

Mémoire ingénieur

Contrôleurs en boucle fermée pour robot hexapode

Virgile Daugé

Année 2015–2016

Stage de fin d'études réalisé dans le laboratoire Institut National de Recherche en Informatique et en Automatique (INRIA)

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Jean-Baptiste Mouret

Encadrant universitaire : Suzanne Collin

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Daugé, Virgile

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31110541

Année universitaire : 2015–2016

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Contrôleurs en boucle fermée pour robot hexapode

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 18 décembre 2016

Signature :

Mémoire ingénieur

Contrôleurs en boucle fermée pour robot hexapode

Virgile Daugé

Année 2015–2016

Stage de Fin d'études réalisé dans le laboratoire INRIA
en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Virgile Daugé
1, rue du Général de Castelnau
54600, Villers-lès-Nancy
t+33 6 63 46 19 65
virgile.dauge@telecommancy.net

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecommancy.eu

INRIA
615 Rue du Jardin botanique,
54600 Villers-lès-Nancy
+33 3 83 59 30 00



Maître de stage : Jean-Baptiste Mouret

Encadrant universitaire : Suzanne Collin

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce stage.

En premier lieu, je remercie Mr Jean-Baptiste Mouret, responsable du projet resibots pour son accueil au sein de l'équipe ainsi que son soutien régulier.

Je remercie également Dorian Goepp pour le temps qu'il m'a consacré, le suivi constant apporté ainsi que ses conseils avisés aidant au bon déroulement du montage du robot.

Je tiens enfin à remercier l'ensemble de l'équipe Life-long Autonomy and interaction skills for Robots in a Sensing ENvironment (LARSEN) pour l'accueil chaleureux reçu dès mon arrivée, ainsi que pour de nombreux échanges passionnants sur différents sujets touchant de prêt ou de loin mon sujet de stage.

Table des matières

Remerciements	v
Table des matières	vii
1 Introduction et contexte	1
1.1 Introduction	1
1.2 Présentation du laboratoire	2
1.2.1 Interconnexion des structures	2
1.2.2 Centre national de la recherche scientifique (CNRS)	2
1.2.3 Institut National de Recherche en Informatique et en Automatique (INRIA)	4
1.2.4 Université de lorraine (UL)	5
1.2.5 Laboratoire lorrain de recherche en informatique et ses applications (LORIA)	5
1.3 Présentation de l'équipe	6
1.3.1 Organigramme	6
1.3.2 Resibots	7
2 État de l'art	8
2.1 Méthode et outils	8
2.1.1 Méthode	8
2.1.2 Outils	11
2.2 Approches Biomimétiques	12
2.2.1 Le vivant comme inspiration	12
2.2.2 Réseaux de neurones	12
2.2.3 Plusieurs niveaux de commande	14
2.2.4 Des comportements adaptatifs	15
2.2.5 Des résultats décevants	16
2.2.6 Une approche inadaptée	16
2.3 Approches plus pragmatiques	17
2.3.1 Le contrôle basé sur des pattes indépendantes	17

2.3.2	Le contrôle complet du robot	19
3	Architecture	22
3.1	Matériel	22
3.1.1	Robot petit Hexapode (PEXOD)	22
3.1.2	Nouvelle plateforme	23
3.1.3	Servomoteurs	23
3.1.4	Capteurs de force	24
3.2	Robot Operating System (ROS)	25
3.2.1	Brève description	25
3.2.2	Concepts Principaux	25
3.3	Noeud Hexapod_description	28
3.3.1	Fichier URDF	28
3.3.2	Génération par Macro	28
3.3.3	Visualisation du modèle	29
3.4	Noeud Hexapod_bringup	29
3.5	Noeud Optoforce	30
3.6	Noeud Rosdart	30
3.7	Noeud Hexapod controller	31
4	Contribution	32
4.1	Principes de base	32
4.1.1	Les entrées du système	32
4.1.2	Module de synchronisation : Hexapod	33
4.1.3	Des pattes indépendantes	33
4.2	Méthodes	35
4.2.1	Gestionnaire de versions	35
4.2.2	Prototypage	35
4.2.3	Intégration	36
4.2.4	Tests en Simulation	36
4.2.5	Tests Réels	37
4.3	Assemblage du robot	37
4.3.1	Existant	37
4.3.2	Mise en oeuvre des servomoteurs	37
4.3.3	Organisation interne	38
4.3.4	Mise on oeuvre des capteurs de force	39

4.3.5	Problèmes de hubs USB	39
4.4	Réalisation du bloc patte	39
4.4.1	Module de génération de trajectoires	39
4.4.2	Module de calculs cinématiques	45
4.4.3	Module de communication	49
4.5	Réalisation du bloc hexapod	50
4.5.1	Création des pattes	50
4.5.2	Synchronisation des données d'entrée	51
4.5.3	Synchronisation des pattes	53
4.6	Modèle du nouveau robot	55
4.6.1	organisation des scripts	55
4.6.2	Modifications pour les calculs cinématiques	56
4.6.3	Modifications pour RosDart	57
4.6.4	Modifications pour les capteurs de force	57
5	Résultats et performances	59
5.1	Résultats Obtenus	59
5.1.1	Une solution de cinématique inverse performante	59
5.1.2	Un Contrôleur facilement adaptable	60
5.1.3	Un contrôleur omnidirectionnel	60
5.1.4	Un contrôleur plus robuste ?	60
5.2	Axes d'amélioration	61
5.2.1	Contrôle de la force	61
5.2.2	Mapper les forces	62
5.2.3	Permettre de modifier les points neutres des pattes	62
5.2.4	Utiliser la centrale inertuelle	62
5.2.5	Planification locale du positionnement des pattes	63
5.2.6	Combiner plusieurs de ces approches	63
5.3	Respect du planning prévisionnel	63
5.4	Ressenti personnel	64
5.4.1	Un accueil Chaleureux	64
5.4.2	Un environnement enrichissant	64
5.4.3	Un domaine intéressant	64
5.4.4	Des difficultés rencontrées	65
5.4.5	Une légère déception	65

5.4.6	Des résultats encourageants	65
6	Conclusion	66
	Bibliographie / Webographie	69
	Liste des illustrations	75
	Listings	77
A	Organigramme LORIA	79
B	Example de fichier URDF	80
	Résumé	81
	Abstract	81

1 Introduction et contexte

1.1 Introduction

A travers les travaux récents du projet resibots présentés dans la publication *Robot that can adapt like animals* [13], l'équipe a montré comment un robot à 6 pattes –que nous nommerons hexapode– peut s'adapter à différents dommages en moins de deux minutes. Néanmoins, l'équipe n'a jusqu'à présent utilisé que des contrôleurs à commande en boucle ouverte,

L'objectif de ce stage est de développer et d'évaluer une (ou plusieurs) méthode(s) de contrôle pour le robot hexapode. Le contrôleur devra tirer parti des 6 capteurs de force situés sous les pieds pour adapter sa marche au terrain, et rester compatible avec les algorithmes d'adaptation développés par l'équipe.

Il faudra donc réaliser un état de l'art des méthodes actuelles de contrôle d'hexapodes ou de robots marcheurs afin de déterminer les potentielles approches permettant de réaliser un contrôleur réactif s'adaptant aux terrains difficiles.

Par la suite l'implémentation et le test de la méthode choisie devront être réalisés afin de pouvoir évaluer expérimentalement l'apport du contrôleur proposé.

Il est également important pour moi de participer à la vie de l'équipe et de découvrir le monde de la recherche. C'est une des raisons –au delà de l'intérêt prononcé pour le sujet– pour lesquelles j'ai choisi d'effectuer mon stage au sein de cette équipe et choisi de répondre à cette problématique.

Le monde de la recherche, notamment en apprentissage automatique m'intéresse en effet tout particulièrement. Sans être sûr de l'avenir, la possibilité de continuer à me former et à en apprendre toujours plus dans le domaine me séduit.

C'est donc dans ce contexte de curiosité et d'intérêt que commence ce stage de fin d'études.

1.2 Présentation du laboratoire

L'équipe LARSEN (voir 1.3) dans laquelle s'effectue le stage appartient à différentes entités distinctes, issues du monde de la recherche et de l'enseignement supérieur. Chaque entité sera présentée ultérieurement et nous allons donc dans un premier temps aborder l'architecture hiérarchique des entités au dessus de l'équipe.

1.2.1 Interconnexion des structures

Dans le but de permettre une compréhension rapide de l'organisation sous-jacente, la figure 1.1 a été réalisée. Comme vous pouvez le voir, l'équipe LARSEN appartient conjointement à la division Grand-Est INRIA et au Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA). Le LORIA, qui sera présenté en 1.2.5 est lui même une entité gérée conjointement par INRIA, le Centre National de la Recherche Scientifique (CNRS) et l'Université de Lorraine (UL).



FIGURE 1.1 – Hiérarchie des entités dont dépend l'équipe LARSEN

1.2.2 Centre national de la recherche scientifique (CNRS)

1.2.2.1 Le CNRS en quelques mots

Le CNRS est un Établissement Public à caractère Scientifique et Technologique (EPST) placé sous la tutelle du ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche. Il a pour objectif de produire du savoir et de le mettre au service de la société.

Le CNRS est présent dans tous les domaines de connaissance. C'est un organisme pluridisciplinaire menant des recherches dans l'ensemble des domaines scientifiques, technologiques et

sociétaux.

Ces différentes disciplines sont regroupées au sein de dix instituts :

- Institut National des Sciences Biologiques (INSB)
- Institut National de Chimie (INC)
- Institut écologie et environnement (INEE)
- Institut des sciences humaines et sociales (INSHS)
- Institut des sciences de l'information et de leurs interactions (INS2I)
- Institut des sciences de l'ingénierie et des systèmes (INSIS)
- Institut national des sciences mathématiques et de leurs interactions (INSMI)
- Institut de physique (INP)
- Institut national de physique nucléaire et physique des particules (IN2P3)
- Institut national des sciences de l'univers (INSU)

La sous-structure qui nous intéresse particulièrement, évoquée en 1.2.1 est INS2I. Cet institut repose sur 49 unités relevant des disciplines informatiques, du traitement du signal, du traitement des images, de l'automatique et de la robotique. L'objectif de cette entité est d'accompagner le développement des sciences informatiques, afin d'en maîtriser les enjeux scientifiques, technologiques et sociaux.

1.2.2.2 Le CNRS en quelques chiffres

le CNRS, compte près de 32 000 personnes, dont 11 000 chercheurs et 13 500 Ingénieurs, techniciens et administratifs.

Son budget pour l'année 2015 été de 3,3 milliards d'euros dont 769 millions en ressources propres.

Le CNRS est implanté à travers toute la France, s'appuyant sur 1100 unités de recherche et de service, dont près de 95% en partenariat avec l'enseignement supérieur.

Le CNRS, c'est aussi 1026 entreprises innovantes créées depuis 1999. Et 5629 familles de brevets.

1.2.3 Institut National de Recherche en Informatique et en Automatique (INRIA)

1.2.3.1 INRIA en quelques mots

INRIA est un institut de recherche dédié au numérique et promeut *l'excellence scientifique au service du transfert technologique et de la société*. INRIA est organisé en équipes-projets qui rassemblent des chercheurs aux compétences complémentaires autour d'un projet scientifique focalisé. Cette organisation induit plus d'ouverture et d'agilité, permettant de mener à bien ses missions avec ses partenaires industriels et académiques. INRIA répond ainsi aux enjeux de la transition numérique, à la fois pluridisciplinaires et très applicatifs de la transition numérique en marche.

INRIA transfert vers les entreprises ses innovations devenant ainsi créatrice de valeur et d'emploi, et ce dans de nombreux domaines, allant de la santé à la sécurité, en passant par les transports, l'énergie, et la communication qui, ensemble, formeront l'esquisse de la ville intelligente de demain, ou encore de l'usine du futur.

1.2.3.2 INRIA en quelques chiffres

INRIA comporte pas moins de 178 équipes projets, dont 139 en collaboration avec des universités et d'autres établissements de recherche, comme c'est le cas pour l'équipe LARSEN.

Ces équipes génèrent 4600 publications scientifiques par an. Dans la même période, 300 thèses sont soutenues.

L'INRIA développe aussi de nombreuses relations avec le monde industriel, dans le cadre de sa mission de transfert de connaissances à la société. L'institut détient 390 brevets. 141 logiciels ont été déposés à l'Agence pour la Protection des programmes (APP) en 2014. L'INRIA a également aidé à la création de 34 start-up depuis 2010.

L'ensemble des ressources de l'INRIA est réparti au sein de 8 centres de recherches à travers la France :

- Bordeaux
- Paris
- Rennes
- Sophia Antipolis
- Grenoble
- Nancy (Lieu de déroulement du stage)
- Lille
- Saclay

Enfin, le siège social est situé à Rocquencourt près de Paris.

Parmi ces ressources, l'on retrouve 2700 collaborateurs de 87 nationalités différentes, dont 1800 scientifiques.

Le budget total de l'INRIA s'élève à 203 M d'euros.

1.2.4 Université de lorraine (UL)

L'UL est issue de la fusion de différentes universités existentes le 1er janvier 2012. Les quatre entités fusionnées sont :

- l'Institut national polytechnique de Lorraine
- l'université Henri Poincaré
- l'université Nancy 2
- l'université Paul Verlaine-Metz

L'UL propose une gamme complète de formations, couvrant de nombreux domaines et coeurs de métiers. Mais outre son rôle d'enseignement supérieur, l'UL comporte également une forte composante orientée vers la recherche, qui nous intéresse ici. L'UL effectue des recherches dans de nombreux domaines scientifiques, technologiques et sociétaux.

Ces différentes disciplines sont regroupées au sein de dix pôles scientifiques :

- Agronomie, Agroalimentaire et Forêt (A2F)
- Automatique, Mathématiques, Informatique et leurs Interactions (AM2I)
- Biologie, Médecine, Santé (BMS)
- Connaissance, Langage, Communication, Sociétés (CLCS)
- Chimie et Physique moléculaire (CMP)
- Énergie, Mécanique, Procédés, Produits (EMPP)
- Matières, Matériaux, Métallurgie, Mécanique (M4)
- Observatoire Terre et Environnement de Lorraine (OTELo)
- Sciences Juridiques, Politiques, Économiques et de Gestion (SJPEG)
- Temps, Espaces, Lettres, Langues (TELL)

Le pôle nous concernant est le pôle AM2I, qui est une des entités au dessus du LORIA.

1.2.5 Laboratoire lorrain de recherche en informatique et ses applications (LORIA)

Le LORIA est une Unité Mixte de Recherche (UMR 7503), qui comme évoqué précédemment est commune à plusieurs établissements : le glsCNRS, l'UL et INRIA.

Sa mission principale est la recherche fondamentale mais aussi appliquée en sciences informatiques. Cette mission lui a été assignée dès sa création en 1997.

Le LORIA est membre de la Fédération Charles Hermite¹ qui regroupe les trois principaux laboratoires de recherche en mathématiques et en informatique de lorraine. Cette fédération permet au Centre de Recherche en Automatique de Nancy (CRAN), à l'Institut Elie Cartan de Lorraine

1. Du nom d'un mathématicien français du XIXe siècle : Charles Hermite, connu pour ses travaux sur la théorie des nombres, les formes quadratiques, les polynômes orthogonaux, les fonctions elliptiques et les équations différentielles.

(IECL) et au LORIA de se hisser au 5e rang des centres universitaires nationaux pour les maths-STIC

Le laboratoire comprend un total de 450 personnes, ce qui en fait un des plus grands laboratoires de la région Lorraine. Les différents projets du laboratoire sont menés par 30 équipes, dont 15 communes avec INRIA.

Ces équipes sont organisées en 5 départements, comme le montre la figure A.1. L'équipe LARSEN est intégrée dans le département 5 : *Systèmes complexes, intelligence artificielle et robotique* .

1.3 Présentation de l'équipe

L'équipe LARSEN a été créée le 1er janvier 2015 à INRIA Nancy.

L'objectif de l'équipe est d'amener les robots ailleurs que dans des laboratoires de recherche et les industries. Pour atteindre cet objectif, l'équipe LARSEN développe des méthodes pour offrir une autonomie à long terme ainsi qu'une capacité d'interaction au robots, prenant en compte des capteurs embarqués ou externes placés dans l'environnement.

Ces capacités sont regroupées autour de l'interaction physique et sociale, de l'apprentissage et du planning gérant l'incertitude. Les expérimentations, concernant particulièrement le service et les robots d'assistance, sont au coeur de la méthodologie de l'équipe.

Les techniques développées par l'équipe vont potentiellement impacter le domaine de la robotique et ainsi catalyser les efforts afin de transférer les robots au sein de notre société

1.3.1 Organigramme

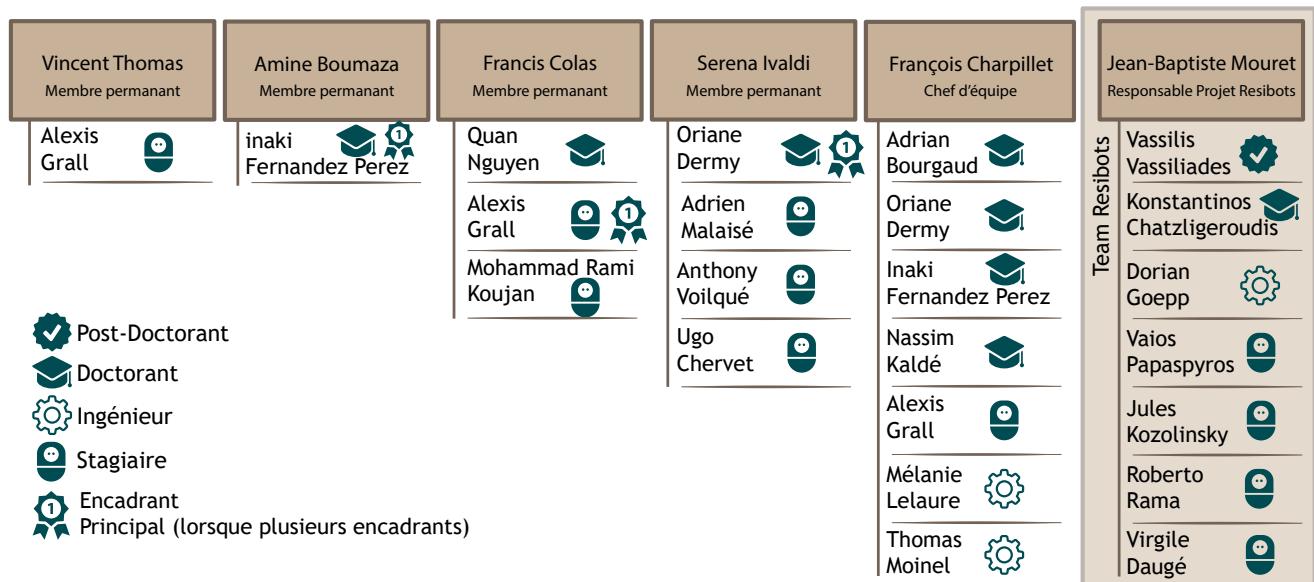


FIGURE 1.2 – Les membres de l'équipe LARSEN

1.3.2 Resibots

Le projet resibot est un projet européen, porté par Jean-Baptiste Mouret, et financé depuis 2015 pour une période de 5 ans par European Research Council (ERC). Il a pour but de combler les lacunes actuelles des robots en terme d'adaptation. En effet, à l'heure actuelle, malgré de nombreuses années de recherches dans le domaine, les robots sont toujours des machines fragiles, qui peuvent facilement s'arrêter de fonctionner lorsqu'elles subissent des imprévus.

L'objectif du projet est de fournir une base algorithmique pour la réalisation de robots économiques, capables de récupérer de dommages imprévus en quelques minutes.

L'approche actuelle de la tolérance aux pannes est héritée des systèmes critiques de sécurité (par exemple des vaisseaux spatiaux ou des centrales nucléaires). Elle est inappropriée pour les robots autonomes à faible coût car elle repose sur des procédures de diagnostics, qui nécessitent des capteurs proprioceptifs² coûteux, et des plans d'urgence, qui ne peuvent pas couvrir toutes les situations possibles qu'un robot autonome peut rencontrer.

Il est ici soutenu que les algorithmes d'essais et erreurs par apprentissage offrent une approche alternative qui ne nécessite pas de diagnostic, ni de plans d'urgence pré-établis. Dans ce projet, l'équipe développe et étudie une nouvelle famille d'algorithmes d'apprentissage permettant aux robots autonomes de découvrir rapidement les comportements compensant les imprévus. Les techniques développées dans ce projet augmenteront sensiblement la durée de vie des robots sans augmenter leur coût et permettront d'ouvrir de nouvelles pistes de recherche pour les machines adaptatives.

2. Des capteurs proprioceptifs permettent de connaître les caractéristiques intrinsèques du robot comme la position de ses membres, les couples appliqués par ses moteurs etc... il permettent donc d'avoir une connaissance avancée de son état actuel.

2 État de l'art

Rappelons la problématique de réalisation d'un contrôleur réactif prenant en compte les capteurs de forces aux extrémités des pattes. Afin de répondre au mieux à la problématique posée, il est nécessaire d'effectuer des recherches. L'idée était ici dans un premier temps de faire le tour des différentes approches actuellement utilisées, leurs résultats et enfin, si ces derniers paraissent concluants, d'étudier voire réaliser l'implémentation de l'approche prometteuse.

Il existe de nombreuses approches, mais il convient de les différencier en deux grande catégories avec des objectifs différents. La première sera présentée en 2.2 et concerne les approches biomimétiques, qui n'ont pas forcément pour but principal d'aboutir à une solution performante, mais plus de découvrir de nouvelles façon de procéder, inspirées du vivant. La seconde, abordée en 2.3 regroupe quant à elle des approches plus classiques sinon plus pragmatiques.

2.1 Méthode et outils

Il existe de nombreuses publications liées directement ou indirectement aux objectifs du stage. Afin d'être capable d'extraire des informations cohérentes et exploitables de cette masse de données, il était nécessaire d'adopter une méthode efficace afin d'éviter de perdre beaucoup de temps dans cette phase préparatoire. L'objectif de cette section est donc de présenter la méthode adoptée, ainsi que les outils utilisés au sein de cette méthode afin de faciliter le stockage et le tri des publications

2.1.1 Méthode

2.1.1.1 Accès aux informations

La phase de recherche de publications s'est effectuée principalement sur l'outil *Google Scholar* présenté en 2.1.2.1. Afin de trouver des articles connexes, et intéressants vis-à-vis de la problématique du stage, il a fallu utiliser des mots-clés pertinents.

L'accès à la plupart des publications scientifiques connexes était fourni et donc payé par l'INRIA. Certaines n'étaient néanmoins pas accessibles, et sans avoir à l'avance une idée fondée sur leur potentielle pertinence, il a fallu s'en passer. La richesse du catalogue de l'INRIA a tout de même permis de toucher un panel important de publications, cette limitation étant survenue un nombre très faible de fois.

Afin de conserver les références des publications trouvées, l'utilisation d'un outil spécialisé a permis de gagner beaucoup de temps. Plusieurs outils de ce type existent, j'ai choisi d'utiliser *Zotero* outil présenté en 2.1.2.2.

Une fois une publication connexe trouvée, il convient bien entendu de l'analyser, voyons maintenant comment.

2.1.1.2 Lecture rapide

Au vu du nombre important de publications dans le domaine visé, et de la particularité de la problématique du stage, il est primordial d'essayer d'évaluer la pertinence d'un papier dans le contexte précis de la réalisation demandée avant d'en lire tous les détails. C'est pour cette raison qu'une lecture rapide est préférable à une lecture approfondie dans un premier temps.

