateur ne peut pas entrer de commandes pour changer la valeur d'une constante sans passer par

le panneau. La puissance de l'environnement d'opération est donc directement liée au talent du concepteur qui a mis au point le panneau. Même principe pour l'analyse de résultats, si c'est prévu par le diagramme en cours d'exécution, pas de problèmes; mais aucune commande ne peut être lancée suite à l'exécution de la boucle.

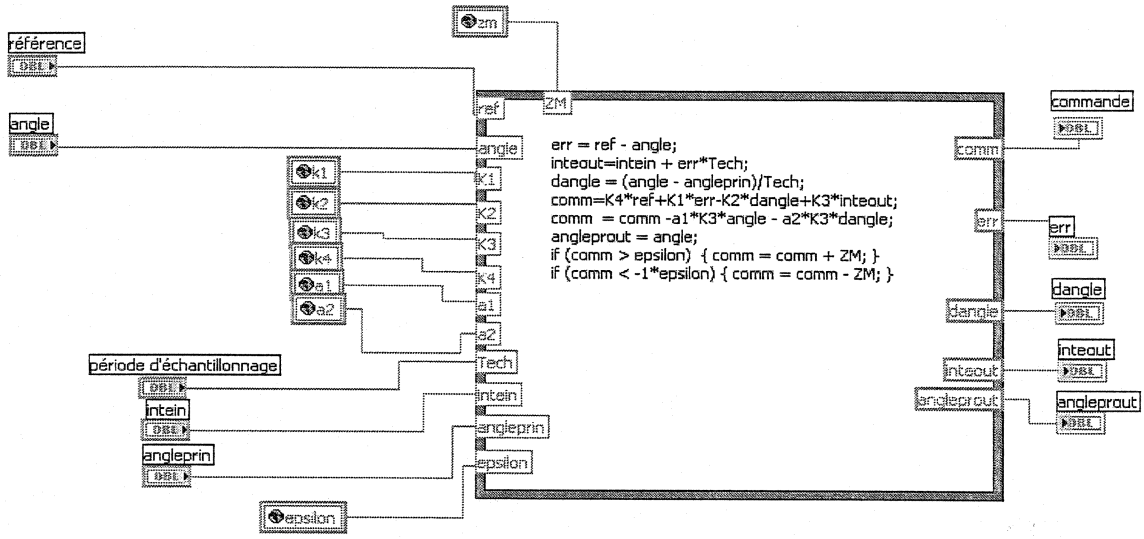
3.2.2 Méthode expérimentale pour les tests et l'analyse d'algorithmes de contrôle dans LabVIEW

Une fois le panneau réalisé et le diagramme de fonctionnement correctement implanté avec tous les blocs de sous-systèmes, le fichier *CTRL_PID.VI* peut être utilisé. Puisque l'environnement d'exploitation demeure Windows (comme pour la conception), il est possible de démarrer directement le contrôleur à partir du bouton *Run* de l'interface de LabVIEW. Pour maximiser les performances, la priorité *haute* est donnée à LabVIEW à partir du gestionnaire de tâches de Windows avant l'exécution de la boucle.

La suite de la procédure expérimentale est directe: choisir les paramètres désirés sur le panneau, exécuter le contrôleur, observer les résultats, conserver ceux qui sont intéressants et, éventuellement, les analyser avec Matlab. Si jamais il faut modifier des paramètres concernant les simulations implantées dans le diagramme, il faut ouvrir le fichier de paramètres globaux et y modifier les valeurs des paramètres concernés. Le diagramme principal utilisera ces nouvelles informations dès le début de la prochaine exécution.

Pour tester un algorithme de contrôle, il est nécessaire de le coder selon les normes de LabVIEW. La figure 3.9 présente le bloc contrôleur implanté dans LabVIEW en utilisant un bloc d'algorithme textuel afin de clarifier le résultat final. Ce sous-système est sauvé dans son propre fichier (*PIDDLK4_ZM.VI*) et est inséré dans le diagramme de la boucle de commande (voir la figure 3.3). L'algorithme implanté contient aussi une fonction de compensation de la zone morte du moteur qui ne fait pas partie de la théorie initiale du PID-DL. Les gains du contrôleur, ainsi que les paramètres du compensateur de zone morte proviennent du fichier des paramètres globaux

(*GLOBAL_PARAM.VI*). Ceci rend plus complexe le remplacement du bloc contrôleur[8], mais clarifie grandement le diagramme général du programme. Le reste des signaux sont passés au bloc lors de l'exécution via les connecteurs du bloc tel que présenté au bas de la figure 3.9.



CONNEXIONS DU BLOC:

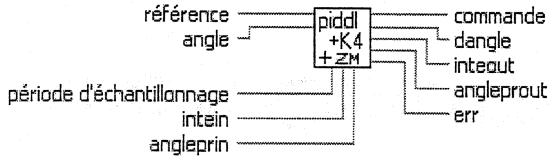


Figure 3.9: Implantation du PID-DL dans LabVIEW

Avec ce bloc inséré dans le diagramme principal, plusieurs essais peuvent être réalisés en variant le type de référence et les gains du contrôleur afin de tester l'algorithme. Puisque le diagramme principal contient aussi le simulateur numérique, il est possible de réaliser ces mêmes tests sur le modèle mathématique et ainsi chercher la correspondance entre la théorie et la pratique. Le changement de mode se réalise par un sélecteur au bas du panneau de contrôle qui permet de choisir entre *sim* (simulation) et *exp* (expérimentation). Des blocs de sélection dans le diagramme assurent le transfert des signaux vers le bon module (modèle de simulation ou bloc des entrées / sorties

8 Il faut ajuster le fichier des paramètres globaux et la sauvegarde des paramètres dans la boucle principale de commande pour chaque nouveau contrôleur.

réelles). Le détail de fonctionnement de ces blocs est présenté en annexe I.

L'interface du programme permet la récupération dans un fichier texte de résultats de toutes les données utiles à l'analyse soit:

- les gains utilisés;
- la référence utilisée (cas de référence fixe seulement);
- le nombre d'itérations de la boucle de commande;
- le vecteur des données de position;
- le vecteur des données de vitesse;
- le vecteur des données de vitesse filtrée;
- le vecteur des données de la commande;
- le vecteur des données de la période d'échantillonnage.

Le fichier brut de données comporte autant de lignes qu'il y a d'itérations dans la boucle, en plus des lignes de commande et de définitions assurant la compatibilité des résultats avec Matlab. Un exemple de ces résultats est montré à l'annexe I.

3.3 Implantation du banc d'essai dans xPC Target

Les algorithmes de la section 3.1 ont été également implantés dans Matlab / Simulink avec l'extension xPC Target. Essentiellement, le simulateur déjà développé pour les simulations du banc d'essai a été adapté afin de pouvoir commander le système réel avec les blocs d'entrée / sortie nécessaires.

La figure 3.10 illustre cette implantation sous la forme d'un schéma bloc dans Simulink dans le fichier *TEST_ASSERVISSEMENT.MDL*. Le détail des blocs se trouve à l'annexe II pour le lecteur intéressé à approfondir cette implantation. Une comparaison de l'implantation actuelle par rapport à l'implantation réalisée dans LabVIEW (voir la figure 3.3) démontre au départ la simplicité d'implantation dans Simulink. La différence principale qui permet cette clarification est que Simulink permet l'utilisation de fichiers scripts pour l'initialisation (voir figure 3.11), ce qui améliore la clarté des diagrammes.

La structure partagée de l'espace de données entre Matlab et Simulink permet également la gestion de plusieurs variables, dont des matrices qui contiennent la totalité

des résultats. Aucun besoin de créer un bloc spécial qui effectue la mise en tableau des données puisque des blocs d'analyse comme les afficheurs (*scopes*) permettent d'effectuer directement la sauvegarde des données dans l'espace de travail. Il est ensuite possible de décider par une simple commande si oui ou non les données sont conservées dans un fichier, sans devoir en faire la mention dès le début de la boucle (se référer au cas précédent dans LabVIEW).

Le schéma Simulink ne fournit pas de panneau pour le contrôle de la boucle, mais il est possible d'avoir l'équivalent. La fonction *guide* de Matlab fournit une aide pour la conception d'interface graphique. Il faut noter cependant que cette interface n'est pas essentielle pour le prototypage d'algorithmes de commande. Quelques scripts bien conçus et quelques manipulations dans Matlab permettent de récupérer et d'analyser les données rapidement et facilement. La figure 3.12 donne toutefois un exemple d'une interface graphique de base. Cette implantation a pris quelques heures et regroupe l'essentiel des fonctions offertes par le panneau de LabVIEW.

