Optimisation de la locomotion de robots bas coût à pattes

Grégoire Passault

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Grégoire Passault**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Optimisation de la locomotion de robots bas coût à pattes

Date de soutenance : 14 décembre 2016

Devant la commission d'examen composée de :

Pierre Blazevic ...	Professeur, ISTY	Rapporteur
Serge Chaumette .	Professeur, LaBRI	Examinateur
Reinhard Gerndt .	Professeur, Ostfalia University of Applied Sciences	Examinateur
Nicolas Jouandeau	Maître de conférences, LIASD	Rapporteur
Olivier Ly	Maître de conférences, LaBRI	Directeur

– 2016 –

Résumé Les robots à pattes promettent de pouvoir marcher sur des terrains irréguliers, voire accidentés. Ils trouvent dès aujourd'hui une application ludo-éducative.
Nous présentons la plateforme Metabot, un robot quadrupède open-source, qui a été développée pour l'éducation. Cette dernière s'inscrit dans le contexte technologique actuel, qui permet, grâce à un accès au prototypage rapide et aux composants sur étagère de construire des robots à pattes autrefois présents uniquement dans les laboratoires. Cette plateforme a été utilisée dans l'enseignement secondaire, afin de permettre à des élèves de découvrir la robotique, ainsi que la programmation.
Nous décrivons par la suite un environnement mis au point dans le but d'étudier la locomotion des robots à pattes, en étendant le contrôleur expert développé sur Metabot à une plus grande famille de robots. Nous avons réalisé une série d'expériences en simulation sur moteur physique que nous avons analysées dans le but de mieux comprendre les règles qui régissent la locomotion des robots à pattes.
Enfin, nous nous intéressons à la locomotion bipède, qui pose le problème de la stabilité. Lors du développement de notre plateforme Sigmaban, un petit robot humanoïde conçu pour participer à la RoboCup, nous avons créé un capteur permettant d'estimer le centre de pression du robot. Nous exploitons ce dernier pour améliorer la stabilité latérale du robot, en créant ainsi une marche en boucle fermée.

Title Optimizing locomotion on low-cost legged robots

Abstract A promise of legged robots is being able walking on irregular or uneven floor. It is already used nowadays in education and entertainment applications.
We introduce the Metabot platform, an open-source quadruped robot developped for education. This takes place in current technological context which allows, thanks to an access to fast prototyping and off-shelf components, building legged robots that were formerly only present in laboratories. This platform was used for teaching in secondary schools, allowing students to discover robotics, and especially programming.
We then describe an environment designed to study legged robots locomotion, extending the expert controller designed for Metabot. We realized some physics simulation experiments and analyzed it to get a better understanding of the legged locomotion underlying rules.
At last, we get a closer look at biped locomotion, for which stability problems arise. When developping our Sigmaban platform, a small-sized humanoid robot created to participate in RoboCup soccer, we designed foot pressure sensors that allow us to estimate the robot center of pressure. We exploit these sensors to improve the lateral stability on the robot, creating a closed-loop walk.

Keywords Robotics, Low-cost, Locomotion, Legged, Gait

Mots-clés Robotique, Bas coût, Locomotion, Pattes, Allure

Laboratoire d'accueil Laboratoire Bordelais de Recherche en Informatique (LaBRI).
Domaine Universitaire
351 cours de la Libération
33405 Talence

Remerciements

Je remercie tout d'abord Olivier Ly, qui a encadré cette thèse, mais qui est aussi à l'origine de notre jeune équipe de robotique. Il a su créer une véritable dynamique nouvelle au sein du laboratoire.

Je remercie Loïc Gondry qui a donné mécaniquement vie à nos robots, Hugo Gimbert qui s'est impliqué tôt dans leur développement.

Je remercie également tous les autres membres de l'équipe Rhoban qui étaient présents au quotidien pour travailler sur nos projets robotiques, dont la RoboCup qui a réussi à nous faire évoluer ensemble vers des objectifs ambitieux : Quentin Rouxel, notre expert en modèle dynamique et odométrique, pour les débats philosophiques sur les fonctionnalités expérimentales de C++, Ludovic Hofer spécialisé dans les modèles statistiques et dans la vision pour son comité poids et mesures et ses e-mail à chapitres, Rémi Fabre qui s'est plongé plus tard dans l'enfer de la vision et qui a un très bon goût pour le nommage des fonctions et des fichiers de configuration, Steve N'Guyen notre bibliographie vivante pour ses critiques, cyniques mais utiles.

Un grand merci aux personnes qui se sont impliquées dans l'association Robot Campus, Guillaume Bea, Pierre Roux, Louis Deguillaume, Pierre Baillache ou encore Laurent Berry qui ont amené avec eux une bonne ambiance dans nos constructions de robots à pattes.

Je tiens aussi à remercier Sylvie Le Laurain et Maïté Labrousse pour avoir réceptionné des centaines de colis pour moi à l'accueil du laboratoire.

Je remercie Denis Lapoire qui s'est impliqué dans le projet Metabot en organisant le Robot Maker's Day, mais qui est aussi à l'origine de l'option robotique à l'ENSEIRB-MATMECA dans laquelle j'ai eu plaisir d'enseigner.

Une pensée également pour tous mes proches qui me soutiennent dans un cadre familial particulièrement agréable, notamment à mes parents pour leur passe efficace de relecture et de corrections.

Enfin, je remercie Jessica qui m'a soutenu et supporté au quotidien pendant notre vie commune qui durera longtemps encore.

Table des matières

Introduction

Histoire et enjeux

Le premier robot bipède, Wabot-1 (Kato [1973]) fut créé par l'université Waseda au Japon en 1973. Le MIT envisage dès les années 1980 d'utiliser des robots à pattes dans le domaine de la conquête spaciale (Angle [1989]). Jusque là, les robots à pattes fonctionnels, peu nombreux, restent principalement des prototypes de laboratoire.

À la fin des années 1990, le robot quadrupède Aïbo est développé et commercialisé par la firme japonaise Sony (Fujita et Kitano [1998]), faisant une avancée dans la démocratisation des robots à pattes en proposant au grand public un animal de compagnie chien-robot. En 1998, après que Kasparov, le meilleur joueur d'échec ait été battu par le superordinateur Deep Blue conçu par IBM (figure 1), les premières rencontres de la RoboCup ont lieu (Kitano *et al.* [1997]). Cet évènement scientifique se présente sous forme de compétitions, dans lesquelles des équipes développent et programment des robots, dont certains sont à pattes. La plateforme Aïbo est d'ailleurs utilisée en premier lieu comme plateforme standard.

La RoboCup annonce comme objectif de créer une équipe de football robotique capable de battre la meilleure équipe humaine d'ici 2050. Ce dernier est aujourd'hui loin d'être atteint, bien que des premiers matchs ont déjà eu lieu à la RoboCup, affrontant des humains contre des robots à roues de la Middle Size League [1] (figure 1).

Ce nouveau challenge se démarque du jeu d'échec principalement par la présence d'environnement bruité que l'on ne connaît que partiellement, et que l'on tente de découvrir à travers l'analyse de capteurs, mais aussi par les difficultés telles que la locomotion, la planification de trajectoire ou encore la synthèse de primitives motrices telles que de frapper une balle avec le pied. Ces dernières se confrontent à un principe de réalité qui mélange physique,

1. La Middle Size League (ou MSL) est une catégorie de la RoboCup dans laquelle deux équipes constituées de jusqu'à 5 robots à roues d'environ 80 cm jouent au football

FIGURE 1 – À gauche, Kasparov qui affronte le superordinateur Deep Blue aux échecs en 1998, et à droite un des premiers matchs humains/robots à la RoboCup, avec les robots à roues de la Middle Size League.

électronique, mécanique, informatique et robotique. L'intégration et la robustesse, omniprésentes, ont également toujours été des enjeux importants de la robotique.

En 2006, le petit robot humanoïde Nao est développé et commercialisé (Gouaillier *et al.* [2009]). Cette plateforme est très utilisée dans le monde de la recherche, et remplace alors le robot Aïbo comme plateforme standard à la RoboCup [2].

De 2013 à 2016, nous [3] avons pris part à cette compétition dans la catégorie humanoïde de petite taille [4] (figure 2), dans laquelle nous avons progressivement développé notre propre plateforme Sigmaban (Fabre *et al.* [2016]), un robot humanoïde d'environ 50 cm de haut et 5 kg. Cette participation nous a permis de nous confronter à l'ensemble des problèmes mentionnés précédemment, dont notamment la locomotion bipède, mais aussi la prise de décisions dans un environnement bruité. Nous avons remporté la 3ème place en 2015 et la 1ère place en 2016 (Allali *et al.* [2016]).

2. La SPL, ou *Standard Platform League* est une ligue de la RoboCup dans laquelle les robots utilisés sont des robots du marché, qui vise à se focaliser sur la programmation de l'intelligence des robots.

3. Notre équipe bordelaise, le Rhoban Football Club

4. La *Humanoid Kid-Size League* est une catégorie humanoïde dans laquelle les robots mesurent entre 40 et 90cm (https://www.robocuphumanoid.org/)

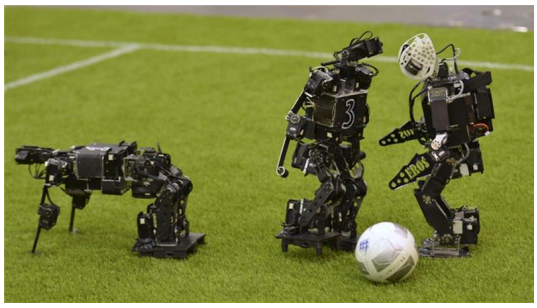


FIGURE 2 – Deux robots Sigmaban (à gauche) affrontant l'équipe indonésienne au cours de la petite finale de la ligue humanoïde de petite taille à Hefei (Chine) en 2015.

La robotique à pattes

La première promesse des robots à pattes est de permettre un accès à des terrains irréguliers, voire accidentés (Brooks et Flynn [1989]). Cette promesse s'inscrit dans un courant de pensée fréquemment décrit comme le fait de faire sortir les robots des usines, où ils agissent dans un environnement parfaitement connu et peuvent représenter un danger pour les opérateurs humains, et de les amener dans d'autres cadres, dans lesquels le degré d'incertitude est plus élevé, afin de les faire travailler en collaboration avec les humains. On parle alors de cobotique.

Dans l'esprit collectif, la robotique à pattes, et plus particulièrement la robotique humanoïde, fait référence au fantasme du robot majordome, qui nous débarasse des tâches les plus répétitives et ennuyeuses de notre quotidien. La première application en la matière se retrouve probablement dans l'accompagnement aux personnes agées. Le robot Romeo d'Aldebaran (Robotics [2010]) est un exemple de projet de recherche dont le but est de réaliser un robot humanoïde d'aide à la personne. C'est d'ailleurs également les enjeux de l'initiative RoboCup@ Home (Wisspeintner *et al.* [2009]), une compétition de robots domestiques dans laquelle ces derniers sont évalués pour leur capacité à aider un humain dans son environnement quotidien.

Le contexte technologique

Aujourd'hui, il est possible non seulement d'accéder à des actuateurs peu coûteux, mais également à des technologies de prototypage. Il est donc possible de concevoir et de construire des robots à pattes à moindre coût (figure 3). Metabot, une plateforme robotique à pattes que nous avons développée dans un cadre ludo-éducatif et qui sera présentée dans cette thèse. Ce robot intègre

12 servomoteurs miniatures et se déplace sur quatre pattes. Il a été reproduit par des utilisateurs sur la base d'un dépôt open-source, qui permet aujourd'hui de partager des conceptions matérielles. La communauté des développeurs qui avait déjà l'habitude de collaborer sur des gros projets logiciels open-source rejoint donc celle du DIY [5], en multipliant les compétences.

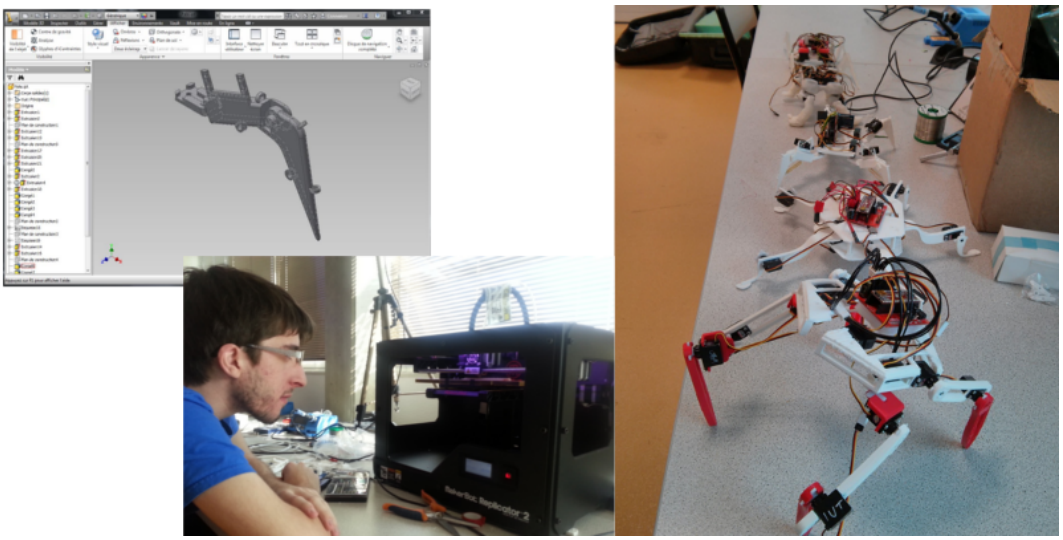


FIGURE 3 – Le prototypage de robots à pattes est aujourd'hui réalisable à l'aide de composants sur étagère et d'imprimantes 3D de bureau. Ces images représentent l'activité de l'association Robot Campus que nous avons créée, invitant les étudiants du campus à venir dessiner et construire leur propre robot à pattes.

Cette robotique bien particulière que l'on peut qualifier de bas coût de par son accessibilité au public est également rendue possible par une démocratisation des technologies de prototypage rapide telles que les imprimantes 3D filament (Jones *et al.* [2011]) mais aussi des produits, notamment électroniques, disponibles sur étagère [6]. La multiplication des FabLabs (Walter-Herrmann et Büching [2014]) jalonne également un renouveau d'interêt pour le *hardware*. Les cours de technologie en collège dans lesquels la conception de pièces 3D assistée par ordinateur (CAO) est un enseignement historique s'équipent désormais d'imprimantes 3D pour matérialiser rapidement, et sans danger les pièces dessinées. Certains parlent d'ailleurs de l'impression 3D comme d'une révolution industrielle (Berman [2012]), dans laquelle la manière de produire et de réparer les objets serait envisagée d'une façon complètement différente

5. Do It Yourself, ou "Faites-le par vous-même", personnes passionnées de bricolage et qui réalisent eux-même des objets

6. Les produits dits "sur étagère" sont produits en masse et facile à se procurer

de celle d'aujourd'hui. Cette idée oppose la fabrication en grande série d'objets similaires, notamment par la technologie d'injection plastique par le moulage, à l'impression 3D qui est, par essence très flexible et modulaire, permettant de créer des objets sur mesure sans investir dans la production d'un moule.

Une communauté autour des robots à pattes

Les robots à pattes qui étaient tout d'abord exclusivement présents dans les laboratoires se sont peu à peu propagés vers un public plus large. C'est dans le but de démocratiser les robots à pattes que nous avons créé le projet Metabot, qui s'est notamment matérialisé sous la forme d'une plateforme robotique à pattes ludo-pédagogique (figure 4).

La difficulté de ce projet est de créer une communauté autour des robots à pattes. Cette communauté est en grande partie intéressée par le caractère anthropomorphe des robots, ou l'imitation du vivant crée de la curiosité.

FIGURE 4 – Aperçu de la plateforme Metabot, version découpée au laser (à gauche) et imprimée en 3D (à droite).

D'un point de vue pédagogique, cette caractéristique est un bon levier pour commencer à aborder des problématiques telles que la physique, les mathématiques ou l'informatique. C'est pour cette raison que notre plateforme trouve une application dans le monde de l'éducation, à travers un environnement que nous avons développé pour permettre à des élèves et étudiants d'apprendre la programmation à l'aide d'un robot, qui joue en fait un rôle de motivation technologique. Ce dernier intègre des moteurs et des capteurs, permettant d'exprimer des concepts abstraites de manière tangible, en illustrant des idées algorithmiques sur le comportement du robot.

Vers une automatisation de la fabrication des robots à pattes

Comme nous l'avons dit, et comme nous le montrerons à travers l'exemple de la plateforme Metabot, il est aujourd'hui de plus en plus facile de construire des robots à pattes, ce qui ouvre la porte de l'exploration, d'un point de vue expérimental, de l'espace des robots à pattes. Une question intéressante est celle de l'automatisation du processus qui permet de construire et de faire marcher ces derniers. Ce processus, nous en avons fait l'expérience à travers un certain nombre de robots. Comment passer d'une plateforme à pattes experte telle que Metabot à une famille de robots la plus grande possible ?

Cette question soulève de nombreux problèmes techniques, mais également des questions plus profondes sur la locomotion, c'est à dire la synthèse d'une primitive motrice permettant de déplacer le robot.

Imaginons par exemple que nous permettions à un utilisateur non-expert de dessiner un robot à pattes à l'aide d'une interface simplifiée dans laquelle il aurait par exemple la possibilité d'assembler des composants standards, éventuellement paramétrés. Le robot ainsi dessiné aurait alors un modèle complètement connu, et le processus de manufacture pourrait être en grande partie automatisé. Cependant, comment faire pour obtenir une locomotion sur ce dernier ?

Nous présenterons un tel environnement, et proposerons un contrôleur générique inspiré de notre expérience ainsi que de la littérature, et le résultat d'expériences qui ont été réalisées en simulation sur un corpus de robots à pattes dans le but de mieux comprendre les règles qui régissent la locomotion de tels robots.

À l'aide de cet outil que nous avons développé, nous avons pu réaliser des optimisations sur un corpus de robots paramétriques. Cette approche nous permet de tester empiriquement un contrôleur sur une grande famille de robots, et d'essayer d'en tirer des invariants, communs à toutes les expériences, permettant de mieux comprendre les lois qui régissent la locomotion à pattes.

Le cas de la locomotion bipède

La locomotion bipède est un cas particulier de la locomotion à pattes, rendu plus difficile par le problème de la stabilité.

Les grands robots humanoïdes tels qu'Asimo (Sakagami *et al.* [2002]) ou

HRP-2 (Kanehira *et al.* [2002]) ont en général une marche calculée hors-ligne, en prenant en compte le modèle dynamique complet et garantissant la stabilité du robot. Ces locomotions ont en général une bonne efficacité, mais ne présentent aucune adaptation à l'environnement ou aux perturbations. D'un autre côté, les petits robots humanoïdes à bas coût que l'on peut par exemple retrouver à la RoboCup se basent sur des contrôleurs plus experts, dont la stabilité repose souvent sur des ajustement empiriques. Ces derniers, en revanche, sont utilisés dans des environnements bruités et perturbés dans lesquels ils doivent prendre des décisions.

Mis à part les démonstrations impressionantes du robot Atlas de Boston Dynamics, peu de robots humanoïdes marchent avec un contrôleur en boucle fermée capable de rejeter des perturbations (Dynamics [2013]).

Ces travaux sont également rendus difficiles dans la pratique à cause de la robustesse (on évite généralement de faire tomber des robots humanoïdes), mais aussi à cause de l'architecture des robots eux mêmes, que l'on voudrait contrôlables en couple, souples et sensibles (Englsberger *et al.* [2014]).

C'est dans cette optique que nous avons développé un capteur à bas coût capable d'estimer le centre de pression d'un petit robot humanoïde, et qui sera présenté dans cette thèse.

Ce capteur a été déployé sur un petit robot humanoïde et utilisé au cours de la compétition de la RoboCup. Nous présenterons une méthode qui permet d'améliorer la stabilité de la marche du robot, créant ainsi un contrôle en boucle fermée.

Chapitre 1

Locomotion à pattes

1.1 Locomotion des animaux

Les premiers travaux concernant l'étude de la locomotion des animaux sont souvent attribués à Muybridge [1899], qui fut le premier à réaliser des photographies séquentielles de plusieurs phases de locomotion chez un grand nombre d'espèces animales (figure 1.1). Dores et déjà, il est possible de constater empiriquement que, malgré la variété d'animaux étudiés, certaines allures, telles que la marche ou le trot, se retrouvent de manière récurrente à travers les espèces.

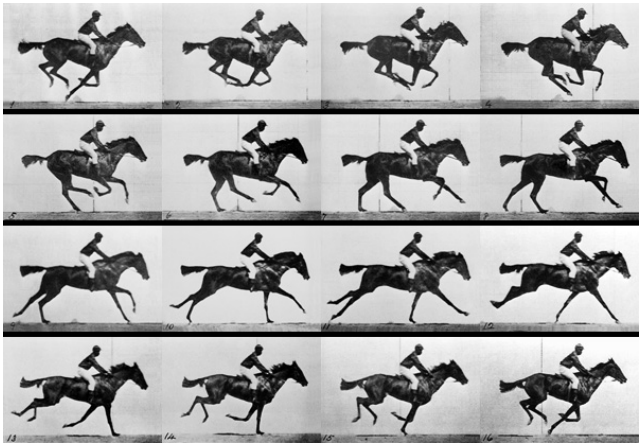


FIGURE 1.1 – Exemple de planche de photographies capturées par Eadweard Muybridge.

Le travail de rationalisation des allures est poursuivi par McGhee [1968], qui propose une formalisation et introduit du vocabulaire pour parler des allures, notamment :

— Le *rapport cyclique* β_i, qui est le temps relatif passé par une patte au sol par rapport à la durée d'un cycle de marche
— Les *phases relatives* τ_i des pattes, désignent le délai entre la pose de la patte 1 et la pose de la patte i
— Une allure est dite *régulière* si les pattes ont toutes le même rapport cyclique, elles font alors toutes le même mouvement à un décalage temporel près
— Une allure est dite *symétrique* si les pattes peuvent être partitionnées en groupes de deux qui ont le même rapport cyclique et qui agissent à une demie phase près

McGhee et Frank [1968b] étudie également la relation entre le critère de stabilité statique et les différentes formes de stabilité quadrupède. Ils utilisent pour cela une représentation simplifiée à la géométrie dans laquelle la surface de sustentation est considérée. La distance longitudinale minimale entre le projeté du centre de masse et la limite de cette surface est utilisé comme objectif d'optimisation (*cf.* figure 1.2). Il est alors montré que la marche quadrupède (*crawling gait*) est l'allure qui satisfait au mieux cette condition, c'est à dire que la marche est l'allure qui garantit la plus grande marge de stabilité statique.

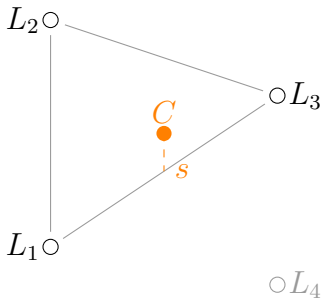


FIGURE 1.2 – Les pattes au sol (L_1, L_2, L_3) forment la surface de sustentation, s est la distance longitudinale entre le projeté du centre de masse C et la zone de sustentation.

Alexander [1984] utilise le nombre de Froude pour étudier les changements d'allures des animaux. Ce nombre sans unité $\frac{v^2}{l}$ met en regard la vitesse de déplacement v à une longueur caractéristique l (en général la longueur de la hanche jusqu'au sol). Il est alors fait l'hypothèse que les animaux adoptent des allures similaires lorsqu'ils ont des nombres de Froude comparables.

Full [1989] montre que des animaux de 2 à 8 pattes peuvent produire des forces de réaction similaires lorsqu'ils se déplacent à une certaine allure sur une plateforme de force, en utilisant deux groupes de pattes alternativement. Il montre également que les animaux ayant une masse plus importante consomment moins d'énergie métabolique pour déplacer un kg de leur corps

sur un mètre de distance (plus l'animal est gros, plus la fréquence à laquelle il marchera sera faible).

Dans le but d'expliquer les similarités observées dans la locomotion des animaux, Blickhan et Full [1993] propose un modèle masse-ressort monopode, qui permet de modéliser une famille de locomotions. Dans ce modèle, l'animal est représenté par une masse m au bout d'un ressort, qui se comprime pendant la phase de support. Pendant la phase de vol, la masse est "lancée" en l'air et soumise uniquement à la gravité.

Ce modèle est retrouvé plus tard sous le nom de SLIP (Spring Loaded Inverted Pendulum), et fait partie des *templates* proposées par Full et Koditschek [1999].

L'étude du système nerveux des animaux a montré l'existence des CPG (*central pattern generators*), des neurones capables de produire des impulsions périodiques (Marder et Bucher [2001]). Ce résultat a été obtenu en comparant les ordres envoyés dans des muscles *in vivo* obtenus à l'aide d'EMG (electromyogramme) avec l'activité des neurones moteurs *in vitro*, en ayant extrait le système nerveux de l'animal. Cette expérience montre qu'il existe un mouvement périodique à l'origine du mouvement, n'étant pas le résultat de stimuli externes auto entretenus (par exemple, une patte qui sentirait qu'elle touche le sol).

1.2 Robotique

1.2.1 Problème de la locomotion

Le problème de la locomotion d'un robot correspond à la création d'un contrôleur capable de produire des déplacements sur un robot. On trouve plusieurs problématiques dans la littérature, que l'on pourrait classifier dans les catégories suivantes :

— **Comment faire avancer le robot sans aucun *a priori* ?** Cette question est en général étudiée du point de vue de l'apprentissage statistique basé sur des réseaux de neurones. Par exemple, Pollack et Lipson [2000] ont développé un système capable de faire co-évoluer morphologie et contrôleur sans *a priori*, obtenant ainsi des créatures manufacturables dévelopées en simulation. Cully *et al.* [2015] ont proposé une solution pour apprendre à des robots à avancer, mais également à s'adapter en cas de dysfonctionnement d'un élément.

— **Comment contrôler le robot ?** C'est à dire par exemple obtenir une vitesse donnée (paramètre dynamique) $(\dot{x}, \dot{y}, \dot{\theta})$. Ce problème donne en général lieu à des approches expertes telles que Hengst *et al.* [2001] ou Hugel *et al.* [2003], un exemple de contrôleur expert développé sur

Aïbo pour la RoboCup, qui sont éventuellement améliorées avec des approches d'optimisation, comme Kohl et Stone [2004].

— **Comment piloter le robot ?** À partir d'un contrôleur et de ses paramètres dynamiques, et éventuellement d'autres informations (limites de vitesse et d'accélération), comment amener le robot à une position cible ? On parlera alors plutôt de planification de trajectoires.

— **Comment franchir des obstacles ?** Ce problème est en général isolé de la primitive de marche, en prenant le contrôle des pattes les unes après les autres. Le robot Little Dog a par exemple été utilisé par Shkolnik *et al.* [2010] pour faire de la planification pas après pas.

Le problème de la synthèse du contrôleur, qui nous intéresse ici, peut aussi s'expliquer simplement en terme de degrés de libertés : une voiture télécommandée, qui disposerait de deux degrés de libertés (la propulsion et la barre de direction), ne laissant pas d'ambiguité quant à son mode de locomotion. En revanche, un robot à pattes ayant douze degrés de libertés : peut éventuellement se déplacer d'une infinité de manières, ce qui en fait un problème sous-contraint, et donc un problème d'optimisation. La présence de ce trop grand nombre de degrés de libertés se traduit en général par l'ajout de contraintes, c'est à dire de critères tels que l'économie d'énergie du robot, et l'ajout de paramètres au niveau du contrôleur, qui permettront d'obtenir des allures différentes.

Remarquons que la présence de pattes n'implique pas forcément un grand nombre de degrés de libertés, un exemple étant le méchanisme de Jansen [2007] qui permet de produire un mouvement similaire à celui d'une patte avec un seul degré de liberté (figure 1.3).

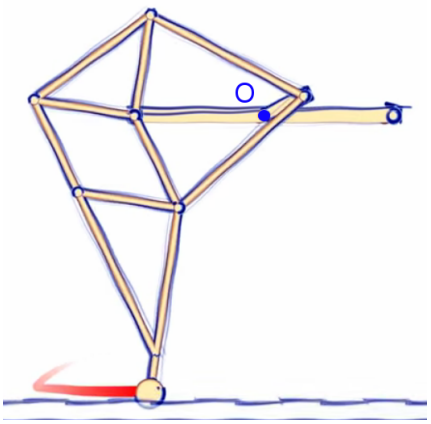


FIGURE 1.3 – Le mécanisme de Jansen produit, à partir d'un mouvement rotatif autour du point O un mouvement similaire à celui d'une patte.

1.2.2 Critères

Des premiers outils importants et récurrents dans toute littérature parlant de locomotion sont les critères de stabilité, qui permettent de s'assurer que le robot ne tombera pas.

