UNIVERSITÉ BLAISE PASCAL

RAPPORT de Stage

pour obtenir le diplôme de

Ingénieur Génie Électrique
Master Robotique

de : **Polytech Clermont-Ferrand et UFR ST**

présentée et soutenue publiquement

par

Jessy LOUIS

le 31 juillet 2016

préparée dans l'équipe MACCS de l'axe ISPR de l'Institut Pascal :

Implementation sur FPGA de modèles robotiques

JURY :

Président du Jury	Jean-Pierre Dérutin
Tuteur Polytech	Jean-Pierre Dérutin
Tuteur de stage	Sebastien Lengagne
Responsable M2 ROB	Youcef Mezouar
Membres	Rafik Smaali
	Christophe Pasquier
	Jacques Laffont

Remerciements

Je tiens tout d'abord à remercier mon professeur et tuteur de stage M.Lengagne sans qui ce stage n'aurait pas eu lieu, mais aussi pour son soutien et son aide apportés tout au long du stage. Je souhaiterais aussi remercier M.Derutin pour son expertise et ses conseils méthodologiques qui m'ont été très utiles lors de ce stage. Je tiens à les remercier aussi pour leurs actions qui m'ont permis d'avoir le financement du LABEX. Je remercie donc particulièrement le LABEX pour son soutien financier qui a permis à ce stage d'exister. Je tiens en plus à remercier l'Institut Pascal pour son accueil au sein de ses locaux ainsi que pour m'avoir fournie le matériel nécessaire à la réalisation de ce stage.

De plus, j'aimerais remercier Sebastien Caux pour son temps et son travail qui vont me permettre de réaliser les derniers tests pratiques et ainsi validé ou non les résultats théoriques. Je tiens aussi à remercier M.Berry et M.Landrault qui m'ont donné les outils utiles à l'implantation du système. Je remercie aussi tous les professeurs de Génie électrique et du Master 2 Robotique pour les connaissances qu'ils m'ont apportés et qui m'a permis de réaliser à bien ce stage.

Pour finir, je remercie toutes les personnes qui ont cru en moi et m'ont encouragé tout au long de mon cursus et de mon stage malgré mes lacunes théoriques.

Résumé

Le but de ce stage était d'initier une nouvelle thématique de recherche concernant l'implantation de modèles robotiques sur une architecture dédiée (FPGA). L'objectif principal est d'accroître l'autonomie des robots industriels, compagnons ou d'intervention. Cette thématique s'inscrit dans deux domaines différents : la robotique et les systèmes embarqués. La première étape était de démontrer la faisabilité de ce projet. Pour ce faire, il a été décidé d'implanter le modèle géométrique direct et dynamique inverse sur FPGA pour un robot Kuka à 7 degrés de liberté. Dans ce rapport nous allons introduire certaines notions théoriques importantes à la compréhension et au développement de ce projet pour ensuite critiquer et quantifier les résultats théoriquement prévus lors de la phase de développement. Ces phases d'apprentissage et de développement ont donné des résultats fortement prometteurs, ce qui a permis de passer à la phase d'implantation et ainsi valider les résultats à l'aide d'un essais d'algorithme génétique.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Présentation du laboratoire

L'institut Pascal est une unité mixte de recherche (UMR 6602) créée en 2012. Il est placé sous la tutelle de l'Université Blaise Pascal, du CNRS (Centre National de la Recherche Scientifique) et de SIGMA. Il rassemble au sein d'une même structure plus de 300 personnes (135 enseignants-chercheurs, 3 chercheurs, 33 BIATSS/ITA, 138 doctorants) rattachées à des domaines disciplinaires relevant des Sciences pour l'Ingénieur (automatique, mécanique, électronique, génie des procédés, robotique) et des Sciences Fondamentales (physique, biochimie), réunis dans quatre axes de recherches et un programme transversal :

- **GePEB (Génie des Procédés, Energétique et Biosystèmes)** : traite de l'ingénierie des bioprocédés à plusieurs échelles avec une approche biologique, chimique et physique multidisciplinaire des processus. Elle travaille dans le domaine de la biochimie, de l'environnement et l'agro-alimentaire
- **MMS (Mécanique, Matériaux et Structures)** : a pour ambition de relever les défis scientifiques que posent les exigences industrielles dans les domaines de la mécanique et du génie civil, à travers des approches innovantes et pluridisciplinaires. Les activités de recherche couvrent un spectre très large allant de la mécanique des matériaux et structures à la conception des mécanismes et des robots, en passant par les méthodes probabilistes, les techniques de mesure sans contact et les modélisations numériques des systèmes complexes.
- **PHOTON (Photonique, Ondes, Nanomatériaux)** : rassemble physiciens et ingénieurs en électronique. Ils couvrent les thèmes de recherche concernant la Nanophotonique et Nanostructures, les Microsystèmes et nanomatériaux, et la Compatibilité électromagnétique.
- **ISPR (Image, Systèmes de Perception, Robotique)** : se concentre principalement sur les domaines de la vision par ordinateur et de la robotique. L'équipe est reconnue pour ses applications dans le domaine des véhicules autonomes. Elle dispose d'ailleurs du site PAVIN (Plateforme Auvergnate pour les Véhicules INtelligents) présenté sur la Figure 1.1.
- **Programme Transversal** : fédère des recherches interdisciplinaires de l'Institut Pascal. ll se décompose en 5 actions : Machines et robots intelligents, innovation pour les bio-procédés, matériériaux et modélisations multi-echelles, approche probabilistes, imagerie quantitative.

Ce stage se déroule dans l'axe ISPR qui comprend 4 équipes :

FIGURE 1.1 – Vue aérienne du site d'expérimentation PAVIN

— **ComSee** : ComSee s'investit les domaines de la vision par ordinateur et des solutions photogrammétriques pour la localisation 3D, la reconstruction 3D, l'étalonnage des capteurs et le suivi en temps réel depuis environ 20 ans. Il a été impliqué dans la création de plusieurs sociétés : Poseidon, DxO, ... ComSee se concentre sur deux principaux paradigmes de recherche, Image Matching et Reconstruction 3D.
— **PerSyst** : Les principaux travaux de l'équipe sont dédiés à de nouvelles approches de localisation prenant en compte à la fois des capteurs de modélisation, du SLAM multi-modalités (lidar, vision, radar, ...), du filtrage optimal, à la localisation non centralisée et multi-robots / multi-capteurs SLAM pour la robotique mobile et en pelotons et à la cartographie des scènes 3D, modélisation et compréhension afin de conduire à une meilleure compréhension des scènes dynamiques.
— **DREAM** : Cette équipe se concentre sur la recherche de pointe dans la modélisation et l'analyse architecturale, langage- approches pour les systèmes embarqués, des caméras intelligentes, middleware et la technologie multi-core . L'expertise du groupe couvre de la conception de concepts à diverses technologies de mise en œuvre pour la construction de prototypes de travail réalistes dans des domaines d'application tels que le traitement d'image et la vision par ordinateur.
— **MACCS** : Nos recherchent portent principalement sur la modélisation et le contrôle de robots mobiles et manipulateurs, la vision des robots, la vision active, la maitrise visuelle et les comportements d'anticipation. Les applications de nos recherches sont liées à la fabrication robotisée et aux systèmes de transport intelligents. L'équipe travaille sur deux thématiques : les véhicules autonomes (AV) et la modélisation, apprentissage et perception pour la commande (MAPC).

Ce stage se déroule dans l'équipe MACCS sur la thématique MAPC en collaboration avec l'équipe DREAM.

Il a été financé par le LABEX (LABoratoire d'EXcellence) IMobS3 "Innovative Mobility : Smart and Sustainable Solutions" a pour ambition de développer des briques technologiques efficientes et respectueuses de l'environnement pour une mobilité innovante des personnes, des biens et des machines en jouant sur le triptyque "Recherche – Formation – Valorisation". Ce LabEx coordonné par l'Université Blaise Pascal de Clermont-Ferrand regroupe les forces de 7 Laboratoires (240 Personnes) issus de 6 Etablissements. Il concentre ses actions autour de trois défis : véhicules et machines intelligents, services et systèmes de mobilité intelligente et procédés de production d'énergie pour la mobilité.

1.2 Motivations

Les pays développés possèdent une population de plus en plus vieillissante. La robotique de service est donc appelée à se développer dans les prochaines années. Elle aura pour but d'accroître l'autonomie des personnes à domicile en effectuant des tâches simples du quotidien ou en alertant les secours en cas de problème. L'objectif des robots, dits compagnons, sera également de diminuer le sentiment d'isolement. Tout comme les robots d'interventions, ces robots compagnons devront être autonomes du point de vue énergétique et du point de vue des capacités de calcul. A l'heure actuelle la quasi totalité des calculs nécessaires aux robots est effectuée sur des ordinateurs embarqués ou est déportée sur des ordinateurs externes pour les tâches les plus complexes. Les temps de calculs trop importants interdisent certains algorithmes de planification ou de commande d'être embarqués pour une utilisation en temps réel malgré qu'ils paraissent très prometteurs en simulation.

Nous proposons, dans le cadre de ce projet, d'amorcer une nouvelle thématique de recherche qui aura pour objectif d'effectuer les calculs « bas niveau » permettant de gérer les mouvements de base des robots (attraper un objet, se déplacer, ...) sur une architecture matérielle dédiée afin qu'ils soient plus performants. L'objectif est d'augmenter les capacités de calculs tout en conservant une consommation d'énergie équivalente voire inférieure.

1.3 Etat de l'art

Dans le domaine des robots manipulateurs et des robots humanoïdes, le temps de calcul est un facteur important. Lors de travaux précédents, il a été développé des méthodes capables de générer des mouvements dynamiques pour des systèmes robotiques complexes mais avec des temps de calcul interdisant une utilisation en temps réel. Par exemple, la génération de mouvements dynamiques multicontact pour le robot HRP-2 [10] peut prendre de quelques minutes à plusieurs jours [11]. Afin de permettre un contrôle en temps réel, certaines méthodes se basent sur des modèles réduits, notamment celle présentée dans [9] assimile les robots humanoïdes à un pendule inverse, alors que d'autres méthodes [14] se basent sur un pré-calcul hors ligne afin de réduire le temps de calcul en ligne pour assurer l'évitement de collision. Pour réduire le temps de calcul, quelques travaux [13, 1] ont trouvé des solutions dans le domaine de l'informatique en utilisant le meta programming qui permet de n'effectuer que les calculs utiles, notamment lors de la multiplication des matrices de transformation dont certaines comportent des zéros. Dans le domaine de la robotique le temps de calcul constitue toujours un problème non résolu qui ne permet pas d'implémenter des algorithmes complexes en temps réel bien qu'ils possèdent de bonnes performances en simulation.

