Année 2008-2009
Rapport de stage d'initiation à la recherche de M1

Interfaçage
Essais-Calculs

Xavier Lagorce

Encadré par :
J-F. Witz – witz@lmt.ens-cachan.fr
H. Leclerc – leclerc@lmt.ens-cachan.fr
A. Delaplace – delaplace@lmt.ens-cachan.fr

Laboratoire de Mécanique et Technologie
ENS Cachan / CNRS / Université Paris 6
61, avenue du Président Wilson, F-94235 CACHAN CEDEX

Merci à Jean-François Witz pour son encadrement soutenu ainsi qu'à Hugo Leclerc pour ses vues sur le noyau linux.

Merci aussi à Arnaud Delaplace pour son modèle et son aide dans l'interprétation des résultats.

Merci enfin à tout le personnel du laboratoire m'ayant aidé dans la prise en main des machines et sans qui rien n'aurait été possible.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le LMT-Cachan [1], créé en 1975, est une Unité Mixte de Recherche commune à l'École Normale Supérieure de Cachan, au CNRS [2] et à l'Université Pierre et Marie Curie (UPMC). Il a été dirigé successivement par Jean Lemaitre (1975-1980), Pierre Ladevèze (1981-1984), Mircea Predeleanu (1985-1992), Giuseppe Geymonat (1993-1996) et Pierre Ladevèze (1997-2005). Depuis 2006, le directeur est Olivier Allix.

Les activités du LMT-Cachan concernent la modélisation des solides et des structures : mécanique des matériaux, mécanique expérimentale, simulation numérique et calcul haute performance. Elles relèvent du domaine que les anglo-saxons appellent « Engineering Sciences ». Les fondamentaux du LMT-Cachan reposent sur la recherche du meilleur niveau international dans chacun de ces domaines. La plupart des recherches, motivées par des problèmes et défis industriels et sociétaux, cherchent à dégager des idées, méthodes et concepts permettant d'apporter des réponses à ces défis sur le long terme et sont souvent menées en relation étroite avec d'autres domaines, tels la physique, la chimie, les mathématiques, le calcul scientifique et l'informatique.

Le LMT-Cachan est organisé en 3 secteurs eux-mêmes structurés en unités thématiques de recherche :

- le secteur "Mécanique et Matériaux" :

  Le domaine d'activités du Secteur concerne tous les aspects liés à la modélisation du comportement mécanique des matériaux à l'état solide ou pâteux. L'objectif final est de développer des modèles prédictifs utilisables pour simuler le comportement de structures lors de leur fabrication ou de leur utilisation. Les techniques de modélisation utilisées sont de plus en plus variées, mais une attention particulière est toujours portée à l'identification et à la validation expérimentale des modèles. Les études, fortement soutenues par un partenariat industriel de longue durée, s'articulent autour de cinq Unités Thématiques de Recherche :
  - Multiphysique et procédés
  - Comportement et ruine des matériaux à microstructure aléatoire
  - Comportement, endommagement et instabilités
  - Fissuration et rupture par fatigue
  - Comportement dynamique des matériaux.
- le secteur "Structures et Sytèmes" :

  Les activités du Secteur concernent la modélisation et le calcul des structures et des systèmes mécaniques. Les objectifs principaux sont de repousser les limites actuelles en

1. Laboratoire de Mécanique et Technologie de Cachan
2. Département des Sciences et Technologies de l'Information et de l'Ingénierie, UMR 8535

modélisation et en calcul, de contrôler les calculs proprement dits ainsi que les résultats et donc les modèles utilisés. Une attention particulière est portée aux matériaux et structures composites, ainsi qu'aux procédés d'élaboration des structures et leurs conséquences sur l'intégrité. Même si les études font une place importante aux travaux fondamentaux, la plupart des recherches sont menées en étroite collaboration avec des industriels, jusqu'à la réalisation de logiciels prototypes. Le Secteur est structuré en cinq Unités Thématiques de Recherche :
  - Vérification et validation
  - Composites et microstructures
  - Stratégies de calcul multiéchelle et parallélisme
  - Intégrité des structures
  - Ingénierie et conception robuste.
- le secteur "Génie Civil et Environnement" :

  Les activités de recherche du Secteur sont orientées vers des applications relatives au génie civil et plus particulièrement à l'analyse et à la conception des ouvrages soumis à d'extrêmes conditions mécaniques et environnementales, l'élaboration de nouveaux matériaux de construction, la modélisation de leur comportement mécanique et de leur durabilité dans un environnement donné. Même si nos thèmes de recherche sont issus de problèmes industriels, ces travaux apportent, en général, des idées scientifiques nouvelles et l'avancement de l'état de l'art actuel. Les projets de recherches sont structurés afin de pouvoir apporter des réponses complètes à des problèmes industriels complexes. Par conséquent, nos recherches sont souvent pluridisciplinaires, comportant des simulations numériques, menées au Centre de calcul du LMT-Cachan, ainsi que des études expérimentales, effectuées au Centre d'essais du LMT-Cachan. Le Secteur « Génie civil & Environnement » est structuré en trois Unités Thématiques de Recherche :
  - Ouvrages sous conditions extrêmes
  - Nano/microstructures et durabilité des matériaux dans leur environnement
  - Rhéologie et formulation des nouveaux bétons.

Outre cette structuration en secteur, les chercheurs disposent d'une bibliothèque et de trois centres de moyens : le Centre d'Essais, le Centre de Calcul et la Cellule "Logiciels".

Enfin le laboratoire est partenaire de l'institut Farman, institut pluridisciplinaire réunissant cinq laboratoires de l'Ecole Normale Supérieure de Cachan.

Ce laboratoire met en place des essais asservis de façon collaborative par des mesures et des calculs de structures virtuelles. Ces essais ont pour but de solliciter le matériau pour :
- déterminer les propriétés avec la meilleure sensibilité,
- se rapprocher de chargements réalistes (qui doivent dépendre de façon bilatérale de la réponse observée et d'un environnement à simuler).

Pour mener à bien ce type d'expérience, il est nécessaire de maîtriser les asservissements, en prenant en compte la variabilité de réponse des pièces à tester, ainsi que la variabilité des temps de réaction des procédures de calcul numérique (méthode des éléments finis, corrélation d'images issues de l'expérience, etc).

L'application visée concernera la ruine de structures de génie civil. L'observation montre que dans bien des cas, c'est la rupture d'un nombre limité d'éléments bien identifiés (une poutre,

un poteau, un linteau, etc) qui est à l'origine de la ruine complète de la structure : l'élément subit un endommagement important, le reste de la structure restant quasiment élastique. Pour étudier le comportement de la structure sans simplifications abusives (lois de comportement trop approximatives, etc), on peut alors utiliser une technique de sous- structuration : l'élément sensible est testé expérimentalement, le reste de la structure est modélisé par exemple dans un code élément fini. Un dialogue permanent entre l'essai et la simulation est alors nécessaire pour rendre compte des bonnes conditions aux limites.

On se focalisera ici sur l'étude d'une structure élancée, chargée en tête, soumise à une sollicitation de type sismique en pied. L'équation de mouvement sera résolue par un code de calcul, la raideur de la structure sera obtenue et actualisée par un essai. L'objectif sera de mettre en place la rétroaction la plus réactive pour parvenir à capter les effets dynamiques.

Pour atteindre ces objectifs, il sera donc nécessaire de résoudre un certain nombre de problèmes. Dans l'état actuel des compétences du laboratoire, la commande des machines consititue un premier problème. Il est en effet impossible, avec l'interface logicielle fournie par le constructeur de réaliser une sollicitation sismique sur une structure.

Heureusement, il existe la possibilité de commander la machine par un signal analogique extérieur. Cette solution implique donc de réaliser une plateforme de commande et de contrôle de la machine. La constitution de cette plateforme sera décrite dans la première partie de ce rapport.

Une fois cette plateforme de contrôle disponible, il va falloir créer un environnement permettant de l'utiliser de manière simple et pérenne. On va donc développer des classes C++ permettant d'abstraire la gestion des différentes interfaces avec le monde physique. Il va aussi falloir implémenter un ordonnanceur pour répartir les capacités de calculs entre diverses tâches comme l'acquisition, l'asservissement, la rétroaction, etc.

Ce framework et son développement seront décrits dans la deuxième partie de ce rapport.

Enfin, une fois tous les outils de base développés, l'expérimentation prévue devient possible. Dans la troisième partie de ce rapport les techniques d'asservissement envisagées seront présentées. L'étude d'une structure élancée, chargée en tête, soumise à une sollicitation de type sismique au pied sera alors décrite dans la quatrième et dernière partie de ce rapport.

Chapitre 1

Machines d'essais et interfaçage

Dans ce premier chapitre, nous décrirons les machines d'essais disponibles et leur influence sur le choix de la solution finalement utilisée pour les commander. Nous exposerons aussi l'interfaçage matériel réalisé.

Sommaire

1.1 Description générale des machines utilisées

Les machines utilisées pour réaliser l'essai central du stage sont des machines permettant de réaliser des solliciations multi-axiales. Elles sont consitutées d'un ou plusieurs vérins hydrauliques ou électro-mécaniques qui permettent d'appliquer des déplacements ou des efforts dans des directions (axes) données.

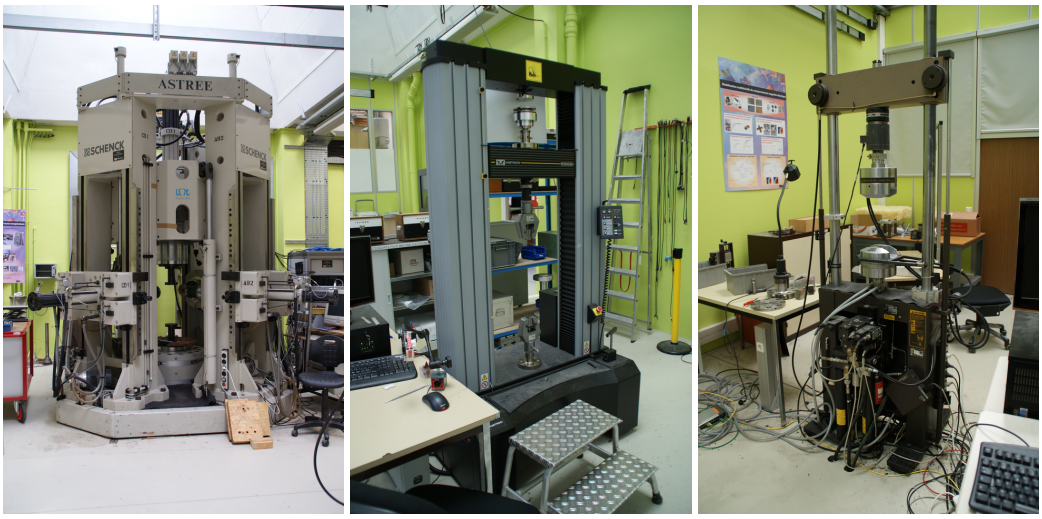


FIGURE 1.1: Quelques machines d'essais présentes au laboratoire

Les différentes machines ont alors plus ou moins d'axes d'actions avec des actionneurs plus ou moins puissants.

Les vérins électro-mécaniques des machines du laboratoire permettent de réaliser des déplacement extrêmenent précis mais ne permettent pas d'obtenir des excitations à des fréquences de plus de quelques Hertz.

On utilisera donc une machine équipée de vérins hydrauliques permettant d'atteindre les fréquences nécessaires pour la génération d'un séisme.

L'essai étant de plus uni-axial, on utilisera une machine ne comportant qu'un seul vérin.

Ceci permet de réduire le cahier des charges de l'interfaçage avec les machines. Mais à des fins de réutilisation de la plateforme développée, il sera toujours gardé à l'esprit que le système doit pouvoir être utilisé avec n'importe quelle machine.

1.2 Deux grandes catégories de machines

Le laboratoire est équipé de machines issues de deux grands fabricants : MTS et Instrom.

Chacun de ces deux fabricants proposent une interface logicielle à leurs machines fonctionnant sous Microsoft Windows. Bien évidemment, ces programmes sont fermés et on ne peut accéder qu'aux fonctionnalités proposées par leur interface graphique. Ceci permet d'avoir une interface commune pour chacune des série de machines mais qui est différente pour les deux marques.

Pour des raisons de sécurité et de secret industriel, les constructeurs ne transmettent aucune documentation sur l'électronique de commande qui permettrait de directement s'interfacer avec elle pour s'affranchir de ces logiciels.

Mais pour les deux marques il est possible de donner des ordres grace à des signaux analogiques qui vont, par exemple, servir de consigne de position ou d'effort à la machine. De tels signaux existent aussi en sortie pour renvoyer des informations telles que la position courante de la machine.

On se servira donc de ces entrées/sorties analogiques pour dialoguer avec les machines car elles sont présentes sur tous les matériels du laboratoire.

Il est à noter que l'électronique des machines Instrom peut communiquer par GPIB, ce qui permet de lui donner un certain nombre d'ordres numériques, comme les réglages de son asservissement interne par exemple. Ces capacités ne seront pas exploitées faute de temps et de manque de généralité.

1.3 L'interfaçage avec les machines

1.3.1 Solutions disponibles au laboratoire

Il s'agit donc maintenant de s'interfacer avec les entrées-sorties analogiques des machines d'essai d'une part, tout en gardant une plateforme capable de réaliser certains calculs, de mettre en place un asservissement, ainsi que de communiquer avec d'autres ressources comme le cluster de calculs ou un appareil photo.

Il nous faut donc une pateforme capable de :

– réaliser des conversions analogique-numérique et numérique-analogique,
– exécuter un programme,
– communiquer avec l'extérieur.

Explorons les solutions disponibles dans le cadre de ce stage pour répondre à ces spécifications.

1.3.1.1 Arduino

Arduino est un projet libre [1] aussi bien au niveau logiciel que matériel basé sur des microcontrôleurs AVR de Atmel.

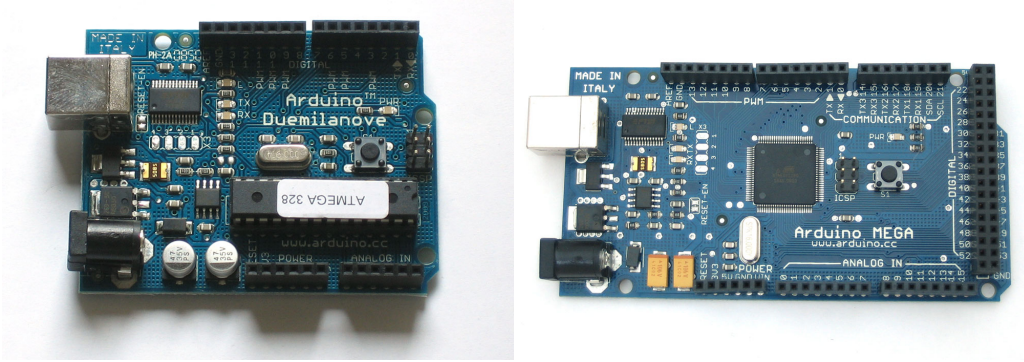
Ceci permet d'obtenir pour un cout très faible une carte possédant un certain nombre de caractéristiques intéressantes :

– possibilité de communiquer avec un hôte en USB
– entrées/sorties numériques
– entrées analogiques 10 bits
– sorties PWM [2]

Cette carte permettrait alors d'héberger le système de contrôle mais ses entrées analogiques ont une résolution assez faible et ses sorties analogiques sont des PWM, ce qui nécessiterait de rajouter un étage de filtrage en sortie de la carte.