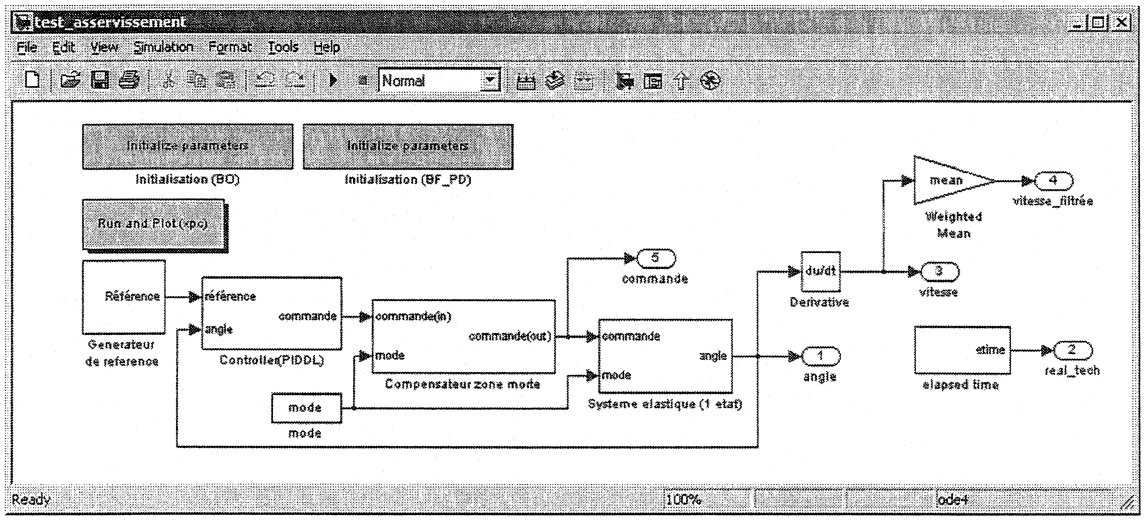


Figure 3.10: Implantation du contrôleur avec xPC Target

```
%Modèle rigide
km=13; taum=0.23;
%Modèle élastique (tf)
a=3.59 ; b=79.92; c=3.74; G=4698;
%Modèle élastique (états; matrices)
a21=-103; a22=9; a23=103; a41=99; a43=8; a44=-99; b21=139;
%Saturation logicielle de la commande
comm_max=7 ;
%Gains des compensateurs
k1=0; k2=0; k3=0; k4=1; a1=0; a2=0
%Paramètres de compensation de la zone morte
epsilon=0.1; w_seuil=1; compensation_zm=0.25;
%Paramètre de génération de la zone morte
zm_gain=1.5; zm_value=0.25; zm_tech=2;
%Résolution des encodeurs
res_fin=200; res_milieu=200;
%Période d'échantillonnage
tech=0.02; stoptime=10;
%Générateur de référence
type=1 %1=échelon; 2=sinusoide
reference=2; sin_frequence=0.015*2*pi; sin_decalage_dc=0;
%Filtre
mode=1; gf=[1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```

Figure 3.11: Script pour l'initialisation du contrôleur avec xPC Target

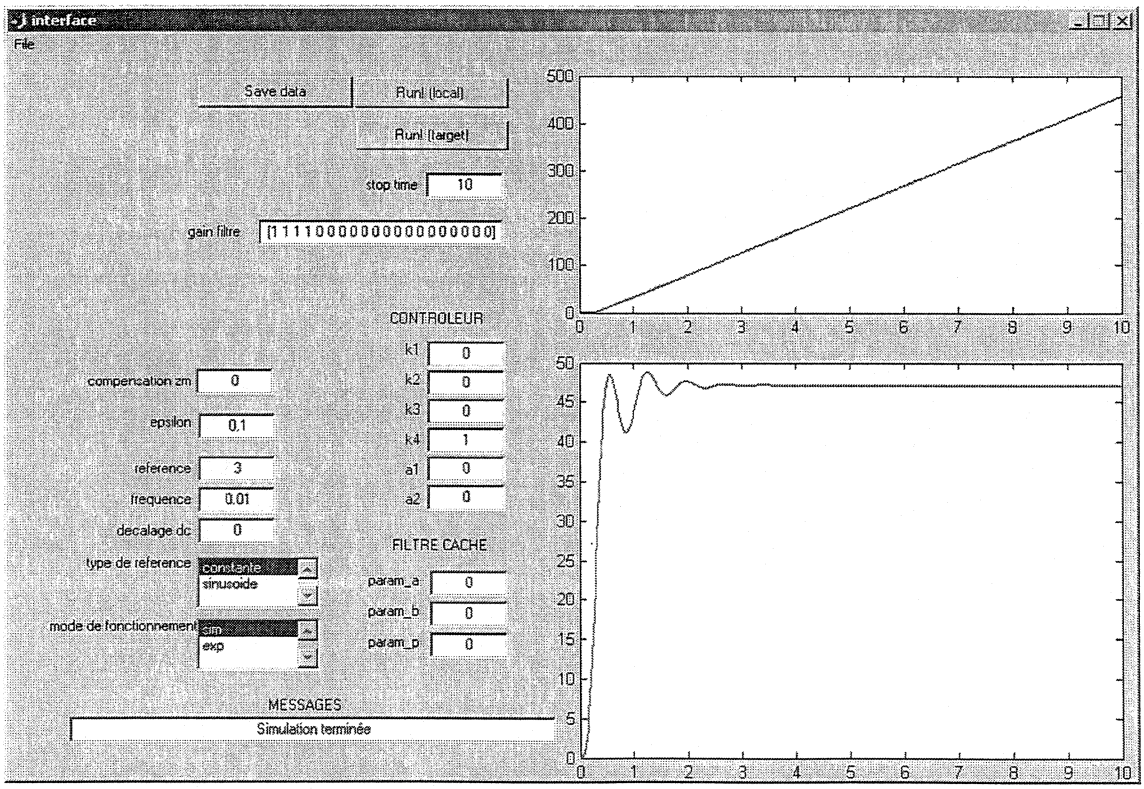


Figure 3.12: Exemple d'interface graphique pour le banc d'essai avec xPC Target

3.3.1 Évaluation initiale de l'implantation dans xPC Target

Voyons comment l'environnement Matlab / Simulink / xPC Target se classe par rapport aux spécifications recherchées pour le choix d'environnement en reprenant les points présentés au premier chapitre de ce mémoire.

3.3.1.1 Compatibilité des éléments de la boucle

Au niveau de la compatibilité des éléments de la boucle, l'environnement Matlab ne pose aucun problème. En effet, tous les éléments nécessaires à la réalisation des contrôleurs se trouvent dans les librairies de Matlab: les pilotes des interfaces d'entrée / sortie se trouvent dans la librairie de xPC Target; les fonctions de récupération des résultats se trouve dans les librairies de xPC Target et Simulink; les fonctions d'analyse des données se trouvent dans les différentes librairies de Matlab. Les interconnexions entres tous les éléments sont assurées par le fait que tous les produits appartiennent à une seule et même compagnie et partagent le même espace de variables.

Puisque la banque de pilotes de xPC Target regroupe plus de 150 cartes, une analyse préalable lors de la sélection matérielle des éléments du banc d'essais permet de limiter le travail de conception. Fait intéressant, les cartes d'entrées / sorties du banc d'essais choisies il y a une dizaine d'années sont supportées par la librairie, ce qui permet de conclure qu'elle contient plusieurs cartes fréquemment utilisées. Les cartes avaient initialement été choisies selon ce critère.

3.3.1.2 Protocole d'échange de données

À la base, Matlab permet l'utilisation de n'importe quel type de communication suivant la programmation du protocole dans Matlab directement ou dans un langage comme le C++. Avant de songer à la programmation, il faut noter que xPC Target supporte la communication sérielle (protocole RS-232) ou réseau (protocole TCP/IP) et que Matlab aussi supporte directement ces protocoles. Finalement, puisque xPC Target assure la tenue d'un journal de données et permet la communication interactive de commandes, la communication et l'échange de données se font sans problèmes.

L'implantation réalisée pour cette étude de cas utilise notamment le protocole TCP/IP pour la communication entre xPC Target et Matlab. Pour la communication avec les cartes d'E/S, xPC Target supporte les BUS utilisés qui sont de type ISA et PCI.

Pour conserver les données, une sauvegarde de l'espace de travail de Matlab (fonction *uisave*) suffit à remplir les besoins puisque la récupération de cet espace par Matlab est possible par la suite (commande *uiload*).

3.3.1.3 Environnement de conception

L'environnement de conception étant principalement Simulink avec quelques incursions dans Matlab pour la configuration des paramètres, on peut affirmer que l'environnement est principalement graphique, et donc très simple à utiliser et à comprendre pour un débutant dans le domaine. Puisque la compilation du modèle et le transfert sur le poste xPC Target sont automatiques, l'effort n'est pas placé dans la programmation textuelle. La possibilité de créer des sous-systèmes qui sont conservés à même le fichier principal permet d'alléger la représentation graphique sans pour autant ajouter de nouvelles dépendances entre les fichiers.

La partie textuelle de l'environnement permet de créer des fichiers scripts assurant l'initialisation et le transfert des paramètres de fonctionnement à tous les blocs concernés via l'espace de mémoire partagée. Ceci allège beaucoup les schémas et permet de diviser les traitements en deux sections: paramètres de fonctionnement et calculs. Le diagramme Simulink n'a que cette dernière partie à traiter.

Matlab étant avant tout un environnement de mathématiques appliquées, il contient aussi des blocs fonctionnels comme la dérivée et l'intégration numérique, des blocs qui ne sont pas disponibles directement dans LabVIEW et qui sont essentiels à l'implantation des boucles de contrôle.

3.3.1.4 Environnement d'exploitation

Les spécifications présentées par la compagnie au sujet de la rapidité et de la stabilité de l'environnement xPC Target sont le point de départ du choix de cet

environnement. Cette section vise à vérifier l'exactitude de ces renseignements dans un cadre expérimental. Il faut noter que l'environnement d'exploitation de xPC Target est propre au logiciel et qu'il est complètement indépendant de Windows, le système d'exploitation utilisé avec l'environnement de conception (Matlab / Simulink).

Stabilité temporelle

L'environnement xPC Target surveille à l'interne l'application de la période d'échantillonnage demandée. Si cette dernière n'est pas respectée, alors le système en informe l'usager et arrête immédiatement[9] l'exécution de la boucle. La stabilité ne devrait donc pas poser un problème. Pour s'en assurer, il suffit d'utiliser la mesure du temps d'exécution de la tâche (TET) qui est effectuée par l'environnement d'exploitation de xPC Target. Cette mesure illustre directement le temps passé par l'environnement pour effectuer les calculs de l'algorithme actuellement en exécution. La figure 3.13 illustre les résultats obtenus pour deux périodes d'échantillonnage différentes (1 ms et 15 ms). La moyenne du TET est affichée en haut de chaque courbe pour fins de référence. Puisque cette valeur est toujours inférieure à la période d'échantillonnage demandée, il est possible d'affirmer que l'environnement n'a aucune difficulté à conserver sa période stable.

En fait, aucun moyen direct de mesure de la période d'échantillonnage réellement appliquée n'est disponible dans xPC Target. Cependant, un journal des instants précis d'échantillonnage est disponible, le vecteur du temps d'exécution. La consultation de ce journal montre que la période d'échantillonnage est rigoureusement appliquée. Pour s'en convaincre, il suffit d'observer l'écart temporel entre deux mesures qui correspond à la période d'échantillonnage ($dt(i)=t(i)-t(i-1), pour \forall i \in [2,n]$ avec dt la période d'échantillonnage appliquée, t le vecteur de temps, i l'itération actuelle et n le nombre d'itérations dans le vecteur t).

La figure 3.14 présente les résultats obtenus. Les droites constantes à la valeur demandée illustrent bien la stabilité dans tous les cas d'exécution, soit pour des périodes

9 L'usager a la possibilité de configurer cette action pour que le noyau d'exploitation ne fasse qu'envoyer un avertissement à l'ordinateur hôte, mais cette option est désactivée par défaut.

d'échantillonnage de 1 ms et 15 ms avec une courte (500 itérations) et une longue (5000 itérations) exécutions. Le tableau 3.6 montre le résumé des résultats de ces essais qui représentent exactement les attentes à l'égard d'un environnement d'exploitation en temps réel: aucun jeu dans la période d'échantillonnage appliquée et une correspondance exacte entre la valeur demandée et la valeur appliquée.

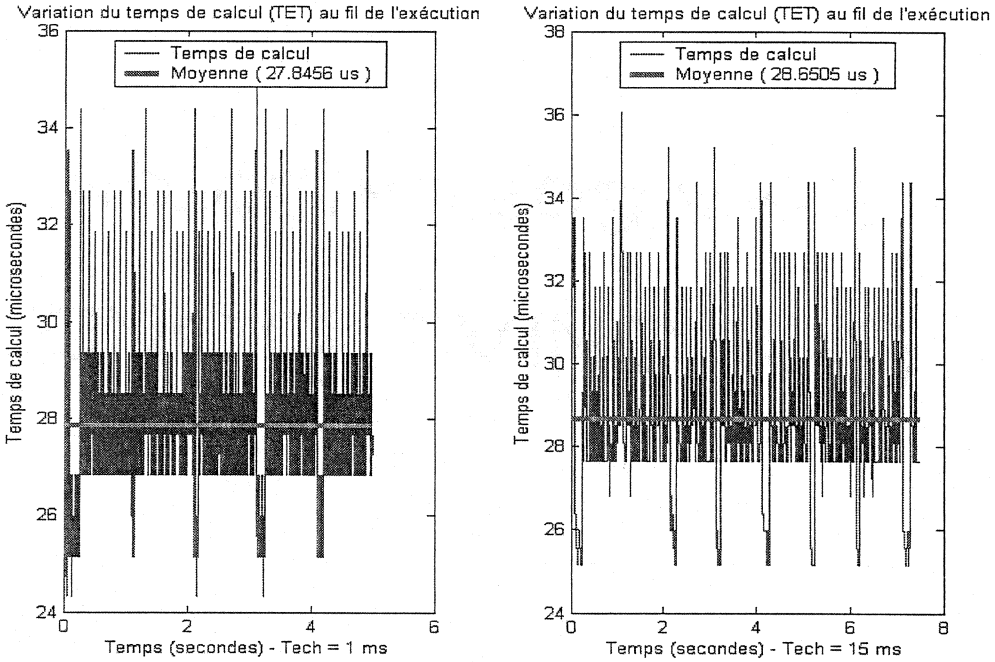


Figure 3.13: Test du temps d'exécution de la tâche dans xPC Target

En fait, ces résultats semblent trop parfaits et un test supplémentaire est réalisé pour les valider. Un oscillateur unitaire est ajouté dans le schéma du système comme le monte la figure 3.15. Cet oscillateur produit un signal 0-5 Volts qui change d'état à chaque itération de la boucle de commande. Ce signal est relié à la deuxième sortie physique de la carte d'E/S analogiques, la carte PCI-6025E, et il est mesuré sur un oscilloscope lors de l'exécution. Théoriquement, le signal obtenu doit avoir une période correspondant au double de la période d'échantillonnage demandée (2 itérations pour un cycle du signal).

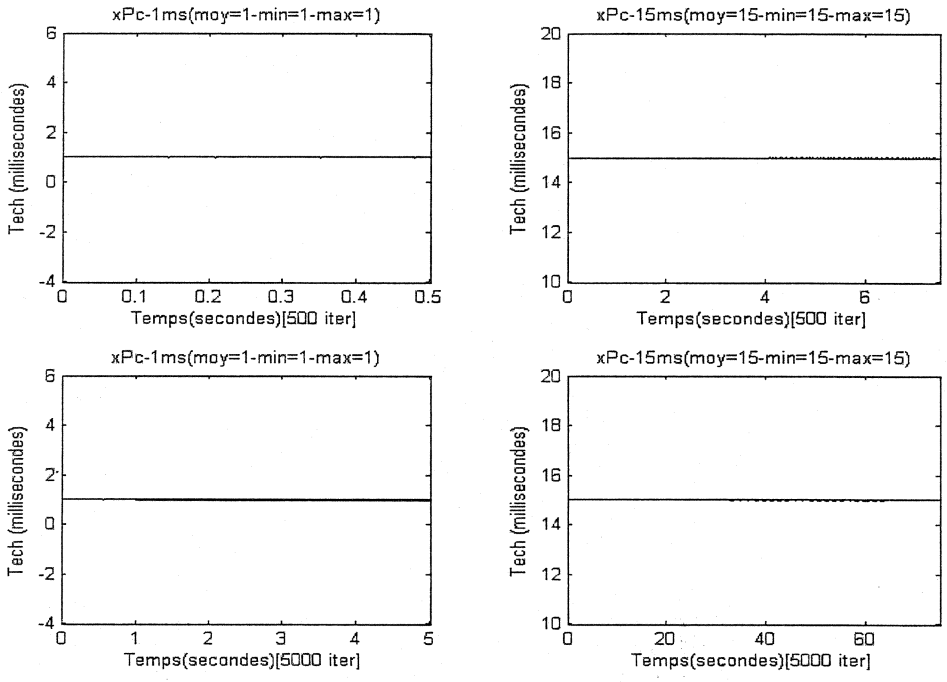


Figure 3.14: Stabilité de la période d'échantillonnage réelle dans xPC Target

Tableau 3.6: Effet de la longueur des essais sur les variations de la période d'échantillonnage

Essai	Moyenne (ms)	Minimum (ms)	Maximum (ms)	Jeu (ms)	Écart (ms)
1ms;500	1	1	1	0	0
15ms; 500	15	15	15	0	0
1ms;5000	1	1	1	0	0
15ms;5000	15	15	15	0	0

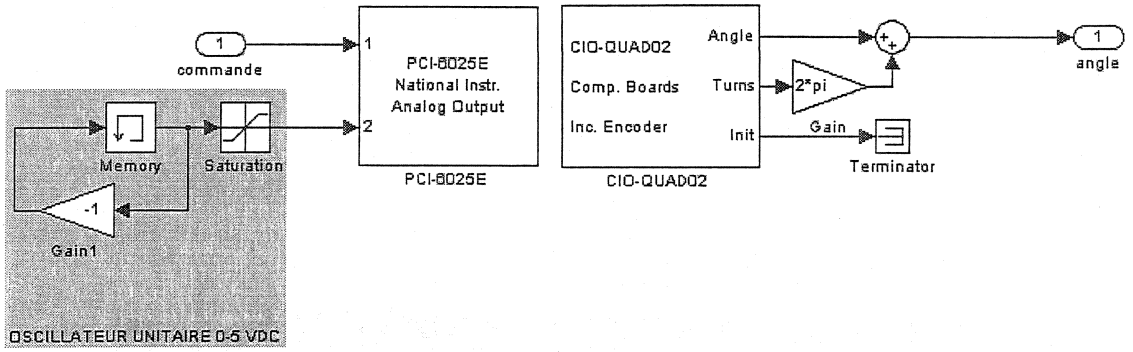


Figure 3.15: Implantation de l'oscillateur unitaire sous xPC Target

Pour obtenir les mesures, un oscilloscope numérique est utilisé (modèle TDS-2012 de Tektronix [30, 31). Il est équipé d'un port de communication série RS-232, ce qui permet la récupération des mesures dans Matlab. La moyenne et la fréquence du signal mesuré sont calculées par l'oscilloscope, ce qui rend les calculs indépendants de xPC Target. Les fonctions de communication entre l'oscilloscope et l'ordinateur ont été entièrement développées dans le cadre de ce projet à partir des documents fournis par le manufacturier. Les modules sont présentés dans un document en préparation qui est inclus à l'annexe III pour fins de références. Le code source en langage M de ces fonctions est disponible sur demande à la section Automation et Systèmes de l'École Polytechnique de Montréal.

Deux essais ont été effectués pour valider les résultats présentés au tableau 3.6. Le premier utilise une période d'échantillonnage de 1 ms et le deuxième de 15 ms. La figure 3.16 (voir section suivante) illustre les résultats obtenus, soit une fréquence de 500 Hz pour le premier essai et une fréquence de $33.\overline{3}$ Hz pour le deuxième. Ceci donne une période d'échantillonnage de 1 ms et 15 ms respectivement (T=1/2F selon la considération mentionnée plus tôt), illustrant une excellente stabilité temporelle.

Rapidité d'exécution

Comme base d'analyse, *The Mathworks* fournit un banc d'essais qui permet de voir les performances de différents processeurs utilisés comme cible pour l'exécution

d'un modèle générique à complexité variable (le modèle utilisé, *F14*, est documenté avec la fonction *xpcbench* utilisée pour lancer le banc d'essais). Ce banc d'essais permet d'avoir une bonne idée des performances atteignables à partir d'un processeur donné. Connaissant cette performance relative, un processeur peut être sélectionné en fonction des besoins de l'algorithme de contrôle. La figure 3.17 présente les résultats de ce banc d'essais quant aux performances.

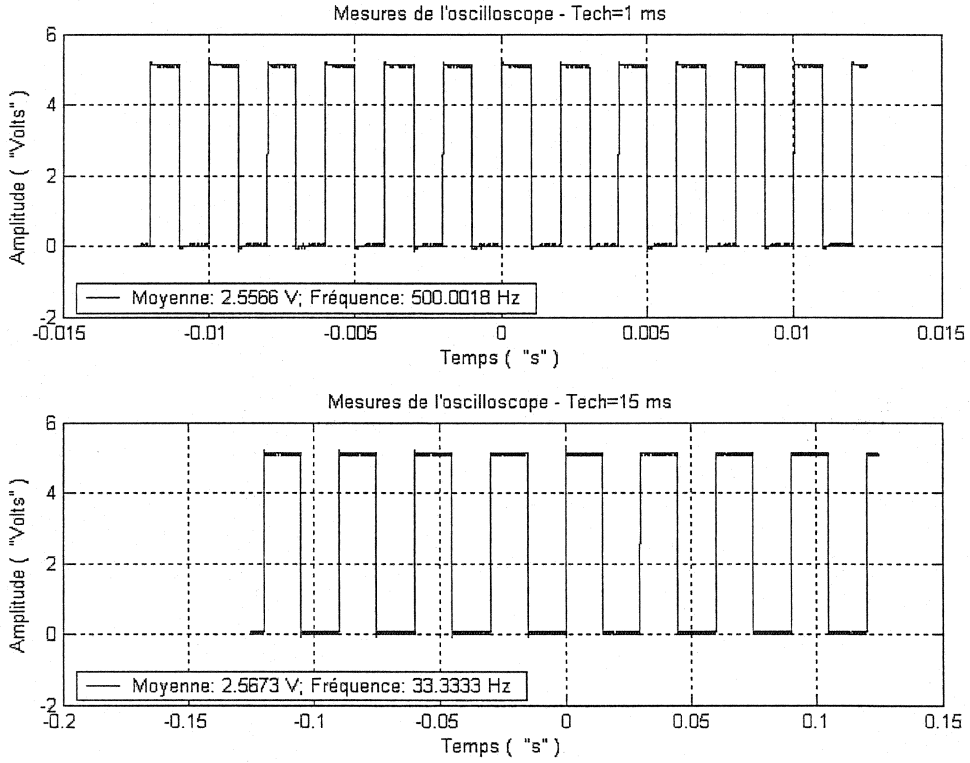


Figure 3.16: Stabilité de la période d'échantillonnage mesurée à l'oscilloscope

Selon les résultats, il n'est pas nécessaire de se doter d'un processeur très puissant pour atteindre des performances adéquates dans xPC Target. Sachant qu'une période d'échantillonnage de l'ordre de la milliseconde est suffisante dans la majorité des cas de contrôle, un processeur à 400 MHz suffit amplement à la tâche puisqu'il exécute avec une période de 755 µs un modèle très complexe (F14*25 est un modèle très exigeant).

Concrètement, lors des essais sur le banc d'essai en asservissement de position, un Pentium II à 366 MHZ est utilisé comme cible pour xPC Target. La fonction *xpcbench* permet aussi d'inclure l'argument *this* (*xpcbench('this')*) pour effectuer le test

directement sur la cible en utilisation, d'où la mention *This machine* sur la figure 3.17 qui donne le résultat pour l'ordinateur retenu.

Minimal achievable sample times in µs

	Minimal	F14	F14*5	F14*10	F14*25
AMD Athlon 1GHz	10	13	24	41	109
Intel P4 1.5GHz	10	15	31	56	133
Intel PIII 600MHz	8	13	39	97	363
Intel PII 400MHz	8	16	58	135	502
This Machine	11	20	61	128	449
AMD K6-2 400MHz	10	24	75	159	755
Intel PI 166MHz	11	44	253	567	1681
Intel PI 90MHz	27	100	590	1400	4600
Intel 486DX 40MHz	43	375	1900	3800	12500

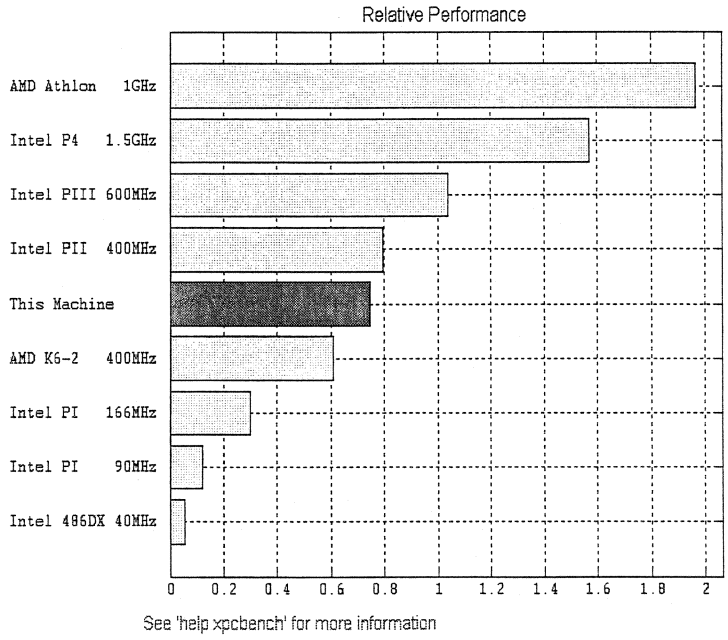


Figure 3.17: Banc d'essai en performance de xPC Target

Fiabilité générale de l'environnement

Puisque l'outil propose son propre environnement d'exploitation, il est clair que tous les paramètres sont théoriquement surveillés pour assurer les meilleures performances et le maintien de ces dernières. Avec les outils de transfert, de supervision et de gestion fournis par le noyau d'exploitation, toutes les informations nécessaires pour s'assurer de la fiabilité de l'environnement peuvent être obtenues.

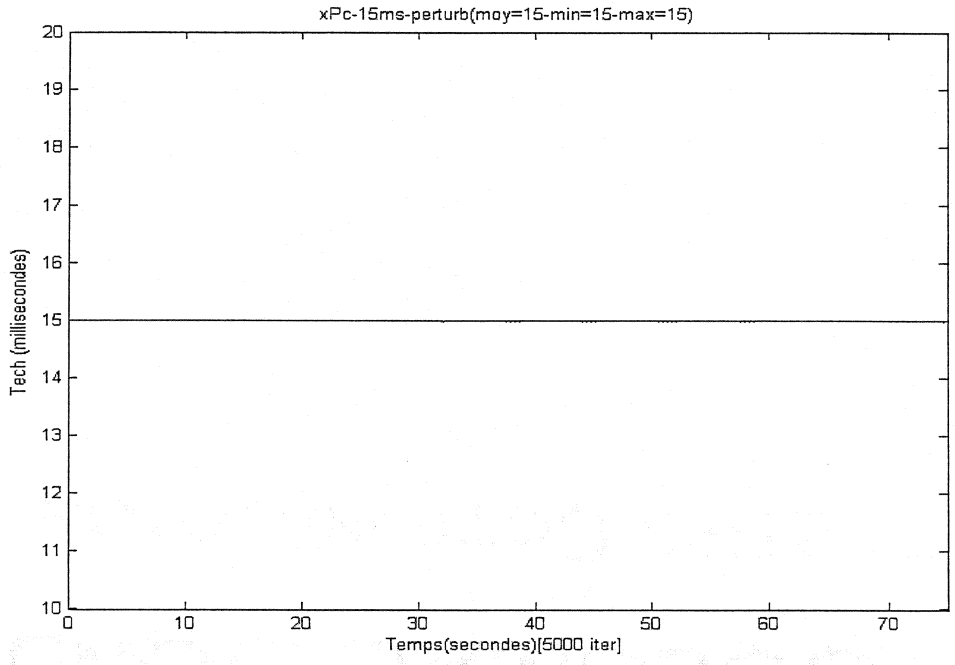


Figure 3.18: Fiabilité de la période d'échantillonnage dans xPC Target en présence de perturbations

Au niveau de la fiabilité temporelle, la figure 3.18 présente une courbe tout aussi droite que lors des tests sans perturbation malgré le fait que les perturbations ont été poussées à l'extrême. En effet, le logiciel LabVIEW a été démarré sur l'hôte pendant l'exécution et des commandes invalides ont été introduites sur l'interface du poste cible qui devait aussi présenter des graphiques de l'évolution des résultats en cours d'exécution. Malgré toutes ses perturbations, une droite constante à la période demandée est obtenue, ce qui confirme la fiabilité de l'environnement d'exécution.

À ce jour, la seule lacune décelée au sujet de l'environnement d'exploitation est la sensibilité de son pilote de carte réseau. En effet, si le poste cible et le poste hôte se

trouvent sur un réseau vaste, la probabilité de collision des paquets TCP/IP nuit au bon fonctionnement de xPC Target (erreur de lecture / écriture TCP/IP dans Matlab). Des erreurs de communication apparaissent alors que toutes les applications réseau fonctionnent très bien de leur côté. Cette sensibilité est jugée normale puisque la contrainte temporelle associée à l'aspect temps réel du noyau xPC Target l'empêche d'attendre la réception des paquets perdus. Ce problème est toutefois rapidement résolu en plaçant les deux ordinateurs (hôte et cible) sur un réseau local isolé du réseau global. Cette stratégie est d'ailleurs conseillée pour toute recherche utilisant ce type d'environnement informatique afin de minimiser les délais de transmission.

3.3.1.5 Environnement d'opération

L'environnement d'exploitation de xPC Target prévoit plusieurs commandes qui permettent une interface complète avec le processus en cours d'exécution. Ainsi, il est possible de programmer une interface très complète dans Matlab (ou un autre logiciel) et d'interagir ensuite avec l'ordinateur cible pour, par exemple, consulter l'état d'exécution, modifier les paramètres du modèle en exécution, récupérer les données partielles, contrôler l'exécution, vérifier le respect des contraintes, etc. Selon le cas, une interface graphique évoluée peut être implantée ou non en conservant des possibilités d'interaction complète. Il est possible de se placer n'importe où dans l'espace défini par la figure 1.4 selon les scripts et les fonctions utilisées par l'interface.

Habituellement, une interface graphique est programmée pour fournir les fonctions d'interface de base, puis la gestion des données et l'analyse des résultats peut se faire avec des commandes directes envoyées via Matlab au processus en cours d'exécution.

Pour fonctionner sur la cible, les modèles Simulink doivent être compilés. Ceci implique que les changements effectués sur le schéma Simulink ne sont pas automatiquement pris en compte par la cible et qu'il faut théoriquement recompiler tout le code et le transférer de nouveau à la cible. Toutefois, des méthodes sont fournies par xPC Target permettant la conception d'un script pour changer la valeur des différents

paramètres du modèle compilé sans devoir reprendre la compilation et le transfert. Ce script permet de réaliser tous les essais sans devoir recompiler le modèle à chaque nouvel ensemble de paramètres. Le détail concernant ces méthodes et l'exécution d'un modèle dans xPC Target est donné dans la littérature accompagnant le logiciel.

L'avantage certain de l'environnement d'opération conçu dans Matlab est de permettre de regrouper les essais et leur analyse, ce qui accélère beaucoup l'exploration de nouveaux algorithmes de commande. La possibilité de conserver les données autant sous la forme graphique (fichiers de figures **.fig*) que sous la forme binaire (fichiers de données **.mat*) **suite à l'exécution**, permet une bonne gestion des résultats.

3.3.2 Méthode expérimentale pour les tests et l'analyse d'algorithmes de contrôle dans xPC Target

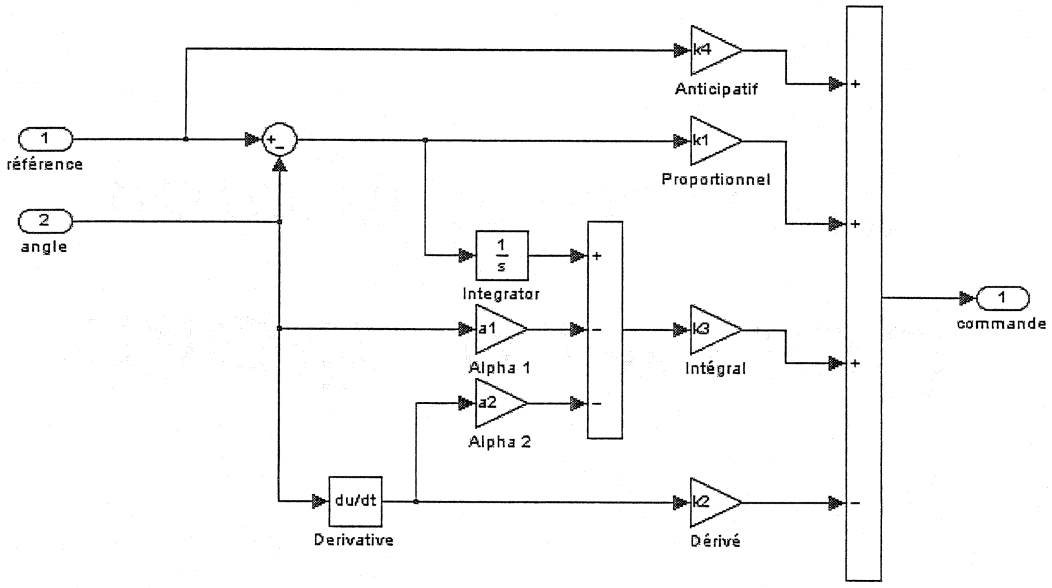
Puisque l'extension xPC Target est utilisée, la méthode expérimentale est très différente du cas où seuls Matlab et Simulink sont utilisés. Tout d'abord, le deuxième ordinateur qui sert exclusivement de cible est démarré avec le noyau xPC Target comme système d'exploitation (Matlab permet de créer une disquette de démarrage à cet effet) et il doit être relié au même réseau local que l'ordinateur de conception (hôte).

Lors de la réalisation d'un test, la boucle est implantée dans Simulink avec l'algorithme de commande à tester. L'option de compilation fournie par l'extension RTW de Simulink est ensuite utilisée. La compilation se termine par le transfert automatique de l'application vers la cible. Si tout se passe bien (aucune erreur de compilation et de transfert), la cible est prête à exécuter le code.

Pour le traitement des données, xPC Target offre la possibilité de conserver un journal de données récupérable à la fin de l'exécution. La commande *getlog* permet la récupération et les commandes de Matlab permettent d'analyser, de sauvegarder et d'afficher les données.

Pour tester un algorithme de contrôle avec cet environnement informatique, il faut le coder selon les normes de l'environnement. Pour la démonstration, l'algorithme PID-DL a été implanté à l'aide des blocs fonctionnels disponibles dans la librairie de

Simulink. La figure 3.19 présente le schéma réalisé; le bas de la figure montre le bloc sous-système qui le contient. Les paramètres de fonctionnement du contrôleur (et de tous les autres blocs d'ailleurs) sont communiqués via l'espace de travail de Matlab, c'est-à-dire qu'il suffit de les déclarer dans un fichier script pour que tous les blocs puissent les connaître.



CONNEXIONS DU BLOC:

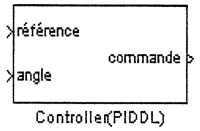


Figure 3.19: Implantation du PID-DL dans xPC Target

L'avantage de Simulink est que tout le code se trouve dans un seul fichier et que les paramètres sont tous définis directement dans l'espace de travail de Matlab. Les modifications du diagramme (ajout / retrait d'un bloc, changement de l'algorithme de contrôle) sont ainsi simplifiées. L'interface graphique réalisée pour cette implantation modifie seulement la valeur des paramètres dans l'espace de travail sans autre interaction avec le schéma Simulink. Cette simplicité est bénéfique dans le cadre de l'objectif de prototypage rapide d'algorithmes de contrôle. D'ailleurs, la simplicité s'applique aussi aux autres blocs comme le générateur de référence qui peut être adapté aisément en

fonction des besoins des essais. Il suffit de remplacer le bloc dans le schéma principal et de définir ses paramètres de fonctionnement dans le script de configuration ou directement dans l'espace de travail.

Essentiellement, les mêmes tests qu'avec l'implantation LabVIEW peuvent être effectués:

- simulation ou expérimentation en changeant le paramètre *mode* dans la configuration initiale (le transfert des signaux de contrôle et de données est automatiquement réalisé dans le schéma Simulink);
- ajustement des paramètres du simulateur (variables dans l'espace de travail de Matlab);
- ajustements des paramètres du contrôleur / changement de contrôleur (variables dans l'espace de travail et bloc aisément remplaçables dans le schéma Simulink).

Tous les paramètres se retrouvent déjà dans l'espace de travail et les données aussi une fois qu'elles sont récupérées de la cible via la commande *getlog*. Pour conserver les données, il suffit de sauvegarder l'espace de travail via la commande *uisave* de Matlab. Les données sauvegardées à la fin de l'exécution sont entre autres:

- les gains utilisés;
- la référence utilisée;
- le nombre d'itérations de la boucle de commande;
- le vecteur des données de position;
- le vecteur des données de vitesse;
- le vecteur des données de vitesse filtrée;
- le vecteur des données de la commande;
- le vecteur des données de la période d'échantillonnage.

Puisque les données sont analysées dans le même environnement, il est aisé de vérifier ces dernières directement lors des essais pour conserver seulement les meilleures. La dernière section de l'annexe II fournit un exemple des données. Différents scripts sont utilisés pour conserver les mêmes noms de vecteurs de données

que dans l'implantation précédente, afin de pouvoir utiliser les mêmes scripts d'analyse dans chaque cas.

3.4 Comparaison des résultats obtenus