Stabilité statique

Le critère de la stabilité statique permet d'assurer que le robot ne tombera pas, en supposant que des mouvements sont suffisament lents pour négliger les aspects dynamiques (il peut donc interrompre sa marche à tout moment sans tomber). Il s'agit de vérifier que la projection du centre de masse sur le sol soit dans la surface de sustentation McGhee et Frank [1968a].

ZMP/CoP

Lorsqu'un robot marche sur le sol, un ensemble de forces de réaction se produit dans sa surface de sustentation. Ces forces sont équivalentes à une seule force résultante, exercée en un point où le moment résultant est nul. Ce point est appelé le centre de pression.

Le ZMP (Vukobratović et Borovac [2004]) est le point sur le sol pour lequel les moments autour des axes X et Y sur le sol générés par la force de réaction sont nuls.

Un robot est dit dynamiquement stable lorsque son ZMP se situe à l'intérieur de sa surface de sustentation. Dans ce cas, le ZMP et le centre de pression coïncident et représentent en fait le même point (Sardain et Bessonnet [2004]).

Lorsque le robot n'est pas dynamiquement stable, c'est-à-dire qu'il "roule" par exemple sur le côté d'un de ses pieds, son centre de pression sera alors sur l'arête autour de laquelle il va rouler, et le ZMP, lui, peut quitter son polygone de sustentation. La position du ZMP par rapport au polygone de sustentation fournit alors une information supplémentaire par rapport au centre de pression (Vukobratović *et al.* [2001]).

On retrouve également l'appelation EMC, pour *Effective Mass Center* (Kang *et al.* [1997]), ou centre de masse effectif, qui traduit plus intuitivement le concept derrière le ZMP qui est peut être considéré comme équivalent au centre de masse dans le cas dynamique.

De la même manière, lorsque ce point est à l'extérieur du polygone de sustentation, le robot va pivoter autour de ce bord. C'est pourquoi le critère

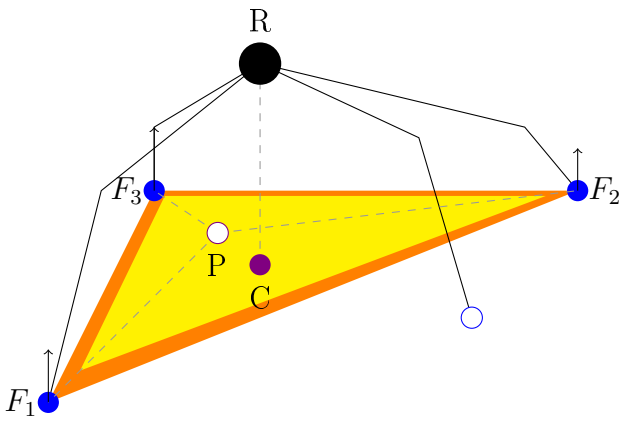


FIGURE 1.4 – Le robot R a trois pattes en contact avec le sol. Ces contacts forment une zone de sustentation (en orange), que l'on réduit généralement pour garder une marge de sécurité (en jaune). Le robot est considéré statiquement stable si le projeté de son centre de masse C est dans la zone, et dynamiquement stable si son centre de pression P est dans la zone. Ce dernier est le point qui annule le moment exercé par les forces de réactions.

proposé est de tenter de garder le ZMP dans le polygone de sustentation.

Pour ces deux derniers critères, on retrouve également en général la notion de marge de sécurité, qui consiste en un rétrécissement du polygone de sustentation permettant de supporter une tolérance aux erreurs du modèle (figure 1.4).

Kajita *et al.* [2001] propose d'approximer le robot par un pendule inversé en 3D (3D-LIPM) avec des jambes téléscopiques (hauteur constante), ce qui permet alors de décrire une approximation du ZMP de manière simplifiée. Remarquons cependant que le calcul du ZMP dépend de l'accélération du centre de gravité, qui peut être difficile à estimer numériquement dans la pratique, si on se base sur une double dérivation de l'état du robot.

Les grands robots humanoïdes, tels qu'Asimo (Sakagami *et al.* [2002]), HRP-2 (Kanehira *et al.* [2002]), WABIAN-2 (Ogura *et al.* [2006]) ou encore plus récemment TORO (Englsberger *et al.* [2014]), intègrent des capteurs de force six axes qui rendent possible l'estimation en temps réel du ZMP.

Cependant, le ZMP est en général utilisé pour créer un contrôleur à l'aide d'une planification hors-ligne, afin de produire un mouvement de marche stable (Kajita *et al.* [2003]).

Cycles limites

Le critère du ZMP impose la contrainte que le pied soit plat sur le sol, et il exclut les marches dans lesquelles le robot roule sur le pied (et donc par exemple le déroulé d'un pied humain). Un robot à pattes qui effectue un trot a également une surface de sustentation petite dans les phases où seulement deux pattes touchent le sol, rendant le critère du ZMP peu adéquat.

McGeer [1990] propose un autre critère, qui permet de quantifier la stabilité globale, et non pas à chaque instant. La méthode proposée est l'étude de la section de Poincaré, qui associe à chaque état du système son état au cycle suivant. La stabilité peut alors être étudiée en linéarisant cette fonction autour d'un point fixe. Le module des valeurs propres de cette application linéaire nous indiquera si elle diverge ou si elle converge vers le point fixe. On parlera alors de "bassins attracteurs", qui confirment si, après chaque pas, le robot se retrouve dans un état stable pour entamer le pas suivant.

Cette technique a été surtout utilisée pour l'étude de la stabilité de marcheurs passifs, qui marchent dans une pente sans aucune forme d'actuation (la marche est purement mécanique) et dans les robots semi-passifs, qui sont partiellement équipés d'actuateurs (Wisse et Hobbelen [2007]).

1.2.3 Robots qui marchent

En complément de la littérature, il est intéréssant de mentionner les prototypes existants qui sont capable de marcher.

Robots humanoïdes

Il existe des grands robots humanoïdes capables de marcher, comme ASIMO de Honda (Sakagami *et al.* [2002]) ou HRP-4 (Kaneko *et al.* [2011]). Ces derniers se basent en général sur des moteurs électriques avec des réducteurs, en général harmoniques, qui minimisent le jeu de l'articulation. Cependant, ces derniers ne laissent aucune place à la passivité : il est impossible de "relâcher" une articulation. De plus, ils adoptent une locomotion "pieds plats" basée sur le respect du critère ZMP.

Le robot Nao (Gouaillier *et al.* [2009]) est un petit robot humanoïde qui a été commercialisé et très utilisé dans la recherche. L'entreprise Aldebaran qui le produit travaille également sur Romeo (Robotics [2010]), un robot à taille humaine (environ 1.5m).

Les robots Flame et Tulip (Hobbelen *et al.* [2008]) sont des exemples de robots semi-passifs. Pour ce faire, une solution mécanique (*Series Elastic Actuator*) proposée dans Pratt et Williamson [1995] est utilisée. Avec cette tech-

nique, le moteur est couplé à l'arbre de sortie à travers un ressort dont la longueur est mesurée, permettant un contrôle en couple et une plus grande passivité.

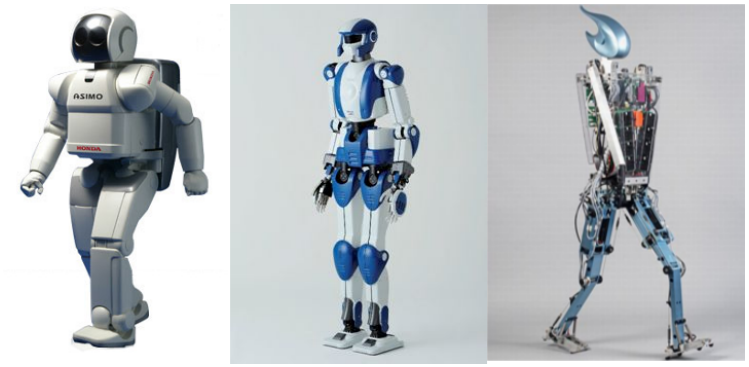


FIGURE 1.5 – De gauche à droite : ASIMO, HRP-4 et Flame.

Big Dog et Little Dog

Big Dog (Raibert *et al.* [2008], figure 1.6) est un robot développé par Boston Dynamics, qui a montré, notamment à travers plusieurs vidéos, sa capacité à marcher sur des terrains irréguliers, voire extrêmes. Leur contrôleur, qui n'est pas très précisément détaillé, est un trot équilibré basé sur le concept de patte virtuelle (*virtual leg* Raibert *et al.* [1986]) qui permet de travailler sur des allures où les pattes sont utilisées par paires (comme le trot, l'amble ou le rebond). Il est dit que sa hauteur et sa posture s'adaptent par rapport au terrain, et que le placement des pattes s'adapte pour compenser l'orientation du robot et du sol par rapport à la gravité.

D'un point de vue mécatronique, le robot est hydraulique, embarquant son propre compresseur et utilise une centrale inertielle et des capteurs de force sous les pattes afin d'identifier les irrégularités du terrain. Il mesure 1 m de haut et pèse une centaine de kg.

Little Dog (figure 1.6) est un robot quadrupède de plus petite taille, basé sur une motorisation électrique. Son architecture permet de réduire la masse placée dans les pattes. A l'instar de Big Dog, il est utilisé dans des travaux de recherche de locomotion sur des terrains irréguliers (comme par exemple Neuhaus *et al.* [2011]).

FIGURE 1.6 – Big Dog (à gauche) et Little Dog (à droite) sont deux célèbres robots de recherche utilisés pour experimenter la marche sur des terrains irréguliers.

RHex

Le robot RHex (Saranli *et al.* [2001]) n'est pas pourvu de réelles pattes, car ce sont en effet des mouvements circulaires qui le font avancer (*cf.* figure 1.7). Ses pattes sont en fait des ressorts qui sont actionnés par des moteurs. Il est important de mentionner sa grande capacité à traverser des terrains accidentés.

Son contrôleur fait agir les pattes alternativement par groupe de trois, mouvement inspiré des insectes.

FIGURE 1.7 – Un aperçu d'une version du robot RHex qui franchit un obstacle.

Il a été montré que, avec un contrôleur bien réglé (Altendorfer *et al.* [2001]), le robot avait un comportement assez proche du SLIP, pendule inversé avec ressorts, connu pour correspondre aux animaux à une certaine allure.

RoboCup

La RoboCup, une compétition internationale de robotique (Kitano *et al.* [1997]) impliquant des robots à pattes, a suscité le développement de contrôleurs sur des robots à pattes tels que le Aïbo (Hengst *et al.* [2001]) et également sur des robots humanoïdes. Si certaines plateformes standard, telles que Aïbo, Nao ou encore DARWIN ont été utilisées dans ce cadre, de nombreux robots

expérimentaux ont aussi été conçus spécialement pour la compétition.

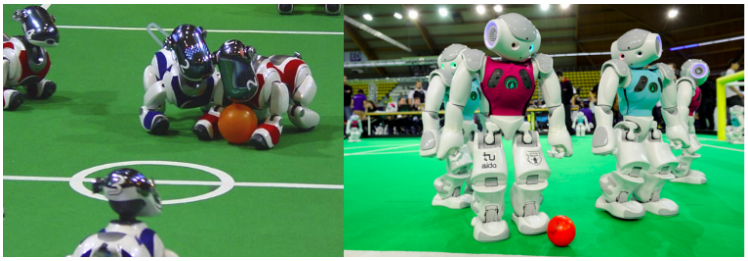


FIGURE 1.8 – À gauche, le robot Aïbo de Sony (1990) et à droite, le robot Nao d'Aldebaran (2006) ont tous deux été des plateformes standards utilisées pour la RoboCup.

Nous avons d'ailleurs développé notre propre contrôleur expert [1] pour le robot humanoïde Sigmaban (Rouxel *et al.* [2015]), basé sur des trajectoires cartésiennes et un modèle géométrique inverse du robot. Ce dernier a été d'ailleurs utilisé de manière opérationnelle pendant les compétitions de 2015 et de 2016.

Ces contrôleurs sont souvent munis d'un certain nombre de paramètres, comme par exemple la fréquence ou la posture du robot, qui forment un grand espace de recherche dans l'optimisation de la vitesse du robot. Ces paramètres, qui sont souvent modifiés manuellement dans un contexte compétitif comme la RoboCup, peuvent aussi être optimisés automatiquement (Kohl et Stone [2004], Jouandeau et Hugel [2013]).

1.3 Robotique évolutionnaire

1.3.1 Apprentissage de la marche

De nombreux travaux sont allés dans la direction de l'apprentissage de la marche sans aucun *a priori*.

Des méthodes statistiques peuvent être utilisées pour cela. Notamment, on peut modéliser le problème sous forme d'apprentissage de politique de transition d'état dans un processus de Markov (par exemple Kimura *et al.* [2001]). Dans cette approche, proche de la théorie des jeux, l'espace de recherche est représenté par un ensemble d'états et un ensemble d'actions, la probabilité

1. Un contrôleur expert est un contrôleur créé sur une base de connaissance empirique et un ensemble d'*a priori*, qui ne sont pas forcément justifiés. Ces contrôleurs proposent en général une série de paramètres qui permettent d'ajuster leur comportement.

d'appliquer une certaine action en fonction de l'état courant est la politique du robot, qui est modifiée selon une fonction de récompense. Une difficulté de cette approche est de bien choisir les états et les actions, mais aussi de les discrétiser correctement [2].

D'un autre côté, on retrouve des méthodes bio-inspirées. Les algorithmes génétiques en sont les exemples les plus standard (Sims [1994]). Dans ces derniers, une population de robots est considérée, appliquant le principe de la sélection naturelle selon le score obtenu par la fonction d'évaluation. À chaque étape, on peut alors croiser des candidats, appliquant éventuellement de la mutation et des enjambements. Un problème est alors de choisir une bonne représentation pour le contrôleur, qui constituera le génotype des robots. Ces derniers reposent en général sur une représentation neuronale, des "briques" élémentaires, telles que des impulsions périodiques, des sinus, des gains etc. qui sont connectées les unes aux autres. L'interêt de ce genre de représentation est de faire évoluer non pas des paramètres mais bien la topologie des ordres moteurs ainsi générés.

1.3.2 Contrôleur et morphologie

Les premiers travaux qui visaient à faire évoluer des morphologies sur des créatures virtuelles se trouvent dans le monde de l'animation graphique, avec Sims [1994] qui présente une manière de faire évoluer simultanément morphologie et contrôleur afin de créer des créatures virtuelles. Le contrôleur est ici constitué de neurones, qui, à partir de signaux d'entrées (capteurs virtuels ou autre neurones) produisent des signaux de sortie pour les effecteurs. Ces créatures sont évaluées à l'aide de simulations physiques. Les créatures produites dans ces travaux ont pour but d'être utilisées dans des animations, et ne sont pas manufacturables, les actuateurs n'étant pas réalistes et les pièces ne correspondant pas à une réalité mécanique. Un exemple plus récent de ce type de travaux est Geijtenbeek *et al.* [2013], qui propose une approche contenant un modèle de muscle pour animer des créatures, avec un résultat assez réaliste.

Le projet GOLEM (Pollack et Lipson [2000]) a été le premier à faire évoluer automatiquement des robots manufacturables. Un exemple de créatures ainsi générées est présenté sur la figure 1.9. Les mouvements produits par son simulateur sont quasi-statiques, ce qui génère des comportements plus faciles à transposer dans la réalité mais aussi généralement plus lents.

Plus récemment, et dans la même lignée, le projet RoboGen (Auerbach *et al.* [2014]) vise à faire évoluer des robots à pattes ou à roues, en améliorant

2. Par exemple, Kimura *et al.* [2001] discrètise l'espace d'état en 256, où les positions angulaires des 8 moteurs sont chacune repréentée par 8 valeurs.

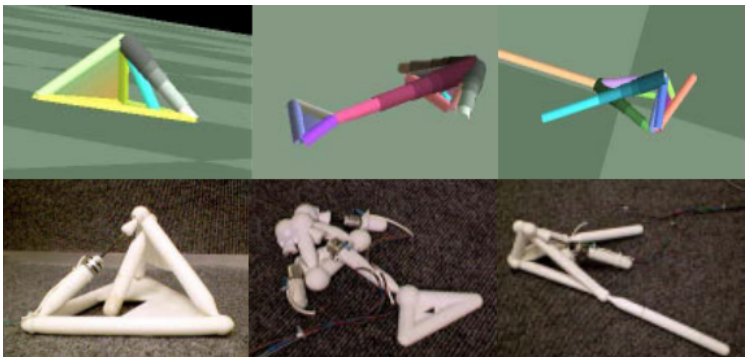


FIGURE 1.9 – Exemples de créatures générées par le projet GOLEM.

simultanément des robots à pattes ou à roues à l'aide d'algorithmes génétiques et de réseaux de neurones.

Megaro *et al.* [2015] a présenté un environnement qui permet de dessiner des robots à pattes manufacturables à l'aide de l'impression 3D et de synthétiser une locomotion, en créant ainsi des robots esthétiques et fonctionnels. L'allure du robot (l'ordre des pattes) est ici précisé par l'utilisateur, et le critère du ZMP est utilisé pour établir la stabilité de la marche.

Les problématiques de morphologies ont déjà été abordées à la RoboCup, notamment dans le cadre de la ligue simulation 3D. Jouandeau et Hugel [2013] propose par exemple une méthode d'optimisation simultanée du contrôleur et de la morphologie d'un robot Nao en simulation. Cette méthode se base sur l'utilisation d'un algorithme d'optimisation en boîte noire, CLOP (Coulom [2011]).

Une problématique différente, mais connexe proposée par Cully *et al.* [2015] est d'essayer de faire marcher des robots endommagés, c'est à dire auxquels on a par exemple retiré une patte. Une notion intéréssante présentée est la mise à jour permanente des informations du robot sur la connaissance des locomotions possibles qu'il peut adopter, en prenant en compte la dualité entre la performance et la confiance en ses informations.

Chapitre 2

La plateforme Metabot

Le robot Metabot est né dans un contexte informel, les réunions d'une association d'étudiants sur le campus de l'université de Bordeaux dont le but était de construire des robots à pattes. Tirant profit des technologies de prototypage rapide, un certain nombre de prototypes ont été conçus (certains sont montrés en figure 2.1).

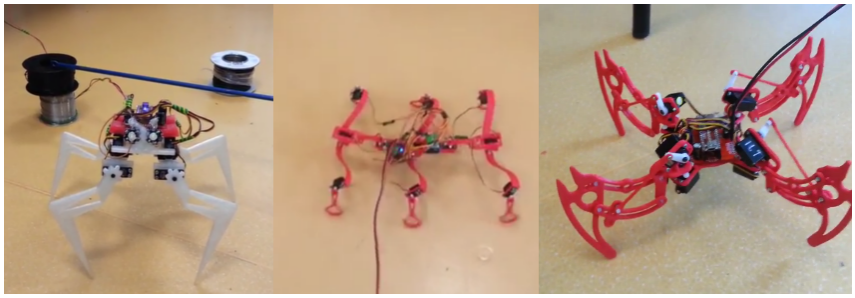


FIGURE 2.1 – Exemples de prototypes conçus dans le cadre associatif.

Cette expérience a créé une base empirique de connaissance au sein de notre équipe sur le fonctionnement des robots à pattes et leur conception. Elle nous a également permi de converger vers une architecture experte efficace de robot quadrupède. Cette plateforme a été par la suite partagée en open-source, développée et commercialisée sous forme de kit (figure 2.2).

Dans ce chapitre, nous présenterons de manière descriptive l'architecture de la plateforme, ainsi que l'environnement logiciel qui a été mis en place qui permet à un utilisateur non expert de piloter et de programmer des comportements sur le robot à l'aide d'un langage visuel simplifié, et du retour sur expérience de son utilisation lors d'une compétition de robotique organisée localement.

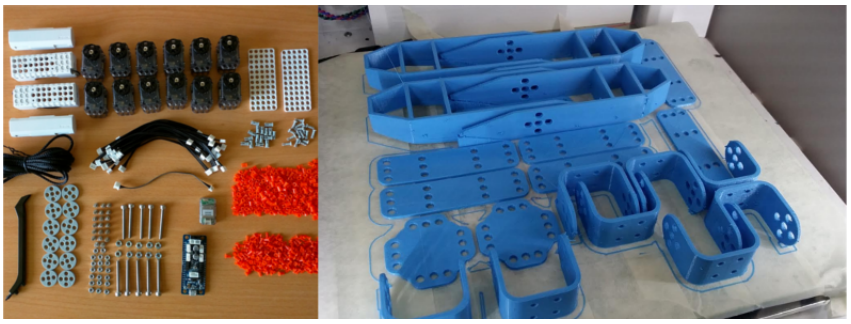


FIGURE 2.2 – Développé à l'origine dans un cadre associatif et ensuite partagé en open-source, la plateforme Metabot est maintenant commercialisée sous forme de kit à assembler.

Le développement de la plateforme soulève le problème du démarrage d'un projet robotique, de par ses aspects industriels, ou comment produire progressivement un robot à bas coût, mais aussi de la mise en place d'une communauté.

Le projet Metabot est, quant à lui, plus large, car il vise également à pouvoir automatiser le processus de création des robots à pattes, ce qui sera approfondi dans le chapitre suivant.

2.1 Contexte

2.1.1 Technologie

Nous avons développé une plateforme robotique à pattes ludo-éducative qui a été commercialisée et déployée dans des enseignements (Passault *et al.* [2016b]). Un aperçu de cette plateforme se trouve sur la figure 4 de l'introduction.

Ce travail est un exemple de robot à pattes non seulement à bas coût, mais aussi open-source et donc reproductible et modifiable par la communauté [1].

Ceci est rendu possible par le contexte technologique, notamment :
— La baisse des coûts de la miniaturisation des micro-contrôleurs et des composants tels que les puces bluetooth, les centrales inertielles ou les batteries lithium
— L'accès à des produits "sur étagère", tels que des servomoteurs ou des composants électroniques

1. https://github.com/Rhoban/Metabot/

— La démocratisation des machines outils de prototypage rapide, comme les imprimantes 3D (Jones *et al.* [2011]) ou les découpeuses laser
— L'aube du partage virtuel d'objet physiques et la multiplication des Fablabs (Walter-Herrmann et Büching [2014])

2.1.2 Social

L'apprentissage de la programmation se développe de plus en plus tôt dans le cursus scolaire.

En France, le programme de Cycle 2 (écoles primaires) de 2015 propose déjà une découverte de la notion d'algorithme, et présente comme exemple d'activité : "Programmer les déplacements d'un robot ou ceux d'un personnage sur un écran" (Education [2015]).

Le programme de Cycle 4 (collèges) de 2016 introduit des projets interdisciplinaires (EPI, enseignements pratiques interdisciplinaires), dont le but est de relier des disciplines et d'appliquer des notions théoriques.

Au lycée, un enseignement exploratoire "informatique et création numérique" a été ajouté en 2015, dont le but est de fournir un aperçu des métiers du numérique aux élèves durant la classe de seconde.

Ces mesures mettent en lumière la volonté de mettre en contact les élèves avec la programmation assez tôt dans leur cursus, mais aussi de leur faire mettre en pratique leurs connaissances dans des projets pluridisciplinaires. Les robots répondent naturellement à ces deux problématiques, de par la diversité des thématiques qu'ils représentent (géométrie, physique, mécatronique...), mais également relativement à la programmation qui est au coeur de leur fonctionnement.

2.2 Architecture

2.2.1 Matériel

Le robot est constitué de douze degrés de libertés, trois moteurs par patte, permettant à chacune d'elles d'atteindre une partie de l'espace environnant.

Actuateurs

Les actuateurs utilisés sont des servomoteurs XL-320, qui peuvent fournir un couple de 0.39 N.m à l'arrêt et une vitesse à vide de 114 tr/min et pesant 16.7g (figure 2.3). Un des éléments qui permet de réduire le coût des moteurs

est la boîte à engrenages qui est en plastique. La fragilisation que ce choix entraîne dans le servomoteur est compensée par un petit mécanisme en plastique situé entre le dernier pignon de l'étage de réduction et l'arbre de sortie qui agit comme un limitateur de couple et "saute" des crans au lieu de se casser (figure 2.4).

Ces servomoteurs communiquent via un bus série *half-duplex*, ce qui permet de les connecter en cascade et d'améliorer l'intégration.

FIGURE 2.3 – Le servomoteur XL-320 utilisé dans Metabot.

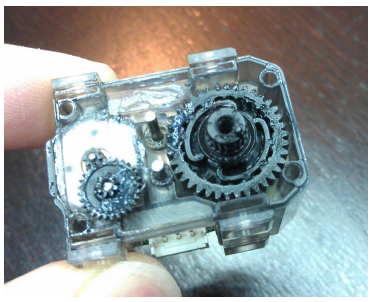


FIGURE 2.4 – Les engrenages internes en plastiques du XL-320.

Electronique

À bord du robot, une carte électronique est dotée d'un petit microcontrôleur 32 bits cadencé à 72 Mhz [2], d'une puce Bluetooth d'une centrale inertielle et d'un capteur de distance frontal (figure 2.5).

L'utilisation de processeur beaucoup plus puissants qui embarqueraient un système d'exploitation (type Linux) serait techniquement envisageable. La Raspberry Pi (Upton et Halfacree [2012]) en est un exemple. Cela permettrait d'ouvrir de multiples possibilités sur ce genre de petits robots, en amenant une puissance de calcul plus grande et en permettant de profiter des avantages tels que le système de fichiers ou l'accès facilité à internet, en embarquant l'équivalent d'un petit ordinateur sur le robot. Cependant, sans parler de l'intégration, ces systèmes ont encore quelques défauts. Premièrement, la consommation qui atteint rapidement plusieurs centaines de mA, mais aussi les problèmes inhérents à l'utilisation du système lui même. Par exemple, Metabot met 2 secondes à démarrer une fois la batterie branchée, et ne peut pas être vulnérable à une corruption de système de fichiers. Aujourd'hui un compromis, il est malgré tout raisonnable de penser que dans le futur, il deviendra

2. ARM Cortex-M3, un processeur embarqué très répandu

de plus en plus intéressant d'intégrer ce type de systèmes dans des projets tels que Metabot.

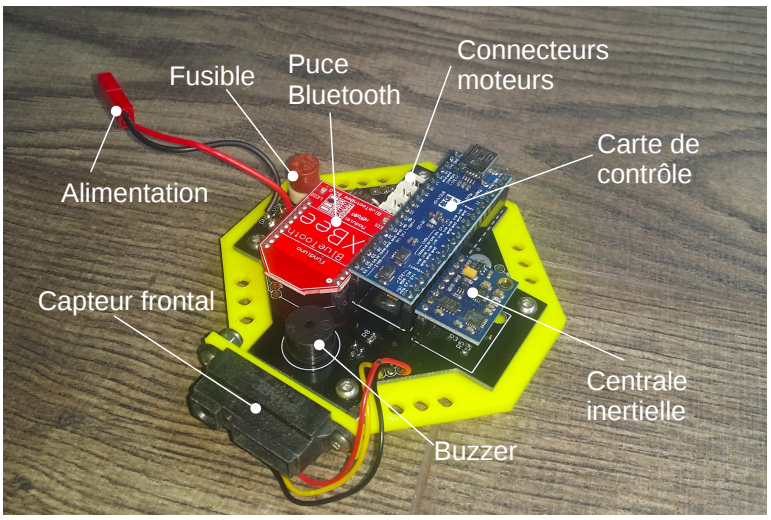


FIGURE 2.5 – La carte électronique de Metabot avec ses composants montés.

Sa batterie est une lithium-polymère 7.4V de 1000 mAh, qui lui fournit une autonomie d'environ 45 minutes. Afin de rendre son utilisation moins dangereuse, le robot est équipé d'un fusible rapide. Si la batterie est trop déchargée, une alerte sonne sur son buzzer pour prévenir l'utilisateur. Si ce dernier ne débranche pas la batterie, le robot peut alors faire volontairement exploser le fusible à l'aide d'un transistor prévu à cet effet, laissant ainsi le circuit ouvert et évitant les risques liés à la décharge trop importante de ces batteries.

Mécanique

Les pièces mécaniques du robot ont été conçues en deux versions, une destinée à l'impression 3D (figure 2.6) et l'autre découpée au laser (figure 2.7).