L'objectif de cette lecture rapide, est d'essayer de cerner plusieurs points dans la publication. De prime abord, il s'agit de comprendre le concept évoqué. Cela revient à répondre aux questions suivantes :

- Que font-ils globalement ?
- Quels objectifs se sont-ils fixés ?
- Quelle approche est mise en avant ?

Ces éléments permettront par la suite de classer les différentes publications en catégories, afin de pouvoir retrouver facilement les informations concernant chaque approche. Cette partie est abordée en 2.1.1.4.

Maintenant que l'on a déterminé l'approche du papier, il est important d'en évaluer rapidement la pertinence. Il s'agit donc de trouver les résultats parfois quantifiés qui sont souvent mis en avant vers la fin de la publication. En effet, l'approche peu paraître vraiment intelligente et intéressante mais finalement s'avérer décevante en termes de résultats pratiques.

Cette brève analyse des résultats est parfois difficile à effectuer car les résultats présentés sont dans certains cas incomplets, et souvent à l'avantage de la méthode. Ce qui est dans une certaine mesure légitime, mais qui peut vite s'avérer problématique, par exemple si les résultats proposés omettent un paramètre important pour notre projet.

C'est pourquoi il est bon de se baser sur d'autres sources que les publications elles-mêmes pour établir notre première impression sur les performances réelles d'une solution.

2.1.1.3 Vidéos

Un des avantages de la robotique, est le côté très visuel des choses. En effet, que l'on travaille en simulation ou directement sur un robot, il est possible d'analyser rapidement la performance d'un comportement simplement en regardant le robot effectuer ses commandes. Cela permet finalement, dans un contexte d'évaluation rapide et globale, de prendre en compte aisément de nombreuses variables, qui seraient certainement moins évidentes à appréhender dans un autre domaine moins visuel.

Vous l'aurez compris, une vidéo en dit long sur le comportement d'un robot, et présente l'intérêt de permettre moins de libertés concernant la mise en avant des résultats arrangeants. Bien qu'il soit toujours possible de réaliser une expérience dans des conditions idéales, on peut assez facilement voir par exemple qu'il n'y a pas d'obstacles, où que ces derniers sont dimensionnés pour faciliter leur franchissement en accord avec la géométrie et les capacités du robot testé. Cela donne donc souvent une idée du meilleur comportement atteignable par cette méthode, ce qui reste une information intéressante.

Lorsque le lien entre la vidéo et la publication est explicite, c'est un très bon moyen d'évaluation rapide des performances d'un algorithme. Dans les autres cas, il faut réellement être prudents sur les conclusions tirées du visionnage d'une vidéo.

2.1.1.4 Classification

La Classification des méthodes de contrôle n'a pas été aisée, car elles opèrent souvent à des niveaux différents. Certaines méthodes issues d'une catégorie seront donc amenées à être utilisées conjointement avec des méthodes issues d'autres catégories opérant à un autre niveau.

2.1.1.5 Lecture approfondie

Une fois la publication jugée suffisamment pertinente selon les méthodes rapides présentées précédemment, puis assignée à une catégorie, il convient de l'analyser plus en détail. L'idée est ici d'aller chercher dans les détails de la publication la compatibilité entre la méthode proposée et la problématique imposée.

En effet, de nombreuses raisons peuvent rendre incompatibles un algorithme avec la plateforme réelle du projet. Par exemple une méthode exploitant un degré de liberté élevé de chaque patte¹, qui pose problème si l'on ne dispose pas de pattes avec un degré de libertés suffisant. De manière plus générale si le robot utilisé comme plateforme d'expérimentation à une géométrie différente de celui sur lequel on travaille, et que l'un des concepts de la méthode proposée dans le papier s'appuie sur cette particularité physique. C'est le cas par exemple pour le robot *RHex*[46] qui a des pattes tournantes, et qui se contrôle par conséquent d'une manière diamétralement opposée à un hexapode classique.

Il est parfois ardu de comprendre l'intégralité d'une publication lors de la première lecture appro-

1. supérieur aux 3 degrés de libertés généralement disponibles sur les pattes d'un hexapode et sur notre hexapode.

fondie, cela entraîne quelques fois des lectures en cascade, une publication s'appuyant souvent sur un travail préliminaire déjà publié. Cette étape à donc consommé beaucoup de temps et d'énergie, au vu du nombre de publications disponibles.

2.1.2 Outils

La construction d'un état de l'art requiert une solide organisation, qui est rendue possible par l'emploi de nombreux outils. Il ne s'agit bien évidemment pas de présenter ici tous les outils utilisés mais de traiter les principaux, qui seront essentiels à la réalisation d'un état de l'art convenable.

2.1.2.1 Google Scholar

Google Scholar est un service de recherche d'articles universitaires, de thèse ou encore de livres scientifiques. Lancé en 2004, le service de Google prétendait déjà couvrir 85% des publications scientifiques. Il reste cependant difficile d'évaluer réellement sa couverture des œuvres existante. En effet, même s'il est avéré qu'il se base sur des éditeurs scientifiques, et des serveurs universitaires, aucune mention précise de ses sources n'est effectuée.

Cependant, certaines informations comme les travaux réalisés par des amateurs, souvent relayées via des blogs spécialisés, ne sont pas indexées ici. Ces sources supplémentaires peuvent parfois s'avérer utile, notamment car elles font la part belle aux détails de l'implémentation, souvent relayée au second plan dans les publications scientifiques.

2.1.2.2 Zotero

Au vu du nombre de publications à couvrir, l'utilisation d'un outil d'archivage était indispensable. Il existe de nombreuses solutions disponibles remplissant le rôle de collecte, d'organisation de citations ou encore d'export de sources.

Parmi ces nombreuses solutions, parfois payantes, Zotero et Mendeley se démarquent par leur complétude et leur gratuité. Ils proposent tous deux les fonctionnalités indispensables. Après avoir brièvement testé les deux concurrents, la facilité d'utilisation de Zotero due à son intégration dans le navigateur Web Firefox m'a conquise.

Cet outil est vraiment pratique, car en plus de la possibilité de stocker et de trier les publications enregistrables en un seul clic, ce dernier permet également d'exporter une bibliothèque sous divers formats. L'intérêt principal de cet export est le format bibTex, utilisé par la suite lors de la rédaction de ce rapport. Ayant de nombreuses références, c'est un gain de temps non négligeable.

2.2 Approches Biomimétiques

S'inspirer de la nature et du vivant pour concevoir de nouveaux types de contrôleur robotiques est en vogue actuellement. Il existe de très nombreuses expérimentations, principalement basées sur des réseaux neuronaux².

Les travaux présentés ici tentent de résumer brièvement et au mieux l'ensemble des approches de contrôles biomimétiques trouvées concernant des hexapodes. Ce n'est donc pas une liste exhaustive, et elle concerne uniquement les travaux accessibles lors de l'été 2016, date de réalisation du stage.

2.2.1 Le vivant comme inspiration

L'objectif des recherches présentées ici est ambivalent. En effet, il ne s'agit pas uniquement de mimer le comportement animal pour trouver un contrôleur efficace, mais il s'agit également de prouver qu'il est possible d'arriver à un contrôleur via ces approches. Pour reformuler plus clairement, atteindre un contrôleur performant est souhaitable, mais prouver qu'il est possible de copier le monde animal l'est tout autant.

L'intérêt de ces approches biomimétiques, comme le souligne une publication relativement ancienne de Cynthia Ferrell [16], réside dans la robustesse, l'adaptivité et le caractère versatile de la locomotion animale, notamment de celle des insectes.

L'intérêt de cette publication est qu'elle compare différentes approches, fait rare dans les publications que j'ai eu l'occasion de lire. La plupart se contentant de proposer leur approche, mettant en avant leurs résultats sans vraiment détailler les conditions des expérimentations.

2.2.2 Réseaux de neurones

2.2.2.1 Pourquoi les utiliser ?

Il est connu que le cerveau humain ou animal est capable de réaliser certaines tâches complexes aussi bien que le plus avancé des ordinateurs. Les phasmes sont par exemple capables de résoudre le problème complexe que pose la marche à l'aide d'un réseau de neurones relativement simple. Des modèles inspirés de la nature appelés réseaux de neurones artificiels (Artificial Neural Network ou ANN en anglais) ont été développés.

2. Un réseau de neurones artificiels, ou réseau neuronal artificiel, est un ensemble d'algorithme dont la conception est à l'origine très schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques

2.2.2.2 Les avantages de leur emploi

L'avantage principal des réseaux de neurones réside dans leur capacité d'adaptation. Un même réseau peut en effet accomplir une tache en s'adaptant à de nombreuses configurations différentes. Il est en revanche nécessaire de l'entraîner pour chacune de ces configurations. Les avantages d'un système basé sur un réseau de neurones sont les suivants :

- Ils peuvent potentiellement gérer une grande quantité de données avec une grande précision ;
- Ils sont potentiellement robustes au bruit, erreurs et autres perturbations ;
- Leur caractéristiques non-linéaires, distribuées, parallèles permettent d'en augmenter la performance.

Leur particularité réside dans le fait qu'ils peuvent apprendre de leur expérience. Comme énoncé précédemment, il faut les entraîner afin de leur permettre de trouver les bons paramètres permettant de réaliser correctement la tâche qui leur est affectée. L'inconvénient de ces systèmes et qu'il est difficile de les analyser, ils doivent donc être souvent considérés comme des boîtes noires.

2.2.2.3 Structure

Un ANN est constitué de noeuds, appelés neurones artificiels, qui échangent des informations entre eux à travers des connexions appelées synapses. Chaque neurone est défini par une entrée, une fonction d'activation, un biais d'activation ainsi qu'une sortie.

Chaque synapse reliant deux neurones est caractérisé par un poids. Le biais de chaque neurone ainsi que le poids de chaque synapse sont les paramètres à trouver. La fonction d'activation est bien souvent une simple fonction de seuil.

Cela peut paraître simple, mais en pratique les réseaux sont souvent complexes. Pour la reconnaissance d'images par exemple, plusieurs dizaines voire centaines de couches cachées peuvent être utilisées. Cela augmente donc considérablement le temps et les données nécessaires pour l'entraînement de l'ANN.

Sans aller si loin, en restant dans le contexte d'un contrôleur d'hexapode, certains exemples comme celui tiré de la publication de Manoonpong & Al[35] visible sur la figure 2.1 sont déjà avancés. Cette publication n'est pas unique en son genre et de nombreuses solutions du même type existent, chacun essayant des variantes afin de rendre le système plus performant. Nous allons donc maintenant aborder le fonctionnement de ce contrôleur.

2.2.2.4 Un exemple détaillé

Nous pouvons remarquer la présence de divers groupes de neurones différents sur la figure 2.1. Ils sont ici au nombre de trois :

- Un groupe Central Pattern Generator (CPG)
- Un groupe Phase Switching Network (PSN)
- Un groupe Velocity Regulator Network (VRN)

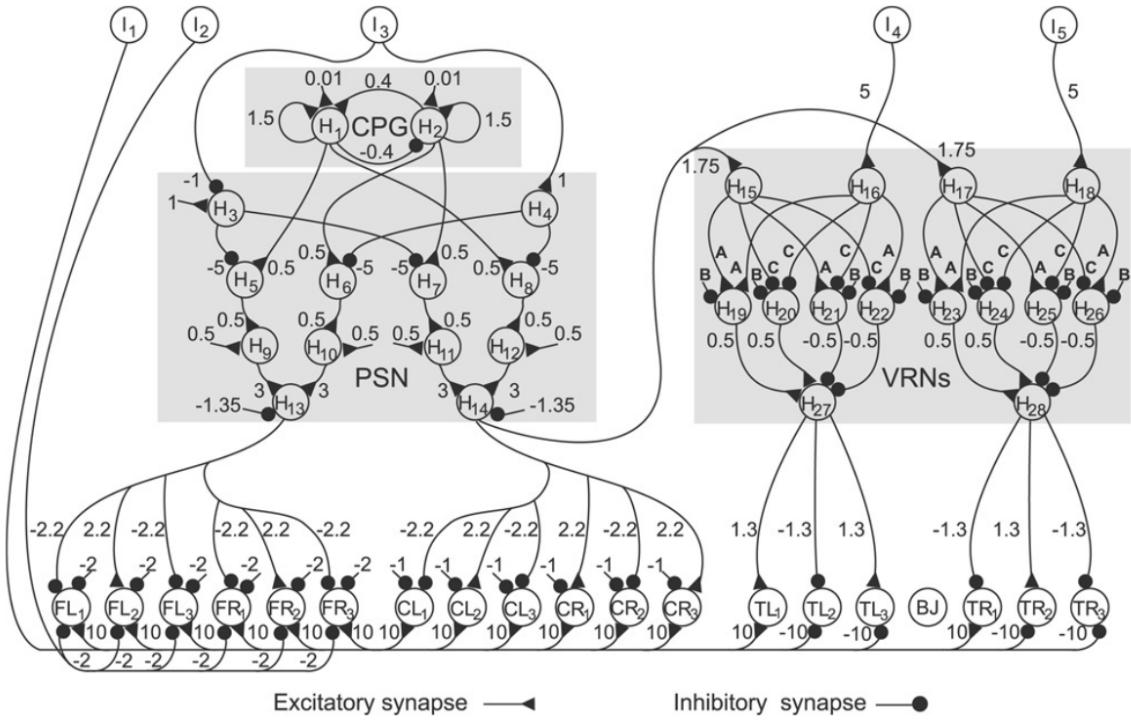


FIGURE 2.1 – Exemple de réseau de neurones d'un contrôleur [35]

Le premier (CPG) a pour but de générer une fréquence désirée afin de synchroniser la marche. Il est relativement simple, et ses poids ont été déterminés de manière empirique.

Le second, (PSN) doit quant à lui déterminer la phase de la patte : doit-elle supporter le poids ou avancer ? Il utilise pour cela la fréquence créée précédemment, ainsi que le neurone d'entrée I₃, qui servira à commander le sens des déplacements latéraux et diagonaux. Ses neurones de sortie contrôlent les servomoteurs des genoux et chevilles. Il a été construit à la main et ses poids choisis empiriquement.

Enfin, le troisième groupe, nommé VRN sert à contrôler la vitesse de la marche. Il se sert de la fréquence initialement générée par le premier et traitée par le second groupe afin de maintenir la vitesse désirée. Ainsi que des entrées I₄ et I₅ afin de déterminer le sens de la marche –Avant, arrière, tourner à droite, tourner à gauche...– en jouant sur le signe de ses derniers. Modifier leur amplitude augmentera la vitesse de marche. Ses neurones de sortie contrôlent quant à eux les servomoteurs des hanches.

La correspondance entre les neurones de sortie et les servomoteurs est donnée en figure 2.2.

2.2.3 Plusieurs niveaux de commande

Si certains travaux comme ceux évoqués précédemment[35] utilisent un réseau de neurones comprenant le contrôle simultané de toutes les pattes ainsi que leur synchronisation, d'autres travaux qui seront brièvement évoqués en 2.2.5 comportent quant à eux une approche de contrôle indépendant de chaque patte. Ce type d'approche paraît plus intéressant, bien qu'il faille certainement entraîner chaque patte en fonction de sa position sur le corps, et qu'il faille également intégrer un mécanisme de synchronisation.

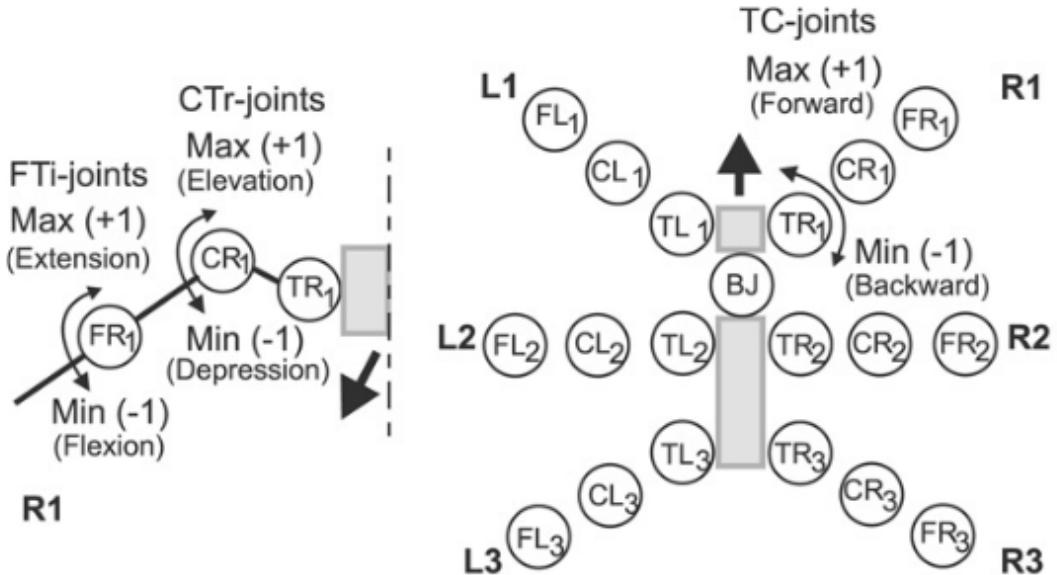


FIGURE 2.2 – Lien avec les servomoteurs[35]

Il est dans tous les cas nécessaire d'implémenter un mécanisme de contrôle des pattes ainsi qu'un mécanisme de synchronisation de ces dernières, les deux pouvant être plus ou moins liés.

2.2.4 Des comportements adaptatifs

L'entraînement doit se faire sur un set de données suffisantes, mais pas trop non plus, sans quoi on risque un surentraînement et le robot ne sera plus capable de généraliser et donc de s'adapter à des situations inconnues. Ce problème survient lorsque durant l'entraînement, l'on expose toujours le robot aux mêmes types d'environnements ou d'obstacles. Petit à petit, il va apprendre à gérer au mieux cet obstacle, jusqu'à potentiellement avoir un comportement optimal pour celui-ci. On peut même parler de comportement spécialisé pour cet obstacle. Il va donc prendre des habitudes qui fonctionnent bien dans ce cas précis, mais qui vont affaiblir le comportement du robot lorsqu'il se retrouvera dans un contexte différent. Il essaiera alors d'appliquer ces routines qui fonctionnaient si bien dans un contexte totalement inapproprié.

Même dans le cas d'une détermination manuelle et empirique des poids des synapses, il n'est pas garanti que le robot soit réellement capable de s'adapter à un cas particulier, non testé préalablement.

Les comportement adaptatifs, grandement mis en avant dans certaines publications ne concernent souvent que des variations mineures de l'environnement. Ainsi le robot bipède *runbot* dont le contrôleur est présenté de la publication *Adaptive, Fast Walking in a Biped Robot under Neuronal Control and Learning* [34], est capable de ne pas tomber lorsqu'il rencontre une rampe inclinée à 8° par rapport au reste du sol.

2.2.5 Des résultats décevants

Après avoir lu de nombreuses publications concernant ce type de contrôleurs ([43], [44] et d'autres) les résultats obtenus étaient souvent décevants au vu de la complexité du travail présenté. Après avoir visualisé quelques vidéos correspondantes sur Youtube, l'enthousiasme créé lors de la lecture des publications s'estompait rapidement.

En revanche d'autres approches paraissent plus prometteuses, comme celle présentée dans la publication récente *Adaptive and robust leg control of a hexapod robot for space exploration*[50]. En effet, après avoir entraîné leur contrôleur de patte sur un terrain plat, celui-ci reste robuste sur un terrain plus chaotique ou encore sur une pente. Ces résultats encourageants ne sont malheureusement pas suffisants pour opter pour cette approche, les travaux et expérimentations ayant été conduits sur une seule patte. Le contrôleur complet devrait être réalisé, mais à l'heure actuelle, rien ne permet d'affirmer que cette approche fonctionnera bel et bien une fois généralisée. De plus, cette approche prend en compte les retours des capteurs, mais très faiblement, et évite de développer cet aspect, qui est rappelons-le au centre de la problématique de ce stage.

2.2.6 Une approche inadaptée

De premier abord, une approche basée sur un réseau neuronal paraît intéressante. En effet, l'idée de construire un contrôleur simplement en s'inspirant voire en copiant une publication présentant son réseau de neurone peut paraître rapide et efficace. En réalité, le contrôleur sera rapidement implémenté, mais il nécessitera de très nombreux réglages, afin d'affiner les poids des différents synapses afin de l'adapter à notre robot.

Si l'on effectue un rapide bilan, peu d'approches biomimétiques donnent à l'heure actuelle une réelle importance aux capteurs, peu d'approches biomimétiques sont réellement robustes, peu d'approches biomimétiques proposent des résultats convainquants.

Si on ajoute l'importante difficulté de la mise en place d'un tel système, de son entraînement et le temps disponible pour mener à bien les objectifs du stage, opter pour une approche biomimétique ne semble pas être optimal.

2.3 Approches plus pragmatiques

Nous avons vu des solutions s'inspirant du vivant et copiant le vivant jusque dans la façon de procéder. Abordons maintenant des solutions plus pragmatiques, parfois inspirées de la nature, mais basées cette fois ci sur un socle de connaissances bien connu que forment la mécanique, l'automatique ou encore la physique.

Contrairement aux approches biomimétiques étudiées précédemment, les approches plus pragmatiques diffèrent tant dans leur logique que dans le niveau – contrôle d'une patte uniquement, planning, synchronisation des pattes etc – ce qui les rends parfois incompatibles et parfois complémentaires

A contrario des réseaux de neurones qui contrôlent tout les aspects de la marche à la manière d'une boite noire, il est ici nécessaire de maîtriser toutes les étapes nécessaires au bon fonctionnement du contrôleur dans son ensemble.

2.3.1 Le contrôle basé sur des pattes indépendantes

L'intérêt de contrôler les pattes séparément est double pour nous. En effet, en dehors des considérations traditionnelles que sont le découplage et la capacité de réflexe local, l'équipe resibots travaille sur des robots endommagés, avec un nombre de pattes variable. Ce type de contrôleur paraît d'emblée plus adapté à notre problématique.

2.3.1.1 Contrôle de trajectoire

C'est un thème peu voire pas abordé dans les publications scientifiques concernant les robots marcheurs. Globalement, soit l'on opte pour une logique de contrôle par position des moteurs, qui présente l'avantage d'être simple de mise en oeuvre et sûr. Cela représente néanmoins l'inconvénient de devoir penser nos mouvements selon la position angulaire des moteurs, ce qui n'est pas réellement naturel.

C'est une solution qui peut sembler suffisante dans un premier temps, mais qui montrera ses limites lorsque le besoin de trajectoires complexes se fera ressentir.

L'autre solution disponible est de travailler en logique cartésienne –c'est à dire de penser aux points que doit atteindre l'extrémité de la patte– puis de convertir ces points objectifs en commandes moteurs grâce à la cinématique. Cette solution sera retenue car malgré son éventuelle complexité de mise en oeuvre, elle permettra par la suite une grande liberté dans la génération des trajectoires de la patte. Elle sera présentée plus en détail dans la section 4.4.2.

2.3.1.2 Contrôle de la force

Le contrôle de la force se base sur différents types de capteurs, plus ou moins précis, mais donnant tous une idée de la force appliquée à l'extrémité de la patte. L'exemple le plus simple est présenté dans la publication *Force Threshold-Based Omni-directional Movement for Hexapod Robot Walking on Uneven Terrain*[26] semble être un bon point de départ.

Il s'agit ici de stopper le mouvement de la patte lors de la détection d'un contact avec un éventuel obstacle. L'objectif est d'éviter au robot de forcer sur cette patte après le contact, ce qui permet à la fois d'éviter que cette patte supporte une trop forte charge, et d'éviter que ce mouvement ne renverse ou déstabilise le robot.