Le domaine de la vision par ordinateur souffre aussi du problème lié au temps de calcul. Une thématique de recherche émerge depuis quelques années pour résoudre ce problème : les smart cameras [3]. Les smart cameras sont des systèmes à base de caméras qui embarquent également une architecture matérielle (électronique) dédiée afin de traiter l'information visuelle et communiquer avec d'autres systèmes environnants. L'objectif est d'avoir un traitement matériel de l'information et ne renvoyer que les infos utiles. Ces systèmes permettent d'avoir une réduction de la consommation d'énergie d'un facteur 10 à 100 pour un temps de calcul comparable voire meilleur. L'implantation d'algorithme sur une architecture dédiée a permis une réduction du temps de calcul dans le domaine de la

vision. Il apparaît intéressant d'évaluer la faisabilité et les performances de l'utilisation d'architecture dédiée dans le domaine de la robotique manipulatrice. Les algorithmes d'optimisation génétiques [16] ont déjà été implémentés sur une architecture dédiée. Le principe de ces algorithmes est de tester plusieurs candidats à la validation des critères selon certaines contraintes (candidat choisis pseudo-aléatoirement).

1.4 Application

1.4.1 Présentation

Dans notre cas, nous appliquerons nos travaux au robot Kuka LWR à 7 dégrés de liberté présenté sur la Figure 1.2.

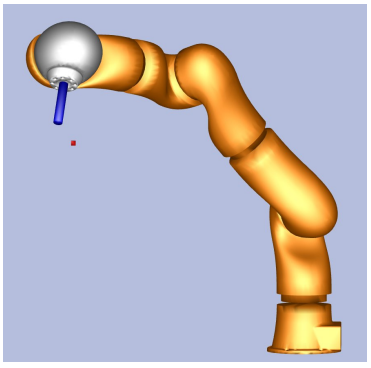

FIGURE 1.2 – Représentation du robot Kuka LWR.

L'application concernera la génération d'un mouvement reliant une position initiale à une position finale de l'organe terminal en évitant les collisions avec l'environnement, tout en minimisant l'énergie consommée.

Partant d'une configuration initiale du robot, le mouvement est généré de manière itérative en optimisant la posture suivante de manière à minimiser la distance entre l'effecteur et sa position désirée, la somme des couples au carré et la somme des accélérations au carré et de manière à respecter les contraintes de vitesse et couple articulaire maximum et les contraintes de collisions. La génération de cette posture suivante s'obtient à l'aide d'un algorithme génétique.

1.4.2 L'algorithme génétique

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné [2].

L'algorithme n'étant pas le sujet du stage mais simplement son application, nous utiliserons l'algorithme suivant. Partant d'une population *élite* composée de N_e configurations intéressantes, l'algorithme génère une population *test* composée N_t configurations à évaluer. Une fois la population *test* évaluée via un modèle, les N_e meilleurs éléments des deux populations forment la nouvelle population *élite*. Cette opération est répétée jusqu'à ce que l'un des deux critères soit validé :

— le nombre d'itération à dépassé un seuil défini,
— la distance euclidienne maximale entre deux configurations S est inférieure à un seuil donné.

La génération d'un candidat C de la population *test* se fait à partir de deux éléments E_1 et E_2 de la population *élite* suivant l'équation 1.1 :

$$C = E_1 + (E_2 - E_1)\alpha + \beta \tag{1.1}$$

Avec :
— α : nombre aléatoire entre -0.1 et 1.1
— β : vecteur aléatoire contenu dans l'ensemble défini par -0.1S et 0.1S

La figure 1.3 traduit l'équation précédente et représente l'ensemble des candidats possibles à partir de deux positions de la population *élite*.

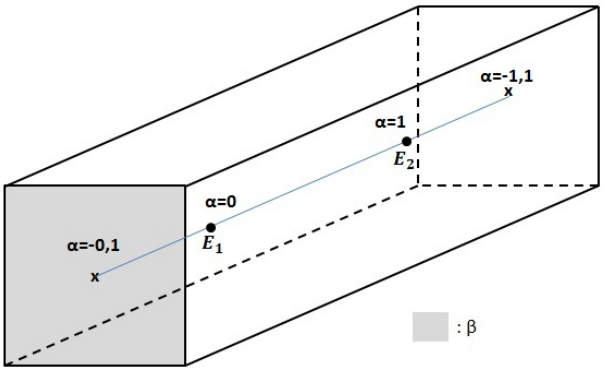

FIGURE 1.3 – Ensemble faisable pour la génération d'un nouveau candidat à partir de deux candidats E_1 et E_2.

Cet algorithme prend en compte un ensemble de contraintes à respecter $g(C) < 0$ et plusieurs critères hiérarchisés $f_i(C)$ ayant chacun un seuil δ_i. Pour déterminer si un candidat C est meilleur qu'un candidat de la population *élite* E, il doit tout d'abord satisfaire les contraintes ainsi que l'algorithme présenté sur la figure 1

Données : C,E deux candidats à évaluer
Résultat : $C < E$ ou $C > E$
Initialization : i le niveau hierarchique le plus élevé;
tant que *résultat non trouvé ou* $i \neq$ *niveau hierarchique le moins élevé* **faire**
  si $f_i(C) < f_i(E)$ **alors**
    retourner $C < E$;
  sinon
    si $f_i(C) < f_i(E) + \delta_i$ **alors**
      i := i-1
    sinon
      retourner $C > E$;
    fin
  fin
fin
retourner $C > E$;

Algorithme 1 : Détermination du meilleur élément entre C et E

1.4.3 Architecture

La figure 1.4 présente l'architecture de fonctionnement de l'optimisation. Elle se décompose en deux parties, la première relative à l'algorithme génétique qui génère des candidats, la seconde en lien avec les modèles robotiques pour évaluer ces candidats.

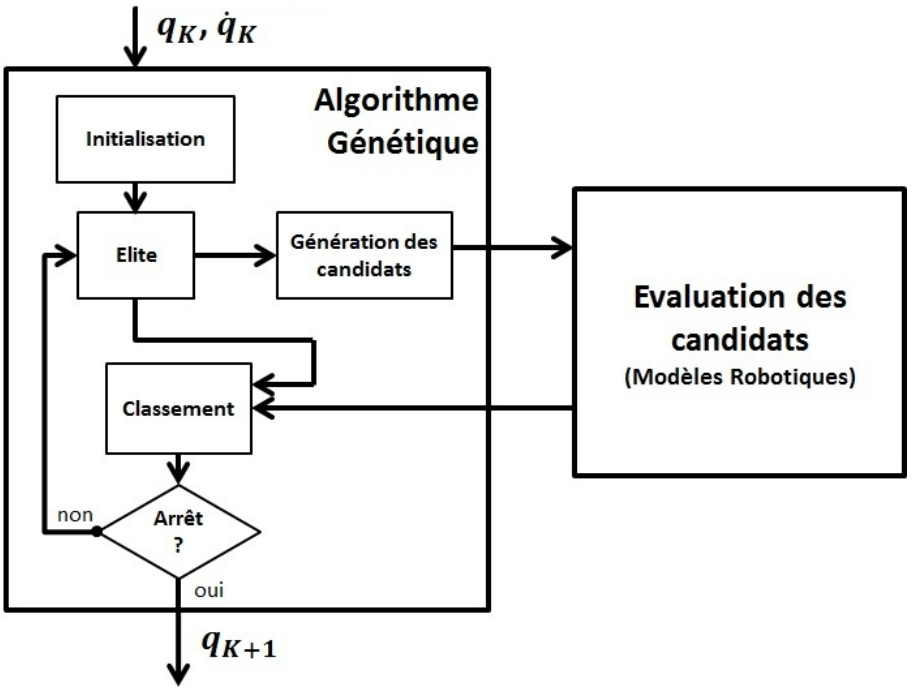

FIGURE 1.4 – Application à réaliser

L'objectif est de réduire le temps d'execution de cette application. Pour ce faire, nous allons nous concentrer sur le temps de calcul des modèles robotiques au niveau.

La première étape consiste à présenter quelques notions standards des domaines de la robotique et des systèmes embarqués. Ensuite, nous présenterons comment réduire les temps de calcul, pour finir par implanter et tester la solution retenue.

Chapitre 2

Notions théoriques

Pour améliorer les temps de calcul, il faut maîtriser les modèles à implanter, le fonctionnement du support de calcul (le système cible), les moyens de représenter les données sur cette cible mais aussi maîtriser les étapes de développement de la cible (la synthèse).

2.1 Les modèles robotiques

2.1.1 Description des robots

Du point de vue de la modélisation, les robots que nous considérons (manipulateurs, humanoïdes...) peuvent être vus comme une chaîne arborescente dont l'origine est un corps de référence qui peut être fixe ou mobile. Il est composé d'une succession de corps rigides reliés entre eux par des articulations. Chaque corps comporte un repère $X \in SE(3)$ comme présenté sur la Figure 2.1. Les modèles mathématiques des robots se basent sur les

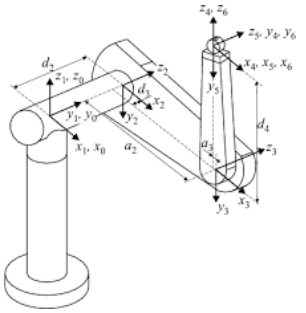

FIGURE 2.1 – Exemple de robot et des repères associés

paramètres dynamiques (masse, inertie, centre de masse...) et sur les relations de passage d'un repère à l'autre. Ces relations de passage peuvent être décrites par six variables (3 translations, 3 rotations), mais certaintes méthodes comme celle de Denavit-Hartenberg permet de réduire à quatre le nombre de variables en choisissant judicieusement la position des repères [19].