La découpe laser permet, quant à elle, d'atteindre des vitesses de production sans pour autant nécessiter l'investissement de la production d'un moule pour l'injection plastique. Un matériau adéquat est le PMMA (acrylique ou encore plexiglas), d'une épaisseur d'environ 3mm. Ce dernier est facile à trouver sur le marché en de nombreux coloris, et est réputé pour bien se découper au laser et peut se plier à chaud. Ce processus peut se faire à bas coût en prototypage, à l'aide d'un gabarit (figure 2.8), et aussi être sous-traîté en production. Le pliage, qui ajoute une étape dans le processus et fragilise les pièces, peut être réduit en utilisant des entretoises et des longues vis, comme présenté sur

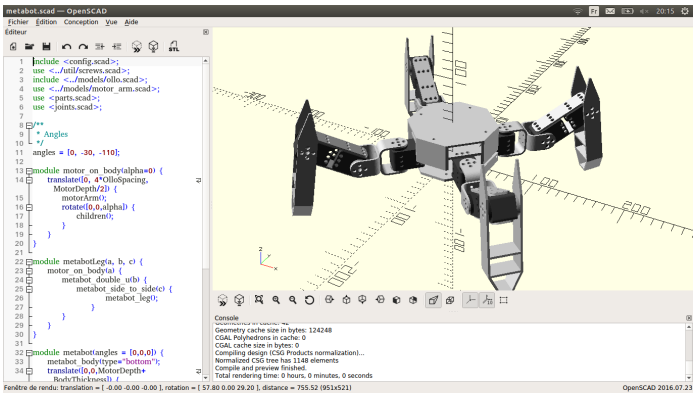


FIGURE 2.6 – Les pièces 3D du robot Metabot.

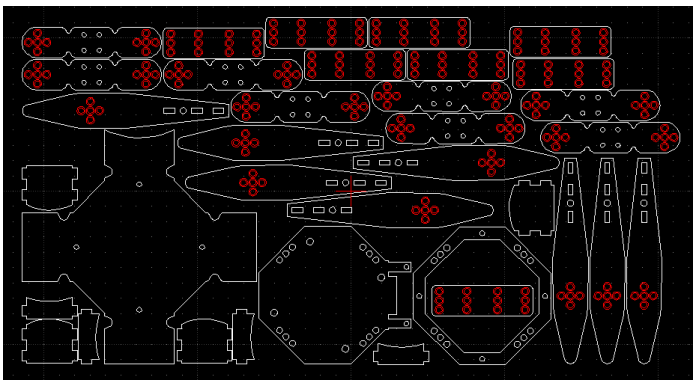


FIGURE 2.7 – Motif à découper au laser pour produire les pièces du robot.

la figure 2.9.

Les pièces, qu'elles soient 3D ou laser, sont assemblées aux moteurs à l'aide de rivets plastiques, qui facilitent le montage et le démontage du robot. Seules quelques vis maintiennent les pièces du corps ensemble. Le système de rivets nécessite un ajustement, afin que ces derniers rentrent un peu en force, limitant ainsi le jeu. Il y a environ 250 rivets dans l'assemblage d'un robot Metabot.

2.2.2 Contrôleur

Un contrôleur expert a été développé afin d'assurer la locomotion du robot. Le modèle géométrique inverse des pattes a été résolu symboliquement, afin de pouvoir contrôler le bout de la patte dans l'espace cartésien. Le modèle cinématique d'une patte du robot est montré sur la figure 2.10.

Les paramètres dynamiques du contrôleurs sont $(\dot{x}, \dot{y}, \dot{\theta})$, respectivement la

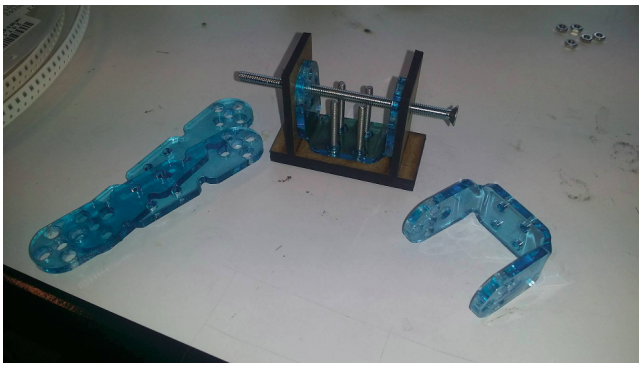


FIGURE 2.8 – Processus de pli d'une pièce du robot en plexiglas découpé au laser à l'aide d'un gabarit.

FIGURE 2.9 – Une longue vis et un emboîtement permet d'assembler la patte de Metabot laser sans pliage.

vitesse d'avance, la vitesse latérale et la vitesse de rotation.

Lorsque le robot est à l'arrêt (les paramètres dynamiques sont nuls), une position cible est donnée aux pattes, c'est la posture statique du robot. Cette dernière représente des paramètres du contrôleur, qui sont la hauteur du tronc, la distance des pattes au centre du robot et un niveau de "crabe" qui permet d'écarter les pattes longitudinalement ou latéralement. Ces trois degrés de libertés permettent de placer le robot dans n'importe quelle position symétrique selon le plan sagittal et le plan frontal (figure 2.11).

Lorsque le robot se déplace, les pattes suivent une trajectoire autour de cette position. Là encore, des paramètres permettent de modifier cette trajectoire : la hauteur à laquelle les pattes se lèvent, et la fréquence de marche (nombre de cycles de marche par seconde). Une approche plus générique pour le développement du contrôleur sera décrite plus précisément en 3.3.2.

Les pattes sont coordonnées les unes par rapport aux autres avec l'ordre

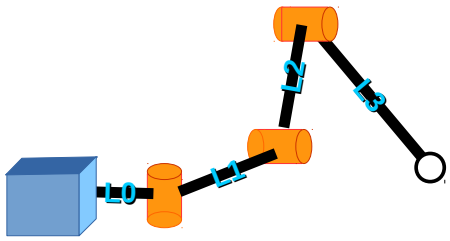


FIGURE 2.10 – Modèle cinématique d'une patte du robot Metabot.

du trot, dans lequel les pattes diagonalement opposée se déplacent de manière synchronisée. Malgré son apparente instabilité, il s'est révélé être un mode de déplacement efficace dans la pratique.

Le robot peut se déplacer de manière holonome et atteindre des vitesses de l'ordre de 40 cm/s. Il est en fait symétrique, il y a quatre "avants" possible, qui représentent un choix arbitraire qui peut être défini logiciellement, matérialisé par l'allumage des LEDs des pattes avant. Malgré l'aspect pratique que cela a pour piloter le robot, cette notion entraînait cependant une confusion dans l'esprit des utilisateurs. Nous avons ajouté des yeux sur la deuxième version afin de résoudre anthropomorphiquement la question du sens du robot qui était un problème dans la première version.

2.2.3 Application mobile

Le robot étant équipé de Bluetooth, il est possible de le piloter depuis un téléphone à l'aide d'une application. L'écran présenté en figure 2.12 permet à l'utilisateur de régler les paramètres du contrôleur du robot et illustre le problème d'optimisation de la marche. Ces paramètres sont ceux présentés précédemment.

Dans la pratique, la locomotion du robot nécessite éventuellement des ajustements selon le type de surface, la pente, le franchissement de petits obstacles. L'ajustement des paramètres du contrôleur délègue à l'utilisateur la possibilité d'optimiser la locomotion selon la situation, tirant alors profit des degrés de liberté du robot.

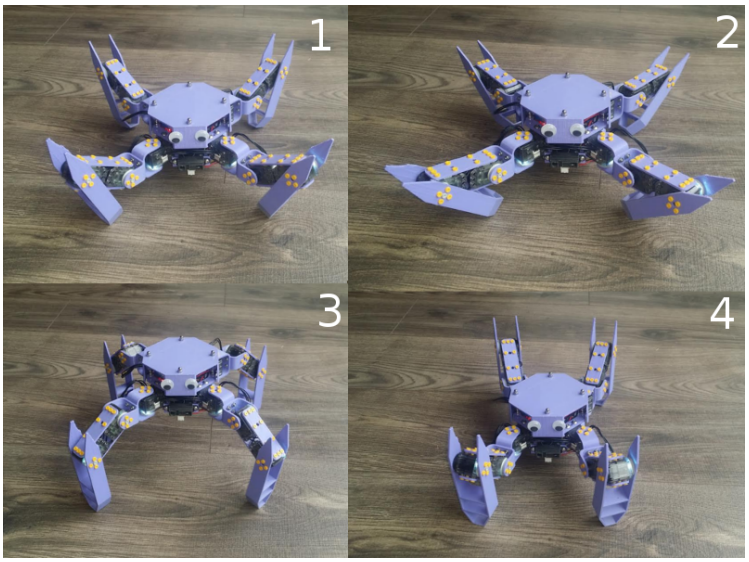


FIGURE 2.11 – Illustration des paramètres de postures, en 1, le robot dans sa position initiale, en 2, le rayon des pattes a été réduit, en bas à gauche, en 3, la hauteur du robot est augmentée, en 4 les pattes sont placées en crabe vers l'avant.

2.3 Environnement de programmation

2.3.1 Approche et problématique

Le robot Metabot trouve une application ludo-pédagogique. L'aspect pluridisciplinaire de la robotique permet d'aborder de multiples sujets d'un point de vue éducatif. Une plateforme à patte représente un objet ludique et attractif et ouvre de nombreuses perspectives pédagogiques. Cependant, cette approche pose à la fois le problème de la découverte de la programmation et celui de la programmation des robots.

Apprentissage de la programmation

La manière d'aborder la programmation est une discussion récurrente, quel que soit l'âge, le niveau ou les prérequis des élèves ou étudiants. Bien que l'algorithmique soit un sujet qui peut être discuté indépendamment, la mise en oeuvre d'un programme requiert un langage de programmation ancré dans une technologie, avec une syntaxe et en général une multitude de détails techniques intrinsèques.

Cette réflexion est d'ailleurs présente dans les écoles informatiques supérieures, dans le choix du tout premier langage avec lequel les étudiants seront en

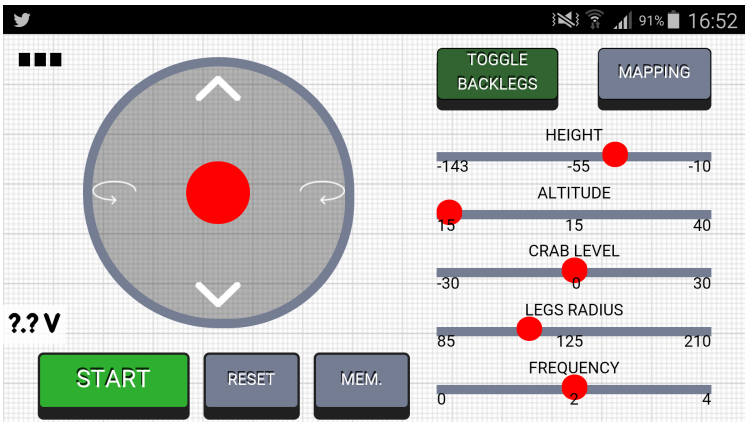


FIGURE 2.12 – Capture d'écran de l'application mobile qui permet de contrôler le robot.

contact. Faut t-il commencer par un langage comme le C, proche du système, fortement typé et compilé, ou au contraire par un langage interprété moins contraignant, comme Python ? Dans le premier cas, les étudiants devront faire face très vite aux problèmes liés au fonctionnement d'un ordinateur, tels que la gestion de la mémoire, en plus de découvrir les concepts théoriques liés à la programmation et à l'algorithmique. Dans le second, la permissivité du langage utilisé risque de les éloigner de la compréhension de la machine. La figure 2.13 propose une schématisation de ce compromi entre la compréhension algorithmique et la compréhension de l'ordinateur :

— **L'algorithmique** : discipline purement mathématique qui peut être abordée de manière indépendante
— **Langages (interprétés et compilés)** : comment exprimer un algorithme dans un langage de programmation
— **Système** : comment sont gérés la mémoire, les processus concurrents, le réseau...
— **Bas niveau** : au niveau électronique, comment fonctionne le processeur pour exécuter le code ou communiquer avec les périphériques, comment sont encodées les données...

À un niveau moins avancé (collège et lycée), l'utilisation d'un langage de programmation, même permissif et simple, pose toujours le problème de l'apprentissage de la syntaxe, qui est un frein dans l'enseignement. En effet, le but de la découverte de la programmation est de présenter les concepts théoriques sous-jacents, et d'éviter autant que possible de se perdre dans des détails syntaxiques ou d'implémentation. Une solution connue à ce problème est l'utilisation de langages de programmation visuels, qui permettent de faire abstraction

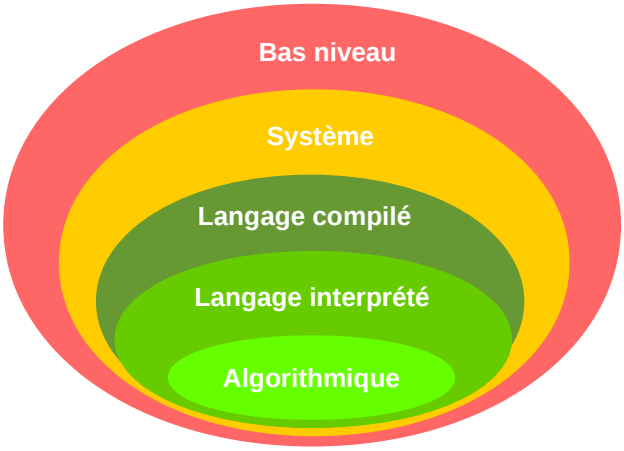


FIGURE 2.13 – Diagramme reflétant le compromis entre l'apprentissage de l'algorithmique et la compréhension sous-jacente du fonctionnement de l'ordinateur.

de la syntaxe.

Ces langages de programmation visuelle ont été tout d'abord proposés par le MIT avec StarLogo puis Scratch (Resnick *et al.* [2009]), qui est aujourd'hui le plus connu et le plus répandu. Ce dernier a insipré App Inventor, une application conçue par le MIT et Google pour concevoir des applications mobiles. Google a alors développé Blockly (Google [2012]), une bibliothèque permettant de construire des langage visuels. Ces technologies sont des outils web, basées sur Flash (Scratch) ou JavaScript (Blockly).

Choregraphe, le logiciel propriétaire permettant de programmer le robot Nao, est aussi un langage de programmation visuel qui inclut un éditeur de machines à états et un séquenceur (Pot *et al.* [2009]). L'édition de code à proprement parler se fait par l'édition de scripts (figure 2.14).

Lego Mindstorm propose également un environnement de programmation propriétaire basé sur un éditeur visuel, dont le concept est proche des machines à états.

Actuellement, Blockly est probablement le meilleur choix pour interfacer de la logique utilisateur, comme personnaliser les blocs et la production de code, car cet outil a été conçu avec les développeurs de langages visuels pour principale cible, permettant une grande flexibilité. Scratch étant en fait une plateforme, il n'est pas possible d'embarquer son éxecution à bord du robot.

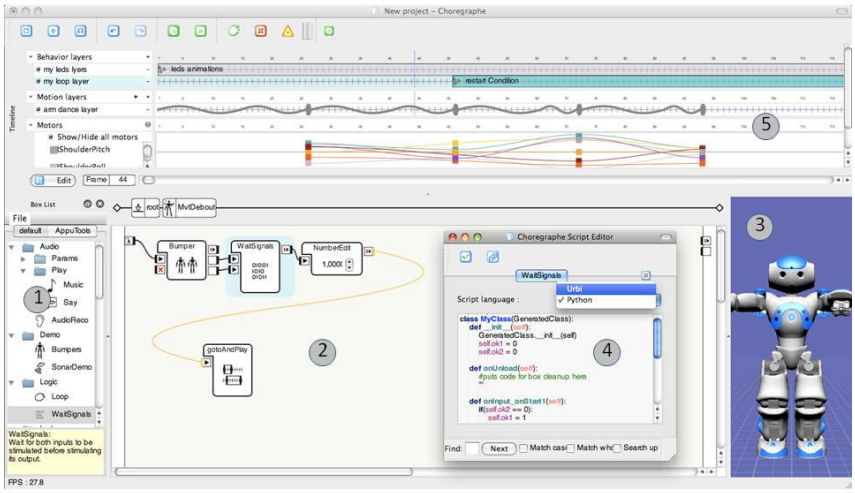


FIGURE 2.14 – Aperçu du logiciel permettant de programmer le robot Nao (de Pot *et al.* [2009]). Cet éditeur propose de programmer des comportements à l'aide de machines à états (2), d'un séquenceur (5) pour éditer des signaux, un éditeur de script (4) permettant aux utilisateurs d'écrire du code et d'un simulateur (3).

Blockly (tout comme Scratch) propose un éditeur basé sur des blocs qui s'emboîtent et s'imbriquent, pouvant alors représenter des séquences, des boucles, des conditions, et toute structure que l'on peut retrouver dans un langage de programmation, en faisant complètement abstraction de la syntaxe (figure 2.15).

En réalité, ces blocs sont très analogues à un langage de programmation, dans lequel la notion de syntaxe a été éludée. L'utilisateur qui conçoit un programme visuel fait plus de travail qu'un utilisateur qui écrit du code, car il fournit l'arbre syntaxique complet de son programme. De plus, la compatibilité des éléments peut être vérifié à la volée, permettant de faire une vérification de type. De cette manière, il n'y a pas d'erreur de syntaxe possible, permettant de se focaliser sur d'autres notions, telles que les bases de l'algorithmique ou le concept de variables, et mettant donc de côté un obstacle important dans la découverte de la programmation.

Une limite de cette approche se retrouve par exemple dans l'écriture d'équa-

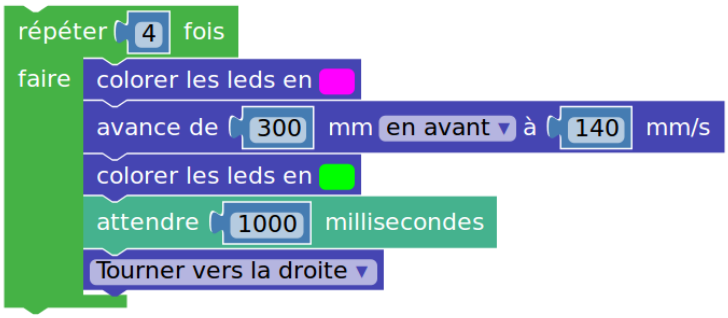


FIGURE 2.15 – Un exemple de programme écrit avec des blocs, on y observe la notion de séquence de boucles.

tions mathématiques, qui ont un arbre syntaxique gros et requièrent donc un grand nombre de blocs (figure 2.16).

FIGURE 2.16 – L'expression $cos(x) + sin(x) * 5$ représentée avec des blocs est très verbeuse.

Programmation de robots

La programmation de robots est une tâche difficile, même pour un utilisateur expert, car l'ensemble des couches présentées sur la figure 2.13 doit être maîtrisé.

Une manière de présenter la difficulté dans le fait de programmer un robot peut être illustrée à travers notre expérience lors de nos participations à la RoboCup, en se confrontant directement au problème de réaliser des robots capables de jouer au football. Après plusieurs années de participation, l'architecture logicielle du robot a évoluée vers une organisation adaptée aux problèmes liés au développement sur un robot. Cette dernière est composée des briques suivantes (présentées sur la figure 2.17) :

— **Les comportements décisionnels**, sont des parties du logiciel responsables de prendre des décisions de haut niveau pour déterminer les actions que le robot veut entreprendre, comme marcher vers la balle, se relever ou attendre. Elles sont en général implémentées par des machines à état, qui sont un outil de représentation pratique du comportement.

— **Les comportements moteurs** produisent des ordres pour les actuateurs du robot, comme par exemple le contrôleur de marche qui est responsable de produire une locomotion à partir des paramètres dynamiques, ou le relevage qui relèvera le robot. Ils s'appuient eux même en général sur des outils de signal, tels que des splines, des moyennes escomptées ou des hysteresis.
Chaque comportement (moteur ou décisionnel) a la particularité de pouvoir être soit lancé soit coupé à tout moment. Il peut également communiquer avec d'autres comportements, mais aussi lancer et couper d'autres comportements. Par exemple le comportement d'approche de la balle lancera la marche et la contrôlera.

— **Les services**, sont des composants du robot toujours actifs dont le but est de produire des informations pour les comportements. Par exemple, le modèle du robot calcule en permanence l'estimation de l'odométrie et publie une position du robot dans le repère du monde.

— Enfin, le **bas niveau** est responsable de communiquer avec les actuateurs et les capteurs, en écrivant et en lisant les valeurs nécessaires dans les registres des contrôleurs embarqués dans les actuateurs et les capteurs.

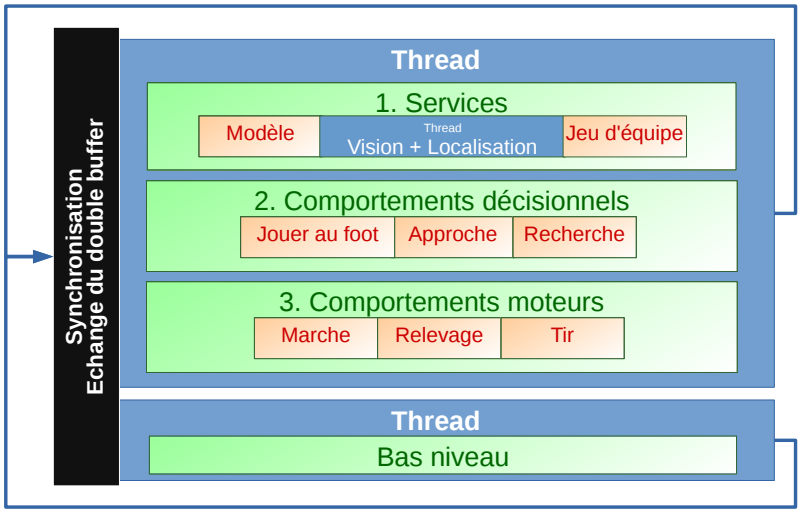


FIGURE 2.17 – Architecture globale utilisée à la RoboCup, avec des exemples de comportements et de services.

La programmation des robots est une thématique abordée dans la littérature. Baillie [2005] a proposé URBI, un langage dont le but est de favoriser la programmation de robots. Ce dernier introduit des concepts qui répondent

aux problèmes standard que l'on rencontre pour programmer un robot. La syntaxe de ce langage était proche du C, avec des mots clés supplémentaires qui permettaient de gérer la temporisation d'un changement de valeur ou le lancement et l'arrêt de tâches parallèles de manière simplifiée. Cependant, le support de ce langage a été arrêté.

ROS (Quigley *et al.* [2009]) est un ensemble de composants logiciels (*middleware*) qui favorisent également le développement de programmes pour des robots, en permettant principalement de séparer le code en différents modules qui communiquent à travers des messages. Son avantage principal est de permettre de distribuer les calculs et les services sur plusieurs processeurs et/ou machines.

Il est évident que ces problèmes de génie logiciel complexes doivent être cachés aux yeux de l'utilisateur non expert qui programme un robot. Pour faire l'analogie avec l'architecture présentée précédemment, il faudrait lui permettre de modifier les comportement décisionnels, en lui fournissant toutes les autres briques de manière transparente.

De plus, une contrainte supplémentaire est que le code qu'il écrit doit être sécurisé. Il faut en effet s'assurer qu'il ne pourra par exemple pas bloquer complètement le processeur du robot, que ce soit en demandant trop de ressources (de mémoire ou de calcul) ou en provoquant un bug (e.g. accès à une mauvaise zone de la mémoire). C'est pour cela qu'il vaut mieux la cloisonner dans un bac à sable.

Enfin, en travaillant sur des robots à bas coût avec des petits microcontrôleurs, le code utilisateur doit être léger en mémoire.

Une solution proposée, qui répond à ces problématiques est de créer une machine virtuelle simple, et de traduire le programme de l'utilisateur (par exemple une scène de blocs emboîtés) en un code binaire. Ce dernier peut être très léger en mémoire et l'utilisation de la machine virtuelle permet une exécution complètement sécurisée et cloisonnée.

Cette approche a été proposée dans Magnenat *et al.* [2011], avec ASEBA, une machine virtuelle légère basée sur les événements et capable d'éxecuter des comportements. Ce système a d'ailleurs été utilisé sur le robot Thymio (figure 2.18), un robot ludo-éducatif, afin de permettre aux utilisateurs de concevoir des programmes pour le robot.

FIGURE 2.18 – Le robot Thymio et son environnement de programmation simplifié basé sur des événements.

Cycle essai/erreur

Un avantage de l'informatique est l'aspect virtuel, qui permet de tester des morceaux de programmes (contrairement par exemple à la production d'une carte électronique). Ces cycles d'essais et d'erreurs permettent d'expérimenter, de prototyper et de découvrir.

Lors de la création d'un programme en utilisant un éditeur visuel, les blocs sont instanciés depuis un catalogue (figure 2.19). Ce mécanisme favorise l'exploration des différentes possibilités du langage sans à avoir à consulter une documentation.

Cependant, tester un comportement sur un robot peut s'avérer plus long et problématique que de tester un programme. Une pratique omniprésente en robotique est la mise en place d'un environnement de simulation permettant de tester des comportements avant de les déployer sur les robots réels. V-REP (Rohmer *et al.* [2013]) et Gazebo (Koenig et Howard [2004]) sont des exemples de ces environnements permettant de simuler la physique, mais également des capteurs ou encore des caméras.

2.3.2 Architecture proposée

En prenant en compte les considérations évoquées dans les parties précédentes, nous avons donc argumenté en faveur de l'utilisation d'un langage de programmation visuel, qui peut être interprété en utilisant une machine virtuelle à bord du robot, pour lequel il est également possible de réaliser des simulations.

L'architecture proposée est alors un environnement dans lequel la machine virtuelle et le comportement du robot (incluant son contrôleur) sont à la fois

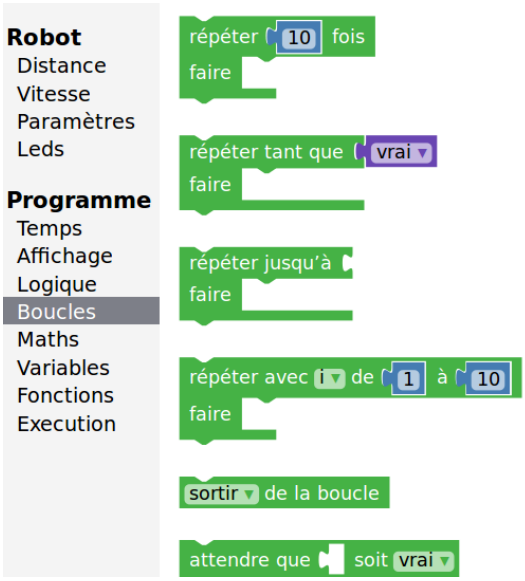


FIGURE 2.19 – Un exemple de panneau contenant des blocs "prototype" qu'il est possible d'instancier dans le programme que l'on construit avec Blockly pour Metabot.

présents dans le robot et dans l'environnement de développement, permettant de simuler le comportement du robot (figure 2.20).

En l'occurence, cela est rendu possible car ces derniers sont tous les deux écrits en C/C++, et cross-compilés soit pour le robot avec la chaîne de compilation ARM soit pour le navigateur web avec Emscripten, outil permettant de compiler du code LLVM en JavaScript (Zakai [2011]).

Le comportement du robot est simulé en exécutant le code dans la machine virtuelle qui est inclue dans l'environnement de programmation, qui permet d'obtenir les angles cibles pour la position de ses articulations. Ses déplacements sont simulés géométriquement en récupérant les paramètres dynamiques donnés au contrôleur. L'interface de programmation permet alors de basculer entre la simulation et la programmation du robot réel (figure 2.21).

Les programmes peuvent être chargés à bord du robot et écrits dans sa mémoire (flash), le rendant complètement autonome dans l'exécution de son code.

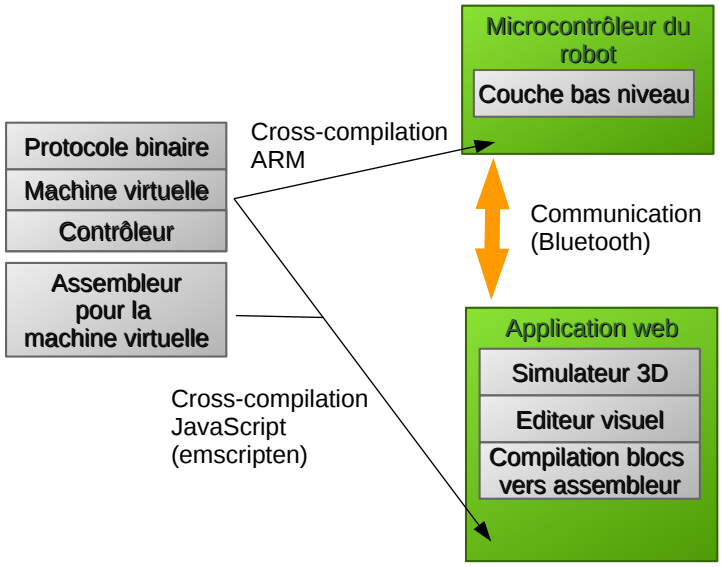


FIGURE 2.20 – L'architecture logicielle utilisée : le code du contrôleur et de la machine virtuelle sont communs entre le robot et l'environnement de programmation.