Cette détection se fait grâce à un simple seuil de force. Une fois la force appliquée à l'extrémité jugée suffisante – au dessus du seuil fixé – on stoppe le mouvement et passe la patte en mode support, cessant alors de déstabiliser le robot et suivant son mouvement tout en servant d'appui. On passe donc grâce à ce procédé d'un comportement diminuant la stabilité du robot à un comportement l'augmentant.

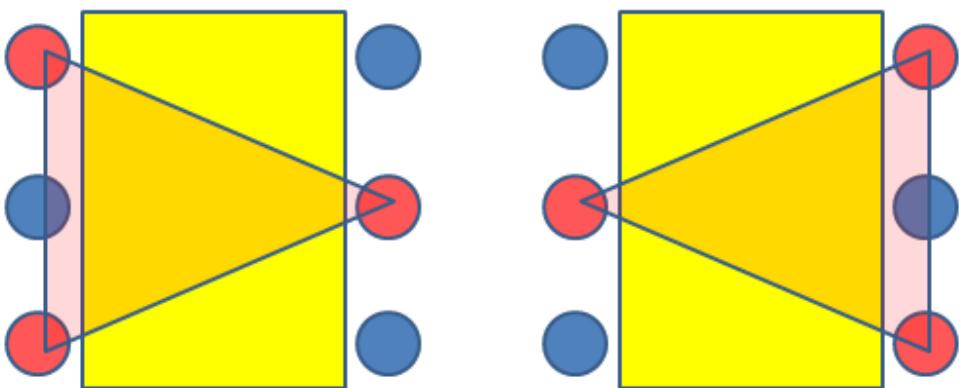
En affinant ce principe, il est possible de maîtriser la force de contact de chaque pied. C'est à dire d'adapter la position et posture de la patte afin de maintenir une pression constante. Cette possibilité est mise en avant dans la publication [30].

Cela apporte plusieurs avantages, d'une part, il est possible non seulement de s'assurer d'une portance suffisante, mais également d'éviter de trop forcer sur une seule patte évitant de l'enfoncer dans un sol meuble. C'est tout à fait approprié dans le cadre de leur étude, visant à faire marcher un hexapode au fond de l'océan. Dans tous les cas, l'apport de la maîtrise des forces appliquées est un apport conséquent et permet d'utiliser des techniques de synchronisation et de positionnement globales plus évoluées. C'est donc une solution vers laquelle on devrait tendre à terme.

2.3.1.3 Synchronisation entre les pattes

La synchronisation peut être totalement indépendante, mais est toutefois nécessaire afin d'obtenir un comportement cohérent. Il existe différents mécanismes de synchronisation, plus ou moins compliqués, souvent basé sur un *duty factor* de chaque patte, indiquant quelle proportion du temps celle-ci doit être posée. La synchronisation n'est pas au coeur de la problématique du stage, on peut donc se permettre de mettre en place un mécanisme de synchronisation simple, en permettant toutefois une certaine souplesse permettant de le remplacer facilement.

Le modèle choisi est un *tripod gait*, déplaçant 3 pattes lorsque les 3 autres sont en contact avec le sol. Il est simple à implémenter, et comporte l'avantage d'être toujours stable car la projection du centre de masse se trouve toujours dans le triangle de support formé par les pattes, comme illustré par la figure 2.3.



In the alternating tripod gait, the hexapod's legs are treated as two groups of three formed by the front and rear legs of one side, and the middle leg of the opposite side.

FIGURE 2.3 – La marche tripod

2.3.2 Le contrôle complet du robot

Une fois les pattes capables d'exécuter un déplacement précis, de maintenir une force désirée sur une position choisie, il devient possible de contrôler le robot à un plus haut niveau.

2.3.2.1 Contrôle omnidirectionnel

Proposer un contrôleur omnidirectionnel permet de révéler tous les avantages d'un robot sur pattes. Le papier *Towards Omnidirectional Locomotion Strategy for Hexapod Walking Robot* [48] présente différentes stratégies pour réaliser un contrôleur omnidirectionnel. Il présente en revanche des approches plus haut niveau, permettant par exemple d'éviter des obstacle. Notre contrôleur n'a pas vocation à évoluer à si haut niveau, il sera donc préférable d'implémenter uniquement la fonctionnalité de base permettant de se déplacer dans toutes les directions sans considération d'évitement.

2.3.2.2 Center Of Mass

L'approche basée sur le Center of Mass (CoM) permet d'augmenter la stabilité du robot. L'idée est de déplacer ce dernier en fonction des perturbations extérieures subies afin d'augmenter la stabilité du robot. On doit donc connaître la posture actuelle du robot ainsi que les forces qui lui sont appliquées. Cela est possible dans notre cas grâce aux codeurs de positions dont sont équipés les servomoteurs et des capteurs de force placés à l'extrémité de chaque patte.

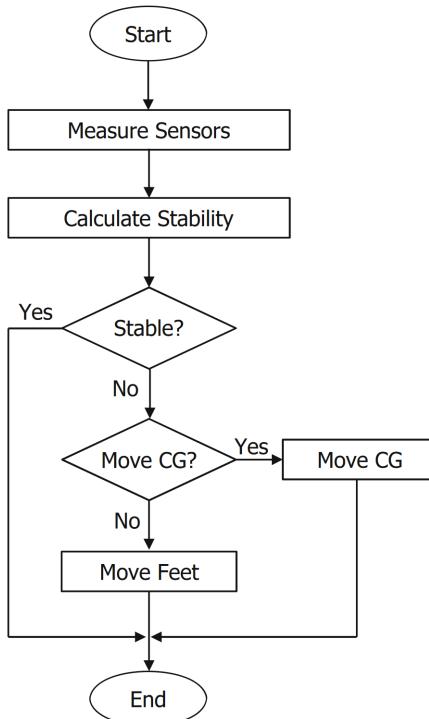


FIGURE 2.4 – Logique décisionnelle de l’approche basée sur le CoM

La publication *Foot Force Based Reactive Stability of Multi-Legged Robots to External Perturbations* [5] explique bien ce concept, et propose de calculer une carte des forces appliquées aux pattes afin de déterminer les perturbations extérieures, puis de les compenser soi en bougeant le centre de masse (auquel ils font référence sous le terme de centre de gravité : CG). Si changer la position du centre de masse sans bouger les extrémités des pattes ne suffit pas ils bougent ses extrémités afin de compenser au mieux. Ce schéma décisionnel est illustré par la figure 2.4.

2.3.2.3 Model Predictive Control (MPC)

Une autre approche consiste à utiliser une méthode maîtrisée et bien connue empruntée à l’automatique. L’idée est de posséder un modèle complet du robot, afin de pouvoir calculer l’impact d’une action sur ce système.

Ces conséquences sont souvent calculées sur plusieurs périodes de contrôle, afin d’avoir une idée des conséquences à long terme de l’action testée. la décision est ensuite prise d’effectuer ou non cette action sur la prochaine période de contrôle.

Puis on réitère l’opération à la fin de cette période. La publication récente *A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control* [41] détaille son emploi. Les capacités de cette approche semblent prometteuses.

Le soucis qui se pose ici et que si l’on maîtrise le modèle du robot, on ne connaît en revanche pas le terrain dans lequel il évolue. Il est donc difficile de prévoir le résultat d’une commande. Si, lors de la marche, l’on décide de poser le pied, le résultat final de cette action dépend de ce qui se trouve réellement au point de contact.

Sans un modèle de l'environnement, il paraît donc difficile de construire un contrôleur stable et robuste, en utilisant un MPC. Il faudrait alors y intégrer la notion d'incertitude afin de le rendre l'ensemble fonctionnel. Cela représente donc une possible évolution de notre contrôleur, mais semble un peu audacieuse pour une première version.

C'est la raison pour laquelle nous n'allons pas nous diriger dans cette voie pour l'instant. Opter pour une approche plus simple basée sur un simple seuil de force, couplé à des calculs cinématiques et un contrôleur omnidirectionnel paraît être une solution plus raisonnable étant donné les contraintes de temps et l'inexpérimentation du stagiaire en la matière.

3 Architecture

L'objectif de ce chapitre est de présenter les éléments nécessaires à la compréhension du travail réalisé qui sera présenté en 4. L'idée est de présenter dans un premier temps le matériel utilisé notamment la plateforme robotique sur laquelle sera implémentée le contrôleur à réaliser. Puis dans un second temps de proposer une entrée en matière sur les différentes composantes de l'architecture logicielle du système de contrôle de l'hexapode, afin de fournir tous les outils assurant au lecteur une bonne compréhension du travail réalisé lors de ce stage.

3.1 Matériel

Abordons maintenant une présentation succincte des principaux matériels qui seront utilisés lors de ce stage. Une première plateforme hexapode nomée PEXOD sera brièvement présentée, ainsi que la nouvelle version réalisée pendant ce stage comprenant quant à elle des capteurs de force. Le travail nécessaire à l'assemblage et au paramétrage de la nouvelle plateforme sera présenté en 4.3.

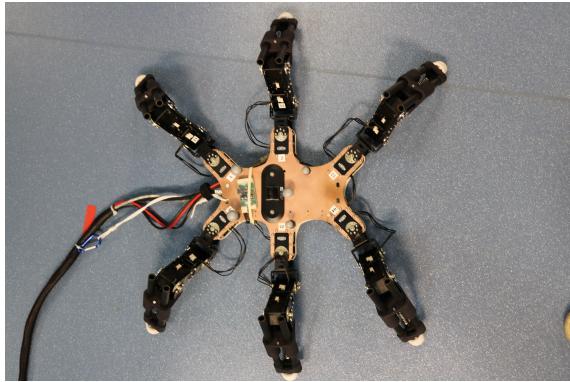
3.1.1 Robot petit Hexapode (PEXOD)

C'est la version originale du robot de l'équipe resibots, qui est visible sur la figure 3.1. Chacune de ses pattes comporte 3 degrés de liberté chacun actionné respectivement par un servomoteur (présenté en 3.1.3). Comme le montre la figure 3.1b, les 3 servomoteurs seront respectivement référencés par la hanche ou "hip", le genou ou "knee", et la cheville ou "ankle".

Le corps du robot, à été réalisé avec deux plaques de Printed circuit board (PCB)¹ percées afin d'y accueillir les servomoteurs des hanches. Il abrite également le module de communication Universal Serial Bus (USB) des servomoteurs.

Les pattes sont ensuite assemblées en fixant les différents servomoteurs entre eux. Enfin, l'extrémité de la patte est constituée de deux pièces imprimées en 3D, ainsi que de deux tubes en fibre de carbone servant de guides couplés à un amortisseur pour réduire la transmission des chocs au reste du robot, qui pourraient à long terme endommager sa structure.

1. Plaques servant à l'origine à imprimer des circuits électroniques à l'aide d'un procédé combinant une couche de protection sensible à la lumière ainsi qu'une exposition à l'acide, ne laissant que le design souhaité dans la plaque de cuivre.



(a) Vue complète

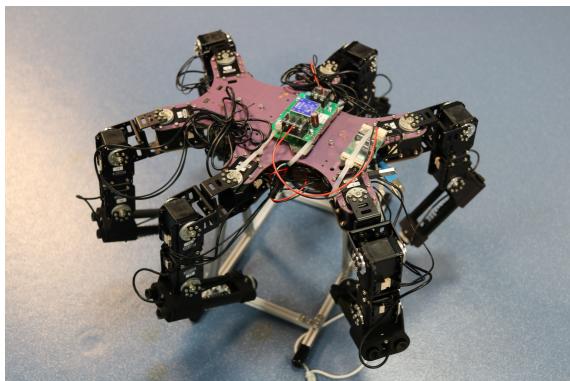


(b) Géométrie d'une patte

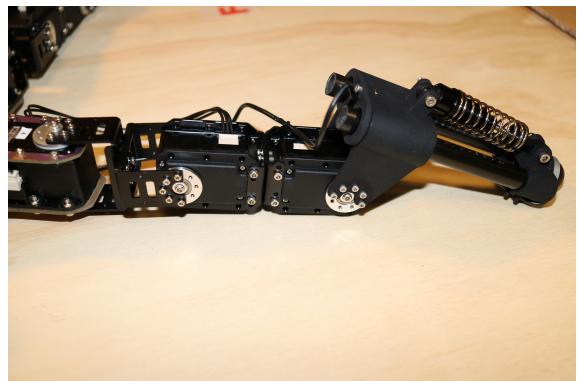
FIGURE 3.1 – Géométrie de l'hexapode d'origine

3.1.2 Nouvelle plateforme

La nouvelle plateforme, dont l'assemblage sera décrit en 4.3, n'est pas fondamentalement différente de la plateforme actuelle. La géométrie de chaque patte évolue, afin de permettre une plus grande liberté de mouvement, et permet également de se dispenser d'une pièce de liaison autrefois nécessaire. Cette nouvelle géométrie est visible sur la figure 3.2b.



(a) Vue complète



(b) Géométrie d'une patte

FIGURE 3.2 – Géométrie du nouvel hexapode

Elle prévoit également un emplacement ainsi qu'un passe câble pour l'ajout d'un capteur de force (présenté en 3.1.4) en extrémité de patte.

3.1.3 Servomoteurs

Les servomoteurs, modèles MX28-AT de la gamme Dynamixels proposée par l'entreprise Robotis, embarquent un correcteur PID². Le servomoteur, après réception d'une commande de position,

2. Correcteur à 3 paramètres, permettant de corriger la commande du système en fonction l'erreur constatée sur la mesure de position du servomoteur. Ces trois paramètre permettent d'agir sur la manière, plus ou moins brutale, rapide consommatrice d'énergie de corriger la commande.

de vitesse ou d'accélération, tente de l'atteindre le plus rapidement possible, puis une fois la commande atteinte, il tente de la maintenir grâce au correcteur.

Un encodeur absolu sans contact ($360^\circ/4096$)³ permet de récupérer l'information de position du servomoteur. Il sera très utile afin de modéliser l'état actuel du robot.

Ils supportent le mode de communication TTL, et sont regroupés dans un réseau de moteurs, relié à une carte de communication USB. Les commandes peuvent être envoyées à tous les servomoteurs en broadcast, ou à chaque moteur en précisant son identifiant.

Ils sont également résistants, et disposent d'une sécurité anti-casse appelée *Overload* (surcharge). Cette dernière empêche le servomoteur de forcer lorsque celui-ci est bloqué et demande trop de courant en entrée. Ce mode de sécurité est vraiment utile car il dispense, au moins pour les tests préliminaires, d'implémenter un système d'arrêt d'urgence haut niveau, qui de toute manière serait moins robuste et donc moins sécurisant pour les servomoteurs.

Lorsque l'un des servomoteurs est passé en mode Overload, il est alors nécessaire de le redémarrer. Il n'existe pas de solution logicielle, la procédure de redémarrage consiste donc à éteindre l'alimentation et à l'allumer à nouveau. Cela a pour conséquence de relâcher tous les moteurs, provoquant l'effondrement de l'hexapode.

Comme nous l'aborderons dans la partie 4.3, et comme nous pouvons le voir sur les figures 3.1b et 3.2b, les servomoteurs font partie intégrante de la structure du robot, ce qui simplifie son design et son assemblage.

3.1.4 Capteurs de force

Les capteurs de force sont de la marque *Optoforce*. Ce sont des capteurs semi-sphériques capables de mesurer les forces qui leur sont appliquées en trois dimensions. En revanche, dans cette gamme de capteurs, seule la composante *z* (perpendiculaire au disque de fixation et passant par son centre) est calibrée. C'est suffisant pour l'application que nous souhaitons en faire, la force la plus importante appliquée le sera souvent suivant cet axe. Vous pouvez voir sur la photo 3.3 l'intégration du capteur sur une patte.



FIGURE 3.3 – Vue du capteur de force situé à l'extrémité de la patte

3. Avec une précision de $360^\circ/4096$ donc $0,088^\circ$ largement suffisante pour notre application.

3.2 Robot Operating System (ROS)

L'architecture du système à réaliser est lourdement impactée par l'utilisation de ROS. C'est pourquoi il me semble indispensable d'en faire une brève description, ainsi que d'en exposer les différents concepts, afin de rendre plus aisée la compréhension de l'architecture du système ainsi que les différents travaux réalisés.

3.2.1 Brève description

ROS, acronyme de "Robot Operating System" est un framework flexible destiné à simplifier le développement de programmes pour robots. C'est une collection d'outils, de librairies et de standards visant à rendre aisée la réalisation de comportement de robots complexes et robustes, et cela sur de nombreuses plateformes robotiques distinctes.

ROS permet également d'unifier de nombreuses initiatives, qu'elles soient individuelles, issues de la recherche ou encore de l'industrie. En effet, cette base commune et ces standards permettent d'intégrer facilement des outils, programmes tierces que l'on aurait en temps normal besoin d'adapter, voire de reprogrammer. Cela permet donc de simplifier le développement de tâches simples pour un humain, qui se révèlent vite complexes à réaliser pour un robot.

ROS étant construit pour encourager la robotique collaborative, il permet en théorie d'utiliser n'importe quel programme et de l'intégrer facilement. Néanmoins, comme tout outil collaboratif, il reste imparfait et de nombreuses choses restent à améliorer.

En effet, ROS ajoute une sur-couche logicielle parfois lourde, autant en terme de compréhension que de temps de calcul. Plus d'informations à propos de ROS peuvent être trouvées sur leur site officiel⁴.

3.2.2 Concepts Principaux

Cette section présentera les principaux concepts de Robot Operating System (ROS), permettant de clarifier un peu le fonctionnement de ROS et donc de l'ensemble du système et donc du contrôleur à réaliser. Ce dernier s'appuie en effet sur d'autres Noeuds ROS déjà existants, son fonctionnement est donc fortement lié à celui de ROS, d'où l'importance de cette brève introduction au mécanismes de ROS.

4. Site ROS : <http://www.ros.org/about-ros/>

3.2.2.1 Noeuds ROS ou Nodes

ROS est composé de noeuds, qui sont en réalité des programmes développés en c++⁵, en python⁶, où encore en lisp⁷. Dans le cadre de ce stage, les noeuds seront développés en C++ pour un soucis de performance, et de conformité au travail de l'équipe. Néanmoins, certains noeuds utilisés ont été développés en Python, mais cela reste transparent pour nous, et ne change aucunement la manière d'interagir avec eux.

3.2.2.2 Communication inter noeuds : les messages

Les différents noeuds communiquent en s'envoyant des messages. Ces messages sont standardisés, il en existe de nombreux types, adaptés aux principaux capteurs et actionneurs utilisés dans le monde de la robotique. Ce sont des structures composées de types simples, cela ressemble beaucoup aux structures que l'on peut définir en C. Ces types de messages sont classés en catégories principales, données dans la liste ci-dessous :

- capteurs : sensors_msgs
- actionneurs : actionlib_msgs
- diagnostics : diagnostic_msgs
- messages standards : std_msgs
- navigation du robot : nav_msgs
- primitives géométriques : geometry_msgs

Vous pouvez voir en 3.1 un exemple de message, "JointState.msg" qui peut contenir toutes les informations relatives à l'état actuel d'un servomoteur. Dans le cadre du stage, nous n'exploitons que les positions des servomoteurs, car l'interface matérielle actuelle ne renvoie que cette information sur l'état du servomoteur.

```
1 std_msgs/Header header
2 string[] name
3 float64[] position
4 float64[] velocity
5 float64[] effort
```

Listing 3.1 – Contenu du JointState.msg

Vous avez sans doute remarqué que le message précédent contient un autre message, le Header, comprenant comme vous pouvez le voir en 3.2 un Identifiant de séquence, s'incrémentant à chaque nouveau message, ainsi que le temps correspondant généralement à l'écriture du message.

```
1 uint32 seq
2 time stamp
3 string frame_id
```

Listing 3.2 – En-tête d'un message

5. Standard c++ : <https://isocpp.org/std/the-standard>

6. Standard Python : <https://docs.python.org/3/library/>

7. Standard Lisp : <http://www.lispworks.com/documentation/common-lisp.html>

3.2.2.3 Communication inter noeuds : les Topics

Les noeuds communiquent entre eux via un mécanisme de publication/souscription. Un noeud envoie un message (voir 3.2.2.2) en le publant sur un Topic. Le Topic en lui même n'est qu'un nom, une adresse, permettant d'identifier le message et son contenu. Un noeud désirant récupérer une donnée en particulier doit souscrire au Topic correspondant. illustrons maintenant nos propos à l'aide de la figure 3.4. L'exemple basique de communication est représenté par le couple de noeuds (*Noeud₀*, *Noeud₁*) Le *Noeud₀* écrit sur le *Topic₀* tandis que le *Noeud₁* souscrit à ce même Topic. Plusieurs noeuds peuvent publier sur le même Topic (comme le *Noeud₄* et le *Noeud₂* écrivant sur le *Topic₂*)⁸, ou souscrire au même Topic (comme le *Noeud₅*), et un noeud peut publier et souscrire à plusieurs Topics. En général, le noeud publant et le noeud souscrivant n'ont pas connaissance de l'existence de l'autre noeud avec lequel ils échangent. Cela permet de découpler la production et la consommation de l'information, cela nous sera par exemple utile pour passer de l'environnement de simulation à l'environnement réel.

L'utilisation des Topics sera indispensable pour communiquer avec les différentes entités composant le système final.

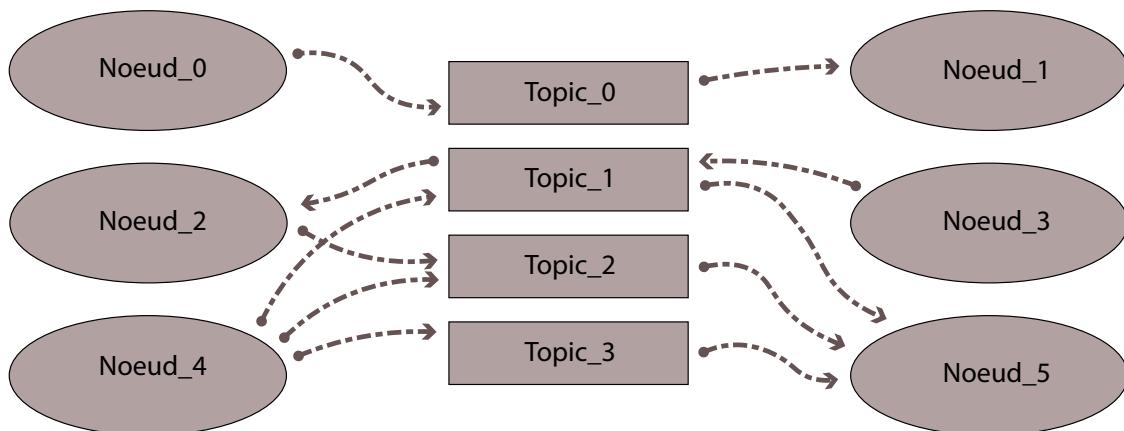


FIGURE 3.4 – Exemple d'utilisation des topics ROS

3.2.2.4 Communication inter noeuds : les Services

Les services permettent d'implémenter un mécanisme de communication Requête/Réponse, qui peut être utile dans un système distribué par exemple. Un service est défini par une paire de messages, un pour la requête et un pour la réponse. Un client envoie une requête et attend la réponse du noeud fournissant le service.

8. Ce cas de figure peut poser problème car il n'existe pas de mécanisme de protection de ressource commune, le *Noeud₅* qui utilise le *Topic₂* peut alors recevoir une information erronée.