Moyennant développement, ces limitations pourraient être levées par l'utilisation de boucliers. En effet, cette carte possède un connecteur permettant d'ajouter des modules. Il existe déjà un

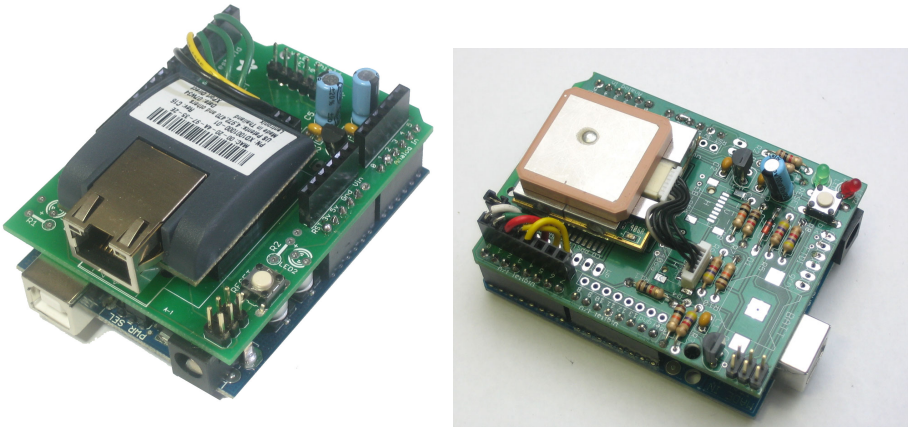
1. `http://www.arduino.cc/`.
2. Pulse Width Modulation ou Modulation à Largeur d'Impulsions



(a) Arduino Duemilanove (b) Arduino Mega

FIGURE 1.2: Les cartes Arduino

grand nombre de modules extrêmement variés comme par exemple un module permettant les communications par ethernet ou la lecture de la position GPS (voir Fig.1.3). On pourrait de même réaliser un module hébergeant des CAN [3] et des CNA [4].



(a) Bouclier Ethernet (b) Bouclier GPS

FIGURE 1.3: Exemples de boucliers d'extension pour cartes Arduino

1.3.1.2 USB-DUX

Cette carte [5] est une carte d'acquisition et de pilotage uniquement qui permet d'obtenir des entrées et des sorties analogiques 12 bits convenables.

Elle communique avec un hôte grace à une liaison USB et a déjà été utilisée par le laboratoire pour acquérir des données durant un essai.

Elle utilise de plus le driver Comedi [6], plateforme libre fonctionnant sous environnement linux.

3. Convertisseur Analogique-Numérique
4. Convertisseur Numérique-Analogique
5. `http://www.linux-usb-daq.co.uk/`.
6. linux COntrol and MEasurement Device Interface `http://www.comedi.org/`.

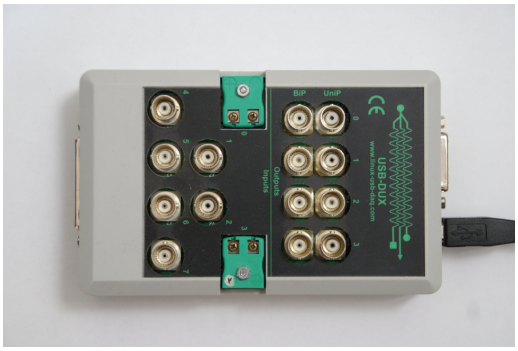


FIGURE 1.4: La carte USB-DUX

1.3.1.3 Artila iPAC 5070

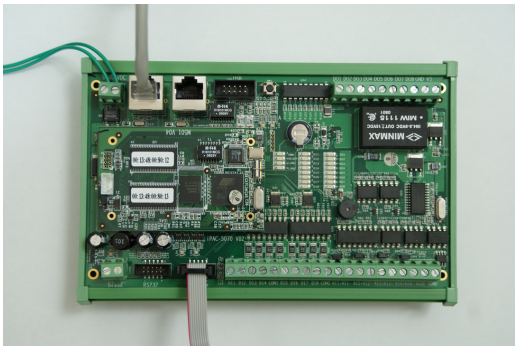


FIGURE 1.5: La carte Artila iPAC 5070

Cette carte [7] est basée sur un microprocesseur ARM fabriqué par Atmel. Elle embarque un système d'exploitation linux permettant un développement aisé des applications. De plus elle possède deux ports ethernet ce qui permet d'isoler physiquement les cartes d'acquisition du reste du réseau si l'on décide de les interfacer en ethernet.

On peut donc utiliser le protocole ssh pour donner des ordres ou lancer des applications qui s'exécuteront directement sur la carte.

Cette dernière posséde de plus un certain nombre d'entrées/sortie comme :

- des entrées numériques opto-isolées
- des entrées analogiques 16 bits avec une fréquence d'échantillonnage de 10 Hz
- des sorties numériques sur Darlington permettant de fortes tensions de sortie
- deux interfaces ethernet
- trois liaisons série
- deux ports USB

Cette carte est donc une très bonne candidate pour le contrôle du système, mais ses entrées/sorties sont trop lentes pour pouvoir être utilisées pour commander une machine. La présence des ports série, USB et ethernet permet par contre de s'interfacer avec la totalité des autres cartes envisagées.

Le fabricant fournit de plus la chaîne de compilation croisée qui permet de compiler des pro-

7. http://www.artila.com/.

grammes sous un système linux normal vers l'architecture ARM de la carte, rendant le développement des plus faciles.

1.3.1.4 Labjack UE9-pro

FIGURE 1.6: La carte Labjack UE9-pro

La labjack UE9-pro [8] est une carte d'acquisition possédant de nombreux atouts. Elle peut être reliée à un hôte en USB ou en ethernet. Elle possède des sorties analogiques 12 bits et des entrées analogiques de précision variable entre 12 et 24 bits.

Elle possède de plus un mode streaming permettant d'atteindre, avec une précision de 12 bits, des fréquences d'échantillonage de 50 kHz.

Le fabricant réalise aussi un certain nombre de cartes d'extensions permettant d'ajouter des fonctionnalités à la carte comme de nouveaux CNA avec un dynamique de sortie plus étendue.

L'utilisation de la carte est aussi assez simple car le fabricant fournit une documentation complète des protocoles de communication avec la carte ainsi qu'une bibliothèque pour la communication USB [9] ainsi que pour la communication ethernet.

1.3.2 Solution retenue

On trouvera en table 1.1 les avantages et inconvénients de chacune des cartes.

Carte	Avantages	Inconvénients
Arduino	projet libre possibilité d'inclure du microcode	faible résolution des entrées sorties PWM
USB-DUX	capacités d'acquisition et de génération correctes	nécessite un driver pour fonctionner
Artila iPAC	noyau Linux embarqué communication ethernet, USB et série	architecture ARM entrées-sorties trop lentes
Labjack UE9	capacités d'acquisition et de génération correctes Communications ethernet ou USB	

TABLE 1.1: Comparatif des différentes cartes disponibles

8. `http://www.labjack.com/`.
9. Cette bibliothèque fonctionne sous linux.

En fin de compte on retiendra deux cartes. La carte Artila iPAC 5070 sera utilisée comme plateforme de contrôle du système et la carte Labjack UE9-pro sera utilisée pour réaliser les communications analogiques avec les machines.

Ceci permet d'obtenir une plateforme simple d'utilisation car son cœur repose sur un noyau linux. Elle permet donc un accès plus facile pour des extensions futures du produit de ce stage.

Son architecture ARM pourrait par contre rendre quelque peu hasardeux la compilation du driver Comedi. Certaines sources Internet assurent sa compilation mais celle-ci n'a pas été tentée durant le stage la chaîne de compilation croisée fournie par Artila étant assez ancienne.

C'est pourquoi on utilisera de préférence la carte Labjack UE9-pro qui ne nécessite pas de bibliothèques complexes pour fonctionner. Celle-ci sera de plus interfacée en ethernet ce qui permet de réduire encore la complexité des communications avec la carte. Ceci permettrait aussi de s'affranchir de la carte Artila et de pouvoir, par exemple, faire tourner la plateforme de contrôle sur le cluster de calcul et de communiquer avec la carte d'acquisition se trouvant près de la machine d'essai, à l'autre bout du laboratoire.

Chapitre 2

Abstraction et framework de développement

Dans ce deuxième chapitre nous décrirons le framework développé de manière à abstraire les différentes cartes utilisées et permettant d'exécuter de manière simple les différentes tâches nécessaires à l'essai à réaliser.

Sommaire

Introduction

Maintenant que l'on a consituté la base matérielle permettant de controller les machines, il va falloir réfléchir à l'architecture logicielle à déployer.

Le but est de développer un framework permettant un usage simple de matériel à disposition mais pouvant être tout aussi facilement étendu pour prendre en compte du nouveau matériel.

Il va de plus falloir réaliser une couche d'abstraction permettant de mettre en œuvre différentes stratégies d'asservissement ainsi que la génération de signaux de consigne.

Dans un cas d'utilisation typique, il va être nécessaire de :
- récupérer les données en provenance d'une carte d'aquisition,
- générer un signal de consigne,
- générer la commande du système,
- envoyer des données à une carte d'acquisition,
- écrire certaines données dans un fichier pour leur exploitation future.

Il va donc être nécessaire d'exécuter plusieurs tâches périodiques (mais pas nécessairement à la même période). On va alors implémenter un ordonnanceur qui va gérer les différentes tâches permettant d'abstraire cette périodicité.

L'ensemble sera développé en C++ sous formes de classes. L'implémentation va alors être décrite dans les sections suivantes. Il en sera de même du fonctionnement général des classes ainsi que de leurs interactions. Aucune documentation détaillée des fonctions ne sera donnée ici, celle-ci étant déjà intégrée au code source fourni en Annexe.

2.1 Gestion des tâches

La gestion des tâches fût la partie la plus difficile à mettre en œuvre dans le framework. En effet plusieurs approches ont été envisagées avant de parvenir à une solution réellement efficace et simple d'emploi.

Toutes les tâches à exécuter étant périodiques, l'idée de base est de réaliser une fonction qui sera appelée par l'ordonnanceur selon la période désirée par l'utilisateur. Bien évidemment, les fonctions appelées périodiquement devront rendre la main suffisamment rapidement pour ne pas bloquer le fonctionnement des autres tâches.

2.1.1 Utilisation des threads du noyau

La première idée envisagée est l'utilisation des threads POSIX puisque la carte fonctionne grâce à un noyau linux.

Un thread ou processus léger est alors créé pour chaque tâche que l'on veut réaliser. Ces threads vont alors appeler la fonction correspondante à l'action à effectuer puis vont attendre le temps nécessaire avec d'appeler de nouveau l'exécution de la tâche.

Ceci fonctionne plutôt bien lorsque toutes les tâches sauf une comportent des fonctions bloquantes. Ces fonctions bloquantes, comme la lecture d'un paquet TCP/IP sur la liaison réseau, endorment le thread jusqu'à ce qu'elles obtiennent leur résultat (l'arrivée d'un paquet TCP/IP par exemple).

Mais lorsque plusieurs tâches doivent s'exécuter sans fonctions bloquantes, on voit apparaître des temps de latences dus aux changement de contexte du processeur lors du passage d'un thread à un autre.

Ces temps de latences sont de l'ordre de plusieurs centaines de milli-secondes sur la carte Artila et sont donc inadmissibles pour notre réalisation.

Il va donc falloir se tourner vers une autre implémentation.

2.1.2 Les timers du noyau

Pour pallier ce problème la seconde solution envisagée fût d'utiliser les timers du noyau.

En effet, le noyau linux permet de déclencher la génération d'un signal POSIX [1] envoyé périodiquement à un processus donné.

Si l'on exéctute la fonction effectuant les actions d'une tâche à chaque fois que ce signal d'alarme est reçu, on va pouvoir obtenir le comportement voulu.

Mais le signal d'alarme ne permet d'avoir qu'un seul timer par processus ce qui est assez limitant pour notre application si plusieurs tâches simultanées sont lancées à des périodes différentes.

On va donc utiliser les timers POSIX. Ceux-ci permettent d'avoir un nombre important de timers indépendants et de pouvoir choisir quel signal va être envoyé au processus périodiquement par tel ou tel timer.

On va alors se servir des signaux temps-réel POSIX. Ceux-ci sont suffisamment nombreux pour que l'on puisse créer autant de tâches que nécessaires [2].

L'utilisation de signaux va quand même avoir certaines conséquences sur le reste du code. Par exemple, on ne peut plus réaliser de pause grâce aux fonction `sleep` [3] ou `usleep` [4] car celles-ci sont interrompues lors de la réception d'un signal.

Cette solution a donc été implémentée. Pour la tester sur la carte Artila, on réalise un enregistrement de points dans un fichier. La tâche d'enregistrement comme décrite dans la section 2.5 page 23 enregistre dans un fichier texte un certain nombre de valeurs surveillées. De manière à n'observer que le comportement des timers, ces valeurs surveillées seront constantes et égales à -1.0. On peut voir un extrait des résultats dans la table 2.1.

Ces résultats mettent en évidence un problème majeur des timers noyau. En effet, la période demandée était de 2 ms et non pas 20 ms comme les résultats le montre. En effet, il s'avère que les timers du noyau ont une granularité bien différente de la précision du l'horloge du processeur. Sur les anciens noyaux linux, cette granularité est justement de 20 ms. Elle peut être réduite en changeant une constante du noyau [5]. La modification de cette constante permet d'obtenir une granularité de 1 ms [6] qui suffirait à notre application.

Le fabriquant nous ayant fourni les sources du noyau modifié utilisé sur la carte [7], nous avons donc pu le recompiler avec la constante adpatée pour réduire la granularité. Malheureusement

1. Portable Operating System Interface for uniX, famille de standards définissant les systèmes Unix
2. Leur nombre dépend de l'implémentation, sous linux il en existe 30 utilisables par l'application
3. Pause d'un certain nombre de secondes.
4. Pause d'un certain nombre de microsecondes
5. La constante HZ. Ce choix est normalement possible dans la configuration du noyau précédant sa compilation.
6. Valeur minimial
7. En accord avec la GNU Public Licence

temps (s)	valeur 1	valeur 2
0.000018	-1.000000	-1.000000
0.020098	-1.000000	-1.000000
0.040057	-1.000000	-1.000000
0.060076	-1.000000	-1.000000
0.080096	-1.000000	-1.000000
0.100116	-1.000000	-1.000000
0.120135	-1.000000	-1.000000
0.140155	-1.000000	-1.000000
0.160174	-1.000000	-1.000000
0.180194	-1.000000	-1.000000
0.200213	-1.000000	-1.000000
0.220233	-1.000000	-1.000000
0.240252	-1.000000	-1.000000
0.260272	-1.000000	-1.000000
0.280323	-1.000000	-1.000000
0.300311	-1.000000	-1.000000
0.320331	-1.000000	-1.000000
0.340350	-1.000000	-1.000000
0.360370	-1.000000	-1.000000
0.380389	-1.000000	-1.000000

TABLE 2.1: Mise en évidence de la granularité des timers noyau

la modification de cette constante empêche le noyau de démarrer correctement. Le fabricant ne pouvant apporter plus d'aide à ce sujet il a fallu trouver une autre solution.

2.1.3 Les HighRes timers

Après quelques recherches dans la documentation du noyau, il est apparu une option fort intéressante : les « High Resolution Timers ». Cette fonctionnalité dont des traces apparaissent à la version 2.6.15 du noyau, permet aux timers de s'affranchir de la constante HZ.

Les timers ont alors une granularité pouvant atteindre la nanoseconde. Malheureusement, le noyau présent sur la carte Artila est un 2.6.14, cette fonctionnalité n'est donc pas disponible.

le fabricant n'ayant fourni aucune information supplémentaire sur les modifications appliquées à son noyau, il a été impossible de compiler un noyau plus récent, la chaine de compilation croisée n'étant, de par son ancienneté, plus compatible avec des versions trop récentes du noyau linux.

Le manque de temps nous a donc poussé à écarter la carte Artila au profit d'un PC portable normal. Le fait d'avoir choisi des cartes possédant un interfaçage standard a permis d'effectuer cette transition de manière totalement transparente et sans aucune modification au code déjà écrit.

Il est à noter que tout le code développé fonctionnerait de la même manière sur la carte Artila si celle-ci si trouvait équipé d'un noyau plus récent qu'il n'est pas impossible de compiler.

On a donc un ordonnanceur permettant de lancer un certain nombre de tâches périodiques, voyons comment s'en servir.

2.1.4 Utilisation pratique

La classe gérant l'ordonnanceur s'appelle `utils::Scheduler`[8].

Au début du programme, il faut appeler la méthode statique `utils::Scheduler::init()` qui va se charger d'initialiser les paramètres de l'ordonnanceur.