2.3.3 Machine virtuelle

Nous avons développé une machine virtuelle très légère dans le but d'interpréter des programmes à bord du robot.

La machine virtuelle permet de cloisonner le code utilisateur dans un bac à sable. Dans cette optique, une taille fixe est allouée statiquement dans la mémoire pour la machine virtuelle, qui n'utilisera que cette zone. De la même manière, la consommation du processeur par la machine virtuelle est complètement maîtrisable, en régulant le nombre de cycles exécutés par seconde. De plus, les nombreuses erreurs qui peuvent arriver en exécutant du code utilisateur (par exemple, la pile qui peut être surchargée ou sous chargée, ou encore la mémoire qui peut être dépassée) sont détectées par la machine virtuelle qui marque le programme correspondant comme "crashé" avec un code d'erreur.

La machine virtuelle ne contient qu'une cinquantaine d'instructions. Un exemple de code assembleur pour cette machine, qui affiche les nombres de 1 à 10 est :

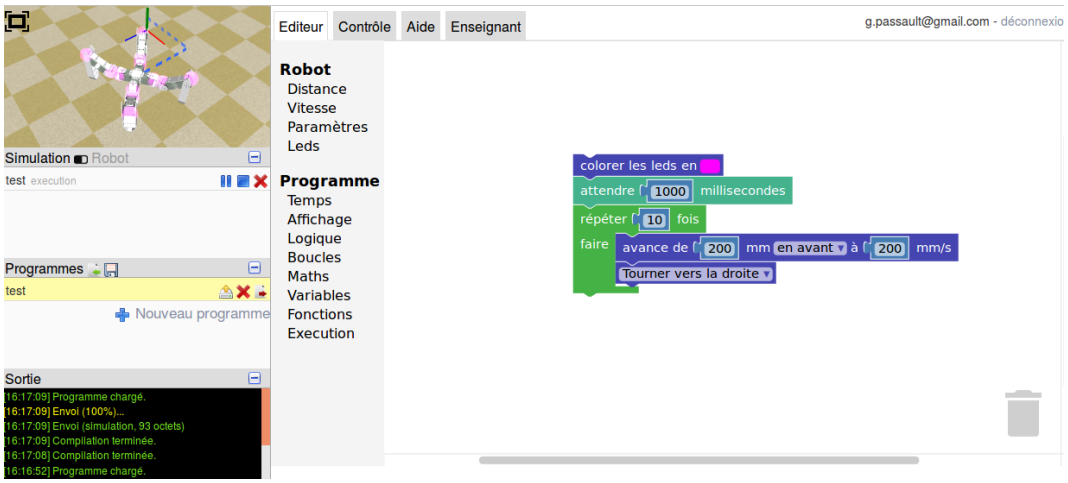


FIGURE 2.21 – L'interface de programmation permet de basculer entre la simulation et le robot réel. La simulation 3D est visible dans le coin supérieur gauche de l'écran et permet de tester les programmes créés.

```
main:
   push 0              ; n=0
loop:
   load 0
   call printv         ; printv(n)
   call println        ; println()
   push 1
   add                 ; n = n+1
   load 0
   push 10
   testle
   jmpc loop           ; if (n <= 10) goto loop
   stops
```

Les commentaires représentent le pseudo-code source équivalent. Comme on peut le constater, les opérations ressemblent à celles que l'on retrouve habituellement dans du code machine. Ici, *printv* et *println* sont des fonctions natives de la machine virtuelle. Le code ci-dessus produira par exemple une fois assemblé un objet binaire de 60 octets.

Afin de pouvoir décrire des programmes parallèles, la machine virtuelle supporte également la notion de tâches concurrentes :

```
.variable n

main:
    pushv n
    push 1
    add
    popv n          ; n = n+1
    push 1000
    call sleep      ; sleep(1000)
    jmp main        ; goto main

.thread
task:
    pushv n
    call printv     ; printv(n)
    call println    ; println()
    push 1000
    call sleep      ; sleep(1000)
    jmp task        ; goto task
```

Dans cet exemple, deux tâches parallèles s'exécutent, la tâche principale *main* qui incrémente la variable n et la tâche *task* qui l'affiche régulièrement. Chaque tâche dispose de son propre contexte, la machine virtuelle maintient par exemple une pile par tâche.

2.3.4 Programmation visuelle

Notre machine virtuelle a été conçue indépendemment de Metabot. Afin de pouvoir l'utiliser pour programmer un robot, il faut écrire les fonctions natives qui permettent à la machine virtuelle d'interagir avec le robot, comme par exemple accéder à la valeur d'un capteur, ou envoyer des ordres au contrôleur du robot.

Compilation

Afin de produire un programme utilisable à bord du robot, il faut tout d'abord compiler ses blocs en bytecode pour la machine virtuelle. Ce processus est très similaire à celui d'un compilateur qui traduit l'arbre syntaxique en code machine. Un exemple est présenté sur la figure 2.22.

Si des blocs sont disjoints, ils seront considérés comme des threads différents, résultant alors en plusieurs comportements parallèles (figure 2.23). Ceci est rendu possible par le support des threads dans la machine virtuelle présentée précédemment.

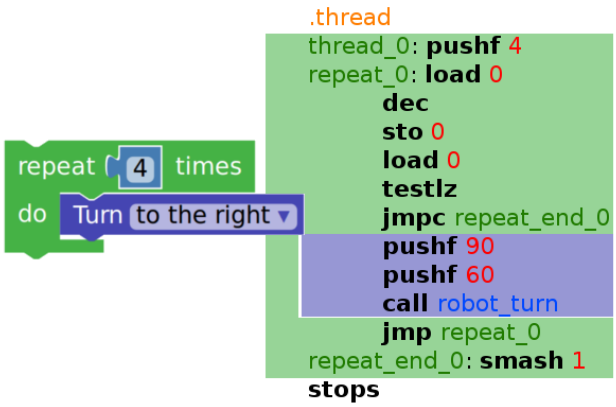


FIGURE 2.22 – Un exemple très simple d'enchaînement de blocs et sa traduction en code assembleur pour notre machine virtuelle. Les couleurs traduisent quelle partie du code correspond à quel bloc.

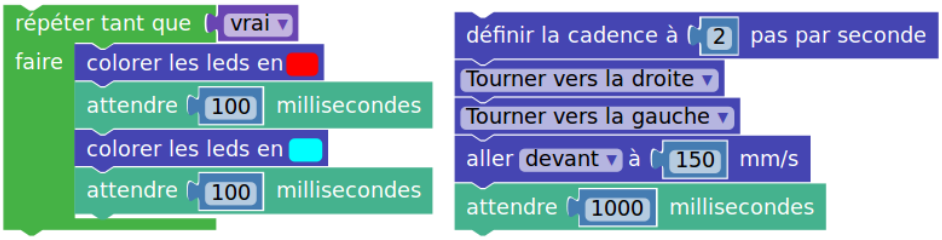


FIGURE 2.23 – Les séquences de blocs sont disjointes, les deux tâches s'éxecuteront en parallèle, et le robot clignotera tout en se déplaçant.

De cette manière, des événements peuvent être surveillés afin de déclencher du comportement. Le code présenté sur la figure 2.24 propose un exemple dans lequel le robot effectuera un parcours, et sera interrompu si son capteur frontal détecte un obstacle. L'interruption et la reprise de la tâche sont des notions assez naturelles, qui sont gérées ici par la mise en pause de la tâche (concrètement, le thread n'est plus exécuté par la machine virtuelle). Les événements sont, eux aussi, gérés par des threads qui scrutent leur condition. On peut traduire en pseudo-code :

```
when (condition) {
    action
}
```

Par :

```
while (true) {
    while (!condition)
    action
    while (condition)
}
```

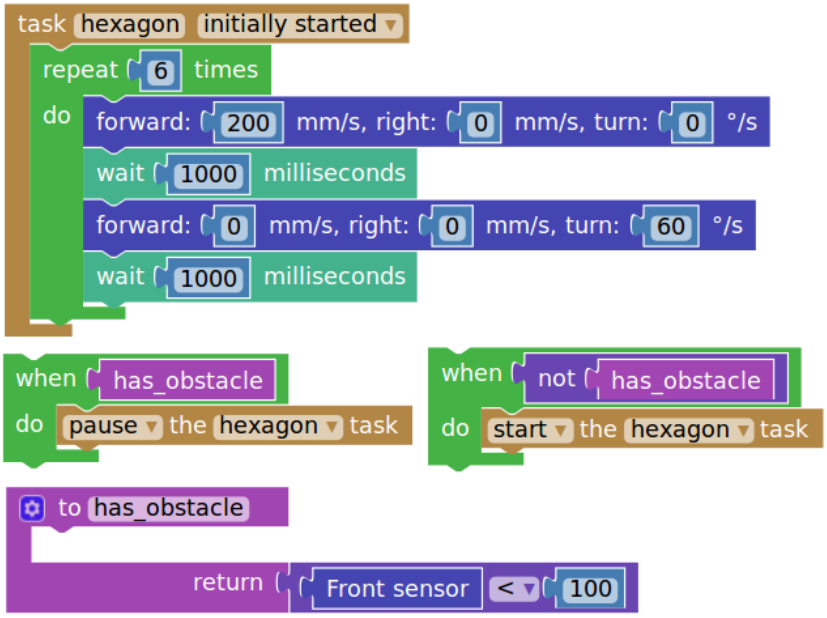


FIGURE 2.24 – Le robot va suivre un parcours et s'interrompre quand il rencontrera un obstacle. Lorsque l'obstacle disparaîtra, il reprendra son parcours.

Ici, la condition testée dans l'événement est elle-même du code utilisateur arbitraire, qui requiert une pile pour être interprétée. La présence de threads dans la machine virtuelle rend naturellement cela possible.

Un autre exemple de l'avantage de la présence des tâches et des processus est la capacité des programmes à se lancer et à se stopper les uns les autres. La figure 2.25 présente un exemple de programme qui provoquera un clignotement et ensuite déclenchera d'autres comportements. Ce dernier peut par exemple permettre de se synchroniser pour déclencher une musique avant le début d'une chorégraphie. Les tâches de cette chorégraphie peuvent par exemple être séparées en deux programmes : un qui gérera les LEDs et l'autre les mouvements. Là encore, ce système est rendu possible par la présence de mulitples programmes simultanés. Cette idée est également similaire à notre architecture durant la RoboCup, plusieurs comportements peuvent être chargés à bord du robot, et activés ou désactivés.

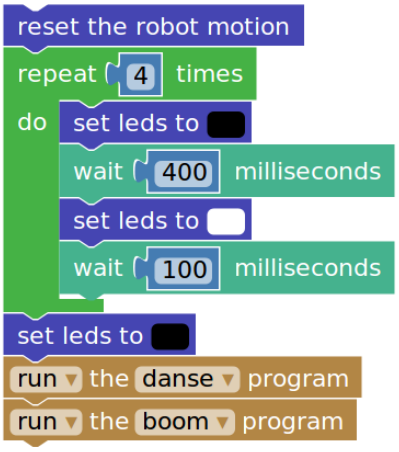


FIGURE 2.25 – Un exemple de programme qui fera clignoter les LEDs du robot quatre fois avant de lancer deux autres programmes.

Locomotion

Des fonctions natives permettent de piloter le contrôleur du robot, en lui envoyant des ordres en vitesse et en changeant ses paramètres (posture, fréquence ou encore levée des pattes).

Un comportement intuitif de la figure 2.24 est le fait que le robot arrêtera de bouger quand la tâche *hexagon* est en pause. Pour ce faire, lorsqu'un thread envoie des ordres au contrôleur du robot, il est marqué comme étant le thread qui pilote le mouvement, et lorsqu'il est stoppé ou mis en pause le robot arrête de bouger.

Il est également possible de piloter le robot pendant qu'il exécute des comportements. Par exemple, un programme peut faire clignoter les LEDs du robot et modifier les paramètres de sa posture pendant qu'un utilisateur le contrôle au joystick.

2.3.5 Utilisation de la plateforme

La plateforme Metabot a été utilisée dans plusieurs collèges et écoles primaires. Dans l'un d'entre eux, où nous sommes intervenus, nous avons pu voir la synergie ainsi créée entre deux matières : la technologie où le robot était dessiné, imprimé en 3D et assemblé par les élèves, et les mathématiques où ils avaient alors l'occasion de le programmer.

Afin de fournir un objectif aux élèves, nous avons également organisé une

compétition au cours d'un événement local (le Robot Maker's Day). Les élèves devaient créer un programme pour faire danser le robot, sur une musique de leur choix. Deux éditions de cet événement ont eu lieu (figure 2.26).

FIGURE 2.26 – Concours de danse organisé sur la plateforme Metabot au cours du Robot Maker's Day de 2015.

2.3.6 Communauté

Le développement de la plateforme a suscité l'interêt de la communauté, notamment des hobbyistes. Certains ont d'ailleur construit leur propre robot avant qu'il ne soit commercialisé sous forme de kit, en suivant les informations partagées en open-source (figure 2.27).

La plateforme a été également utilisée dans un cadre artistique, en pilotant le robot sur une plateforme piézoélectrique afin qu'il produise des sons (figure 2.28). Il a également été interfacé avec I-score, un séquenceur interactif qui permet de coordonner des systèmes hétérogènes dans une scénographie (Baltazar *et al.* [2014]). Le robot a été produit sur scène au festival d'Uzeste de 2016.

Les robots ont également été présentés et éprouvés lors de nombreuses démonstrations et événements, et utilisés dans des matchs de football lors de l'animation de la fan zone durant l'Euro 2016 à Bordeaux (figure 2.29), au cours de laquelle les visiteurs avaient la possibilité de piloter les robots à la manette dans des matchs de deux robots contre deux sur un terrain de football à taille réduite.

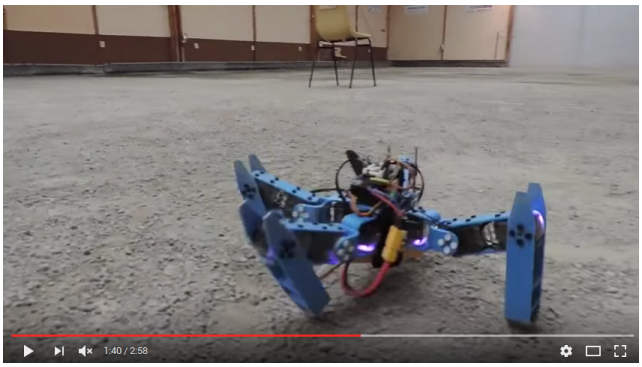


FIGURE 2.27 – Ce robot a été construit par un hobbyiste en utilisant le dépôt open-source de Metabot. Il a commandé lui-même les composants et imprimé lui-même les pièces, fabricant une réplique du robot dans sa première version.

FIGURE 2.28 – Le robot Metabot piloté sur une plateforme piézoélectrique pendant le festival de Jazz Uzeste Musical 2016.

2.4 Conclusion

Nous avons montré qu'il était possible de concevoir des robots à pattes à bas coût à travers l'exemple de la plateforme Metabot.

Nous avons également décrit un environnement de programmation permettant à un utilisateur non expert de programmer des comportements autonomes sur le robot, et argumenté en faveur de l'utilisation d'une machine virtuelle légère et simplifiée dans cette optique. Cette dernière propose entre autres un système de processus qui permet de multiplier les possibilités de programmes.

La plateforme Metabot a été non seulement éprouvée par notre équipe à travers de nombreuses heures de test mais aussi déployée et utilisée par des personnes tierces, en particulier des enfants, qui ont été capables de la prendre

FIGURE 2.29 – Matchs de football sur des terrains à taille réduite au cours de l'Euro 2016 (projet ayant reçu le soutien de la mairie de Bordeaux).

en main et de la programmer.

Des élèves de différents niveaux se sont familiarisés avec l'environnement de programmation et la plateforme, avec une motivation qui les a poussés à participer au concours que nous avons organisé.

Il n'existe pas encore d'étude qui met clairement en lumière la qualité pédagogique de l'utilisation des robots dans un enseignement. Ce serait d'ailleurs un travail important à réaliser pour poursuivre dans cette voie.

Cependant, nous avons fait l'expérience de l'enthousiasme, mais aussi du prétexte provoqué par l'utilisation de Metabot pour aborder des sujets pédagogiques pendant des enseignements. L'argument de la curiosité et de la motivation est omniprésent dans la proposition de l'utilisation des robots pour l'éducation (par exemple dans Riedo *et al.* [2013]).

Chapitre 3

Robots paramétriques (Metabot Studio)

Dans le chapitre précédent, nous avons présenté une plateforme à pattes dont la locomotion a prouvé son efficacité dans la pratique.

Cependant, la conception du robot repose sur des connaissances empiriques, certains choix ont été faits arbitrairement. Notamment, le contrôleur et ses paramètres "sortie d'usine", mais aussi les dimensions et la forme des pièces.

Se pose alors la question de la généralisation de la production des robots à pattes, ou comment étendre de la manière la plus automatisée la famille de Metabot ? Imaginons par exemple fournir un environnement logiciel à un utilisateur non expert qui lui permettrait de dessiner un robot, comment se présenterait un tel environnement pour permettre d'automatiser la manufacture du robot ? Comment pourrait-on généraliser le contrôleur expert de Metabot à une famille plus grande de robots pour proposer un contrôleur efficace automatiquement ?

Dans ce chapitre, nous présenterons tout d'abord *Metabot Studio*, un environnement que nous avons conçu dans le but de pouvoir modéliser, mais aussi manipuler en simulation des robots paramétriques. Le but de ce dernier est de pouvoir facilement tester des algorithmes sur des robots divers et variés dont le modèle complet (géométrique et dynamique) est connu, mais également dont la morphologie peut être modifiée. Ensuite, nous décrirons des expériences d'optimisation que nous avons proposées dont le but est d'essayer de proposer un contrôleur générique capable de piloter une famille de robots à pattes. Enfin, nous présenterons le résultat de ces expériences et essaierons d'en tirer des conclusions sur le contrôle des robots à pattes.

3.1 Modélisation

3.1.1 Rappels théoriques

Un robot peut être représenté par des pièces connectées entre elles par des articulations. Nous considérerons ici la famille des robots ouverts (pas d'architectures parallèles) utilisant des actuateurs rotoïdes.

Modèle géométrique

La chaîne cinématique d'une patte dans le repère du tronc du robot peut se représenter sous la forme :

$$P = M_1 R_z(\theta_1) M_2 R_z(\theta_2) \dots M_{n-1} R_z(\theta_{n-1}) M_n$$

Où M_i est une matrice de transformation homogène et $R_z(\theta_i)$ une rotation (nous choisirons par convention de faire tourner nos articulations autour de l'axe z).

Classiquement, les matrices de transformation représentent ici un changement de repère, qui correspond à une translation et une rotation. D'un point de vue géométrique, une pièce dans un robot qui relie deux articluations peut donc se résumer en 6 valeurs $(\theta, \gamma, \phi, x, y, z)$.

Modèle dynamique

Pour obtenir un modèle dynamique, les informations inertielles sont nécessaires, elles correspondent à la masse d'une pièce m, son centre de masse (x, y, z) ainsi que son tenseur d'inertie, qui définit la répartition de la masse, défini par :

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{xz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

Où :

$$\begin{aligned} I_{xx} &= \textstyle\sum_i (y_i^2 + z_i^2) * m_i \\ I_{yy} &= \textstyle\sum_i (x_i^2 + z_i^2) * m_i \\ I_{zz} &= \textstyle\sum_i (x_i^2 + y_i^2) * m_i \\ I_{xy} &= I_{yx} = -\textstyle\sum_i (x_i * y_i) * m_i \\ I_{yz} &= I_{zy} = -\textstyle\sum_i (y_i * z_i) * m_i \\ I_{xz} &= I_{zx} = -\textstyle\sum_i (x_i * z_i) * m_i \end{aligned}$$

On remarquera que la matrice d'inertie étant symétrique, elle est diagonalisable par une matrice orthogonale. Cela signifie qu'il existe une rotation de

la pièce après laquelle les coefficients hors diagonale de la matrice sont nuls. Cette rotation définit les axes principaux d'inertie.

Les modèles utilisés dans Metabot Studio s'appuieront sur ces éléments classiques.

Implémentation

La géométrie et la dynamique permettent de résumer le comportement physique qu'aura une pièce. Cependant, il existe évidemment une infinité de pièces ayant le même modèle géométrique et dynamique. La conception d'une pièce repose en général sur des contraintes plus pragmatiques, au sens pratique du terme (fixations, facilité d'impression 3D, esthétique...).

3.1.2 Construction des pièces

Concepts

Nous appelerons **modèle** la représentation d'un objet 3D qui existe déjà, telle qu'un servomoteur, un palonnier ou une carte électronique, et qui peut être intégré dans le robot, et **pièce** la représentation d'un objet 3D qui doit être construit afin d'assembler le robot.

Les pièces représentent donc la partie modulaire du robot, qui vont déterminer comment les degrés de libertés sont agencés. Une pièce a une géométrie, mais aussi éventuellement des paramètres.

Afin de construire des modules que l'on peut assembler virtuellement, nous proposons d'introduire la notion de **composant**. Un composant est une partie du robot qui contient des modèles et des pièces. Ils expriment en réalité la connaissance de comment assembler les pièces. Ils peuvent être alors assemblés entre eux, pour cela, ils présentent des **points d'ancrages**. Afin d'assembler deux composants, il faut apairer un point d'ancrage sur chacun d'eux, qui doivent être compatibles. Un exemple de composant est présenté figure 3.1.

Un composant pouvant comporter plusieurs pièces, il peut donc lui aussi avoir des paramètres. Plusieurs instances du même composant pourront donc être différentes. La figure 3.2 présente trois instances du même composant avec des paramètres différents.

Partant de ce principe, un robot est donc un ensemble d'instances de composants connectées entre elles par des points d'ancrages. Ne considérant pas les robots parallèles, l'architecture du robot peut être représentée comme un arbre, dont la racine est également son tronc. La figure 3.3 présente un exemple

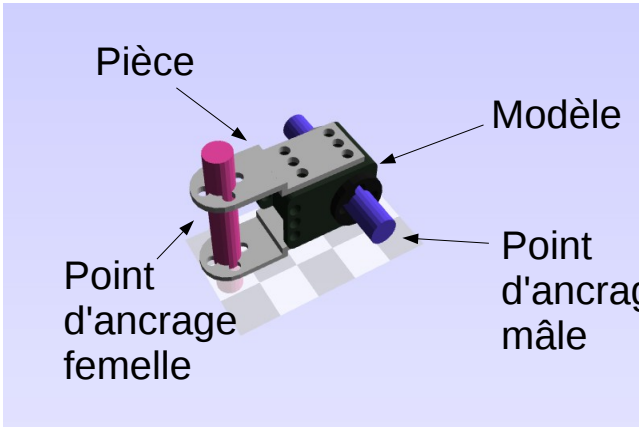


FIGURE 3.1 – Exemple de composant comportant des pièces, le modèle d'un moteur et deux points d'ancrages, un mâle et un femelle.

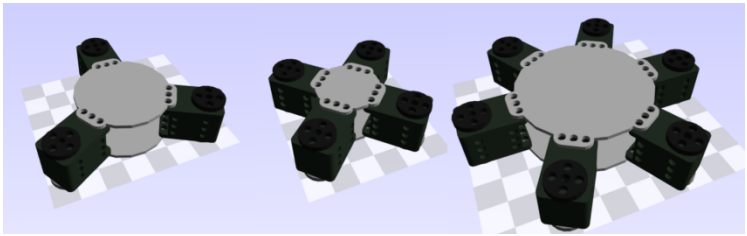


FIGURE 3.2 – Trois instances du même composant avec des paramètres différents.

de modèle de robot quadrupède, comportant 12 degrés de libertés et 13 composants.

Représentation

Il existe plusieurs manières de représenter un objet 3D, l'une de ces manières est la géométrie constructive, qui consiste à décrire les formes géométriques à l'aide de primitives et d'opérations booléennes (figure 3.4).

Il est donc possible de décrire un arbre de géométrie constructive, dont les noeuds sont des opérations :

— Changement de base (translation et rotation)
— Différence
— Union
— Intersection

Et les feuilles des formes pures (cube, sphere, cylindres...).

Le logiciel OpenSCAD (OpenSCAD [2010]) permet de manipuler des objets modélisés en géométrie constructive. Sa particularité est qu'il n'offre aucune

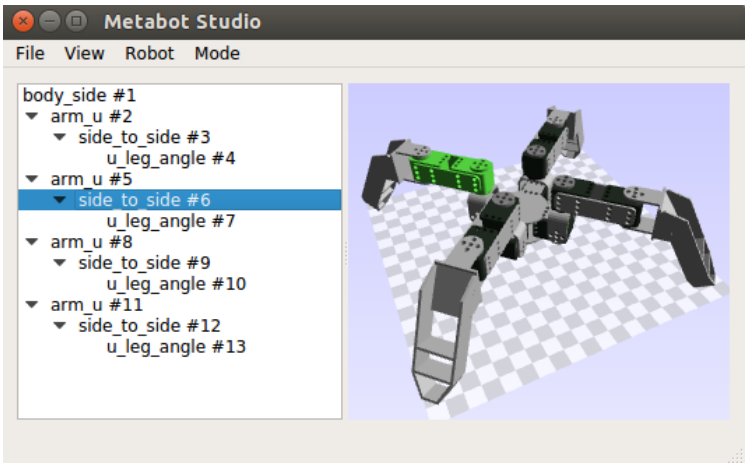


FIGURE 3.3 – Un modèle de robot quadrupède, la structure arborescente de ses composants est présentée à gauche.

interactivité autre que l'édition textuelle dans un langage (le SCAD) qui, à l'image d'un langage de programmation, comporte des boucles, des variables et des fonctions (figure 3.5). Ce code est ensuite compilé dans un autre langage, le CSG, qui est en réalité un sous ensemble du SCAD et qui représente en fait un arbre de géométrie constructive.

Par exemple, le code SCAD suivant :

```
// exemple.scad
module something(r) {
    translate([0,0,5])
        cube([r,r*2,r]);
}

translate([5,0,0])
something(3);
```

Contient un module avec un argument qui est instancié. Ce code sera compilé en :

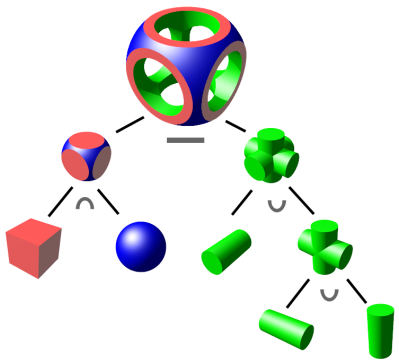


FIGURE 3.4 – L'objet présenté en haut de l'image est constitué d'une série d'opérations booléennes qui partent de formes élémentaires, c'est la géométrie constructive.

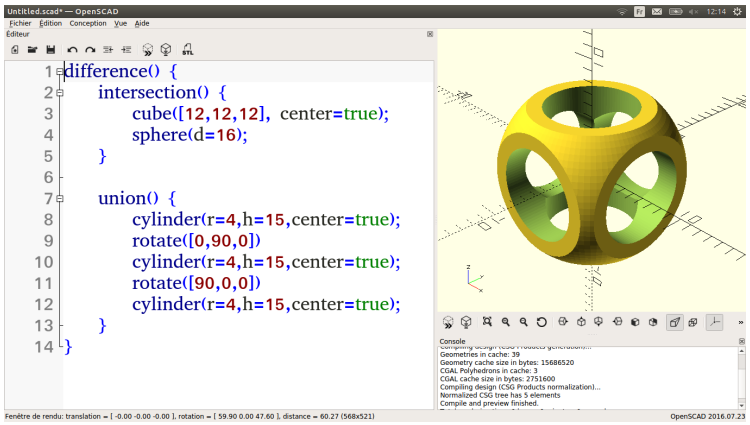


FIGURE 3.5 – Exemple de la figure 3.5 reproduite dans OpenSCAD.

```
// exemple.csg
multmatrix([[1, 0, 0, 5], [0, 1, 0, 0],
            [0, 0, 1, 0], [0, 0, 0, 1]]) {
  group() {
    multmatrix([[1, 0, 0, 0], [0, 1, 0, 0],
                [0, 0, 1, 5], [0, 0, 0, 1]]) {
      cube(size = [3, 6, 3], center = false);
    }
  }
}
```

Qui ne contient que des opérations élémentaires. En l'occurence les matrices de transformations et le cube. Les calculs sont "aplatis", il n'y a plus de

variable, la géométrie de la forme pure est connue.

Le CSG peut ensuite être converti en facettes, créant ainsi un objet 3D. Ce processus se nomme la polygonisation.