3.3 Noeud Hexapod_description

3.3.1 Fichier URDF

Noeud Ros sert à la mise à jour et à la visualisation du fichier URDF⁹ contenant le modèle du robot. Ce dernier permet de stocker de nombreuses informations dont :

- les liens ou "links" : tout objet solide non animé.
- les articulations ou "joints" : liaisons entre les liens, dans notre cas les servomoteurs principalement.
- les formes destinées à la génération du visuel de chaque lien.
- les formes destinées aux calculs de collisions de chaque lien.
- les positions de chaque articulation, cela nous sera utile pour calculer la cinématique du robot.
- les capteurs embarqués, ainsi que le nom du Topic sur lequel ils devront écrire.
- les poids et centre de gravité de chaque élément.

Cette liste n'est pas exhaustive, il est également possible de stocker des meshs et d'autres informations. Un exemple tiré d'une jambe de notre robot est disponible en annexe B.1. Dans cet exemple, vous pouvez voir un joint et un link complets.

C'est un format de fichier basé sur le standard XML¹⁰, il est donc lisible facilement avec n'importe quel éditeur de texte, et il permet également d'étendre les balises disponibles afin d'ajouter des données correspondant à une nouvelle fonctionnalité.

Ce fichier de description du modèle de notre robot sera utilisé à la fois par le noeud ROS de simulation présenté en 3.6 et par le contrôleur présenté en 3.7.

3.3.2 Génération par Macro

Comme vous avez pu le remarquer, stocker ne serait-ce qu'un link et un joint associé demande de nombreuses informations, et créer ce fichier à la main se révélerait long et fastidieux. Fort heureusement, les données du modèle comportent de nombreuses redondance. Il est possible de générer ce fichier de description URDF à l'aide d'un langage de macro appelé Xacro¹¹. Nous l'utiliserons dans la partie 4.6, où seront présentés la syntaxe et le travail réalisé sur le modèle du robot.

9. URDF est l'acronyme de Unified Robot Description File.

10. Standard XML : <https://www.w3.org/TR/xml/>

11. Xacro pour XML macros : <http://wiki.ros.org/xacro>

3.3.3 Visualisation du modèle

Afin de valider le modèle du robot, il fallait un outil puissant capable à la fois de visualiser le modèle et de récupérer des données. Être capable enfin de changer les positions des servomoteurs permet de récupérer les positions des extrémités des pattes du robot dans une position particulière, ce qui sera vraiment utile dans la partie de calcul de la cinématique du robot en 4.4.2.

Fort heureusement, ROS propose un outil adapté à ces besoins, nommé Rviz¹². Il a fallu mettre à jour le fichier de description afin d'y inclure des "transmissions" permettant de manipuler directement le modèle en lui envoyant des commandes de position pour chaque servomoteur. Ces ajouts seront détaillés dans la section 4.6.

3.4 Noeud Hexapod_bringup

Ce noeud ROS permet de lancer l'interface matérielle avec les servomoteurs. Il contient de nombreuses informations utiles comme les corrections d'angles de chaque servomoteur, et permet également de lier les ids réels des moteurs avec des étiquettes, plus faciles à appréhender, permettant de savoir facilement à quel servomoteur on envoie une commande.

C'est un noeud qu'il convient de lancer avant chaque expérimentation sur le robot réel, il se contente d'attendre une commande de moteur sur un topic ROS particulier.

Seule la commande en position est possible, et la vitesse d'exécution de la commande est fixe, paramétrée à la vitesse angulaire maximale des servos-moteurs. Ceci a pour conséquence des mouvements pouvant être très brusques selon la différence entre la commande et le statut actuel.

Le contrôleur ROS utilisé par hexapod_bringup permet de gérer des trajectoires, ensembles de commandes associées à un temps donné. Ce contrôleur permet également de récupérer une information quant au déroulement de la trajectoire, permettant ainsi de savoir par exemple si le point donné est erroné¹³, s'il a bien été atteint, ou s'il n'a pas pu être atteint à temps. Cette information peut se révéler extrêmement utile, notamment lors des phases de déboggage.

12. Documentation disponible ici : <http://wiki.ros.org/rviz>

13. Plusieurs causes possibles comme par exemple un point qui se situerait avant le temps présent.

3.5 Noeud Optoforce

Ce noeud permet de communiquer avec un capteur de force. Il lance l'interface matérielle permettant de récupérer la valeur étalonnée de la force appliquée sur le capteur. C'est un noeud ROS fourni par le vendeur des capteurs de force.

Auparavant noeud indépendant qu'il fallait lancer pour chaque capteur de force, il a été modifié par Dorian Goep afin de l'intégrer dans Hexapod_bringup, qui s'occupe désormais de tout préparer afin de pouvoir récupérer les valeurs sur des topics aux noms contenant l'identifiant du capteur, qu'il faudra par la suite lier à la patte correspondante.

3.6 Noeud Rosdart

Noeud permettant de simuler l'hexapode dans son environnement grâce à la librairie Dynamic Animation and Robotics Toolkit (DART) tout en communiquant avec ROS permettant ainsi de ne pas changer le code entre les tests en simulation et les tests réels. Le noeud Rosdart se substitue aux interfaces matérielles de manière totalement transparente. Ce noeud n'existe pas lors de mon arrivée dans l'équipe. Sa création a été plébiscitée par le besoin d'un simulateur générique pour les robots contrôlés avec ROS. Nous pouvons en voir l'interface sur la figure 3.5.

Après avoir réalisé un prototype simple, un membre plus expérimenté de l'équipe, Konstantinos Chatzilygeroudis, a réalisé une version plus générique et plus puissante, permettant de gérer les capteurs de forces et de charger les contrôleurs directement depuis un fichier Unified Robot Description File (URDF) contenant la description du robot : ses liens, ses actionneurs, ses formes pour la visualisation, ses formes pour les collisions etc. C'est cette version qui sera utilisée durant le reste du stage.

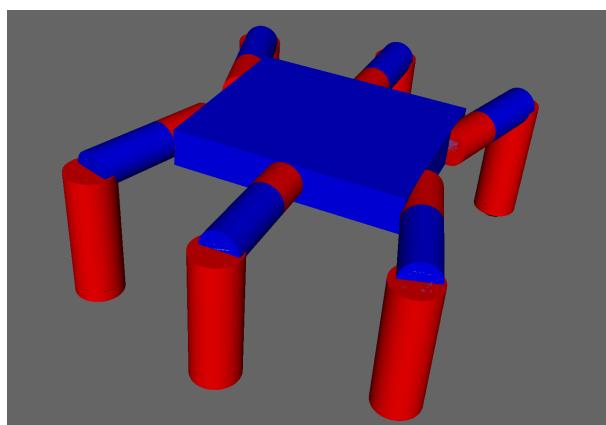


FIGURE 3.5 – Capture d'écran de l'interface graphique de RosDart

3.7 Noeud Hexapod controler

C'est le noeud destiné à contrôler le robot, qui devra envoyer des commandes aux moteurs en fonction des capteurs du robot et des entrées de plus haut niveau comme des commandes de direction, de vitesse ou de position. C'est l'objectif de ce stage, ce noeud sera réalisé entièrement lors de celui ci, il constituera donc le principal travail effectué et sera détaillé dans le chapitre 4.

4 Contribution

Les bases permettant une bonne compréhension du système et du projet ayant été présentées, nous pouvons maintenant aborder le cœur des travaux réalisés dans le cadre de ce stage.

L'idée ici est de présenter les phases de conceptions des différentes composante du système, les problèmes rencontrés, les méthodes utilisées pour les identifier ainsi que les solutions mises en place pour pallier à ces imprévus.

4.1 Principes de base

Le contrôleur se basera principalement sur différents concepts tirés des approches pragmatiques présentées en 2.3. Comme évoqué auparavant, les approches biomimétiques, bien qu'intéressantes, se révèlent peu documentées, et certainement trop complexes pour l'objectif à réaliser.

Combiner différentes approches de différents niveaux, basées sur des concepts pratiques et connus, devrait permettre d'atteindre l'objectif fixé dans le temps imparti.

Une mécanique de modules découplés sera également mise en place afin de rendre l'ensemble polyvalent.

Afin de simplifier la compréhension des mécanismes du contrôleur, nous nous référerons à la figure 4.1 dans cette section.

4.1.1 Les entrées du système

L'idée d'un contrôleur omnidirectionnel ayant séduit l'équipe resibots, ce dernier devra être capable de prendre en commande un vecteur de déplacement ainsi qu'une fréquence de déplacement¹.

Les codeurs de positions intégrés aux servomoteurs seront pris en compte par le système, notamment pour la génération de la trajectoire des pattes.

Les capteurs de forces situés à l'extrémité de chaque pattes doivent également être pris en compte afin d'adapter le comportement au terrain et ainsi le rendre plus robuste aux terrains difficiles.

1. Ici la fréquence de déplacement correspond au nombre de pas effectués par secondes.

4.1.2 Module de synchronisation : Hexapod

Comme vu précédemment dans de nombreuses publications issus à la fois de la biomimétique et des approches plus pragmatiques, il est bon d'avoir des pattes indépendantes afin de pouvoir découpler la synchronisation de la génération de trajectoires de chaque patte.

Nous aurons donc un bloc fonctionnel *Hexapod* chargé de la création des pattes, puis de la synchronisation de ses dernières en fonction de la fréquence de commande globale. Cela permet notamment de travailler avec un nombre variable de pattes, ce qui est utile dans notre cas pour simuler par exemple la perte d'une patte. L'autre avantage de ce découplage et qu'il permet de changer facilement le mécanisme de synchronisation des pattes pour implémenter une autre approche si cela s'avère nécessaire. Le bloc *Hexapod* a également pour rôle de récupérer les positions des servomoteurs et les valeurs des capteurs de force puis de les propager aux pattes concernées par ses informations. L'intérêt de récupérer ses valeurs ici et non directement dans les pattes est de pouvoir, si le temps nous le permet d'implémenter par la suite des algorithmes de type MPC. Ces informations seront en effet nécessaire à l'établissement d'un modèle du robot ou encore d'une carte des forces appliquées sur ce dernier.

4.1.3 Des pattes indépendantes

Enfin, pour compléter les rouages du mécanisme, les pattes jouent un rôle important. Étant indépendantes, elles comportent chacune une génération de trajectoires, un module de calcul de cinématique et un module de communication.

4.1.3.1 Module de génération de trajectoires

En effet, pour simplifier et surtout mieux maîtriser les trajectoires réalisées par les pattes, la génération de ces mêmes trajectoires se fera en coordonnées cartésiennes, et non dans l'espace de position des servomoteurs. Cela présente de nombreux avantages, le principal étant que raisonner en cartésien est plus naturel pour nous, nous permettant de déplacer la patte exactement comme on le souhaite. La génération des trajectoires sera présentée en 4.4.1.

4.1.3.2 Module de calculs cinématiques

On l'a vu, il est plus intéressant de générer les trajectoires en coordonnées cartésiennes. Seulement les moteurs, eux, doivent être commandés en positions angulaires. Il faut donc une connexion à double sens permettant de passer aisément d'un domaine à l'autre. C'est le module de calculs cinématiques présenté en 4.4.2, second élément composant chaque patte.

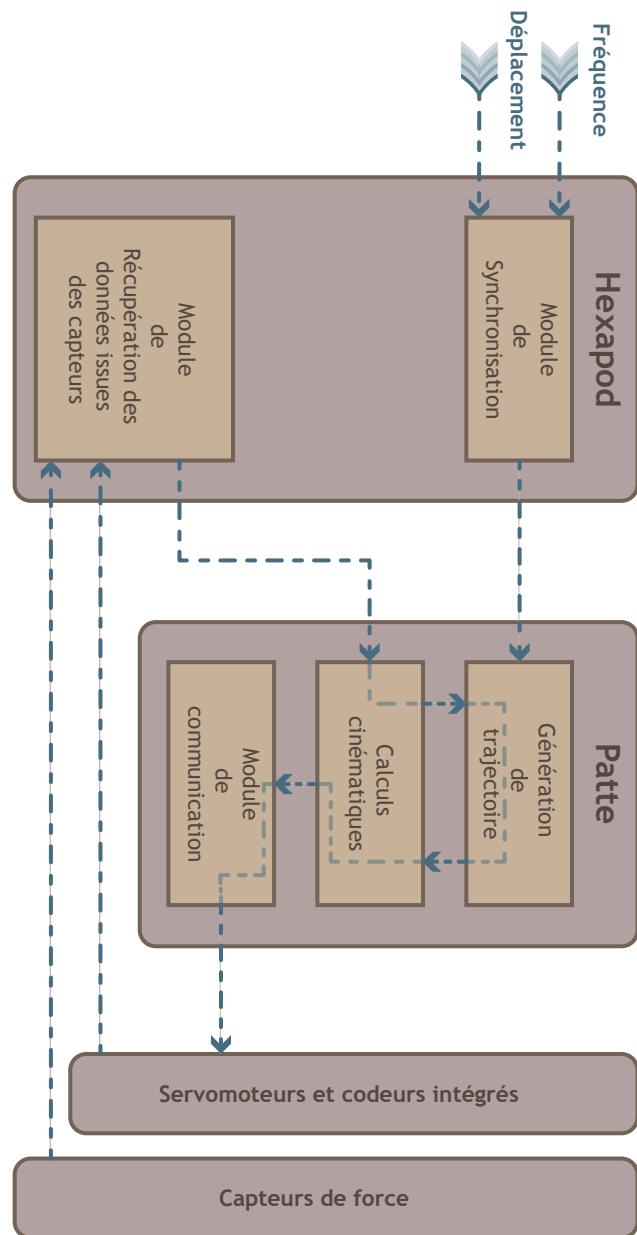


FIGURE 4.1 – Schéma global du contrôleur

4.1.3.3 Module de communication avec les servomoteurs

Enfin il est nécessaire de communiquer les trajectoires créées puis traduites en ordres aux servomoteurs. C'est le rôle du module de communication. Son rôle est assez explicite, et son fonctionnement sera présenté en 4.4.3.

Ces éléments, une fois réalisés et interconnectés, permettront à notre contrôleur de remplir les objectifs qui lui sont fixés. Rappelons : un nombre de paramètres d'entrée faible, une mobilité omnidirectionnelle, et enfin une prise en compte des capteurs de force afin de permettre une réactivité vis à vis du terrain.

4.2 Méthodes

La recherche impose une méthode particulière. L'incertitude quant aux résultats que produira une solution ainsi que le manque occasionnel d'informations sur l'implémentation de cette dernière peuvent conduire à une importante perte de temps. En effet, il est possible qu'en raison du manque de documentation, du manque de mise à jour d'une librairie, une solution autrefois fonctionnelle ne le soit plus, demande beaucoup de temps pour adapter son code aux éventuels changements d'API ou encore les deux. Nous en verrons deux exemples concrets dans la section 4.4.2.

Afin d'éviter de perdre trop de temps en essayant d'implémenter proprement une fonctionnalité issue d'une librairie/solutions incertaine, il convient de travailler au plus rapide, afin d'arriver à un résultat, positif ou non, dans les plus bref délais.

De prime abord confiant dans l'apparente simplicité de la tâche à réaliser, j'ai commencé par réfléchir aux bonnes façon de procéder, d'emblée prêté attention à l'optimisation, porté au final beaucoup d'attention mais surtout de temps dans la réalisation et le développement d'une solution, avant même d'avoir la certitude que cela allait fonctionner.

C'est un revers qui m'a permis d'en tirer une leçon, et de mettre à jour ma façon de procéder pour la suite du stage, me permettant par la suite un gain de temps non négligeable.

La méthode émergeant de ce problème est de réaliser un prototypage rapide afin de s'assurer rapidement de la validité pratique de la solution essayée.

4.2.1 Gestionnaire de versions

L'équipe utilise deux serveurs Git, l'un accessible à tous sur github² qui comprend donc les travaux suffisamment aboutis et non confidentiels pour être publiés librement. Elle contient donc le dépôt *hexapod_ros* contenant les noeuds *hexapod_description* et *hexapod_bringup*.

L'autre est un serveur interne au LORIA. Il nécessite une connexion locale ou le recours à un Virtual Private Network (VPN) afin d'y accéder. Il comporte quant à lui les travaux expérimentaux et/ou pas assez développés à l'heure actuelle. C'est là qu'est sauvegardé le noeud réalisé pendant ce stage (*hexapod_controller*) ainsi que le noeud *rosdart*.

4.2.2 Prototypage

Après les premiers écueils, le prototypage s'est rapidement imposé comme une évidence afin de sauvegarder du temps. L'idée est relativement simple, il s'agit ici de chercher à converger rapidement vers des résultats, afin d'identifier le potentiel d'une librairie. Il est en effet inutile de consacrer trop de temps à l'implémentation propre et commentée d'une solution si celle-ci s'avère inefficace ou pire inappropriée.

2. L'adresse est la suivante : <https://github.com/resibots/>

4.2.2.1 Réalisation du prototype

Le prototype est découpé du code existant, il est indépendant et testé sur un exemple similaire permettant toutefois de valider ou non sa pertinence. Ne pas l'intégrer au reste du code directement permet d'éviter de prendre du temps dans une architecture compliquée. L'idée est ici de "simuler" le contexte d'exécution désiré, d'effectuer un test puis de comparer avec les résultats attendus.

Ce découplage permet également d'ignorer dans un premier temps les éventuels problèmes de dépendance de librairies ou de compatibilité avec ROS. Ces problèmes sont bien évidemment contournables, mais l'idée ici reste de gagner du temps.

4.2.3 Intégration

Une fois une solution validée par un prototype, il convient de l'intégrer dans l'architecture existante, afin d'en valider la pertinence réelle. On est alors confrontés à différents problèmes d'intégration dans ROS, ou encore d'incompatibilité en l'état avec l'architecture.

4.2.3.1 Adaptation de la solution

De nombreux changements sont parfois à prévoir en passant d'un prototype codé rapidement et indépendamment à une implémentation plus propre et complexe au sein de l'architecture de notre contrôleur. Ce fut principalement le cas pour les solutions de calculs cinématiques vues en 4.4.2. La souplesse d'utilisation des librairies étant fortement variables, les adaptations nécessaires dépendent beaucoup de la solution, et peuvent nécessiter de faibles changements comme la réécriture d'une partie de la librairie.

4.2.4 Tests en Simulation

En plus d'une validation numérique chiffrée du comportement obtenu à partir d'une solution, il est important de valider le comportement du robot exploitant cette solution. Toutefois, dans le but de préserver le robot de comportements destructeurs due à des erreurs de calculs d'une solution inadaptée, il est conseillé de simuler l'expérience dans un premier temps.

Lors de ces tests il est également bon d'enregistrer des données comme les trajectoires générées, afin de pouvoir vérifier de manière précise que l'on a atteint le comportement initialement désiré. De plus, de nombreuses données ont parfois des influences trop minimes pour être répercutées de manière visible lors d'une marche rapide du robot.

Il est par la suite possible d'étudier ces données via ROSBAG par exemple qui permet d'enregistrer et de rejouer un scénario entier, comprenant tous les messages échangés sur les topics ROS.

Il est également possible de visualiser les données sur un graphe, en utilisant par exemple l'outil

matplotlib³

4.2.5 Tests Réels

Une fois le comportement validé en simulation (le comportement paraît bon, il n'y a pas d'auto-collisions⁴, pas de comportement destructif) il est important de tester le contrôleur sur le robot réel.

En effet, de nombreux paramètres ne sont pas pris en compte dans la simulation et d'autres divergent de la réalité. Comme par exemple la non-perfection des servomoteurs qui n'est pas prise en compte dans la simulation. Autre différence, les ressorts ne sont pas modélisés en simulation, ne permettant pas d'identifier d'éventuels problèmes de forces. Encore un point, et non des moindres, les servomoteurs sont considérés parfaits en simulation, et même si les modèles installés sur le robot sont de bonne qualité, il ne peuvent jamais atteindre exactement la commande demandée.

Pour ces raisons, il s'avère nécessaire de tester le comportement directement sur le robot, dans des conditions réelles d'utilisation. Cela n'empêche pas, au contraire de continuer d'enregistrer des données en vue de leur éventuelle étude ultérieure en cas de défauts apparents.

4.3 Assemblage du robot

4.3.1 Existant

Le design des pattes avait déjà été réalisé lors de mon arrivée, les servomoteurs étaient déjà arrivés, les pièces imprimées en 3D l'étaient également. Le corps du robot, fait d'une plaque de circuit imprimé était déjà découpé. Il restait juste à agencer l'intérieur du robot pour y ajouter les circuits de traitement des capteurs de forces, de l'IMU ainsi que les hubs USB et les hubs de puissance.

4.3.2 Mise en oeuvre des servomoteurs

Les servomoteurs sont paramétrables, et ont par défaut un identifiant égal à 1. Il est indispensable de les changer, il n'est pas possible de les distinguer et donc de leur envoyer des commandes différentes. Un utilitaire développé par l'équipe *libdynamixel*⁵ permet d'effectuer ce changement

3. matplotlib est une librairie open-source créée par John Hunter, elle permet la visualisation de données à travers de nombreuses figures différentes, comme des graphes, diagrammes, histogrammes etc...

4. Dans notre cas, une auto-collision peut survenir quand deux pas adjacentes reçoivent des commandes les rapprochant. Des valeurs limites de positions des moteurs sont définies, mais elles ne sont pas suffisamment restrictives pour éviter l'auto-collision dans un cas défavorable.

5. C'est une librairie prenant en charge les servomoteurs permettant de communiquer simplement avec eux via une interface en ligne de commande et proposant une interface c++. Elle est disponible sur le github de l'équipe : <https://github.com/resibots/libdynamixel>

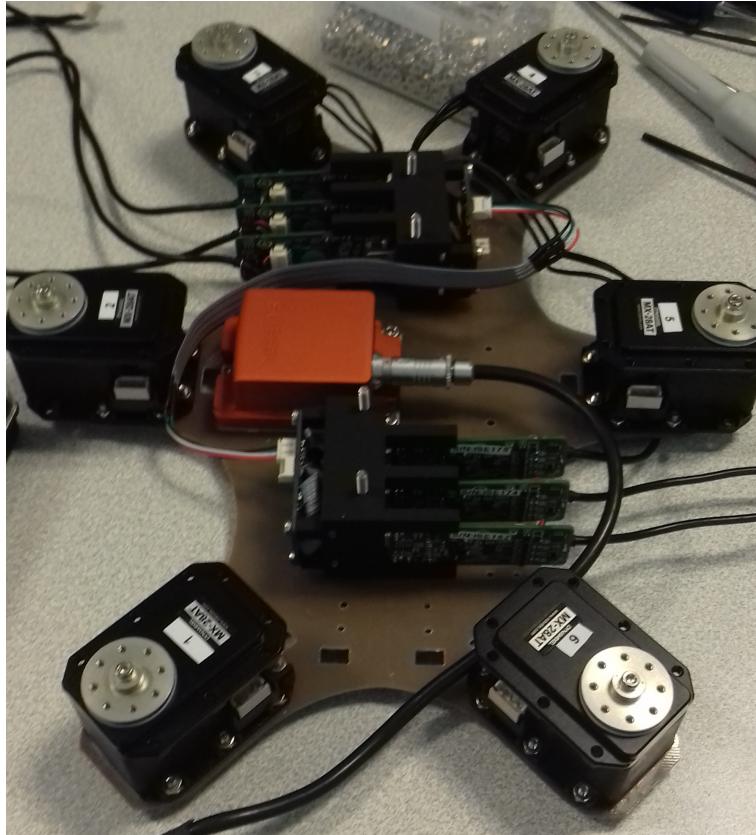


FIGURE 4.2 – Organisation interne de l'hexapod

d'identifiant par simple ligne de commande (qui ne sera pas détaillée ici car elle est encore sujet à changements). Il est néanmoins nécessaire de connecter les servomoteurs un à un. Afin de simplifier l'entretien du robot lors d'un éventuel démontage, chaque servomoteur a été étiqueté avec son identifiant. C'est une opération assez simple mais chronophage.