Ensuite, pour ajouter une tâche, il suffit d'appeler la méthode `utils::Scheduler::addTask()`. Cette fonction prend en paramètre la fonction à exéctuer périodiquement, la période à laquelle l'exécuter ainsi qu'un paramètre optionnel à passer à cette fonction lors de son appel et le temps au bout duquel commencer à exécuter la tâche. En cas de succès, la fonction renverra alors un identifiant correspondant à la tâche qui vient d'être lancée.

La fonction exécutant les actions de la tâche devra quant à elle prendre en paramètre l'identifiant de la tâche pour laquelle elle a été appelée, le temps courant au moment de l'appel ainsi que la paramètre optionnel passé à `addTask`.

Ceci permet par exemple de passer un objet en paramètre à une tâche, comme un correcteur ou un générateur.

2.2 Interfaçage avec les cartes

2.2.1 Interface générique

2.2.1.1 Description générale

De manière à rendre le framework développé le plus générique possible, nous avons créé une classe abstraite décrivant une interface générique pour les cartes d'acquisition. Celle-ci contient un certain nombre de fonction relativement répandues parmi les différentes cartes disponibles.

Voici un extrait des fonctionnalités à implémenter :

- Gestion synchrone des entrées-sorties numériques.
- Gestion synchrone des convertisseurs numérique-analogique.
- Gestion synchrone des convertisseurs analogique-numérique.
- Gestion asynchrone des fonctionnalités synchrones.
- Gestion d'un flux de données en provenance de la carte.

La gestion asynchrone permet, si la carte le supporte, de n'envoyer qu'un seul paquet de données à la carte pour réaliser plusieurs opérations.

Le prototype de cette classe abstraite, `acquisition::Card`, peut être consulté en annexe B.1.1. Pour interfacer une carte d'acquisition avec le reste du framework, il suffit de dériver cette classe et d'écrire ses méthodes pour générer la carte à interfacer.

Si certaines fonctionnalités ne sont pas présentes sur la carte, elles doivent si possible être émulées (par exemple, les entrées-sorties asynchrones peuvent devenir synchrones) car elles peuvent être requises par d'autres classes du framework. Par exemple, les correcteurs utilisent le mode asynchrone pour communiquer avec la carte. Leur utilisation est donc impossible si ce mode n'est pas implémenté.

2.2.1.2 Communication avec la carte

L'interface propose deux méthodes pour la gestion de la communication avec la carte.

8. Voir les définitions en annexe B.4.1 et le code en annexe C.4.1

La méthode `open` doit établir la communication avec la carte et la préparer pour que les autres méthodes de communication fonctionnent.

La méthode `close` doit fermer proprement les moyens de communications avec la carte.

Le passage des paramètres de communication avec la carte (comme par exemple l'adresse IP, l'identifiant du port série, etc) devraient typiquement être donné au constructeur de l'objet carte dérivé.

2.2.1.3 Gestion des entrées-sorties synchrones

Un résumé des méthodes d'entrées-sorties synchrones se trouve table 2.2.

Méthode	Fonction
`setDAC`	Permet de définir la valeur de sortie d'un CNA
`getADC`	Permet de récupérer la valeur d'entrée d'un CAN

TABLE 2.2: Méthodes d'E/S synchrones

La fonction `setDAC` prend en paramètres une valeur de tension à appliquer ainsi que l'identifiant du convertisseur sur lequel appliquer cette tension.

La fonction `getADC` prend en paramètres un pointeur vers la variable dans laquelle écrire le résultat de la convertion, l'identifiant du convertisseur à utiliser ainsi que la précision de la convertion.

2.2.1.4 Gestion des entrées-sorties asynchrones

Un résumé des méthodes d'entrées-sorties asynchrones se trouve table 2.3.

Méthode	Fonction
`setDIOAsync`	Permet de définir le sens et la valeur des entrées/sorties numériques
`getDIOAsync`	Permet de récupérer le sens et la valeur des entrées/sorties numériques
`setDACAsync`	Permet de définir la valeur de sortie d'un CNA
`getADCAsync`	Permet de récupérer la valeur d'entrée d'un CAN depuis le buffer de réception
`planifyADC`	Permet de demander une conversion analogique - numérique lors de la prochaine opération
`sendFeedback`	Exécute toutes les opérations asynchrones en attente

TABLE 2.3: Méthodes d'E/S asynchrones

Le fonctionnement de ses méthodes est assez évident et leurs paramètres sont décrits dans le code et les en-têtes. Il est quand même bon de s'attarder sur le fonctionnement des convertisseurs analogiques-numériques en mode asynchrone.

Une conversion analogique-numérique se fait alors en trois étapes :

1. Planifier la conversion à l'aide de la méthode `planifyADC`.

2. Réaliser la conversion ainsi que les autres opérations aysnchrones en attente à l'aide de la méthode `sendFeedback`.
3. Récupérer le résultat à l'aide de la méthode `getADCAsync`.

Voici par exemple un extrait de code qui va récupérer dans la variable `tension` la valeur de l'entrée analogique `AIN0`, et placer la tension en sortie du convertisseur `DAC0` à $2,5$ V de manière asynchrone. On considère que `carte` est un pointeur vers un objet de type `Card` et que la conversion analogique-numérique sera réalisée avec un précision de 14 bits :

```
carte->planifyADC(AIN0, 14);
carte->setDACAsync(2.5, DAC0);
carte->sendFeedback();
carte->getADCAsync(&tension, AIN0);
```

2.2.1.5 Mode streaming

Un résumé des méthodes importantes pour la gestion du flux d'acquisition se trouve table 2.4.

Méthode	Fonction
`streamOpen`	Permet d'ouvrir et de configurer le flux d'acquisition
`streamStart`	Permet de démarrer le flux
`streamStop`	Permet d'arrêter le flux
`streamClose`	Permet de fermer le flux et la communication associée
`getStreamedValues`	Renvoie un pointeur vers le buffer dans lequel sont stockées les valeurs contenues dans le flux

TABLE 2.4: Méthodes de gestion de flux d'acquisition

La gestion de l'acquisition par flux peut permettre si la carte le permet d'obtenir une fréquence d'échantillonnage plus stable ainsi que de meilleures performances (en terme de fréquence d'échantillonnage maximale).

2.2.2 La carte Labjack UE9

La classe `acquisition::Labjack` hérite de la classe `acquisition::Card` et implémente la communication avec la carte Labjack UE9. Le prototype de la classe peut être consulté en annexe B.1.2 et le code associé en annexe C.1.1.

La communication avec la carte se fait en TCP/IP par une liaison ethernet. Ceci permet d'être compatible avec une grande gamme d'hôtes différents.

Il est à noter que cette implémentation utilise une bibliothèque fournie par le fabricant pour certaines opérations comme, par exemple, la conversion entre les tensions et leurs représentations binaires.

2.2.3 Autre exemple : la platine XYZ

Durant ce stage, une interface pour un autre matériel a été écrite. La classe `acquisition::PI_Serial`, dont le prototype peut être consulté en annexe B.1.3 et le code en annexe C.1.2, permet de controller une plateforme XYZ (voir figure 2.1).

FIGURE 2.1: La plateforme XYZ

Ici, la communication se fait par une liaison série RS-232. Le protocole a été déterminé en observant les trames générées par le programme de contrôle fourni par le fabricant. Il s'avère qu'il s'agit en fait d'une communication en texte clair, très facile à implémenter à partir de la documentation fournie, à l'exception de quelques commandes d'initialisation contenant des caractères particuliers.

Cette plateforme doit être utilisée pour déplacer un objectif photographique ce qui permettra de suivre la pointe d'une fissure se propageant.

2.3 Génération de signaux

2.3.1 Modèle générique de générateur

Pour faciliter la création de nouveaux générateurs, un squelette a été créé dans la classe `acquisition::Generator`. Le prototype de cette classe peut être consulté en annexe B.2.1 et le code associé en annexe C.2.1.

Cette classe implémente la logique de base d'un générateur et les mécanismes nécessaires au changement périodique de sa sortie. Il ne sera donc pas nécessaire de créer explicitement une tâche pour cela, ceci étant déjà réalisé dans le squelette.

Pour implémenter un nouveau générateur, il suffit alors de dériver cette classe et de définir la méthode `acquisition::Generator::computeOutput()` qui va être appelée périodiquement pour calculer la sortie du générateur.

Les méthodes importantes de la classe `acquisition::Generator` sont résumées table 2.5.

Méthode	Fonction
`start`	Permet de démarrer la génération du signal de sortie
`stop`	Permet d'arrêter la génération du signal de sortie
`isStarted`	Indique si le générateur est activé
`getOutput`	Renvoie la sortie courante du générateur
`getOutputP`	Renvoie un pointeur vers la sortie du générateur (pour utilisation dans l'enregistreur de valeurs, voir 2.5)

TABLE 2.5: Méthodes principales de la classe `acquisition::Generator`

2.3.2 Les générateurs implémentés

Un certain nombre de générateurs ont été implémentés pour les besoins du stage, en voici un rapide descriptif :

2.3.2.1 Les générateurs de fonction

Des générateurs de fonctions classiques ont été implémentés comme un générateur de sinusoïdes, `acquisition::SineGenerator`, et un générateur de créneaux, `acquisition::SquareGenerator`.

Il existe aussi un générateur particulier, le `acquisition::TransitionGenerator`. Celui-ci permet de générer une rampe entre deux valeurs en un temps donné, ce qui autorise des transitions lentes entre deux régimes (comme le changement du générateur créant la consigne appliquée au système).

Sa méthode `goTo` permet aussi de générer une suite de rampes laissant alors l'utilisateur commander à souhait les déplacements d'un système sans implémenter une classe de générateur particulière.

2.3.2.2 Les générateurs de signaux pré-calculés

Ces classes permettent de générer un signal dont les valeurs sont stockées dans un fichier. Chaque ligne de ce fichier doit contenir une date et une valeur.

La valeur est alors différemment interprétée selon le générateur considéré :

- `acquisition::pointsFileGenerator` l'interprétera comme la sortie à générer.
- `acquisition::int1FileGenerator` l'interprétera comme la dérivée première de la sortie à générer.
- `acquisition::int2FileGenerator` l'interprétera comme la dérivée seconde de la sortie à générer.

Les intégrations réalisées par ces générateurs utilisent la méthode des trapèzes et utilisent l'intégralité des points contenus dans le fichier, même si la période de génération du signal est plus importante que la période des points du fichiers.

2.3.2.3 Les simulateurs

Pour réaliser la rétroaction nécessaire pour ce stage, la classe `acquisition::RodGenerator` a été implémentée.

Ce générateur va réaliser les simulations nécessaires pour déterminer la consigne à appliquer au système en fonction des grandeurs mesurées et de la consigne qui lui est donnée.

Son fonctionnement sera décrit en détails plus loin dans ce rapport.

2.4 Génération de commande

La commande à appliquer au système à commander est générée en fonction de la consigne et de la position courante du système par des classes de correcteurs. Ceux-ci permettent d'implémenter diverses stratégies d'asservissement qui seront décrites dans le chapitre suivant.

2.4.1 Modèle générique de correcteur

De même que pour la création de nouveaux types de générateurs, la création de nouveaux correcteurs est facilitée par une classe squelette `control::Controller` dont la déclaration peut être consultée en annexe B.3.1 et le code associé en annexe C.3.1.

Un correcteur possède un certain nombre de paramètre :

- Une référence : c'est un générateur donnant la consigne que le système doit suivre.
- Une carte : c'est la carte d'acquisition qui permet de communiquer avec le système.
- L'identifiant du CNA relié à l'entrée de commande de la machine.
- L'identifiant du CAN relié à une sortie de la machine indiquant la valeur courante de la grandeur à asservir.
- L'identifiant optionnel du CNA relié à une entrée de la machine permettant d'afficher la consigne à suivre (et permettant d'observer les performances de l'asservissement en temps-réel sur l'interface de la machine).

La stratégie d'asservissement est quant à elle implémentée dans la méthode `Control`. C'est cette méthode qui doit être redéfinie par une classe fille pour changer cette stratégie.

La classe de base `control::Controller` n'implémente pas de stratégie particulière et ne fait que recopier la consigne sur la commande du système.

Il est à noter que la consigne attendue par le correcteur est exprimée en mm. C'est la classe `control::Controller` qui se charge de la communication avec la machine (au travers de la carte d'acquisition) et qui va convertir la position en tension et vice-versa. Ceci permet, à partir d'une tension entre 0 et $+5$ V de donner une consigne de mouvement entre -10 et $+10$ mm.

À des fins de simplifications, l'utilisation d'un correcteur est recommandé même si aucun asservisement ne doit être implémenté.

2.4.2 Correcteurs implémentés

Quelques correcteurs différents ont été implémentés et seront décrits plus en détails dans le chapitre suivant.

On peut déjà citer un correcteur PID classique, un correcteur PII constitué d'un correcteur PI surbouclé par un intégrateur ainsi qu'un correcteur PID dont les paramètres peuvent être déterminés automatiquement par un essai sur la machine.

2.5 Inscription des données dans un fichier

De manière à pouvoir exploiter ultérieurement les résultats obtenus pendant une expérience, il est indispensable de pouvoir garder une trace des grandeurs mesurées ou calculées. C'est la fonction de la classe `acquisition::Logger`.

Le prototype de la classe peut être consulté en Annexe B.1.4 et le code associé en Annexe C.1.3.

Lors de la création d'un enregistreur, il est nécessaire de lui transmettre un nom de fichier dans lequel les données vont être écrites ainsi que la fréquence à laquelle les données doivent être écrites dans le fichier.

L'écriture est alors démarrée par la méthode `start` et suspendue par la méthode `stop`.

Pour ajouter des valeurs à écrire dans le fichier, il suffit d'utiliser la méthode `addValue` qui prend en paramètre un pointeur vers la variable à enregistrer.

L'enregistreur va alors, périodiquement, écrire dans le fichier des lignes contenant le temps courant et les valeurs des variables inscrites pour enregistrement [9].

Afin d'optimiser les écritures sur le disque dur, ces lignes sont en fait enregistrées dans un buffer d'une taille donnée qui sera réellement écrit sur le disque dur lorsqu'il est plein.

9. Les valeurs sur une même ligne sont séparées par des tabulations.

Chapitre 3

Techniques d'asservissement

Dans ce troisième chapitre nous décrirons les stratégies d'asservissement envisagées ainsi que les résultats obtenus.

Sommaire

3.1 But du système

Le but est de réaliser l'asservissement d'un vérin au travers de l'interface de la machine d'essai. On va donc utiliser le framework décrit en 2.4.

Il s'agit ici de ne réaliser que la logique d'asservissement, la communication physique entre le système et la machine [1] étant transparente [2].

On va donc se concentrer sur la méthode d'asservissement à appliquer. Le processus à contrôler étant simple, on va utiliser une rétroaction PID [3].

On va de plus tenter de mettre en œuvre des méthodes permettant de régler automatiquement les correcteurs réalisés.

3.2 L'interface avec la machine

L'interface avec la machine doit tout de même être prise en compte car elle intervient sur le comportement du système apparent à commander. En effet, il est impossible de se passer de l'asservissement déjà présent dans l'interface de contrôle fournie par le fabricant.

Le système que l'on va commande va donc être composé du vérin bouclé par un correcteur PID.

De manière à pouvoir s'affranchir des différences entre les constructeurs, on ne veut pas utiliser le correcteur intégré. Ses paramètres seront placés à des valeurs arbitraires [4].

Un résumé du système complet est présenté figure 3.1.

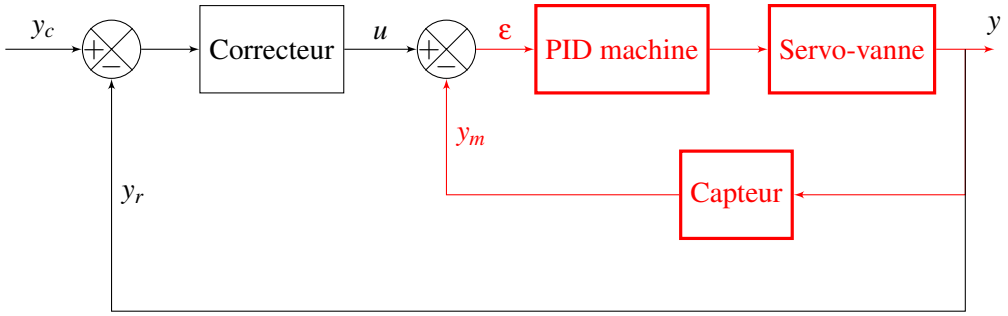


FIGURE 3.1: Système complet asservi

Sur la machine utilisée lors de ce stage, le réglage des paramètres du correcteur PID se fait manuellement dans l'interface graphique fournie par le constructeur. Il n'est donc pas possible de régler automatiquement le correcteur intégré à la machine. Par contre, ses paramètre peuvent être choisis de manière à obtenir une réponse indicielle de constante de temps apparente dont l'ordre de grandeur est choisi, stable et, par exemple, qui ne possède pas de régime transitoire oscillant.