Afin de pouvoir manipuler les concepts présentés précédemment, nous avons enrichi le langage SCAD pour y ajouter des méta-données. Ces méta-données permettent de "taguer" l'arbre de construction géométrique afin d'être capable de délimiter les pièces, modèles et points d'ancrages. Elles permettent également d'exporter les paramètres des composants.

Pour bien comprendre ce système, un exemple est présenté sur la figure 3.6.

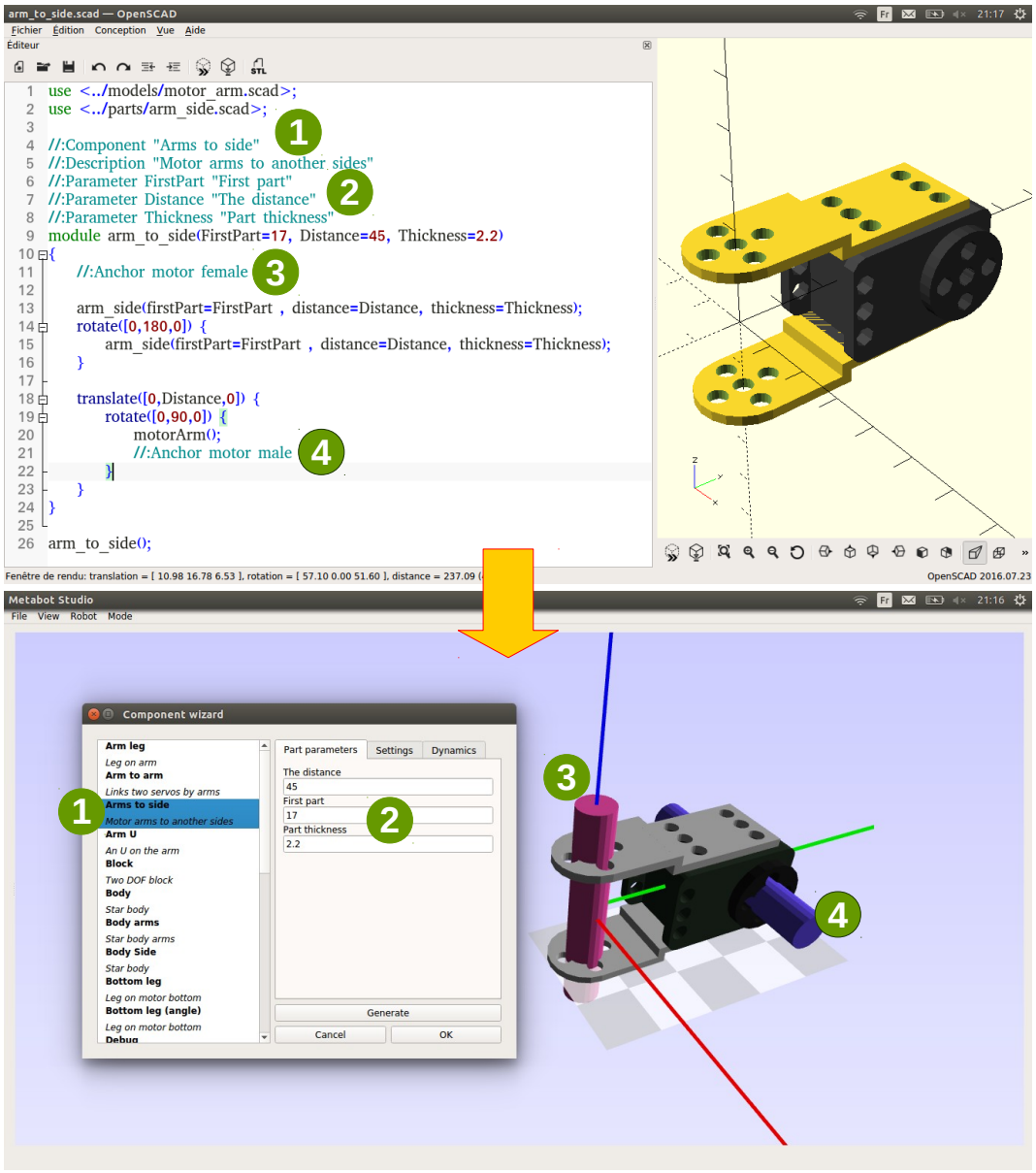


FIGURE 3.6 – Cet exemple montre la correspondance entre le code et les métadonnées que nous y avons ajouté et le résultat dans Metabot Studio. En (1), le nom et la description du composant qui permettent de le proposer dans la liste, en (2), la description des paramètres que l'utilisateur peut modifier, en (3) et en (4), des points d'ancrage qui apparaîtront dans l'éditeur pour connecter des composants.

3.1.3 Construction du robot

Une fois les composants conçus, il est possible de proposer un outil permettant d'éditer visuellement des assemblages de robots.

Bien que la création des pièces soit une tâche experte, l'édition du robot ne nécéssite pas beaucoup de connaissances, et pourrait être fait par un utilisateur non-expert. C'est d'ailleurs un interêt de la méthode du catalogue de composants.

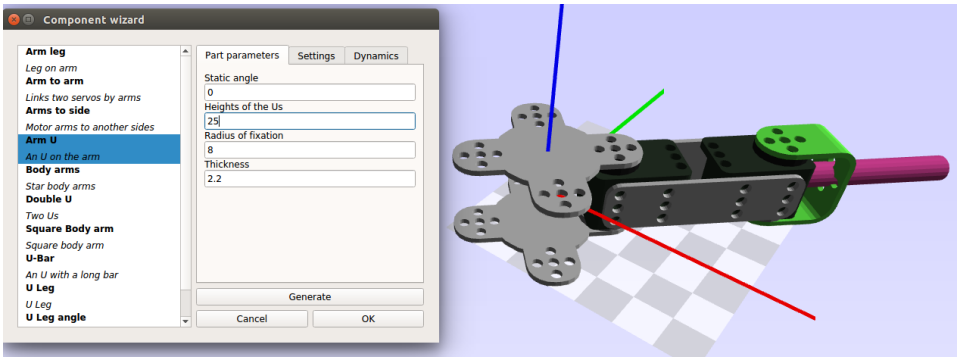


FIGURE 3.7 – L'éditeur graphique de Metabot Studio.

Cet éditeur présenté sur la figure 3.7 permet de créer des instances de composants, et d'éditer leurs paramètres. Les points d'ancrages peuvent alors être sélectionnés afin d'insérer un autre composant en proposant tous ceux qui disposent d'un point d'ancrage compatible.

Enfin, un robot peut lui même avoir des paramètres, qui sont ses paramètres morphologiques. Ces derniers ont en fait un impact sur les paramètres des composants, qui, comme présenté précédemment, ont eux même un impact sur ceux des pièces [1].

Manufacture

Afin de produire le robot, il est possible d'exporter chacune de ses pièces de chacun de ses composants. Ces dernières peuvent alors par exemple être imprimées en 3D.

Dans cette optique, nous avons également développé Plater [2], un outil permettant d'agencer automatiquement les pièces générées sur un ou plusieurs plateaux d'imprimante 3D, en essayant d'optimiser le nombre de pièces pour

1. Le processus d'édition est présenté sur la vidéo https://youtu.be/smHctwi05Ic
2. https://3dprintingindustry.com/news/3d-printed-parts-served-silver-platter-35241/

chaque plateau(figure 3.8). Notons que l'impression de pièces peut prendre des heures.

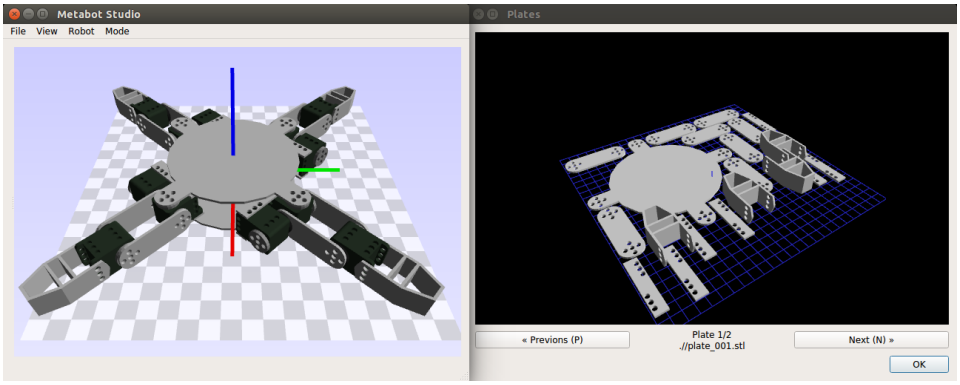


FIGURE 3.8 – Les pièces du robot de gauche sont automatiquement exportées et disposées sur un plateau prêt à être imprimé.

Ce logiciel convertit tout d'abord les pièces en une représentation aplatie discrète (en "pixels"), avec une certaine résolution (typiquement $1mm^2$). Ces représentations intermédiaires sont des sortes de vues de haut des pièces. Le problème à résoudre, bien que simplifié, est alors celui du *bin packing* en 2D. Un ensemble d'heuristiques permettent alors d'essayer de nombreuses combinaisons de placement pour positionner les pièces en produisant le moins de plateaux possible. Une stratégie standard est de placer les pièces les plus encombrantes en premier.

Plater a été partagé en open-source [3], il est disponible en ligne de commande ou avec une interface graphique permettant de spécifier la liste des pièces et leur orientation, mais aussi les spécifications du plateau de l'imprimante ainsi que les paramètres de l'optimiseur (figure 3.9).

Il est également possible de produire la liste du matériel requis pour fabriquer le robot, il faut pour cela énumérer les modèles, qui peuvent éventuellement être annotés de méta-données permettant de connaître par exemple son prix et son nom complet (figure 3.10).

3.1.4 Calcul dynamique

Notre objectif étant de simuler les robots avec un moteur physique, un prérequis est de calculer son modèle dynamique.

3. https://github.com/Rhoban/Plater

FIGURE 3.9 – Notre logiciel Plater, open-source, avec une interface graphique qui permet de générer automatiquement des plateaux contenant un ensemble de pièces.

```
3   include <motor_values.scad>;
4   use <../util/rounded.scad>;
5
6   //:Model "Motor" [0.15,0.2,0.15]
7   //:Mass 16.7
8   module motor(thickness=2.2) {
9       //:BOM "XL-320 Servo" 21.9
            http://www.robotis-shop-en.com/?act=shop_en.goods_view&GS=1611
10      translate([0,-MotorArmOffset,-MotorDepth/2]) {
11          color([0.25,0.25,0.25]) {
12              difference() {
13                  rounded(MotorWidth,MotorHeight,MotorDepth,3,center=true);
14
```

Dialog

Bill of materials

Name	Quantity	Price	Total	More
Ollo pulley	16	$ 0.2	$ 3.2	Buy
Ollo rivet	192	$ 0.05	$ 9.6	Buy
XL-320 Servo	12	$ 21.9	$ 262.8	Buy
			$ 275.6	

OK

FIGURE 3.10 – Extrait de pièce contenant des métadonnées concernant la liste des pièces à acheter (bill of materials, ou BOM). À droite, l'écran présentant les éléments collectées depuis toutes les métadonnées d'un robot, générant ainsi la liste d'achat globale et l'estimation du coût des pièces.

Cinématique

Connaissant les composants et les changements de bases associés à leurs points d'ancrage, ainsi que la position du bout des pattes, la chaîne cinématique directe de chaque patte est donc connue.

Calcul de la dynamique des pièces

Afin de calculer le modèle dynamique du robot, il faut être capable d'évaluer les masses et les inerties, comme présenté précédemment.

Une manière de faire est d'approximer les objets 3D par des *voxels*, c'est à dire des cubes de petites tailles. Pour ce faire, chaque pièce est bornée par une boîte englobante (*bounding box*), qui est subdivisée en cubes. Un test dé-

termine pour chacun de ces cubes s'il est ou non dans la forme, en utilisant la méthode des pairs/impairs (expliquée dans Hormann et Agathos [2001], figure 3.11).

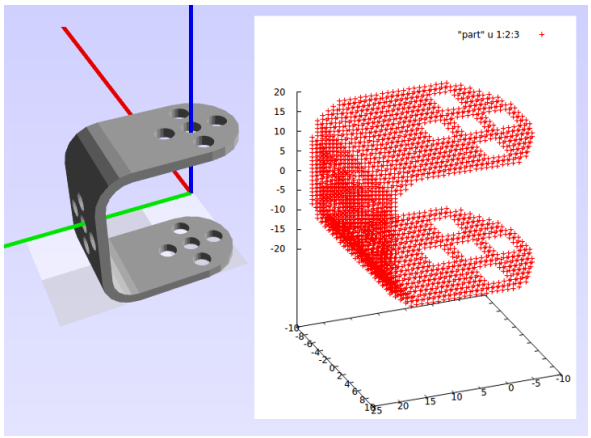


FIGURE 3.11 – La pièce est approximée par des petits cubes (ou *voxels*) qui la discrétise.

Ainsi, le volume de la pièce peut être approximé en sommant le volume de chaque cube. Pour les pièces à morphologie variable, une densité du matériau est requise pour convertir ce volume en masse (par exemple, la densité du PLA, plastique fréquemment utilisé en impression 3D est d'environ 1.25 kg/l). La masse des modèles est quant à elle connue, une approximation est de la répartir uniformément entre les différents cubes. La matrice d'inertie peut alors être calculée numériquement.

Le centre de masse est obtenu en calculant la moyenne des cubes, et le référentiel inertiel (axes principaux d'inertie) en diagonalisant la matrice d'inertie.

Modèle de collision

Pour réaliser des calculs physiques, les collisions représentent en général une difficulté, car le calcul des collisions sur le véritable modèle du robot serait en réalité à la fois très coûteux en ressources et très instable, à cause du nombre important d'arêtes et de facettes. C'est pour cela qu'il est préférable d'utiliser un autre modèle du robot simplifié, généralement constitué uniquement de formes pures (comme des boîtes, des sphères et des cylindres, figure 3.12).

Chaque pièce et modèle conçu doit alors également proposer une alternative en formes pures. Ceci est rendu plus simple par la géométrie constructive,

car cette dernière s'appuie déjà sur des formes pures. Un modèle de collision peut alors être comme un arbre ne comprenant que des unions, des matrices de transformations et des formes pures.

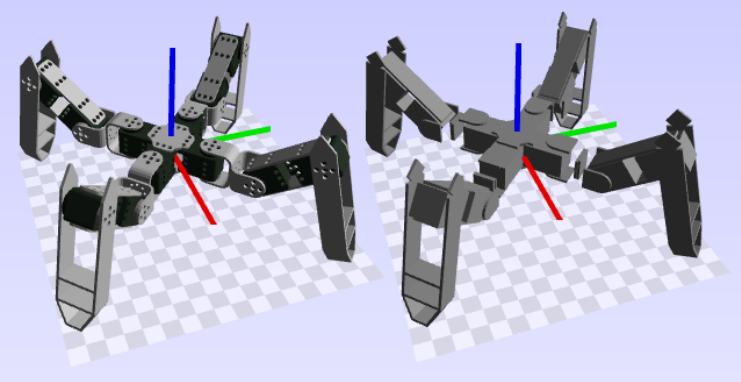


FIGURE 3.12 – Un modèle de robot (à gauche) et son approximation en formes pures (à droite) destinée à la simplification des calculs de collision.

Pour la plupart des pièces sur lesquelles nous avons travaillé, l'approximation en forme pure a simplement consisté en la simplification conditionelle de son arbre géométrique.

Certaines pièces sont proches les unes des autres, pouvant entrainer des autocollisions liées à des petites erreurs numériques lors de la simulation. Typiquement, le palonnier du moteur qui est collé à ce dernier risque de le toucher en permanence si une petite imprécision numérique survient. Pour cela, une première parade est d'introduire manuellement de l'espace (figure 3.13). Cependant, les formes étant pures, il est également facile de les "rétracter" légèrement afin de créer une petite marge numérique.

Il est alors possible de collecter les formes pures et leur emplacement dans la pièce dans l'arbre de construction géométrique afin d'obtenir le modèle de collision du robot.

3.2 Simulation

3.2.1 Moteur physique

Le comportement des robots peut alors être simulé en utilisant un moteur physique. Nous avons utilisé la bibliothèque Bullet (Coumans *et al.* [2013],

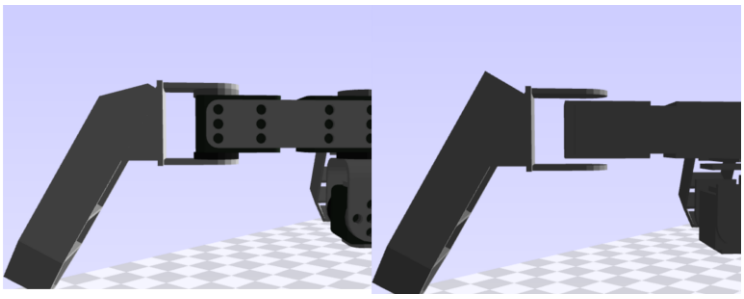


FIGURE 3.13 – Le modèle de collisions (à droite) prévoit une petite marge par rapport au modèle des vraies pièces (à gauche) pour éviter les problèmes liés aux imprécisions numériques.

Coumans [2003]). Notre environnement de conception et de simulation, spécifiquement conçu pour la simulation des robots à pattes, peut par exemple être comparé à un environnement standard de simulation robotique tel que V-REP (Rohmer *et al.* [2013]) ou Gazebo (Koenig et Howard [2004]). Ces derniers n'étaient pas adaptés au fait de charger et de décharger intensivement des modèles de robots, ce que nous allons faire pour l'optimisation.

De plus, le workflow de Metabot Studio est différent (figure 3.14). Dans V-REP, la physique et l'affichage est incorporée dans un seul programme, qu'il est possible de piloter via un serveur ou une API. Gazebo est découplé en un serveur, qui effectue les calculs et un client qui permet de visualiser le monde physique et d'interagir avec lui. Dans ce cas, le code utilisateur communique avec le serveur de la même manière que le client.

Metabot Studio est découpé en deux modes, le premier permet de modéliser des robots, et donc d'éditer interactivement des modèles et de changer des paramètres. Les robots ainsi créés peuvent être chargés à l'aide d'une API et manipulés, la physique faisant ici partie du code utilisateur. Le deuxième mode de Metabot Studio permet de visualiser les robots manipulés par le code utilisateur. Les rôles sont inversés, car la simulation a entièrement lieu dans le code utilisateur, permettant d'accéder aux informations du moteur physique lui-même directement si besoin.

Par exemple, ce modèle permet d'accéder à la position du robot, à son centre de masse et aux forces de réactions dues aux collisions avec le sol à chaque pas de temps du simulateur (typiquement 1000 Hz). Dans ce mode, Metabot Studio ne joue plus que le rôle d'un client graphique.

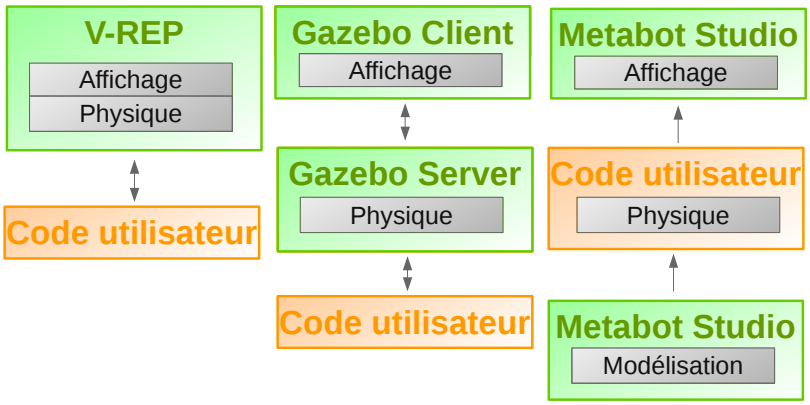


FIGURE 3.14 – Différences entre le workflow dans *V-REP*, *Gazebo* et *Metabot Studio*.

3.2.2 Modélisation des moteurs

Dans le moteur physique, les moteurs sont des degrés de liberté rotatifs (*hinge*). Des couples sont appliqués directement sur les pièces qu'ils connectent selon une simulation d'asservissement.

L'asservissement des moteurs a été modélisé dans le moteur physique par deux contrôleurs proportionnels. Le premier détermine une vitesse cible en fonction de l'erreur en position, et le deuxième détermine un couple cible en fonction de l'erreur en vitesse. C'est une architecture de contrôleur classique pour piloter des moteurs en position.

De plus, les moteurs ont une limite de couple et de vitesse. À noter que plus la vitesse du moteur est grande, moins il peut fournir de couple, suivant une loi linéaire telle que présentée en figure 3.15.

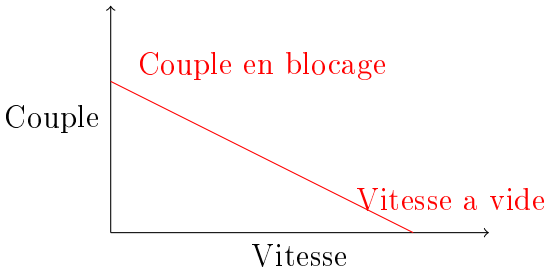


FIGURE 3.15 – Le couple vs la vitesse maximum d'un moteur DC.

Ce modèle ne correspond pas à la réalité des servomoteurs à bas coût, dans laquelle l'asservissement se fait bien sur la position, mais en utilisant un correcteur proportionnel sur le rapport cyclique du pont en H, et donc sur le voltage appliqué au moteur.

De plus, les frottements du moteur (statiques et fluides) ainsi que l'inertie de la boîte à engrenages ne sont pas modélisés. Cette dernière est due au fait que certaines roues dans l'étage de réduction, bien qu'ayant une masse légère, ont une vitesse de rotation importante.

De plus, bien que le pas de temps du simulateur physique soit en général inférieur à la milliseconde, la mise à jour des ordres moteur se fait bien plus rarement, à cause des limitations mécatroniques (vitesse du bus, du microcontrôleur etc.). Nous simulons également ce phénomène en imposant une limite dans la mise à jour des ordres (de 50 Hz).

3.2.3 Modélisation du jeu

Dans la logique de modéliser des robots à bas coûts, qui emploient des moteurs de moins bonne qualité, et des fixations mécaniques moins précises, nous avons ajouté dans le modèle de simulation une notion de jeu.

D'ailleurs, le robot Metabot présente lui-même du jeu sur ses degrés de liberté, qui sont principalement liés en l'occurence aux engrenages plastiques, à l'utilisation des rivets plastiques et des pièces réalisées sur des imprimantes 3D peu précises.

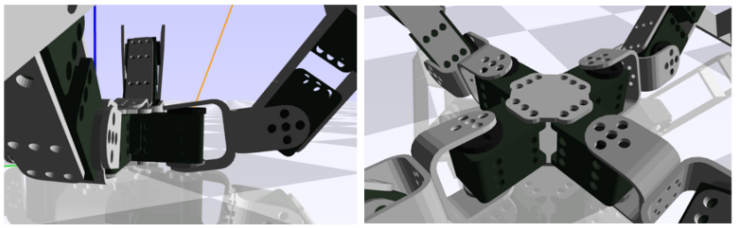


FIGURE 3.16 – Rendu de simulations avec des paramètres de jeu exagérés en latéral (à gauche) et en torsion (à droite).

Trois formes de jeux sont pris en compte :

— Le jeu de l'axe du moteur lui même
— Le jeu latéral, selon une orientation ortogonale à l'axe (figure 3.16)
— Le jeu de torsion autour de l'axe du moteur (figure 3.16)

Ces derniers peuvent être modélisés à l'aide d'une articulation passive ajoutée dans le modèle du robot, qui est un cône avec torsion présenté en figure 3.17. Ce dernier offre trois degrés de libertés, qui correspondent aux jeux d'axe, latéral et de torsion.

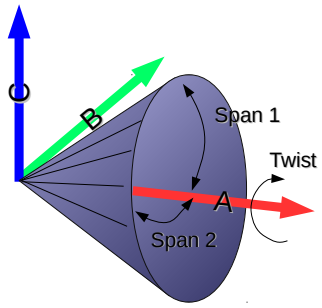


FIGURE 3.17 – Degré de liberté conique passif avec torsion.

Nous supposons que cette passivité ajoute du réalisme dans le comportement du robot simulé et imposera au contrôleur d'être plus robuste aux imprécisions mécaniques.

Pour les expériences, ce jeu a été mesuré manuellement sur une articulation utilisant le même moteur que celui utilisé par les robots paramétriques.

3.2.4 Limites

Bien que la simulation physique produise des résultats ressemblant à la réalité, il est important de mentionner que cette méthode possède de nombreuses limites :

- Le pas de temps de simulation Δt, est un compromis arbitraire entre la qualité de la simulation et le temps de calcul
- La gestion des collisions est un des éléments les plus complexes à simuler, omniprésent dans la locomotion des robots à pattes
- La friction du robot avec le sol a un impact sur la capacité des pattes à glisser, mais elle pourrait être déterminée expérimentalement pour un support donné
- Le robot est approximé par des corps solides. Or, dans la réalité, ça n'est pas forcément le cas (certaines pièces étant un peu élastique)
- Le solveur de contraintes du moteur physique a lui aussi des méta-paramètres qui peuvent avoir une influence sur la qualité de la simulation. En effet, la méthode de résolution des contraintes (collisions, frictions, joints etc.) est itérative (Catto [2005]), le principal paramètre est le nombre d'itérations de cet algorithme.

— Comme expliqué précédemment, le modèle des moteurs est incomplet

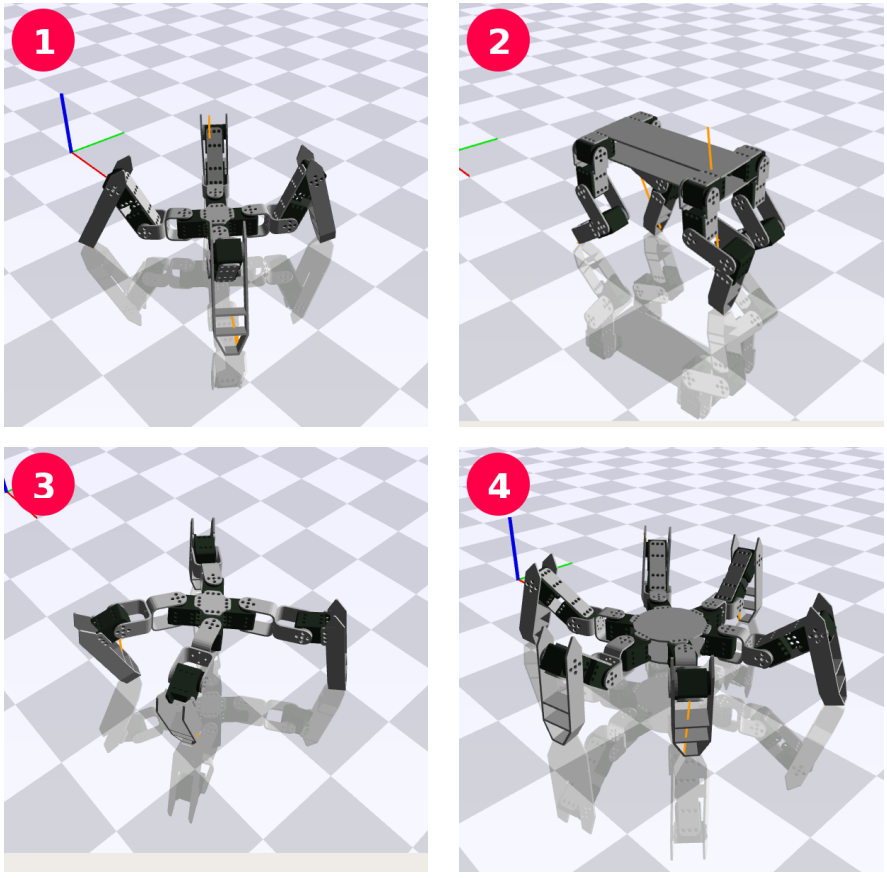
3.3 Expériences

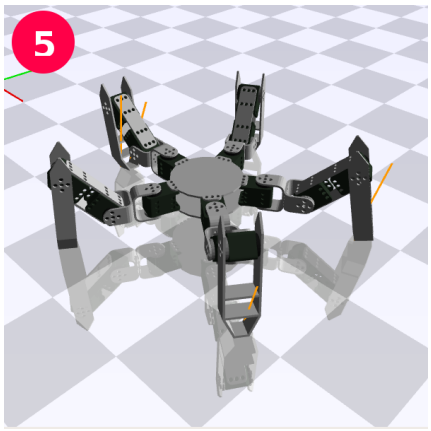
Nous allons présenter ici des expériences, qui ont été réalisées en utilisant l'environnement Metabot Studio détaillé précédemment. Le but de ces expériences est de produire des résultats empiriques sur la locomotion à pattes, et d'améliorer les connaissances sur le réglage des paramètres du contrôleur.