Après avoir changé les identifiants, il convient de changer la vitesse de communication des servomoteurs. En effet, sa vitesse par défaut n'est pas au maximum de ses capacités, ce qui réduit la fréquence maximale de contrôle possible. La capacité du bus USB étant suffisante pour les 18 servomoteurs réglés au maximum, ce maximum de 1 000 000 de bauds a été choisi.

Cette opération est réalisable en broadcast, en une seule ligne de commande, permettant de changer cette vitesse sur tous les servomoteurs.

4.3.3 Organisation interne

Avec l'ajout des capteurs de force et d'une centrale inertielles, les contraintes de place à l'intérieur du corps du robot sont devenues importantes. Il a donc fallu réfléchir à une organisation permettant de tout intégrer dans le corps du robot, tout en étant capable de retirer facilement les cartes de traitement des capteurs de force. Le design retenu est visible sur la figure 4.2. Nous pouvons y voir les servomoteurs des hanches aux extrémités de la carcasse, la centrale inertuelle au milieu et les hubs/racks USB pour les capteurs de force. Viendront s'ajouter par la suite les circuits de puissance. et le connecteur USB extérieur.

4.3.4 Mise en oeuvre des capteurs de force

Afin d'intégrer les capteurs de force dans le boîtier, il a fallu sortir les cartes de traitement de leur protection initiale en aluminium, afin de pouvoir les intégrer dans un rack spécialement conçu pour l'occasion par Dorian Goepp. Il a également fallu souder des connecteurs aux hubs USB afin de pouvoir réaliser la connexion par la suite.

4.3.5 Problèmes de hubs USB

L'alimentation par USB était trop peu puissante pour alimenter en simultané tous les capteurs de forces connectés. Il a donc été nécessaire d'ajouter une alimentation externe afin d'alimenter en conséquence les hubs USB. Cette alimentation externe est un régulateur de tension connecté sur le circuit d'alimentation des servomoteurs.

4.4 Réalisation du bloc patte

Nous aborderons ici la réalisation bloc fonctionnel indépendant d'une patte et des modules qui le composent.

4.4.1 Module de génération de trajectoires

Le mode de déplacement d'un robot/animal qui comporte des pattes peut se décomposer en deux phases à exécuter pour chacune de ses pattes, indépendamment du nombre de pattes ou de leur géométrie.

Dans le but de simplifier la compréhension du mécanisme, nous pouvons donc analyser notre propre façon de marcher. Chacune de nos jambes, passent tour à tour d'une phase de support, où notre pied est en contact avec le sol et supporte notre poids, tout en nous propulsant dans la direction du mouvement de notre corps, à une phase de "swing" où nous levons le pied afin d'atteindre la prochaine position d'appui.

Il suffit alors de bien synchroniser les jambes afin de garder l'équilibre pendant la marche.

4.4.1.1 Utilisation du patron de conception Stratégie

Comme vu précédemment, chacune des pattes de notre hexapode devra être capable d'effectuer différents types de mouvements, en fonction de la phase dans laquelle elle se trouve.

Dans un premier temps, seuls deux phases : support et swing seront utilisées. Mais dans l'avenir, on peut facilement imaginer qu'il sera nécessaire d'implémenter d'autres phases, comme par exemple comme la procédure de réaction à un contact prématuré (un obstacle) détecté à l'extrémité d'une patte.

Dans le but de pouvoir changer dynamiquement pendant l'exécution du programme l'algorithme de génération de trajectoire d'une patte, nous emploierons un patron de conception nommé Strategy pattern.

Comme l'illustre la figure 4.3, son principe est simple. Chaque objet de type patte se voit assigner une stratégie implémentant l'interface Strategy, offrant une fonction de génération de trajectoire ainsi qu'une fonction de vérification de la force appliquée sur le capteur de force.

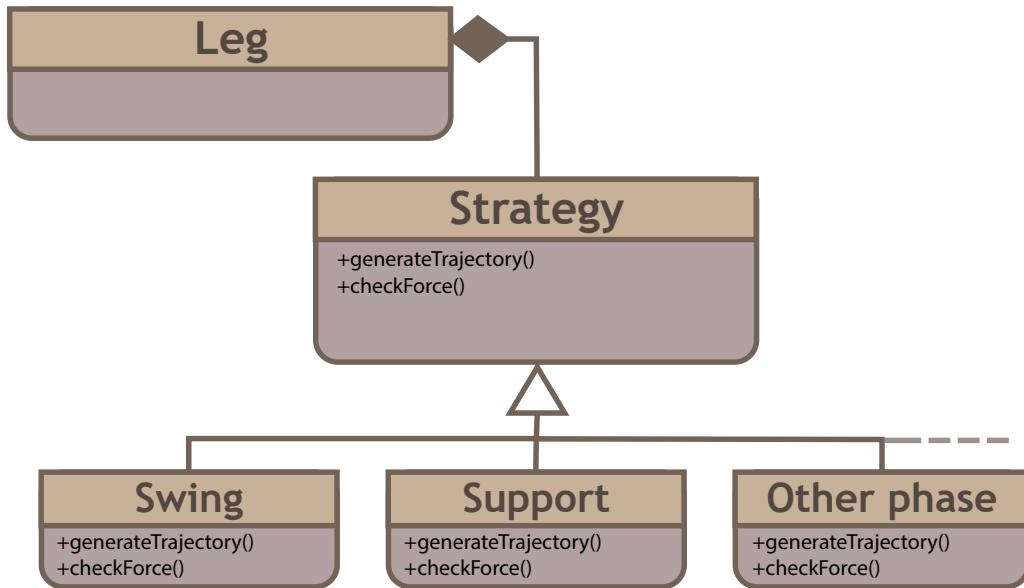


FIGURE 4.3 – Schéma UML de l'implémentation du pattern Strategy

Le changement de stratégie s'effectue soit lorsque l'un nouveau cycle commence, soit sur la base de la vérification de force. Notamment lorsque l'on touche un obstacle avant la fin programmée de la trajectoire.

L'avantage de ce mécanisme réside entre autres dans le découplage opéré entre la synchronisation globale et dans la réalisation des trajectoires.

4.4.1.2 Support

La patte en phase de support doit non seulement porter le robot, mais également permettre au corps du robot de suivre le mouvement des pattes qui sont en phase swing afin de le faire avancer. Il faut donc que le point de contact avec le sol reste fixe tandis que le corps du robot doit suivre la direction du mouvement.

Le principal problème ici est que l'on commande en position l'extrémité de la patte du robot et non le corps directement. L'astuce illustrée par la figure 4.4a est la suivante : afin d'effectuer le vecteur de déplacement souhaité (ici représenté en bleu) on utilisera le contact avec le sol. En maintenant ce contact avec le sol, et en comptant sur un minimum d'adhérence de la surface de contact, on fixe l'extrémité de la patte, qui permettra à la patte de déplacer le corps du robot d'un vecteur inverse du vecteur de commande bleu.

Il suffit alors d'inverser ce vecteur de commande ce qui donne le vecteur représenté ici en vert, afin d'obtenir le comportement désiré. Il suffit par la suite de calculer, à partir de la position actuelle qui peut être calculée⁶ à partir des positions des servomoteurs données par les codeur de position. On ajoute par la suite le l'inverse du vecteur de commande à cette position pour trouver la position, à atteindre à la fin de la trajectoire.

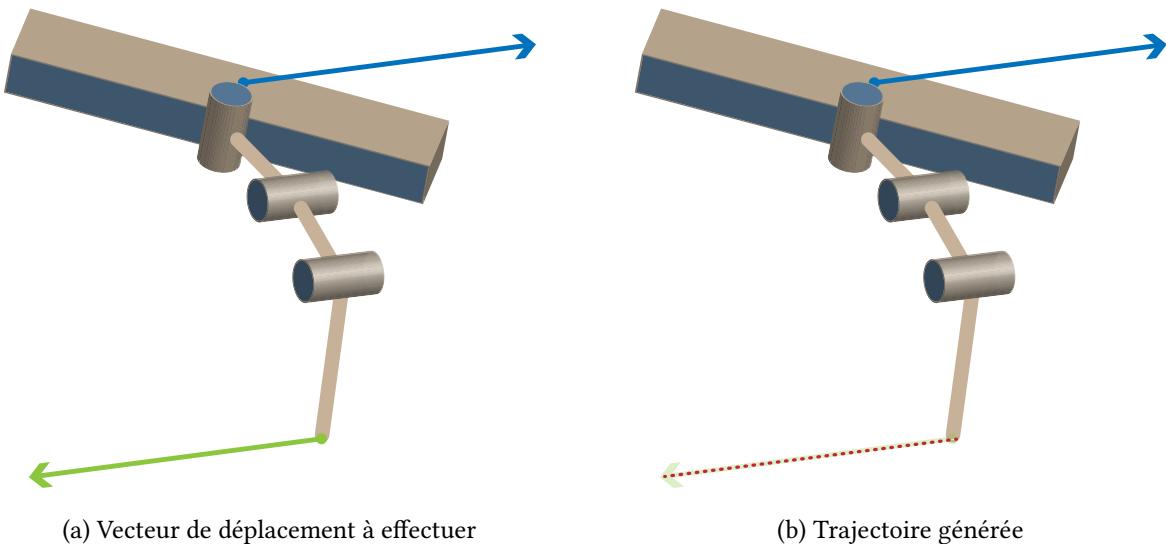


FIGURE 4.4 – Schéma de la génération de trajectoire support

Une fois l'origine et la destination de la trajectoire connue, il suffit de générer des points en n étapes, en ajoutant à chaque fois une fraction du vecteur de commande correspondant à :

$$objectif = position + \frac{commande}{n} \quad (4.1)$$

avec *objectif* qui correspond à la position à atteindre à la fin de l'étape, *position* qui correspond à la position de l'étape précédente, ou de la position actuelle de la patte dans le cas de la première itération.

Puis on répète cette opération n fois, afin de générer tous les points de la trajectoire représentés dans la figure 4.4b par les points rouges le long du vecteur vert.

Plus le nombre d'itérations n est grand, plus la trajectoire sera précise. En effet, comme l'on peut le voir sur la figure 4.5 si l'on donne directement la dernière position aux servomoteurs, il vont tenter de l'atteindre le plus vite possible. Sans forcément suivre une trajectoire rectiligne (ici une trajectoire probable est représentée en pointillés rouges. Comme vous pouvez le voir la trajectoire réelle empruntée par la patte dans ce cas est loin d'être rectiligne, ce qui peut provoquer des glissements lorsque plusieurs pattes d'un côté opposé effectuent leur phase de support. C'est pour cette raison que l'on crée une trajectoire avec plusieurs points intermédiaires, permettant de lisser la trajectoire réelle de la patte.

En pratique, il a été déterminé empiriquement que $n = 100$ était un bon compromis entre temps de calcul nécessaire et précision de la trajectoire. En effet, même si une simple suite de divisions puis d'additions n'est pas très gourmande en puissance de calculs, le passage du domaine cartésien aux

6. Grâce à la cinématique directe

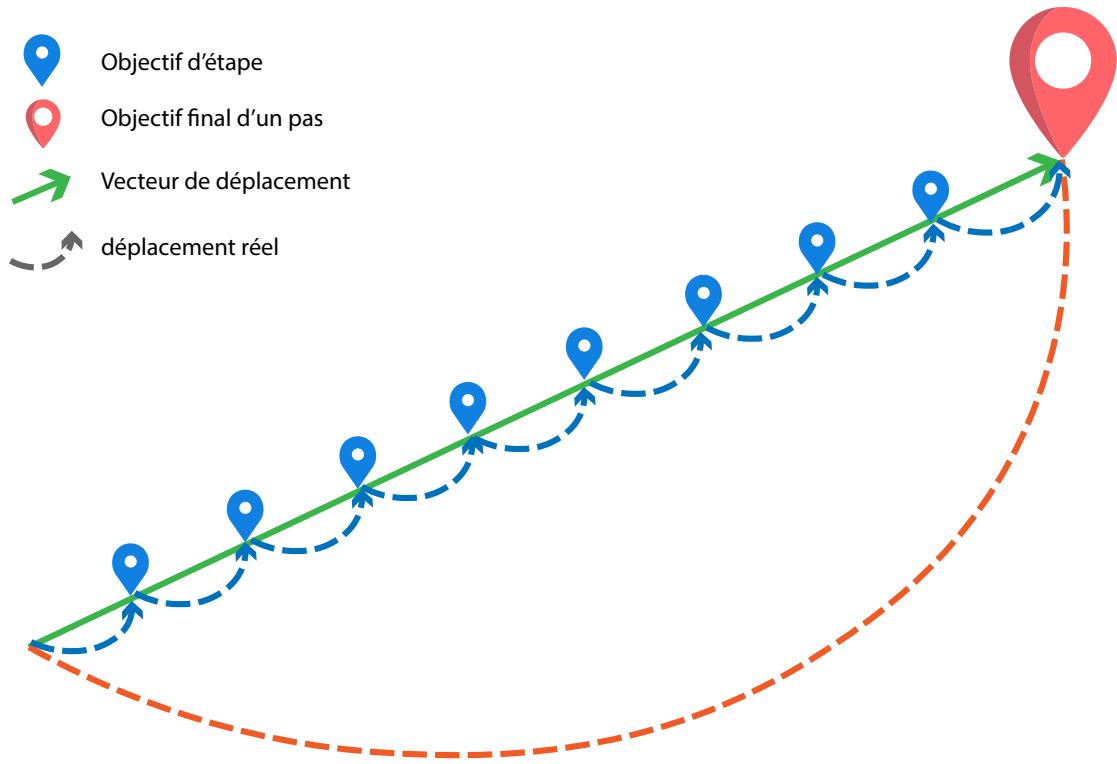


FIGURE 4.5 – Lissage de la trajectoire

commandes de moteurs est quant à elle plus exigeante comme nous le verrons dans la partie 4.4.2. Il convient donc de limiter le nombre de points générés afin de pouvoir travailler en temps réel.

4.4.1.3 Swing

Concernant la phase de swing, l'objectif est de faire avancer la patte du vecteur de commande. Pour x et y , il suffit alors de suivre ce vecteur. En revanche, il faut lever la patte afin d'éviter les frottements avec le sol. La composante z est alors augmentée progressivement jusqu'à atteindre le milieu de la trajectoire, puis diminuée jusqu'à atteindre la même valeur qu'au début de la phase de swing. La figure 4.6 illustre ces propos. Vous pouvez voir que pour pouvoir continuer le mouvement selon le vecteur de commande (ici en bleu), il faut déplacer la patte de ce même vecteur, on calcule donc l'objectif du pas en additionnant le vecteur à la position courante de l'extrémité de la patte, ce qui représente le vecteur de déplacement vert. La trajectoire ici générée est représentée en rouge.

La trajectoire en triangle ainsi générée avait l'inconvénient de faire glisser l'extrémité de la patte sur le sol, ce qui a pour conséquence de ne pas totalement respecter le vecteur de déplacement commandé. Il faut donc trouver une alternative, afin de générer une trajectoire plus adaptée sur l'axe z .

Il existe de nombreuses possibilités pour générer une trajectoire en deux dimensions sont infinies. Mais de quoi avons nous besoin ? Ces quelques critères ont permis de choisir une courbe plus adaptée : Il fallait que la trajectoire soit perpendiculaire au sol lors du contact, afin de mieux adhérer et ainsi éviter les glissements. Il fallait également que la patte se lève rapidement afin

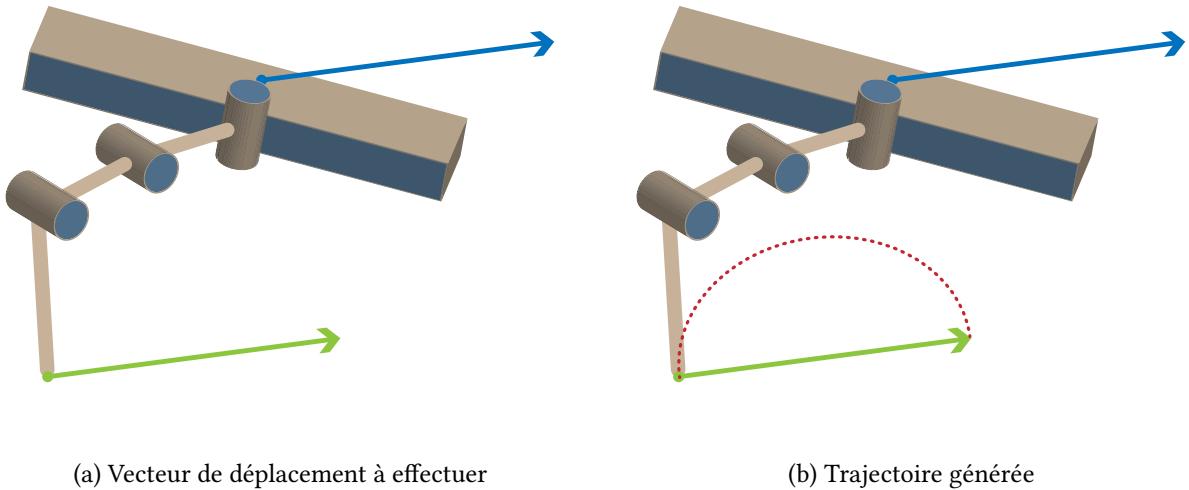


FIGURE 4.6 – Schéma de la génération de trajectoire swing

d'éviter naturellement de nombreux obstacles de petites dimensions qu'il aurait fallu gérer autrement. Enfin il fallait que le calcul soit simple et que l'équation de la courbe soit hautement paramétrisable.

Après quelques recherches, il semblait qu'une demi-ellipse soit adaptée à nos besoins en répondant à tous nos critères.

L'équation de la Courbe est la suivante :

$$zEtape = zMax * \sqrt{1 - \frac{k}{0.5 - 1} * \frac{k}{0.5 - 1}} \quad (4.2)$$

Ici $zMax$ permet de déterminer la hauteur maximale de la courbe et ainsi d'ajuster la hauteur de la patte selon nos besoins. k correspond ici au temps de l'étape. la figure 4.7 montre les courbes x (en bleu), y en vert, et z en rouge, qui correspond à l'ellipse décrite ci-dessus.

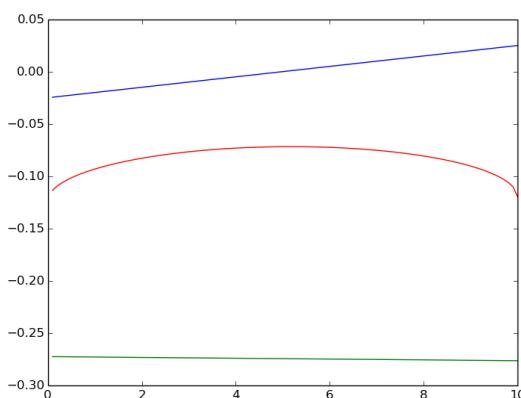


FIGURE 4.7 – Trajectoire elliptique générée

4.4.1.4 Correction du vecteur de déplacement

Après avoir implémenté et testé ces deux phases, il s'est avéré que les perturbations étaient conservées dans le temps, menant à une dérive. C'est logique car la position à atteindre est calculée à partir de la position actuelle, qui n'est pas exactement la position commandée à cause des perturbations extérieures comme la gravité.

Il faut alors trouver une solution afin de calculer le point objectif de manière absolue afin de compenser cette dérive quelques soient les perturbations extérieures. L'idée retenue, illustrée sur la figure 4.8 est de calculer l'objectif à atteindre à partir d'un point fixe dans l'espace.

Le choix le plus simple et logique était d'utiliser le point neutre de chaque patte, correspondant à la position cartésienne de l'extrémité de la patte lorsque les servomoteurs sont à leur position initiale. Ce point se calcule facilement grâce au module de cinématique directe présenté en 4.4.2. Une fois cette position déterminée, il suffit de lui ajouter la moitié du vecteur de commande afin de trouver la position à atteindre à la fin du pas.

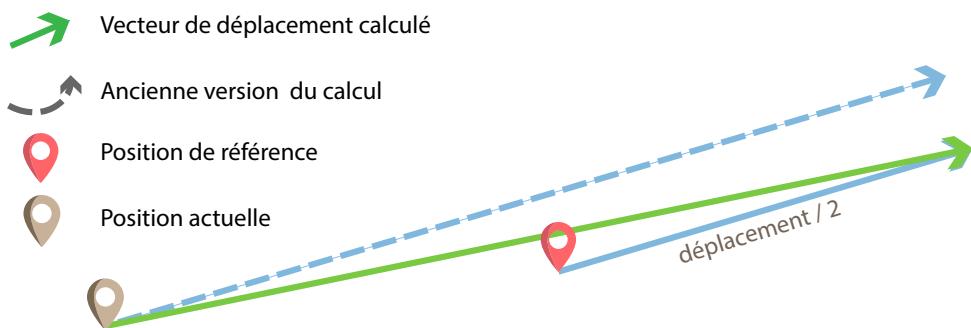


FIGURE 4.8 – Principe de la correction du vecteur de déplacement

Utiliser cette astuce permet donc de compenser les dérives dans le temps, mais n'évite pas les erreurs dues au correcteur intégré des servomoteurs. La figure 4.9 illustre la dérive de l'ancienne version et la façon de compenser décrite précédemment. On constate bien qu'il y a toujours une inévitable erreur, mais celle-ci est minime, et surtout elle ne se cumule pas. Cette erreur est donc rendue négligeable par la compensation.

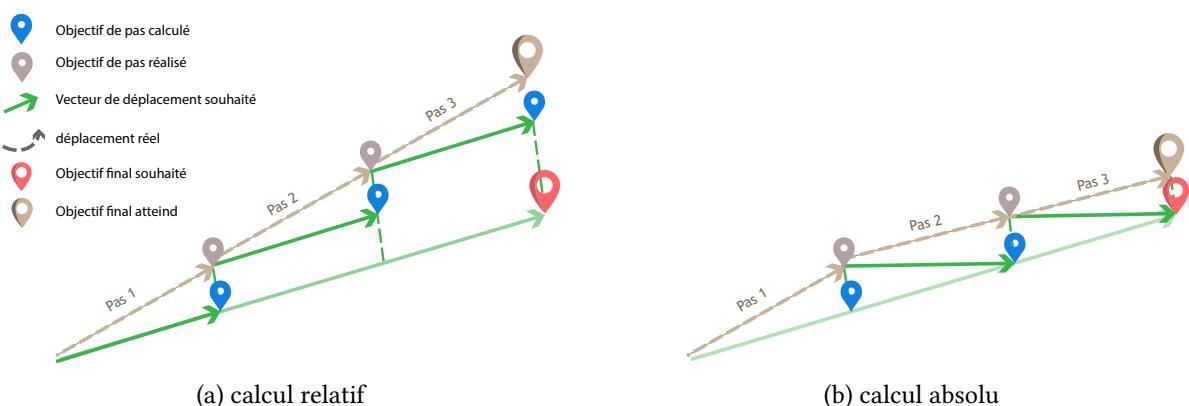


FIGURE 4.9 – Effets de la compensation par calcul absolu de l'objectif

4.4.2 Module de calculs cinématiques

Ce module à pour objectif de passer du repère cartésien au repère de commande des servomoteurs. le repère cartésien est représenté par un vecteur de coordonnées :

$$cart = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (4.3)$$

Tandis que le repère de commande est composé de trois angles en radians :

$$jnt = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \quad (4.4)$$

Différentes solutions ont été testées lors de ce stage avant d'aboutir à une solution adaptée et fonctionnelle, voyons maintenant lesquelles.