Suite à la présentation de deux implantations différentes, il est important de les analyser en les comparant car chacune possède ses avantages et ses inconvénients. Les résultats présentés aux annexes I et II montrent que les résultats obtenus avec chaque implantation sont semblables, donc ce n'est pas à ce niveau que la différenciation est effectuée. En passant outre les résultats, l'objectif dans cette section est de comparer point par point les deux environnements informatiques.

3.4.1 Comparaison des environnements de conception

Au niveau des environnements de conception, il y a une différence nette entre les deux environnements informatiques. LabVIEW utilise une conception à deux sections entièrement interdépendantes, tandis que xPC Target n'utilise essentiellement que le diagramme Simulink. Ce partage dans le cas de LabVIEW rend le remplacement des blocs principaux beaucoup plus complexe puisque ces derniers se trouvent à la fois dans chacune des sections. Le retrait (ou l'ajout) d'un bloc dans l'une des sections peut provoquer des conflits dans l'autre. Le contrôleur en est un bon exemple: il dépend à la fois du panneau où se trouvent les boîtes de saisie des gains, du fichier de paramètres globaux où se trouvent les valeurs actuelles de ces gains et du diagramme où se trouve physiquement le bloc du contrôleur. Avec xPC Target, le bloc est indépendant des autres blocs du schéma, avec seulement la définition de ses paramètres dans l'espace de travail de Matlab. L'interface graphique qui peut être ajoutée dans cet environnement est relié strictement aux variables de l'espace de travail sans créer d'interdépendances entre les blocs.

L'espace de travail commun de Matlab / Simulink évite aussi de devoir fournir une interface complexe de sauvegarde des résultats, puisque ces derniers se trouvent dans l'espace de travail dès leur récupération. Pour le transfert vers la cible, quelques

commandes permettent le passage direct de paramètres vers l'environnement d'exploitation de xPC Target sans ajouter de liens complexes.

Il est clair que xPC Target convient mieux à la tâche dans le cadre d'une implantation variable des algorithmes de contrôle. La force de LabVIEW demeure au niveau de la programmation intuitive de l'interface, en parallèle avec le diagramme de fonctionnement. Cependant, le remplacement de blocs est une opération plus complexe dans cet environnement.

3.4.2 Comparaison des environnements d'exploitation

En présumant une implantation efficace dans chaque environnement informatique, les tests sur la période d'échantillonnage sont les plus significatifs. Il est clair à ce niveau que xPC Target permet une bien meilleure performance temporelle que LabVIEW comme le démontrent les figures 3.5, 3.6, 3.8, 3.13, 3.14, 3.16, 3.17 et 3.18 et les tableaux 3.3, 3.4, 3.5 et 3.6.

Autre point de comparaison concernant les environnements d'exploitation: l'exécution de la boucle localement (LabVIEW) versus l'exécution en mode hôte – cible (xPC Target). La rapidité nettement supérieure de xPC Target est obtenue en utilisant un deuxième ordinateur relié en réseau. Cela implique du matériel supplémentaire qu'il faut considérer lors de la comparaison. Toutefois l'abaque de la figure 3.17 montre que des ordinateurs plus anciens comme un Pentium I à 166 MHz permettent l'obtention de bonnes performances, ce qui n'engendre pas de coûts excessifs pour une nette amélioration de performances. De plus, l'interaction entre les deux ordinateurs est simple via l'utilisation de scripts qui permettent d'automatiser la séquence de mise en fonction et de récupération des données.

3.4.3 Comparaison des environnements d'opération

Au niveau de l'environnement d'opération, l'équivalence entre les deux environnements informatiques est difficile à discuter, car la simplicité d'opération est

fonction des interfaces fournies lors de l'implantation de la boucle de contrôle. Une implantation bien réalisée implique que ce n'est pas au niveau de cet environnement que les performances vont se départager. Il est cependant possible de discuter de la difficulté relative de fournir un environnement d'opération complet dans chacun des environnements informatiques.

D'abord, au niveau des interfaces offertes, LabVIEW ne permet pas l'envoi textuel de commandes: toute information passe par le panneau de contrôle ou par la modification du diagramme de fonctionnement. Il faut donc prévoir une interface très complète via le panneau afin de pouvoir modifier les paramètres d'opération de la boucle de commande. Sous Matlab, le simple fait d'utiliser des variables de fonctionnement dans l'espace de travail partagé permet de donner à l'usager la possibilité de modifier les paramètres de fonctionnement. La simplicité offerte à l'opérateur par une interface graphique demeure souhaitable, or les résultats présentés démontrent qu'il est possible d'en inclure une sous Matlab.

Ensuite, au niveau de l'analyse des données, il est clair que Matlab offre beaucoup plus de possibilités puisqu'il permet une analyse post-exécution, ce que LabVIEW ne permet pas. Lors de l'étude d'algorithmes de commande, l'analyse post-exécution est souvent plus importante que l'analyse en cours d'exécution. Pour pallier à cette absence dans LabVIEW, les données obtenues sont récupérées et envoyées dans Matlab via un fichier texte pour être analysées. Ainsi, si Matlab est utilisé du début à la fin, il n'est pas nécessaire de sauvegarder les données intermédiaires, seuls les bons résultats sont conservés. Beaucoup plus de fonctions d'analyse en cours d'exécution auraient pu être incluses dans l'implantation LabVIEW pour limiter les échanges avec Matlab, mais cela aurait dégradé les performances du contrôleur sans pour autant fournir toutes les fonctions d'analyse nécessaires.

3.5 Conclusions de la première étude de cas

Ce chapitre regroupe de nombreux éléments d'analyse des deux environnements informatiques retenus pour l'étude de cas. Le meilleur moyen de regrouper ces éléments

est de les placer dans un tableau résumant chaque point d'analyse et le résultat obtenu pour chaque environnement. Les tableaux 3.7 et regroupent ces résultats.

L'observation de ces résultats synthétiques permet deux conclusions principales pour cette première étude de cas:

- l'environnement d'exploitation de xPC Target est beaucoup plus performant (stabilité, rapidité et fiabilité);
- l'environnement de conception de xPC Target convient beaucoup mieux au prototypage rapide d'algorithmes de commande (simplicité de remplacement des blocs et de configuration générale de fonctionnement).

LabVIEW convient beaucoup mieux à un environnement industriel fixe où l'interface avec l'opérateur est une priorité et où le temps d'apprentissage pour l'utilisation doit être minimal. La différenciation ne se fait pas au niveau des résultats d'exécution de la boucle ouverte comme le montrent les résultats à la fin des annexes I et II.

Chaque implantation fournie dans le cadre de cette première étude de cas est fonctionnelle, ce qui remplit l'objectif de maîtrise des environnements. Du même coup, deux environnements complets sont maintenant disponibles pour permettre l'étude d'algorithme dans un contexte de recherche (xPC Target surtout) ou d'utilisation en laboratoire (LabVIEW surtout).

Tableau 3.7: Résumé de l'analyse de l'environnement informatique lors de la première étude de cas

Élément d'analyse	Résultat LabVIEW	Résultat xPC Target
COMPATIBILITÉ DES ÉLÉMENTS DE LA BOUCLE		
Tous les éléments de la boucle sont disponibles	NON (pilote carte CIO_QUAD02 absent des librairies disponibles)	OUI (vaste librairie de blocs offerts incluant les cartes d'E/S utilisées)
Ajout d'élément possible par programmation	OUI (programmation possible du pilote)	OUI (programmation en C++ avec un compilateur intégré à Matlab)
PROTOCOLE D'ÉCHANGE DE DONNÉES		
Types utilisés	Transfert des données sur le disque dur.	Communication TCP/IP avec la cible lors de l'exécution.

Élément d'analyse	Résultat LabVIEW	Résultat xPC Target
ENVIRONNEMENT DE CONCEPTION		
Implantation combinée possible (sim. / exp.)	OUI (légèrement complexe)	OUI (simple)
Présence des outils d'analyse	NON	OUI
Facilité d'utilisation (algorithmes variables)	Moyen (blocs difficiles à remplacer et à relier).	Simple
Possibilité d'interface graphique	OUI (directement intégrée dans le logiciel)	OUI (il faut l'implanter avec la fonction *guide* de Matlab)
Type d'environnement	Graphique	Combiné
Couverture du cycle de développement (fig. 1.3)	Commande d'un procédé physique bien défini.	Matlab en général couvre la totalité du diagramme.
ENVIRONNEMENT D'EXPLOITATION		
Stabilité de la période d'échantillonnage	Oscillations importantes sans vérification du logiciel	OUI avec vérification interne venant de xPC Target). Mesures à l'oscilloscope à l'appui.
Rapidité d'exécution	Période d'échantillonnage minimale: 10 ms selon les résultats	Le banc d'essai *xpcbench* indique une période minimale de 449 µs .
Fiabilité générale	Grande sensibilité aux perturbations de l'environnement d'exploitation.	Les perturbations demeurent sans effet.
ENVIRONNEMENT D'OPÉRATION		
Type	Graphique	Combiné (selon la conception.
Possibilités	Limitées au panneau créé lors de la conception.	Multiples outils disponibles (éléments graphiques et textuels).
Temps d'apprentissage	Très rapide puisqu'il s'agit d'une interface graphique.	Si l'interface est essentiellement graphique, très rapide.
Simplicité de création de l'interface d'opération	Simple puisque la création est directe	Relativement complexe à partir de l'outil *guide* de Matlab.
Possibilités offertes	Téléopération (non explorée); consultation de données en cours d'exécution (oui); modifications des paramètres en cours d'exécution (oui); consultation des états de fonctionnement de l'environnement d'exploitation (non)	Téléopération (non explorée); consultation de données en cours d'exécution (oui); modifications des paramètres en cours d'exécution (oui); consultation des états de fonctionnement de l'environnement d'exploitation (oui)

Chapitre 4 : Étude expérimentale: robot mobile

L'objectif de la deuxième étude de cas est d'approfondir les possibilités offertes par xPC Target. Un résultat collatéral important de cette poursuite est la mise en place d'une plate-forme mobile pour tester différents algorithmes de guidage de véhicule.

Des études d'algorithmes de guidage de véhicules occupent plusieurs équipes de chercheurs provenant de plusieurs universités. L'exemple le plus récent (2001) à l'École Polytechnique de Montréal [32] utilise comme banc d'essai un robot mobile développé commercialement, le Pioneer 2_DX de la société ActivMedia [33] qui est fourni avec son environnement de programmation et d'utilisation, Saphira. Ce projet a alors permis de fournir une illustration théorique et expérimentale d'algorithmes de guidage, mais l'environnement informatique utilisé dans ce projet (développé avec Saphira) n'a pas été réutilisé par la suite dû à un manque de performances et à une relative complexité d'utilisation (code C++ avec nécessité de compiler avant chaque test).

La conception mécanique et électrique d'un robot mobile a été réalisé dans le cadre de ce projet. Suite aux conclusions du précédent chapitre, l'environnement informatique choisi est xPC Target, puisque LabVIEW ne comble pas les besoins de performances spécifiques à nos besoins. Un rapport technique [34] résume la conception mécanique et logicielle du robot qui ne sera pas reprise en détails dans ce mémoire. Ce même véhicule a fait le sujet d'un article de conférence présenté au Portugal en 2004 [35].

L'implantation réalisée pour le robot mobile est plus complexe que celle du banc d'essai de l'étude de cas précédente, principalement dû au fait que le robot utilise une boucle de contrôle à deux niveaux. De plus, le pilote de la carte d'acquisition des encodeurs optiques a dû être programmé, puisqu'il n'était pas disponible dans la librairie de xPC Target.

Contrairement au cas du banc d'essai qui correspond beaucoup plus à un contexte d'utilisation pédagogique, ce robot doit pouvoir servir dans plusieurs projets de recherche. Sa conception et son utilisation doivent être clairement définies pour les utilisateurs subséquents. Sa structure à deux niveaux permet d'illustrer pleinement le

potentiel offert par xPC Target en termes de communications et d'échange d'informations.

4.1 Implantation du robot mobile avec xPC Target

Comme dans le cas du banc d'asservissement, l'implantation regroupe trois sections: les fichiers scripts de configuration, le diagramme Simulink intégrant les blocs xPC Target et l'interface graphique. Le diagramme Simulink présente cependant une différence significative. Puisqu'une structure de contrôle à deux niveaux est utilisée, il y a deux diagrammes Simulink plutôt qu'un seul, soit un pour chaque niveau de contrôle.

Le diagramme principal assure la boucle fermée pour l'asservissement de vitesse, et il s'exécute sur le robot mobile; le second diagramme effectue le contrôle de haut niveau. Le premier niveau inclut les modules de simulation et d'E/S réelles, le contrôleur et les fonctions de communication. La tâche de haut niveau pour le guidage de véhicule est de fournir les consignes de vitesse à la boucle fermée et elle inclut des fonctions de communications en plus de l'interface graphique et des scripts de configuration et d'utilisation.

4.1.1 Modèle de simulation pour le robot

Afin de pouvoir établir le parallèle entre la simulation et l'expérimentation, il faut choisir un modèle mathématique pour simuler le comportement du robot. Le modèle retenu est présenté dans le rapport présenté plus tôt [34]. La figure 4.1 reprend le schéma Simulink qui effectue son implantation et le tableau 4.1 fournit la valeur des paramètres requis. La section de droite du schéma assure que le modèle de simulation offre les mêmes sorties et les mêmes entrées que le modèle expérimental. Cette correspondance est nécessaire pour comparer rapidement les essais simulés et les essais expérimentaux.

Analytiquement, le modèle est incomplet puisqu'il ne représente aucune non-linéarité associée à un robot mobile comme:

- la friction;

- la réponse non linéaire des moteurs;
- les différences au niveau des paramètres du modèle;
- la décharge des batteries.

Toutefois, il permet d'effectuer tous les tests nécessaires pour fournir une application directe à l'environnement informatique xPC Target. Un modèle plus complet peut être inclus lors de recherches futures, sans modifier pour autant les conclusions présentées dans ce mémoire. C'est l'avantage d'une conception par bloc: chaque bloc est indépendant des autres dans la mesure où ses entrées / sorties sont compatibles avec le reste du module. Les paramètres spécifiques au modèle mathématique (gains statiques et constantes de temps) sont déterminés via des essais en boucle ouverte qui sont documentés dans le rapport de conception [34].

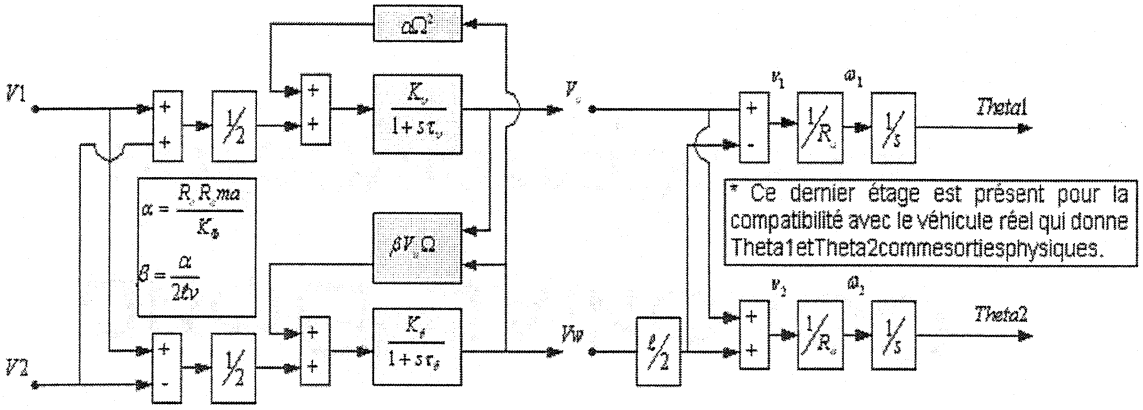


Figure 4.1: Schéma conceptuel du simulateur utilisé pour le robot

Tableau 4.1 : Ensemble des paramètres pour la modélisation du robot mobile

PARAMÈTRES	SYMBOLE	VALEUR
CHARIOT (paramètres mesurés)		
Distance axe des roues – centre de rotation	a	0 m
Masse du véhicule	m	15 kg
Distance entre les roues motrices	l_v	0.334 m
Rayon des roues motrices	R_a	0.07 m
MOTEURS (paramètres théoriques tirés des spécifications)		

PARAMÈTRES	SYMBOLE	VALEUR
Tension de référence des batteries	E	24 V
Constante de couple	K_ϕ	n*0.0459 N.m/A
Constante de force contre électromotrice	K_p	n*0.0459 N.m/A
Résistance interne	R_e	2.49
Vitesse maximale sans charge	w_{max}	44.4 rad/s
Couple maximal absolu	T_{PK}	2.10 N.m
Couple maximal en régime continu	T_C	0.43 N.m
Couple de friction	T_F	0.0483 N.m
Coefficient de frottement visqueux	b	2.6 x 10^{-6} N.m.s/rad
Ratio d'engrenage	n	11.5
Résolution des encodeurs	$cnts$	n*500 impulsions / tour
MODÈLE (paramètres identifiés; voir les essais dans le rapport de conception [])		
Gain statique en vitesse linéaire	K_v	0.098 m/s.V
Constante de temps en vitesse linéaire	τ_v	0.68 s^{-1}
Gain statique en vitesse angulaire	K_θ	0.49 rad/s.V
Constante de temps en vitesse angulaire	τ_θ	0.36 s^{-1}

4.1.2 Modèle d'entrées / sorties réelles pour le robot

Un module permettant les entrées / sorties réelles avec le robot mobile est nécessaire pour les essais expérimentaux. Ce module possède comme sorties deux encodeurs optiques reliés à l'axe de chacune des roues. Ces capteurs donne l'évolution relative de la position angulaire des roues en cours d'exécution. Le module possède comme entrées les commandes allant à chacun des moteurs du robot. Chaque moteur permet le déplacement du robot en fournissant aux roues une force proportionnelle au courant qui circule dans ses bobines.

4.1.2.1 Pilote pour la lecture des encodeurs

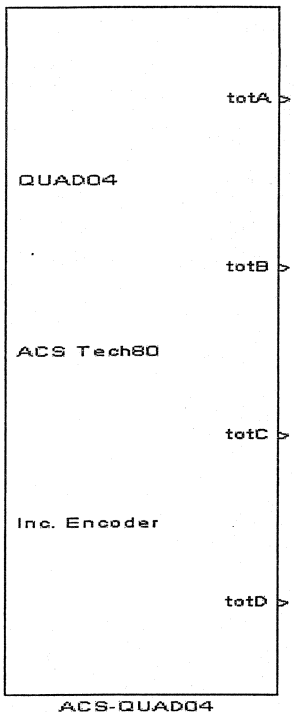


Figure 4.2: Bloc de lecture des encodeurs optiques

Lors du développement du robot mobile, aucune carte d'interface PC-104[10] [6, 7] de lecture d'encodeurs optiques compatible avec la lecture en quadrature[11] n'était supportée par la librairie xPC Target. La carte 5912 de la compagnie ACS Tech80 [36] a alors été retenue principalement pour sa caractéristique d'acquisition simultanée sur ses quatre entrées. Le pilote a été implanté avec l'aide des manuels techniques fournis par le fabriquant [37, 38], d'un exemple fourni par une autre carte, ainsi que des fonctions gabarits fournies par Simulink et xPC Target. Le seul désavantage associé à la programmation maison du pilote est que les fichiers sources doivent toujours se trouver dans le répertoire actuel de travail. Ceci est essentiel pour permettre la compilation dans xPC Target des modèles utilisant le pilote. Ce désavantage peut être résolu en envoyant les fichiers dans le répertoire général de Matlab, mais il ne faut pas oublier alors de transférer les fichiers sources avec les modules Simulink lors du passage vers un autre ordinateur de développement.

Le code source du pilote n'est pas présenté dans ce mémoire car il utilise une librairie appartenant à la société ACS Tech80 ce qui empêche sa diffusion publique. Pour les diagrammes utilisant le pilote, il faut inclure le bloc de la figure 4.2 pour faire le lien entre le code compilé (une librairie dynamique nommée *encacsquad.dll*) et le reste du diagramme. Toutefois, l'insertion du bloc dans le diagramme implique de modifier la commande de compilation utilisée par RTW. La modification se fait dans les paramètres de configuration de RTW (voir la figure 2.1) dans la case *Make command* qui doit contenir la commande suivante: *make_rtw USER_SRCS="encacsquad.c te5912s_1.c".*

10 Nous devons utiliser le protocole PC-104 car il s'agit d'un robot mobile qui utilise un ordinateur embarqué dont le bus de communication fonctionne sur ce protocole.

11 La quadrature est une méthode pour améliorer la précision des encodeurs optiques en utilisant deux signaux dont la phase est décalée de 90°, ce qui implique un matériel d'acquisition compatible.

Pour la programmation du pilote, beaucoup de temps a été passé à comprendre la structure de programme employée par Simulink afin de bien saisir l'interaction entre le code en C++ et le bloc qui l'utilise dans le diagramme Simulink. Le passage des paramètres entrés dans le bloc Simulink vers le code, ainsi que l'affectation des sorties calculées dans le code vers les sorties du bloc, ont posés le plus de problèmes. De plus, la compilation du code pour créer le pilote requiert une attention spéciale pour permettre la compatibilité avec Simulink. Il faut passer par le compilateur de Matlab, *mcc*, via la commande suivante: *mcc -S -v encacsquad.c Te5912s_1.c*, qui donne le *dll* nécessaire au bloc dans Simulink pour fonctionner. Le fichier *Te5912s_l.c* est le code source de la librairie fournie par ACS Tech80 pour permettre l'utilisation des fonctions de communication avec la carte.

4.1.2.2 Pilote pour la commande des moteurs

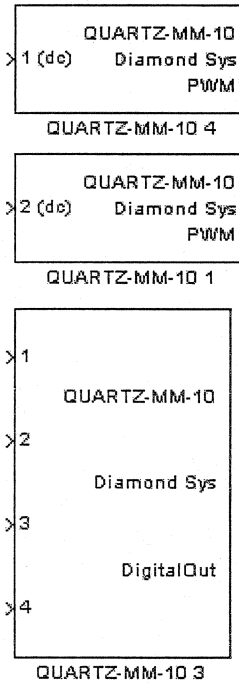


Figure 4.3: Blocs de commande PWM des moteurs

La commande par modulation de largeur d'impulsion, ou *Pulse Width Modulation* (PWM) en anglais [39], est une technique d'envoi de commande à un actionneur dont la constante de réaction est suffisamment grande pour agir comme filtre. Les moteurs à courant continu (CC) sont un exemple d'un actionneur agissant comme filtre.

Cette technique est utilisée sur le robot mobile, car l'amplification en puissance d'un signal PWM demande moins d'énergie, ce qui est important pour les batteries. De plus, cette technique utilise une sortie numérique qui passe par un pont en H pour l'amplification, ce qui constitue un choix moins coûteux qu'une sortie analogique passant par un amplificateur linéaire de puissance.

La carte retenue pour la sortie numérique des signaux PWM est la Quartz MM-10 [40] de la compagnie Diamond Systems. La carte est supportée par la librairie xPC Target avec un contrôleur PWM déjà intégré au pilote. La figure 4.3 montre les trois blocs utilisés pour la commande PWM: 2 blocs pour l'envoi du signal PWM (un pour chaque moteur) et un bloc pour l'envoi des paramètres des signaux PWM (direction de rotation, activation des moteurs). Physiquement, une fréquence de 400 kHz est utilisée pour la porteuse PWM, afin de ne pas faire vibrer inutilement les moteurs qui ont un temps de réponse très court (environ 9 ms pour le temps de réponse mécanique et 0.9 ms pour le temps de réponse électrique; données tirées du site de Pittman [41]).

4.1.3 Contrôleur de premier niveau