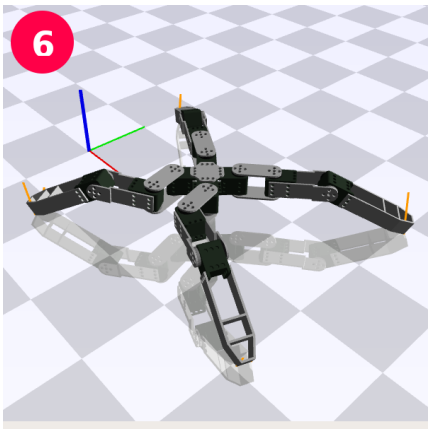
3.3.1 Corpus

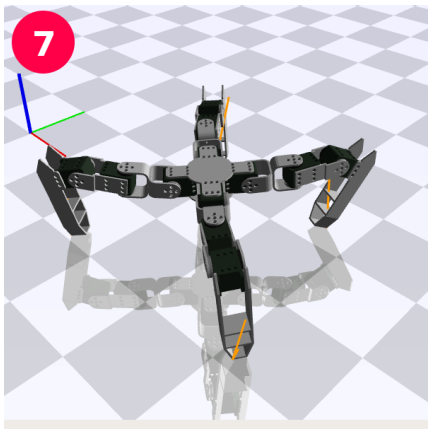
Pour mener nos expériences, nous avons créé un corpus de robots, se basant sur des pièces du même genre que celles de Metabot, et sur le même servomoteur bas coût (torque en blocage : 0.4 Nm, vitesse à vide : 4π rad/s).

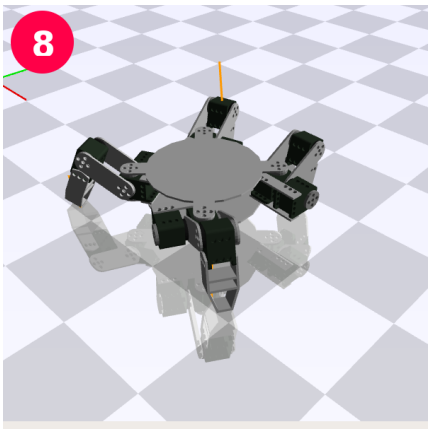
Les robots du corpus sont différents par leur nombre de pattes, l'agencement de leur degrés de libertés mais aussi par leur forme :

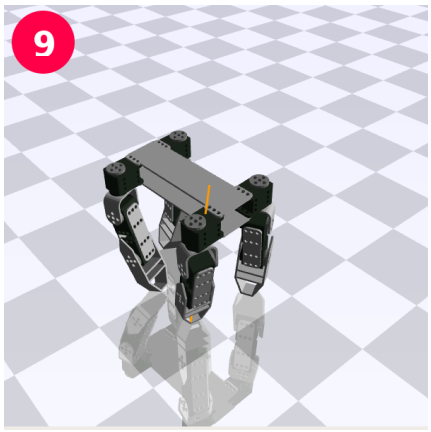


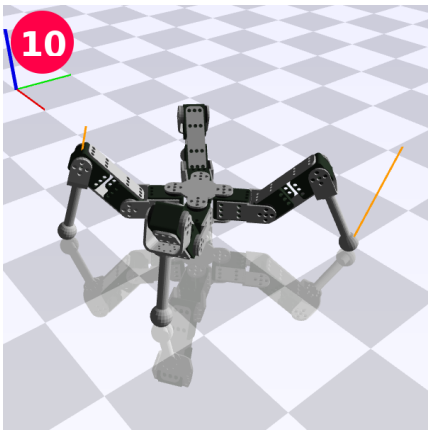

5

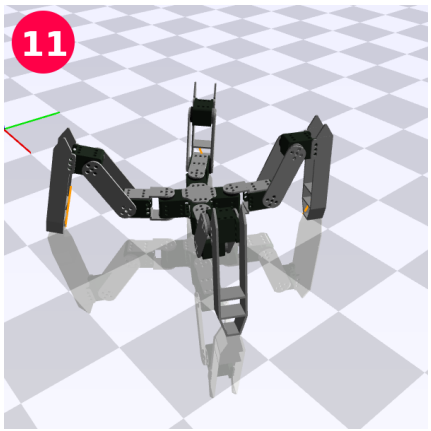

6

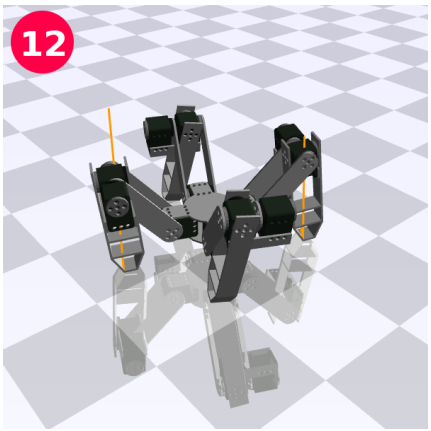

7

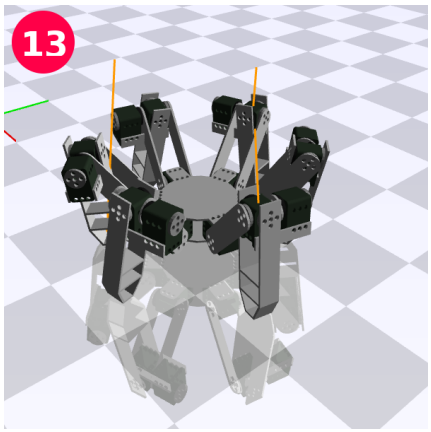

8

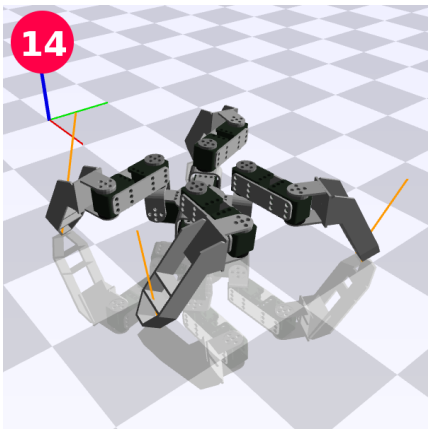

9


10

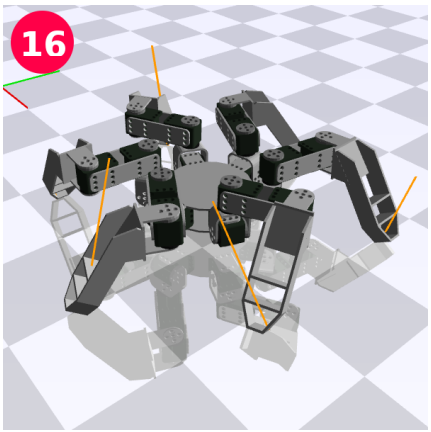

11

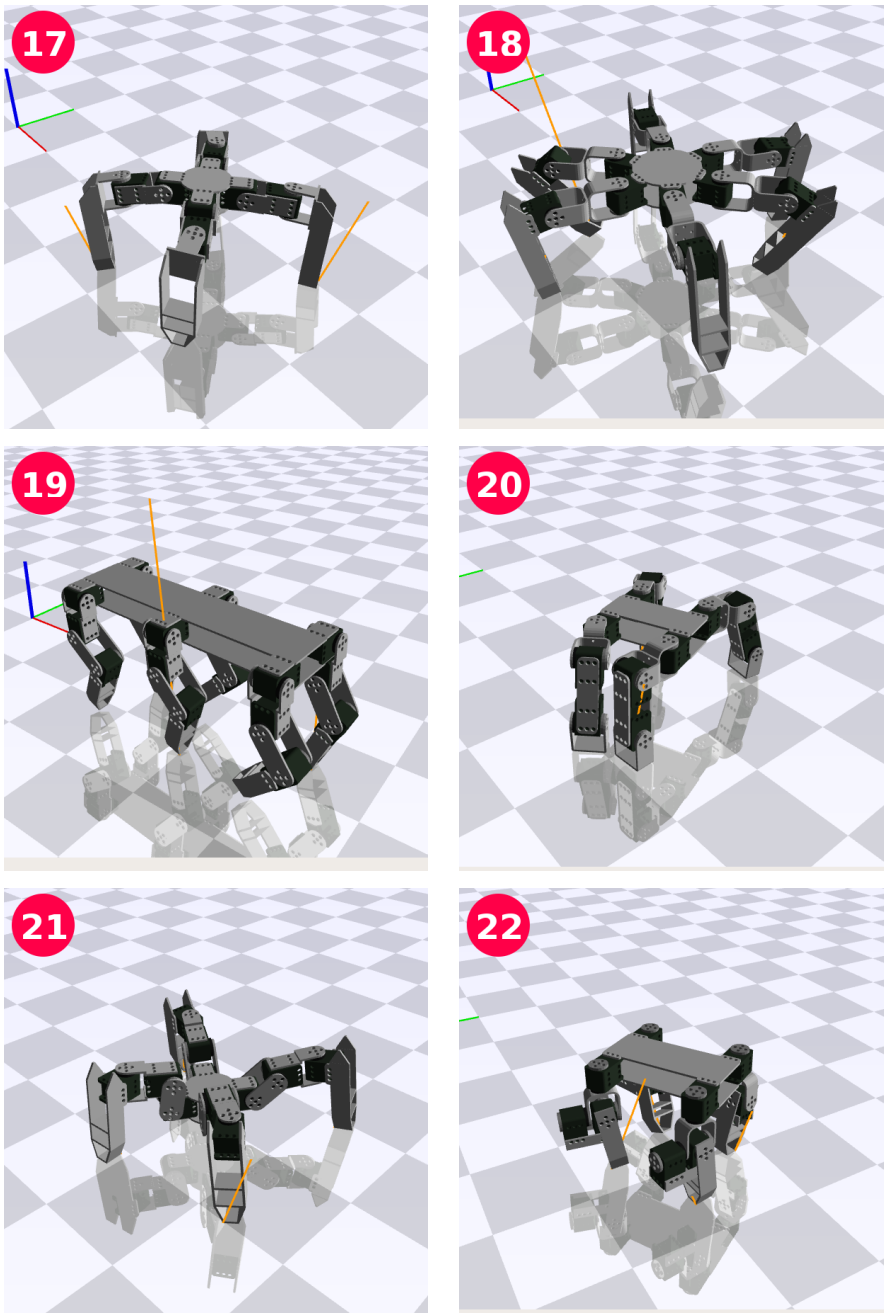

12


13


14


15


16



Ils proposent tous des paramètres morphologiques modofiables, qui sont bornés à chaque fois dans un intervalle.

Les robots sont également définis comment ayant un "avant" qui est le long de l'axe x dans le repère de leurs troncs.

Les pattes de ces robots n'ont pas plus de 3 degrés de libertés. Intuitivement, avoir plus de 3 degrés de libertés crée des ambiguïtés pour atteindre un point dans l'espace opérationnel (x, y, z). Avec 3 degrés de liberté, il existe tout

de même des ambiguïtés de symétrie (figure 3.18), qui sont ici résolues manuellement en limitant l'intervalle d'angles que certains moteurs peuvent atteindre.

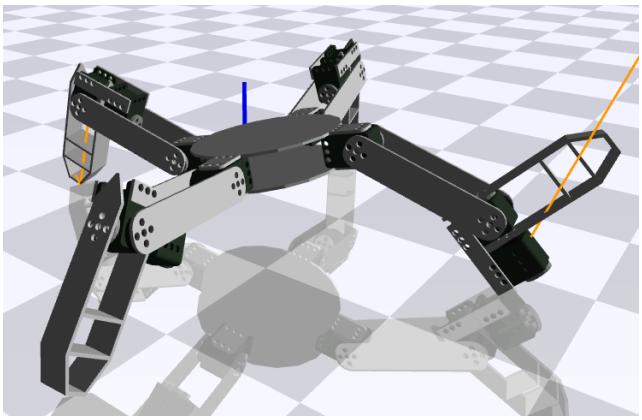


FIGURE 3.18 – Exemple typique du problème de l'ambiguïté cinématique : il y a deux façons d'atteindre le même point dans l'espace cartésien.

Remarquons que l'approximation qui consiste à dire que les pattes n'ont pas de masse, comme RHex (Saranli *et al.* [2001]) ou Little Dog (Neuhaus *et al.* [2011]), est inexacte dans notre cas, tout comme dans la plateforme Metabot, les pièces étant légères et l'essentiel de la masse étant bien dans les pattes.

3.3.2 Contrôleur générique

Présentation

Nous présentons ici un contrôleur inspiré de la littérature ainsi que de notre propre expérience, dont le but est de faire marcher les robots du corpus. Ce dernier peut alors être testé et optimisé sur l'ensemble des robots et des expériences.

L'objectif étant à la fois de valider le fonctionnement du contrôleur empiriquement, mais aussi d'essayer de tirer de conclusions plus générales sur les paramètres des robots optimisés.

Comme expliqué, les robots ont un "avant" le long de l'axe x, la "gauche" le long de l'axe y et les pattes peuvent donc être numérotées en les ordonnant dans le sens trigonométrique (figure 3.19).

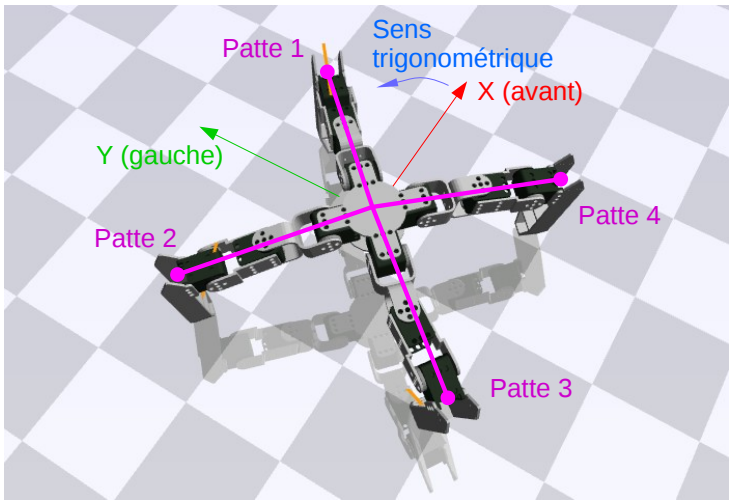


FIGURE 3.19 – La convention utilisée, l'avant du robot est le long de l'axe x et les pattes sont numérotées dans l'ordre trigonométrique.

Fonctionnement

Dans le contrôleur que nous proposons, les pattes suivent des trajectoires périodiques, passant par plusieurs points de contrôle, représentés sur la figure 3.20. Les points A et P sont connus dans la littérature sous le nom AEP (*Anterior Extreme Position*) et PEP (*Posterior Extreme Position*). Le point 0 correspond à la position de la patte lorsque les paramètres dynamiques sont nuls (c'est à dire la posture du robot arrêté). Enfin, le point H correspond au point le plus haut atteint par la patte.

Des polynômes cubiques sont utilisés pour extrapoler les trajectoires, de manière à avoir une dérivée (et donc une vitesse) continue. Idéalement, des polynômes quintiques permettraient d'obtenir une dérivée seconde continue, et donc éviter les pics d'accélération. Cependant, ces robots à bas coût produisent des ordres à basse fréquence pour leurs moteurs, typiquement entre 50 et 100 Hz, les polynômes quintiques n'auront donc plausiblement pas beaucoup d'influence sur le résultat. De plus, cela introduirait plus de paramètres à optimiser dans le contrôleur.

La position 0 est un premier paramètre du contrôleur, il s'agit de la posture du robot lorsque les paramètres dynamiques sont nuls (on la retrouve parfois sous le nom de *home position* dans la littérature). La position de A et de P relativement à 0, ainsi que leur vitesse sont donnés par les paramètres dynamiques du robot. En effet, afin de respecter des trajectoires lisses, la patte a déjà atteint sa vitesse au sol avant de se poser et est encore à sa vitesse au sol

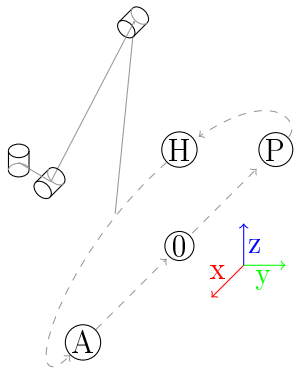


FIGURE 3.20 – La trajectoire suivie par l'extremité de la patte du robot.

au moment de se lever. En revanche, la position de H et sa vitesse sont des paramètres du contrôleur qui modifient la trajectoire de la patte en vol (figure 3.21).

La durée relative qu'une patte passe au sol par rapport au temps du cycle (nommé rapport cyclique dans la littérature) et sa phase relative aux autres pattes sont également des paramètres. Ces derniers définissent l'allure (*gait*) du robot. Aussi, la fréquence, c'est à dire le nombre de cycles que fait le robot en une seconde est un paramètre.

Les *a priori* du contrôleur sont :
- L'allure est régulière (les pattes font le même mouvement à un intervalle de temps de différence)
- La posture est symétrique (pour un robot quadrupède, en position arrêtée, les pattes du robot sont à des positions symétriques selon les axes sagittaux et frontaux)
- Le robot marche avec le tronc "parallèle" au sol (pas d'inclinaison)
- Le point de contrôle H est dans le plan formé par (A, E) et $\overrightarrow{z}$

Et les paramètres de ce contrôleur sont :
- **x, y et z**, la posture du robot quand les paramètres dynamiques sont nuls (c'est à dire le point 0 de la figure 3.20)
- **freq**, la fréquence à laquelle le robot marche (le nombre de pas par seconde pour une patte)
- **Hx, Hz et Hs**, les paramètres du point H
- **support**, le ratio de temps passé au sol pour une patte
- **p2, p3 ... pn** les phases des pattes relatives à la patte 1.

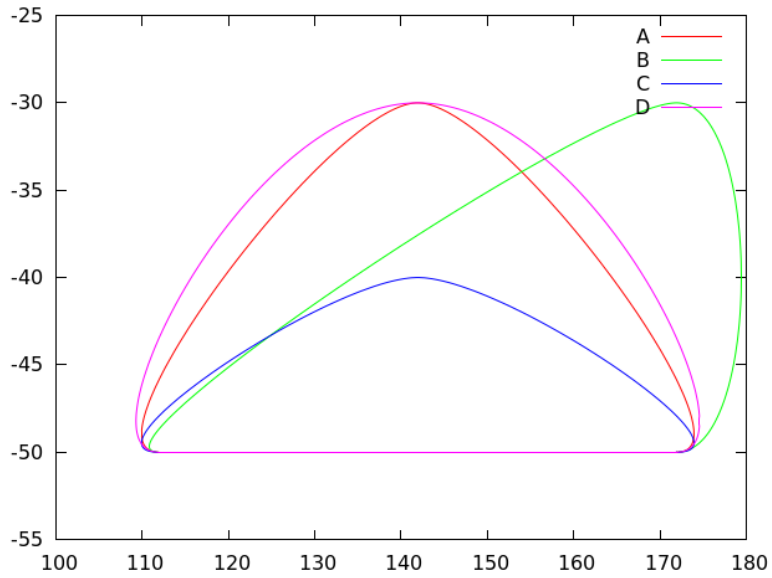


FIGURE 3.21 – La trajectoire obtenue pour la patte avec différents réglages des points de contrôle. A est une trajectoire de référence, la position longitudinale de H est modifiée en B, sa hauteur en C (levée de la patte) et sa vitesse en D.

En réalité, les *a priori* sont aussi des paramètres que nous choisissons d'ignorer pour simplifier les expériences. Toute la faisabilité de l'amélioration d'un contrôleur générique repose sur le fait d'essayer de réduire le nombre de paramètres.

Pilotage

Nous supposons que les pattes ne glissent pas sur le sol, ce qui est équivalent à dire que lorsqu'un patte est posée par le robot, sa position restera fixe dans le repère du monde jusqu'à ce qu'il la lève (bien entendu, les glissements peuvent en revanche effectivement se produire dans la simulation du moteur physique).

Ainsi, si son tronc se déplace linéairement dans le repère du monde, la patte devra se déplacer linéairement dans la direction opposée le long du sol.

Si le robot tourne à vitesse constante en même temps qu'il avance, sa trajectoire sera un cercle tel que présenté sur la figure 3.22. Il tourne alors autour du point C, et selon le rayon de courbure constant $l = \frac{r}{\theta}$. Afin que la patte ne glisse pas, elle doit tourner autour du point C, suivant la trajectoire suivante dans le repère du robot :

$$\begin{cases} P_x cos(t) - (P_y + l) sin(t) \\ P_x sin(t) + (P_y + l) cos(t) - l \end{cases}$$

Où P_x et P_y représente la position de la patte à l'arrêt (posture du robot). Dans le cas particulier où le robot n'avance pas, il tournera autour de lui même, le point C coïncidera avec le centre du robot et la trajectoire des pattes se résumera par :

$$\begin{cases} P_x cos(t) - P_y sin(t) \\ P_x sin(t) + P_y cos(t) \end{cases}$$

Soit une rotation autour du centre du robot.

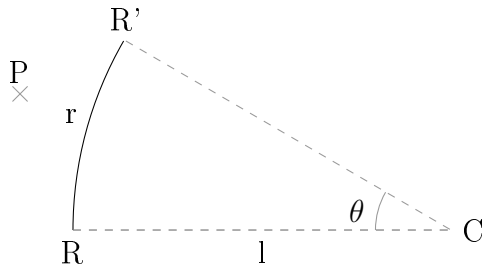


FIGURE 3.22 – Schéma du robot qui avance et tourne en même temps.

3.3.3 Evaluation du score des robots

Expériences de la marche

Dans l'expérience de la marche, le robot est simplement placé initialement à l'origine du repère et évolue sur un sol plat.

Nous proposons pour ces expériences plusieurs scores, le premier est simplement $\frac{\Delta t}{x}$, c'est à dire l'inverse de la vitesse moyenne selon l'axe x [4].

Un second $\frac{C\Delta t}{x}$, qui prend en compte la consommation C du robot pendant l'expérience. Cette consommation est estimée à partir de la somme des torques appliqué par l'ensemble des actuateurs au cours de l'expérience.

Expériences des points de contrôle

Un de nos objectifs étant la production de robots pilotables, nous avons également proposé une expérience qui mettait en pratique les paramètres dy-

4. Les problèmes d'optimisations sont généralement présentés comme des problèmes de minimisation, d'où l'utilisation de l'inverse de la vitesse

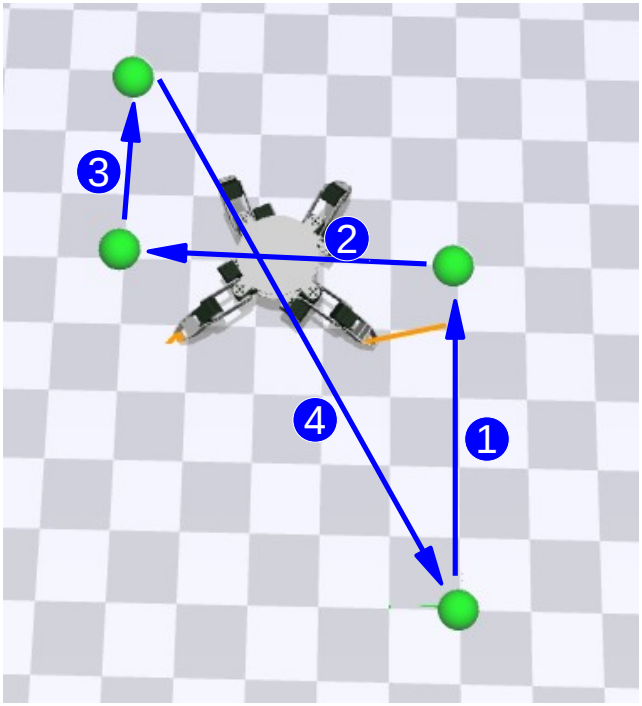


FIGURE 3.23 – Les checkpoints (en vert) sont des points cibles à atteindre par le robot. Lorsque ce dernier est suffisament proche d'un checkpoint, le nouveau checkpoint devient le point suivant du parcours montré ci-dessus.

namiques du robot. Dans cette dernière, le robot doit atteindre des points de contrôle (checkpoints) dans l'ordre (figure 3.23). Les paramètres du contrôleur du robot sont mis à jour de la manière suivante :

$$\begin{cases} \dot{\theta} = min(k_\theta \Delta_\theta, \theta_M) \\ \dot{x} = \cos(\Delta_\theta) x_M \end{cases}$$

Où :

— Δ_θ représente l'erreur angulaire par rapport à la cible
— k_θ, θ_M et x_M sont des paramètres à optimiser (resp. le gain d'asservissement en rotation, la vitesse angulaire maximum et la vitesse d'avance maximum)

Lorsque le robot est arrivé suffisamment près d'une cible, la cible suivante est alors utilisée.

Là encore, nous avons utilisé deux scores, le premier étant la durée t de l'expérience, et le second Ct la durée multiplié par le coût.

Si tous les points de contrôle ne sont pas atteints, le score $nB + d$ est utilisé, où n est le nombre de points de contrôle restant à atteindre, B un très

grand nombre, et d la distance au prochain point de contrôle, créant ainsi un "escalier" permettant d'aider l'algorithme d'optimisation à converger.

Pénalités

Des pénalités sont appliquées aux robots qui s'auto-collisionnent, ainsi qu'aux robot qui touchent le sol avec une autre pièce que leurs pattes. Ceci permet d'éviter les comportements de "rampants". Une telle pénalité a déjà été utilisée sur un robot physique par Maes et Brooks [1990] à l'aide de boutons sous le ventre.

3.3.4 Algorithme d'optimisation

Nous avons utilisé la méthode d'optimisation en boîte noire CMA-ES pour ces expériences (Hansen et Ostermeier [2001]), avec l'algorithme BIPOP.

Le principe de CMA-ES est de sélectionner à chaque itération un certain nombre de candidats selon une loi normale multivariée. La moyenne et la matrice de covariance de cette loi sont des paramètres initiaux de l'algorithme. La fonction de score est alors évaluée pour chacun de ces candidats, ce qui permet de calculer une nouvelle moyenne et de mettre à jour la matrice de covariance. De cette manière, la qualité de la moyenne s'améliore, et la matrice de covariance nous donne une direction à suivre lors de l'échantillonage des candidats, selon les axes qui expliquent le mieux la variance pour améliorer le score (figure 3.24).

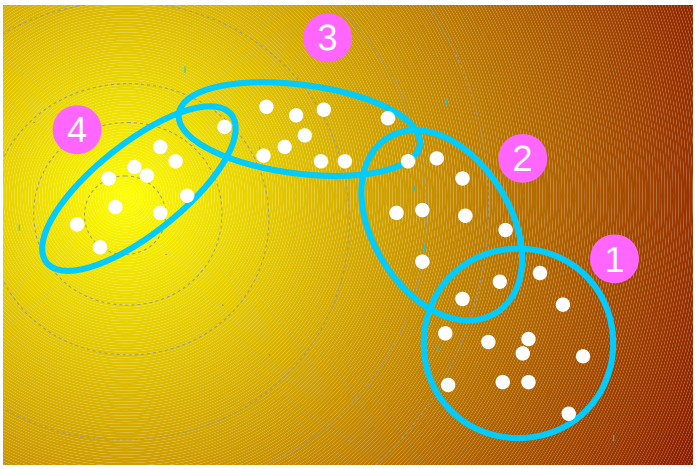


FIGURE 3.24 – Illustration de CMA-ES, les candidats sont les points blancs, et les ellipses bleues les matrices de covariances de chaque itération.

Cet algorithme nécessite néanmoins des méta paramètres qui modifient ses conditions d'arrêts, ou de redémarrage. Un des plus importants à régler dans notre cas, chaque expérience pouvant durer plusieurs secondes, est la tolérance de la fonction de score (*ftolerance*), qui détermine à quel point le score à besoin d'être amélioré pendant un certain nombre de générations pour arrêter la recherche. En effet, un mauvais réglage peut entraîner des calculs inutiles et très longs qui améliorent de manière très insignifiante le score, là ou il serait plus intéréssant d'essayer d'autres candidats.

Malgré la popularité et les performances de CMA-ES, le problème que nous essayons d'optimiser ici est largement multimodal, et nous ne pouvons évidemment pas affirmer que le résultat qui est trouvé par cette méthode est globalement optimal.

La bibliothèque *libcmaes* (Benazera [2014]) a été utilisée. Cette dernière permet de réaliser les expériences d'optimisation en tirant profit des processeurs disponibles pour paralléliser les évaluations de la fonction de score.

3.3.5 Considérations pratiques

Afin d'améliorer les performances, les côtes ont été arrondies au mm, et les pièces ainsi que leur dynamique qui ont déjà été calculées sont gardées en cache.