4.4.2.1 La cinématique en quelques mots

La cinématique est une branche de la mécanique classique, c'est plus précisément l'étude des mouvement dans le temps d'un robot indépendamment des causes. Elle permet de connaître la position, la vitesse ou encore l'accélération d'un point à un instant donné en fonction de la géométrie du robot. C'est avec cet outil que nous pourrons passer des coordonnées cartésiennes calculées précédemment aux angles des moteurs permettant de positionner l'extrémité de la patte à la position désirée.

La cinématique directe⁷ permet donc de passer du domaine de commande au domaine cartésien. C'est la plus facile des deux à calculer et de nombreuses librairies remplissent cette tâche.

La cinématique inverse⁸ permet quant à elle de passer du domaine cartésien au domaine de commande. C'est la partie qui nous intéresse le plus. Elle permet donc de trouver les trois angles des moteurs en connaissant la position cartésienne à atteindre. Cette dernière est en revanche plus compliquée à calculer.

4.4.2.2 Utilisation de KDL

Parmi les solutions disponible, Kinematics and Dynamics Library (KDL) paraissait adaptée à notre problème, et avait l'avantage d'être facilement intégrable sous ROS. De plus c'est une librairie assez populaire, ce qui laissait penser que l'on parviendrait à obtenir des résultats satisfaisants.

KDL à besoin de la chaîne cinématique, en d'autres termes de la géométrie du robot afin de pouvoir calculer la cinématique directe et inverse du robot. Plusieurs solutions sont possibles afin de

7. Souvent référencée en tant que "Forward Kinematic" ou FK

8. Souvent référencée en tant que "Inverse Kinematic" ou IK

renseigner cette chaîne. On peut la créer à la main à l'aide de l'objet KDL : `:Chain`. C'est assez rapide si l'on connaît la géométrie du robot. En revanche, si cette dernière est amenée à changer, comme ça sera le cas entre les différents modèles d'hexapods, il faudra la mettre à jour en dur dans le code à chaque modification de la géométrie du robot.

Fort heureusement, KDL met à disposition un parser permettant de récupérer cette chaîne cinématique à partir d'un fichier URDF présenté précédemment. Cela à l'avantage de permettre d'éviter d'avoir la géométrie du robot codée en dur. Nous aborderons le détail de cette implémentation qui sera gardée par la suite en 4.5.1.1

Par la suite, KDL propose des solveurs directs et indirects. Plusieurs implémentations sont disponibles, exploitant chacune un algorithme différent. Le solveur direct récursif fonctionne correctement, après vérifications sur un modèle simple de test puis sur le modèle de notre hexapod, nous obtenons des résultats précis⁹. Nous conserverons donc ce solveur direct.

En revanche, la plupart des implémentations du solveur indirect causaient des erreurs mémoire. Il a donc fallu tester avec valgrind afin d'en trouver une adaptée. Les tests sur le robot basique se sont avérés concluants. En revanche, les tests effectués sur notre modèle de l'hexapod l'étaient moins. L'un des deux algorithmes sans erreurs mémoire ne prenait pas en compte les limites des servomoteurs. Il trouvait donc des solutions inatteignables en réalité. L'autre algorithme, lui prenant en compte les limites, ne trouvait pas de solutions.

En effet, chaque patte de notre robot ne comporte que 3 degrés de liberté, ce qui réduit considérablement le nombre de solutions possibles pour atteindre un point donné. La plupart des utilisateurs de KDL utilisent cet algorithme sur des bras avec plus de 6 degrés de libertés. L'algorithme n'est donc pas adapté pour nos besoins car il existe bel et bien des solutions.

Cette partie a demandé beaucoup de temps et d'énergie car il a fallu débugger une librairie inconnue, à grands renforts de Vallgrind et gdb. La documentation proposée par KDL n'était pas correcte, forçant à écumer le web afin de trouver une solution viable. L'API ayant grandement évolué au cours du temps, de nombreux tutoriels de tiers étaient dépassés, conduisant régulièrement à une perte de temps.

4.4.2.3 Utilisation de IKFast

Une autre solution à première vue prometteuse est IKFast, proposée par la suite Open Robotics Automation Virtual Environment (OPENRAVE)¹⁰.

IKFast permet de générer des fichiers c++ sans dépendance à la librairie principale. C'est un avantage certain car cela permettrait de ne pas avoir à installer de librairie sur la machine faisant tourner le contrôleur. La solution finale serait donc plus légère, et beaucoup plus portable.

En revanche, OPENRAVE ne propose pas de fichiers compilés pour la version 14.04 d'Ubuntu (celle de la machine qui m'est attribuée). Il m'a donc fallu compiler cette librairie pour cette version d'Ubuntu, en suivant un tutoriel dédié présent sur la page github d'OpenRave. Quelle ne fut

9. Les résultats sont précis à un dixième de millimètre près ce qui est amplement suffisant dans notre cas car bien plus précis que les servomoteurs eux mêmes.

10. Openrave est une librairie complète visant à aider à tous les niveaux de la réalisation d'un système robotisé.

pas ma surprise quand, après avoir exécuté à la lettre les installations requises, et avoir compilé pendant un long moment la librairie elle même avant de pouvoir l'installer, l'exemple de base destiné à vérifier le bon fonctionnement retourna une erreur de segmentation.

Après de longues recherches sur internet, couplées à des essais infructueux tirés de tutoriels de tiers donnant les astuces pour faire fonctionner OpenRave sur la version 14.04 d'Ubuntu, je dus me rendre à l'évidence. Il n'était pas possible à l'heure actuelle d'installer l'intégralité de la suite OpenRave en l'état. Je me suis donc mis en quête des parties utiles de cette dernière pour réaliser la cinématique inverse. Après avoir désactivé la compilation de tous les autres modules, j'ai enfin obtenu un module certes limité, mais fonctionnel.

Une fois de plus, l'import de la chaîne cinématique depuis un fichier de description URDF est supportée. Cela a permis d'importer rapidement le modèle et de commencer les tests aussitôt.

Après quelques tests fructueux sur le modèle simple évoqué précédemment, les tests effectués sur le modèle de notre hexapod n'aboutissaient à aucune solution. Les autres solutions trouvées sur internet semblent similaires, et le temps consacré à la cinématique n'a déjà que trop augmenté. Il faut donc trouver une autre solution, une solution qui donnera un résultat dans tous les cas, quitte à manquer un peu de précision.

4.4.2.4 Crédit de mon solveur

La solution retenue afin de résoudre le problème de cinématique inverse est de la traiter comme n'importe quel problème d'optimisation. L'idée est ici de minimiser la distance entre le point cible et le point réellement atteint.

L'équipe ayant déjà travaillé avec la librairie d'optimisation NLOpt. L'avantage de cette librairie réside dans son encapsulation de nombreux algorithmes permettant en changeant une seule variable de tester un autre algorithme. Cela permet de trouver rapidement un algorithme d'optimisation adapté à notre solution sans pour autant devoir les réimplémenter systématiquement.

Il est important de préciser que l'on doit travailler dans deux domaines. En effet, la fonction de valeur passée à l'algorithme est calculée à partir des coordonnées cartésiennes, tandis que l'algorithme tente quant à lui de trouver la bonne valeur de position angulaire des servomoteurs, permettant d'aboutir à cette minimisation.

la première étape de la fonction de valeur consiste à passer de la commande en position angulaire calculée par l'algorithme en coordonnées cartésiennes de l'extrémité de la patte. Cela correspond à la fonction Fk présentée en 4.5. On utilise ici la fonction de cinématique directe de KDL introduite précédemment.

$$[x \ y \ z] = Fk(\theta_1, \theta_2, \theta_3) \quad (4.5)$$

Une fois ces coordonnées obtenues, on peut calculer la distance entre le point actuellement testé et la position que l'on cherche à atteindre. Cette fonction de valeur correspond à l'équation 4.6

$$resultat = \sqrt{(x - x_{ref})^2 + (y - y_{ref})^2 + (z - z_{ref})^2} \quad (4.6)$$

NLOpt utilise donc cette fonction de valeur pour estimer la pertinence d'une solution, et modifie cette solution en conséquence d'une manière différente selon l'algorithme utilisé.

De nombreux algorithmes sont disponibles, ils sont regroupés en trois catégories :

- Optimisation globale
- Optimisation locale
 - avec gradient
 - sans gradient

Un algorithme global va parcourir l'ensemble de l'espace de solution afin de trouver le minimum global et d'éviter les minimums locaux. En revanche, il sera moins précis sur la localisation précise de ce minimum. C'est pourquoi il est recommandé dans le but d'affiner la recherche de lancer un algorithme local avec comme position de départ la sortie de l'optimiseur global. L'inconvénient de cette solution est donc un temps de calcul plus élevé, surtout en considérant que l'on ne veut pas forcément la meilleure solution possible mais une solution rapide respectant nos critères de précision. Nous n'utiliserons donc pas d'algorithme global.

Nous utiliserons la version sans gradient car il n'est pas évident – voire pas possible – de calculer le gradient de notre fonction de valeur qui comprend un calcul de cinématique directe. Les différents algorithmes de cette catégorie ont tous été testés afin d'en déterminer le plus performant pour notre problème.

4.4.2.5 Réglage du solveur

Après avoir testé la génération de trajectoire puis la commande des moteurs via la cinématique inverse, il s'est avéré que la patte effectuait bien le mouvement désiré, mais en tremblant. Après avoir récupéré les données à la sortie de l'algorithme de génération de trajectoire, et vérifié leur exactitude, le problème doit se situer en aval. Deux hypothèses s'offraient alors, soit le solveur de cinématique inverse n'est pas assez précis, soit les servomoteurs n'arrivent pas à suivre une commande aussi précise.

Il a donc été nécessaire d'ajouter un module d'enregistrement des commandes envoyées aux servomoteurs, afin de pouvoir vérifier la validité de celles-ci. L'idée est de récupérer les trajectoires contenant les ordres aux moteurs ainsi que les temps associés à chaque ordre, juste avant leur envoi, permettant ainsi de les écrire dans un fichier nommé selon la phase, le numéro de la patte et le numéro du pas : *JNT_leg_0_traj_5.swing* par exemple.

Par la suite, il suffit de représenter les données dans un graphe, ce qui a ici été réalisé avec iPython¹¹. La figure 4.10a illustre bien ce problème de tremblement. La courbe bleu représente la commande de la hanche, la verte du genou et la rouge de la cheville. On voit bien ici que les valeurs changent parfois grandement d'un temps à l'autre ce qui provoque ses tremblements. Savant que la courbe générée avant passage dans le domaine de commande est parfaitement lisse, le problème vient donc du solveur de cinématique inverse.

Ma première hypothèse fut que la précision de l'algorithme d'optimisation était insuffisante. L'étude de la documentation de NLOpt a permis d'identifier la manière de régler la précision.

11. iPython offre un support pour la visualisation de tous types de données via une interface en ligne de commande (utilisée ici) ou des GUI. <https://ipython.org/>.

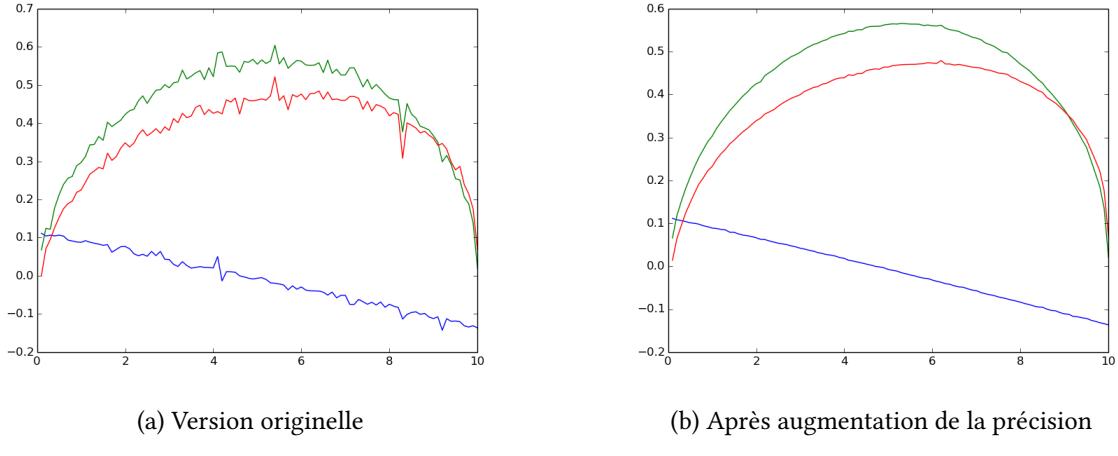


FIGURE 4.10 – Trajectoire traduite par le solveur de cinématique inverse

Elle était par défaut réglée sur 10^{-3} , et la passer à 10^{-4} a suffit à régler le problème, comme vous pouvez le voir sur la figure 4.10b (le code couleur reste identique).

Comme on peut le voir, la précision des calculs est tout à fait suffisante pour notre application, les commandes sont passées en centimètres et la précision est de l'ordre du millimètre.

4.4.3 Module de communication

Le module de communication se contente d'envoyer les trajectoires aux contrôleurs de trajectoires lancés par le noeud *hexapod_bringup*. Auparavant appartenant au bloc Hexapod dans les premières versions, il est désormais intégré aux pattes, leur permettant d'effectuer des trajectoires complémentaires de compensation par exemple, indépendamment de la synchronisation globale.

Pour cela, il faut créer un client qui se chargera d'envoyer les trajectoires et d'attendre les réponses du contrôleur de trajectoire lancé d'*hexapod_bringup*. Lors de sa création, il faut préciser l'adresse de publication, qui doit correspondre à celle sur laquelle écoute le contrôleur de trajectoires. Dans notre cas, il s'agit d'une chaîne de caractères de la forme :

$$/dynamixel_controllers/leg_0_controller/follow_joint_trajectory \quad (4.7)$$

On attend ensuite une réponse du serveur, pour être sûr que celui-ci est démarré. Dans le cas contraire les trajectoires ne pourront être envoyées, et le robot n'effectuera pas les actions commandées.

Par la suite, il faut demander au client d'envoyer l'objectif à atteindre qui contient notre trajectoire et donc l'ensemble des positions angulaires de commandes avec leurs timings associés. Le procédé est illustré par le listing 4.1

Il est également possible d'attendre le retour du contrôleur sur le statut de chaque point de la trajectoire, il faut alors appeler cette méthode dans un thread séparé, pour éviter de bloquer toute l'exécution.

```

1 // SENDING TRAJECTORY
2 void Leg::send_trajectory( const double& duration , const trajectory_msgs::JointTrajectory& trajectory )
3 {
4     control_msgs::FollowJointTrajectoryGoal goal;
5     goal.trajectory = trajectory;
6     _traj_client->sendGoal(goal);
7
8     // Wait for Results status
9     _traj_client->waitForResult(ros::Duration(duration));
10    if ( _traj_client->getState() != actionlib::SimpleClientGoalState::SUCCEEDED)
11        ROS_WARN_STREAM( "Trajectory execution for leg_"
12                         << legNum << " failed with status "
13                         << _traj_client->getState().toString() << " '" );
14 }

```

Listing 4.1 – Implémentation de la souscription à un topic ROS

4.5 Réalisation du bloc hexapod

4.5.1 Crédit des pattes

La création des pattes est assez simple, la partie la plus avancée étant la récupération de la chaîne cinématique. Le bloc hexapod se contente par la suite de créer les n pattes, de leur attribuer un numéro et de les stocker en mémoire.

4.5.1.1 Récupération du modèle du robot

Il est possible de charger un arbre cinématique – équivalent au modèle du robot – à partir d'un fichier URDF directement, et même si cela représente l'avantage d'être simple, l'inconvénient majeur de cette technique est qu'il faudrait coder en dur le chemin du fichier (ou simplement son nom si on laisse ROS le charger) ce qui veut dire qu'il faudrait éditer le code systématiquement en cas de changement de robot. Ce serait certes plus rapide que de refaire la chaîne cinématique, mais cela n'est toujours pas une bonne façon de faire.

Une autre solution existe, lors du démarrage du robot par le noeud *hexapod_bringup*, le fichier de description de ce dernier est chargé dans un paramètre de ros : `\robot_description`. Il est alors possible de récupérer depuis ce dernier une chaîne de caractère contenant l'intégralité du fichier URDF. L'avantage et que nous n'aurons ainsi plus besoin de modifier le code si le modèle du robot change.

Par la suite, une fonction intégrée à *KDL* : `Parser` permet de convertir cette chaîne de caractère en un arbre cinétique. La technique utilisée et décrite précédemment est illustrée par le listing 4.2.

```

1 KDL::Tree Hexapod::getTreeFromUrdf()
2 {
3     KDL::Tree my_tree;
4     std::string robot_desc_string;
5
6     if (!node.getParam("/robot_description", robot_desc_string)) {
7         ROS_ERROR("Failed to get /robot_description parameter");
8     }
9     if (!kdl_parser::treeFromString(robot_desc_string, my_tree)) {
10        ROS_ERROR("Failed to construct kdl tree");
11    }
12    return my_tree;
13 }
```

Listing 4.2 – Récupération de l’arbre cinématique depuis robot_description

4.5.1.2 Extraction de la chaîne cinématique

Une fois cet arbre récupéré, il faut en extraire la chaîne cinématique qui nous intéresse. Dans notre cas il s’agit d’extraire une à une les chaînes des pattes, partant du corps du robot et atteignant les capteurs de force. Le listing 4.3 effectue cela. Nous avons maintenant les données nécessaires pour calculer indépendamment les cinématiques de chaque patte.

```

1 KDL::Chain Hexapod::extractChain(KDL::Tree tree, std::string chainStart,
2                                   std::string chainTip)
3 {
4     KDL::Chain chain;
5     if (!tree.getChain(chainStart, chainTip, chain))
6         ROS_ERROR_STREAM("Unable to extract Chain " << chainStart << "-->" 
7                           << chainTip << " !");
8     return chain;
9 }
```

Listing 4.3 – Extraction de la chaîne cinématique

4.5.2 Synchronisation des données d’entrée

L’objectif ici est de synchroniser les différentes pattes, puis de générer des trajectoires adaptées afin de respecter les contraintes de temps imposées par le paramètre "fréquence"

4.5.2.1 Souscription au Topic ROS

Les données en entrée sont lues sur des Topics ROS, à une fréquence choisie lors de la souscription. Cette souscription est en effet accompagnée d’un Timer, qui appellera la fonction choisie à la fréquence choisie. J’ai choisi de mettre à jour directement ces valeurs dès leur arrivée, via des fonctions de callback de la classe Hexapod comme vous pouvez le voir dans le listing 4.4. La

souscription s'effectuant directement dans la classe exapod après avoir crée l'objet ros : :NodeHandle(). Ce dernier doit rester dans le scope au risque de ne plus synchroniser les données. La classe Hexapod transmettant également certaines données directement aux classes Leg si besoin. Les positions angulaires données par les codeurs intégrés aux servos-moteurs sont donc propagées automatiquement là où on en a besoin. Il en va de même pour les valeurs des capteurs de force.

```

1 Hexapod :: Hexapod () {
2     // initialisation de l'hexapod
3     joint_states_sub = node.subscribe( "/joint_states" ,
4                                         jointFrequency ,
5                                         &Hexapod :: jointStatesCallBack ,
6                                         this );
7 }
8 void Hexapod :: jointStatesCallBack ( const sensor_msgs :: JointState& msg )
9 {
10    // Recuperation et traitement des donnees du message
11 }
```

Listing 4.4 – Implémentation de la souscription à un topic ROS

4.5.2.2 Augmenter la fréquence de synchronisation

Dans un premier temps, en l'absence de capteurs de force, une fréquence de synchronisation élevée n'était pas nécessaire. il suffisait en effet de récupérer la position des servomoteurs une fois par pas, afin de connaître leur position nécessaire au calcul de la trajectoire.

Le code présenté dans le listing 4.5 suffisait alors à récupérer l'information requise, les Callbacks (présentés en) étaient alors appelés une fois par itération. L'instruction `ros ::spinOnce()` lançant la synchronisation une seule fois avant de rendre la main, permettant au reste du système d'effectuer ses calculs et opérations.

```

1 // éCration de l'hexapod
2 // Assignation des Callbacks
3 while( ros :: ok () ){
4     ros :: spinOnce () ; // Effectue la synchronisation
5 }
```

Listing 4.5 – Implémentation simple de la synchronisation de données

Il existe bien un outil simple permettant de synchroniser en permanence les données, afin de permettre de tester en continu les valeurs des capteurs de forces. Il s'agit de `ros ::Spin()` mais il ne rend pas la main ce qui ne permet pas d'effectuer d'opération supplémentaires.

La solution se trouve dans un autre outil de ROS, permettant d'effectuer cette mise à jour dans un thread séparé. il s'agit de l'outil *AsyncSpinner*. Il est relativement simple à utiliser et permet de garantir une synchronisation des données à la fréquence définie lors de la souscription au Topic. Son emploi est détaillé dans le listing 4.6. Il permet de synchroniser en permanence dans d'autres threads, libérant ainsi le thread principal pour nos calculs. C'est la solution qui sera retenue pour synchroniser les données.

```

1 // Creation de l'hexapod
2 // Assignment des Callbacks
3 ros::AsyncSpinner spinner(4); // Use 4 threads
4 spinner.start();
5 ros::waitForShutdown();

```

Listing 4.6 – synchronisation dans un thread séparé

4.5.3 Synchronisation des pattes

4.5.3.1 Ros ::Rate

Peu précis, Ros ::Rate encapsule un mécanisme assez simple, qui consiste en l'enregistrement du timeStamp lors de la création de l'objet, ou plus tard, lors de l'appel à la méthode sleep de cette même classe. Il est alors possible de récupérer le temps écoulé depuis l'un de ces événements à n'importe quel moment. Lors de la création de l'objet, on précise une fréquence désirée, qui servira à calculer le temps durant lequel il faut mettre en pause l'exécution pour obtenir la fréquence précédemment définie. Cette pause s'effectuera à l'appel de la fonction sleep.

```

1 ros::Rate loop_rate(frequency);
2 for (size_t i = 0; i < numSteps; i++) {
3
4     ros::spinOnce();
5
6     hexapod->setDuration(1.0 / frequency - loop_rate.cycleTime().toSec());
7     hexapod->sync();
8
9     loop_rate.sleep();
10 }
11 hexapod->relax();

```

Listing 4.7 – Implémentation de Ros ::Rate sous ROS

Après avoir mesuré les temps à l'aide du chrono proposé par c++11, il s'est avéré que cette solution était peu précise, et de surcroît très aléatoire. Il n'était donc pas envisageable de conserver cette manière de procéder.

4.5.3.2 Ros ::Timers + Callbacks

Meilleure manière de procéder, les Timers fournissent la garantie du respect de la fréquence choisie. L'implémentation n'est pas plus compliquée que Ros ::Rate, c'est donc une solution qui devrait être plus mis en avant dans les tutoriels ROS. Il suffit ici de créer un timer ROS comme l'illustre le code donné en 4.8

```

1 void hexapodSyncCallBack( const ros::TimerEvent& e )
2 {
3     hexapod->setDuration(1 / frequency); // Definit la duree de la trajectoire
4     hexapod->sync(); // Lance le calcul des trajectoires
5     steps++;
6 }
7 ros::Timer timer = n.createTimer(ros::Duration(1 / frequency),
8                                 hexapodSyncCallBack);
9 while (steps < numOfSteps) {
10     ros::spinOnce();
11 }
12 hexapod->relax();

```

Listing 4.8 – Implémentation des Timers sous ROS

Il est nécessaire d'ajouter un `ros::Spin()`, afin que les timers puissent fonctionner. Dans notre cas, il est même préférable d'utiliser `ros::SpinOnce()`, qui n'est pas une boucle infinie, et qui à l'avantage de nous permettre de limiter le nombre de pas à exécuter à chaque fois que l'on lance le contrôleur. Ce qui est fort pratique quand la zone de test est limitée, afin d'éviter une malencontreuse rencontre avec l'enceinte de l'arène.