2.1.2 Modèle géométrique direct

Le modèle géométrique direct (MGD) est l'ensemble des relations qui permettent d'exprimer la situation de n'importe quel repère ou de l'organe terminal dans un repère global,

en fonction de ses coordonnées articulaires :

$$X = f(q) \tag{2.1}$$

$q \in \mathbb{R}^n$ étant le vecteur des variables articulaires et n : nombre de degrés de liberté tel que :

$$q = [q_1, q2, ..., qn]^T \tag{2.2}$$

Le vecteur X contient les coordonnées des repères ou la position de certains points du robot. Pour des robots à peu de degrés de liberté, il est possible d'avoir une expression analytique. Il est également possible d'utiliser une méthode itérative pour calculer la position de chaque repère en partant du repère de base. Dans notre cas, nous avons utilisé le logiciel de calcul formel MAPPLE en reprenant les travaux disponibles dans la bibliothèque HUMANS [18, 15] pour générer le code C du modèle géométrique direct. Il est à noter que cette fonction requiert le calcul des cosinus et sinus des variables articulaires puis contient uniquement des opérateurs basiques (additions et multiplications).

Modèle Dynamique inverse Ce modèle permet de déterminer les couples articulaires en fonction des coordonnées articulaires et des efforts extérieurs.

$$\Gamma = f(q, \dot{q}, \ddot{q}, F_{ext}) \tag{2.3}$$

Où :

- $\Gamma \in \mathbb{R}^n$ représente les couples des articulations
- $q \in \mathbb{R}^n, \dot{q} \in \mathbb{R}^n, \ddot{q} \in \mathbb{R}^n$: respectivement les coordonnées, vitesses et accélérations articulaires
- F_{ext} : représente les efforts extérieures

Pour cela, il existe 2 formalismes :

- **Formalisme de Lagrange** Ce formalisme décrit les équations en se basant sur l'équation de conservation d'énergie cinétique et potentielle :

$$\forall i \in \{1, \ldots, n\}\ \Gamma_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} \tag{2.4}$$

  Où :
  - L : Lagrangien du système égal à E-U
  - E : Energie cinétique totale du système
  - U : Energie potentielle totale du système

  Le formalisme de Lagrange décrit le modèle dynamique inverse sous la forme suivante :

$$\Gamma_i = A(q) \times \ddot{q} + H(q, \dot{q}) + J^T F_{ext} \tag{2.5}$$

  Avec :
  - A : matrice inertie $\in \mathbb{R}^{n \times n}$
  - H : vecteur Gravité, Coriolis et Centrifuge $\in \mathbb{R}^n$
  - J : Jacobienne traduit l'effet des efforts extérieurs sur les couples articulaires (obtenue en dérivant le modèle géométrique direct par rapport à q)

  De même que pour le calcul du modèle géométrique direct, il requiert le calcul des cosinus et sinus des variables articulaires puis contient uniquement des opérateurs basiques (additions et multiplications). L'expression analytique des matrices A et H est donné par l'utilisation de HUMANS et MAPPLE.Les forces extérieurs sont considérées nulles dans notre application.

— **Newton-Euler** Ce formalisme propose un algorithme de calcul récursif. Il existe plusieurs formes de cet algorithme :
  — La forme séquentielle de Luh, Walker et Paul qui permet d'être utilisé par la plupart des unités de calcul, il est de complexité d'ordre n et son temps d'exécution augmente linéairement en fonctions de n.
  — La forme parallèle [6] (disponible uniquement pour les robots parallèles) permet de réduire ce temps d'exécution, il est d'ordre log n mais ne peut être utilisée que par des unités de calcul parallèles ou plusieurs unités de calcul séquentielles gérées par un maitre (séquentielle).

Dans notre cas, Nous utiliserons le formalisme de Lagrange qui ce prête mieux à la parallélisation qu'une méthode itérative. Cette parallélisation est recherchée dans le but d'améliorer les temps de calcul. Pour comprendre, il nous faut des notions sur les différentes support de calcul, c'est à dire les différentes architectures possibles pouvant réaliser nos calculs.

2.2 Supports de calcul

2.2.1 CPU

Un CPU (Central Processing Unit) constitue le processeur utilisé dans les ordinateurs. Il peut avoir plusieurs coeurs qui lui permettent une certaine capacité de parallélisation (c'est-à dire qu'il peut exécuter plusieurs instructions en même temps). Le défaut est que cette parallélisation est possible uniquement si les coeurs n'utilisent pas les mêmes ressources en même temps (comme la mémoire). Sa force vient de sa fréquence de fonctionnement qui peut aller jusqu'à environ 4 GHz. Il possède une mémoire cache à plusieurs niveau (L1, L2, L3) qui lui permet de réduire les temps d'accès aux données mise en cache. Chaque niveau de cache est différent, le niveau L1 aura plus de mémoire que les autres mais de moins bonnes performances temporelles, tandis que ce sera l'inverse pour le niveau L3. Ce qui fait de lui un bon candidat pour les algorithmes récursifs comme la méthode de Newton-Euler qui est plus apte à rentrer entièrement dans le cache de plus bas niveau (le plus rapide).

2.2.2 FPGA

Le FPGA (Field-Programmable Gate Array) est un circuit intégré. Il possède une architecture fixe reconfigurable. C'est-à-dire qu'il dispose d'un nombre limité d'éléments logiques mais nous pouvons choisir lesquels utiliser et comment les relier. Les circuits reprogrammables peuvent jouir de la même souplesse d'exécution logicielle qu'un système basé processeur puisqu'il est possible de recréer un processeur sur FPGA, mais ils ne sont pas limités par le nombre de cœurs de traitement disponibles, ils sont juste limités par la fréquence d'utilisation qui est inférieure à celle d'un CPU (jusqu'à 1GHz). Contrairement aux processeurs, les FPGA sont parallèles par nature, de sorte que plusieurs opérations de traitement différentes ne se trouvent pas en concurrence lors de l'utilisation des ressources. Chaque tâche de traitement indépendante est affectée à une section spécifique du circuit, et peut donc s'exécuter en toute autonomie sans dépendre des autres blocs logiques. En conséquence, vous pouvez accroître le volume de traitement effectué sans que les performances d'une partie de l'application n'en soient affectées pour autant. Ainsi, le problème de temps de calcul devient un problème d'espace (de nombres de blocs logiques). Ceci

fait de lui un candidat idéal pour les algorithmes parallélisables comme le formalisme de Lagrange.

2.2.3 GPU

C'est un processeur graphique utilisé dans nos ordinateurs pour tout ce qui est application graphique. Il est équipé de millliers de coeurs ce qui lui permet de traiter de grandes quantités de données rapidement grace à la parallélisation. Certains travaux se base sur le GPU ou compare ses performances par rapports aux deux autres architecture vues précédemment (Graphic Processing Unit) [17]. En se basant sur l'implantation sur les tests de l'implantation de la librairie BLAS (Basic Linear Algebra Subroutines) sur ces trois architectures, nous pouvons voir Figure 2.2-a qu'en terme de temps, plus la dimension de la matrice est importante plus il est avantageux d'utiliser un FPGA ou un CPU. De plus, la Figure 2.2-b nous montre qu'en terme énergétique, il est préférable d'utiliser un FPGA. Dans notre cas, nous avons écarté le GPU aux vues des résultats de comparaison entre FPGA, CPU et GPU publiés dans [7].

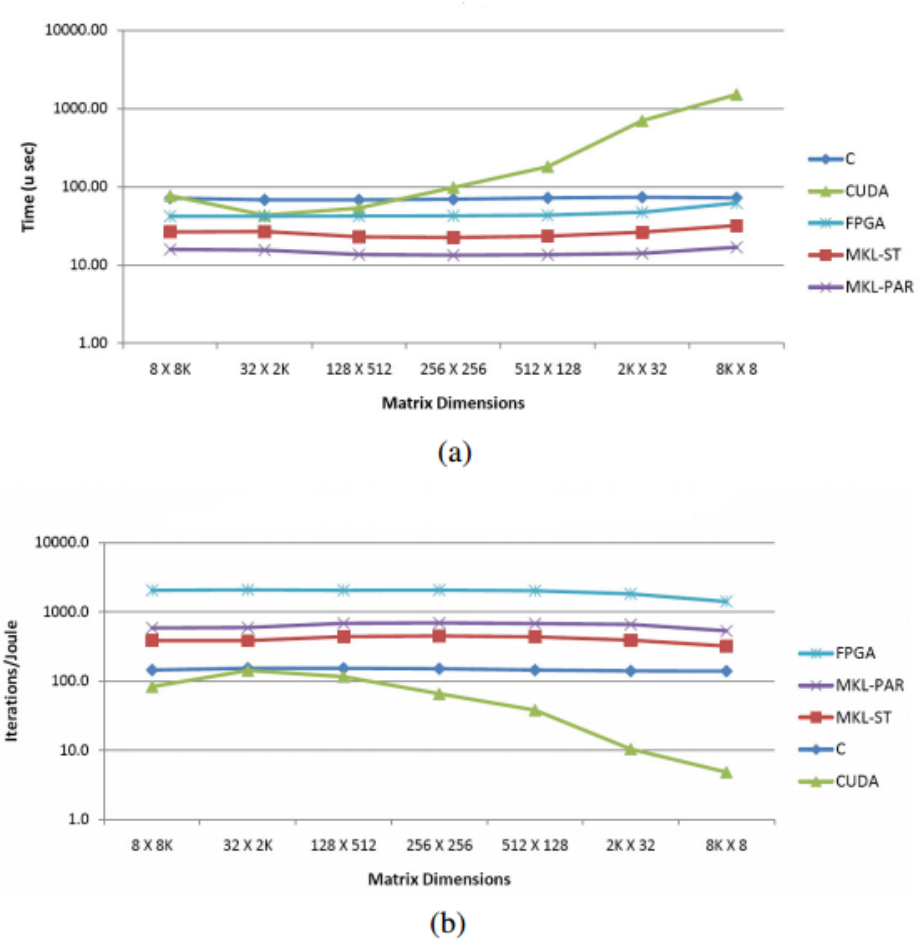

FIGURE 2.2 – Comparaison FPGA, CPU et GPU

2.2.4 Conclusion

Tout d'abord, nous savons qu'en général un CPU qui se prête mieux à la méthode itérative est utilisé. Pour notre cas, nous avons décidé d'utiliser un FPGA pour notre système et d'implanter la méthode de Lagrange.