D'autres machines du laboratoire permettent de transmettre les paramètres du PID interne par liaison GPIB, on pourrait alors envisager de régler le correcteur de la machine en utilisant, par exemple l'approche par apprentissage itératif décrit dans [2].

1. La génération de la commande et la lecture de la grandeur mesurée.
2. car gérée par le framework
3. Proportionnel, Intégrale, Dérivée.
4. qui permettent tout de même d'obtenir un système stable.

Cette méthode consiste à minimiser un critère s'exprimant en fonction des paramètres du correcteur comme, par exemple, l'intégrale de l'erreur quadratique entre le consigne et la sortie du système. Malheureusement elle n'est pas utilisable sur la machine exploitée pendant ce stage car elle peut nécessiter un nombre important d'itérations qui prendraient un temps important [5].

3.3 Première approche d'un correcteur PID

Le premier correcteur réaliser est une implémentation simple d'un correcteur PID analogique.
La valeur de la commande appliquée au système va donc être égale à la somme :
– d'un terme proportionnel à la consigne,
– d'un terme proportionnel à l'intégrale de la consigne,
– d'un terme proportionnel à la dérivée de la consigne.

Le signal obtenu est échantillonné, pour réaliser l'intégrale on va donc utiliser une intégration par la méthode des trapèzes. La dérivée sera obtenue par l'approximation de la dérivée à gauche.
Ce correcteur est implémenté dans la classe `Controll:PIDController` dont on peut voir le prototype en annexe B.3.2 et le code en annexe C.3.2.

Ce correcteur permet de contrôler correctement le système mais il faut le régler manuellement avant de pouvoir utiliser la machine, ce qui demande du temps et une certaine expérience.
On va donc réaliser un autre correcteur qui va pouvoir s'auto-régler.

3.4 Correcteur PID auto-réglé

3.4.1 Calcul de la commande

3.4.1.1 Correcteur analogique

Le correcteur étant ici réglé de manière automatique, on va affiner son fonctionnement.
Dans le domaine analogique, on va utiliser la forme parallèle du correcteur PID :

$$C(s) = K_p + \frac{K_i}{s} + K_d \frac{s}{1 + (K_d/K)s} \tag{3.1}$$

On peut remarquer que l'action dérivative est implémentée avec un filtre du premier ordre qui permet d'atténuer l'effet du bruit hautes fréquences dans la commande.
Cette forme du PID est intéressante car elle permet d'obtenir facilement un correcteur P, PI ou PID en annulant certains paramètres.

Mais pour exprimer les paramètres du correcteur, on préfère plutôt utiliser les paramètres K_p, T_i et T_d tels que :

$$T_i = \frac{K_p}{K_i} \tag{3.2}$$

$$T_d = \frac{K_d}{K_p} \tag{3.3}$$

En effet, ces paramètres ont plus de sens physique ([1]) :

5. Un opérateur devant entrer les paramètres du PID dans l'interface de la machine à chaque itération.

– T_i, temps intégral, correspond au temps nécessaire pour que la variation de la sortie du système soit égale à celle de l'entrée.
– T_d est le temps dérivatif.

Le réglage automatique du correcteur sera réalisé dans le domaine analogique et permettra d'obtenir les valeurs des paramètres K_p, K_i et K_d ou K_p, T_i et T_d à utiliser.

3.4.1.2 Passage en numérique

Pour que les résultats pratiques correspondent plus précisément aux performances attendues, on va devoir passer dans le domaine de la transformée en z pour obtenir une équation de récurrence facilement implémentable sur notre plate-forme numérique.

Pour passer de la transformée de Laplace à la transformée en z, on va utiliser l'intégration par les rectangles supérieurs.

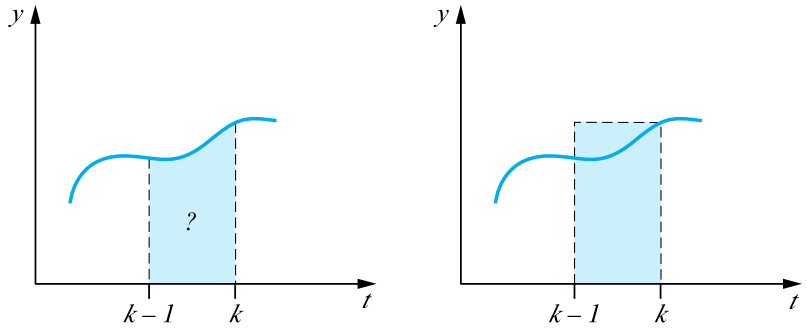


FIGURE 3.2: Intégration par les rectangles supérieurs (d'après [1])

Si l'on considère la figure 3.2, on trouve que :

$$I_k = I_{k-1} + T_e y_k, \tag{3.4}$$

où T_e est la période d'échantillonnage.

La transformée en z est alors :

$$zI(z) = I(z) + TzY(z) \tag{3.5}$$

d'où

$$I(z) = \frac{Tz}{z-1} Y(z) \tag{3.6}$$

L'équation (3.6) nous amène donc à considérer « l'équivalence » des opérateurs :

$$\frac{1}{s} = \frac{Tz}{z-1} \tag{3.7}$$

En remplaçant dans la fonction de transfert analogique de notre correcteur (équation (3.1)), on obtient sa fonction de transfert numérique :

$$C(z) = K'_p + K'_i \frac{z}{z-1} + K'_d \frac{z-1}{z-\gamma} \tag{3.8}$$

Avec, par rapport aux coefficient de la fonction de transfert numérique :

$$K'_p = K_p \tag{3.9}$$
$$K'_i = K_i T_e \tag{3.10}$$
$$K'_d = \frac{K K_d}{K T_e + K_d} \tag{3.11}$$
$$\gamma = \frac{K_d}{K T_e + K_d} \tag{3.12}$$

Si on note (u_n) la commande à appliquer au système et (e_n) la consigne, l'inversion de la transformée en z dans l'équation (3.8) permet d'obtenir l'équation de récurrence suivante :

$$u_n = (K'_p + K'_i + K'_d)e_n - \left(K'_p(1+\gamma) + \gamma K'_i + 2K'_d\right) e_{n-1} + (\gamma K'_p + K'_d)e_{n-2} + (1+\gamma)u_{n-1} - \gamma u_{n-2} \tag{3.13}$$

C'est cette équation qui est implémentée dans la classe `Control::AutoPIDController` pour calculer la commande à appliquer au système

Il reste donc à déterminer automatiquement les valeurs à attribuer à K'_p, K'_i et K'_d. (La valeur de γ étant déterminée à partir des autres paramètres pour obtenir le filtrage voulu).

3.4.2 Identification du procédé

Pour déterminer les paramètres du correcteur il faut identifier le système. Plusieurs choix s'offrent alors à nous.

3.4.2.1 Modèle du premier ordre

Le modèle paramétrique le plus simple est un permier ordre avec deux paramètres : le gain statique G_0 et l'échelle de temps τ_e (constante de temps approchée). Sa fonction de transfert est alors :

$$G(s) = \frac{G_0}{1 + s\tau_e} \tag{3.14}$$

3.4.2.2 Modèle amorti avec retard

Pour obtenir une meilleure approximation qu'avec l'équation (3.14), on peut considérer le retard apparent L et la constante de temps apparente τ. On obtient alors la fonction de transfert suivante :

$$G(s) = \frac{G_0}{1 + s\tau} e^{-sL} \tag{3.15}$$

C'est le modèle le plus utilisé pour le calcul des paramètres du régulateur PID et c'est celui-ci que nous allons utiliser.

Ce modèle possède une autre caractéristique parfois utilisée dans les méthodes de réglage, le paramètre a, représenté graphiquement par l'intersection de la tangente au point d'inflexion avec l'axe des ordonnées. Elle est liée aux autres paramètres par la relation

$$a = G_0 \frac{L}{T}$$

Les paramètres de ce modèle peuvent se déterminer graphiquement comme le montre la figure 3.3 :

On trace la tangente à la réponse indicielle en son point d'inflexion. L'intersection de cette tangente avec l'axe horizontal donne la valeur de L et l'abscisse du point d'intersection avec l'horizontale $y = G_0$ donne $L+T$. Mais cette méthode n'est pas utilisable dans le cas d'une détermination automatique.

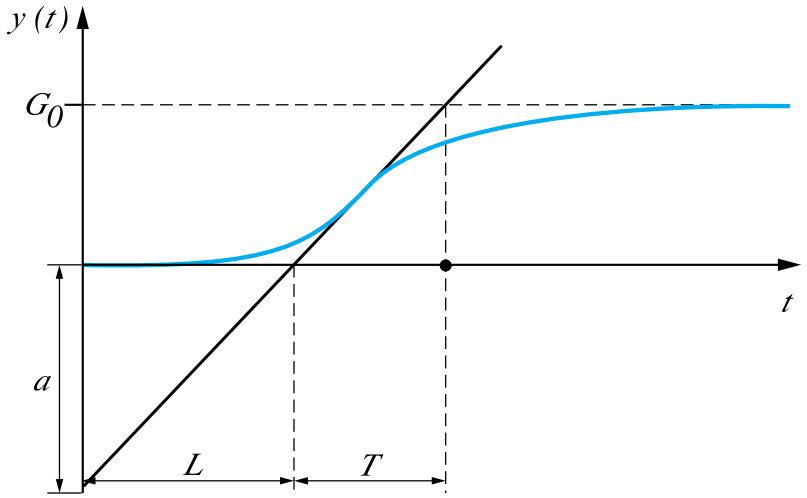


FIGURE 3.3: Estimation graphique des paramètres du modèle amorti avec retard (d'après [1])

En effet, la détermination du point d'inflexion et le tracé de la tangente peut être très imprécis sur certaines courbes. Pour éviter ces inconvénients, Broïda a étudié la position du point d'inflexion pour des système d'ordre 2 jusqu'à 6. Il propose en première approximation de calculer les paramètres du modèle (3.15) en déterminant les abscisses de temps t_1 et t_2 pour lesquelles la réponse du procédé a atteint respectivement 28% et 40% de la valeur finale (voir figure 3.4).

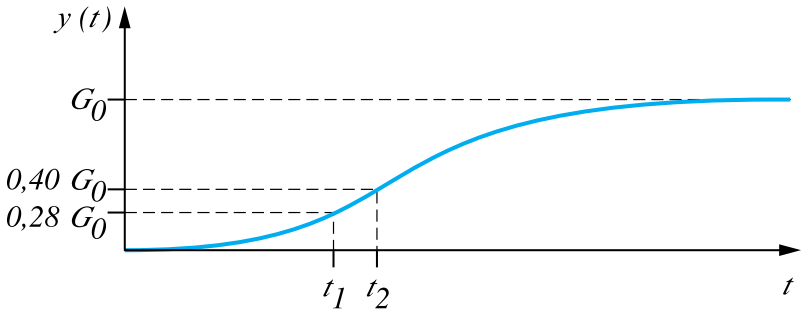


FIGURE 3.4: Estimation par le modèle de Broïda (d'après [1])

Avec ces valeurs, [3] nous donne :

$$T = 5,5(t_2 - t_1) \tag{3.16}$$

$$L = 2,8t_1 - 1,8t_2 \tag{3.17}$$

On peut donc déterminer les paramètres du modèle de manière très simple. C'est ce que réalise la méthode `GetBroidaModel()` de la classe `Control::AutoPIDController`.

Cette méthode envoie un premier échelon au système pour mesurer son gain statique. Plusieures mesures sont réalisées de manière à améliorer la détermination de G_0.

Elle envoie ensuite un second échelon pour mesurer t_1 et t_2. Pour améliorer la précision sur la mesure de ces grandeurs, on approxime la courbe par une droite autour des points intéressant : prenons l'exemple de t_1. On cherche tout d'abord le dernier point tel que la sortie n'a pas encore dépassé 27% de la valeur finale. On retient alors la valeur de la sortie et le temps correspondant. On cherche ensuite le premier moment pour lequel la sortie est supérieure à 29% de la valeur finale. On retient alors la valeur de la sortie et le temps correspondant.

Ces deux points nous permettent d'approximer la réponse du système par une droite au voisinnage de 28% de sa valeur finale et de déterminer plus précisément t_1.

On fait de même pour t_2 et on peut alors déduire T, L et a. On pourrait réaliser plusieurs échelons pour augmenter la précision de la mesure de t_1 et t_2. Il a été choisi de n'en faire qu'un seul pour obtenir un réglage plus rapide.

3.4.3 Réglage du correcteur

Détaillons maintenant les différentes méthodes permettant de déterminer les paramètres du correcteur PID analogique à partir de notre connaissance du procédé à commander.

3.4.3.1 Méthodes basées sur la réponse fréquentielle

Un certain de nombre de méthodes de réglage reposent sur l'étude de la réponse fréquentielle du système.

En particulier, ces méthodes cherchent à amener le système à des oscillations limites, en boucle fermée pour déterminer des points intéressants du diagramme fréquentiel du procédé.

On peut citer par exemple la méthode de Ziegler-Nichols, la méthode de Strejc ou encore l'expérience du relais en boucle fermée ([4], [5] et [6]) qui permet d'obtenir facilement n'importe quel point du diagramme de bode du système.

Malheureusement le principe même de ces méthodes est gênant pour notre application car elles nécessitent de mettre le système en oscillations libres, ce qui peut être dangereux sur certaines machines.

Ces méthodes ne seront donc pas utilisées.

3.4.3.2 Méthodes basées sur le modèle amorti avec retard

Les procédés à commander étant stables, il n'y a aucun problème à faire un essai en boucle ouverte. On peut donc utiliser des méthodes basées sur le modèle (3.15) dont l'identification automatique des paramètres a été traitée en 3.4.2.2.

Méthode de Ziegler-Nichols temporelle

La méthode temporelle de Ziegler-Nichols est justement basée sur ce modèle. Les paramètres du correcteur sont donnés dans la table 3.1 en fonction des paramètres a et L du procédé. On peut voir les réponses obtenues pour les correcteurs P, PI et PID sur la figure 3.5

Régulateur	K_p	T_i	T_d
P	$1/a$		
PI	$0,9/a$	$3L$	
PID	$1,2/a$	$2L$	$L/2$

TABLE 3.1: Paramètres du régulateur PID obtenus par la méthode de Ziegler-Nichols temporelle

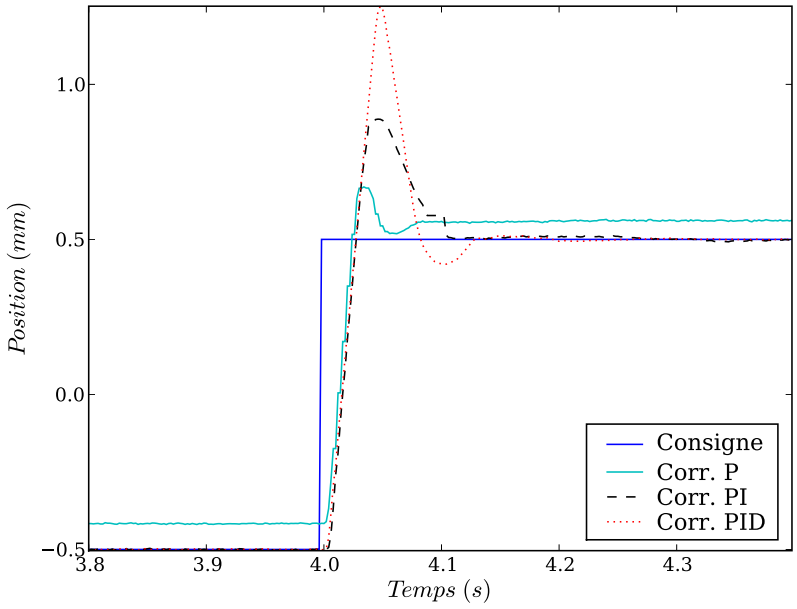


FIGURE 3.5: Résultats obtenus en boucle fermée par la méthode de Ziegler-Nichols temporelle

Méthode de Chien-Hrones-Reswick

Cette méthode présente une amélioration par rapport à la méthode précédente.