Le contrôleur de premier niveau est une boucle fermée en vitesse qui permet l'implantation de nombreux algorithmes de deuxième niveau. Cette implantation peut facilement être modifiée par la suite, par exemple en permettant la commande directe de chacun des moteurs. Normalement, le contrôleur de premier niveau ne change pas d'une application à l'autre puisque la majorité des algorithmes à l'étude pour le guidage d'un robot mobile concernent le calcul des points de consignes à envoyer à ce contrôleur. Ainsi, une fois bien ajusté par des essais en boucle ouverte et par l'identification des paramètres des fonctions de transfert concernant la vitesse, il peut être utilisé pour plusieurs essais.

La figure 4.4 présente le fichier *vehicule_vitesses.mdl*. On y retrouve à gauche le schéma du contrôleur de bas niveau qui inclut le module des E/S réelles. La section de droite contient les blocs de remplacement possible du module des E/S réelles pour effectuer la simulation. Le bloc de simulation du haut possède seulement les sorties pour l'angle de chaque roue (correspondance avec le module réel), alors que le bloc du bas possède des sorties intermédiaires utiles pour des essais en simulation seulement.

Pour le passage de la simulation à l'expérimentation, deux options sont possibles. La première est de faire comme dans l'étude de cas précédente et d'inclure dans un bloc les deux modèles (simulateur et module des E/S réelles) avec la logique pour distribuer

les signaux en fonction du mode d'utilisation. La seconde option est de créer deux blocs différents et de compiler avec *Real Time Workshop* (RTW) deux versions de la boucle de contrôle, soit une version pour chaque mode de fonctionnement. La première option demande plus de calculs et donne un schéma moins clair, ce qui a justifié l'emploi de la deuxième solution lors de l'implantation. Le transfert du diagramme compilé correspondant au mode désiré lors des essais peut se faire automatiquement dans le script de configuration et demande très peu de temps.

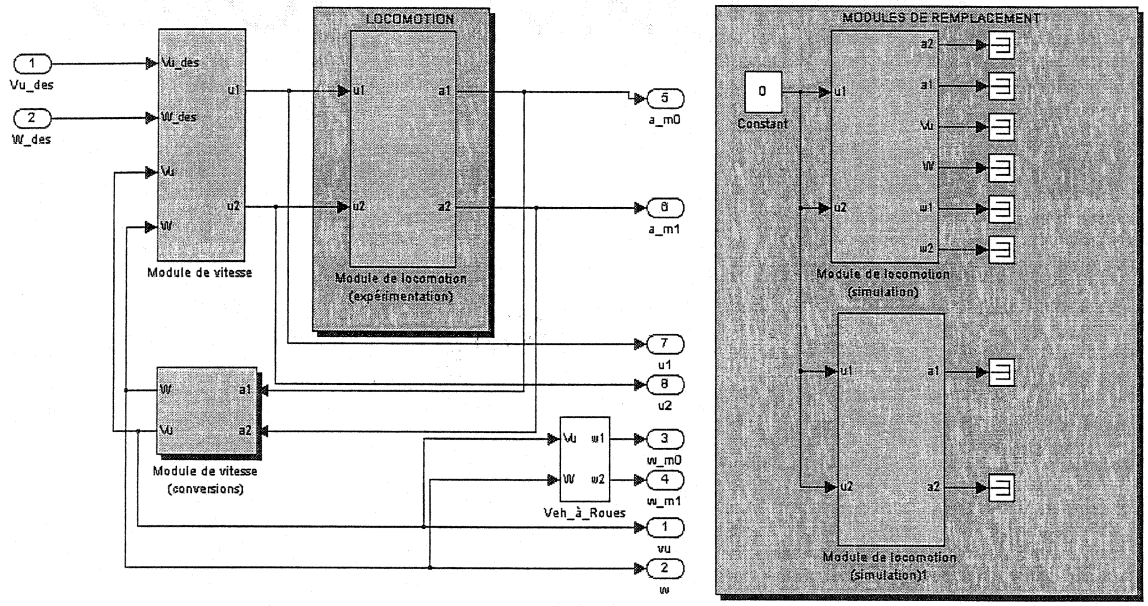


Figure 4.4: Contrôleur de premier niveau dans xPC Target

4.1.4 Contrôleur de deuxième niveau

Ce deuxième niveau de contrôle illustre de nouvelles possibilités de xPC Target. Ce contrôleur effectue l'acquisition des commandes de l'opérateur, calcule les points de consigne et envoie les consignes à la boucle fermée implantée dans le premier niveau de contrôle. La figure 4.5 illustre un exemple de contrôle à ce niveau (fichier *controle_simple.mdl*). On y retrouve à gauche un bloc de lecture pour les commandes (nommé *Joystick*), au centre les boîtes contenant la mention *To xPC Target* et *From xPC Target* et, à droite, des afficheurs (*scope*).

L'algorithme de contrôle implanté dans ce diagramme est simplement le passage direct de la commande provenant du bloc *Joystick* vers la boucle fermée. Les implantations plus complexes peuvent s'inspirer de ce diagramme qui démontre l'acquisition de consignes et l'échange d'informations avec xPC Target.

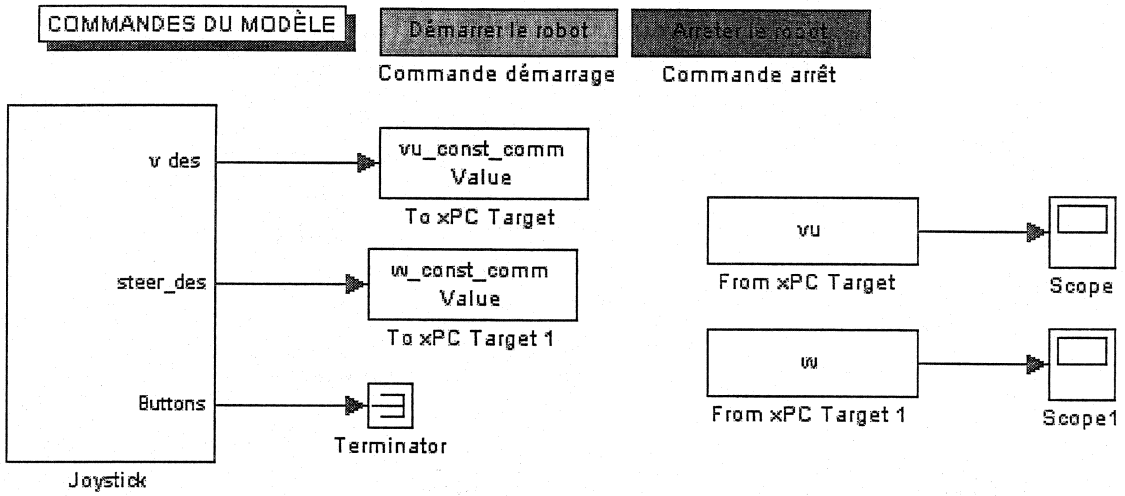


Figure 4.5: Contrôleur de haut niveau dans xPC Target

Le bloc *Joystick* utilise le pilote de la librairie *Virtual Reality* [42] disponible sur le site de *The MathWorks* pour effectuer l'acquisition des signaux de l'ensemble volant / pédales utilisé au laboratoire. Pour Simulink, il suffit que l'interface soit reconnue par Windows comme étant un contrôleur de jeu (*game controller*) pour que l'acquisition fonctionne. Cette nécessité de passer par Windows implique que le contrôleur de deuxième niveau ne peut être transféré dans l'environnement xPC Target. En effet, cet environnement ne supporte pas la libraire de réalité virtuelle qui utilise des fonctions propres au système d'exploitation Windows. Or, puisque Simulink peut communiquer aisément avec l'environnement xPC Target, l'implantation demeure possible dans la mesure où ce contrôleur de deuxième niveau demeure dans Simulink.

Pour les blocs contenant la mention *To xPC Target* et *From xPC Target*, il s'agit de blocs de liaison entre Simulink et le diagramme actuellement en exécution dans l'environnement d'exploitation de xPC Target. Ces blocs n'ont qu'à connaître le nom de

l'application et celui du paramètre à modifier (blocs *To*) ou à lire (blocs *From*) pour permettre la communication. Puisque xPC Target est un environnement d'exploitation en temps réel et que Simulink fonctionne dans Windows, il est certain que la communication des paramètres n'est pas en temps réel, d'où l'importance de placer la totalité de la boucle fermée dans le contrôleur de premier niveau. Si un réseau local est utilisé pour la communication entre Simulink et xPC Target, les délais de transmission demeurent suffisamment petits pour permettre un contrôle efficace de la part de l'opérateur.

La simplicité du contrôleur de la figure 4.5 est flagrante. Il est certain que les algorithmes de guidage testés avec cette interface peuvent être plus complexes. Dans cette étude de cas, le maximum d'effort est mis sur l'étude de l'environnement informatique, pas sur l'élaboration d'un contrôleur en particulier. Cette considération n'influence pas les résultats de l'analyse de l'environnement informatique appliqué au robot mobile étudié puisque les interfaces demeurent les mêmes que dans un cas complexe. Des stagiaires ont d'ailleurs démontré son potentiel en effectuant l'étude du guidage en marche arrière du robot mobile joint à une remorque en un temps très limité. Les résultats de cette étude se trouvent dans un rapport technique [43] disponible à l'École Polytechnique de Montréal.

4.1.5 Communication entre les niveaux de contrôle

Pour permettre la communication entre les deux niveaux de contrôle, un lien d'échange de données est nécessaire. Puisque le système étudié est un robot mobile, un lien physique direct est encombrant et justifie l'emploi d'une technologie sans fil. Or, l'expérience démontre que les blocs xPC Target pour l'échange de données entre Simulink et la cible utilisent le même lien que celui utilisé lors du transfert de modèles compilés vers la cible. Sachant que le lien réseau TCP/IP a été retenu depuis le début pour ce transfert, le lien sans fil utilisé doit supporter ce protocole de communication.

Pour l'implantation, le choix s'est arrêté sur le module de réseau sans fil ME-102 de la société Netgear [44]. Ce module s'adapte à n'importe quelle carte réseau existante

et utilise le protocole 802.11b pour le passage des paquets TCP/IP sur des canaux sans fil à un débit maximal de 11 Mbps. Deux modules seront utilisés, un premier sur le robot mobile et un deuxième relié au réseau local. Chaque module est configuré pour constituer un pont (*bridge*) qui remplace directement la connexion physique habituellement réalisée à l'aide d'un fil réseau. L'utilisation de ce module est tout à fait transparente pour la carte réseau du robot mobile.

4.1.6 Interface graphique

Bien que simple, la commande du robot mobile via xPC Target demande la connaissance de plusieurs commandes pour le transfert des fichiers et pour l'initialisation du modèle. De plus, la nécessité de devoir toujours avoir dans le répertoire actuel de travail les fichiers de programmation du pilote de la carte de lecture des encodeurs demande une attention supplémentaire. Pour simplifier l'utilisation, une interface graphique élémentaire a été implantée lors de la conception et est présentée à la figure 4.6. Cette interface regroupe les commandes d'initialisation, de vérification et de transfert des modèles de base. Elle peut servir d'exemple de base pour en développer de plus complexes qui regrouperont la récupération, l'analyse et le traçage des données. Cette option n'est pas incluse sur l'interface de base, car les données utiles dépendent grandement de l'algorithme de commande à l'étude (les deux niveaux). L'expérience démontre que l'utilisation de scripts Matlab est plus efficace pour la gestion des données dans un contexte dynamique de manipulations.

4.1.7 Scripts de configuration et d'exécution

L'implantation complète du robot mobile contient plusieurs paramètres dont l'initialisation et la modification sont réalisées par différents scripts Matlab. Il y a par exemple le script d'initialisation pour la création et l'affectation des paramètres de fonctionnement, *INIT_VEHICULE.M*. Un deuxième script, *RUN_INTERACTIVE.M*, permet l'exécution automatique des essais en regroupant en séquence:

- la saisie des paramètres de l'essai;
- l'envoi du contrôleur de premier niveau vers la cible xPC Target;
- le démarrage des deux niveaux de contrôle;
- la récupération des données de la cible;
- l'affichage des résultats et leur sauvegarde.

Pour donner une idée précise des commandes impliquées, ce fichier regroupe 82 lignes dans sa version actuelle.

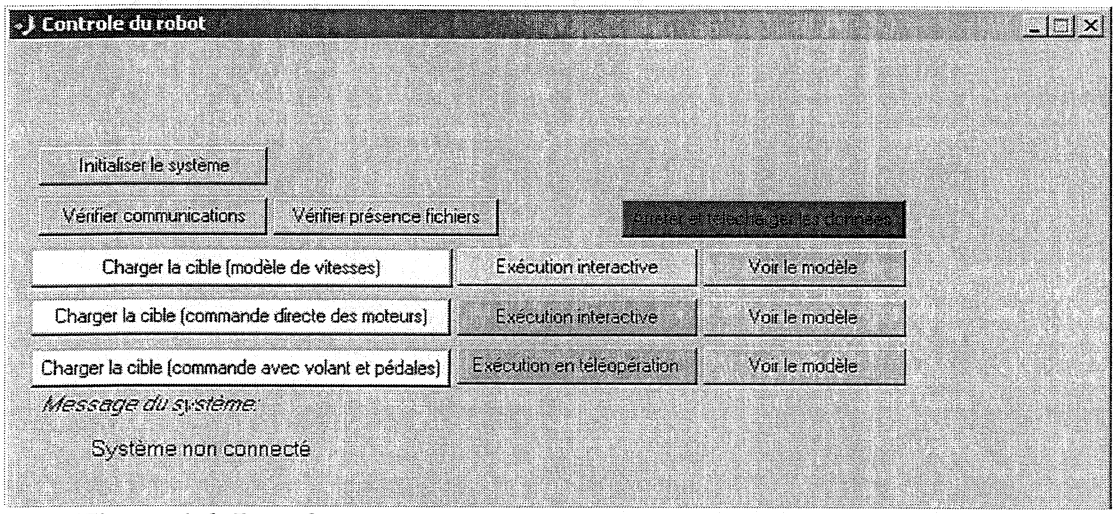


Figure 4.6: Interface graphique d'opération dans Matlab (xPC Target)

Il est intéressant de noter qu'il est facile de modifier ces scripts pour accommoder tout type de contrôleur sans devoir reprogrammer les fichiers principaux. Si un nouveau bloc est ajouté dans la structure de la boucle fermée, alors il faut simplement recompiler le modèle avec le nouveau bloc, en s'assurant que tous ses paramètres de fonctionnement ont la valeur désirée dans l'espace de travail de Matlab.

4.2 Évaluation de l'implantation dans xPC Target

L'évaluation initiale de l'environnement xPC Target a été effectuée dans le cadre de l'implantation du banc d'essais en asservissement de position au chapitre 3. Or,

quelques points d'analyse peuvent être repris à ce sujet en considérant le double niveau de contrôle utilisé dans l'implantation actuelle du robot mobile. La communication continuelle entre l'ordinateur hôte et l'ordinateur cible pour l'envoi des commandes et la lecture des données modifie les exigences en termes de communication.

L'analyse de chaque section de l'environnement informatique n'est pas reprise dans le détail dans ce chapitre puisque l'environnement de conception est le même et que les conclusions du chapitre 3 concernant xPC Target demeurent valides.

4.2.1 Compatibilité des éléments de la boucle

L'implantation du robot mobile a fourni deux défis: la boucle à deux niveaux et la programmation d'un pilote compatible avec xPC Target pour la carte de lecture des encodeurs optiques. Ces difficultés donnent une meilleure idée des possibilités offertes en termes de compatibilité des éléments de la boucle.

Pour réaliser la boucle à deux niveaux, xPC Target offre des blocs de communication entre l'hôte et la cible. Ces blocs utilisent une période d'échantillonnage séparée qui est déterminée lors de la conception du modèle de deuxième niveau. Une période de 10 millisecondes a été utilisée dans l'exemple fourni pour ne pas surcharger inutilement le contrôleur de premier niveau. Ce choix est justifié par le fait que l'opérateur qui pilote l'ensemble volant / pédales n'a pas un temps de réponse plus rapide que 10 ms dans tous les cas. Le contrôleur de deuxième niveau s'exécute en parallèle dans Simulink sur l'ordinateur hôte pendant que la boucle fermée s'exécute elle sur le robot mobile dans l'environnement temps réel de xPC Target pour des performances optimales.

Pour la programmation du pilote, quelques problèmes de compatibilité sont survenus. En effet, la compilation du *DLL* nécessaire au fonctionnement du bloc a nécessité l'obtention du code source de la librairie des fonctions de la carte de lecture des encodeurs. Le désavantage est que ce code appartient à la société ACS Tech80, ce qui limite la diffusion du pilote réalisé lors cette implantation.

4.2.2 Protocole d'échange de données

Pour cette implantation, la cible est un ordinateur embarqué qui doit communiquer avec des cartes d'E/S qui utilisent le protocole PC/104. Or, puisque le pilote de la carte Quartz-MM10 est disponible dans la librairie xPC Target, il est clair que l'environnement supporte ce protocole. Pour la carte de lecture des encodeurs, le pilote programmé lors de l'implantation assure lui aussi la compatibilité avec le protocole PC/104.

Au niveau de l'échange entre les deux contrôleurs (premier et deuxième niveau), le lien sans fil utilise le protocole 802.11b pour le transfert TCP/IP des données. L'avantage de ce protocole est qu'il est complètement transparent pour les deux ordinateurs qui l'utilisent, ce qui ne pose aucun problème de compatibilité.

4.2.3 Environnement d'exploitation pour l'expérimentation

Bien que l'environnement d'exploitation soit le même que dans le chapitre précédent, la cible est entièrement différente. L'ordinateur embarqué, les cartes d'E/S PC/104 (dont une avec un pilote programmé maison) et le lien réseau sans fil n'ont pas fait partie de l'analyse lors de la première implantation. À prime abord, l'effet de ces nouveaux éléments sur les performances temporelles de l'environnement d'exploitation demeure inexploré. Les mesures de stabilité temporelle et de rapidité temporelle sont donc reprises sur cette nouvelle cible.

Stabilité temporelle

Puisque le PC embarqué est nettement plus puissant que celui de l'implantation précédente (processeur à 566 Mhz), la stabilité est testée avec des périodes d'échantillonnages plus petites, soient 200 µs et 1 ms. D'après les tests menés avec l'oscilloscope dans le chapitre précédent, les mesures fournies par xPC Target sont fiables. Ces tests de fiabilité avec l'oscilloscope ne seront pas repris, car ils nécessitent la modification des connexions électriques à l'intérieur du robot mobile pour saisir le signal de l'oscillateur.

Le premier test de stabilité qui est effectué concerne le temps d'exécution de la tâche (TET). Le modèle utilisé lors des tests est *VEHICULE_VITESSES.DLM* qui est la version compilée du fichier *VEHICULE_VITESSES.MDL* présenté plus tôt. Comme le démontre la figure 4.7, le temps de calcul est toujours inférieur à la période d'échantillonnage demandée (Tech=200 µs) et il n'est pas influencé par le mode de fonctionnement (simulation ou expérimentation). Cette conclusion se base sur le fait que la moyenne du TET affichée dans la légende de chaque figure est presque la même pour chaque cas. On peut donc supposer une excellente stabilité temporelle sur le robot mobile. Il est intéressant de noter que la présence du pilote maison non optimisé ne semble pas nuire aux performances du système complet.

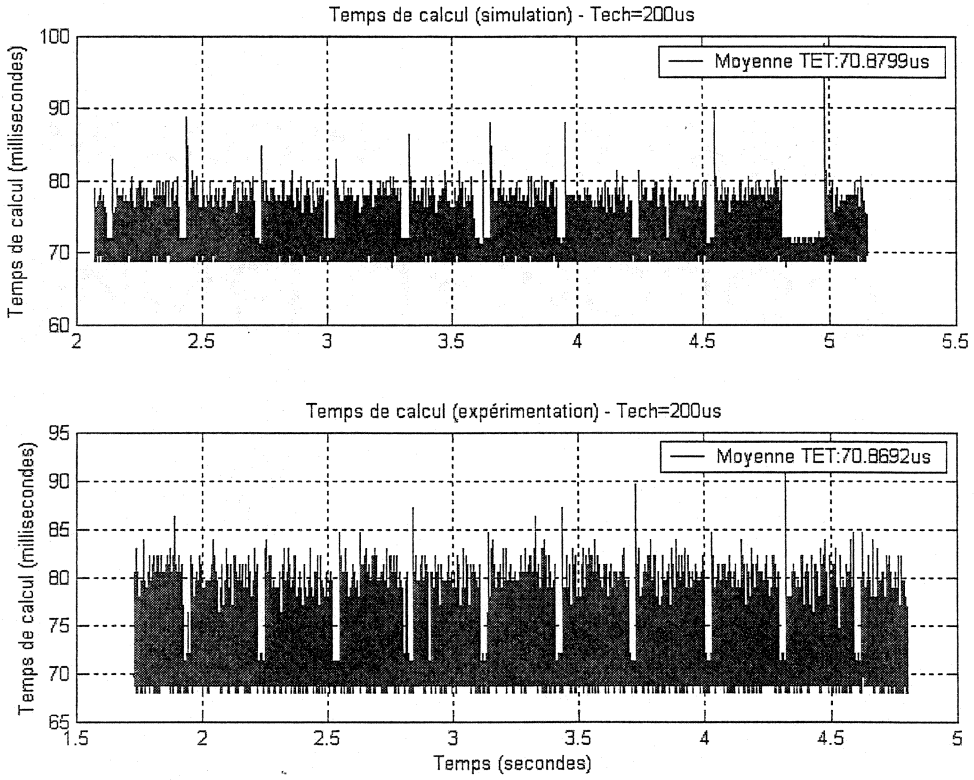


Figure 4.7: Test de la période de calcul (TET) dans xPC Target

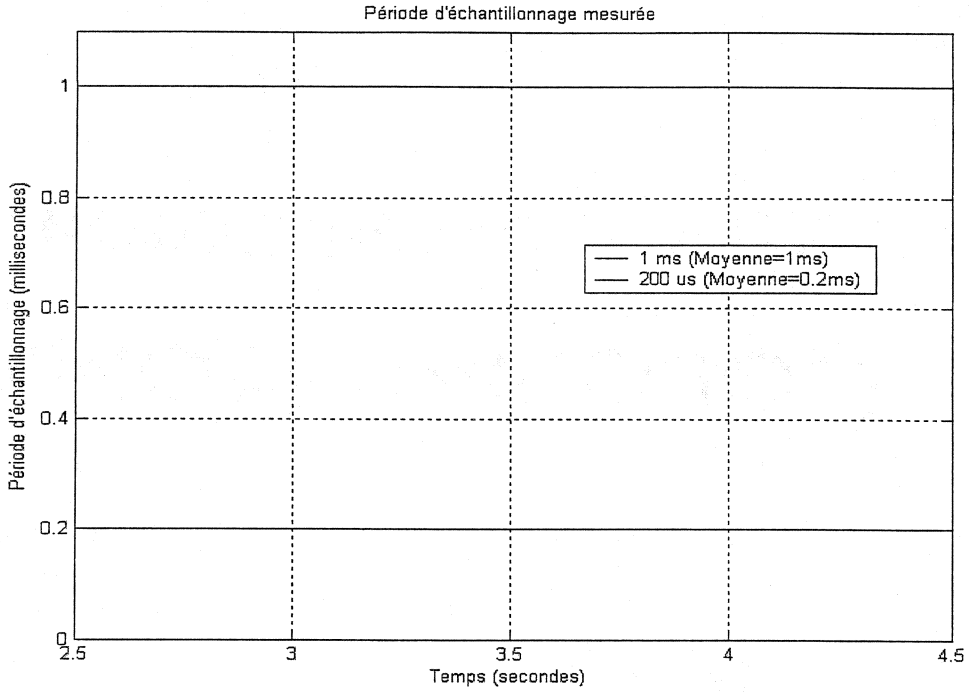


Figure 4.8: Test de la période d'échantillonnage dans xPC Target

Pour bien visualiser la stabilité, la figure 4.8 montre l'évolution de la période d'échantillonnage appliquée durant l'exécution du même modèle avec des périodes demandées de 200 µs et 1 ms. La figure illustre en effet une stabilité temporelle sans failles.

Rapidité d'exécution

Au départ, la commande *xpcbench('this')* est lancée sur le robot mobile pour évaluer sa performance relative avec le modèle générique utilisé par cette fonction. Les résultats (partie textuelle seulement pour une économie d'espace) se trouvent à la figure 4.9. Il est clair que l'ordinateur embarqué est nettement plus performant que l'ordinateur utilisé dans l'étude de cas du chapitre 3 (période d'échantillonnage minimale de 294 µs comparativement à 449 µs pour le cas F14*25). Les programmes atteindront donc de bonnes performances temporelles, ce qui est important pour le guidage de véhicule mobile. Dans les implantations précédentes de véhicules mobiles utilisés dans les

laboratoires, une période d'échantillonnage de dix millisecondes était souvent la plus petite période atteignable. Avec l'implantation actuelle, une période de une milliseconde est facilement atteignable comme le démontrent les résultats de la section précédente.

Minimal achievable sample times in µs

	Minimal	F14	F14*5	F14*10	F14*25
AMD Athlon 1GHz	10	13	24	41	109
Intel P4 1.5GHz	10	15	31	56	133
Intel PIII 600MHz	8	13	39	97	363
This Machine	10	17	47	97	294
Intel PII 400MHz	8	16	58	135	502
AMD K6-2 400MHz	10	24	75	159	755
Intel PI 166MHz	11	44	253	567	1681
Intel PI 90MHz	27	100	590	1400	4600
Intel 486DX 40MHz	43	375	1900	3800	12500

Figure 4.9: Banc d'essai en performance de xPC Target

4.2.4 Environnement d'opération

Bien que l'environnement général d'opération demeure le même que dans l'implantation précédente, il est clair que le contrôleur à deux niveaux vient modifier l'interaction entre l'usager et le système à l'essai. Il y a maintenant deux niveaux de test des algorithmes: le niveau de la boucle fermée qui se trouve sur le robot (cible) et le niveau de détermination des points de consigne qui se trouve sur le poste hôte. Si un algorithme de contrôle de premier niveau est testé, alors l'environnement d'opération est le même que dans l'implantation précédente et chaque changement doit être transféré à la cible. Si le test concerne un algorithme de détermination des points de consigne

(deuxième niveau), les changements sont alors strictement locaux et le modèle chargé sur la cible demeure identique. L'environnement d'opération demeure aussi simple et rapide d'utilisation dans chaque cas.

Autre point intéressant avec une boucle à deux niveaux, la boucle de deuxième niveau en s'exécutant sur l'hôte peut faire l'acquisition des données en cours d'exécution, ce qui permet un meilleur suivi des résultats sans attendre la fin de l'exécution. Cependant, il ne faut pas oublier que puisque Simulink ne fonctionne pas en temps réel, le transfert des données n'est pas synchrone, d'où la nécessité de transférer la totalité de la boucle fermée sur le robot mobile pour ne pas créer d'instabilité. De plus, il faut noter que puisque la boucle fermée s'exécute avec une période d'échantillonnage d'environ 1 ms et le contrôleur de deuxième niveau avec une période de 10 ms (selon la configuration de l'essai en cours), une grande partie des données internes à la boucle fermée ne seront pas transférées de cette manière.

4.3 Utilisation du robot mobile avec l'environnement xPC Target

Dans l'implantation actuelle, l'emphase est mise sur la simplicité d'utilisation et sur la mise en place d'une boucle de contrôle à deux niveaux. Pour la compilation et le transfert d'un contrôleur de premier niveau, ainsi que pour la récupération directe des résultats des essais sur ce type de contrôleur, la méthode générale présentée au chapitre précédent s'applique.

Pour tester un algorithme de deuxième niveau avec l'environnement xPC Target, il faut le coder selon les normes de l'environnement comme pour le contrôleur de premier niveau. Tous les blocs disponibles dans la librairie de Simulink peuvent être utilisés lors de la conception, puisque ce contrôleur ne sera pas compilé avec RTW et s'exécutera sur le poste de conception.

Afin de faciliter l'exécution des essais, l'interface graphique présentée plus tôt (voir figure 4.6) peut être utilisée. Les fonctions de transfert et de vérification qu'elle contient assurent le fonctionnement du robot mobile et de son environnement informatique. Par exemple, l'interface vérifie la présence des fichiers nécessaires pour le

pilote de la carte de lecture des encodeurs optiques, ainsi que la présence des fichiers utiles au fonctionnement du robot mobile, comme le module qui implante la boucle fermée en vitesses (*VEHICULE_VITESSES.MDL*) et le module qui implante la boucle ouverte en tensions (*VEHICULE_TENSIONS.MDL*). Ce deuxième module a été créé pour permettre l'implantation d'algorithmes de deuxième niveau qui nécessitent la commande directe des moteurs plutôt que la commande en vitesses du robot mobile. L'interface regroupe aussi une commande pour l'initialisation et une commande pour la récupération des données brutes.

Pour l'analyse des données, la liberté est laissée entièrement à l'usager puisque chaque application de guidage du véhicule possède ses propres besoins. L'exemple fourni par l'implantation actuelle permet à l'usager de bien comprendre les fonctions d'échange entre Simulink et xPC Target pour concevoir une boucle de contrôle à deux niveaux. Les résultats obtenus lors de cette implantation se trouvent dans le rapport technique concernant la conception du robot mobile [34] et ne sont pas repris ici. Un exemple complet de l'utilisation du robot mobile pour le guidage assisté d'un véhicule en marche arrière est le sujet d'un deuxième rapport publié aux même éditions [43].

4.4 Conclusions sur la deuxième étude de cas

Ce chapitre regroupe quelques nouveaux éléments d'analyse concernant l'environnement xPC Target. Le même moyen que lors de l'étude de cas précédente est utilisé pour regrouper ces éléments (voir tableau 4.2).

Tableau 4.2: Résumé de l'analyse de l'environnement informatique xPC Target lors de la deuxième étude de cas

Élément d'analyse	Choix / Description
COMPATIBILITÉ DES ÉLÉMENTS DE LA BOUCLE	
Tous les éléments de la boucle sont disponibles	NON (vaste librairie de blocs offerts incluant la carte Quartz-MM10 mais pas la carte de lecture des encodeurs optiques)

Élément d'analyse	Choix / Description
Ajout d'élément possible par programmation	OUI (programmation en C++ avec des modèles de base et un compilateur intégré à Matlab)
PROTOCOLE D'ÉCHANGE DE DONNÉES	
Types supportés	Communication avec le protocole TCP/IP avec la cible (avec lien sans fil utilisant le protocole 802.11b). Communication PC/104 entre le processeur de la cible et les cartes d'E/S.
ENVIRONNEMENT DE CONCEPTION	
Implantation combinée possible (simulation et expérimentation)	OUI
Présence des outils d'analyse	OUI
Facilité d'utilisation	Simple autant pour la simulation que pour l'expérimentation.
Possibilité d'interface graphique	OUI (mais il faut l'implanter avec la fonction *guide* de Matlab ce qui demande du temps et une bonne connaissance du langage M)