2.3 Représentation des données

Nous avons donc choisi notre cible de calcul, il nous faut étudier les typages des données utilisables sur cette cible afin de pourvoir implanter notre algorithme. Il existe deux façons de représenter des nombres décimaux, le codage virgule fixe ou le codage virgule flottante. L'un est utilisé dans les architectures logicielles utilisant les CPU, et l'autre est utilisé dans les architectures matérielles telles les FPGA.

2.3.1 Virgule flottante

Le principe de ce codage repose sur le même principe que l'écriture scientifique, c'est à dire qu'un nombre est réprésenté par son signe, son exposant et sa valeur (appelée mantisse). Par exemple, en C, le type float possède 1 bit de signe + 8 bits d'exposant + 23 bits de mantisse en simple précision (1+11+52 bits en double précision). De cette manière, nous pouvons coder une grande plage de valeur et représenter des nombres très petits ainsi que des nombres très grands. Ce codage est très utilisé dans les architectures logicielles du fait de sa plage de valeur pseudo-infini et de sa grande précision. Mais il est très peu utilisé dans les architectures matérielles car il a besoin d'unité de calculs dédié (ce qui augmente le coût matériel). Les calculs flotants prennent plus de temps que le codage virgule fixe car ils réalignent la virgule pour obtenir le même exposant avant chaque addition. Donc il est déconseillé pour notre algorithme qui comprend uniquement des additions et multiplications.

2.3.2 Virgule Fixe

Le principe de ce codage est de représenter la partie entière du nombre et la partie décimal sur un nombre de bits fixe. Ce qui veut dire que pour un nombre total de bits, plus il y aura de bits décimaux, plus nous serons précis mais la plage de nombres représentables diminuera.
Exemple : Un codage 8bits dont 2 décimaux aura une plage de valeur de $[-32,75; 31.75]$ avec une précision de 0.25. Tandis qu'un codage 8bits dont 4 décimaux aura une plage de valeur de $[-8,9375; 7.9375]$ avec une précision de 0.0625
Ce codage est utilisé dans les architectures matérielles en raison de sa rapidité pour calculer des sommes ou produits car il n'y a pas besoin d'opération d'alignement.

2.3.3 En général

Le codage flotant est utilisé pour créer et vérifier des algorithmes de calculs sur ordinateur puis est transtypé en codage virgule fixe lors de l'implantation matérielle, ce qui peut engendré des problèmes comme montré dans la section C

2.4 Synthèse sur FPGA

La synthèse sur FPGA consiste à :

- **Décrire l'architecture matérielle à réaliser :** Il est possible de décrire notre architecture matérielle (sous forme de fichier VHDL par exemple) en utilisant les éléments de base intégrés dans le logiciel de développement (Quartus dans notre cas). Cependant, nous avons aussi la possibilité d'écrire nos propres blocs logiques et forcer le synthétiseur à les utiliser. C'est l'unique étape réalisée par le développeur. Les performances finales dépendent également des étapes suivantes qui sont gérées par un programme appelé le synthétiseur. L'architecture peut être décrite en "behavorial" (comportemental) ou en structurel. Dans le premier cas, le developpeur décrit le comportement que devra avoir le système et le synthétiseur s'occupe de choisir les blocs nécessaires à sa réalisation en respectant les contraintes imposées. L'autre cas se fait à plus bas niveau, il consiste à réaliser le système en instanciant soit même chaque bloc (prédéfinis ou fait sur-mesure), ainsi le développeur laisse moins de liberté au synthétiseur lui permettant d'avoir un meilleur contrôle du système et avoir plus d'option de configuration. Ce travail peut être très difficile mais très important lorsqu'on a de contraintes de temps ou d'espace important.
- **Réaliser le modèle RTL (Register Transfer Level)** : c'est-à-dire décrire l'implantation sous forme d'éléments séquentiels (les registres ou bascules) et de combinaisons logiques entre les différentes entrées/sorties. Cette étape est réalisée par le synthétiseur de Quartus (logiciel d'Altera de développement sur FPGA)
- **Réaliser la synthèse logique :** c'est l'étape qui permet de transformer la description RTL du circuit en une description au niveau portes logiques. Ceci est aussi réalisé par le synthétiseur de Quartus.
- **Placement et routage :** C'est le processus lors duquel les différentes parties d'un circuit sont automatiquement placées et connectées en fonction du problème à résoudre. Cette étape est aussi géré par le synthétiseur de Quartus.

Finalement, la réalisation du VHDL de notre application est une des tâches que j'ai réalisé pendant le stage, le reste de la partie sur FPGA est réalisé par le synthétiseur sur lequel nous n'avons aucun controle mis à part quelques réglages possible. Ce qui veut dire que nous n'avions la main que sur la partie VHDL.

Conclusion

Nous avons decidé d'utiliser un FPGA en utilisant le formalisme de Lagrange qui permet une grande possibilité de parallélisation puisque nous ne gérons que des opérations simples (additions, multiplication) sur des applications matricielles (additions et multiplications de matrice) qui sont aussi hautement parallélisable. Mais pour cela, il nous faut bien étudier l'algorithme afin de gérer au mieux ces parallélisations et essayer d'optimiser chaque opérations. C'est ce que nous verrons au prochain chapitre.

Chapitre 3

Developpement

Le but de ce chapitre est de trouver comment développer notre système sur FPGA de manière à optimiser les temps de calcul. Nous devons jouer sur les opérateurs d'additions et de multiplications car ce sont les opérations de base de notre algorithme. Nous pouvons tirer partie du parallélisme des calculs.

Pour rappel, nous avons l'équation $\Gamma = A \times \ddot{q} + H$. Nous ne pensons pas pouvoir gagner du temps de calcul sur le remplissage des matrices A et H ; nous avons concentré les efforts sur les opérations de base (additions et multiplications) et la parallélisation de la multiplication matricielle et de l'addition vectorielle.

Nous avons choisis de ne pas faire les calculs des cosinus et sinus sur FPGA considérant ces valeurs comme des entrées de notre système.

3.1 Additionneurs

Théorie

L'addition est l'élément de base de notre algorithme, c'est pourquoi il faut chercher l'additionneur le plus rapide entre le "+" VHDL et nos propres "surchages". Il existe plusieurs types d'additionneurs, le plus commun est l'additionneur à propagation de retenue montré en Figure 3.2. Le principe est simple et repose sur de la combinatoire. Cependant,

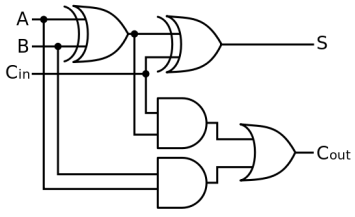

FIGURE 3.1 – schéma logique du Full Adder 1bit

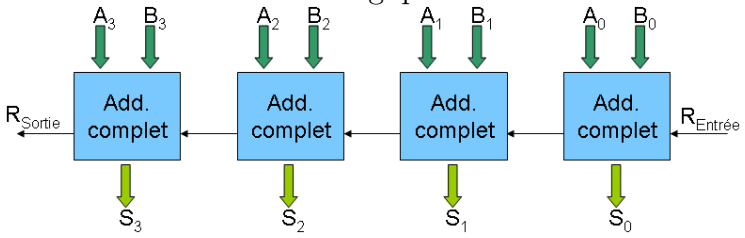

FIGURE 3.2 – Additionneur 4bits à propagation de retenue

la retenue d'un bloc va dans l'additionneur suivant, ce qui fait que la propagation de retenue est d'autant plus longue qu'il y a de bits et le delai de propagation est d'autant plus grand que les donnnées sont représentées sur un grand format. D'autres types d'additionneurs ont été créés pour résoudre ce problème (une étude plus complète a été faite dans [12]) :

— le CSA (Carry Select adder) : cet Adder se compose en plusieurs Blocs contenant plusieurs cellules CSA (voir Figure 3.3). Chacune de ces cellules (exceptée la première du bloc) est composé de 2 adders à propagation de retenue et d'un multiplexeur 1 bit. Tout cela pour un delai d'ordre $n^{1/2}$.

— l'additionneur à retenue bondissante (Carry Skip Adder) : Le principe est plus complexe, cette methode se base sur l'adder à propagation de retenue et sur un bloc de comparaison pour interpréter les retenues de chaque bloc. Chaque bloc peut travailler en partie en parallèle et la détermination du delai se fait par l'analyse du pire cas (qui est lorsque la retenue est générée au premier bloc et doit être propagée jusqu'au dernier bloc pour calculer la dernière somme). Le delai normal sans optimisation est d'O($n^{1/2}$) comme le CSA, mais il est possible d'optimiser ce delai en jouant avec le nombre de blocs, les tailles de ceux-ci ainsi qu'avec le niveau de saut. Par exemple pour un Adder de ce type en 32bits, nous avons un delai théorique de propagation de la retenue 25 fois le delai d'une porte (idem pour le CSA) contre 62 fois le delai d'une porte pour l'Adder à propagation de retenu. Or après synthèse des blocs, nous pouvons obtenir un delai de 18 fois le delai d'une porte.

— l'Adder parallèle CS (Carry Save) : Sûrement le plus rapide. Il repose sur un codage particulier des opérandes. Chaque bit devient 2 bits (voir tableau 3.1).
Ces deux opérandes modifiés rentrent sur 2 étages de full adder Figure 3.4. La retenue pour l'étage suivant est calculée au premier étage mais utilisé qu'au deuxième. Ce qui permet de ne pas perdre de temps avec les propagations de retenues Figure 3.5. C'est pourquoi le nombre de bloc CS (et donc le nombre de bits des opérandes) ne change rien au delai (correspond au délai d'un Full Adder 2bits donc deux fois le delai d'un Full Adder 1bits).

codage naturel	0	1	1 + retenue
codage Carry Save	00	01 ou 10	11

TABLE 3.1 – Codage Carry Save

Nous pouvons en déduire qu'en théorie, c'est l'additionneur Carry Save qui est le plus performant. Des test ont été réalisés dans la partie suivante avec Quartus, un logiciel de developpement sur FPGA d'Altera.