En effet elle permet d'obtenir des systèmes plus amortis en boucle fermée, le critère étant un dépassement de 0% ou de 20%. Ce qui est d'autant plus intéressant que certaines expérimentations peuvent nécessiter que le système n'ait pas de dépassement (pour éviter la déterioration anticipée d'un échantillon par exemple).

Les paramètres du correcteur sont calculés pour le rejet de perturbation (table 3.2, résultats figure 3.6) et pour le suivi de consigne (table 3.3, résultats figure 3.7)

Méthode de Cohen-Coon

Cette méthode est toujours basée sur le modèle (3.15). Elle cherche à rejeter les pertubations.

Les paramètres du régulateur PID ont été déduits par des calculs analytiques et numériques et sont exprimés en fonction des caractéristiques du procédé $a = G_0 L/T$ et $\tau = L/(L+T)$.

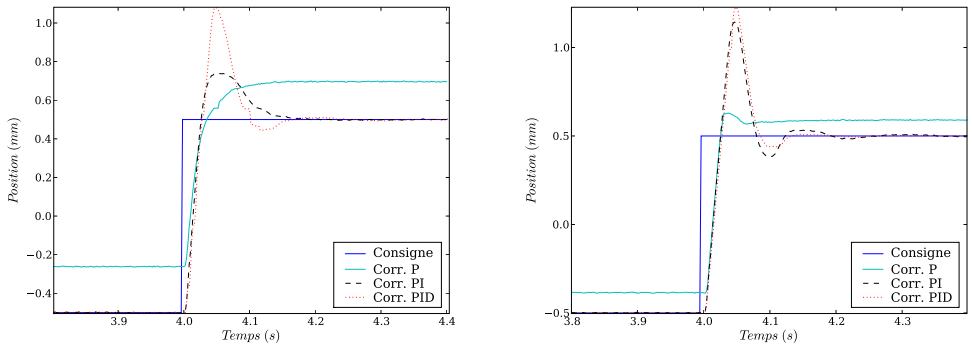
Malheureusement cette méthode donne en général un coefficient d'amortissement ξ trop faible et donc une boucle fermée mal amortie.

Dépassement	0%				20%		
Régulateur	K_p	T_i	T_d		K_p	T_i	T_d
P	$0,3/a$				$0,7/a$		
PI	$0,6/a$	$4L$			$0,7/a$	$2,3L$	
PID	$0,95/a$	$2,4L$	$0,42L$		$1,2/a$	$2L$	$0,42L$

TABLE 3.2: Paramètres du régulateur PID obtenus par la méthode de Chien-Hrones-Reswick pour le rejet de perturbations

Dépassement	0%				20%		
Régulateur	K_p	T_i	T_d		K_p	T_i	T_d
P	$0,3/a$				$0,7/a$		
PI	$0,35/a$	$1,2T$			$0,6/a$	T	
PID	$0,6/a$	T	$0,5L$		$0,95/a$	$1,4T$	$0,47L$

TABLE 3.3: Paramètres du régulateur PID obtenus par la méthode de Chien-Hrones-Reswick pour le suivi de consigne

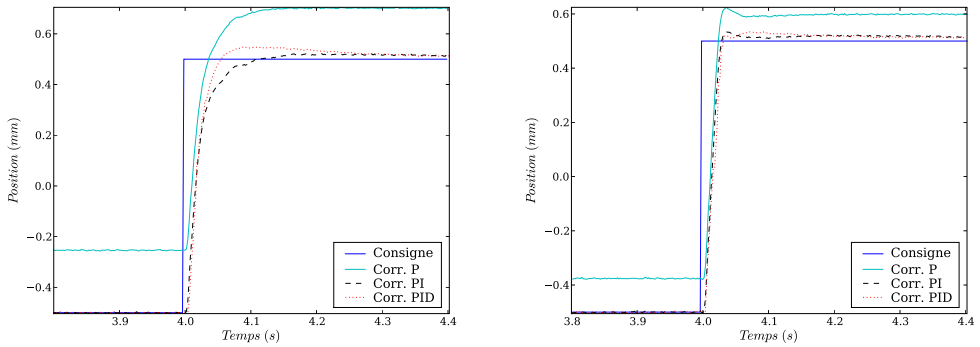


(a) 0% de dépassement (b) 20% de dépassement

FIGURE 3.6: Résultats obtenus en boucle fermée par la méthode de Chien-Hrones-Reswick pour le rejet de perturbations

Régulateur	K_p	T_i	T_d
P	$\frac{1}{a}\left(1+\frac{0,35\tau}{1-\tau}\right)$		
PI	$\frac{0,9}{a}\left(1+\frac{0,92\tau}{1-\tau}\right)$	$\frac{3,3-3,0\tau}{1+1,2\tau}L$	
PD	$\frac{1,24}{a}\left(1+\frac{0,13\tau}{1-\tau}\right)$		$\frac{0,27-0,36\tau}{1-0,87\tau}L$
PID	$\frac{1,35}{a}\left(1+\frac{0,18\tau}{1-\tau}\right)$	$\frac{2,5-2,0\tau}{1-0,39\tau}L$	$\frac{0,37-0,37\tau}{1-0,81\tau}L$

TABLE 3.4: Paramètres du régulateur PID obtenus par la méthode de Cohen-Coon



(a) 0% de dépassement (b) 20% de dépassement

FIGURE 3.7: Résultats obtenus en boucle fermée par la méthode de Chien-Hrones-Reswick pour le suivi de consigne

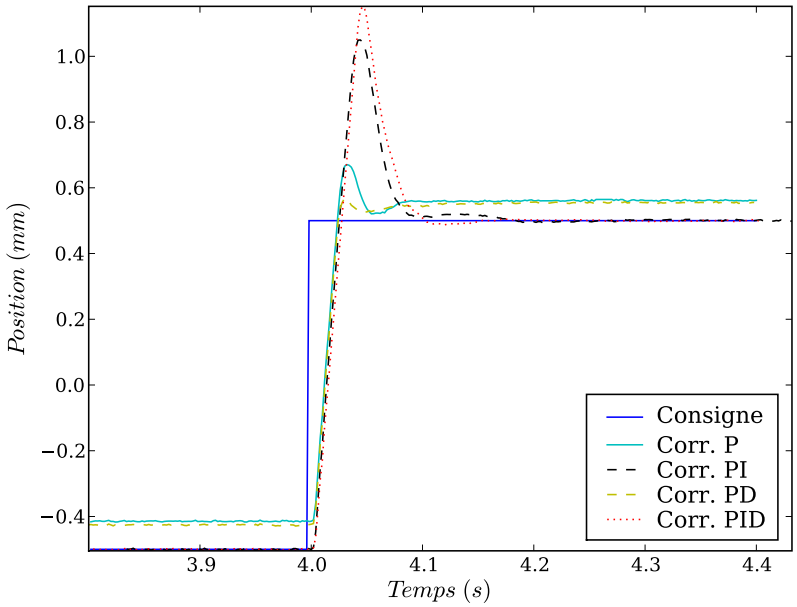


FIGURE 3.8: Résultats obtenus en boucle fermée par la méthode de Cohen-Coon

3.5 Performances et limites

On peut voir sur les figures 3.5, 3.6, 3.7 et 3.8 que le correcteur obtenu permet d'asservir le système.

Si on compare les performances obtenues avec celles du système en boucle ouverte dont la réponse est présentée figure 3.9, l'amélioration est évidente.

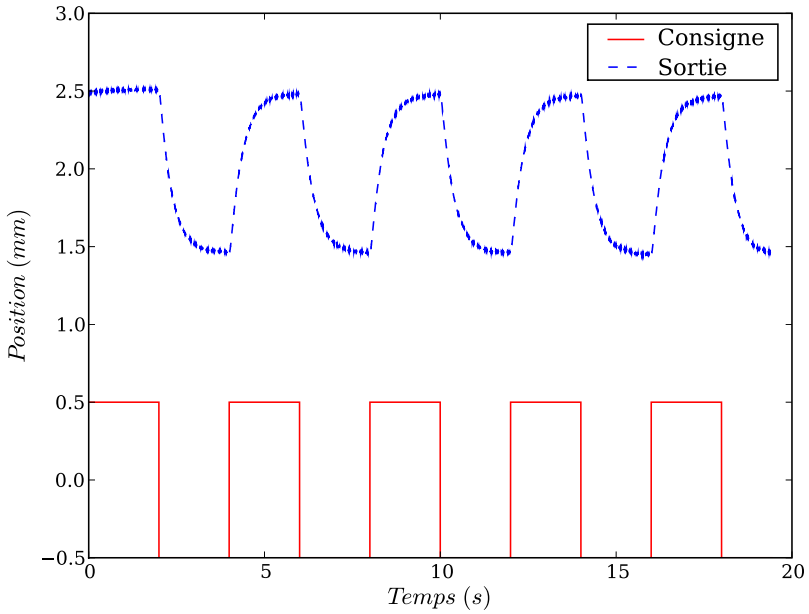


FIGURE 3.9: Réponse indicielle en boucle ouverte

Le réglage automatique du correcteur PID est donc validé, mais il comporte un certain nombre de limites.

En effet, si on regarde les résultats obtenus par auto-réglage, les performances ne correspondent pas toujours à ce qui est attendu, notamment au niveau du dépassement. Les paramètres obtenus peuvent alors être affinés manuellement mais bien souvent les essais ne comportent que des excitations « douces ». La sortie ne présentant alors pas de dépassement, un réglage supplémentaire n'est pas obligatoirement nécessaire.

Ceci peut s'expliquer par l'imprécision sur la détermination du modèle. En effet, cette mesure est limitée par la fréquence d'échantillonnage du système. On ne peut donc pas espérer avec le matériel utilisé pendant ce stage obtenir une valeur extrêmement précise de t_1 et t_2 lors de l'approximation du modèle de Broïda.

Or de faibles variations de ces paramètres entrainent des variations importantes des coefficients du correcteur.

Les performances du correcteur seront donc d'autant moins bonnes que la réponse en boucle ouverte sera rapide comme le montre la figure 3.10 pour laquelle le correcteur a été réglé par la

méthode de Chien-Hrones-Reswick pour le suivi de consigne et sans dépassement... ce qui n'est vraiment pas réalisé.

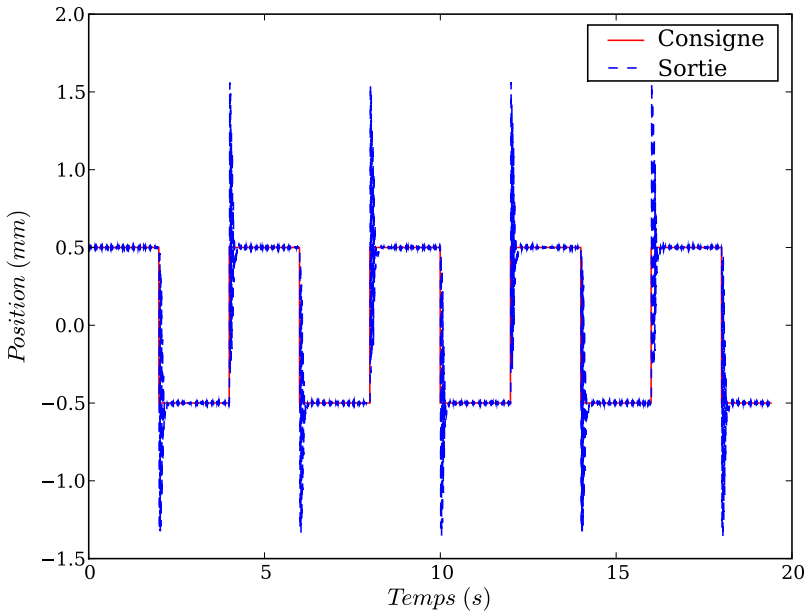
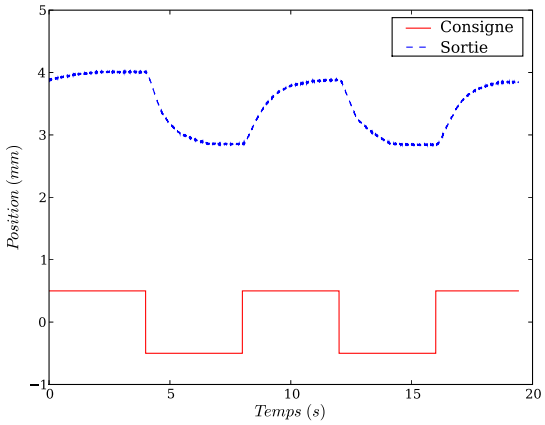


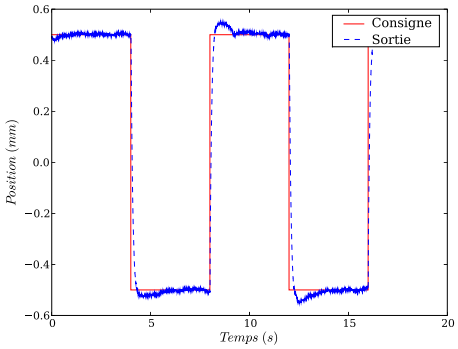
FIGURE 3.10: Résultats obtenus pour une boucle ouverte rapide par la méthode CHR en suivi de consigne sans dépassement

Au contraire, comme le montre la figure 3.11, pour une boucle ouverte très lente les performances sont bien meilleures... même si les dépassements obtenus ne sont pas tout à fait ceux attendus.

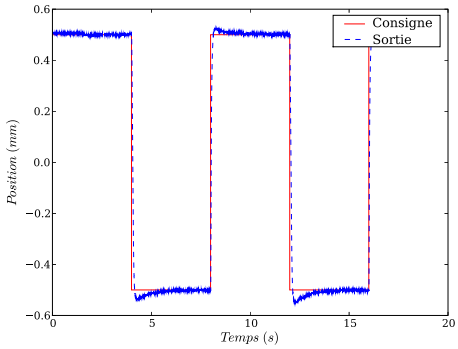
On peut aussi noter que la réponse est différente lors de la montée et de la descente du vérin ; ceci s'explique par la non-linéarité de ce dernier.



(a) Boucle ouverte



(b) Réponse par le méthode CHR en suivi avec 0% de dépassement



(c) Réponse par le méthode CHR en suivi avec 20% de dépassement

FIGURE 3.11: Résultats obtenus sur une boucle ouverte très lente

Chapitre 4

Essai dynamique

Dans ce dernier chapitre nous décrirons l'essai dynamique dont la réalisation est le but de tout ce qui a été développé précédemment.

Sommaire

4.1 Présentation

4.1.1 L'essai

L'essai que l'on va réaliser consiste à déterminer la réponse d'une structure à une excitation sismique.

On cherche à valider toute la chaîne décrite dans ce rapport, on va donc utiliser un système dont on connait une solution analytique : une poutre en bois en flexion simple.

Cette poutre est encastrée dans un mur qui subit une excitation sismique. À l'autre extrêmité de la poutre est attachée une masse ponctuelle.

La sous-structuration envisagée ici est de remplacer la masse ponctuelle, dont la mise en œuvre peut être périlleuse, par une sollicitation exercée par un vérin prenant en compte les efforts d'inertie dus à cette dite masse.

Le schéma de cette sous-structuration est présentée figure 4.1.