Type d'environnement	Combiné (schéma pour l'implantation de la boucle et scripts pour la configuration et l'utilisation)
Couverture du cycle de développement	Matlab en général couvre la totalité du diagramme avec son environnement de programmation, Simulink, RTW et xPC Target.
ENVIRONNEMENT D'EXPLOITATION	
Stabilité de la période d'échantillonnage	OUI
Rapidité d'exécution	Le banc d'essais *xpcbench* indique une période minimale de 294 µs pour une application générique complexe exécutée sur la cible.
Fiabilité générale	Les perturbations demeurent sans effet et l'environnement inclut une surveillance active du respect de la période d'échantillonnage demandée.
ENVIRONNEMENT D'OPÉRATION	
Type	Combiné si prévu lors de la conception.
Possibilités	Exécution et analyse des données dans le même environnement.
Temps d'apprentissage	Si l'interface est essentiellement graphique (tout dépendant de la conception), très rapide. Plus le concepteur laisse de libertés à l'usager, plus la connaissance du langage M devient nécessaire et plus le temps d'apprentissage augmente.

Élément d'analyse	Choix / Description
Simplicité de création	Relativement complexe à partir de *guide* pour un système complexe et un bon contrôle de ses paramètres de fonctionnement.
Possibilités offertes	Téléopération (oui); consultation de données en cours d'exécution (oui); modifications des paramètres en cours d'exécution (oui); consultation des états de fonctionnement de l'environnement d'exploitation (oui)

Selon ces résultats, il est clair que xPC Target fournit une excellente architecture informatique pour le prototypage d'algorithmes de commande utilisant deux niveaux de contrôle. Les fonctions de communication entre l'hôte et la cible permettent un transfert rapide des données entre les deux ordinateurs, ce qui permet à un opérateur d'obtenir de bonnes performances lors de l'étude d'un algorithme. Un exemple de ces performances est illustré dans [43].

Cette étude de cas complète l'objectif de maîtrise de l'environnement xPC Target dans le cadre du prototypage rapide d'algorithmes de commande. Le banc d'essais ainsi fourni a déjà permis la complétion d'une étude concrète et il est présentement utilisé pour deux autres projets impliquant le guidage d'un véhicule mobile. Ceci indique que l'objectif secondaire est lui aussi atteint en fournissant un banc d'essais fiable et facile à utiliser.

Chapitre 5 : Recherches actuelles et futures

Lors de la comparaison entre xPC Target et LabVIEW au chapitre 3, les résultats démontrent que xPC Target convient mieux au prototypage rapide d'algorithmes de commande grâce à ses performances, sa simplicité d'utilisation, et grâce au fait qu'il regroupe tous les outils nécessaires pour passer de l'analyse du problème à l'analyse des résultats finaux. Ceci a justifié sa sélection pour l'implantation plus complexe du chapitre 4. Maintenant il est intéressant de montrer comment cet environnement peut combler les besoins d'autres systèmes mécatroniques tout en analysant ses éventuelles limites.

5.1 Autres projets

En fait, alors même que ce projet est en voie de se terminer, déjà plusieurs autres utilisent le travail réalisé, et ce autant au niveau académique qu'au niveau de la recherche. Les projets se basent sur les conclusions d'analyse de l'environnement xPC Target et / ou sur les fichiers réalisés dans le cadre des études de cas présentées dans ce mémoire. On peut citer les quelques projets suivants qui sont en cours ou déjà finalisés dans les laboratoires de la section Automation et Systèmes de l'École Polytechnique de Montréal:

- palan industriel (utilisé dans le cadre du cours ELE4202);
- contrôle de température d'un four électrique (stage de recherche sujet d'un rapport technique publié [45]);
- pendule inversé (en cours de réalisation pour le cours ELE4202);
- guidage en marche arrière d'un véhicule avec remorque (stage de recherche sujet d'un rapport technique publié [43]);
- manoeuvres automatiques de stationnement pour un véhicule (stage de recherche);
- modèle d'hélicoptère (démonstration pour un camp scientifique).

Chacun de ces projets (sauf le dernier) est soit un document pédagogique, soit un projet de recherche (dont deux avec publication), ce qui illustre pleinement le potentiel de l'environnement xPC Target. Pour le dernier exemple, il s'agissait de trouver une architecture de contrôle temps réel suffisamment simple pour être présentée à des groupes d'enfants sans devoir passer par des étapes compliquées de compilation et de transfert vers une cible temps réel. Avec cet environnement, une interface graphique agréable et un excellent contrôle sur le système ont pu être obtenus rapidement pour pouvoir effectuer des démonstrations interactives.

L'environnement xPC Target a un grand potentiel d'enseignement. Il est suffisamment simple et portable pour permettre son emploi dans de petits projets, tout en étant assez performant pour contrôler de grands procédés applicables à l'industrie. L'environnement ne nécessite pas de scripts de compilation (*makefile*) particuliers, et il regroupe peu de commandes à apprendre pour son utilisation générale. Pour l'usager qui connaît déjà Matlab et Simulink, l'apprentissage de RTW et de xPC Target demande peu de temps. Le fait de pouvoir obtenir rapidement un premier résultat grâce à l'automatisation de la compilation et du transfert vers la cible est très motivant pour l'usager débutant.

Les blocs d'E/S de xPC Target peuvent même être considérés simplement comme un ajout à n'importe quel diagramme Simulink d'abord conçu pour la simulation. Puisque Matlab permet de créer directement la disquette de démarrage de la cible xPC Target et que la compilation du diagramme est automatique via RTW, le passage au cas réel de contrôle à partir du simulateur peut être réalisé en moins d'une heure pour un usager qui connaît déjà xPC Target. Évidemment, la configuration des E/S et la calibration du système physique peuvent prendre du temps, mais le passage de l'un à l'autre est initialement très simple.

5.2 Génération de code embarqué

Bien que la maîtrise de l'environnement xPC Target pour le prototypage d'algorithmes de commande soit démontrée, certains projets nécessitent de pouvoir faire

encore un pas de plus dans le cycle de recherche en automatisation (voir figure 5.1 en rappel de la figure 1.3).

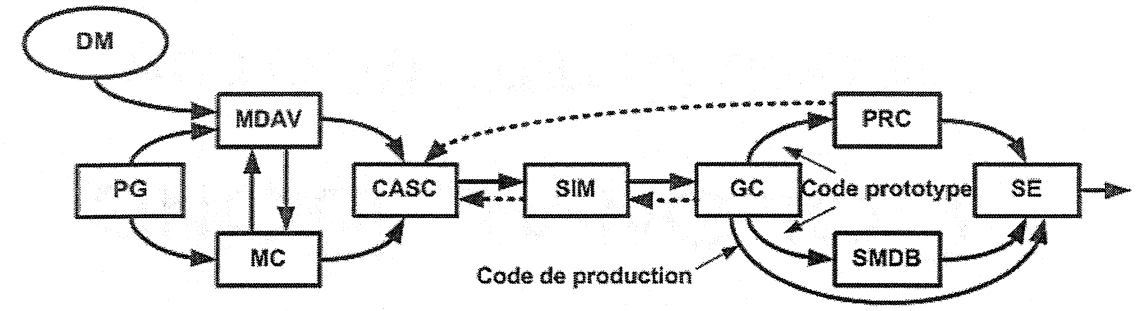


Figure 5.1: Étapes de développement d'un algorithme de contrôle (rappel)

Les implantations des chapitres 3 et 4 vont de la formulation du problème de génie (PG) jusqu'à la fin des tests théoriques et expérimentaux (PRC et SMDB) dans un environnement de test très puissant. Qu'arrive-t-il si le concepteur souhaite produire à grande échelle un microprocesseur utilisant l'algorithme testé dans le but de le commercialiser? En fait, xPC Target fournit un environnement dit embarqué pour l'exécution automatique du modèle implanté. Le code embarqué peut alors fonctionner sur le même matériel que le code de test, c'est-à-dire tout ordinateur compatible PC. Cet environnement, dans sa version embarquée, peut donc servir à la production finale dans la mesure où des frais doivent être payés pour chaque produit vendu qui utilise le noyau temps réel de xPC Target. Pour un cas de vente à grande échelle, il est clair que de cibler un environnement payant comme celui-ci n'est pas très rentable, mais la tâche est réalisable.

5.3 Implantation sous un environnement alternatif

De dire que xPC Target est le seul environnement capable de combler les besoins de prototypage rapide d'algorithmes de commande serait incorrect, et ce autant en termes de performances que de facilité d'utilisation. Plusieurs environnements, mêmes ceux de

conception maison, peuvent offrir un potentiel équivalent pour un projet particulier. Prenons le cas du véhicule mobile présenté au chapitre 4 de ce mémoire. L'environnement xPC Target a été retenu car il comble les besoins du projet. Or un projet de recherche dans le laboratoire de Mécatronique de l'École Polytechnique de Montréal utilise exactement le même type de robot dans le cadre de recherches sur des algorithmes de coopération multirobots.

Bien que la notion de coopération[12] vienne complexifier le projet, une étude préliminaire à ce sujet a été réalisée et l'environnement xPC Target pourrait permettre cette implantation en créant notamment les modules suivants:

- une interface pour la communication avec plus d'une cible xPC Target à partir d'un même poste hôte;
- une machine à états finis pour implanter les algorithmes de coopération sur l'hôte.

Si un outil nécessaire à ces modules n'est pas disponible dans Matlab / Simulink / xPC Target, alors il est toujours possible de le programmer en C++ et de l'intégrer dans le modèle.

L'équipe actuelle de recherche sur ce projet a choisi d'utiliser MICROB [46], une librairie conçue par l'institut de recherche d'Hydro-Québec (IREQ), pour parvenir à ses fins. Or, si cette équipe soumet son architecture aux critères proposés dans ce mémoire, ils arriveront certainement à la conclusion que MICROB, utilisé dans l'environnement d'exploitation QNX, comble pleinement les objectifs de performance nécessaire à la réalisation du projet. Ayant eux-mêmes une grande expérience dans la programmation C++ (requis pour travailler avec MICROB), la simplicité d'utilisation lors de la l'opération n'est pas un problème si l'interface est efficace. Finalement, dans la structure de leur code, le contrôleur est une fonction qui est facilement remplaçable pour remplir les objectifs de prototypage rapide d'algorithme de commande.

Par contre, il est clair que l'implantation spécifique d'un projet sous MICROB, entièrement réalisée en C++, peut difficilement convenir à un autre projet sans une

12 La coopération entre plusieurs robots implique que ces derniers doivent communiquer entre eux, ce qui implique l'utilisation de plusieurs cibles différentes dans xPC Target.

grande refonte du code puisque plusieurs fonctions demeurent dépendantes du système physique impliqué et du matériel utilisé dans ce dernier. Cependant, il demeure qu'un environnement maison programmé à partir d'une librairie comme MICROB permet plusieurs applications allant de la plus simple à la plus complexe et ce en respectant les contraintes mentionnées dans ce travail de recherche. Un document disponible sur le site Internet de MICROB illustre d'ailleurs quelques possibilités d'implantation concrète [47].

5.4 Ajout de sécurité intrinsèque

Tout bon environnement informatique utilisé pour le prototypage expérimental d'algorithmes de commande doit inclure des fonctions de sécurité. Jusqu'à maintenant, l'attention était portée sur les performances des environnements de conception, d'exploitation et d'opération. Dans un cadre d'essais expérimentaux, il est clair que la sécurité des personnes et des biens doit être assurée en tout temps.

L'ajout de la sécurité dans l'environnement xPC Target peut être vu comme l'ajout d'un bloc dans les diagrammes de fonctionnement des algorithmes étudiés. Voici quelques exemples utiles de blocs de sécurité:

1. limite de vitesse d'évolution pour une sortie ou un paramètre interne;
2. limite absolue pour la valeur d'une sortie ou d'un paramètre interne;
3. détection de perte de communication entre l'hôte et la cible.

Le fonctionnement de ces blocs est simple à la base: si la condition de sécurité interne au bloc n'est pas respectée, alors le système doit bloquer tous ses actionneurs. Évidemment, l'action à prendre pour bloquer les actionneurs dépend du système utilisé: pour des moteurs, ceci implique l'envoi d'une commande nulle; pour une vanne d'écoulement, ceci implique l'envoi de la commande de fermeture (qui peut être non nulle).

L'implantation dans xPC Target des deux premiers blocs mentionnés en exemple est simple. Un commutateur sur la commande qui est envoyée suffit à assurer la sécurité. Un exemple de commutation est donné à la figure 5.2. Le bloc de sécurité détecte le dépassement d'une valeur critique pour la sortie *angle* du système et envoie le cas

échéant l'ordre d'arrêt.

L'implantation du bloc associé au dernier exemple est plus complexe. Toutefois, si le fait de perdre la communication implique un possible manque à la sécurité, il est essentiel de l'inclure dans la boucle de contrôle. Le chapitre 4 présente un bon exemple de système où ce module est nécessaire. En effet, si le système perd la communication réseau alors que le contrôleur de deuxième niveau envoi une consigne non nulle, alors la boucle fermée va appliquer indéfiniment cette consigne, ce qui implique que le robot mobile n'arrêtera pas.

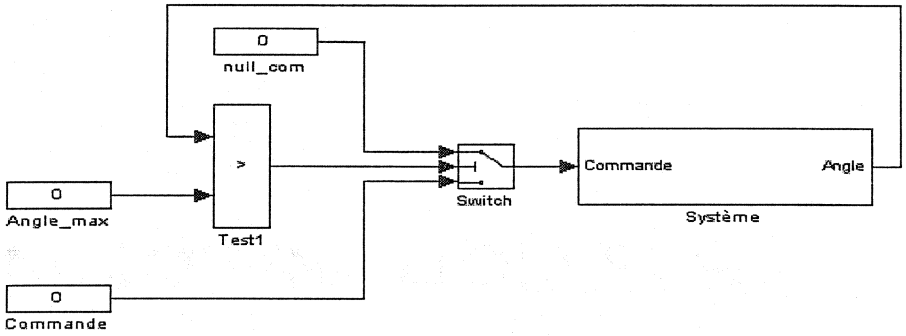


Figure 5.2: Sécurité sur la valeur maximale

Le fonctionnement du bloc de sécurité réseau implique une communication continuelle avec la cible pour évaluer sa capacité à recevoir les commandes. Cependant, il ne faut pas pour autant nuire à la communication nécessaire pour le fonctionnement des essais. Pour cette raison et parce que l'expérience démontre que les blocs *To xPC Target* n'envoient que les changements de valeur du signal qui leur est associé, le signal communiqué à ce module sera une sinusoïde lente dont la période d'échantillonnage sera de 10 ms. Ce signal, intitulé *watchdog_signal*, est envoyé au module de sécurité présent sur la cible. Ce module applique l'algorithme suivant:

$ds = ws(i) - ws(i-1)$
si $ds = 0$
    $s = s + dt$
sinon
    $s = 0$

avec ds la différence de signal entre deux itérations de calcul de la cible, ws le signal *watchdog_signal* reçu par la cible, s le temps passé avec une différence nulle, dt la

période d'échantillonnage et (*i*,*i-1*) les instants présent et passé d'échantillonnage.

La valeur *s* de cet algorithme représente l'intégration du temps passé avec une différence nulle entre deux échantillons du signal de sécurité. En utilisant une limite maximale permise, la perte de communication peut être détectée suivant la valeur de *s*. Évidemment, la valeur limite doit prendre en compte que le signal de sécurité varie seulement à toutes les 10 ms dans le cas présent, d'où l'utilisation de la valeur de 250 ms comme entrée dans l'exemple de la figure 5.3. La figure présente l'exemple en trois sections: la partie qui doit être incluse dans le module Simulink de l'ordinateur hôte, la partie qui doit être incluse dans le modèle Simulink de la cible, et la partie illustrant le détail du module de calcul.

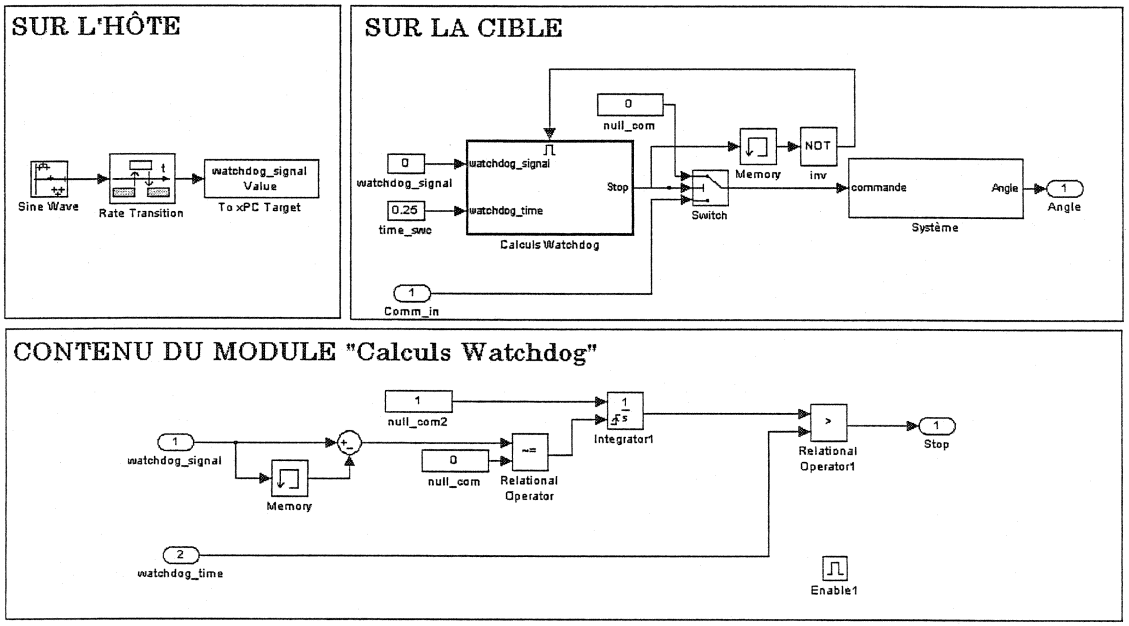


Figure 5.3: Sécurité sur la perte de communication réseau

Ce qui est intéressant au sujet de ce dernier exemple de sécurité est qu'il permet de minimiser l'effet du seul désavantage de xPC Target détecté lors de ce travail de recherche: sa sensibilité extrême à la qualité des communications réseaux. En intégrant ce module, le fait de perdre la communication n'implique plus de perdre complètement le contrôle du système asservi.

Conclusion

Les deux études de cas présentées dans ce mémoire illustrent bien les possibilités offertes par xPC Target pour le prototypage expérimental rapide d'algorithmes de contrôle à partir de modèles Simulink. Ainsi, l'objectif principal de cette recherche est atteint. Les qualités de cet environnement sont nettes avec: un opération aisée, de nombreux outils offerts pour la conception et l'opération, des performances excellentes et un bon rendement au niveau des paramètres généraux comme le coût, le temps d'implantation et l'expertise requise.

Dans un objectif de validation expérimentale du choix effectué, une implantation complète d'une boucle d'asservissement de position pour un ensemble moteur CC / volants d'inertie a été réalisée dans chaque environnement. Ce système est en fait un banc d'essais standard en contrôle de procédés. Tout en démontrant les possibilités offertes par les deux environnements informatiques retenus, l'utilisation du banc d'essais a permis de fournir deux implantations logicielles complètes pouvant servir à la recherche et / ou à l'enseignement.

L'expérience concrète d'implantation dans LabVIEW et xPC Target fournie par cette première étude de cas a permis de départager les deux environnements informatique. Les résultats obtenus montrent à la fois une équivalence et des différences marquées entre les deux environnements. Le résultat final est apparu plus efficace dans le cas de xPC Target, car il fournit un environnement complet allant de la conception du modèle jusqu'à l'analyse finale des résultats, et ce autant au niveau des essais en simulation que des essais en expérimentation. L'environnement LabVIEW ne permet pas une analyse aussi aisée des résultats et le diagramme final de l'implantation est beaucoup plus lourd que celui obtenu dans xPC Target, ce qui complexifie le prototypage rapide des algorithmes de commande.

Au niveau des performances temporelles de l'environnement d'exploitation, il est clair selon les résultats obtenus que xPC Target est à la fois plus rapide et plus stable en terme de période d'échantillonnage applicable à l'exécution de la boucle fermée. La stabilité de cette dernière a même été prouvée à l'aide d'un oscilloscope afin de s'assurer

de l'exactitude des mesures fournies par xPC Target. On pouvait s'attendre à ces performances puisque cet environnement est de type hôte – cîble. Son système d'exploitation permet de convertir n'importe quel ordinateur de bureau en un puissant contrôleur temps réel contrôlé par l'ordinateur hôte via le lien réseau. D'autres cibles temps réel de type hôte - cible, comme Real Time Windows Target de Mathworks ou VxWorks Tornado de WindRiver, auraient pu être utilisées, mais leurs caractéristiques (fonctionnalités, coût) n'équivalaient pas celles de xPC Target selon l'analyse préliminaire.

Suivant l'obtention de si bons résultats pour la première étude de cas, une deuxième implantation plus complexe a été réalisée. Une boucle de commande à deux niveaux pour le contrôle en téléopération d'un robot mobile a été implantée dans l'environnement xPC Target. Rapidement un modèle de simulation et d'expérimentation a été développé pour des tests en boucle ouverte ou fermée sur le robot mobile (avec un contrôle en vitesses ou en tensions). Des étudiants en stage de recherche ont pu utiliser le robot mobile dans le cadre d'une étude complète concernant le guidage assisté en marche arrière d'un véhicule avec remorque. L'obtention des résultats en expérimentation à partir de leurs modules de simulation leur a pris moins de deux semaines, ce qui représente un délai très court pour un projet de ce type.

L'implantation de l'environnement informatique du robot mobile a nécessité la création d'un pilote pour la carte de lecture des encodeurs optiques présents sur le système physique. Cette implantation a donc permis de maîtriser de nouvelles possibilités offertes par l'environnement Matlab / Simulink / xPC Target concernant l'ajout de blocs de conception maison. Après quelques difficultés au niveau de la compilation et de la compatibilité avec le modèle Simulink, le bloc obtenu pour le pilote de la carte a permis une implantation complète et performante.

D'ailleurs, puisque le robot mobile inclut son propre ordinateur embarqué et un module de programmation maison, les mesures de performance concernant l'environnement d'exploitation de xPC Target ont été reprises. L'influence de la présence supplémentaire du module de téléopération dans la boucle a été analysée. Puisque ce

module implique une communication continuelle entre les ordinateurs hôte (le poste de développement) et cible (le robot mobile), on peut s'attendre à une dégradation des performances. Cependant, les résultats ont démontré que l'environnement d'exploitation peut facilement atteindre une période d'échantillonnage de 1 ms, tout en communiquant constamment avec l'hôte, et ce autant lors des essais en simulation que lors des essais en expérimentation. Les résultats confirment donc le maintien des performances de xPC Target dans le cadre de cette nouvelle implantation.

Toutefois, la présence d'un lien réseau sans fil dans cette implantation a introduit un problème imprévu: xPC Target est très sensible à la qualité du lien réseau. Puisque lors d'une perte de communication en mode de téléopération le robot mobile échappe complètement au contrôle, un problème majeur de sécurité est introduit. Pour pallier à ce problème, un bloc de sécurité détectant la perte de communication a été proposé dans ce mémoire et il sera testé prochainement sur le robot. Ce bloc, couramment intitulé *chien de garde*, permet de détecter les éventuelles pertes de communication et interrompt alors la commande des moteurs pour assurer la sécurité.

La simplicité d'utilisation et les performances offertes par xPC Target en font un environnement informatique de choix pour la conception, l'exploitation et l'opération d'une boucle de commande. Dans un projet, le concepteur doit souvent faire le choix entre la simplicité d'utilisation et le potentiel de réalisation offert par un environnement informatique. Or avec xPC Target, les études de cas présentées dans ce mémoire démontrent qu'il est possible de conserver les deux: simplicité d'utilisation et potentiel de réalisation dans un seul et même environnement. Lors de l'implantation, le concepteur profite d'une interface à la fois graphique (Simulink) et textuelle (Matlab) pour partager la tâche de programmation, puis il peut ensuite mettre l'accent sur la création d'un environnement d'opération avec une interface graphique complète pour la simplicité d'utilisation. Si cette interface n'est pas créée, alors il demeure possible d'effectuer toutes les tâches relatives aux essais à partir des commandes Matlab directement.