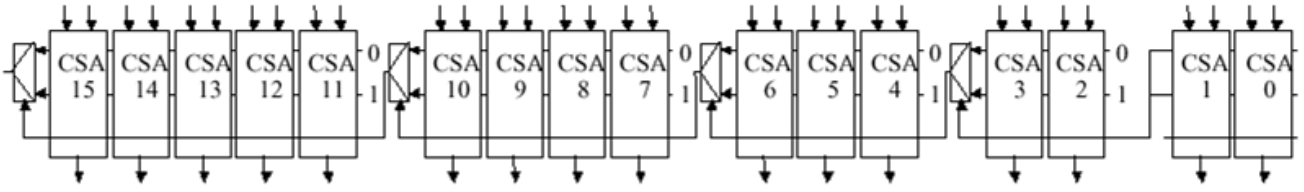

FIGURE 3.3 – Adder 16 bits avec 5 blocs Carry Select Adder

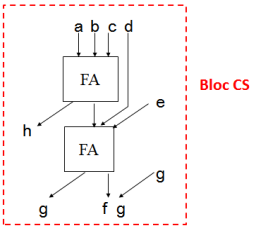

FIGURE 3.4 – Bloc Carry Save

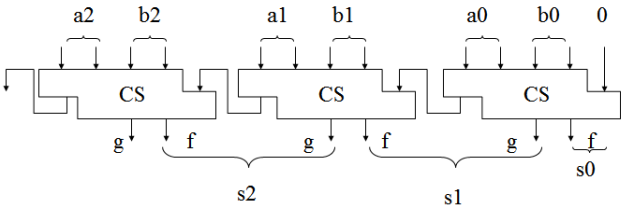

FIGURE 3.5 – Adder CS

Validation Pratique

En théorie, nous avons vu que le Adder CS correspond le plus à nos critères mais Quartus, le logiciel de developpement sur FPGA d'Altera, possède ses propres additionneurs, le "+" VHDL et le module générique LPM_ADD (Library of Parameterized Modules, ADD :adder). Cependant Altera ne nous laisse pas la possibilité de voir comment ils sont fait, donc la solution pour les comparer est de les tester directement en réalisant l'opération $A + B$ de plusieurs façons différentes sous Quartus II 9.1. Nous cherchons le plus rapide d'entre eux dans notre cas.

Dans un premier temps, testons les additionneurs avec les réglages de synthèse par défaut de l'Annexe A. Comme nous le montre le tableau 3.2, avec ces réglages, l'additionneur le plus rapide est bien le Carry Save avec un résultat en un peu moins de 10ns. Nous pouvons aussi remarqué que le temps de calcul du Carry Save 32bits est environ le même que celui du Full adder 1bit comme attendu. En changeant les réglages comme en

Type	Temps (ns)
"+" VHDL	11.094
LPM 32bits	12.023
FA 1bits	9.698
FA théorique 32bits (32xFA 1bits)	310.3 (32x9.698)
FA pratique 32bits	21.270
CS 32bits	9.866

TABLE 3.2 – Comparaison des temps des additionneurs avec le réglage par défaut

Annexe B, nous obtenons d'autres résultats présentés dans le tableau 3.3. Chaque additionneur a gagné en rapidité mais là encore, c'est le CS qui se démarque avec un résultat en moins de 8ns. Ces tests nous montrent les temps de calcul pour un seul de chaque additionneur mais ce n'est pas suffisante pour affirmer lequel est le mieux pour notre cas. C'est pourquoi, nous avons testé la mise en cascade de ces additionneurs, c'est à dire faire l'opération (((((A+B)+C)+D)+E)+F) qui correspond mieux à notre cas. Nous remarquons un renversement de situation en faveur du "+" VHDL montré sur le tableau 3.4. En

Type	Temps (ns)
“+” VHDL	10.998
LPM 32bits	10.803
FA 1bits	7.120
FA théorique 32bits	227.8
FA pratique 32bits	17.315
CS 32bits	7.756

Table 3.3 – Comparaison des temps des additionneurs avec le réglage 2

Type	Temps (ns)
“+” VHDL	17.657
LPM 32bits	21.685
FA théorique 32bits	1139
FA pratique 32bits	41.954
CS 32bits	22.030

Table 3.4 – Comparaison des temps de (((((A+B)+C)+D)+E)+F) avec chaque additionneur

théorie, lors de la mise en cascade, les temps de propagation de chaque adder s'additionne selon le nombre d'étages (5 fois dans notre cas). Or, ici nous pouvons voir que le temps de propagation est inférieur à la théorie. En effet, le "+" VHDL donne un résultat en 17ns contre 22ns pour le CS, alors qu'ils devraient sortir un résultats respectivement de 55ns et 40ns. Le fait est que le synthétiseur de Quartus optimise selon des paramètres réglables et d'autres où nous n'avons aucun contrôle. Ce qui pose problème pour la prédictibilité des résultats bien que les résultats soient satisfaisants. En sachant cela, nous opterons pour le “+” VHDL qui obtient de meilleures performances temporelles lors de la mise en cascade. Pour l'operateur multiplier (qui n'est qu'une succession d'additions et de décalages), nous utiliserons aussi le “x” VHDL, puisque, même si nous ne l'avons pas traité, nous pouvons supposer qu'il aura de meilleures performances. Il est plus censé d'essayer d'optimiser la parallélisation.

3.2 Parallélisation

L'équation qui nous intéresse (2.5), contient une multiplication matricelle et une addition vectorielle. Nous allons mettre en avant 2 niveaux de parallélisation possible sur la multiplication et 1 niveau pour l'addition.

3.2.1 Multiplication

Notons R le vecteur résultant de la multiplication matricelle ($R = A \times \ddot{q}$) et R_i le i-ème élément de ce vecteur. L'équation (2.5) nous montre le calcul d'un élément du vecteur résultant.

$$R_i = \sum_{j=1}^{n} A_{i,j} \times \ddot{q}_j \tag{3.1}$$

Nous pouvons voir que le calcul des n éléments du vecteur résultant ne dépend pas d'un autre élément de ce meme vecteur. Nous pouvons paralléliser le calcul de chaque élé-

ment du vecteur résultant. Par conséquent le temps de calcul du vecteur R correspondra au temps de calcul d'un élément de R_i. C'est ce que nous appelerons le 1er niveau de parallèlisation de la multiplication matricielle.

Nous pouvons constater un deuxième niveau de parallélisation. En effet, le calcul d'un élément du vecteur résultant est composé de n multiplications indépendantes entre elles, ce qui signifie que chaque multiplications peut se faire en parallèle.

Il y a aussi les $(n-1)$ additions que nous pouvons exécuter deux par deux et faire de même avec chaque résultats Figure 3.6.

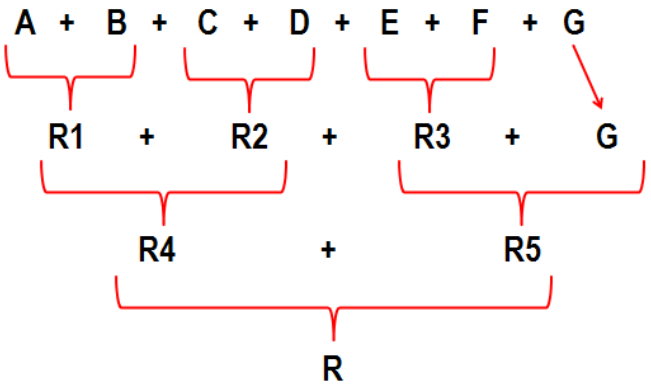

FIGURE 3.6 – Execution des addition

Ce qui nous donne au final, un temps de calcul des additions de $LOG_2(n-1)$ [1]. Ainsi, l'équation (3.2) nous indique le temps de calcul T_M qu'il faudra pour réaliser la multiplication matricielle en fonction du nombre de degrés de liberté n et du temps de calcul d'une addition t_a et multiplication t_m.

$$T_M = 1 \times t_m + LOG_2(n-1) \times t_a \tag{3.2}$$

Addition

Pour l'addition, l'équation (3.3) montre que chaque élément du vecteur de couple peut être fait en parallèle. Et donc, le temps de calcul du vector résultat est le temps de calcul d'un élément de ce vecteur qui correspond au temps de calcul d'une addition.

$$\forall i \in \{1, \ldots, n\} \Gamma_i = R_i + H_i \tag{3.3}$$

Ainsi, l'équation (3.4) nous indique le temps de calcul T_D qu'il faudra pour réaliser le modèle dynamique inverse en fonction du nombre de degrés de liberté et du temps de calcul d'une simple addition et multiplication.

$$T_D = T_M + t_a = t_m + (LOG_2(n-1) + 1) \times t_a \tag{3.4}$$

Ce qui nous donne en théorie, un tant de calcul de 59ns avec 7 degrés de liberté, avec $t_a = 11ns$ et $t_m = 15ns$ (sans compter le temps de calcul des matrices A et H). Les résultats théoriques sont très prometteurs, il nous faut donc passer à l'implantation sur cible pour valider ceci.

1. Nous définissons $LOG_2(n)$ comme l'arrondi à l'entier supérieur de $log_2(n)$

Chapitre 4

Implantation

4.1 Implantation de l'algorithme

L'implantation sur FPGA des modèles robotiques est présentée Figure 4.1. Nous pouvons voir les différents modules que nous allons créer en VHDL sur Quartus Prime 15.1.