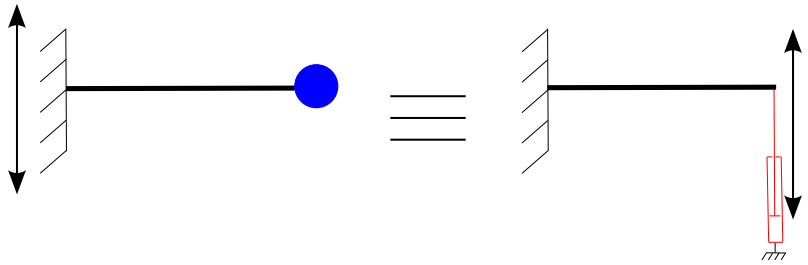


FIGURE 4.1: Schéma de l'expérience à réaliser et de l'essai équivalent

La structure va donc effectivement être soumise à la déformation réelle sans pour autant nécessiter le reste du système. Ce qui en fait un essai pseudo-dynamique. Il est aussi des cas où la structure simulée ne peut l'être en temps réel, il est nécessaire dans ce cas de faire l'essai en temps dilaté pour réaliser l'essai plus lentement qu'en réalité en introduisant une force qui va compenser les effets dynamiques de manière à ce que la simulation ait le temps de s'exécuter. Un séisme d'une dizaine de secondes peut alors prendre plusieurs heures pour être réalisé !

C'est de cette manière que ce genre d'essai est actuellement réalisé au laboratoire ([7]). Notre but est donc également de pouvoir les réaliser en temps réel.

4.1.2 Le montage

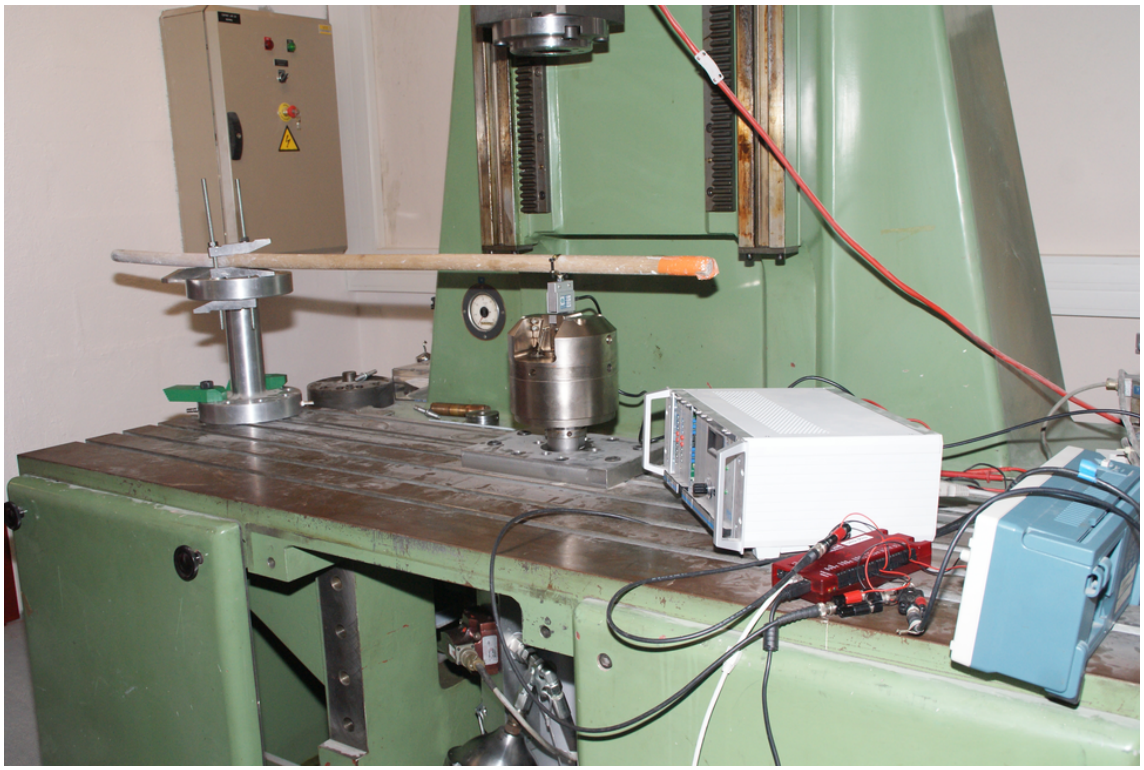
On peut voir le montage final sur la figure 4.2(a). On peut y voir l'encastrement, présenté plus en détail figure 4.2(b), la poutre en bois, et la fixation au vérin au travers du capteur d'effort présentée figure 4.2(c).

On peut également y voir l'étage électronique permettant de traiter le signal émis par le capteur d'effort pour le rendre utilisable par notre carte d'acquisition.

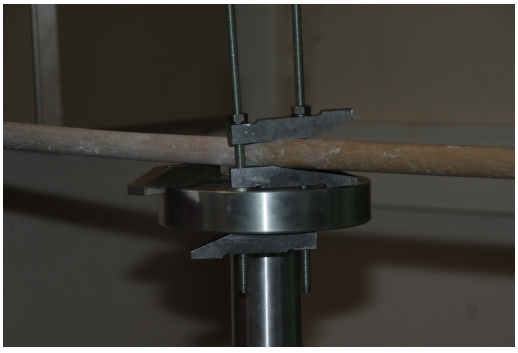
On peut aussi noter la présence d'une cale apparemment inutile à l'arrière de la liaison encastrement entre la poutre et le bati de la machine. Celle-ci permet en fait d'éviter le mouvement de la partie de la poutre située à « l'extérieur » de l'expérience et améliore l'idéalité de l'encastrement ainsi réalisé.

La figure 4.3 récapitule la chaine complète du montage. Le système de contrôle décrit dans les chapitres précédents va donc générer la tension u_{cmd} en fonction de l'excitation sismique et de la tension u_{eff} lue depuis la machine. (La machine nous renvoie aussi la position réelle du vérin, ce qui permet de réaliser l'asservissement, à des fins de clarté, ce retour n'a pas été représenté sur la figure 4.3.)

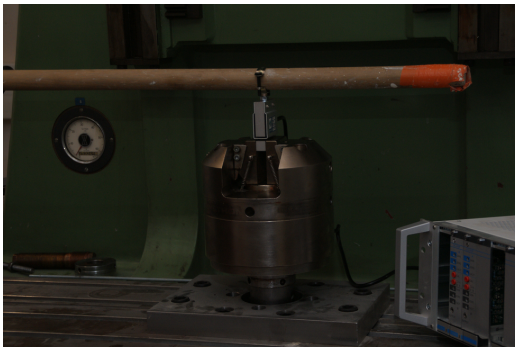
Il est important que les liaisons réalisées dans le montage le soient avec soin. En effet, de petites imprécisions peuvent introduire des termes physiques les éloignant beaucoup de leur modèle idéal. Le système aurait alors un comportement très différent du modèle utilisé pour l'expérience.



(a) Montage complet



(b) Encastrement



(c) Capteur d'effort

FIGURE 4.2: Montage expérimental

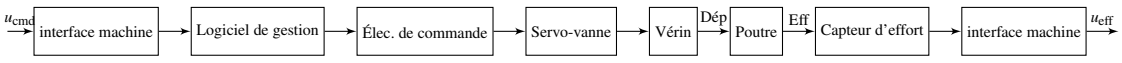


FIGURE 4.3: Chaine complète du montage

4.2 Première étape : génération du séisme

La première étape pour réaliser l'essai est d'être capable de générer l'excitation sismique sur la poutre. Pour cela on peut utiliser les classes `Acquisition::PointsFileGenerator` ou `Acquisition::Int2FileGenerator` qui permettent de lire les déplacements ou les accélérations à appliquer au vérin dans le temps.

On obtient alors les déplacements du vérin représentés sur la figure 4.4 qui correspondent bien au séisme que l'on attendait. On peut donc générer un séisme en temps-réel et en amplitude réelle sur la machine d'essai.

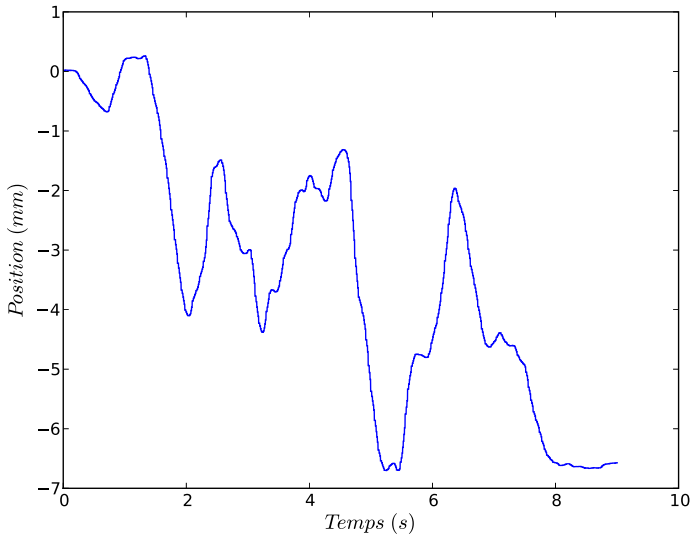


FIGURE 4.4: Génération d'une excitation sismique : Déplacement de l'extrêmité du vérin

4.3 Deuxième étape : intégration de la simulation

Il va ensuite falloir prendre en compte l'effort exercé par la poutre sur le vérin ainsi que les équations de la dynamique traduisant le déplacement de la poutre en fonction de l'accélération du séisme et de la masse normalement présente à son extrêmité.

4.3.1 Solution envisagée pour l'intégration au framework

Pour cela, on réalise la classe de générateur `Acquisition::RodGenerator`. Celle-ci est reliée à :

- un générateur qui donne les accélérations succesives du séisme,
- une carte d'acquisition qui permet de récupérer l'effort exercé par la poutre sur le vérin,
- un correcteur qui s'occupe d'appliquer la consigne générée au vérin.

C'est dans cette classe que le schéma d'intégration numérique [1] de ces équations de la dynamique est implémenté. Il doit normalement permettre d'appliquer à la poutre la déformation à

1. Le schéma utilisé est un schéma aux différences finies classique.

laquelle elle aurait été soumise dans le cas réel.

4.3.2 Problèmes de stabilité du système

Lorsque l'on réalise alors un essai avec ce générateur, on observe une divergence de la sortie même en l'abscence de séisme comme on peut l'observer sur la figure 4.5 (L'écrêtement observé sur la courbe correspond à la saturation du signal électrique de commande).

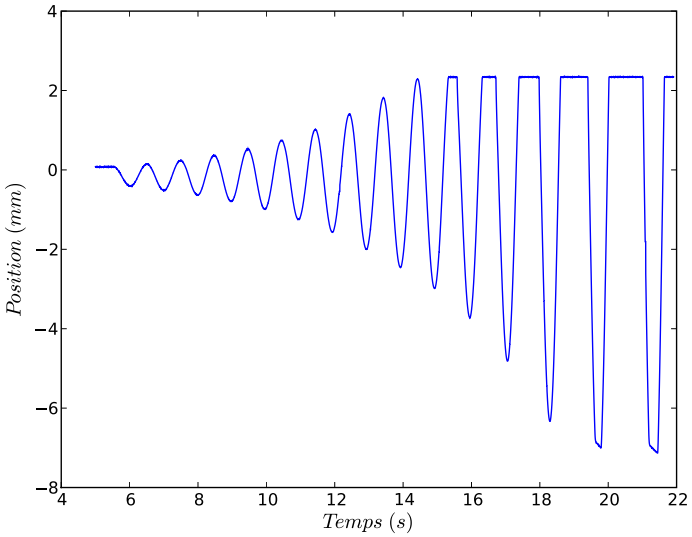


FIGURE 4.5: Divergence de la sortie avec une schéma d'intégration numérique basé sur les différences finies (masse de 100 Kg, pas de séisme).

Le schéma d'intégration numérique utilisé n'étant pas inconditionnellement stable, on va le remplacer par le schéma α-OS discuté dans [8]. Celui-ci devrait permettre plus de libertés au niveau de la fréquence d'échantillonnage puisqu'il est inconditionnellement stable. il possède de plus un facteur d'amortissement numérique qui devrait permettre d'éviter au système de diverger.

Le schéma α-OS a été développé par Hughes, Hilbert et Taylor et est très utilisé pour la simulation pseudo-dynamique [2].

Pour résoudre le problème à l'instant t_{n+1} en connaissant toutes les grandeurs du système à l'instant t_n, il repose sur deux étapes :

– Une étape prédictive :

$$\tilde{d}_{n+1} = d_n + \Delta t v_n + \frac{\Delta t^2}{2}(1-2\beta)a_n \quad (4.1)$$

$$\tilde{v}_{n+1} = v_n + \Delta t(1-\gamma)a_n \quad (4.2)$$

– Une étape corrective :

$$d_{n+1} = \tilde{d}_{n+1} + \Delta t^2 \beta a_{n+1} \quad (4.3)$$

$$v_{n+1} = \tilde{v}_{n+1} + \Delta t \gamma a_{n+1} \quad (4.4)$$

2. Pour la réalisation d'essais dont l'échelle de temps est dilatée.

Avec β et γ choisis tels que :

$$\beta = \frac{(1-\alpha)^2}{4} \tag{4.5}$$

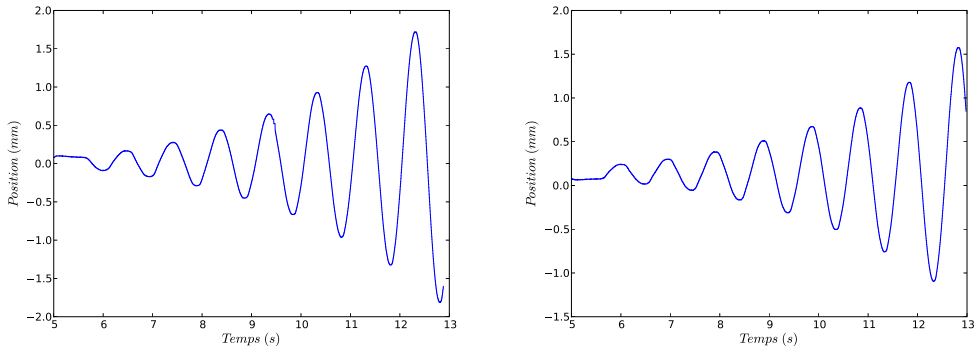
$$\gamma = \frac{1-2\alpha}{2} \tag{4.6}$$

Enfin, l'expression de a_{n+1} est déterminée à partir des caractéristiques intrinsèques du système, de l'accélération du séisme ainsi que de l'effort exercé par la poutre sur le vérin.

Pour utiliser la méthode α-OS il faut donc, après avoir déterminé les paramètres intrinsèques du système [3], réaliser à chaque pas de temps :

1. Calcul du déplacement prédictif $\tilde{d}_{n+1}$ (Éq. 4.1).
2. On impose $\tilde{d}_{n+1}$ à la structure.
3. On mesure l'effort exercé par la poutre sur le vérin $\tilde{r}_{n+1}$.
4. Correction du déplacement : étape d'*I-Modification* [4].
5. Calcul de la vitesse prédicitive $\tilde{v}_{n+1}$. (Éq. 4.2).
6. Calcul de a_{n+1}.
7. Correction de d_{n+1} et v_{n+1} (Éq. 4.3 et 4.4).

Le coefficient $\alpha \in]-\frac{1}{3};0]$ permet de spécifier un amortissement numérique ciblé sur les hautes fréquences. Mais comme le montre la figure 4.6, même en présence de ce facteur d'amortissement numérique le système diverge toujours.



(a) Amortissement numérique nul (b) Amortissement numérique maximum

FIGURE 4.6: Divergence de la sortie avec une schéma d'intégration numérique basé sur le schéma α-OS.

En fait, cette divergence n'est pas un problème numérique mais vient d'une caractéristique du système qui a été négligée. En effet, le bois est un matériau visqueux qui possède donc un certain amortissement. Ne pas le considérer dans la simulation numérique revenait alors à introduire un

3. La plupart sont choisis nuls car ils traduisent des comportements qui ne seront pas pris en compte. Seule la raideur de la poutre sera déterminée par un essai préalable.

4. Cette étape n'est pas nécessaire et ne sera pas réalisée ici

amortissement « négatif » dans le schéma numérique qui apporte de l'énergie au système et le fait diverger.

On peut introduire cet amortissement dans le schéma aux différences finies mais le terme introduit dépend de la masse située à l'extrêmité de la poutre, il faut donc réajuster ce paramètre lorsque l'on change cette masse.