4.6 Modèle du nouveau robot

Comme nous avons pu le voir, notamment dans la partie 4.4.2 et dans la partie 3.6, le modèle du robot joue un rôle primordial dans notre contrôleur. Sans lui impossible de tester le robot en simulation, impossible de déplacer les pattes. De plus, une simple erreur dans le modèle peut avoir des conséquences néfastes. En effet, une erreur sur les dimensions des pattes peut conduire à un calcul de cinématique éloigné de la vérité, provoquant ainsi des commandes sans rapport avec la réalité physique du robot. On pourrait donc lui demander de passer à travers son corps ou autre, provoquant dans le meilleur des cas une légère erreur de position des pattes et dans le pire des cas des collisions avec les autres composantes du robot.

4.6.1 organisation des scripts

Chaque robot comprend quatres modules¹² permettant de générer le modèle du robot :

- Principal, nommé avec le nom du robot
- body contenant la description du corps
- legs contenant les pattes et leurs positions
- leg contenant la description d'une patte

4.6.1.1 Module principal

Le module principal à pour but d'appeler les autres scripts, il contient aussi les propriétés xacro. Ces propriétés sont des ensembles clé-valeur et leur syntaxe est donnée en 4.9. L'exemple donné défini par exemple l'épaisseur du premier segment d'une patte.

```
1 <xacro:property name="leg1_width" value="0.02"/>
```

Listing 4.9 – Couple clé-valeur correspondant à un paramètre

Afin de charger les autres scripts, on utilisera la fonction de recherche de ROS nommée *find()* permettant de trouver un package ros, ce qui permet d'être indépendant du chemin réel sur la machine, ce qui est indispensable afin de rendre le programme portable. La syntaxe est illustrée par le listing 4.10.=, dans lequel le script de génération du corps du robot est chargé.

```
1 <!-- Load xacro modules/parts of the robot -->
2 <xacro:include filename="$(find hexapod_description)/urdf/modules/body.xacro" />
```

Listing 4.10 – Chargement d'un autre script

12. Ces modules sont disponibles ici : https://github.com/resibots/hexapod_ros/tree/marta-and-pexod-optoforce/hexapod_description/urdf/modules

4.6.1.2 Module body

Le module body défini le lien de base, sur lequel toutes les pattes seront fixées. Il comporte aussi les informations d'inertie du corps. Nulle nécessité de le modifier pour nous. Il sera néanmoins modifié par Dorian Goepp afin de respecter au mieux le visuel du robot et de pouvoir distinguer l'avant du robot.

4.6.1.3 Module Legs

Le module leg, comme annoncé précédemment, positionne et crée les pattes. Le listing 4.11 montre la manière de créer une patte, lui donnant les paramètres dont elle a besoin pour générer tous ses composants. Ce module ne sera pas non plus modifié dans le cadre du stage.

```
1 <xacro:leg index="0" xPos="${-body_length/2}" yPos="${body_width/2}" sign="1" />
```

Listing 4.11 – Crédit d'une patte

4.6.1.4 Module leg

C'est le module le plus complet de tous, il contient tout ce qu'il faut pour générer le visuel, la géométrie pour les collisions et les calculs, les poids et dimensions de chaque composant des pattes. Plusieurs modifications sont à effectuer sur ce script afin d'ajouter des fonctionnalités à l'ensemble du système.

Dans un premier temps, la géométrie des pattes a été mise à jour par Dorian Goepp, afin de correspondre au design des nouvelles pattes. Détailons maintenant les autres modifications effectuées.

4.6.2 Modifications pour les calculs cinématiques

KDL a besoin de la chaîne cinématique pour effectuer la cinématique directe, qui est indispensable pour connaître la position actuelle de l'extrémité de la patte mais aussi pour les calculs de cinématique inverse présentés en 4.4.2. Or cette chaîne cinématique est extraite du fichier de description.

Lors du calcul de cinématique, KDL prends pour extrémité de la chaîne le dernier joint et non le dernier lien. Cela pose problème car dans notre cas, le dernier joint est situé au niveau de la cheville, qui est réellement loin de l'extrémité réelle de la patte. Après avoir cherché la bonne manière de faire dans la documentation de KDL, il s'est avéré qu'il fallait ajouter un nouveau lien attaché au bout de la chaîne cinématique par un joint qui servira alors de référence.

Le listing 4.12 contient l'ajout effectué en extrémité de chaîne. On y définit l'origine du joint qui servira de référence pour les calculs cinématiques. Après vérification, les calculs obtenus sont désormais corrects.

```

1 <joint name="dummy_joint_${index}_fix" type="fixed">
2   <parent link="leg_${index}_3"/>
3   <child link="force_sensor_${index}" />
4   <origin rpy="-${sign*PI_2} 0 0" xyz="0 ${sign*leg3_length} ${leg3_dist}"/>
5 </joint>
6 <link name="dummy_link_${index}" />

```

Listing 4.12 – Correction de l’extrémité de la patte

4.6.3 Modifications pour RosDart

Le fichier de description généré pars les scripts xacro était utilisable en l’état, mais ne permettait pas le contrôle des servomoteurs ce qui empêchait toute simulation pertinente. En effet, il manquait une cartographie entre les servomoteurs et les liens qui leurs étaient associés. Dans le but d’indiquer ces paires dans le fichier de description, il faut ajouter des transmissions. La syntaxe est assez simple, elle est décrite dans le listing 4.13. Dans cet exemple, nous ajoutons une transmission au niveau de la hanche. Nous lions le joint *bodyLeg_x* au servomoteur *moteurx1*.

```

1 <transmission name="tran ${index}1">
2   <type>transmission_interface / SimpleTransmission</type>
3   <joint name="body_leg_${index}">
4     <hardwareInterface>PositionJointInterface</hardwareInterface>
5   </joint>
6   <actuator name="motor ${index}1">
7     <hardwareInterface>PositionJointInterface</hardwareInterface>
8     <mechanicalReduction>1</mechanicalReduction>
9   </actuator>
10 </transmission>

```

Listing 4.13 – Ajout des transmissions

Une fois ces modifications effectuées pour les trois servomoteurs de chaque patte, il est devenu possible de contrôler le robot en simulation.

4.6.4 Modifications pour les capteurs de force

Afin de récupérer les valeurs des capteurs de force en simulation, Konstantinos Chatzilygeroudis a ajouté un nouveau type de balise permettant de préciser la position du capteur ainsi que le topic ROS sur lequel l’on souhaite qu’il publie ses données. Il faut positionner ce lien à l’extrémité de la patte.

Or il se trouve que l’on a déjà ajouté un lien précédemment pour les besoins de la cinématique, il suffit donc d’exploiter ce lien afin d’y greffer le capteur.

```

1 <joint name="force_sensor_${index}_fix" type="fixed">
2   <parent link="leg_${index}_3"/>
3   <child link="force_sensor_${index}" />
4   <origin rpy="${-sign*PI_2} 0 0" xyz="0 ${sign*leg3_length} ${leg3_dist}" />
5 </joint>
6 <link name="force_sensor_${index}" />
7 <dart_sensor type="force">
8   <link>force_sensor_${index}</link>
9   <frequency>1000</frequency>
10  <topic>optoforce_${index}</topic>
11 </dart_sensor>

```

Listing 4.14 – Modifications et ajout du capteur

5 Résultats et performances

5.1 Résultats Obtenus

5.1.1 Une solution de cinématique inverse performante

Comme présenté précédemment en 4.4.2.5, les résultats obtenus sont très satisfaisants et notre solution de cinématique inverse trouve toujours une solution, même dans le cas où la position demandée n'est pas atteignable.

L'algorithme d'optimisation va alors trouver la solution la plus proche possible de l'objectif souhaité, tout en avertissant l'utilisateur que la solution n'est pas atteignable. Il est même possible d'afficher la distance d'erreur. L'algorithme est en effet programmé pour s'arrêter après x itérations, permettant de ne pas consommer trop de temps machine.

Le tableau 5.1 récapitule les performances des différents algorithmes. Le temps moyen d'une itération correspond au temps nécessaire au calcul de la cinématique inverse d'un point, celui pour les six pattes correspond au temps nécessaire pour calculer un point par patte, et la dernière colonne donne la fréquence maximale d'emploi de l'algorithme – c'est à dire combien de points par seconde et par pattes peut-on calculer en exploitant toute la puissance disponible.

Ces benchmarks ont été réalisés sur la machine qui m'a été fournie avec une précision réglée sur 10^{-3} qui représente l'erreur tolérée à laquelle l'algorithme s'arrêtera. Cette machine est bien moins puissante que la machine sur laquelle le contrôleur tournera réellement, ce qui garantit son bon fonctionnement.

Algorithme	Temps moyen d'une itération en us	Temps pour les 6 pattes	Fréquence max Hz
Cobyla	400	2400	416,6666667
BOBYQA	100	600	1666,666667
NEWUOA	370	2220	450,4504505
PRAXIS	180	1080	925,9259259
NELDERMEAD	70	420	2380,952381
SBPLX	110	660	1515,151515

TABLE 5.1 – Tableau comparatif des performances (réalisé sur 10 000 itérations)

Il va sans dire que nous avons choisi l'algorithme le plus rapide, *NELDERMEAD*¹. Cet algorithme présente l'avantage d'être efficace dans le cas d'une fonction non dérivable, avec un problème de faible dimension (c'est le cas ici car $N = 3$). Même après avoir augmenté la précision du solveur, le temps nécessaire pour calculer un point ne dépasse pas les 100 us, ce qui reste totalement raisonnable pour notre application.

Cette solution de calcul de cinématique inverse permet donc de générer de nombreuses trajectoires selon nos besoins.

5.1.2 Un Contrôleur facilement adaptable

L'emploi du pattern strategy permet de créer et d'intégrer rapidement des comportements additionnels apportant une grande adaptabilité à l'ensemble. Cela permettra également de suivre différents axes d'amélioration en simultané, en changeant par exemple la procédure en cas de contact pour effectuer le contrôle de la force évoqué dans l'état de l'art.

5.1.3 Un contrôleur omnidirectionnel

La capacité du contrôleur à évoluer dans toutes les directions en suivant un simple vecteur de commande est un avantage en soi. Le précédent contrôleur de l'équipe ne proposait en effet qu'une marche simple, les comportements omnidirectionnels devaient être appris. Cela représente un gain de temps et d'énergie non négligeable.

Cette capacité omnidirectionnelle permet également de dégager de nouveaux comportements, par exemple aborder un terrain difficile en diagonale peut permettre de gagner en efficacité.

5.1.4 Un contrôleur plus robuste ?

Les premiers tests ont permis de déterminer que le robot avait moins tendance à se déstabiliser lors du franchissement d'un obstacle. En effet, le fait de stabiliser son assiette et de suivre le mouvement lors de la détection d'un contact avec un obstacle évite au robot de se surélever en continuant à pousser sur la patte forçant sur l'obstacle.

Ces tests ont été réalisés dans l'arène de l'équipe (visible en figure 5.1, préparée pour l'occasion avec des obstacles divers et variés afin de simuler un environnement difficile.

Le principal problème rencontré se situe dans la reproduction des conditions de l'expérience. En effet, afin de mesurer l'apport de la prise en compte des capteurs dans la capacité du robot à évoluer en terrains difficiles, nous effectuons un essai avec les capteurs et un essai sans. Il est difficile de positionner le robot exactement dans la même position initiale, ce qui change fortement l'angle d'attaque du robot et influe donc sur sa capacité à franchir le parcours.

1. La méthode de Nelder-Mead est un algorithme d'optimisation non-linéaire qui a été publiée par John Nelder et Mead en 1965. C'est une méthode numérique heuristique qui cherche à minimiser une fonction continue dans un espace à plusieurs dimensions.

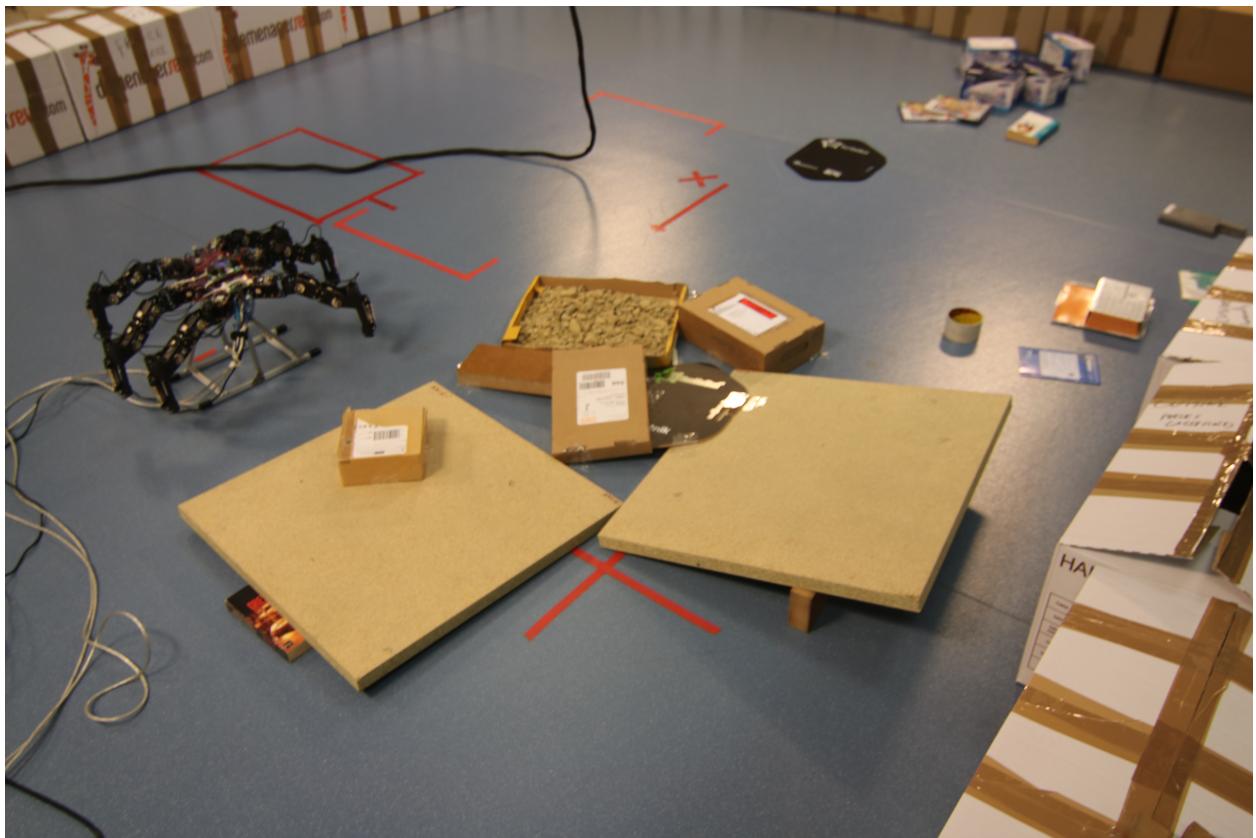


FIGURE 5.1 – Terrain de test

L'idée serait à terme d'essayer de positionner le robot avant l'expérience grâce au système de suivi infrarouge de l'arène de l'équipe. Cela permettrait à la fois d'obtenir des conditions quasi-similaires, mais également d'effectuer des mesures concrètes de la performance des essais avec et sans capteurs.

Ces essais sont déjà en cours et seront poursuivis d'ici à la fin du stage, afin de fournir à l'équipe une information plus précise concernant les résultats obtenus par le nouveau contrôleur.

5.2 Axes d'amélioration

Il existe de nombreux progrès réalisables sur la version actuelle du contrôleur. Cette liste d'améliorations est potentiellement infinie, c'est la raison pour laquelle ne seront présentés que les principaux axes.

5.2.1 Contrôle de la force

Comme exposé dans l'état de l'art en 2.3.1.2, implémenter un contrôle plus précis de la force peut s'avérer intéressant pour garantir un bonne stabilité de la plateforme. Un asservissement de la force de chaque patte peut éventuellement être effectué afin d'obtenir une stabilité optimale. Il suffirait alors de mesurer la force, et d'ajuster la position de la patte en conséquence. l'éloignant de point de contact dans la direction de la force mesurée si celle ci est trop forte, ou au contraire

pousser dans la direction opposée de la force si celle-ci n'est pas suffisante.

5.2.2 Mapper les forces

Si asservir les forces aux extrémités des pattes peut s'avérer intéressant, créer une carte des forces exercées et réagir aux perturbations extérieures en contrôlant l'ensemble des pattes de concert optimisant ainsi la posture et les réactions locales.

5.2.3 Permettre de modifier les points neutres des pattes

Une piste intéressante de compensation de dommage pourrait consister à déplacer les positions neutres des pattes, utilisées pour calculer l'objectif de chaque pas présenté en 4.4.1.4. Déplacer ce centre permettrait donc de compenser l'équilibre du robot dans le cas d'une perte de patte. Il permettrait également de générer une variété plus riche de comportements, ce qui peut représenter un intérêt certain pour l'équipe dans le cadre de l'utilisation de leurs algorithmes d'apprentissage.

Mise à jour : cette fonctionnalité a été intégrée, toutefois sans mécanisme intelligent permettant d'adapter automatiquement ces points de référence. Cela permet par exemple de modifier la hauteur du robot, ce qui influe beaucoup sur sa capacité de franchissement ainsi que sur sa stabilité.

5.2.4 Utiliser la centrale inertie

L'apport du traitement des données de la centrale inertie permettrait de commander les pattes en fonction de l'assiette du robot. On pourrait par exemple facilement ajuster la hauteur de chaque patte afin d'ajuster cette assiette, en fonction du comportement désiré.

Cela permettrait également de compenser une défaillance, dans le cas d'une patte ayant sectionnée, il serait possible d'incliner l'assiette du robot vers l'arrière afin de conserver une posture stable en repositionnant le centre de masse à l'intérieur du polygone de support des pattes.

Cette centrale inertie permettrait également de détecter un mouvement brusque comme une chute, qui associé à un grand changement de force appliquée en extrémité de patte, permettrait de savoir quelle patte est en train de glisser et donc quelle patte il faut tenter de stabiliser.

Un autre apport potentiel sera la prise en compte de l'orientation du robot pour les calculs de trajectoires. Au lieu de calculer comme c'est le cas actuellement sans prendre en compte sa rotation par rapport à la gravité, nous pourrions adapter les trajectoires des pattes afin de maintenir celles-ci parallèles à la gravité. Cela permettrait d'augmenter grandement la stabilité de la plate-forme.

5.2.5 Planification locale du positionnement des pattes

Utiliser un mécanisme de vision permettrait également de faire du planning, pas pour trouver le chemin à effectuer, mais pour savoir où poser sa patte de manière sûre. cela permettrait d'éviter de glisser, du moins dans la plupart des cas. L'autre avantage de l'ajout d'une cartographie locale est l'obtention d'un modèle de l'environnement, redonnant de l'intérêt aux approches basées sur le modèle telles que les MPC.

5.2.6 Combiner plusieurs de ces approches

Bien entendu, de nombreuses améliorations sont indépendantes, mais peuvent parfois être couplées, afin d'obtenir un contrôleur encore et toujours plus robuste, encore plus fiable.

5.3 Respect du planning prévisionnel

5.3.0.1 Planning prévisionnel

Étant néophyte dans le domaine ciblé par le stage et n'ayant par conséquence que peu d'expérience, il n'a pas été évident de planifier correctement l'ensemble du projet. Ce planning a donc été réalisé après la phase de prise en main et d'état de l'art.

Prévisionnel	Avril	Mai	Juin	Juillet	Août	Septembre
Formation et état de l'art						
Simulateur						
Cinématique inverse						
Contrôleur omnidirectionnel						
Seuil de force						
Contrôle de la force						
Fonctionnalités avancés						

FIGURE 5.2 – Planning prévisionnel

Comme vous pouvez le voir sur la figure 5.2, la réalisation des composants indispensables a été planifiée en premier, à savoir le module de cinématique inverse et le contrôleur omnidirectionnel. Le plan était d'atteindre ses objectifs, puis de finaliser une première version basique du contrôleur utilisant un seuil de force pour détecter un contact.

Par la suite, il était prévu d'ajouter de plus en plus de fonctionnalités tels qu'un contrôle avancé de la force appliquée aux pattes et puis si possible un MPC.

5.3.0.2 Planning réel

Il n'est jamais évident de prévoir exactement le déroulement d'un projet. Mais si l'on regarde la figure 5.3, on constate que mise à part le retard important engendré par la cinématique inverse, le reste des jalonnements prévus s'est enchaîné dans l'ordre prévu.

Toutes les fonctionnalités prévues n'ont pu être implémentées, mais la version actuelle du contrôleur et fonctionnelle, et les améliorations possibles sont décrites ici-même. De plus, le stage n'est pas encore fini, et il reste suffisamment de temps pour en ajouter quelques unes.

Réel	Avril	Mai	Juin	Juillet	Août	Septembre
Formation et état de l'art	■					
Simulateur		■				
Cinématique inverse			■	■		
Contrôleur omnidirectionnel				■	■	
Seuil de force					■	
Contrôle de la force						■
Fonctionnalités avancées						■

FIGURE 5.3 – Planning réel

5.4 Ressenti personnel

5.4.1 Un accueil Chaleureux

Dès mon arrivée et jusqu'au terme de ce stage, l'accueil de l'équipe LARSEN fut réellement amical. Partager ensemble les repas du midi a créé de nombreuses occasion de discuter de sujets diverses et variés et de découvrir le rôle et les investigations de chacun.

5.4.2 Un environnement enrichissant

Lors du déroulement de ce stage ont eu lieu des conférences enrichissantes dans le cadre des 40 ans du LORIA. C'est une chance d'avoir pu assister aux démonstrations effectuées par des précurseurs dans leur domaine.

5.4.3 Un domaine intéressant

La robotique est un domaine qui m'a toujours intéressé, et plus particulièrement l'apprentissage. Ce stage a donc représenté pour moi une réelle opportunité de découvrir en direct l'application concrète d'algorithmes d'apprentissage, et d'avoir de surcroît des explications quand à leur fonctionnement particulier. Et ce grâce à de nombreux échanges avec mes collègues.

5.4.4 Des difficultés rencontrées

Comme nous avons pu le voir dans la partie 5.3.0.2, toutes les étapes prévues n'ont pu être réalisées au terme de ce stage. Le robot est certes fonctionnel, mais beaucoup d'améliorations supplémentaires auraient pu être implémentées.

En effet, les difficultés rencontrées lors de la réalisation des calculs cinématiques ont grandement retardé la mise en œuvre du reste du contrôleur. Cette tâche semblait facile et à demandé bien plus d'énergie et d'investissement que prévu, mais avec ma faible expérience en la matière, il était difficile de voir venir ces déconvenues.

5.4.5 Une légère déception

Ce retard s'étant reporté sur l'ensemble du projet, il n'a pas été possible d'implémenter les fonctionnalités plus avancées imaginées lors de ce stage. Le fait de ne pas avoir pu travailler sur ces ajouts améliorés me laisse un goût amer, car cela constitue la partie la plus gratifiante du stage. Même si tout le travail préliminaire est indispensable et conséquent, c'est la partie la plus passionnante.

5.4.6 Des résultats encourageants

Néanmoins, les résultats obtenus sont encourageants, et malgré le fort retard provoqué par la cinématique inverse, le projet à quand même aboutit à un contrôleur omnidirectionnel fonctionnel, et prenant en compte les retours des capteurs afin de s'adapter au terrain.

Il reste toutefois quelques jours de stage et quelques améliorations à explorer, ce qui est encourageant.