Un programme principal lançait donc des instances de simulation, responsables uniquement de fournir un score d'après un jeu de paramètres. L'architecture est présentée en figure 3.25.

Les calculs ont été exécutés sur des machines de calcul parallèle (Par exemple : AMD Opteron 48 coeurs à 2.2 Ghz). Chaque optimisation ayant une durée variable allant généralement de 2 à 10 heures.

3.4 Résultats

À l'aide du contrôleur présenté précédemment, tous les robots ont été capables de marcher, et d'atteindre les points de contrôle donnés.

Les robots ainsi optimisés peuvent être vus dans cette vidéo : https://youtu.be/GF1KM7JrmC0

3.4.1 Durée de support

En traçant l'histogramme des durées de support, la distribution semble normale autour d'une moyenne d'environ 0.44 (figure 3.26). Autrement dit,

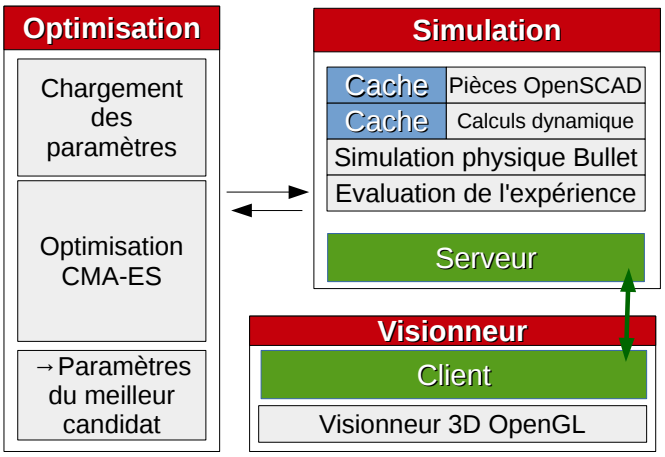


FIGURE 3.25 – Architecture des expériences d'optimisation.

chaque patte passe en moyenne 44% du cycle de marche au sol.

Une explication à cette apparente non-stabilité est que le robot s'affaisse, à la fois à cause de l'asservissement du moteur mais aussi à cause de la simulation du jeu.

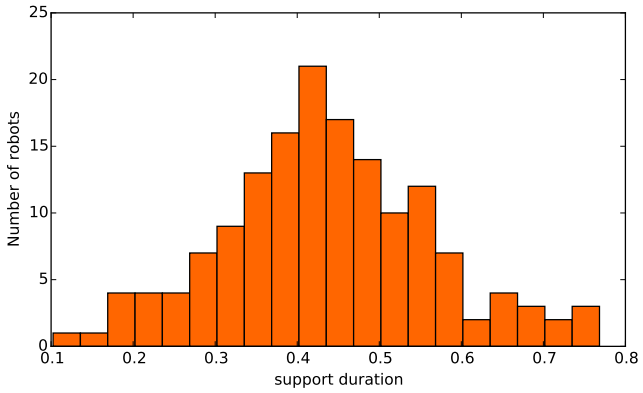


FIGURE 3.26 – Histogramme de la durée de support (ou rapport cyclique) pour l'ensemble des robots optimisés.

Quasiment aucun des robots optimisés n'adoptent une locomotion statiquement stable. En fait, le rapport cyclique est un compromis entre la stabilité et la vitesse. En effet, à vitesse égale, une patte à rapport cyclique haut devra parcourir au sol une plus grande distance. En conséquence, elle devra aller plus

vite pour revenir à sa position antérieure. Aussi, la taille maximum des pas est bornée par l'espace atteignable de la patte, rendant plus avantageux les petits rapports cycliques pour la vitesse.

3.4.2 Phases des pattes

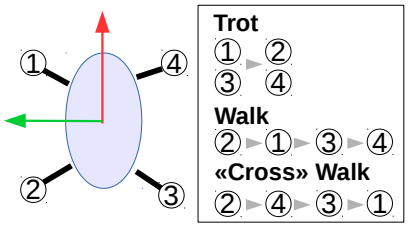


FIGURE 3.27 – Les allures (ordre de levée des pattes) mentionnées dans cette partie.

En traçant les points (p_2, p_3, p_4) (paramètres du contrôleur, cf 3.3.2) pour tous les quadrupèdes optimisés, toutes expériences confondues, il apparaît que les points se regroupent dans une ellipsoide assez plate (*cf.* figure 3.28) qui est en fait presque un axe. Une analyse en composante principale révèle d'ailleurs que 70% de la variance est expliquée par un seul axe (figure 3.29).

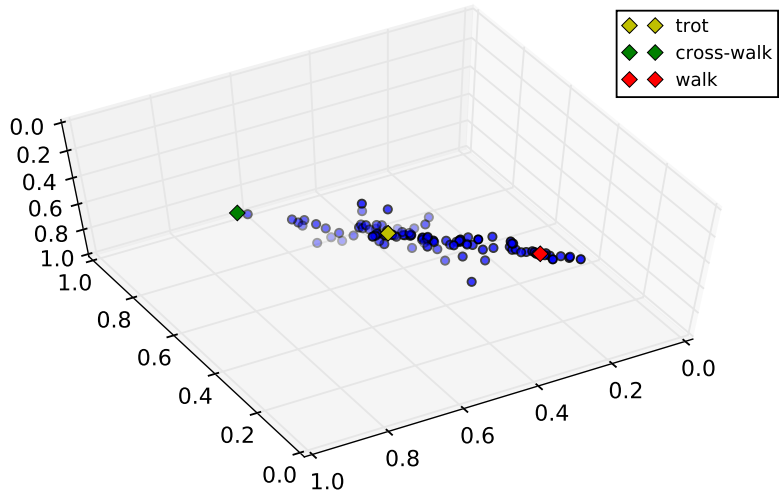


FIGURE 3.28 – p_2, p_3 et p_4 pour tous les robots quadrupèdes optimisés sont dans une ellipsoide (p_3 a été décallé de 0.5 pour la centrer).

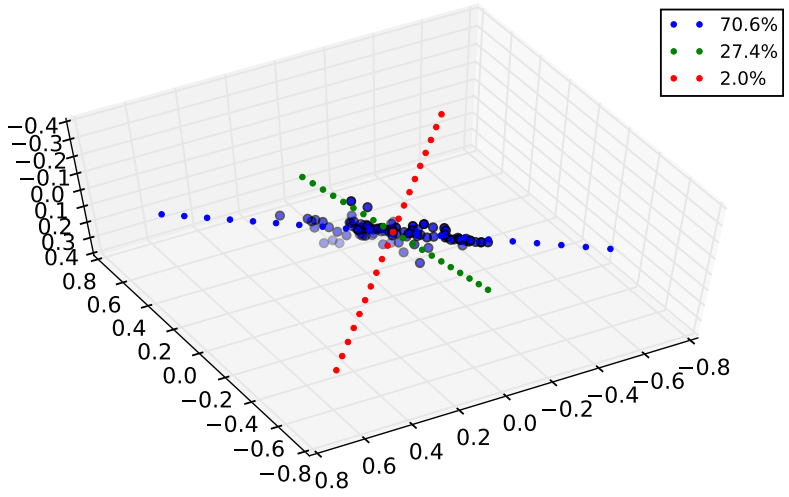


FIGURE 3.29 – Analyse en composante principale des points de la figure 3.28, les données sont centrées et la légende correspond aux proportions des variances expliquées par chacun des axes principaux.

Une interprétation géométrique de l'axe est l'apparition d'une symétrie des pattes opposées par le plan sagittal, chaque patte opposée par ce dernier effectuant le même mouvement à un demi cycle près. Le degré de liberté correspondant à l'axe principal du nuage de points étant le déphasage entre les pattes avant et arrière. Une autre façon de le dire est que ni les deux pattes avant ni les deux pattes arrière ne sont levées en même temps.

Lorsque ces deux groupes de pattes sont décalés d'un demi cycle, il s'agit du trot (au centre de la figure), et un déphasage dans un sens nous conduit vers la marche (figure 3.27).

Cet axe réduit le degré de liberté de l'allure à un seul scalaire, ce qui signifie qu'il existe un ou des critères permettant de déterminer qu'une allure est optimale.

Un critère standard pour analyser les locomotions lentes, dans lesquelles le robot a toujours au moins trois pattes au sol est de mesurer la distance entre le projeté de son centre de masse au sol avec la bordure la plus proche de son polygone de sustentation. Ainsi, plus le point est loin de cette bordure, moins le robot risquera de rouler sur une arête de ce polygone et donc de tomber, ou de gaspiller de l'énergie. Ce critère est donc une évaluation de la stabilité statique de la marche, que l'on peut noter :

$$\lambda_s = -\int_t m(P_{zmp}, S)\mathrm{d}t$$

Où t désigne le temps et m est la distance de la position du ZMP P_{zmp} à la bordure du polygone de sustentation S. Plus ce nombre est petit, plus la marge de sécurité qui permet d'éviter de sortir de la surface de sustentation est grande.

Cependant, les robots qui ont été étudiés dans nos expériences ont convergé vers des durées de support variables, dont la plupart suggère que la locomotion n'est pas stable. Ce résultat n'est pas surprenant, étant donné que la locomotion stable n'est pas rapide (comme expliqué dans la section précédente), et coïncide avec l'expérience empirique que nous avions de la locomotion sur la plateforme Metabot, où le trot a été choisi comme locomotion car il était dans la pratique efficace.

Le critère de stabilité n'a donc évidemment pas de sens dans ce cas, car il peut exister des phases dans lesquelles seulement deux des pattes du robot sont au sol, résultant en un polygone de sustentation réduit à un segment, dans lequel il est peu probable que le ZMP se trouve. Autrement dit, les robots roulent autour de ces axes quand ils exécutent ces locomotions. En revanche, la position du ZMP du robot fournit malgré tout une information, car sa distance au polygone de sustentation nous indique avec quel bras de levier un moment peut se créer sur le robot (figure 3.30). Nous proposons donc un critère d'efficacité dans le cas instable, qui peut se noter :

$$\lambda_u = \int_t d(P_{zmp}, S)\mathrm{d}t$$

Où t désigne le temps et d mesure la distance entre la position du ZMP P_{zmp} à le polygone de sustentation du robot S, ou 0 si le point est dans le polygone. Les deux critères λ_s et λ_u sont en fait des cas disjoints, nous proposons donc un critère synthétique λ :

$$\lambda = \left| \begin{array}{l} \lambda_s \text{ si } \lambda_u = 0 \\ \lambda_u \text{ sinon} \end{array} \right.$$

Si λ est négatif la locomotion sera stable, et plus la valeur sera faible plus la marge de sécurité sera bonne. Si λ est positif, la locomotion sera instable (le robot roulera sur des arêtes), et plus la valeur sera faible moins il roulera.

Nous avons fait l'expérience géométriquement, en simulant notre contrôleur et en récupérant la position du bout des pattes à chaque pas de temps d'un cycle de marche. Le critère λ ci-dessus peut donc être intégré numériquement. En faisant l'approximation que le ZMP du robot est le projeté du centre de

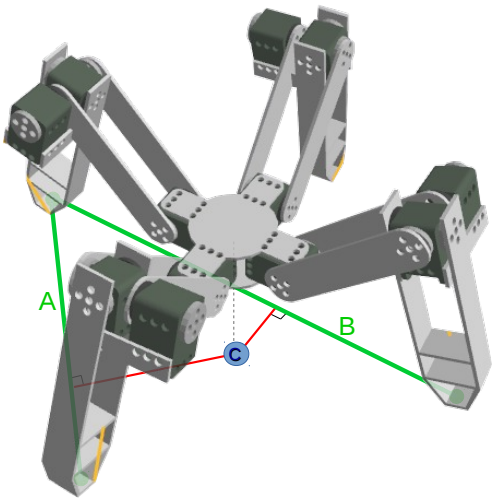


FIGURE 3.30 – Si seulement deux des pattes du robot sont au sol à un instant donné, son polygone de sustentation est en fait un segment. A et B en sont des exemples. La distance du ZMP (ici le point C) à l'axe fournit une information intéréssante. Ici, il apparaît que rouler autour de l'axe B est plus stable que de rouler autour de l'axe A.

son tronc sur le sol, le problème est uniquement géométrique. Il s'agit donc simplement de calculer la distance d'un point à un polygone pour chaque pas de temps d'un cycle de marche. Des exemples de cette représentation géométrique sont montrés sur les figures 3.31 et 3.32.

En simulant ce calcul avec un rapport cyclique de 0.75, les allures ayant un score λ le plus bas sont situées sur l'axe autour de la marche (figure 3.33). Le critère est différent, mais ce résultat est en fait similaire à McGhee et Frank [1968b]. En fait, dans le cas précis de la marche, λ est négatif, car l'ensemble des positions du robot au cours du cycle incluent le projeté du centre du masse.

Avec un rapport cyclique de 0.5, les meilleures allures sont situées sur l'axe autour du trot (figure 3.34). Ce résultat est dû au fait que le trot propose des polygones de sustentation qui sont en fait des diagonales successives, minimisant ainsi le levier et donc améliorant la stabilité finale, ici, λ est positif.

Kurazume *et al.* [2002] a proposé une approche

Nous avons recommencé cette simulation géométrique pour toutes les valeur de rapport cyclique entre 0.2 et 0.8, en gardant à chaque fois les phases de la meilleure allure, et l'ensemble des points obtenus se trouve former le même axe que celui des robots optimisés (figure 3.35). Malgré les approximations du modèle géométrique que nous venons de proposer, il semble donc que notre

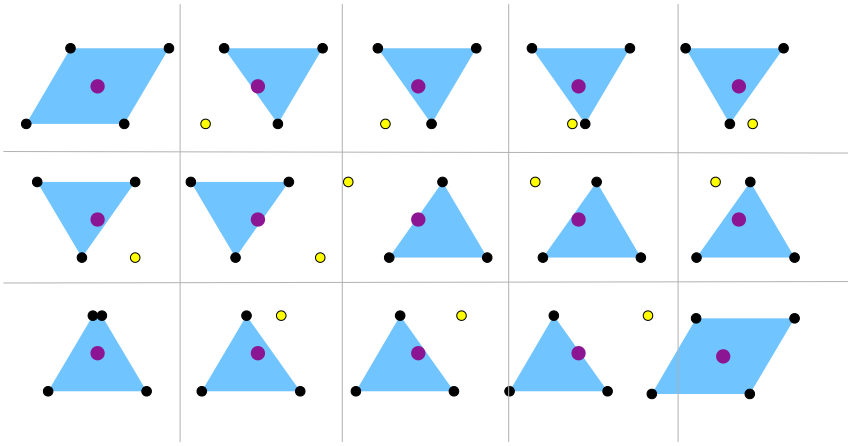


FIGURE 3.31 – La représentation géométrique de la marche (le sens de marche est vers la droite, le diagramme se lit comme une bande dessinée), la patte levée est représentée en jaune et les pattes au sol en noir, le centre de masse du robot (en violet) est maintenu dans la surface de sustentation (en bleu). Ici, $\lambda < 0$. Cette allure est λ-optimale pour un rapport cyclique de 0.75.

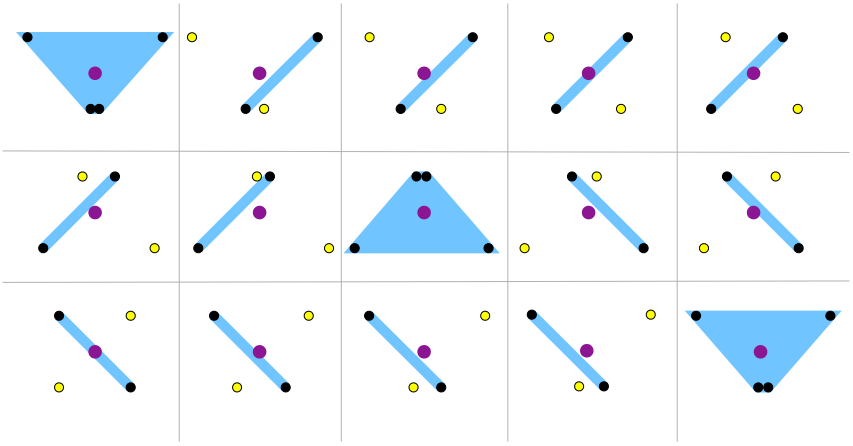


FIGURE 3.32 – Idem que la figure 3.31 mais pour le trot. Ici, la surface de sustentation est résumée par un segment pendant les phases où seulement deux pattes sont au sol. Ici, $\lambda > 0$. Cette allure est λ-optimale pour un rapport cyclique de 0.5.

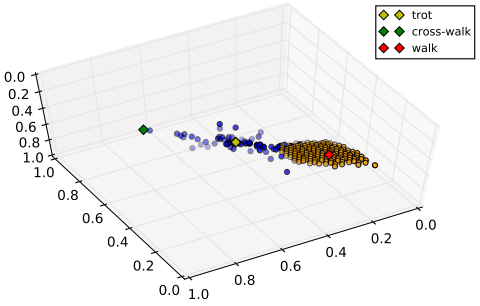


FIGURE 3.33 – Les points orange sont ceux qui ont obtenus le meilleurs score suite à la simulation géométrique de stabilité, pour un rapport cyclique de 0.75.

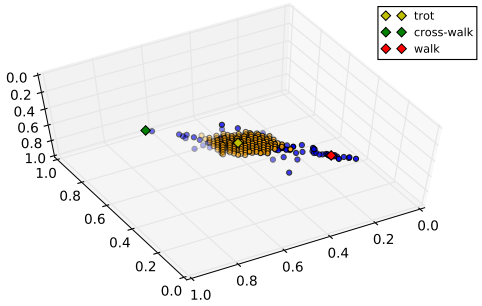


FIGURE 3.34 – Idem que la figure 3.33, avec un rapport cyclique de 0.5.

critère corrobore les résultats obtenus.

Nous avons réalisé la même expérience sur un robot à 6 pattes. Lorsque le rapport cyclique est de 0.5, l'allure λ-optimale est une allure stable dans laquelle deux groupes de trois pattes agissent ensemble, qui ressemble de ce fait au trot et que l'on retrouve parfois sous le nom de *tripod gait* (Lee *et al.* [1988], figure 3.36).

En baissant le rapport cyclique à 0.4, la *tripod gait* ne peut plus être optimale car, pour synchroniser les pattes par groupes de trois il existerait des phases ou aucune d'entre elles ne toucherait le sol. En conséquent, les allures λ-optimales adoptent des déphasages qui produisent des instants avec deux pattes de support (figure 3.37). De telles allures ont été trouvées en simulation.

L'interêt de cette approche est qu'elle suggère une méthode permettant de proposer des allures basées sur un critère géométrique qui est très rapide à estimer. Par exemple, elle peut s'exécuter sur un robot à 5 pattes, pour trouver une allure à adopter (figure 3.38).

En réalité, le seuil de rapport cyclique β à partir duquel λ deviendra positif correspond au moment ou la stabilité statique ne peut plus être assurée, c'est à dire lorsque la surface de sustentation ne peut plus être au moins un triangle à tout moment , et se situe à partir du seuil :

$$\beta < \frac{3}{N}$$

Où N est le nombre de pattes (Ting *et al.* [1994], Song [1984]). Pour des robots de 4, 5 et 6 pattes, ce seuil correspond à des rapports cycliques respec-

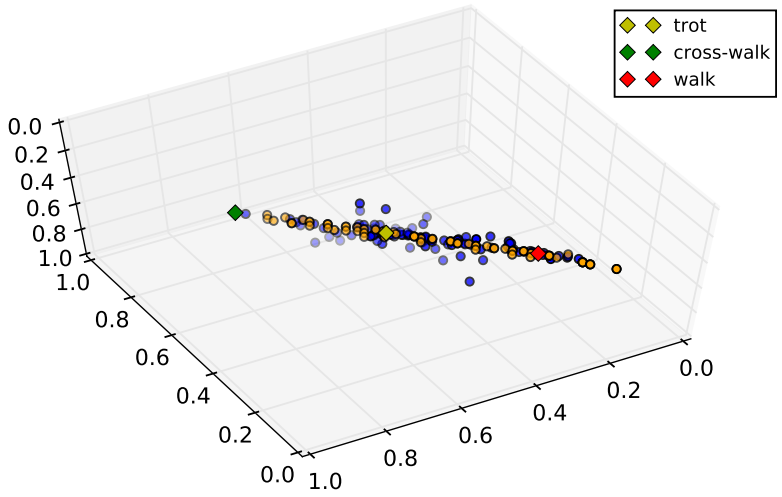


FIGURE 3.35 – Les points orange sont λ-optimaux pour des valeurs variables de rapport cyclique de 0.2 à 0.8.

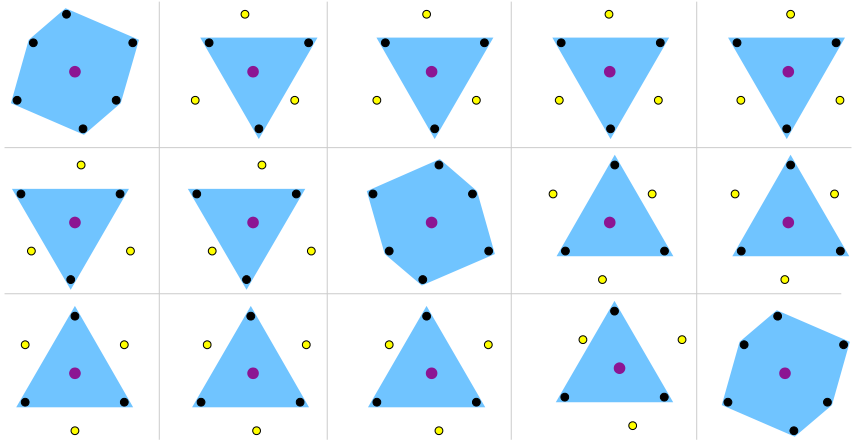


FIGURE 3.36 – La *tripod gait* est stable et λ-optimale pour un robot de six pattes dont le rapport cyclique est 0.5.

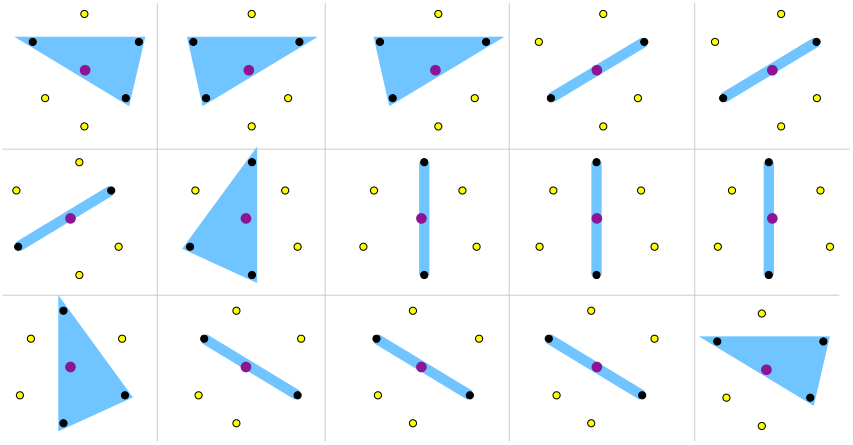


FIGURE 3.37 – Cette allure instable est λ-optimale pour un robot de six pattes avec le rapport cyclique de 0.4.

tifs de 0.75, 0.6 et 0.5.

En se basant sur le calcul numérique de λ, nous pouvons également tester à quel point les allures optimales restent les mêmes lorsque le robot suit un vecteur de direction différente.

Pour cela, nous avons trouvé les allures λ-optimales en faisant varier le vecteur de direction que le robot devait suivre (en translation pure). Dans le cas d'un robot quadrupède et d'un robot hexapode, avec un rapport cyclique de 0.5, le trot et la *tripod gait* restent les marches λ-optimales quel que soit l'orientation du vecteur (figure 3.39). Autrement dit, lorsqu'un robot quadrupède ou hexapode se déplace en crabe ou en selon une diagonale, les phases des pattes restent, selon ce critère les mêmes. En revanche, pour un robot pentapode, l'allure λ-optimale est différente selon l'un des cinq cadrans dans lequel le vecteur de direction se situe.

3.4.3 Position du point H

La position du locus (c'est à dire du point de contrôle H dans la figure 3.20) le long de l'axe de la patte est en moyenne sur l'ensemble des robots un peu vers l'avant (moyenne 0.22).

3.4.4 Taille des robots

En faisant la somme des paramètres morphologiques d'un robot, il est possible d'obtenir une métrique liée à son envergure. Ainsi, on constate, ce qui est plutôt intuitif, que :

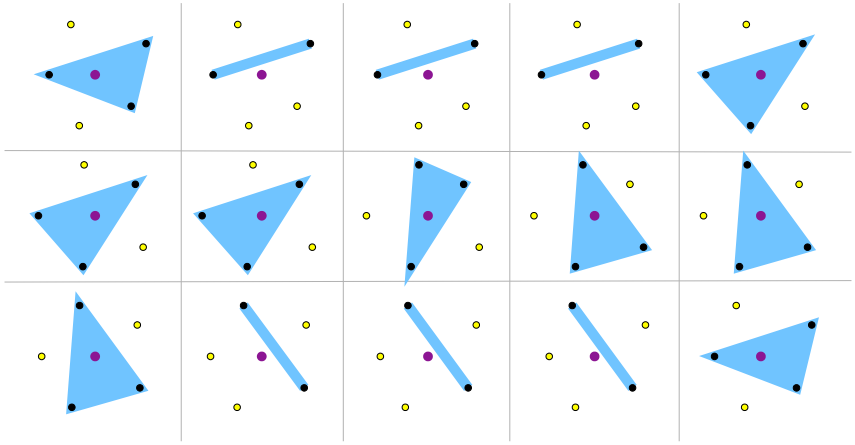


FIGURE 3.38 – Cette allure instable est λ-optimale pour un robot de cinq pattes avec le rapport cyclique de 0.5.

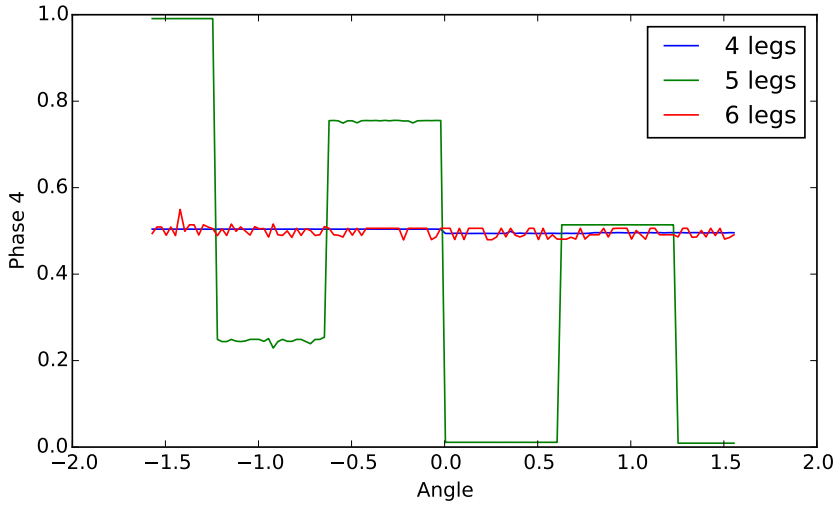


FIGURE 3.39 – Phase de la patte 4 dans l'allure λ-optimale correspondant à un vecteur de direction orientée selon l'angle donné en abscisse, pour des robots de 4, 5 et 6 pattes. La patte 4 est ici caractéristique de l'allure.

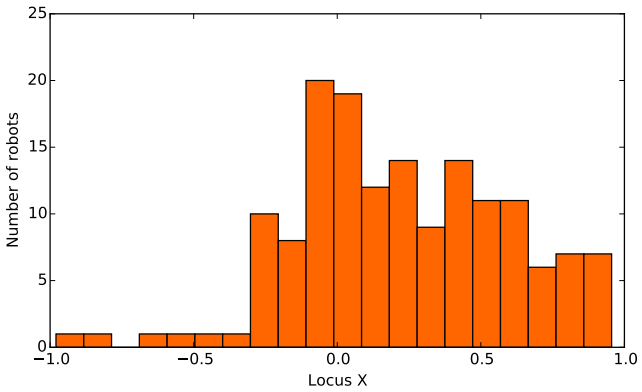


FIGURE 3.40 – Position du locus (point H) le long du pas (normalisé entre -1 et 1 par rapport à la taille du pas).

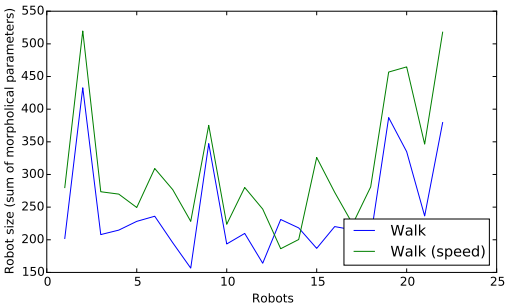


FIGURE 3.41 – Taille des robots obtenus dans les expériences marche (critère vitesse/consommation) et marche (critère vitesse).