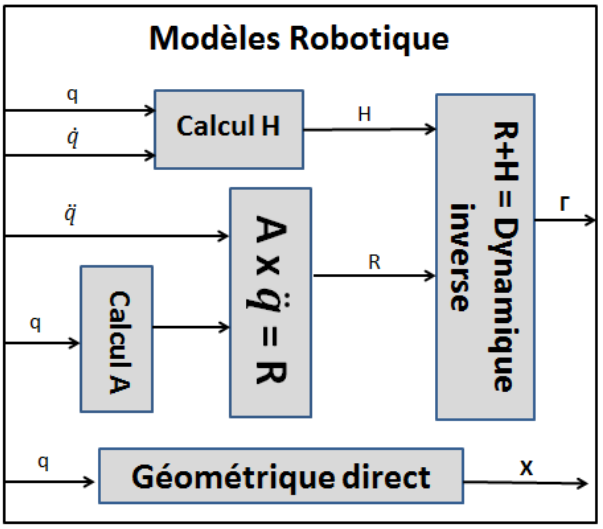

FIGURE 4.1 – Architecture de la partie "modèle géométrique"

Cette décomposition en bloc va nous permettre de valider chaque module indépendemment pour debugger le système plus facilement et rapidement. Une fois tous les modules validés, il nous suffira de tester afin de valider l'ensemble avant de commencer le transtypage en virgule fixe qui peut être une grande source d'erreur.

4.2 Transtypage

Un mauvais transtypage peut emmener à des résultats incohérents. La Figure 4.2 montre la réponse à un échelon d'un système du deuxième ordre simulé avec différents formats de représentation. Si on considère le format double comme la référence, nous constatons que les petits formats de données virgules fixe produisent un résultat erroné.

Un transtypage mal réalisé signifierait avoir des résultats inexploitables voire dangereux selon l'application faite. C'est pourquoi, il est important de valider cette partie. Le

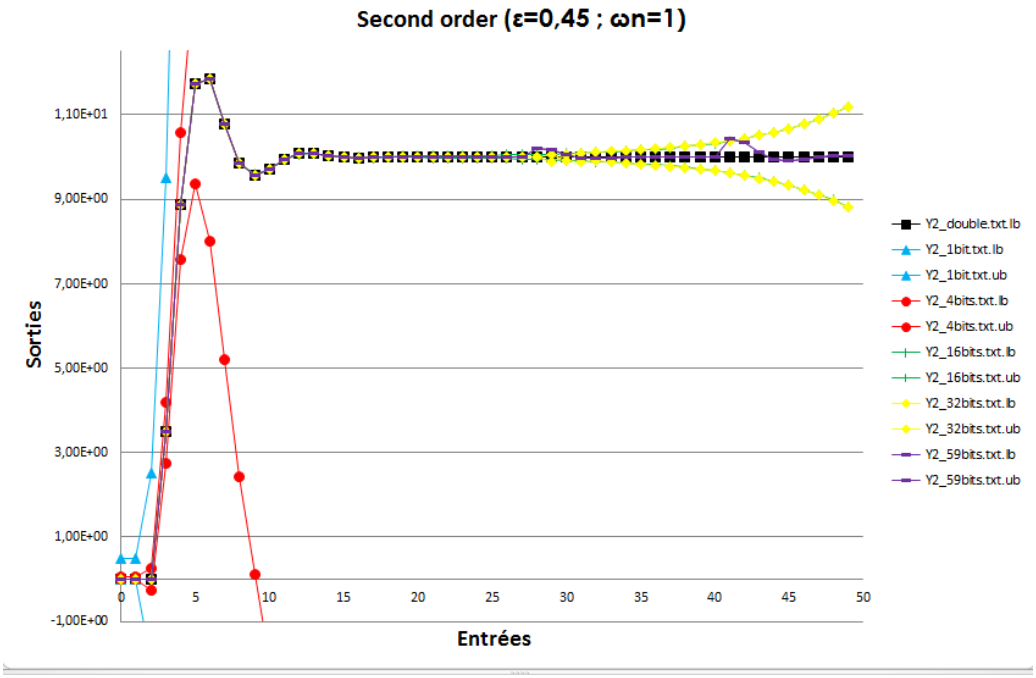

FIGURE 4.2 – Résultat du test

transtypage est un travail lourd et difficile. Le transtypage détermine le format de données (nombre de bits) nécessaire pour garantir une précision donnée des résultats.

Le principe est de déterminer la pire combinaison d'entrées menant à la moins bonne précision et de s'assurer que cette précision est dans le seuil souhaité. Des travaux ont été réalisés afin de trouver des méthodes analytiques à ce soucis de transtypage [5]. Malgré c'est travaux, le transtypage reste un travail lourd, c'est pourquoi il a été développé un outil encore expérimental nommé IDfix, qui automatise le transtypage et que nous avions utilisé pour notre application.

4.2.1 IDfix

IDfix est un projet réalisé par l'équipe CAIRN de l'INRIA et de l'IRISA [4]. Il a été créé dans le but de faciliter le développement sur systèmes dédiés en automatisant la phase de transtypage. En effet, il traduit un programme C réalisé en virgule flottante en un programme C/C++ réalisé en virgule fixe, pour qu'ensuite les outils Vivado ou Catapult (outils de traduction C/C++ vers modèle RTL) nous donne le modèle RTL résultant. Ce qui permet, en théorie, à un programmateur C de pouvoir développer aussi sur systèmes dédiés, puisque la partie transtypage et génération du modèle RTL est automatisée. Il ne reste plus qu'à lancer le logiciel de la cible (Comme Quartus ou Xilinx pour les FPGA) pour compiler le design.
Le principe est de lui indiquer les signaux d'entrées avec leur plage de valeurs, le programme à convertir, quelles sont les sorties et la précision voulue de ses sorties (sous forme de rapport signal/bruit en dB). Ainsi, il sort le programme converti, la dynamique de chaque sortie et vaiables internes et la précision réellement obtenue. En théorie cela parait simple, pourtant beaucoup de problèmes ont été rencontrés lors de la mise en pratique. Les principaux sont :

— **Problème d'installation** : En suivant le guide d'installation sur leur site [4], nous avons installer Eclipse et Octave nous permettant d'utiliser IDfix. Après dif-

férents tests en utilisant les programmes exemples fournis pour nous rapprocher ensuite vers notre programme final, nous sommes tombé sur des erreurs lors de la compilation ("noise library function not found"). Il s'avère qu'il y a deux type de programme, lti ou non lti, l'un pour signifier que les calculs utilisé sont linéaires l'autre pour les calculs non linéaires. Pour les programmes de types lti, l'utilisation d'Octave est suffisante, mais pour les modèles non lti, il faut utiliser matlab. Donc en théorie, après avoir réinstaller IDfix avec matlab, tout devrait fonctionner. En pratique, après avoir essayer l'installation sur différents ordinateurs, nous avons réussi à faire fonctionner les modèles non lti que sur un seul ordinateur.

— **Problème de résultat** : Après avoir écrit le modèle dynamique et géométrique en C acceptable par IDfix, nous avons essayé de le transtyper avec cet outil, avec succès après correction de quelques erreurs, cependant les résultats semblaient étranges. En effet, nous avions une précision d'approximativement -3000dB alors qu'en théorie nous devions obtenir approximativement -150dB pour avoir une erreur ne dépassant pas 10^{-4}. De plus, les dynamiques de certaines variables ne contenaient pas de partie décimal (dans notre cas, c'est étrange). Ce problème demeure encore.

Malgré l'aide et la réactivité du support technique d'IDfix, certaines erreurs subsistent et rendent incertains les résultats obtenus. C'est un outil prometteur qui pourrait grandement faciliter le developpement sur systèmes dediées mais est finalement encore trop incertain pour l'utiliser dans notre cas. C'est pourquoi, nous avons décidé de résoudre le problème de transtypage en créant un programme qui résolvera ce problème à l'aide d'une méthode empirique.

4.2.2 Méthode empirique

Le principe de cette méthode est simple, il consiste à tester la précision obtenue en sortie en fonction des intervalles d'entrées, choisies aléatoirement, de largeur égale au pas de quantification des données.

Exemple : Soit le système de la Figure 4.3 avec :

— R = A + B
— la partie entière de A et B codées sur 4bits
— $A \in [-8; 7]$
— $B \in [-8; 7]$
— On veut R soit précis à 10^{-2}

La méthode commencera les tests avec 1bit décimal. Nous avons donc un pas de quantification de 0.5. Elle va donc donné un encadrement de valeurs codables à A et B à partir d'une valeur aléatoire non codables (3.51 deviendra $[3.5; 4]$).

Le test commence avec 1bit décimal, puis si la largeur de l'interval R est supérieur à

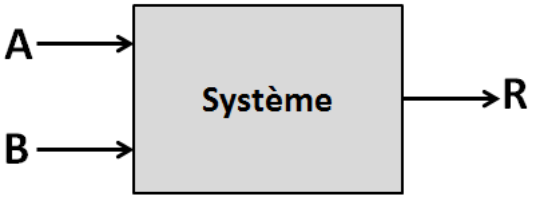

FIGURE 4.3 – Exemple Transtypage

l'erreur maximum voulue, elle essaiera avec 2bits décimaux. La méthode incrémentera le

nombre de bits décimaux à chaque fois que l'erreur est dépassée et s'arretera lorsque le nombre de bits décimaux est resté inchangé pendant un certains nombre de valeurs d'entrées décidé par l'utilisateur au lancement du programme. Cette méthode utilise l'analyse par interval décrite sur le site [8] pour manipuler les différents encadrements.
Ainsi, en choissisant un nombre élévé en condition d'arrêt, nous pouvons obtenir un résultat se rapprochant du résultat que nous aurions obtenu en utilisant manuellement les méthodes d'analyse décrite par l'IRISA [5]. Pour avoir un résultat certain, il aurait fallu tester toutes les combinaisons de valeurs d'entrées possibles pour chaque bits décimaux rajouté. Donc en reprenant notre exemple, nous avons 4bits entier et 1bit décimal au départ du programme, il faudra donc tester $2^5 \times 2^5$ possibilité, si l'erreur est dépassée tester avec 4bits entier et 2bits décimaux donc $2^6 \times 2^6$ possibilités, etc... Ceci uniquement pour 2 entrées. Dans notre cas, pour le modèle géométrique nous avons 28 entrées, donc pour 4bits entier et 1bit décimal, nous avons $(2^5)^{28} = 1.4e^{42}$ possibilités seulement pour la première itération du nombre de bits décimaux. Donc si chaque possibilité prend 1ns en étant très optimiste, il nous faudrait $16,2e^{27}$ années pour obtenir un résultat. Cette méthode génère énormément de calcul qui ne peuvent être fait en un temps raisonnable. C'est pourquoi nous ne pouvons pas avoir de résultat certifié, juste un résultat supposé correct selon le nombre pris en condition d'arret.