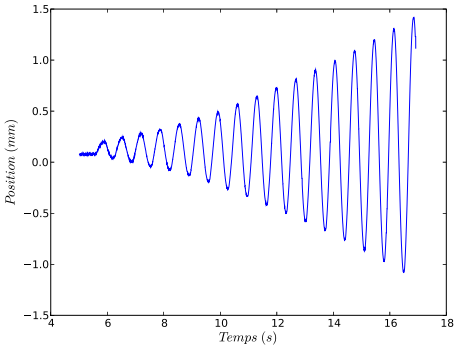
On préfère donc l'introduire dans le schéma α-OS où il apparait dans certaines des caractéristiques intrinsèques du système qui avaient été considérées nulles.

Pour régler la valeur de ce paramètre, on lui attribue une valeur importante que l'on réduit jusqu'à se trouver à la limite d'oscillations. De toutes petites oscillations ne doivent alors ni s'amortir, ni s'amplifier. Ceci garanti une valeur physiquement correcte du coefficient d'amortissement. On peut voir sur la figure 4.7 les réponses obtenues pour différentes valeurs du coefficient d'amortissement C. On retiendra $C = 120$ N.s/m dont l'ordre de grandeur est convenable par rapport à la raideur du matériau. Tous les essais suivants seront réalisés avec cette valeur du coefficient d'amortissement.

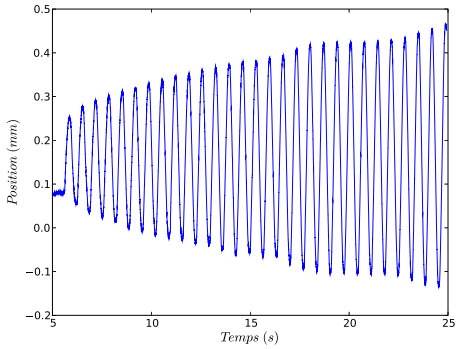
Il est à noter que ces oscillations en régime permanent proviennent d'un offset sur la mesure de l'effort exercé par la poutre sur le vérin. En effet, physiquement la position de repos du système correspond à la position pour laquelle cet effort est nul.

Si la position initiale de la poutre ne correspond pas à un effort nul, le système possède une énergie qui, si le système est réglé pour ne pas s'amortir, va engendrer des oscillations autour de la position d'équilibre de la poutre.

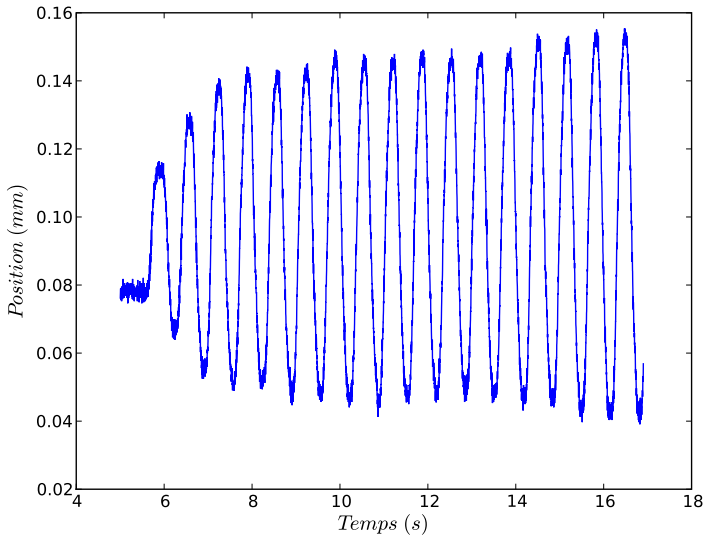
Comme le montre la figure 4.8, on peut maintenant réaliser un essai sans que le système ne diverge.



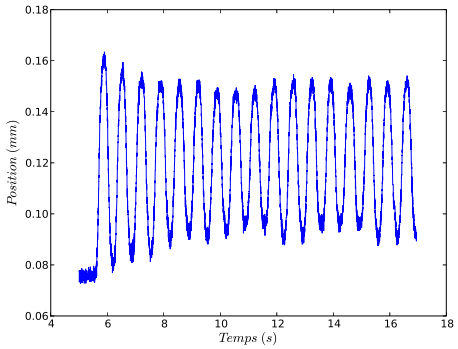
(a) $C = 50$ N.s/m



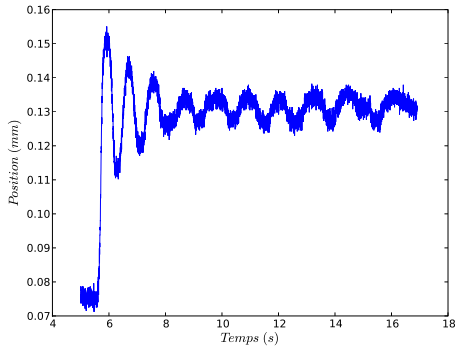
(b) $C = 80$ N.s/m



(c) Valeur retenue : $C = 120$ N.s/m



(d) $C = 200$ N.s/m



(e) $C = 500$ N.s/m

FIGURE 4.7: Réponses obtenues sans séisme pour différentes valeurs du coefficient d'amortissement C (et une masse de 50 Kg).

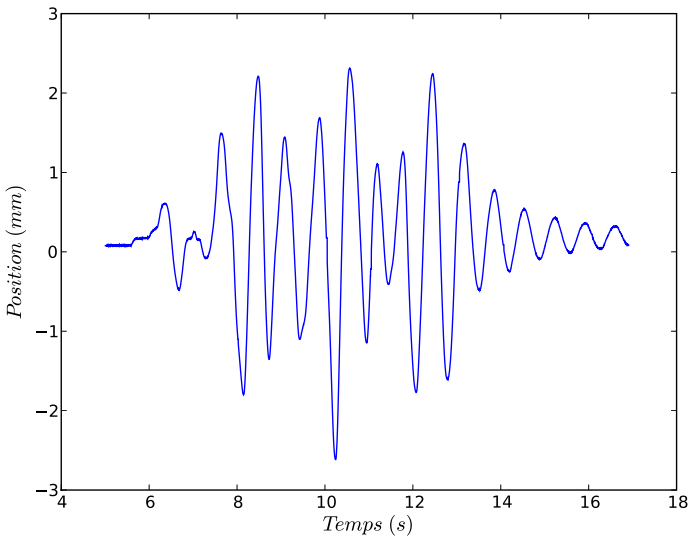


FIGURE 4.8: Essai réalisé avec la méthode α et une masse de 50 Kg

4.3.3 Problème de comportement de la carte d'acquisition

On observant les enregistrements expérimentaux on s'apperçoit que les valeurs mesurées par la carte d'acquisition ne changent que toutes les 10 millisecondes alors que la fréquence de streaming est réglée de manière à obtenir de nouveaux échantillons toutes les millisecondes [5].

L'enregistrement présenté en table 4.1 montre un fonctionnement non anticipé (et non documenté ...) de la carte Labjack cause de ce problème.

temps (s)	valeur 1	valeur 2
0.009596	2.500000	3.341325
0.009626	2.500000	3.341325
0.009638	2.500000	3.343820
0.009649	2.500000	3.342572
0.009660	2.500000	3.342572
0.009672	2.500000	3.341325
0.009683	2.500000	3.343820
0.009697	2.500000	3.342572
0.009709	2.500000	3.342572
0.009722	2.500000	3.341325
0.019364	2.500000	3.342572
0.019394	2.500000	3.341325
0.019406	2.500000	3.341325
0.019417	2.500000	3.341325
0.019428	2.500000	3.341325
0.019439	2.500000	3.341325
0.019451	2.500000	3.342572
0.019465	2.500000	3.342572
0.019477	2.500000	3.342572
0.019489	2.500000	3.342572
0.029447	2.500000	3.342572
0.029478	2.500000	3.341325
0.029489	2.500000	3.342572
0.029501	2.500000	3.340077
0.029514	2.500000	3.342572
0.029534	2.500000	3.341325
0.029545	2.500000	3.343820
0.029557	2.500000	3.341325
0.029568	2.500000	3.343820
0.029580	2.500000	3.342572

TABLE 4.1: Répartition temporelle des paquets de streaming envoyés par la carte Labjack

Pour obtenir cet enregistrement, toutes les tâches bloquantes autres que la lecture des paquets du streaming sont supprimées. De plus, on s'arrange pour qu'à chaque fois qu'un paquet soit reçu,

5. Les échantillons sont envoyés par paquets de 16, l'acquisition se faisant sur 2 voies, chaque paquet contient 8 valeurs pour chaque voie, la fréquence d'échantillonnage est donc réglée à 8 kHz pour chaque voie ce qui devrait bien donner un paquet de données toutes les 1 ms.

ses données soient écrites dans le fichier d'enregistrement.

Ce qu'il alors est important de remarquer est la répartition des instants de réception des paquets. En effet, on peut voir que l'on reçoit 10 paquets d'un coup toutes les 10 ms. Il y a donc bien un paquet toutes les millisecondes en moyenne comme attendu mais on obtient une partie de ces valeurs en retard.

Or pour pouvoir réaliser la rétroaction, on a besoin des valeurs mesurées au moment même où elles sont mesurées. La fréquence d'échantillonnage utile pour la rétroaction n'est alors plus que de 100 Hz au lieu de 1 kHz...

On va donc abandonner le mode streaming pour utiliser les fonctions d'entrées-sorties asynchrones. Celles-ci permettent des fréquences d'échantillonnage absolues maximales plus faibles que le mode streaming mais les valeurs sont récupérées toutes les 2 ms ce qui permet de multiplier par 5 la valeur de la fréquence d'échantillonnage apparente par rapport au mode streaming.

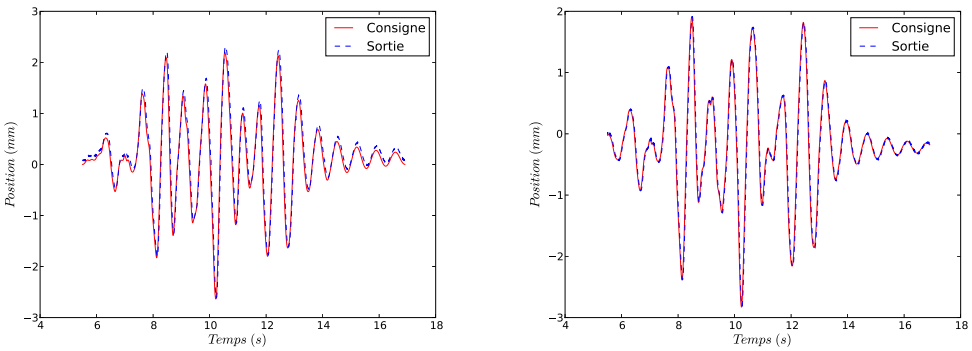
4.4 Troisième étape : intégration de l'asservissement

Dans les essais précédents, l'asservissement était laissé à la charge de la machine [6] de manière à être certains qu'il n'influence pas le schéma numérique par sa période de commande du même ordre de grandeur que la fréquence d'échantillonnage des différents signaux du système.

Pour l'ajouter, il suffit de remplacer le correcteur générique utilisé précédemment pour abstraire la communication avec la machine par le correcteur PID auto-réglé décrit dans le chapitre précédent.

L'essai commence donc par le réglage du PID, puis l'expérience en elle-même est réalisée. Comme on peut le voir sur la figure 4.9, la gestion de l'asservissement par la plateforme de contrôle [7] améliore les résultats obtenus.

Un meilleur réglage de la machine pourrait permettre d'obtenir les mêmes performances. Mais ce réglage n'est en général pas effectué dans le laboratoire. La figure 4.9(a) a en effet été réalisée avec le correcteur de la machine réglé de la même manière que pour les essais réalisés au laboratoire.



(a) Asservissement géré par la machine

(b) Asservissement par PID auto-réglé

FIGURE 4.9: Essai réalisé avec asservissement géré par la plate-forme ou la machine.

6. son PID interne avait été correctement réglé.
7. le PID interne de la machine a préalablement été déréglé.

4.5 Résultats obtenus

On peut voir sur les figures 4.10, 4.11 et 4.12 une comparaison entre les résultats expérimentaux et les résultats attendus suite à une simulation.

On remarque que les résultats concordent d'autant mieux que la masse située à l'extrêmité de la poutre est importante.

En effet, pour un masse de 10 Kg, les déplacements sont très faibles ce qui rend les résultats moins précis. Les critères des études sismiques étant les fréquences et l'amplitude des oscillations observées, l'essai avec une masse simulée de 10 Kg montre des caractéristiques correctes assez proches des résultats simulés par rapport aux études habituelles.

De plus, le modèle utilité pour les simulations est extrêmement simple et ne rend pas forcément compte de tous les comportements du système réel.

L'écart obtenu peut donc être dû à une erreur au niveau de la simulation plutôt qu'au niveau de l'expérience.

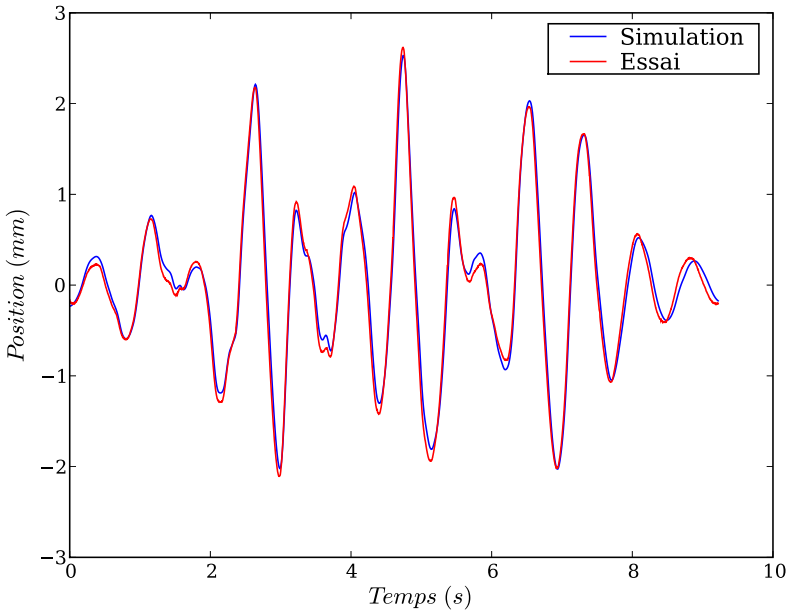


FIGURE 4.10: Comparaison entre la simulation et les résultats expérimentaux pour une masse de 50 Kg

Les résultats obtenus sont donc tout à fait corrects vis à vis des études sismiques précédemment réalisées.