Bien sûr, puisque Matlab, Simulink, Real Time Workshop et xPC Target sont des outils en constante évolution, l'exploration de leurs possibilités ne peut être considérée

comme terminée suite à cette recherche. Par exemple, il reste encore a explorer les possibilités de xPC Target au niveau de la commande de cibles multiples, l'analyse des délais dans les communications hôte – cible, l'analyse de la gestion de la mémoire de la cible, l'intégration de la communication entre deux cibles xPC Target. On pourrait aussi comparer xPC Target avec Real Time Windows Target, deux produits de la société Mathworks, mais dont le deuxième permet l'exécution sur le même ordinateur dans un environnement indépendant de Windows (hôte et cible sur le même poste de travail). Les possibilités sont encore nombreuses...

Liste des références

(en ordre d'apparition dans le texte)

[1] ITRON, *Survey Results on Real-Time OS Use Trends and ITRON*, [En ligne]. http://www.assoc.tron.org/eng/research/data/survey1999/result-e.html (Dernière consultation de cette page: 5 juillet 2004).

[2] ITRON, *Sans titre, figure agrandie*, [En ligne]. http://www.assoc.tron.org/eng/research/data/survey1999/graph-e/real3.gif (Dernière consultation de cette page: 5 juillet 2004).

[3] MathWorks, *The MathWorks – Products – Matlab – The Language of Technical Computing*, [En ligne]. http://www.mathworks.com/products/matlab/index.html (Dernière consultation de cette page: 5 juillet 2004).

[4] Kuhn, Bernhard, Lineo, *An Overview of Real-time Linux*, [Document PDF en ligne], The Open Group, 74 pp. http://www.opengroup.org/rtforum/info/apr2001/slides/kuhn.pdf (Dernière consultation de cette page: 1 spetembre 2004).

[5] Pyarali, Irfan, Schmidt, D.C., Cytron, Ron K., Techniques for Enhancing Real-Time CORBA Quality of Service, **IEEE Proceedings**, May 2003, 17pp.

[6] PC/104 Consortium, *PC/104 | Technology | PC/104 Info*, [En ligne]. http://www.pc104.org/technology/reg_info.html (Dernière consultation de cette page: 5 juillet 2004).

[7] PC/104 Consortium, *PC/104 Specification, Version 2.5, November 2003*, [Document PDF en ligne]. http://www.pc104.org/technology/PDF/PC104%20Spec%20v2_5.pdf (Dernière consultation de cette page: 5 juillet 2004).

[8] Mathworks, *The MathWorks – Products – Simulink – Simulation and model-based design*, [En ligne].
http://www.mathworks.com/products/simulink/index.html
(Dernière consultation de cette page: 5 juillet 2004).

[9] Mathworks, *The MathWorks – Real Time Workshop – Generate optimized, portable, and customizable code from Simulink models*, [En ligne].
http://www.mathworks.com/products/rtw/description1.shtml
(Dernière consultation de cette page: 5 juillet 2004).

[10] Mathworks, *The MathWorks - Real-Time Windows Target - Run Simulink models on a PC in real time*, [En ligne].
http://www.mathworks.com/products/rtwt/
(Dernière consultation de cette page: 1 septembre 2004)

[11] WindRiver, *Wind River - VxWorks 5.x,* [En ligne].
http://www.windriver.com/products/device_technologies/os/vxworks5/
(Dernière consultation de cette page: 1 septembre 2004)

[12] MathWorks, *The MathWorks – xPC Target – Perform real-time rapid prototyping and hardware-in-the-loop simulation using PC harware*, [En ligne].
http://www.mathworks.com/products/xpctarget/index.html
(Dernière consultation de cette page: 5 juillet 2004).

[13] Opal-RT, *RT-LAb Solo*, [En ligne].
http://www.opalrt.com/catalog/products/1_rt-lab/2_solo/index.html
(Dernière consultation de cette page: 13 septembre 2004).

[14] MathWorks, *The MathWorks – Products – xPC TargetBox – Industrial PC for real-time rapid prototyping*, [En ligne].
http://www.mathworks.com/products/xpctargetbox/
(Dernière consultation de cette page: 5 juillet 2004).

[15] National Instruments, *Acquire, Analyze, and Present Data with LabVIEW – Products and Services – National Instruments*, [En ligne].
http://www.ni.com/aap/
(Dernière consultation de cette page: 5 juillet 2004).

[16] National Instruments, *LabVIEW Real-Time from National Instruments – Products and Services – National Instruments*, [En ligne].
http://volt.ni.com/niwc/common.jsp?page=labviewrt_lvrt_intro
(Dernière consultation de cette page: 5 juillet 2004).

[17] Hays, J., *Advanced Real Time Programming Techniques*, [Document PPT en ligne], National Instruments, 25 pp.
ftp://ftp.ni.com/pub/devzone/at2d.ppt
(Dernière consultation de cette page: 5 juillet 2004).

[18] Kodosky, J., *Is LabVIEW a general purpose programming language?*, The VIEW, National Instruments, [Document HTML en ligne].
http://www.ni.com/devzone/lvzone/view_archived1.htm
(Dernière consultation de cette page: 5 juillet 2004).

[19] National Instruments, *LabVIEW: Ease of Use AND Powerful Development – Tutorial – NI Developer Zone – National Instruments*, [Document HTML en ligne].
http://zone.ni.com/devzone/conceptd.nsf/webmain/F34045D2CC5357F486256D3400648C0F?opendocument
(Dernière consultation de cette page: 5 juillet 2004).

[20] Anonyme, *The Linux Home Page at Linux Online*, [En ligne].
http://www.linux.org/
(Dernière consultation de cette page: 6 juillet 2004).

[21] SGI, *SGI – OpenGL:Home Page*, [En ligne].
http://www.sgi.com/software/opengl/
(Dernière consultation de cette page: 6 juillet 2004).

[22] WxWidgets, *WxWidgets Home*, [En ligne].
http://www.wxwindows.org/
(Dernière consultation de cette page: 6 juillet 2004).

[23] DeSantis, R.M, Hurteau, R., Authié, G., A Real-Time Computer Control Demonstrator for Use in a University Control Laboratory, **IEEE Transactions on education**, vol e-25, no1, 1982, pp 18-28.

[24] Hurteau, R., DeSantis, R.M., **Cours ELE3201 – Guide des laboratoires – Hiver 2004**, Rapport interne, Département de génie électrique, École Polytechnique de Montréal, 2004, 66 pp.

[25] DeSantis, R.M, A Novel PID Configuration for Speed and Position Control, **Journal of Dynamic Systems, Measurement, and Control**, Transactions of the ASME, vol 116, 1994, pp 542-549.

[26] Demers-Roy, C., Analyse expérimentale et simulée d'un banc d'essai en asservissement de position avec un contrôleur PID-DL, Département de génie électrique, École Polytechnique de Montréal, Montréal, 2004, [En préparation].

[27] Measurement Computing, *2&4 Channel Quadrature Encoder Input Boards – User's Manual – Revision 2*, 20 pp., [Document PDF en ligne].
http://www.measurementcomputing.com/PDFManuals/CIO-QUAD0x.pdf
(Dernière consultation de cette page: 7 juillet 2004).

[28] Finley, T., *Two's Complement*, [Document HTML en ligne].
http://www.duke.edu/~twf/cps104/twoscomp.html
(Dernière consultation de cette page: 7 juillet 2004).

[29] National Instruments, *Benchmark Execution Speed of LabVIEW Applications – Tutorial – Development Library – National Instruments*, [Document HTML en ligne].
http://zone.ni.com/devzone/conceptd.nsf/webmain/DC9B6DD177D91D6286256C9400733D7F?opendocument&node=DZ52061_US
(Dernière consultation de cette page: 5 juillet 2004).

[30] Tektronik, Programmer Manual TDS200-,TDS1000-, and TDS2000-Series Digital Oscilloscope (071-1075-01), Tektronix, Beaverton, Orlando, 2004.

[31] Tektronik, User Manual TDS1000- and TDS2000-Series Digital Storage Oscilloscope (071-1064-00), Tektronix, Beaverton, Orlando, 2004.

[32] Lesot, B., Stabilisation d'un tracteur-remorque: étude expérimentale de différents contrôleurs, Département de génie électrique, École Polytechnique de Montréal EPM/RT-01-03, Montréal, 2001, 26 pp.

[33] ActivMedia, *Robots & Robotics Accessories*, [En ligne].
http://www.activrobots.com/
(Page consultée en dernier le 13 juillet 2004).

[34] Beaudry, J., Demers-Roy, C., Un véhicule robotisé pour le prototypage rapide de stratégies de guidage intelligent, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-14, Montréal, 2003, 27 pp.

[35] Demers-Roy, C., Mirales, F., Olivier, J.-F., De Santis, R.M., Hurteau, R., Assisted Guidance For a Tractor-Trailer With Off Axle Hitching, **5th IFAC Symposium on Intelligent Autonomous Vehicles**, Lisbon, july 2004, 5 pp.

[36] ACS Tech80, *Model 5912 PC104 PC104 Encoder Interface*, [En ligne].
http://www.acs-tech80.com/products/5912/
(Dernière consultation de cette page: 8 juillet 2004).

[37] ACS Tech80, Model 5912 Quadrature Encoder Technical Reference, ACS Tech80, Minneapolis, Minnessota, 1994.

[38] ACS Tech80, Model 5912 Software Guide, ACS Tech80, Minneapolis, Minnessota, 1994.

[39] Anonyme, *commande moteur pwm*, [En ligne].
http://d.nardi.free.fr/moteurPWM.htm
(Dernière consultation de cette page: 5 juillet 2004).

[40] Diamond Systems, *Quartz-MM: Advanced Counter / Timer and Digital I/O PC/104 Module*, [En ligne].
http://www.diamondsystems.com/products/quartzmm
(Dernière consultation de cette page: 8 juillet 2004).

[41] PennEngineering Motion Technologies, *Welcome to PennEngineering Motion Technologies*, [En ligne].
http://www.pennmotion.com/quick_index.html
(Dernière consultation de cette page: 14 juillet 2004).

[42] MathWorks, *The MathWorks – Products – VirtualReality Toolbox – Animate and visualize Simulink systems in three dimensions*, [En ligne].
http://www.mathworks.com/products/virtualreality/
(Dernière consultation de cette page: 5 juillet 2004).

[43] Demers-Roy, C., Miralles, F., Olivier, J.-F., Guidage assisté par ordinateur d'un tracteur remorque, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-08, Montréal, 2003, 27 pp.

[44] Netgear, *802.11b Wireless Acces Point ME 102*, [Document PDF en ligne].
http://www.netgear.com/pdf_docs/me102.pdf
(Dernière consultation de cette page: 9 juillet 2004).

[45] Blanc, L., Rainbault, V., Contrôle en température d'un four cylindrique à trois zones de chauffe, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-08, Montréal, 2003, 38 pp.

[46] IREQ, *Microb – Modules intégrés pour le contrôle de robots – Manuel de l'usager*, 121 pp., [Document PDF en ligne].
http://www.robotique.ireq.ca/microb/fr/Manuel_usager.pdf
(Dernière consultation de cette page: 5 juillet 2004).

[47] IREQ, *Microb – Modules intégrés pour le contrôle de robots – Manuel des applications*, 57 pp., [Document PDF en ligne].
http://www.robotique.ireq.ca/microb/fr/Manuel_applications.pdf
(Dernière consultation de cette page: 5 juillet 2004).

ANNEXE I: Schémas d'implantation LabVIEW du banc d'essais en asservissement de position

Afin de fournir une meilleure illustration du travail réalisé, cette annexe présente les différents blocs programmés pour l'implantation du banc d'essais en asservissement de position. Ces figures sont une reproduction directe des diagrammes dans LabVIEW. Les sections panneaux de ces blocs ne contiennent aucun élément particulier et ne sont pas montrées. LabVIEW n'utilise aucun script, seulement le panneau spécial des paramètres globaux qui est présenté aussi dans l'annexe. De plus quelques résultats d'exécution démontrent le fonctionnement de la boucle implantée et sont présentés à la fin.

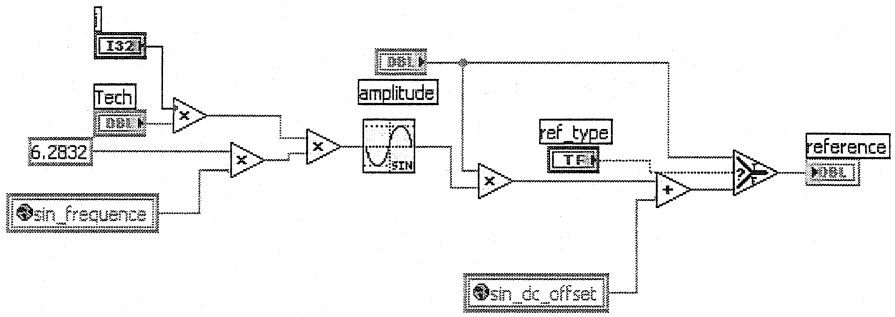
Sections du diagramme LabVIEW

La figure 3.3 au chapitre 3 illustre un aperçu de l'implantation de la boucle de commande, mais la conception de certains blocs représentant les sous-systèmes de la boucle doit être détaillée. En effet, le grand nombre de signaux et de fonctions oblige un regroupement des blocs dans des sous-systèmes afin d'alléger le diagramme. Six blocs nécessitent une présentation plus détaillée: *ref*, *sys_sr_el*, *piddl+k4+zm*, *filtre cache*, *osc* et *create tab PID*. Sans ces blocs, le test des algorithmes de commande ne peut être réalisé tel que prévu par les objectifs de recherche. La division en sous-systèmes permet d'ailleurs le remplacement de blocs, ce qui est essentiel pour le changement d'algorithmes de commande lors des essais.

Bloc REF

Le bloc *REF* contient les fonctions propres à la génération du signal de référence de position. Selon le test à effectuer, la sortie de ce bloc est soit un échelon, soit une sinusoïde. Un simple sélecteur à deux voies suffit à faire la transition entre les deux types de référence. La création de ce bloc spécifique permet une implantation plus complexe dans le futur. Les paramètres de fonctionnement de ce bloc se retrouvent sur le panneau principal de contrôle et sont envoyés lors de l'exécution dans le fichier des

paramètres globaux. Les blocs dont l'identificateur contient un symbole de globe terrestre (*sin_frequence* et *sin_dc_offset* dans la figure AI.1) permettent la lecture des valeurs correspondantes dans le fichier des paramètres globaux.



FigureAI 1: Diagramme pour le bloc ref de l'implantation LabVIEW

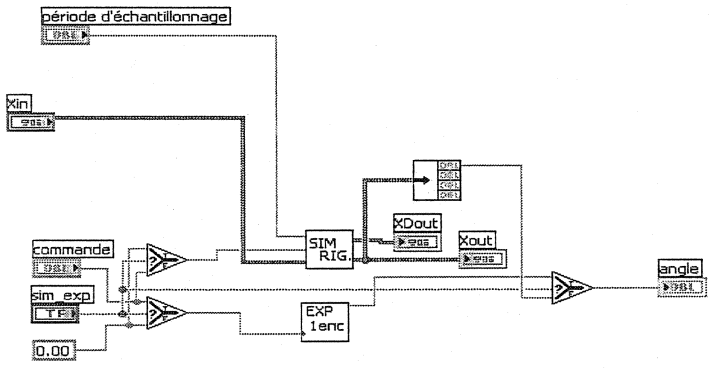
Bloc SYS SR_E1

Dans cette recherche, un des objectifs principaux lors de l'exploration des environnements informatiques est de permettre le passage rapide entre la simulation et l'expérimentation. L'implantation doit donc inclure un bloc qui permet aisément ce passage. Pour ce faire, il doit inclure les pilotes pour commander les cartes d'entrées / sorties, mais aussi un algorithme de simulation du comportement du système utilisé. Ce bloc inclut deux systèmes avec la logique nécessaire pour gérer l'activation de l'un ou l'autre selon le cas d'exécution demandé sur le panneau de contrôle principal. La logique de commutation est représentée à la figure AI.2 qui montre le diagramme général du bloc système.

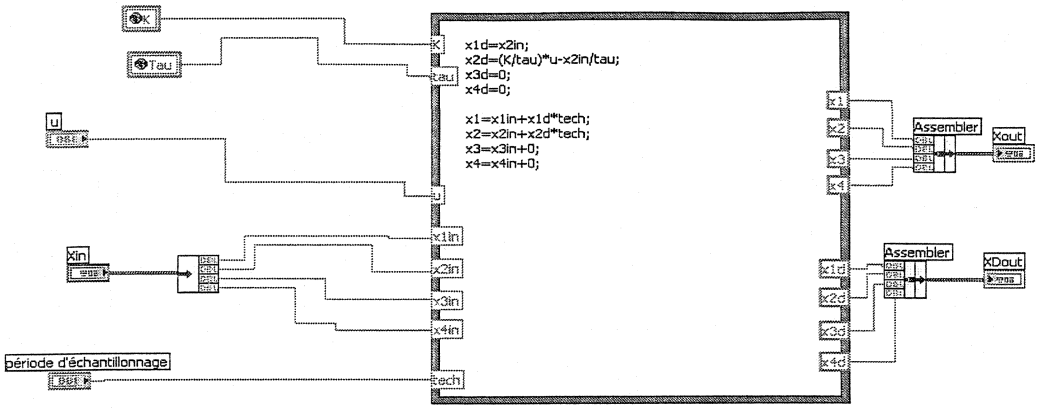
Le bloc *SIM RIG* de la figure AI.2 contient le simulateur qui est implanté par un algorithme textuel qui effectue le calcul de la fonction de transfert du système. Pour l'exemple utilisé ici, la fonction de transfert est celle entre la tension et la position pour un système avec lien rigide. La figure AI.3 donne le contenu du bloc pour cet exemple. Pour changer de type de simulateur (passer au modèle d'états par exemple), il faut changer l'algorithme textuel et s'assurer que le bloc ait les entrées / sorties nécessaires pour permettre les calculs. Quant aux paramètres nécessaires au fonctionnement des

simulateurs, il se trouvent exclusivement dans le fichier des paramètres globaux et ne sont pas inclus sur l'interface de la boucle de contrôle. Ces paramètres changent peu et leur inclusion sur le panneau principal ajouterait trop de connexions dans le diagramme principal. Les blocs de la figure AI.3 dont l'identificateur contient l'illustration d'un globe terrestre (K et Tau) permettent la lecture des paramètres dans le fichier des paramètres globaux.

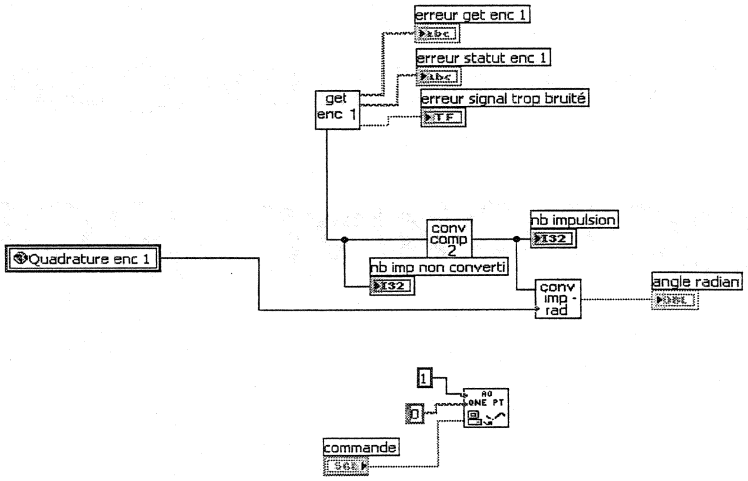
Le bloc *EXP 1 ENC* de la figure AI.2 contient le pilote pour la lecture de l'encodeur optique, ainsi que le pilote pour la sortie de la commande. Lors d'un changement de carte E/S ou lors de l'ajout d'un capteur ou d'un actionneur, il est nécessaire de modifier ce bloc. Concrètement, pour ce banc d'essais, il faut pouvoir ajouter la lecture d'un deuxième encodeur optique lorsqu'il est nécessaire dans le montage (par exemple lors du test d'un algorithme de commande par retour d'états avec lecture de la position du premier volant). Le cas physique d'E/S avec un encodeur optique et une sortie analogique pour la commande est représenté à la figure AI.4. Le bloc (quadrature) dont l'identificateur contient l'illustration d'un globe terrestre permet la lecture du paramètre de fonctionnement dans le fichier des paramètres globaux.



FigureAI 2: Diagramme pour le bloc sys_sr_e1 de l'implantation LabVIEW



FigureAI 3: Diagramme pour le bloc sim contenu dans le bloc sys sr_e1

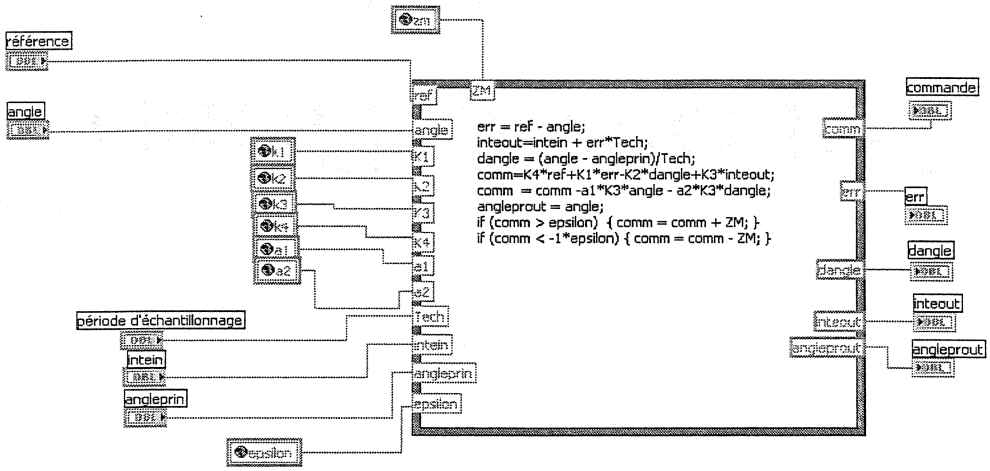


FigureAI 4: Diagramme pour le bloc real contenu dans le bloc sys sr_e1

Bloc PIDDLK4+ZM

Le bloc *PIDDLK4+ZM* contient l'algorithme de commande en cours de test. Dans le cas représenté, il s'agit d'un algorithme PID Dual Loop (PIDDL) avec gain anticipatif (K4) et compensation de la zone morte (ZM). L'algorithme utilise une représentation textuelle facile à comprendre et à modifier, ce qui facilite le remplacement de ce bloc pour tester différents algorithmes. Le détail théorique en arrière de cet algorithme est laissé au lecteur puisqu'il ne fait pas partie des objectifs de ce travail de recherche. Le panneau principal de la boucle de commande contient tous les paramètres nécessaires au fonctionnement du contrôleur et les envoie dans le fichier

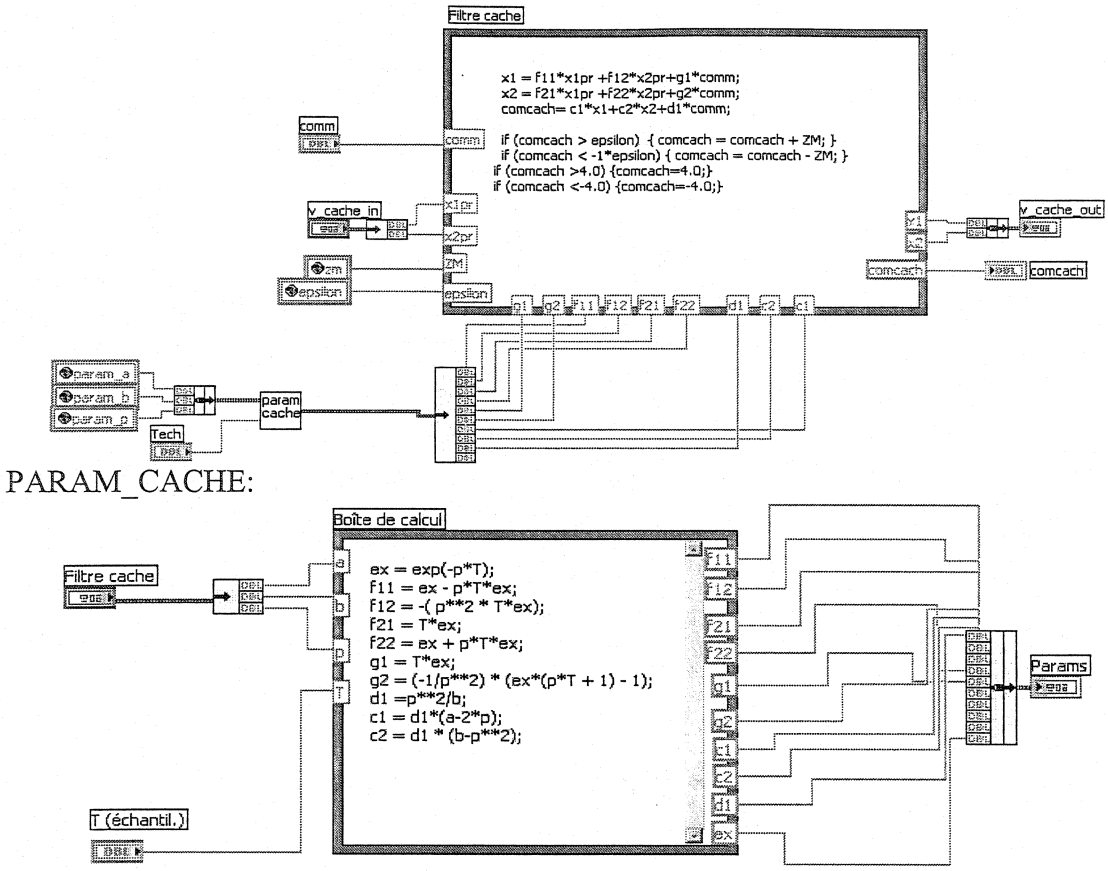
des paramètres globaux. Ces valeurs (*k1, k2, k3, k4, a1, a2, zm, epsilon*) sont ensuite récupérées de la même manière que dans les blocs précédents.



FigureAI 5: Diagramme pour le bloc piddlk4+zm de l'implantation LabVIEW

Bloc FILTRE CACHE

Le contenu du bloc *FILTRE CACHE* peut être inclus dans le bloc du contrôleur car il fait partie de l'algorithme de commande. Il est cependant exclu du bloc contrôleur pour permettre une division plus complète des éléments de la boucle. D'ailleurs, le panneau principal de la boucle permet de l'inclure ou non dans les calculs selon un sélecteur présent sur l'interface. Encore une fois, le détail théorique de l'algorithme est laissé au lecteur puisqu'il ne fait pas partie des objectifs actuels. Notez qu'il est divisé en deux blocs: l'un pour l'attribution des paramètres (forme temporelle du filtre) et l'autre pour l'application du filtre à la commande. Le panneau principal permet l'ajustement des paramètres du filtre (*param_a, param_b, param_p, epsilon, zm*) qui sont ensuite envoyés dans le fichier des paramètres globaux et récupérés comme dans les blocs présentés précédemment. La figure AI.6 fournit une illustration de l'implantation des deux sections de ce bloc.



FigureAI 6: Diagramme pour le bloc filtre cache de l'implantation LabVIEW

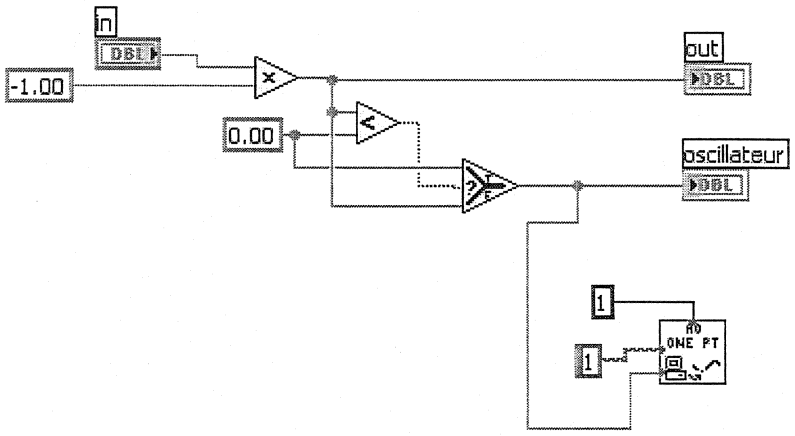
Bloc OSC

Le bloc *OSC* contient l'algorithme de génération de l'oscillateur unitaire utile pour l'étude de la stabilité temporelle. Il génère un signal variant entre une valeur de référence et zéro et qui change d'état à chaque itération de la boucle. Ce signal est ensuite envoyé sur le canal 1 de la carte d'E/S analogique. Ce bloc ne nécessite aucun paramètre de fonctionnement et est complètement indépendant des autres. La figure AI.7 illustre le contenu de ce bloc.

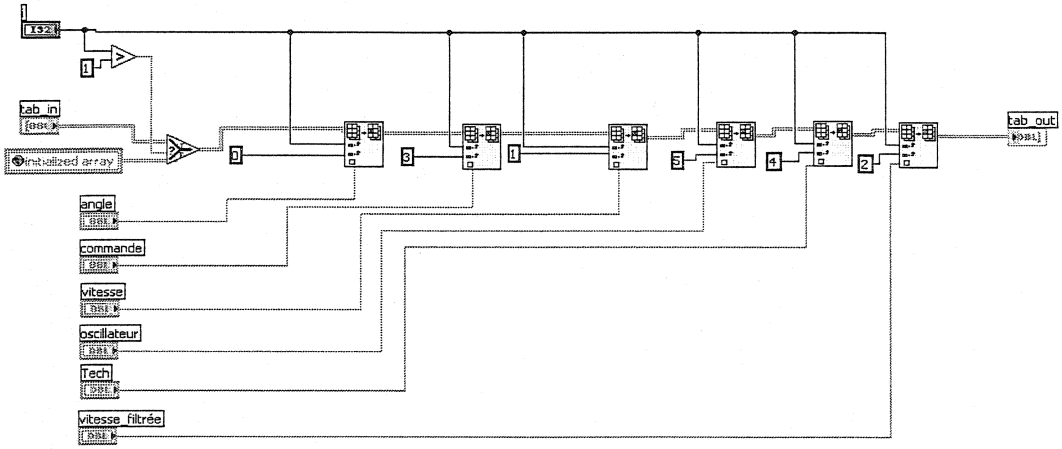
Bloc CREATE TAB PID

Le bloc *CREATE TAB PID* contient les blocs nécessaires à l'indexation des données telle qu'elle a été discutée dans le chapitre 3 de ce mémoire. Ce bloc ne nécessite pas de paramètres de fonctionnement, mais un tableau initial doit lui être fourni. Ce tableau initial contient autant de colonnes que de variables à conserver et

autant de lignes que d'itérations dans l'essai en cours et il est envoyé via le fichier des paramètres globaux. Dans l'implantation actuelle, il contient, dans l'ordre, les vecteurs de données suivants: angle, vitesse, vitesse filtrée, commande, période d'échantillonnage appliquée, et signal de l'oscillateur. Le tableau est créé à chaque itération, mais il sort de la boucle seulement lorsque l'essai est terminé pour ne pas occuper inutilement le processeur lors de l'essai. La figure AI.8 montre le contenu du bloc dans cette implantation.



FigureAI 7: Diagramme pour le bloc osc de l'implantation LabVIEW

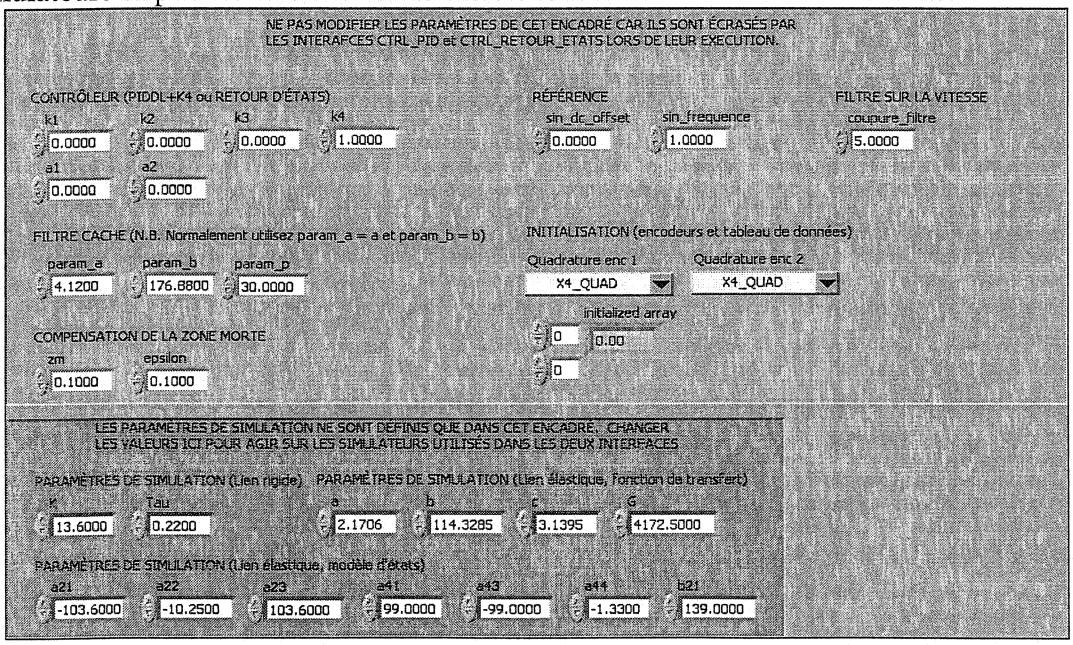


FigureAI 8: Diagramme pour le bloc create tab pid de l'implantation LabVIEW

Paramètres globaux

Pour permettre le fonctionnement de plusieurs blocs sans devoir systématiquement passer tous les signaux via des fils dans le diagramme principal, un fichier spécial a été créé : *global_param.vi*. Ce fichier contient les paramètres globaux à tout le diagramme. Ce fichier est simplement un panneau (aucun diagramme associé) dont les contrôles sont accessibles dans n'importe quel autre diagramme, soit en lecture, soit en écriture. Cette méthode permet de passer directement un paramètre à un bloc comme c'est le cas avec les gains du contrôleur. Il serait aussi possible de passer le même paramètre à plus d'un bloc, mais cet exemple n'est pas illustré dans cette implantation.

Dans un diagramme, tous les signaux dont l'identification contient une icône représentant le globe terrestre proviennent de ce fichier, soit pour la lecture (signal sortant) ou l'écriture (signal entrant). La figure AI.9 montre le panneau du fichier *global_param.vi*. On y retrouve les paramètres du contrôleur, de la référence, du filtre sur la vitesse, de l'initialisation, de la compensation de la zone morte et des différents simulateurs implantés lors des essais effectués lors de ce travail de recherche.



FigureAI 9: Panneau du fichier des paramètres globaux utilisés par l'implantation LabVIEW

Résultats d'exécution

Afin de mieux démontrer les possibilités offertes par l'implantation de la boucle de contrôle dans LabVIEW, un double test a été effectué. Cet essai consiste en une simulation et une expérimentation utilisant les paramètres suivants:

- signal sinusoïdal comme référence;
- contrôle en boucle ouverte (k4=1 et tous les autres gains à 0 avec le contrôleur *PIDDLK4+ZM*);
- absence de filtre cache;
- absence de compensation de la zone morte.

Les paramètres utilisés pour le simulateur sont très approximatifs, car les tests en boucle ouverte nécessaires à leur identification exacte n'ont pas été effectués sur le banc d'essais actuellement utilisé. Les résultats présentés dans cette section ont été obtenus en environ 20 minutes. Seul le fichier *CTRL_PID.VI* a été utilisé pour les essais en variant les paramètres présents sur l'interface du panneau. L'analyse pour l'obtention des figures s'est faite avec le script Matlab *plot_all_labview.m* qui s'occupe de charger les résultats et de tracer les graphiques. Ce script est recopié dans les lignes qui suivent.