Pour notre application, nous avons utilisé cette méthode avec une condition d'arret de 1 000 000. Le résultat obtenu et de 25bits décimaux en prenant en compte les erreurs de troncature. La partie entière est choisie en considérant la valeur maximale de toutes les variables lors des 1 000 000 de tests. Ainsi, notre format de virgule fixe sera 32 bits dont 25bits décimaux.

Nous pouvons ainsi modifier les fichiers VHDL de notre application pour changer nos variables entières en format virgule fixe. A ce stade, le FPGA est prêt, il ne reste qu'à mettre en place la communication entre l'ordinateur et la cible.

En utilisant IDfix, nous aurions eu l'avantage d'avoir le format virgule fixe de chacune des variables utilisées. Avec la méthode empirique, chaque variable dispose du même format, ce qui n'est pas optimal.

4.3 Communication

La communication permettra d'utiliser notre implantation avec l'algorithme génétique, pour valider le bon fonctionnement du système et pour savoir si cette implantation obtient de meilleures résultats temporels que d'autres système. Il existe plusieurs communication possible entre un FPGA et un ordinateur mais c'est l'USB qui a été choisis car il a déjà été développé au sein du labo un outil gérant cette communication.

4.3.1 GPStudio

L'équipe DREAM a créé un outil de mise en place de la communication USB entre un FPGA et l'ordinateur. Cet outil est nommé GPStudio, il permet d'utiliser l'USB sans avoir à soi-même le développer. Nous avons juste à insérer notre VHDL corespondant à notre application dans le projet GPStudio, de le connecter au module "usb" développé par l'équipe DREAM et de compiler le tout. En observant la Figure 4.4, il parait évident de modifier les entrées et sorties de notre application de manière à n'avoir qu'une seule entrées série et une seule sortie série. Ceci a été fait en créant un module "Sérialisateur" et Désérialisateur. Ainsi, nous avons mis en place l'USB sans difficulté majeure. Mais

nous avons d'abord tester cet outil avec un VHDL faisant juste la multiplication de deux matrices (2×2).

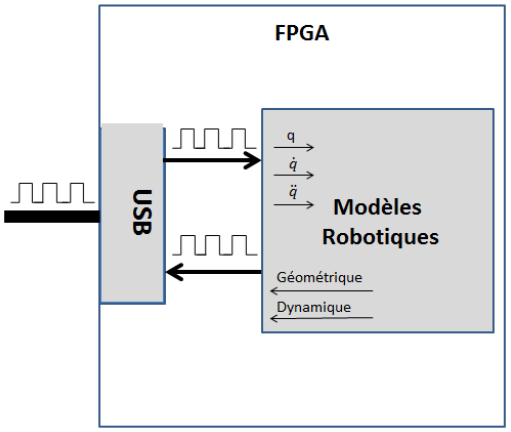

FIGURE 4.4 – Implantation avec GPStudio

4.3.2 Tests

Nous avons fait un premier test (après validation des modules sous Quartus) pour tester l'utilisation de GPStudio sur un système simple (multiplication de matrice 2x2). Après validation des test cet exemple, nous avons pu tester l'implantation de notre application en Figure 4.5. Ce qui nous a permis de confirmer le bon fonctionnement de notre sys-

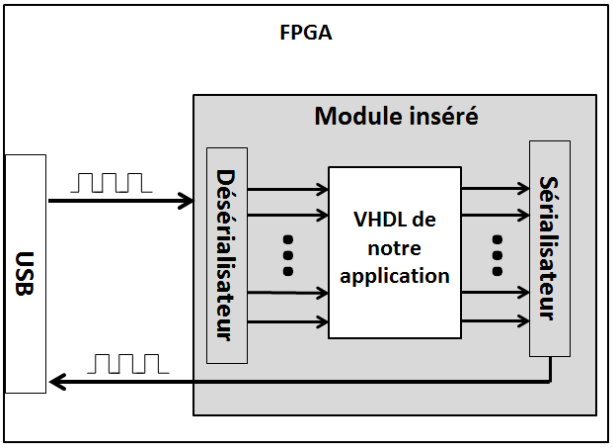

FIGURE 4.5 – Implantation avec GPStudio

tème. Cependant, le problème de l'USB est le temps non-déterministe de l'empaquetage des données avant envoi qui peut aller jusqu'à 1ms, ce qui nous empecherait d'avoir de meilleures temps de calcul que d'autres systèmes sur CPU qui n'ont pas ce soucis de communication. La solution serai d'envoyer un grand nombre de données dans un seul paquet de manière à calculer suffisamment d'échantillons pour pallier au temps de communication et concurrencer les autres systèmes.

Chapitre 5

Application

L'implantation étant réalisé, il nous reste à la tester sur notre application et vérifier les temps obtenus.

5.1 Description

L'application consistera à générer le mouvement d'un robot Kuka à 7 degrés de liberté comme montré en Figure 5.1 dans le but que l'effecteur atteigne un point donné, en évitant les obstacles et en minimisant l'énergie. L'algorithme génétique (exécuter sur PC) générera des candidats $q et \dot{q}$ (on en déduit $\ddot{q}$) à ce mouvement, qui seront envoyés en USB au FPGA. Il récupèrera ces candidats et générera pour chacun d'eux, le modèle géométrique et dynamique correspondant, qu'il renverra en USB. Ensuite, lorsque le PC recevra ces valeurs, l'algorithme génétique testera s'il y en a qui répondent aux critères (position de l'effecteur, la somme des couples au carré, la somme des accélérations au carré) tout en respectant les contraintes (couple max et collision). S'il y en a, la posture correspondant est généré, sinon une nouvelle série de candidats est regénérée (voir Figure 1.4).

5.2 Résultats

En théorie, pour obtenir de meilleures résultats temporels que les autres applications sur CPU, il nous faut valider l'équation (5.1).

$$2 \times t_{com} + N \times T_{mod} < NT_{CPU} \tag{5.1}$$

— t_{com} : temps d'envoi/reception d'un paquet
— T_{mod} : temps de calcul des modèles robotiques sur FPGA
— T_{CPU} : temps de calcul des modèles robotiques sur CPU
— N : nombre de candidats traités

Le but étant de trouver le nombre minimum de candidat à traiter qu'il faudrait pour battre le temps des autres implantations.

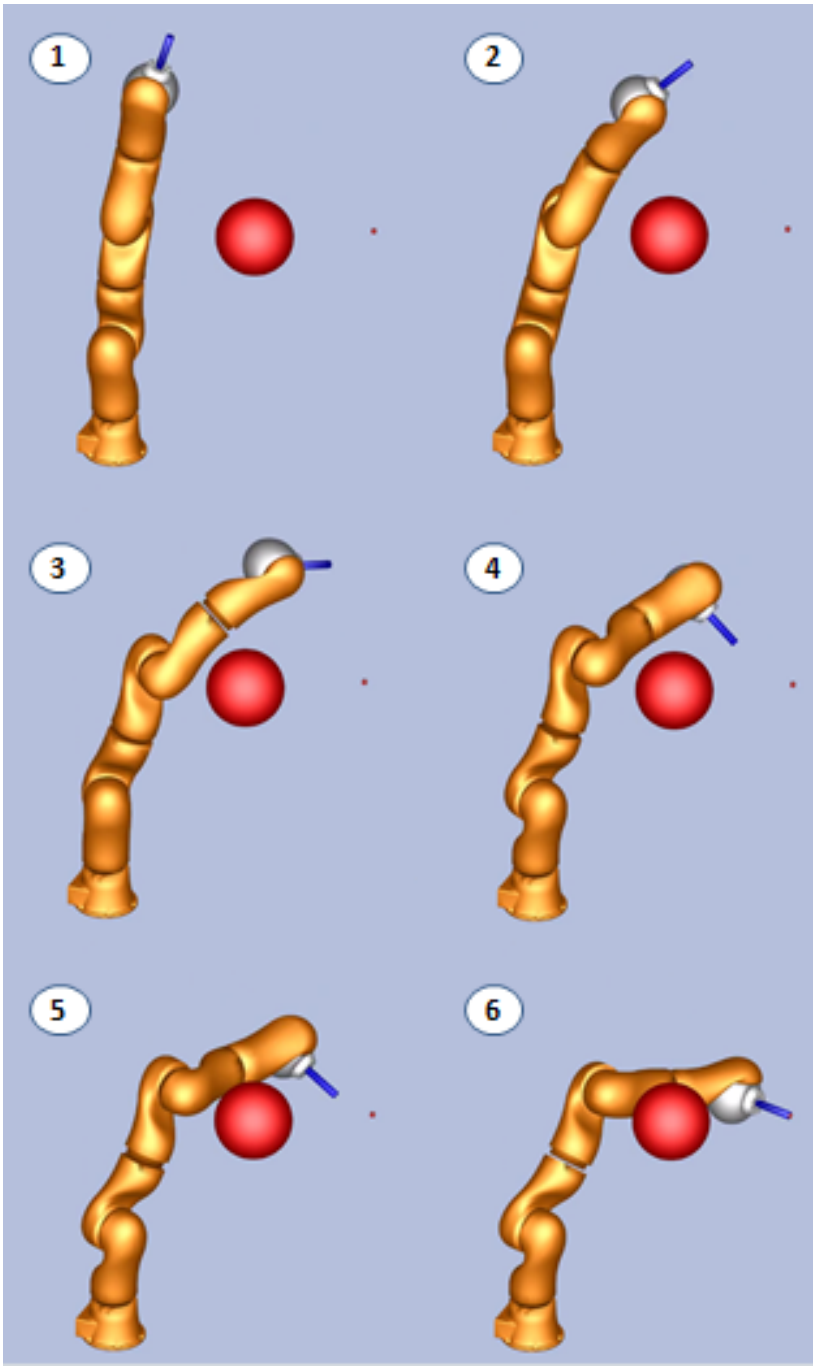

FIGURE 5.1 – Mouvement du robot Kuka à 7ddl

Chapitre 6

Conclusion et Perspective

L'application consistait à générer le mouvement d'un robot pour aller d'un point A à un point B en évitant les obstacles et en minimisant l'énergie. Le but étant de pouvoir faire mieux en terme de temps de calcul que l'implantation réalisé sur CPU. A l'aide d'une étude théorique en amont et d'une analyse en profondeur des problématiques des systèmes dédiées ainsi que des algorithmes de robotique, nous avons pu choisir et réaliser notre propre implantation sur FPGA qui possède des temps de calcul inférieur aux modèles sur CPU (de l'ordre de la centaine de nanosecondes sur FPGA contre quelques microsecondes sur CPU). Cependant, ces résultats ne tiennent pas compte du temps de communication entre le PC et le FPGA. Des tests sont en cours avec une communication USB entre le PC et le FPGA pour réaliser un système entièrement fonctionnel, mais les résultats attendus ne nous permetterait pas de faire de la planification en temps-réel puisque nous devons générer un grand nombre d'échantillons dans un seul paquet USB pour palier au temps non-déterministe de l'empaquetage qui est de l'ordre de la milliseconde. Malgré tout, il existe d'autres solutions que cette communication qui pourrait palier à ce problème et ainsi pouvoir utiliser notre système pour du temps-réel. En effet, nous pouvons développer une connection dédiée entre le FPGA et l'ordinateur. Mais la meilleure solution selon moi, serait d'implanter le système sur ordinateur ayant un FPGA intégré puisque la communication entre les deux serait dediée et optimisée. Une autre perspective consisterai à s'affranchir de l'ordinateur et de développer tous les algorithmes nécessaire sur des FPGA que l'on pourra intégré au robot.