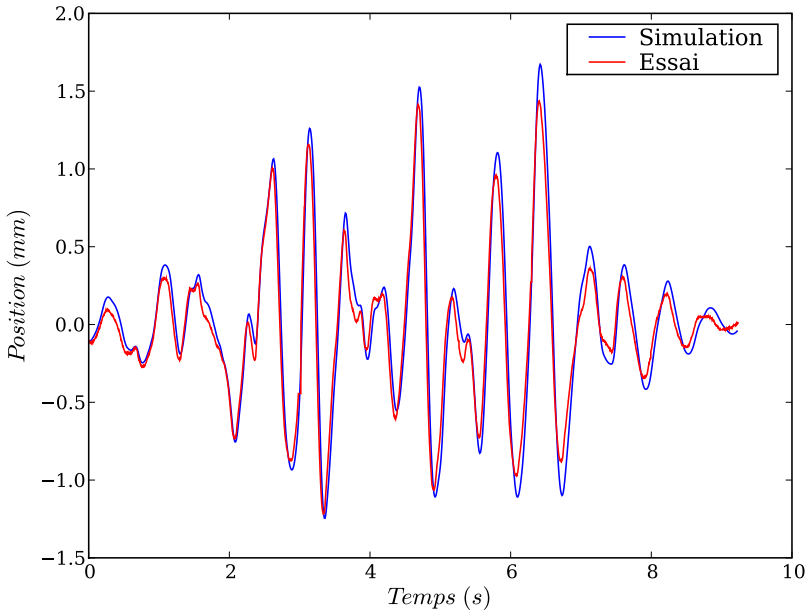


FIGURE 4.11: Comparaison entre la simulation et les résultats expérimentaux pour une masse de 30 Kg

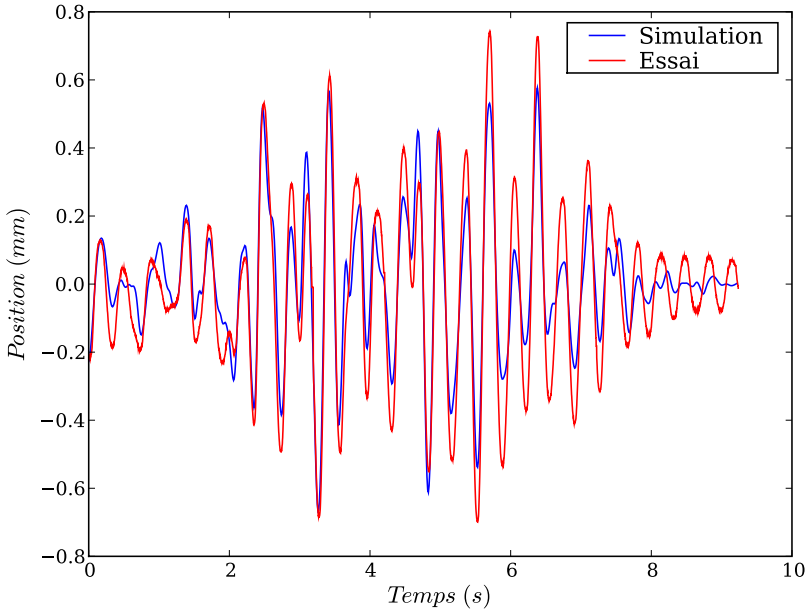


FIGURE 4.12: Comparaison entre la simulation et les résultats expérimentaux pour une masse de 10 Kg

Conclusion

En conclusion, on peut dire que les objectifs fixés en début de stage sont atteints.

En effet, l'essai dynamique de la poutre avec masse concentrée à son extrêmité est concluant, validant ainsi la plateforme de contrôle développée au cours de ces deux mois.

Ce stage a été très intéressant car il se situe à l'interface entre plusieurs domaines qui possèdent chacuns leurs spécificités propres et entre lesquels les ponts ne sont pas toujours évidents.

Il m'a permis de développer un système multitâche de contrôle complet sur des plateformes adaptées aux systèmes embarqués grâce à la flexibilité du noyau linux. Mais aussi d'employer des concepts d'asservissement plus avancés qu'habituellement avec notamment le réglage automatique des paramètres de correction ce qui permet un usage plus simple et plus générique du correcteur développé.

L'interaction entre cette partie de contrôle d'un processus physique et une partie simulation ouvre de plus des perspectives très intéressantes, notamment pour ce qui est de la commande prédictive des système, car après tout, si on peut simuler le fonctionnement du système, pourquoi ne pas s'en servir pour le commander de manière plus optimale ?

J'ai aussi pu observer les limites de tout cela ainsi que l'importance de la validité des modèles utilisés à la base du système. En effet, si ces modèles ne sont pas correctement adaptés aux procédés mis en jeu comme ce fut le cas au début de l'essai, les résultats peuvent devenir catastrophiques.

Il est aussi à noter que le système de contrôle développé durant ce stage est complètement générique. En l'état il peut être utilisé pour commander une grande diversité de systèmes possédant des entrées/sorties analogiques et ce aussi bien pour des applications fixes qu'embarquées.

Il a de plus été conçu pour pouvoir être étendu de manière simple ce qui permet avec peu d'efforts d'intégrer de nouveaux systèmes à la plateforme. On pourrait par exemple imaginer ajouter une interface I2C [8] pour permettre l'utilisation de toute une gamme de dispositifs utilisant ce mode de communication.

8. de la même manière qu'a été ajoutée l'interface pour la communication série avec la platine XYZ.

Annexe A

Programmes de démonstration

A.1 Routines de bas niveau

A.1.1 Génération d'un sinus et d'un signal de synchronisation

Ce programme montre l'utilisation des méthodes de bas niveau de la classe `Acquisition::Labjack`.

Dans certaines expériences, l'échantillon est excité en régime permanent selon une sinusoïde. Il est alors intéressant de pouvoir prendre des images de cet échantillon à différents moments du cycle. Or un appareil photo classique ne peut pas prendre une succession d'images suffisamment proche [1] pour obtenir une représentation correcte.

Pour réaliser cela, on va prendre une photo dans chaque arche de la sinusoïde mais en introduisant un déphasage entre ces déclenchements : au lieu de prendre des images à 0, T_e, $2T_e$, ... on va prendre des images à 0, $T + T_e$, $2T + 2T_e$, ... où T_e est la période apparente de prise d'images et T la période de la sinusoïde.

Le programme suivant permet de générer cette sinusoïde ainsi que le signal de synchronisation permettant de déclencher la prise d'un image par un appareil photo.

A.1.2 Génération d'une rampe

De la même manière, cet exemple permet de générer une rampe en sortie de la carte Labjack.

Un signal de synchronisation est générée quand la sortie de la carte dépasse un certan pourcentage de sa valeur maximale.

Ceci montre la facilité avec laquelle générer des évènements arbitraires même en utilisant uniquement les méthodes de bas niveau de la bibliothèque.

1. temporellement parlant.

A.2 Mode streaming pour la lecture de données

Le programme suivant montre comment acquérir des données avec le mode streaming de la carte Labjack.

Il génère un sinus en sortie de la carte et détermine les valeurs minimales et maximales vues sur une entrée analogique de la carte.

A.3 Correcteur PID auto-réglé

Le programme suivant montre comment utiliser la classe PID auto-réglé.

Il démontre l'identification des paramètres du procédé, la détermination d'un correcteur ainsi que l'utilisation de ce correcteur.

Il permet de plus de tester les différents correcteurs disponibles de manière aisée.

A.4 Programmes d'expérimentation

A.4.1 Génération d'un séisme

Le programme suivant montre comment générer un séisme sur la machine à partir d'un fichier contenant la liste des positions successives à appliquer au vérin.

Une constante définie ou non au début du fichier permet de choisir entre un correcteur PID et un correcteur PII pour l'asservissement de la machine.

A.4.2 Simulation d'une poutre

Le programme suivant met en oeuvre un générateur de simulation pour réaliser l'essai central de ce stage. Ici l'asservisement est laissé à la machine.

Son fonctionnement est décrit de manière plus détaillée dans le chapitre correspondant de ce rapport.

A.4.3 Essai dynamique complet

Le programme suivant permet de réaliser l'essai central de ce stage. Par rapport au programme précédent, celui-ci ajoute la gestion de l'asservissement. Un correcteur PID est automatiquement déterminé au début du programme pour asservir la machine qui est placée dans une configuration ou sa réponse en boucle ouverte est trop lente pour générer correctement le séisme.

Les résultats associés sont montrés dans le chapitre correspondant de ce rapport.

Annexe B

En-têtes du framework

B.1 Acquisition

B.1.1 Interface générique pour les cartes d'acquisition

Cette classe abstraite permet d'implémenter une interface commune à toutes les cartes d'acquisition utilisées. L'implémentation de touts les fonctions de cette interface garantit l'intégration d'une carte d'acquisition dans le framework.

B.1.2 Interfaçage de la carte Labjack UE9

Cette classe dérive de l'interface générique pour les cartes d'acquisition et permet d'interfaçer la carte d'acquisition Labjack utilisée durant le stage.

Elle permet la réalisation d'entrées-sorties synchrones au asynchrones ainsi que d'utiliser le mode streaming de la carte qui envoit des échantillons automatiquement.

B.1.3 Interfaçage de la plateforme XYZ

Cette classe montre que l'interfaçage des cartes d'acquisition peut englober des modes de communications très variés. Ici, la communication avec les cartes de commande de la platine XYZ se fait au travers du port série.

B.1.4 Écriture de données dans un fichier

Cette classe permet d'écrire périodiquement des données dans un fichier et ainsi de pouvoir exploiter les résultats expérimentaux

B.2 Générateurs

B.2.1 Générateur générique

Cette classe permet d'implémenter l'architecture générique d'un générateur. Pour définir un nouveau générateur, il suffit de dériver cette classe et de redéfinir la méthode `acquisition::Generator::computeOutput()` qui implémente véritablement la génération d'un signal.

B.2.2 Générateur de sinus

Cette classe implémente un générateur de sinusoïde.

B.2.3 Générateur de créneaux

Cette classe implémente un générateur de créneaux.

B.2.4 Générateur de transitions

Cette classe implémente un générateur capable de créer des rampes entre 2 valeurs.

Ceci permet de facilement faire varier une consigne entre 2 valeurs dans un temps donné de manière à, par exemple, changer de générateur de signal en entrée du système, au simplement réaliser une consigne manuellement.

B.2.5 Lecture des positions depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier.

Ceci permet de créer un signal dont les valeurs sont enregistrées dans un fichier de points.

B.2.6 Lecture des vitesses depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier. Mais contrairement au précédent générateur, celui-ci considère que le fichier indique la dérivée du signal à générer et va donc l'intégrer par la méthode des trapèzes.

B.2.7 Lecture des accélérations depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier. Mais contrairement au précédent générateur, celui-ci considère que le fichier indique la dérivée seconde du signal à générer et va donc l'intégrer deux fois par la méthode des trapèzes.

Ce générateur est très pratique pour générer des séismes à partir de leurs accélérogrammes.

B.2.8 Simulateur de poutre

Ce générateur implémente la simulation d'un poutre avec une masse concentré en bout.

C'est lui qui permet de réaliser la boucle de retour déterminant les déplacements à effectuer en fonction du séisme imposé à la structure et de l'effort réel exercé par la poutre se trouvant dans la machine de traction.

B.3 Asservissement

B.3.1 Correcteur générique

Cette classe permet d'implémenter l'architecture d'un correcteur. Pour définir un nouveau correcteur, il suffit de dériver cette classe et de redéfinir la méthode `control::Controller::Control()` qui implémente véritablement la fonction de transfert du correcteur.

Ce squelette implémente tout de même une fonction de contrôle qui ne fait que recopier la consigne sur la commande du système. Cette classe peut donc aussi être utilisée pour abstraire l'envoi d'une commande au système lorsque l'on ne veut implémenter d'asservissement.

B.3.2 Correcteur PID

Cette classe implémente un correcteur PID analogique dans lequel l'intégration est réalisée par la méthode des trapèzes et la dérivée par une dérivée à gauche.

B.3.3 Correcteur PII

Cette classe implémente un correcteur PII. Il correspond à un correcteur PI surbouclé par un autre intégrateur.

B.3.4 Correcteur PID numérique auto-réglé

Cette classe implémente un correcteur PID numérique possédant des méthodes de réglage automatique de ses paramètres.

B.4 Utilitaires

B.4.1 Ordonnanceur

B.4.2 Mesure du temps

Cette classe ne contient qu'une fonction statique permettant d'obtenir le temps courant de manière indépendante du système d'exploitation. La résolution obtenue est d'1 ms sous Microsoft Windows et d'1 μs sous linux.

Annexe C

Définitions des classes du framework

C.1 Acquisition

C.1.1 Interfaçage de la carte Labjack UE9

Cette classe dérive de l'interface générique pour les cartes d'acquisition et permet d'interfaçer la carte d'acquisition Labjack utilisée durant le stage.

Elle permet la réalisation d'entrées-sorties synchrones au asynchrones ainsi que d'utiliser le mode streaming de la carte qui envoit des échantillons automatiquement.

C.1.2 Interfaçage de la plateforme XYZ

Cette classe montre que l'interfaçage des cartes d'acquisition peut englober des modes de communications très variés. Ici, la communication avec les cartes de commande de la platine XYZ se fait au travers du port série.

C.1.3 Écriture de données dans un fichier

Cette classe permet d'écrire périodiquement des données dans un fichier et ainsi de pouvoir exploiter les résultats expérimentaux

C.2 Générateurs

C.2.1 Générateur générique

Cette classe permet d'implémenter l'architecture générique d'un générateur. Pour définir un nouveau générateur, il suffit de dériver cette classe et de redéfinir la méthode `acquisition::`

`Generator::computeOutput()` qui implémente véritablement la génération d'un signal.

C.2.2 Générateur de sinus

Cette classe implémente un générateur de sinusoïde.

C.2.3 Générateur de créneaux

Cette classe implémente un générateur de créneaux.

C.2.4 Générateur de transitions

Cette classe implémente un générateur capable de créer des rampes entre 2 valeurs.

Ceci permet de facilement faire varier une consigne entre 2 valeurs dans un temps donné de manière à, par exemple, changer de générateur de signal en entrée du système, au simplement réaliser une consigne manuellement.

C.2.5 Lecture des positions depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier.

Ceci permet de créer un signal dont les valeurs sont enregistrées dans un fichier de points.

C.2.6 Lecture des vitesses depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier. Mais contrairement au précédent générateur, celui-ci consifdère que le fichier indique la dérivée du signal à générer et va donc l'intégrer par la méthode des trapèzes.

C.2.7 Lecture des accélérations depuis un fichier

Cette classe implémente un générateur capable de lire sa sortie depuis un fichier. Mais contrairement au précédent générateur, celui-ci considère que le fichier indique la dérivée seconde du signal à générer et va donc l'intégrer deux fois par la méthode des trapèzes.

Ce générateur est très pratique pour générer des séismes à partir de leurs accélérogrammes.

C.2.8 Simulateur de poutre

Ce générateur implémente la simulation d'un poutre avec une masse concentré en bout.

C'est lui qui permet de réaliser la boucle de retour déterminant les déplacements à effectuer en fonction du séisme imposé à la structure et de l'effort réel exercé par la poutre se trouvant dans la machine de traction.

C.3 Asservissement

C.3.1 Correcteur générique

Cette classe permet d'implémenter l'architecture d'un correcteur. Pour définir un nouveau correcteur, il suffit de dériver cette classe et de redéfinir la méthode `control::Controller::Control()` qui implémente véritablement la fonction de transfert du correcteur.

Ce squelette implémente tout de même une fonction de contrôle qui ne fait que recopier la consigne sur la commande du système. Cette classe peut donc aussi être utilisée pour abstraire l'envoi d'une commande au système lorsque l'on ne veut implémenter d'asservissement.

C.3.2 Correcteur PID

Cette classe implémente un correcteur PID analogique dans lequel l'intégration est réalisée par la méthode des trapèzes et la dérivée par une dérivée à gauche.

C.3.3 Correcteur PII

Cette classe implémente un correcteur PII. Il correspond à un correcteur PI surbouclé par un autre intégrateur.

C.3.4 Correcteur PID numérique auto-réglé

Cette classe implémente un correcteur PID numérique possédant des méthodes de réglage automatique de ses paramètres.

C.4 Utilitaires

C.4.1 Ordonnanceur

Bibliographie

[1] Besançon-Voda (A.) et Gentil (S.). Régulateurs PID analogiques et numériques. *Techniques de l'ingénieur*, R7416.

[2] Tong Heng Lee Jian-Xin Xu, Sanjib K. Panda. *Real-time Iterative Learning Control : Design and Applications*. Springer.

[3] Broïda (V.). Extrapolation des Réponses Indicielles Apériodiques. *Automatisme*, XVI, 1969.

[4] Åström (K.J.) et Hägglund (T.). Automatic Tuning of PID Controllers. *ISA Research Triangle Parc*, 1988.

[5] Schmitt (L.). Sur l'autoréglage des Asservissements en Position dans les Systèmes de Commande d'Axe Industriels. *PHD thesis Institut Polytechnique de Grenoble*, 1990.

[6] Besançon-Voda (A.) et Landau (I.D.). Procédé et dispositif d'ajustement d'un régulateur PID. *French Patent*, 95/05364.

[7] Ragueneau (F.) et Desmorat (R.) Souid (A.), Delaplace (A.). Pseudodynamic testing and nonlinear substructuring of damaging structures under earthquake loading. *Engineering Structures*, 31 :1102–1110, 2009.

[8] Combescure (D.) et Pegon (P.). alpha-Operator Splitting time integration technique for pseudodynamic testing Error propagation analysis. *Soil Dynamics and Earthquake Engineering*, 16 :427–443, 1997.