```
% plot_all_labview.m
% chargement des résultats (expérimentation)
asser_lv_bo_sin_exp
close all
% garder les données utiles
t_exp=temps;
p_exp=position;
v_exp=vitesse;
% chargement des résultats (simulation)
asser_lv_bo_sin_sim
close all
% garder les données utiles
t_sim=temps;
p_sim=position;
v_sim=vitesse;
% paramètres du simulateur pour l'essai mentionné
a=1.6848; b=82.8845; c=2.9176; G=3563;
% tracer la figure de position
figure;
plot(t_sim,p_sim,':g',t_exp,p_exp,'-b')
axis([t_sim(1) t_sim(length(t_sim)) 0 90])
grid
title('Résultats pour l''angle (simulation vs expérimentation)')
xlabel('Temps (secondes)')
ylabel('Angle (radians)')
m=['(a=' num2str(a) ';b=' num2str(b) ';c=' num2str(c) ';G=' num2str(G) ')']
legend(['Simulation ' m],'Expérimentation',0)
```

```
figure;
plot(t_sim,v_sim,':g',t_exp,v_exp,'-b')
axis([temps_sim(1) temps_sim(length(temps_sim)) -50 50])
grid
title('Résultats pour la vitesse (simulation vs expérimentation)')
xlabel('Temps (secondes)')
ylabel('Vitesse (radians / seconde)')
m=['(a=' num2str(a) ';b=' num2str(b) ';c=' num2str(c) ';G=' num2str(G) ')']
legend(['Simulation ' m],'Expérimentation',0)
```

Selon les résultats des figures AI.10 et AI.11, il est clair que le simulateur donne un trop grand gain statique par rapport au système réel. Cependant, quelques essais avec une bonne analyse des résultats permettraient d'obtenir une excellente correspondance.

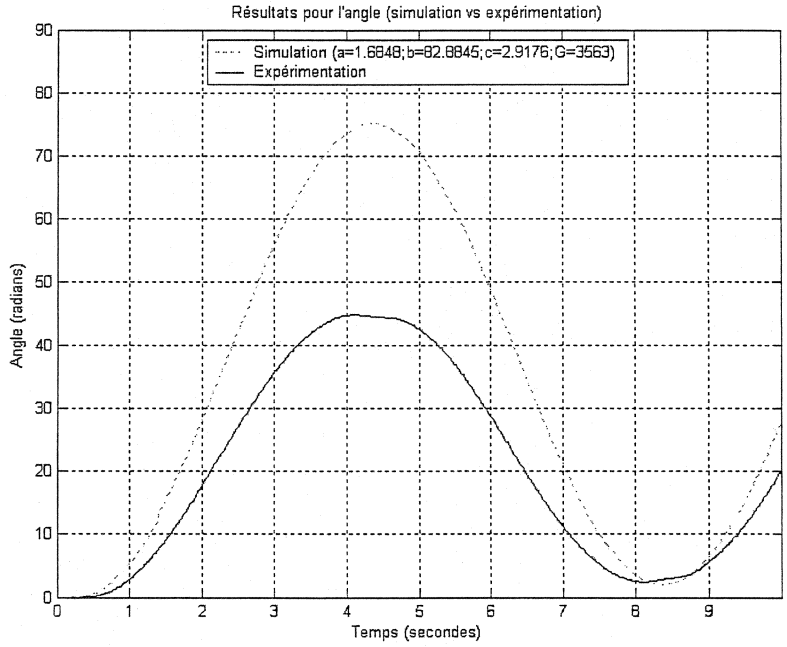


Figure AI.10: Résultats de simulation et d'expérimentation (angle)

Malgré la performance un peu décevante de LabVIEW au niveau de la période d'échantillonnage (voir le chapitre 3 de ce mémoire), les résultats obtenus sont intéressants: les courbes ne sont pas excessivement bruitées et le contrôle est fonctionnel. Évidemment, un contrôle en boucle fermée trop strict (gains élevés) est impossible, car la période d'échantillonnage utilisée (Tech=20ms) est relativement lente. L'objectif n'est pas de démontrer cette affirmation, mais l'expérience dans le laboratoire illustre clairement ce phénomène d'instabilité relié à la période d'échantillonnage trop longue.

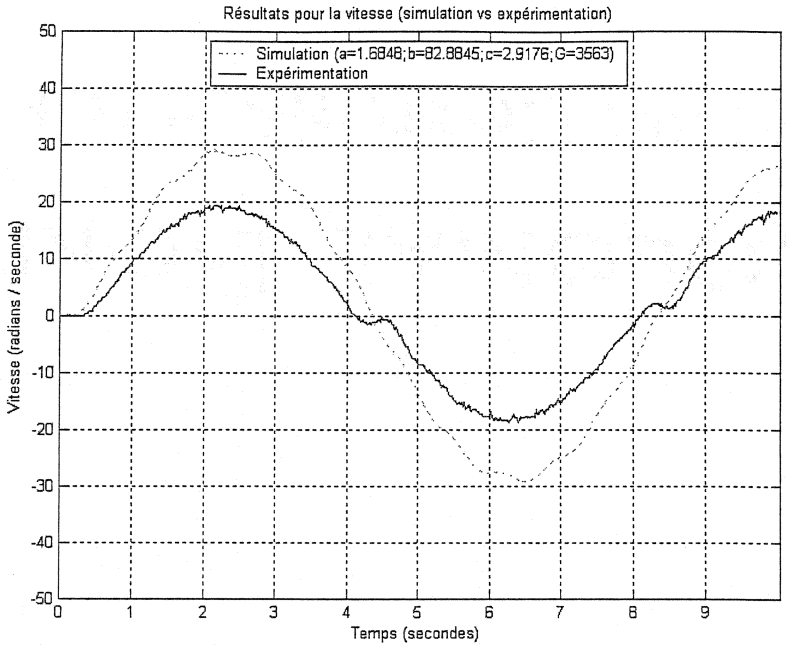


Figure AI.11: Résultats de simulation et d'expérimentation (vitesse)

ANNEXE II: Schémas d'implantation xPC Target du banc d'essais en asservissement de position

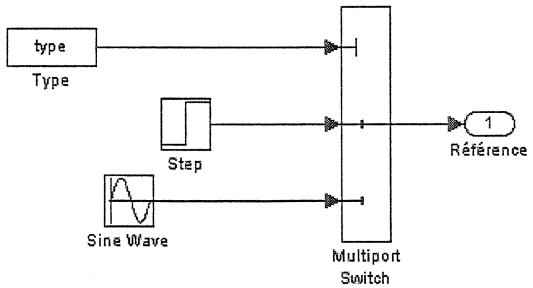
Afin de fournir une meilleure illustration du travail réalisé, voici les différents blocs programmés avec xPC Target pour l'implantation du banc d'essais en asservissement de position. Ces figures sont une reproduction directe des schémas de Simulink et des scripts Matlab.

Sections du schéma Simulink

D'aspect très simple, le schéma principal de la boucle de commande regroupe essentiellement cinq blocs qui sont en fait des sous-systèmes: *Reference*, *Controleur*, *Compensation zone morte*, *Filtre cache* et *Systeme*. Le reste des fonctions sont assurées directement par le schéma principal et sont discutées à la fin.

Bloc REFERENCE

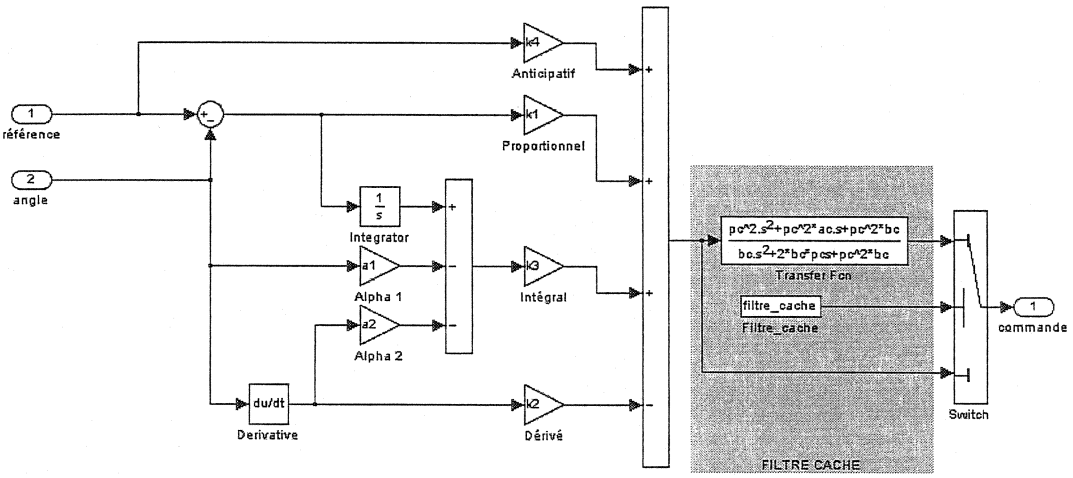
Comme pour l'implantation LabVIEW, ce bloc regroupe deux type de signaux de référence: un échelon ou une sinusoïde. Un bloc sélecteur permet de choisir l'un ou l'autre des signaux selon la valeur du paramètre *type*. À la différence de LabVIEW, tous les paramètres de ce bloc se retrouve dans le fichier script de configuration et n'ont pas à être passés au bloc. Seule une sortie est nécessaire pour envoyer la référence dans la boucle de commande. La figure AII.1 Illustre le contenu du bloc.



FigureAII 1: Schéma du bloc *Reference* de l'implantation xPC Target

Bloc CONTROLEUR

Le bloc *CONTROLEUR* contient l'algorithme du contrôleur en cours de test. Les fonctions disponibles dans Simulink conviennent très bien à une implantation graphique de l'algorithme. L'utilisation d'une fonction entièrement codée dans un script (algorithme textuel) est possible , mais la représentation graphique offre une meilleure lisibilité de l'algorithme. Les paramètres de ce bloc sont contenus dans le même fichier script d'initialisation que le bloc précédent. Ce fichier script peut être exécuté par le schéma principal avant l'exécution de la boucle de commande. Ce bloc peut être facilement substitué par un autre algorithme de commande en ne remplaçant que les paramètres de fonctionnement dans le script d'initialisation. Actuellement, la figure AII.2 implante l'algorithme PIDDL avec gain anticipatif comme dans l'implantation LabVIEW.

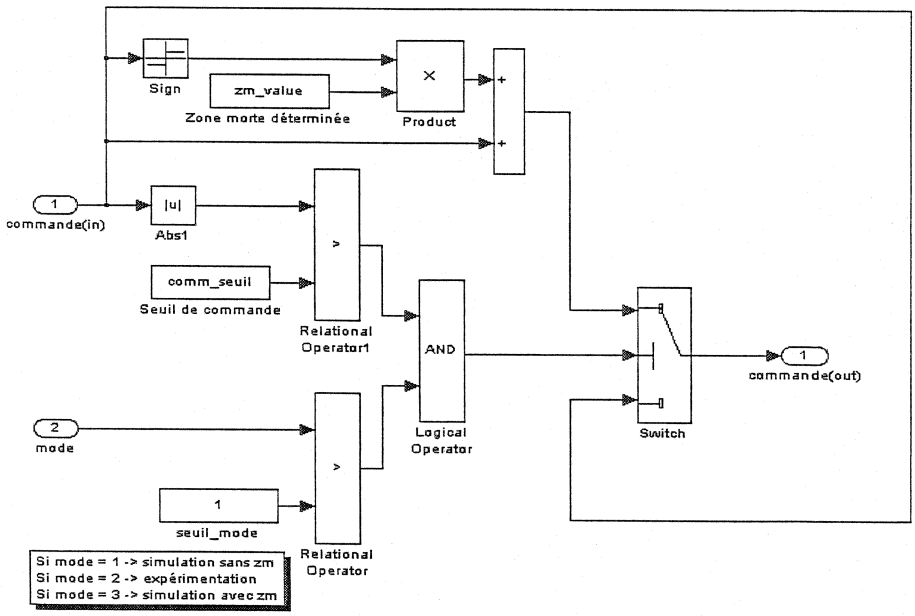


FigureAII 2: Schéma du bloc *Controleur* de l'implantation xPC Target

Bloc COMPENSATION ZONE MORTE

Le bloc Compensation zone morte contient l'algorithme de compensation de la zone morte, comme son nom l'indique. La compensation est réalisée à l'extérieur du bloc contrôleur simplement pour mieux l'isoler lors des essais en décidant si oui ou non la compensation est effectuée. La figure AII.3 illustre l'algorithme. Le paramètre *mode* qui permet de fixer le mode de fonctionnement de la boucle de commande (simulation

ou expérimentation) permet aussi de décider si la compensation de la zone morte est réalisée ou non. Les autres paramètres de ce bloc (*zm_value, comm_seuil*) sont inclus dans le fichier d'initialisation.



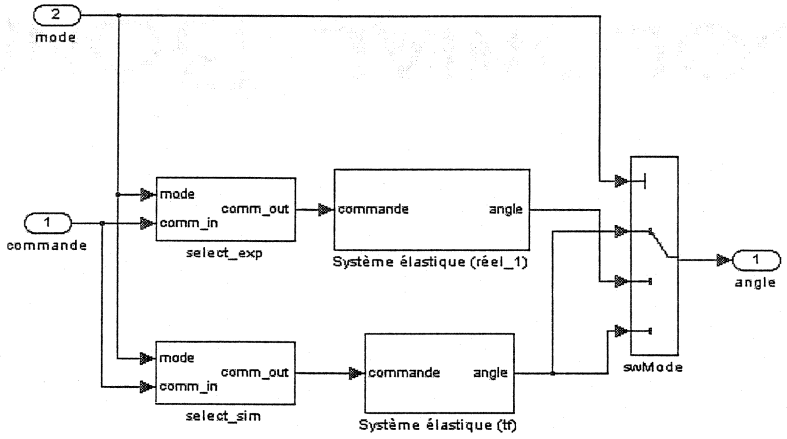
FigureAII 3: Schéma du bloc *Compensation zone morte* de l'implantation xPC Target (indépendant)

Bloc FILTRE CACHE

Comme dans le cas de LabVIEW, cette section de l'algorithme de commande est séparée du calcul principal afin de mieux diviser le traitement de la commande. Cependant, les calculs sont inclus dans le schéma du contrôleur pour conserver un diagramme général simple pour la boucle de contrôle. Les éléments qui se trouvent dans la boîte grisée à la figure AII.2 réalisent l'implantation du filtre cache: une fonction de transfert et un sélecteur suffisent. Le paramètre *filtre_cache* permet d'activer ou non le filtre en variant la position du sélecteur. Comparativement à l'implantation LabVIEW, il y a une grande simplification en passant directement par la fonction de transfert sans devoir passer par la transformation de Laplace inverse pour retrouver la forme temporelle du filtre.

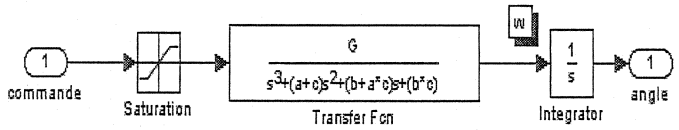
Bloc SYSTEME

Comme dans le cas de LabVIEW, le bloc *Systeme* doit permettre le passage rapide entre la simulation et l'expérimentation. Comme le montre la figure AII.4, l'implantation contient un simulateur, un module d'E/S réelles et la logique de sélection permettant le passage d'un système à l'autre en changeant simplement le paramètre *mode*.



FigureAII 4: Schéma du bloc *Systeme* de l'implantation xPC Target

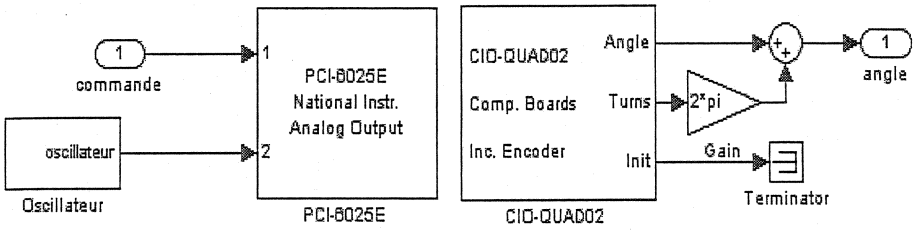
Le bloc *Système élastique (tf)* contient le simulateur qui est implanté graphiquement comme tous les autres algorithmes utilisés dans l'implantation xPC Target. La figure AII.5 illustre l'implantation avec la fonction de transfert du lien élastique. Le bloc peut être remplacé à tout moment suivant la modification simultanée du fichier de configuration pour inclure les paramètres du nouvel algorithme.



FigureAII 5: Schéma du bloc *Système élastique (tf)*

Le bloc *Système élastique (réel_1)* contient le pilote pour la lecture de l'encodeur optique, ainsi que le pilote pour la sortie de la commande sur le canal 0 de la carte d'E/S analogique. Ces pilotes se trouvant dans la librairie spécialisée de xPC Target, il suffit de connaître le modèle de la carte utilisée et d'inclure le bloc correspondant dans le schéma. L'ajout d'E/S est possible en modifiant les paramètres de ces pilotes. S'il y a un changement de carte, alors il faut trouver la nouvelle carte dans la librairie. Si le pilote est absent de la librairie, il est possible de le créer à partir des données du fabricant et des

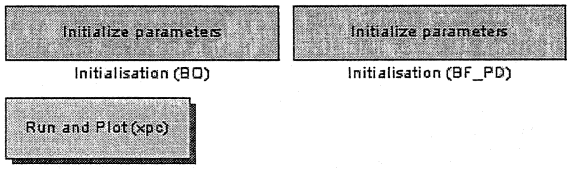
fichiers modèles fournis par xPC Target. La figure AII.6 illustre l'implantation actuelle avec la carte CIO-QUAD02 pour l'encodeur optique (fournit la position angulaire du deuxième volant) et la carte PCI-6025E pour la sortie analogique (commande et oscillateur).



FigureAII 6: Schéma du bloc *Système élastique (réel_1)*

Autres blocs

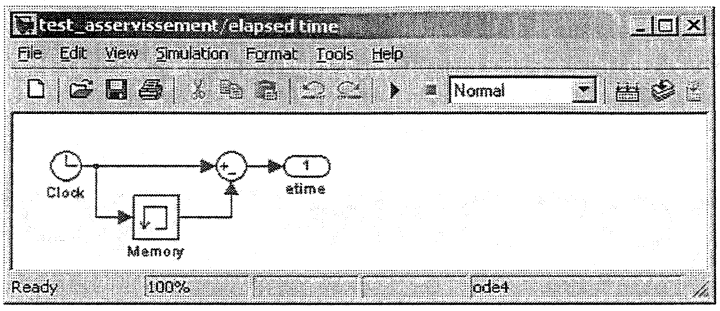
Le schéma général contient quelques autres blocs qui servent au contrôle des simulations et à la sortie des résultats utiles. Les trois blocs en haut à gauche (voir figure AII.7) sont des boutons de commande qui lors d'un double clic exécutent des commandes pré-établies: initialisation de la boucle ouverte, initialisation de la boucle fermée, et exécution avec traçage des résultats. Ces commandes sont regroupées dans des scripts Matlab réalisés lors de l'implantation de la boucle.



FigureAII 7: Commandes incluses dans le diagramme de la boucle de commande

Le bloc *Mean* permet de filtrer les données selon une moyenne pondérée de vingt itérations du signal de la dérivée de l'angle, ce qui donne la vitesse filtrée du système (la filtration est nécessaire car l'encodeur optique fournit un signal dont la dérivée est très bruitée).

Le bloc *Elapsed time* effectue la différence entre la valeur actuelle et la valeur à l'itération précédente du signal d'horloge, ce qui permet le calcul direct de la période d'échantillonnage lors de l'exécution dans l'environnement temps réel.



FigureAII 8: Schéma interne du bloc *Elapsed Time*

Le bloc *Systeme elastique reel 1* contient aussi un bloc Oscillateur comme le montre la figure AII.6. Ce bloc contient le diagramme de l'oscillateur qui permet le test de la période d'échantillonnage avec l'oscilloscope. La figure 3.15 du chapitre 3 résume cette implantation en montrant le contenu du bloc.

Résultats d'exécution

Afin de mieux démontrer les possibilités offertes par cette implantation, le même test qu'à l'annexe I a été effectué. Les résultats illustrés dans cette section ont été obtenus en 10 minutes, analyse des données incluse. Seul le script *test_tech_xpc.m* et le modèle *tests_asservissement.mdl* ont été utilisés pour réaliser les essais, récupérer les résultats et afficher ces derniers. Le script est reproduit dans les lignes qui suivent.