Annexe A

Reglage1

— Optimization Technique Speed
— Power up don't care
— filter effort Fast fit
— Optimize hold timing I/O Paths and Minimum TPD paths
— Perform physical synthesis for combinationnal logic
— Perform register retiming
— Effort Level Normal

Annexe B

Reglage2

— Optimization Technique Speed
— Power up don't care
— filter effort Standart fit
— Optimize hold timing All paths
— Optimisation multi-corner timing
— Perform physical synthesis for combinationnal logic
— Perform register retiming
— Effort Level Extra
— Tout dans fitter netlist optimization
— Tout dans optimize for fitting

Annexe C

Test virgule fixe

Nous savons qu'un système de second ordre peut se mettre sous la forme de l'équation (C.1).

$$\frac{S}{E} = \frac{b_0+b_1 z}{z^2-2ze^{-\varepsilon w_n T_e}\cos(w_p T_e)+e^{-2\varepsilon w_n T_e}}$$

$$b_0 = e^{-2\varepsilon w_n T_e} + e^{-\varepsilon w_n T_e}\left(\frac{\varepsilon \sin(w_p T_e)}{\sqrt{1-\varepsilon^2}} - \cos(w_p T_e)\right)$$

$$b_1 = 1 - e^{-\varepsilon w_n T_e}\left(\frac{\varepsilon \sin(w_p T_e)}{\sqrt{1-\varepsilon^2}} - \cos(w_p T_e)\right) \tag{C.1}$$

$$w_p = w_n\sqrt{1-\varepsilon^2}$$

$$T_e = 1$$

Pour des soucis de clarté, nous allons introduire les variables des équations (C.2). (Rappel : $e^a * e^a = e^{2a}$)

$$\begin{aligned} A_d &= e^{-\varepsilon w_n} \\ k_1 &= 2 * Ad * cos(wp) \\ k_2 &= Ad * Ad \end{aligned} \tag{C.2}$$

Nous obtenons ainsi l'équation (C.3).

$$\frac{S}{E} = \frac{b_0 + b_1 z}{z^2 - k_1 z + k_2} = \frac{b_0 z^{-2} + b_1 z^{-1}}{1 - k_1 z^{-1} + k_2 z^{-2}} \tag{C.3}$$

En sachant que z^{-n} représente l'instant t-n (où t est l'instant présent) et en remaniant l'equation (C.3) de façon à isoler la sortie à l'instant présent, nous obtenons l'équation de récurrence (C.4).

$$S_t = b_0 E_{t-2} + b_1 E_{t-1} + k_1 S_{t-1} - k_2 S_{t-2} \tag{C.4}$$

Test de la reponse à un échelon ($E = 10$) d'un filtre de 2nd ordre ($\varepsilon = 45, w_n = 1$) avec plusieurs formats de virgule fixe. Le résultat de chaque format est représenté sous forme d'interval où seul les bornes sont représentées.

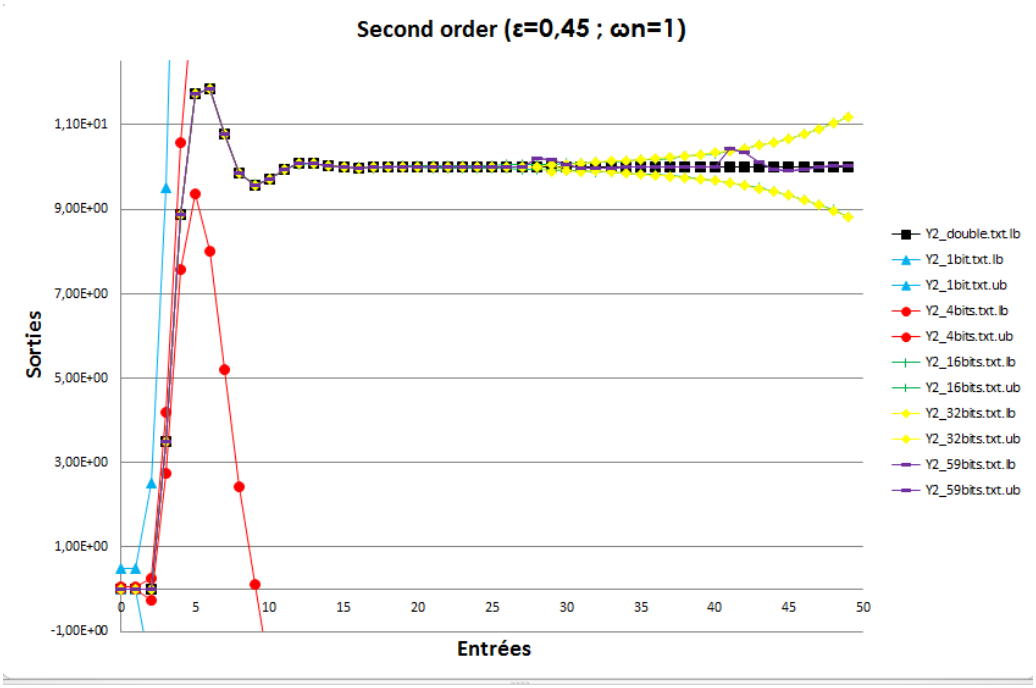

FIGURE C.1 – Résultat du test

Bibliographie

[1] https ://github.com/stack-of-tasks/pinocchio.

[2] https ://fr.wikipedia.org/wiki/Algorithme_genetique.

[3] M. Birem and D. F. Berry. A modular fpga-based smart camera architecture. *Journal of Systems Architecture*, 60(6) :519–527, 2014.

[4] Cairn. Infrastructure for the design of fixed-point systems (idfix). http ://idfix.gforge.inria.fr/doku/doku.php.

[5] D. M. E. de recherche R2D2 IRISA/INRIA ENSSAT Universite de Rennes1. Methodologies de conversion automatique en virgule fixe pour les applications de traitement du signal.

[6] T. T. B. B. K. Djouani. Calcul parallele de la dynamique inverse : Applications aux robots a chaines cinematiques ouvertes. 3rd International Conference : Sciences of Electronic, Technologies of Information and Telecommunications, Mars 2005.

[7] C. Grozea, Z. Bankovic, and P. Laskov. Fpga vs. multi-core cpus vs. gpus : Hands-on experience with a sorting application. In R. Keller, D. Kramer, and J.-P. Weiss, editors, *Facing the Multicore-Challenge*, volume 6310 of *Lecture Notes in Computer Science*, pages 105–117. Springer Berlin Heidelberg, 2010.

[8] IAMOCC. Interval analysis mocc. http ://iamooc.ensta-bretagne.fr/.

[9] S. Kajita, T. Nagasaki, K. Kaneko, and H. Hirukawa. Zmp-based biped running control. *IEEE Robotics & Automation Magazine*, 14 :63 – 72, 2007.

[10] K. Kaneko, F. KANEHIRO, S. KAJITA, H. HIRUKAWA, T. KAWASAKI, M. HIRATA, K. AKACHI, and T. ISOZUMI. Humanoid robot HRP-2. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1083–1090, Apr./May 2004.

[11] S. Lengagne, J. Vaillant, A. Kheddar, and E. Yoshida. Generation of whole-body optimal dynamic multi-contact motions. *International Journal of Robotics Research*, 2013. accepted.

[12] LIRMM. Operateurs. www.lirmm.fr/ rouzeyre/PDF/operateurs.ppt.

[13] M. Naveau, J. Carpentier, S. Barthelemy, O. Stasse, and P. Soueres. Metapod ? template meta-programming applied to dynamics : Cop-com trajectories filtering. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 401–406, Nov 2014.

[14] N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux, and E. Yoshida. Fast humanoid robot collision-free footstep planning using swept volume approximations. *IEEE Transactions on Robotics*, 28(2) :427–439, 2012.

[15] L. B. R. P.-G. Pierre-Brice Wieber, Florence Billet. The humans toolbox, a homogeneous framework for motion capture, analysis and simulation. In *the seventh*

International Symposium on Computer Methods in Biomechanics and Biomedical Engineering, 2006.

[16] N. S. K. Y. T. Tachibana, Y. Murata and M. Ito. Flexible implementation of genetic algorithms on fpgas. In *ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, pages 236–236, 2006.

[17] P. Vicente, R. Ferreira, L. Jamone, and A. Bernardino. Gpu-enabled particle based optimization for robotic-hand pose estimation and self-calibration. In *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*, pages 3–8, April 2015.

[18] P.-B. Wieber and F. Billet. Hybrid dynamics for the simulation of rehabilitation to walking by fes. In *the seventh ISB Symposium on 3D analysis of human movement*, 2006.

[19] E. D. Wisama Khalil. Bases de la modelisation et de la commande des robots manipulateurs de type serie, Mai 2012.