6 Conclusion

Quelques jours avant le terme du stage, l'objectif principal de détermination et d'implémentation d'un contrôleur réactif pour hexapod est rempli. Cela a été rendu possible dans un premier temps par une analyse détaillée de la problématique, puis une recherche méthodique de solutions potentielles, à travers la lecture de nombreuses publications scientifiques sur le sujet.

Après avoir acquis une idée plus précise des possibilités offertes, un choix raisonnable a permis d'opter pour une solution atteignable, autant sur le point pratique dans son implémentation que dans ses concepts théoriques, permettant de maîtriser le projet jusqu'à son terme et ce malgré quelques écueils.

Des difficultés réellement enrichissantes sont survenues au fur et à mesure de l'avancement du projet. La principale ayant sans nul doute été la réalisation du module de cinématique inverse, qui n'avait pas été perçu comme obstacle potentiel. Après avoir échoué à faire fonctionner des librairies – pourtant spécialisées – dans notre contexte particulier, il a fallu faire preuve de persévérance afin de concevoir et développer ma propre solution, qui s'avère même plus avantageuse sur certains points (voir 4.4.2). Cette expérience a donc représenté pour moi une réelle leçon.

L'adoption d'une méthode de tests et validations successifs a permis d'avancer de manière saine et sécurisée dans le projet. Les fonctionnalités ont été ajoutées au fur et à mesure pour aboutir à une version fonctionnelle.

Il reste de nombreuses améliorations possibles, et je tenterais autant que faire ce peut de pousser le projet encore plus loin dans les jours à venir. La réalisation de mesures de performances étant toujours en cours, ces dernières seront analysées avant le départ du stage afin de fournir une analyse complète de l'apport réel des travaux réalisés durant ce stage.

D'un point de vue personnel, je dois avouer une certaine déception de ne pas avoir pu à ce jour implémenter les fonctionnalités identifiées et théoriquement préparées.

Ce stage a représenté une expérience réellement enrichissante, me donnant un accès au monde de la recherche et à nombre de ses connaissances. Qu'elles viennent de l'accès aux publications scientifiques, du suivi de conférences –menées par des experts de leur domaine– ou encore de l'apport journalier de mes collègues, ces sources d'informations n'ont fait que renforcer mon intérêt pour la science, et plus particulièrement pour la robotique.

Le monde de la recherche me plaît, et j'aimerais si l'occasion s'en présente continuer mes études dans cette voie. A la suite de ce stage, j'ai encore quelques doutes quant à ma capacité réelle à poursuivre en thèse. C'est dans ce contexte flou que je recherche actuellement les opportunités s'offrant à moi, qui me permettraient de continuer à en apprendre toujours plus dans le domaine. J'envisage actuellement d'effectuer une mission à l'étranger, afin de voyager et d'y découvrir d'autres cultures, d'autres horizons, ce qui m'aidera sans aucun doute à mieux cerner l'orientation que je veux donner à ma vie, aussi bien personnelle que professionnelle.

Bibliographie / Webographie

- [1] Hexapod robot AMOS II : Adaptable locomotion under neural control and adaptive forward models - YouTube.
- [2] ikfastloader.cpp 0.8.0.
- [3] Walking Robot LAURON V - Hexapod on Wooden Slope - YouTube.
- [4] Walking Robot LAURON V : Grasping and other skills developed for DLR SpaceBot Cup - YouTube.
- [5] Mahdi Agheli and Stephen S. Nestinger. Foot Force Based Reactive Stability of Multi-Legged Robots to External Perturbations. *Journal of Intelligent & Robotic Systems*, 81(3-4) :287–300, June 2015. 20
- [6] V. Barasuol, J. Buchli, C. Semini, M. Frigerio, E. R. De Pieri, and D. G. Caldwell. A reactive controller framework for quadrupedal locomotion on challenging terrain. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561, May 2013.
- [7] Randall D. Beer, Roger D. Quinn, Hillel J. Chiel, and Roy E. Ritzmann. Biologically Inspired Approaches to Robotics : What Can We Learn from Insects ? *Commun. ACM*, 40(3) :30–38, March 1997.
- [8] T. Boaventura, J. Buchli, C. Semini, and D. G. Caldwell. Model-Based Hydraulic Impedance Control for Dynamic Robots. *IEEE Transactions on Robotics*, 31(6) :1324–1336, December 2015.
- [9] Martin Buehler. Dynamic locomotion with one, four and six-legged robots. *JOURNAL-ROBOTICS SOCIETY OF JAPAN*, 20(3) :15–20, 2002.
- [10] R. Campos, V. Matos, and C. Santos. Hexapod locomotion : A nonlinear dynamical systems approach. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 1546–1551, November 2010.
- [11] Jie Chen, Yubin Liu, Jie Zhao, He Zhang, and Hongzhe Jin. Biomimetic Design and Optimal Swing of a Hexapod Robot Leg. *Journal of Bionic Engineering*, 11(1) :26–35, January 2014.
- [12] Hillel J. Chiel, Randall D. Beer, Roger D. Quinn, and Kenneth S. Espenschied. Robustness of a distributed neural network controller for locomotion in a hexapod robot. *Robotics and Automation, IEEE Transactions on*, 8(3) :293–303, 1992.
- [13] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553) :503–507, May 2015. 1
- [14] Fred Delcomyn and Mark E Nelson. Architectures for a biomimetic hexapod robot. *Robotics and Autonomous Systems*, 30(1–2) :5–15, January 2000.

- [15] Kenneth S. Espenschied, Roger D. Quinn, Randall D. Beer, and Hillel J. Chiel. Biologically based distributed control and local reflexes improve rough terrain locomotion in a hexapod robot. *Robotics and Autonomous Systems*, 18(1–2) :59–64, July 1996.
- [16] Cynthia Ferrell. A comparison of three insect-inspired locomotion controllers. *Robotics and Autonomous Systems*, 16(2–4) :135–159, December 1995. 12
- [17] Michael R. Fielding and G. Reg Dunlop. Omnidirectional Hexapod Walking and Efficient Gaits Using Restrictedness. *The International Journal of Robotics Research*, 23(10–11) :1105–1110, January 2004.
- [18] Jose A Galvez, Joaquin Estremera, and Pablo Gonzalez de Santos. A new legged-robot configuration for research in force distribution. *Mechatronics*, 13(8–9) :907–932, October 2003.
- [19] Elena Garcia and Pablo Gonzalez De Santos. An improved energy stability margin for walking machines subject to dynamic effects. *Robotica*, 23(01) :13–20, 2005.
- [20] Angel Gaspar Gonzalez-Rodriguez, Antonio Gonzalez-Rodriguez, and Fernando Castillo-Garcia. Improving the energy efficiency and speed of walking robots. *Mechatronics*, 24(5) :476–488, August 2014.
- [21] P. Gregorio, M. Ahmadi, and M. Buehler. Design, control, and energetics of an electrically actuated legged robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(4) :626–634, August 1997.
- [22] Qing-Jiu Huang and Kenzo Nonami. Humanitarian mine detecting six-legged walking robot and hybrid neuro walking control with position/force control. *Mechatronics*, 13(8–9) :773–790, October 2003.
- [23] Marco Hutter, Philipp Leemann, Stefan Stevsic, Andreas Michel, Dominic Jud, Mark Hoepflinger, Roland Siegwart, Ruedi Fagi, Christian Caduff, Markus Loher, and others. Towards optimal force distribution for walking excavators. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 295–301. IEEE, 2015.
- [24] Auke Jan Ijspeert. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, 84(5) :331–348, April 2001.
- [25] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots : A review. *Neural Networks*, 21(4) :642–653, May 2008.
- [26] A. Irawan and K. Nonami. Force Threshold-Based Omni-directional Movement for Hexapod Robot Walking on Uneven Terrain. In *Modelling and Simulation 2012 Fourth International Conference on Computational Intelligence*, pages 127–132, September 2012. 18
- [27] Addie Irawan, Md Moktadir Alam, Yee Yin Tan, and Mohd Rizal Arshad. CENTER OF MASS-BASED ADMITTANCE CONTROL FOR MULTI-LEGGED ROBOT WALKING ON THE BOTTOM OF OCEAN. *Jurnal Teknologi*, 74(9), 2015.
- [28] Addie Irawan, Kenzo Nonami, and Mohd Razali Daud. Optimal Impedance Control with TSK-Type FLC for Hard Shaking Reduction on Hydraulically Driven Hexapod Robot. In *Autonomous Control Systems and Vehicles*, pages 223–236. Springer, 2013.
- [29] Addie Irawan, Kenzo Nonami, Hiroshi Ohroku, Yasunaga Akutsu, and Shota Imamura. Adaptive Impedance Control with Compliant Body Balance for Hydraulically Driven Hexapod Robot. *Journal of System Design and Dynamics*, 5(5) :893–908, 2011.
- [30] Addie Irawana and Md Moktadir Alamb. Adaptive impedance control based on CoM for hexapod robot walking on the bottom of ocean. 18

- [31] K. Kamikawa, T. Arai, K. Inoue, and Y. Mae. Omni-directional gait of multi-legged rescue robot. In *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*, volume 3, pages 2171–2176 Vol.3, April 2004.
- [32] HyunGyu Kim, DongGyu Lee, Yanheng Liu, Kyungmin Jeong, and TaeWon Seo. Hexapedal Robotic Platform for Amphibious Locomotion on Ground and Water Surface. *Journal of Bionic Engineering*, 13(1) :39–47, January 2016.
- [33] Junmin Li, Jing Wang, Simon X. Yang, Kedong Zhou, and Huijuan Tang. Gait Planning and Stability Control of A Quadruped Robot. ? ? ? ?
- [34] P. Manoonpong, T. Geng, B. Porr, and F. Wörgötter. The RunBot Architecture for Adaptive, Fast, Dynamic Walking. In *2007 IEEE International Symposium on Circuits and Systems*, pages 1181–1184, May 2007. 16
- [35] P. Manoonpong, F. Pasemann, and F. Wörgötter. Sensor-driven neural control for omnidirectional locomotion and versatile reactive behaviors of walking machines. *Robotics and Autonomous Systems*, 56(3) :265–288, March 2008. 13, 14, 15, 75
- [36] Poramate Manoonpong, Tao Geng, Tomas Kulvicius, Bernd Porr, and Florentin Wörgötter. Adaptive, Fast Walking in a Biped Robot under Neuronal Control and Learning. *PLOS Comput Biol*, 3(7) :e134, July 2007.
- [37] Shixin Mao, Erbao Dong, Hu Jin, Min Xu, Shiwu Zhang, Jie Yang, and Kin Huat Low. Gait Study and Pattern Generation of a Starfish-Like Soft Robot with Flexible Rays Actuated by SMAs. *Journal of Bionic Engineering*, 11(3) :400–411, July 2014.
- [38] Markus Eich, Felix Grimminger, and Frank Kirchner. Adaptive compliance control of a multi-legged stairclimbing robot based on proprioceptive data. *Industrial Robot : An International Journal*, 36(4) :331–339, June 2009.
- [39] L. S. Martins-Filho and R. Prajoux. Locomotion control of a four-legged robot embedding real-time reasoning in the force distribution. *Robotics and Autonomous Systems*, 32(4) :219–235, September 2000.
- [40] V. Matos and C. P. Santos. Omnidirectional locomotion in a quadruped robot : A CPG-based approach. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3392–3397, October 2010.
- [41] Maximilien Naveau, Manuel Kudruss, Olivier Stasse, Christian Kirches, Katja Mombaur, and Philippe Soueres. A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control. 2016. 20
- [42] Victor Ragusila and M. Reza Emami. Modelling of a robotic leg using bond graphs. *Simulation Modelling Practice and Theory*, 40 :132–143, January 2014.
- [43] Guanjiao Ren, Weihai Chen, Sakyasingha Dasgupta, Christoph Kolodziejski, Florentin Wörgötter, and Poramate Manoonpong. Multiple chaotic central pattern generators with learning for legged locomotion and malfunction compensation. *Information Sciences*, 294 :666–682, February 2015. 16
- [44] Cristina P. Santos and Vítor Matos. CPG modulation for navigation and omnidirectional quadruped locomotion. *Robotics and Autonomous Systems*, 60(6) :912–927, June 2012. 16
- [45] U. Saranli, M. Buehler, and D. E. Koditschek. Design, modeling and preliminary control of a compliant hexapod robot. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*, volume 3, pages 2589–2596 vol.3, 2000.

- [46] Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. RHex : A Simple and Highly Mobile Hexapod Robot. *The International Journal of Robotics Research*, 20(7) :616–631, January 2001. :2001
- [47] A. Schneider, H. Cruse, and J. Schmitz. A biologically inspired active compliant joint using local positive velocity feedback (LPVF). *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(6) :1120–1130, December 2005.
- [48] F. Seljanko. Towards omnidirectional locomotion strategy for hexapod walking robot. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 143–148, November 2011. 19
- [49] N. Suzuki, K. Nonaka, and K. Sekiguchi. Model predictive obstacle avoidance control with passage width constraints for leg/wheel robots. In *2015 IEEE Conference on Control Applications (CCA)*, pages 330–335, September 2015.
- [50] Roberto Travaglini. Adaptive and robust leg control of a hexapod robot for space exploration. 2015. 16
- [51] Miomir Vukobratović and Branislav Borovac. Zero-moment point—thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01) :157–173, 2004.
- [52] Z. Wang, J. Kinugawa, H. Wang, and K. Kazuhiro. The Simulation of Nonlinear Model Predictive Control for a Human-following Mobile Robot. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 415–422, December 2015.
- [53] J. Xu, L. Ge, J. Wang, Q. Wei, and H. Ma. ZMP preview control based compliance control for a walking quadruped robot. In *2015 IEEE International Conference on Information and Automation*, pages 7–12, August 2015.
- [54] Zhijun Yang, Daqiang Zhang, Marlon V. Rocha, Priscila M. V. Lima, Mehmet Karamanoglu, and Felipe M. G. França. Prescription of rhythmic patterns for legged locomotion. *Neural Computing and Applications*, pages 1–15, March 2016.
- [55] HongChao Zhuang, HaiBo Gao, ZongQuan Deng, Liang Ding, and Zhen Liu. A review of heavy-duty legged robots. *Science China Technological Sciences*, 57(2) :298–314, 2014.

Liste des illustrations

1.1	Hiérarchie des entités dont dépend l'équipe LARSEN	2
1.2	Les membres de l'équipe LARSEN	6
2.1	Exemple de réseau de neurones d'un contrôleur [35]	14
2.2	Lien avec les servomoteurs[35]	15
2.3	La marche tripod	19
2.4	Logique décisionnelle de l'approche basée sur le CoM	20
3.1	Géométrie de l'hexapode d'origine	23
3.2	Géométrie du nouvel hexapode	23
3.3	Vue du capteur de force situé à l'extrémité de la patte	24
3.4	Exemple d'utilisation des topics ROS	27
3.5	Capture d'écran de l'interface graphique de RosDart	30
4.1	Schéma global du contrôleur	34
4.2	Organisation interne de l'hexapod	38
4.3	Schéma UML de l'implémentation du pattern Strategy	40
4.4	Schéma de la génération de trajectoire support	41
4.5	Lissage de la trajectoire	42
4.6	Schéma de la génération de trajectoire swing	43
4.7	Trajectoire elliptique générée	43
4.8	Principe de la correction du vecteur de déplacement	44
4.9	Effets de la compensation par calcul absolu de l'objectif	44
4.10	Trajectoire traduite par le solveur de cinématique inverse	49

5.1	Terrain de test	61
5.2	Planning prévisionnel	63
5.3	Planning réel	64
A.1	Organigramme du LORIA	79

Listings

3.1	Contenu du JointState.msg	26
3.2	En-tête d'un message	26
4.1	Implémentation de la souscription à un topic ROS	50
4.2	Récupération de l'arbre cinématique depuis robot_description	51
4.3	Extraction de la chaîne cinématique	51
4.4	Implémentation de la souscription à un topic ROS	52
4.5	Implémentation simple de la synchronisation de données	52
4.6	synchronisation dans un thread séparé	53
4.7	Implémentation de Ros ::Rate sous ROS	53
4.8	Implémentation des Timers sous ROS	54
4.9	Couple clé-valeur correspondant à un paramètre	55
4.10	Chargement d'un autre script	55
4.11	Création d'une patte	56
4.12	Correction de l'extrémité de la patte	57
4.13	Ajout des transmissions	57
4.14	Modifications et ajout du capteur	58
B.1	Exemple de fichier URDF, ici une partie de la jambe de notre robot	80

A Organigramme LORIA

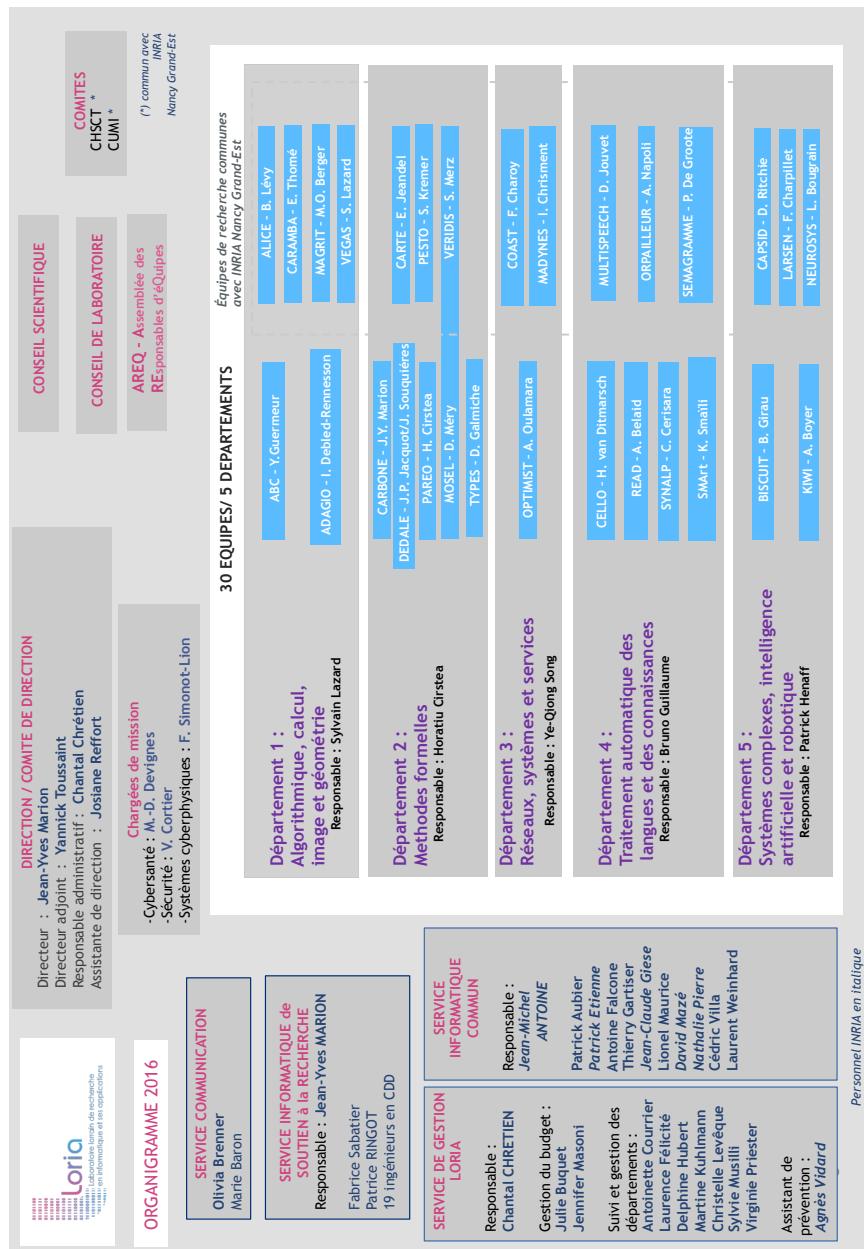


FIGURE A.1 – Organigramme du LORIA

B Example de fichier URDF

```
1 <joint name="leg_2_1_2" type="revolute">
2   <parent link="leg_2_1"/>
3   <child link="leg_2_2"/>
4   <limit effort="30.0" lower="-0.78539816339" upper="0.78539816339" velocity="7.0"/>
5   <origin rpy="0 0 0" xyz="0 0.06 0"/>
6   <axis xyz="1 0 0"/>
7   <dynamics damping="0.0"/>
8 </joint>
9 <link name="leg_2_2">
10  <visual>
11    <origin rpy="1.57079632679 0 0" xyz="0.01 0.0425 0"/>
12    <geometry>
13      <cylinder length="0.085" radius="0.02"/>
14    </geometry>
15    <material name="Blue"/>
16  </visual>
17  <collision>
18    <origin rpy="1.57079632679 0 0" xyz="0.01 0.0425 0"/>
19    <geometry>
20      <box size="0.02 0.02 0.085"/>
21    </geometry>
22  </collision>
23  <inertial>
24    <!-- CENTER OF MASS -->
25    <origin rpy="1.57079632679 0 0" xyz="0.01 0.0425 0"/>
26    <mass value="0.184"/>
27    <!-- box inertia: 1/12*m(y^2+z^2), ... -->
28    <inertia ixx="0.000116916666667" ixy="0" ixz="0" iyy="0.000116916666667"
29      iyz="0" izz="1.22666666667e-05"/>
30  </inertial>
</link>
```

Listing B.1 – Exemple de fichier URDF, ici une partie de la jambe de notre robot

Résumé

Ce mémoire relate l'expérience vécue lors d'un stage de fin d'études effectué à l'INRIA dans le but d'obtenir le diplôme d'ingénieur délivré par Telecom Nancy. L'objectif de ce stage est la réalisation d'un ou plusieurs contrôleurs pour hexapode, devant prendre en compte des capteurs de force. Seront présentés dans cet ouvrage les outils, méthodes et solutions qui ont permis d'atteindre l'objectif fixé. Un état de l'art tente de donner les bases théoriques et les raisons du choix effectué, privilégiant les approches pragmatiques aux approches biomimétiques. Le contrôleur réalisé s'appuie sur des pattes indépendantes, capables de générer leur propre trajectoire dans l'espace cartésien, ainsi qu'un module de cinématique inverse chargé de convertir ses trajectoires en ordres pour les actionneurs des pattes. Un module de contrôle omnidirectionnel sera chargé quant à lui de donner les objectifs aux pattes ainsi que de s'assurer de leur bonne synchronisation. Le contrôleur ainsi réalisé permet d'adopter un comportement plus stable dans des environnements difficiles. Seront donc présentés les différents modules, comprenant une partie théorique expliquant leur fonctionnement et une partie détaillant leur implémentation. Par la suite, un chapitre détaillera les résultats obtenus ainsi qu'une description des expérimentations ayant permis de les obtenir. **Mots-clés : Hexapode, locomotion, robotique, cinématique, marche omnidirectionnelle**

Abstract

This paper describes the experience during an internship conducted at the gls INRIA in order to get the engineering degree from Telecom Nancy. The objective of this course is the design and realization of one or more controllers for hexapod, who should take force sensors into account. Will be presented in this work the tools, methods and solutions that have allowed to reach the target. A state of the art will attempts to provide the theoretical basics and reasons why favoring pragmatic approaches to biomimetic approaches. Realized controller relies on independent legs, capable of generating their own path in Cartesian space, an inverse kinematics module responsible for converting those trajectory in commands for the actuators of the legs. Omnidirectional control module will be responsible for giving each leg goals as well as ensure their proper synchronization. The thus achieved controller allows to adopt a more stable performance in harsh environments. Will thus be presented the different modules, with a theoretical part explaining their operation and some detailing their implementation. Subsequently, a chapter will detail the results and a description of experiments that enabled to get them. **Keywords : Hexapod, locomotion, robotics, kinematics, omnidirectional walking**