```
% test_tech_xpc.m

idmode=getparamid(tg,'mode','Value');
% Chercher manuelle l'identificateur du paramètre P7 de l'item
%...
idtechcio='P10';
% Chercher manuelle l'identificateur du paramètre P5 de l'item
%...
idtechpci='P33';
mode=input('Mode de fonctionnement: simulation (1) ou expérimentation (2)?');
set(tg,idmode,mode);
tech=input('Période d''échantillonnage à utiliser (secondes)?');
tg.sampletime=tech;
set(tg,idtechcio,tech);
set(tg,idtechpci,tech);
stoptime=input('Nombre d''itération à effectuer pour la boucle de commande?');
stoptime=stoptime*tech;
tg.stoptime=stoptime;
+tg;
disp('Exécution en cours...')
while str2num(tg.exectime)<tg.stoptime
  pause(1);
end
```

```
out=getlog(tg,'outputlog');
time=getlog(tg,'timelog');
tet=getlog(tg,'tetlog');
mt=mean(tet(2:length(tet)));
mtd=mt*ones(1,length(tet));
figure;
plot(time(2:length(time)),1000000*tet(2:length(tet)))
hold
plot(time(2:length(time)),1000000*mtd(2:length(tet)),'-r','linewidth',3)
title('Variation du temps de calcul (TET) au fil de l''exécution');
xlabel(['Temps (secondes) - Tech = ' num2str(1000*tech) ' ms']);
ylabel('Temps de calcul (microsecondes)')
legend('Temps de calcul',['Moyenne ( ' num2str(1000000*mt) ' us )'],0)
figure;
plot(time(2:length(time)),1000*out(2:length(time),2));
axis([time(1) time(length(time)) 1000*tech-1 1000*tech+1]);
title('Variation de la période d''échantillonnage au fil de l''exécution');
xlabel(['Temps (secondes) - Tech = ' num2str(1000*tech) ' ms']);
ylabel('Période d''échantillonnage (millisecondes)')
delta_tech=out(:,2);
uisave
```

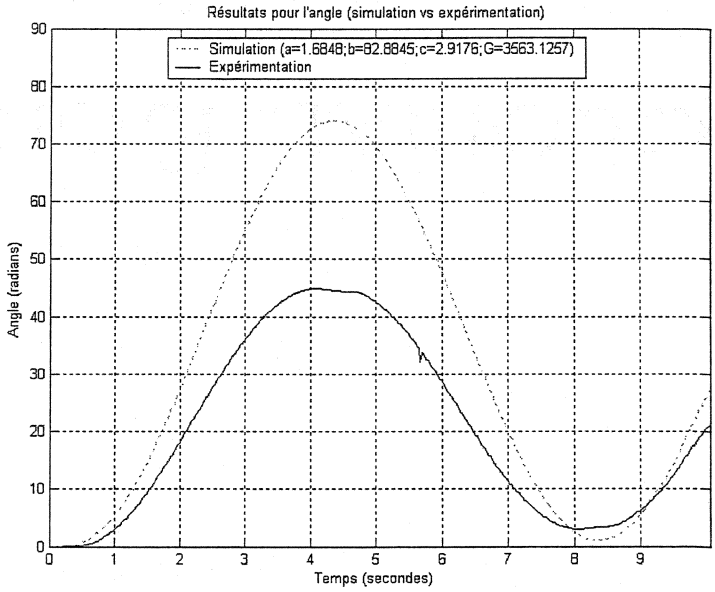
Les résultats sont conservés à la fin de chaque essai via la commande *uisave* et la récupération post-exécution est possible via la commande *uiload.* La création du graphique passe les fonctions habituelles de Matlab (*plot*, *hold*, *legend*, etc).

Comme le montrent les figures AII.9 et AII.10, il est clair que le simulateur donne un trop grand gain statique par rapport au système réel. C'est le même phénomène que lors de l'implantation précédente, les paramètres du simulateur sont mal ajustés. Quelques essais permettraient de rétablir cette situation qui ne met pas en cause le fonctionnement de cette implantation.

L'excellente performance de xPC Target au niveau de la période d'échantillonnage nous permet d'obtenir d'excellents résultats: les courbes ne sont pas bruitées et le contrôle est parfaitement fonctionnel.

Un phénomène intéressant est survenu lors de cet essai: une pointe de grande amplitude sur la courbe de vitesse (voir figure AII.10). Cette pointe est due au fait que l'encodeur optique rate parfois une itération, ce qui donne une vitesse instantanée immense pour l'itération en cours. Ce problème est d'ordre matériel et n'a rien à voir avec l'environnement d'exploitation. En effet, pour créer à l'interne une pointe de cette amplitude, la période d'échantillonnage doit être totalement erronée et xPC Target aurait alors cessé l'exécution. De plus, la courbe expérimentale de l'angle confirme le

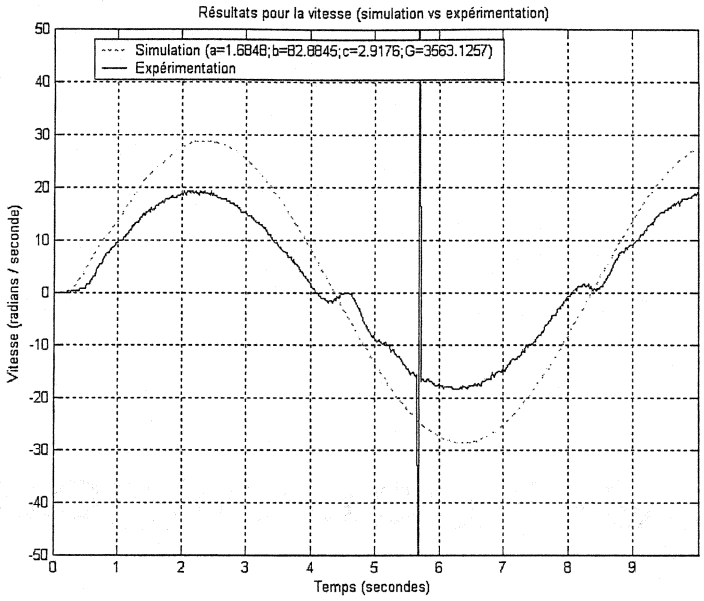
changement brusque de position à cet instant. Cette pointe, jointe à une période d'échantillonnage de 1 ms donne l'aberration observée sur la courbe de vitesse.



FigureAII 9: Résultats de simulation et d'expérimentation (angle)

Une comparaison des résultats avec ceux obtenus avec LabVIEW montre seulement de faibles différences. En fait, la différence provient surtout de la période d'échantillonnage: l'essai dans xPC Target utilise une période de 1 ms et l'essai dans LabVIEW une période de 20 ms. Ceci explique entre autres l'aspect plus oscillant de la simulation de vitesse dans LabVIEW. De plus il est impossible de garantir que les méthodes de calcul sont exactement les mêmes dans les deux environnements.

Globalement, les résultats correspondent et les deux implantations permettent d'effectuer les tests. Cependant, xPC Target est plus performant et plus rapide à utiliser dans le cas présent vue la nécessité d'analyser les données dans Matlab.



FigureAII 10: Résultats de simulation et d'expérimentation (vitesse)

ANNEXE III: Code pour la commande par port série de l'oscilloscope TDS-2012

COMMANDE PAR PORT SÉRIE

OSCILLOSCOPE TDS2012 DE TEKTRONIX

Par Cédric Demers-Roy, Mai 2004

Les quelques fonctions présentées dans les pages qui suivent permettent l'essentiel des communications avec l'oscilloscope et des programmes maisons peuvent être implantés avec la fonction *send_comm* et le manuel de programmation de l'oscilloscope qui présente l'usage de chacune des commandes envoyées. Le script d'exemple permet de rapidement avoir une vue globale du fonctionnement de l'oscilloscope avec le lien de communication série. Puisque la totalité du code réside dans Matlab et que la récupération du signal est sous forme de vecteurs, l'analyse des données est grandement simplifiée.

Matériel nécessaire

Ces fonctions ont été réalisées pour communiquer avec l'oscilloscope via le port de communication série COM1 d'un ordinateur de bureau (ou d'un portable). Il faut un minimum de matériel pour assurer leur fonctionnement:

- l'oscilloscope TDS 2012 (les programmes peuvent peut-être fonctionner avec d'autres oscilloscopes de la même famille, mais nous n'avons pas fait de tests à cet effet);
- le module de communication TDS2CMA ou le module de mémoire et de communication TDS2MEM installé sur l'oscilloscope (pour le support des commandes utilisées);
- un câble série avec connecteurs DB9 femelle – femelle;
- l'installation du pilote TEKVISA (CD fourni avec l'oscilloscope).

Pour les tests, la borne de test en bas à droite de l'oscilloscope permet de mesurer une onde carrée 0-5 VDC à une fréquence de 1 kHz. Évidemment, n'importe quel signal peut être mesuré et acquis via ces programmes (sauf pour le programme exemple *comm_oscillo.m* qui se base sur ce signal de test pour avoir les sorties attendues).

Programmes fournis

Ces programmes ont été réalisés avec la version 6.5 de Matlab et nous ne les avons pas testés avec aucune autre version du logiciel. Théoriquement, le code est compatible avec toutes les versions dans la mesure où la boîte à outils de communication série se trouve dans les options installées. Évidemment, pour fonctionner ces scripts nécessitent de relier l'oscilloscope au port série (COM1) et l'oscilloscope doit être mis en fonction et avoir terminé avec succès ses tests de démarrage (si ce n'est pas le cas, l'oscilloscope doit être vérifié).

La première opération à effectuer dans tous les cas est de lancer le script *init_comm* à l'invite de commande Matlab. Ce script va simplement construire l'objet *s* servant à la communication via le port série. Toutes les fonctions ont besoin de cet objet, il ne faut surtout pas le détruire en cours de route... Le script fait ensuite quelques tests pour s'assurer du fonctionnement du lien avant de redonner le contrôle à Matlab.

La suite dépend entièrement de l'usager, mais le script *comm_oscillo* est un bon point de départ puisqu'il regroupe plusieurs fonctions utiles et qu'il est bien commenté à la fois lors de l'exécution et dans le code. Voici une brève description de chacun des programmes.

init_comm

Type: *script*

Paramètres: *aucuns*

Sorties: *l'objet s représentant le port série pour toutes les autres fonctions*

Ce script sert de départ à toute session d'échange avec l'oscilloscope car c'est lui qui crée l'objet de communication série avec les bons paramètres pour pouvoir

communiquer avec l'oscilloscope. Il initialise le port et ajuste entre autres la vitesse de communication à 9600 bauds. Vous devez vous assurer que l'oscilloscope utilise la même vitesse. Pour vous en assurer et / ou modifier cette valeur sur l'oscilloscope, simplement aller dans le menu *UTILITY / OPTIONS / CONFIGURASTION RS-232* et consulter / modifier le choix *BAUD*.

Si dans ses affichages le script donne la mention *INITIALISATION TERMINÉE ANORMALEMENT*, c'est que l'oscilloscope n'a pas retourné TDS2012 comme identificateur et nous ne sommes pas sûr du fonctionnement des autres programmes dans ce cas précis.

erreur_comm

Type: *fonction*

Paramètres: *s, l'objet de communication*

Sorties: *m, message qui explique l'erreur survenue (en réponse à une commande)*
e, code de l'erreur survenue (en réponse à une commande)

Cette fonction permet de détecter d'éventuelles erreurs lors de l'envoi d'une commande à l'oscilloscope. Il est recommandé d'utiliser cette fonction après chaque envoi de commande pour s'assurer de la validité de la réponse (s'il y en a une). La fonction *send_comm* intègre cette fonction afin d'effectuer cette vérification justement. Si le code d'erreur retourné est 0, alors il n'y a pas d'erreur et la réponse de l'oscilloscope est jugée correcte. Un bon exemple d'erreur est lorsque nous demandons la mesure d'une fréquence alors que le signal mesuré est non périodique, l'oscilloscope va détecter cette anomalie.

check_comm

Type: *fonction*

Paramètres: *s, l'objet de communication*

Sorties: *e, code de l'erreur de communication*
m, message explicant l'erreur de communication

Cette fonction sert de vérification pour les communications en début d'autres programmes. Elle vérifie que l'objet *s* existe et qu'il est fonctionnel pour communiquer avec l'oscilloscope. Il vérifie l'existence du port COM1, l'ouverture du port, la vitesse de 9600 bauds, le tampon de 8192 octets et le délai d'attente de 50. L'ouverture du port est réalisée lorsque nécessaire et les autres vérifications offrent la possibilité de remédier à la situation en offrant le choix à l'usager via une fenêtre de questionnement. La fonction vérifie aussi si des données sont toujours présentes dans le tampon de réception. Cette vérification est importante car sinon les réponses aux commandes subséquentes seront décalées dans le tampon. S'il reste des données, elles sont récupérées, mais la fonction termine avec une erreur indiquant cet état du port. S'il reste encore des données et que l'affichage est actif (pas de ';' après l'appel de la fonction), alors elle affiche le nombre d'octets encore disponible. À la limite, il est possible d'appeler à nouveau cette fonction de test tant que l'erreur retournée n'est pas zéro... La seule erreur que la fonction ne peut accepter est l'inexistence de l'objet *s*.

comm_oscillo

Type: *script*

Paramètres: *s, l'objet de communication (défini dans l'espace actuel de travail)*

Sorties: *messages divers dans Matlab (le script contient plusieurs pauses)*

Ce programme est un exemple de communication avec l'oscilloscope qui contient beaucoup de commentaires, autant dans le code que par des affichages divers dans Matlab. Il constitue un bon point de départ pour comprendre le fonctionnement de l'oscilloscope et le protocole d'emploi des commandes et des différents scripts fournis ici. Pour donner les résultats attendus, ce script nécessite la mesure du signal de test de l'oscilloscope via le canal 1. Pour ce faire, simplement relier la sonde de ce canal à la borne de test au bas de l'oscilloscope et assurez-vous via la commande *autoscale* de ce dernier que l'onde carrée 0-5 VDC à 1 kHz est bel et bien mesurée. Lorsque c'est le cas, lancez le script. Plusieurs messages vont alors s'afficher avec quelques vérifications d'erreurs. Normalement, une seule erreur doit se produire vers le milieu du script avec

les messages suivants:

(Commmande 'measu:immed:type freq' pour une mesure immédiate de fréquence)

(Commmande 'measu:immed:value?' pour obtenir la mesure en question)

Réponse: :MEASUREMENT:IMMED:VALUE 9.9E37

*****Erreur: Code: 16; Message: (:ALLEV 2202,"Measurement error, No period found; ")*

Cette erreur est normale car si vous observez bien le signal acquis, vous notez qu'il ne contient qu'une transition de 0 à 5 VDC, il n'y a donc pas de période et donc pas de mesure de fréquence possible. Lorsque la commande de mesure de fréquence est envoyée, elle retourne une valeur à la puissance 37, ce qui est aberrant, l'erreur que retourne l'oscilloscope est justement que la période n'a pas pu être mesurée, d'où le chiffre aberrant. **N.B. Il se peut qu'au début du script l'oscilloscope retourne l'erreur suivante pour indiquer la mise en fonction du module de communication (ce qui n'est pas vraiment une erreur mais plutôt un événement interne):**

Erreur: Code: 128; Message: (:ALLEV 401,"Power on; ")

send_comm

Type: *fonction*

Paramètres: *s, l'objet de communication*
comm, la commande à envoyer (chaîne de caractères)
echo, le choix d'afficher ou non les messages anodins (affiche si 1)
byval, le choix de convertir une réponse en valeur numérique (conversion si 1)

Sorties: *m, le message de réponse si la commande nécessite une réponse*
mn, la valeur numérique contenue dans la réponse si la conversion est demandée

Cette fonction est de loin la plus complexe de celles que nous proposons, elle permet l'envoi de théoriquement n'importe quelle commande à l'oscilloscope, avec ou sans réponse et avec ou sans conversion de la réponse au format numérique. Elle effectue toutes les vérifications d'erreur et retourne les éventuels messages d'erreur dans la chaîne de réponse à la commande. Si une conversion numérique est demandée, il faut

alors quand même consulter la réponse textuelle pour s'assurer que la réponse est valide...

La fonction détecte automatiquement si elle doit ou non aller chercher une réponse en détectant la présence du *'?'* dans la commande envoyée. Si une conversion numérique est demandée (*byval=1*), alors la fonction retourne aussi la variable *mn* qui contient la valeur numérique de la réponse. Si la commande envoyée ne donne pas de réponse numérique, alors *mn* sera vide et si la conversion n'est pas demandée *mn* a la valeur nulle.

get_wave

Type: *script*

Paramètres: *s, l'objet de communication (défini dans l'espace actuel de travail)*

Sorties: *messages divers dans Matlab, les vecteurs t et y pour le temps et le signal ainsi que tous les paramètres du signal (préambule)*

Ce script regroupe toutes les commandes pour récupérer le préambule et les données pour le signal actuellement affiché sur l'oscilloscope. Évidemment, ce dernier doit avoir la mention *'Trigd'* ou *'Acq complete'* pour que ce signal veuille dire quelque chose (l'oscilloscope effectue une acquisition instantanée sinon). Une fois le signal récupéré, le script regroupe les fonctions de conversion pour retrouver le signal tel qu'il est affiché sur l'écran de l'oscilloscope, les vecteurs *t* et *y* contiennent respectivement le temps et le signal convertis. La commande *plot(t,y)* lancée après ce script devrait donner une figure qui ressemble beaucoup à l'écran de l'oscilloscope.

Ce script peut récupérer des données acquises manuellement (bouton *autoscale* de l'oscilloscope et autres ajustements) ou par un script comme celui contenu dans *acquire_and_get*. Les valeurs récupérées peuvent être conservées dans un fichier *MAT* ou dans une figure, bien que ces commandes d'archivage ne soient pas incluses dans le script.

acquire_and_get

Type: *script*

Paramètres: *s, l'objet de communication (défini dans l'espace actuel de travail)*

Sorties: *messages divers dans Matlab, les vecteurs t et y pour le temps et le signal ainsi que tous les pamètres du signal (préambule) et une figure*

Ce script effectue la configuration d'une acquisition de séquence suivant certains paramètres prédéterminés. Le script *get_wave* est ensuite utilisé pour récupérer les données et une figure est créée en utilisant les données du préambule et les vecteurs de données récupérées. Le script offre la possibilité de sauvegarder les résultats par l'entremise de la commande *uisave* de Matlab.

Il est possible de modifier les paramètres de l'acquisition en modifiant le script, quitte à y ajouter des commandes supportées par l'oscilloscope. Le guide de programmation du module de communication est alors fortement recommandé. Les quelques lignes qui suivent rapportent les options d'acquisition actuellement utilisées. La commande *factory* place l'oscilloscope dans un état prédéterminé à partir duquel il est possible de construire sa propre configuration sans dépendre des paramètres actuels. Les autres commandes ajustent les différents paramètres de notre essai: l'échelle en Volts, l'échelle temporelle, le niveau du déclencheur et le type d'acquisition.

```
fprintf(s, 'factory')
fprintf(s, 'ch1:volts 2.0')
fprintf(s, 'hor:main:scale 500e-6')
fprintf(s, 'trig:main:level 2.4')
% Déclencher une acquisition
fprintf(s, 'acquire:stopafter sequence')
fprintf(s, 'acquire:state on')
```

interface_oscillo(.m et .fig)

Type: *script multi-fonctions et figure*

Paramètres: *s, l'objet de communication (défini dans l'espace actuel de travail)*

Sorties: *aucune*

Programme permettant la création et l'utilisation de l'interface graphique

proposée plus loin.

run_interface_oscillo

Type: *script*

Paramètres: *s, l'objet de communication (défini dans l'espace actuel de travail)*

Sorties: *aucune*

Programme permettant l'emploi de l'interface graphique en définissant l'objet s comme étant global pour être partagé parmi les multiples fonctions de l'interface graphique.

Conception de programmes personnels

Ces programmes regroupent les bases de la communication avec l'oscilloscope, l'usage direct de ces programmes est utile au départ mais ils servent surtout de point de départ pour la conception de programmes complets. La possibilité de récupérer et conserver les données sur un ordinateur est non négligeable et fournit un net avantage vis-à-vis les oscilloscopes analogiques d'où l'importance d'en profiter. Le CD fourni avec ce document contient le code de chacun de ces programmes et peut être utilisé librement en mentionnant la source lorsque les programmes fournis sont directement utilisés.

Conception d'une interface graphique

Un exemple d'interface graphique est aussi fourni avec les programmes afin d'illustrer la capacité peu reconnue de Matlab de fournir une interface automatique pour l'emploi de commandes plutôt que de se limiter strictement à l'invite textuelle de commande. Cette interface est lancée en entrant la commande *run_interface_oscillo* à l'invite de commande de Matlab. Il faut débuter les opérations par la commande *Initialize* au bas de l'interface afin de créer l'objet de communication série (à moins qu'il existe déjà bien sûr).

Bien que très simple, cette interface permet l'initialisation, la configuration par défaut, l'acquisition par ordinateur et la saisie des données en plus de fournir une ligne d'envoi direct de commandes. Ces quelques fonctions sont des appels directs aux programmes présentés dans ce document. La commande *Update* permet de configurer l'acquisition suivant les paramètres proposés sur l'interface (divisions des canaux, divisions temporelles, valeur du déclencheur *trig*). Après cette commande, la commande *Get wave* permet de récupérer le résultat tel qu'il est affiché sur l'écran de l'oscilloscope.

La ligne d'envoi de commande de cette interface supporte deux types d'envoi: l'envoi d'une commande unique ou l'envoi d'une suite de commandes. Le premier type d'envoi demande simplement d'écrire la commande avec ou sans apostrophes (ex.: *'id?'* ou *id?* comme texte entré) et le deuxième type utilise le type cellule (*cell*) de Matlab avec la notation complète (ex.: *{'id?'; 'factory'}* comme texte entré). Les commandes sont envoyées l'une à la suite de l'autre avec récupération et affichage des résultats dans la ligne de message au bas de l'interface. Chaque message demeure affiché une seconde avant l'envoi de la commande suivante. Si la commande ne retourne aucun message, alors la ligne affiche cet état particulier.

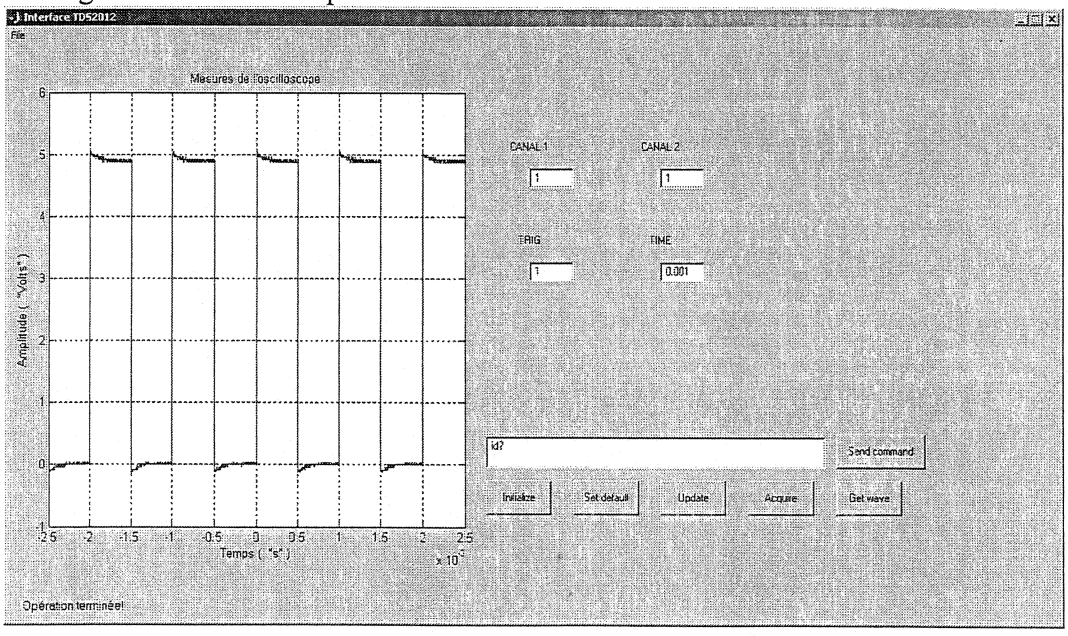


Figure AIII.1 : Exemple d'interface graphique pour la commande de l'oscilloscope

Cette interface n'est qu'une base pour la conception d'un engin complet car elle ne regroupe que très peu de fonctionnalités. L'envoi de commandes multiples est toutefois bien pratique pour effectuer des acquisitions rapides sans passer par un script complet.

Exemple d'utilisation des programmes

Voici point par point un exemple complet d'utilisation des programmes proposés ici (sans l'interface graphique).

1) Lancez la commande *init_comm*.
2) Lancez la commande *comm_oscillo* et observez les résultats afin de vérifier la bonne fonction du module de communication.
3) Utilisez la fonction send_comm:

   send_comm(s,'id?')

   send_comm(s,'measu:immed:type mean')

   send_comm(s,'measu:immed:value?')
4) Utilisez la commande *acquire_and_get* et observez le résultat (une figure représentant l'écran de l'oscilloscope). Vous pouvez alors sauver ou non les résultats (les vecteurs t et y contiennent les valeurs de temps et de mesure et quelques autres variable permettent de retracer les paramètres de mesure utilisés).
5) Changez manuellement les paramètres de l'oscilloscope (divisions du canal 1, divisions temporelles) directement avec les boutons de l'oscilloscope, puis lancez la commande *get_wave* pour obtenir de nouveau l'écran de l'oscilloscope (vous devez faire manuellement la commande *plot(t,y)* lorsque les données sont récupérées car le script *get_wave* ne trace pas les résultats automatiquement).

Ceci donne un exemple complet de ce qu'il est possible de faire avec les fonctions de communication de l'oscilloscope. Le manuel de programmation fourni avec le module de communication devient alors nécessaire pour la conception de programmes plus complexes.