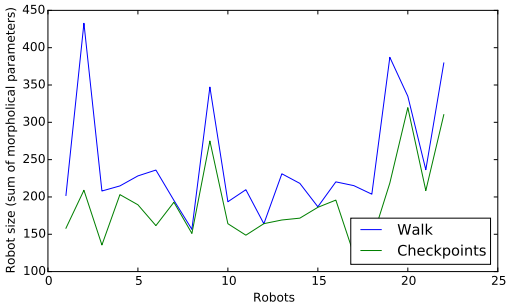


FIGURE 3.42 – Taille comparative des robots obtenu dans les expériences marche et checkpoints.

— Les robots produits par l'optimisation selon le critère de la vitesse sont globalement plus grands que ceux produits avec la contrainte d'optimisation de la consommation (figure 3.41). Cela s'explique simplement par le fait que la cinématique permet de faire de plus grand pas, étant donné que les bras de leviers importent moins.
— Les robots produits pour l'optimisation des checkpoints sont plus petits que ceux produits pour optimiser la marche avec la consommation, cela s'explique par la maniabilité du robot (rapport entre le poids et la puissance) qui est accrue (moins d'inertie).

3.4.5 Points de contrôle

On remarque que l'ensemble des robots a bien été capable de réaliser l'expérience des points de contrôle (figure 3.43).

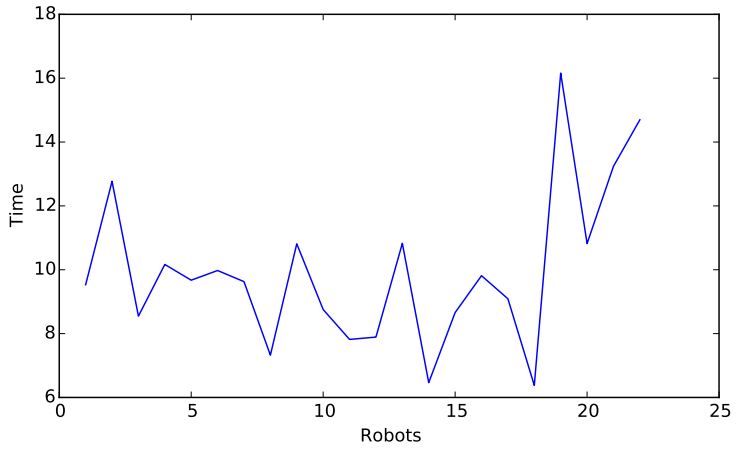


FIGURE 3.43 – Durée de chaque expérience optimisée des points de contrôle pour chaque robot, en secondes.

Cela démontre que le contrôleur est capable de les faire avancer et tourner, de manière à les piloter vers des points arbitraires.

3.4.6 Swing

Nous avons lancé les mêmes expériences avec une option de swing permettant au robot de balancer son corps latéralement au cours de chaque pas. Ce swing se matérialise par deux paramètres qui sont l'amplitude et la phase (figure 3.44).

Ce test a été motivé par l'idée que le centre de masse du robot aurait alors plus de chances de suivre les bordures de son polygone de sustentation.

Kurazume *et al.* [2002] a proposé une approche visant à améliorer le trot en essayant de minimiser les accélérations du corps pendant la marche, ce qui passe par le suivi du ZMP des diagonales formées par les pattes au sol pendant le cycle de marche.

Les courbes comparant le score obtenu sur les robots avec et sans le swing (figure 3.45 et 3.46) sont très proches, ce qui nous indique que la méthode d'optimisation a convergé de manière répétable vers des scores similaires pour

l'ensemble des robots, mais il ne semble pas y avoir une amélioration particulière liée à l'ajout de ce paramètre.

Il est probable que le swing gaspille de l'énergie mécanique, rendant son utilisation peu avantageuse.

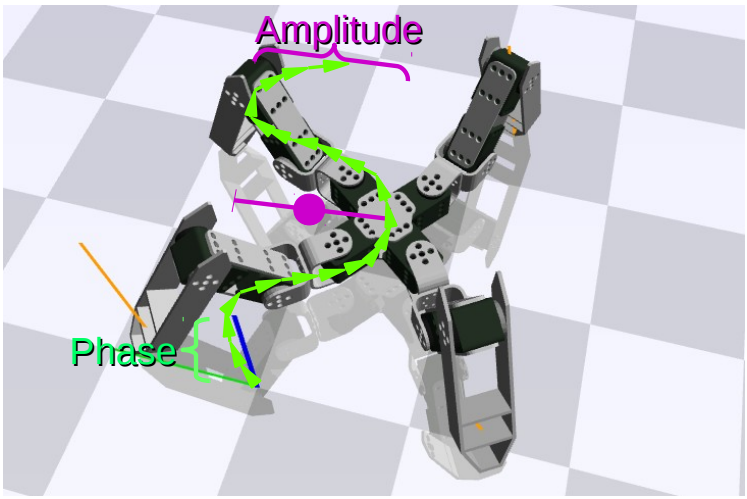


FIGURE 3.44 – Le paramètre de swing est un mouvement oscilatoire latéral paramétré par une amplitude et une phase qui s'ajoute au contrôleur.

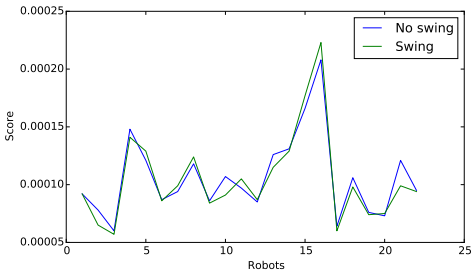


FIGURE 3.45 – Comparaison du meilleur score obtenu pour chaque robot avec et sans le swing (expérience de marche).

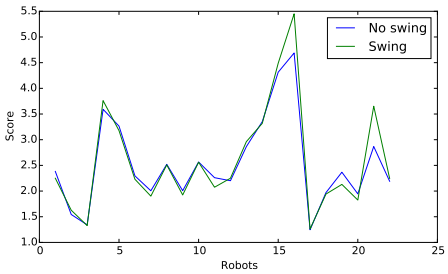


FIGURE 3.46 – Comparaison du meilleur score obtenu pour chaque robot avec et sans le swing (expérience de points de contrôle).

3.5 Conclusion

Dans ce chapitre, nous avons présenté un environnement permettant de manipuler des robots à pattes paramétrique. Nous avons également décrit une

première série d'expériences permettant d'évaluer un contrôleur générique pour les robots à pattes.

Il a été montré en simulation que ce contrôleur expert inspiré de la littérature est efficace sur un ensemble de robots qui ont servi de corpus, puisqu'il permet de les piloter lors de l'expérience des points de contrôle, et donc de les amener en un point cible.

Nous avons également montré que les allures des robots quadrupèdes suivaient une règle de stabilité, et avons proposé un critère qui semble correspondre à cette règle.

Chapitre 4

Capteurs de pression bas coût

Les robots humanoïdes à taille humaine récents intègrent en général des capteurs de force six-axes dans leurs chevilles (Englsberger *et al.* [2014] en présente un exemple). iCub est sans doute l'exemple du plus petit robot humanoïde à intégrer de tels capteurs (Tsagarakis *et al.* [2007]). Cependant, il n'existe pas d'alternative à bas coût. De plus, la conception mécanique d'un capteur six-axes requiert une certaine précision et un banc de calibration précis est nécéssaire pour obtenir les forces et les moments à partir des mesures.

D'un autre côté, la robotique à patte promet d'obtenir des robots robustes aux imperfections de son environnement et aux éventuelles perturbations qu'il rencontre. C'est par exemple un des caps qui a été franchi à la RoboCup en 2015, en commençant à faire jouer les robots humanoïdes sur du gazon synthétique d'environ 3 cm d'épaisseur. Cette modification du règlement a rendu la locomotion des robots plus difficile, favorisant les robots plus grands, et les forçant à effectuer des adaptations mécaniques et dans le contrôle.

C'est pourquoi nous avons conçu une variante à bas coût de capteur de pression, qui a été déployée et utilisée sur notre robot Sigmaban (Fabre *et al.* [2016]), qui a participé à la RoboCup en ligue soccer kid size où nous sommes arrivés respectivement en 3ème et en 1ère place en 2015 et en 2016 (Allali *et al.* [2016]). C'était d'ailleurs le seul robot humanoïde de notre ligue à être doté de capteurs de pression sous les pieds.

4.1 Architecture

4.1.1 Choix des jauges de contraintes

Pour mesurer des forces, deux technologies sont généralement proposées, les FSR (pour *force sensor resistor*) et les jauges de contraintes (figure 4.3).

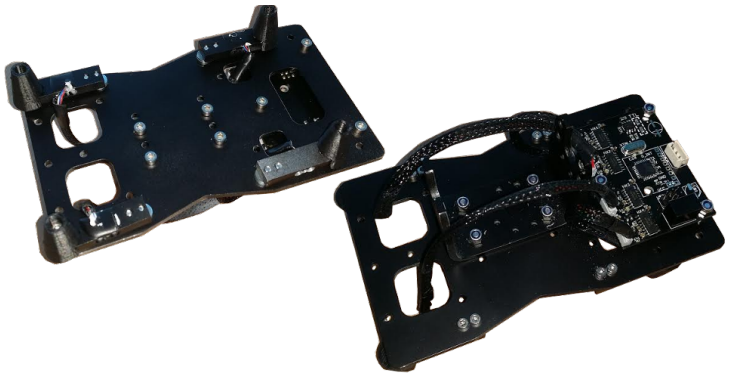


FIGURE 4.1 – Une photo du prototype vue de dessous (à gauche) et de dessus (à droite).

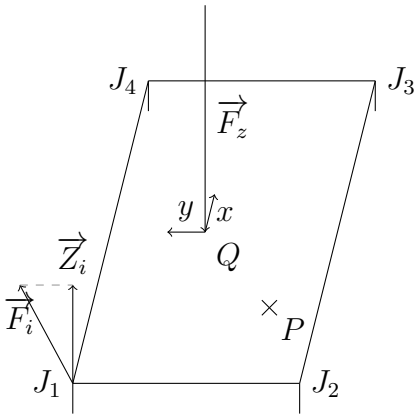


FIGURE 4.2 – Schéma des symboles utilisés, les quatre jauges sont placées en quatre points J_i et mesurent la composante verticale Z_i de la force appliquée.

Les FSR sont composées de trois couches : un réseau de pistes électroniques entrelacées, un "espaceur" et un polymère conducteur. Lorsqu'une force est appliquée, le polymère s'appuie contre les pistes, les mettant en contact et diminuant la valeur de la résistance globale.

Les petits robots humanoïdes Nao (Gouaillier *et al.* [2009]) et DARwIn-OP (Ha *et al.* [2011]) intègrent ou proposent d'intégrer des FSR sous les pieds pour mesurer les forces de réactions du sol.

Les jauges de contraintes sont des réseaux de résistances capable de mesurer des petites déformations d'éléments mécaniques sur lesquels elles sont

collées.

Notre expérience avec ces deux technologies nous a fait conclure pour plusieurs raisons que les jauges de contraintes fournissaient une meilleure contribution en la mesure de force que les FSR.

Un premier problème lié aux FSR est le caractère non linéaire de leur réponse en fonction de la force appliquée (Hollinger et Wanderley [2006]), qui nécessite une calibration spéciale. De plus, le résultat est assez sensible à la façon d'appuyer sur le polymère. En effet, il est plus difficile d'intégrer mécaniquement les FSR, car, contrairement aux jauges de contraintes qui ne viennent que mesurer la déformation de la mécanique, les FSR requièrent un mécanisme qui permettra de faire passer toute la force par eux (figure 4.4). De plus, la résilience dans le temps [1] des FSR s'est révélée incertaine, ce qui est probablement dû à l'élasticité du polymère.

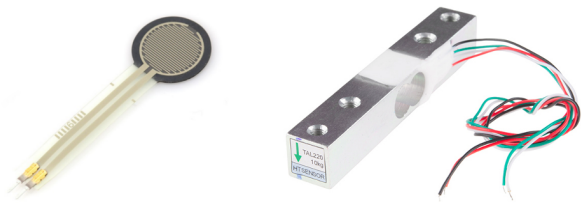


FIGURE 4.3 – Les FSR (à gauche) et les jauges de contraintes (à droite) sont deux technologies permettant de mesurer des forces.

4.1.2 Fonctionnement

Le principe de fonctionnement repose sur une très légère déformation mécanique d'un élément qui est mesuré à l'aide d'un réseau de résistances qui se déforment. Ces réseaux de résistances sont appelés jauges de contraintes.

Ces jauges sont collées à des parois d'une forme mécanique, qui est souvent conçue pour contenir un point d'inflexion permettant de mesurer simultanément une expansion et une compression (*cf.* figure 4.5). Cette configuration permet de placer les résistances en pont de Wheatstone, amplifiant ainsi le signal de sortie, tel que présenté sur la figure 4.6.

1. La résilience dans le temps désigne la capacité d'un corps à retrouver ses conditions initiales après une altération. En l'occurence, dans le cas des capteurs de force, elle désigne leur capacité à retrouver leur tare, c'est à dire que les valeurs du capteur ne dérivent pas au cours du temps et des chocs qu'il subit.

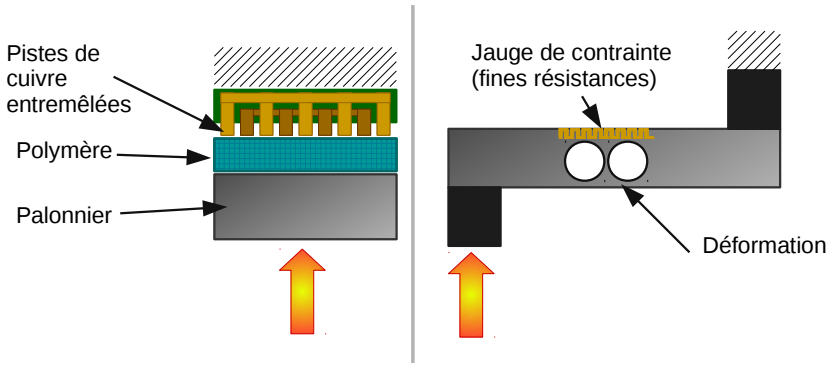


FIGURE 4.4 – Dans le cas des FSR (à gauche), un réseau de pistes de cuivres entremêlées est placé sous un polymère conducteur, qui lui même est placé sous un palonnier (élément mécanique qui sert à "répartir" l'appui). Plus la force sera grande, plus le polymère sera compressé contre les pistes faisant baisser la résistance. Les jauges de contraintes (à droite) sont des résistances fines que l'on colle sur un élément mécanique qui se déforme. La déformation provoque une modification très légère de la longueur de la résistance, et fait varier sa valeur.

La variation infime de la résistance se mesure donc par une variation infime du voltage à la sortie de ce pont. Pour pouvoir échantilloner cette variation de typiquement quelques mV, un amplificateur est nécéssaire, généralement couplé avec un convertisseur analogique numérique de type $\Sigma\Delta$.

Le modèle utilisé pour établir alors la force qui s'applique sur un élément mécanique est linéaire :

$$F = K_i C_i V_i$$

Où K_i (N/m) est la rigidité mécanique, C_i (m/V) la relation linéaire entre le voltage mesuré dans la jauge de contrainte et la déformation mécanique et V_i (V) le voltage mesuré. On utilisera donc une phase de calibration pour trouver la valeur de $K_i C_i$, qui est le lien linéaire entre le voltage et la force appliquée (N/V).

4.1.3 Capteurs six-axes

Les capteurs six-axes industriels prennent en général la forme d'un volant à plusieurs branche, sur lesquelles des réseaux de résistances sont installés (figure 4.7).

Un tel capteur requiert l'usinage d'une pièce et un travail minutieux d'in-

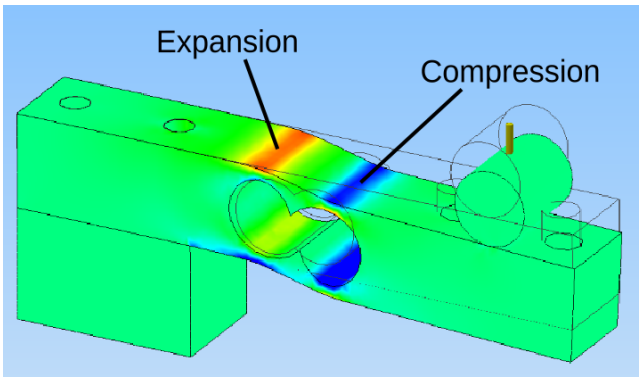


FIGURE 4.5 – Simulation en élément fini d'une forme mécanique utilisée de manière standard dans les mesures de forces. La déformation exagérée montre que la cavité créé un point d'inflexion qui provoque une expansion et une compression.

tégration. Il n'existe pas aujourd'hui de tel produit à bas coût. Une difficulté de ce système est la procédure de calibration. La méthode standard permettant de passer des mesures $V = (v_1, v_2, v_3, v_4, v_5, v_6)$ aux forces et moments $(F_x, F_y, F_z, M_x, M_y, M_z)$ est l'approximation linéaire, c'est à dire calibrer la matrice M telle que $F = MV$. Cette méthode nécessite d'appliquer des forces et des moments connus sur le capteur, et donc un banc de calibration. Traversaro *et al.* [2015] propose par exemple une méthode pour calibrer un tel capteur *in situ* en utilisant les données de l'accéléromètre.

4.1.4 Architecture proposée

Le capteur proposé se compose de quatre crampons par pied, attachés à des barres sur lesquelles sont collées des jauges de contraintes (*cf.* figure 4.1). Les contacts du pied avec le sol peuvent donc être résumés par quatre points, tout en gardant un polygone de sustentation similaire à un pied rectangulaire.

On pourrait défendre que cette configuration est plus proche de la voûte plantaire humaine, qui est en fait comparable à une matrice de multiples capteurs orientés dans la même direction.

Ces barres, avec les jauges de contraintes sont aujourd'hui des parties mécaniques standard, disponible à bas coût. Elles sont notamment utilisées dans les balances électroniques. Les résistances montées en pont de wheatstone fournissent un voltage faible qui a besoin d'être amplifié par un composant électronique. Là encore, des amplificateurs peu coûteux et performants sont disponibles.

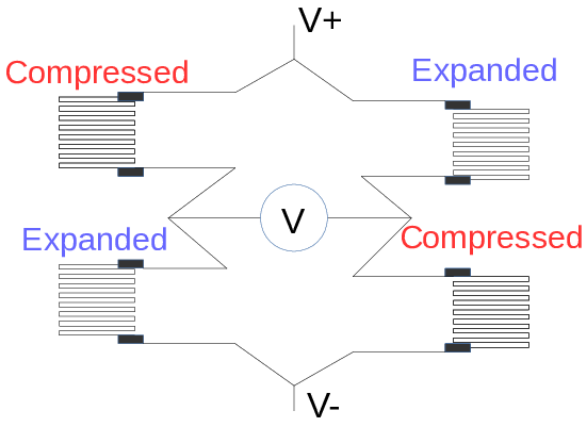


FIGURE 4.6 – Configuration en pont complet : quatre résistances très sensibles à la déformation sont collées sur la barre, deux de chaque côté. Ainsi, la différence de voltage est amplifiée.

Nous avons conçu une carte électronique contenant les amplificateurs, mais aussi un petit microcontrôleur qui permet d'agréger les informations. Ce dernier communique sur le même bus série que celui des servomoteurs utilisés dans le robot, en utilisant le même protocole de communication (Dynamixel). Cela permet de le connecter sur le dernier moteur, au niveau de la cheville du robot, et ainsi de faciliter l'intégration électronique (figure 4.8).

Sur notre carte, quatre amplificateurs (HX711) nous permettent d'atteindre une vitesse d'échantillonage de 80 Hz, avec 24 bits. Bien entendu, ces 24 bits ne sont pas tous utilisables, les derniers correspondant à du bruit.

La calibration de chaque jauge peut se faire simplement de manière indépendante des autres, en plaçant des masses connues dessus et en effectuant une régression linéaire. On peut éventuellement dissocier la tare de la jauge, c'est à dire sa valeur à vide de son coefficient linéaire. En effet, la tare est assez facile à recalculer, même *in situ*, car il suffit par exemple de soulever le robot au dessus du sol et de prendre des échantillons.

La capacité des jauges à reprendre leur position initiale, et donc à garder la même tare (ou résilience), est liée à la qualité des éléments mécaniques. On pourrait comparer le système à un ressort qui, avec le temps, se détend et a une longueur à vide plus longue.

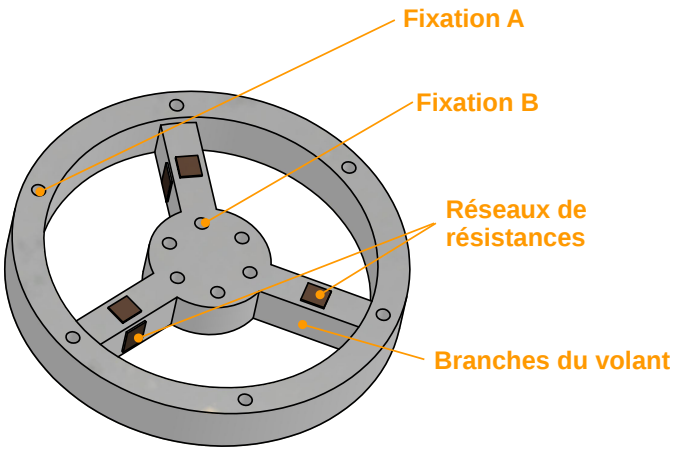


FIGURE 4.7 – Exemple de forme mécanique de capteur six-axes : un volant à trois branche. Des jauges sont collées sur deux plans des trois branches.

L'architecture de ces capteurs a été partagée en open-source avec la communauté (Rouxel *et al.* [2015])[2].

4.2 Modèle et calcul du centre de pression

Les six valeurs qui sont calculées par un capteur six-axes sont les forces (F_x, F_y, F_z) et les moments (M_x, M_y, M_z) qui sont appliqués au niveau de la cheville.

Cependant, comme il est remarqué dans Sardain et Bessonnet [2004] et dans Dekker [2009], les translations parallèles au sol (F_x, F_y) ainsi que le moment autour de l'axe z (M_z) sont en général annulés par les frictions avec le sol. Ceci est dû au fait que l'on suppose que le pied ne glisse pas sur le sol. En conséquence, bien que la mesure de (F_x, F_Y) et de (M_z) apporte des informations sur l'état du robot, elles ne sont en réalité pas utilisées dans le calcul du centre de pression.

Le pied est en contact avec le sol en n points de coordonnées Ji (voir figure 4.2). En considérant la structure mécanique des jauges, la force mesurée par une jauge correspond en fait à la composante verticale $\overrightarrow{Z_i} = \overrightarrow{z}[\overrightarrow{F_i}.\overrightarrow{z}]$ de la force $\overrightarrow{F_i}$ réellement appliquée en ce point.

Notons alors que :

2. https://github.com/Rhoban/ForceFoot/

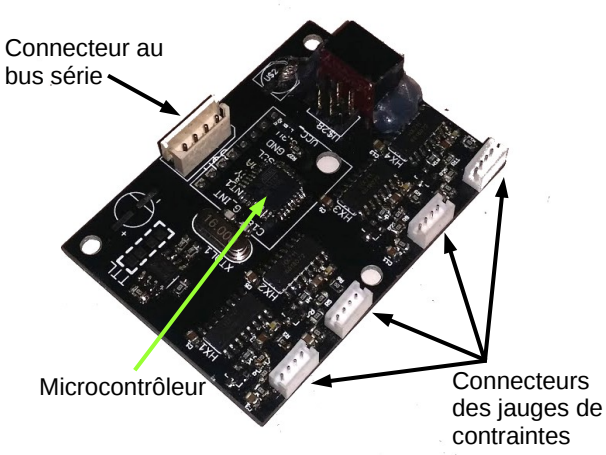


FIGURE 4.8 – La petite carte électronique placée dans le pied des robots.

$$F_z = \sum_i F_i$$

Est la composante verticale de la force appliquée sur le sol. Lorsque le robot est immobile, cette valeur correspond à $m.g$ et nous indique donc la masse du robot.

Le moment en tout point Q dans le plan du sol s'exprime (*cf.* Sardain et Bessonnet [2004], eq (3)) :

$$M_Q = \sum_i^n \left\| \overrightarrow{Z_i} \right\| \overrightarrow{QJ_i}$$

Où Q et J_i sont des vecteurs de coordonnés. Le centre de pression est le point P où les moments s'annulent, c'est à dire, en décomposant selon les axes x et y :

$$\sum_i^n \left\| \overrightarrow{Z_i} \right\| J_i = \sum_i^n \left\| \overrightarrow{Z_i} \right\| P$$

D'où :

$$P = \frac{\sum_i \left\| \overrightarrow{Z_i} \right\| J_i}{\sum_i \left\| \overrightarrow{Z_i} \right\|}$$

Géométriquement, il s'agit en fait du barycentre des coordonnées des jauges pondérées par l'amplitude des forces mesurées. Notons que ce point ne peut en

effet donc pas sortir de la surface du pied (ce qui est une propriété du centre de pression).

De plus si on suppose que les jauges sont toutes mécaniquement similaires (que leur coefficient linéaire est le même), le calcul du centre de pression est possible sans calibration à part la tare.

4.3 Expérimentation sur le robot Sigmaban

Le dispositif a été testé sur le petit robot humanoïde Sigmaban (Allali *et al.* [2016], figure 4.9). Ce dernier mesure environ 55 cm de haut, et a été utilisé pour participer à la RoboCup dans la catégorie Kid Size, une compétition dans laquelle il devait jouer au football sur un terrain réduit sur du gazon synthétique d'environ 3 cm de haut.

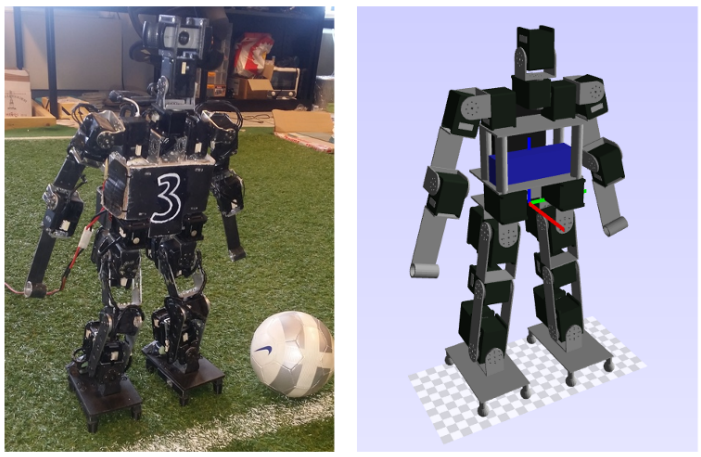


FIGURE 4.9 – Le robot humanoïde Sigmaban ainsi que son modèle dynamique complet.

4.3.1 Définition du pied de support

Une première application de ce type de capteur est de définir le pied de support effectif à chaque instant. Cette information nous permet notamment d'améliorer le modèle odométrique du robot.

C'est d'ailleurs un problème que nous avons étudié. Pour ce faire, l'ensemble des capteurs du robot permettent d'estimer son état, à partir de la position des moteurs et de l'orientation donnée par la centrale inertielle. Le lacet est estimé à partir de l'intégration du gyromètre en Z. Les capteurs de

pressions permettent de placer le pied de support courant au sol. Lorsque le pied de support change, le modèle cinématique du robot permet de déterminer le déplacement qui a été fait dans le repère du monde. Ces mesures sont imprécises et doivent être corrigées pour correspondre le mieux possible à la réalité (Rouxel *et al.* [Accepted]).

4.3.2 Comparaison avec le modèle

Nous avons créé le modèle le plus fidèle possible du robot, d'un point de vue cinématique et dynamique (figure 4.9). Ce modèle nous permet donc d'estimer la position du centre de masse. Pour cela, la lecture des encodeurs des moteurs permet de connaître la posture du robot (des pièces relativement les unes aux autres), et ainsi de calculer la position du centre de masse du robot, connaissant la masse de chacune des pièces, dans un repère choisi comme par exemple celui d'un des pieds.

Nous avons alors comparé ce résultat avec l'estimation de la position du centre de pression donné par le capteur, durant un mouvement très lent, au cours duquel la cheville recevait des ordres en position en suivant un sinus (statiquement, le projeté du centre de masse au sol coïncide avec le centre de pression).

Les courbes tracées en figure 4.10 présentent cette expérience, et nous donnent une idée sur la précision des valeurs calculées à l'aide de notre capteur.

4.3.3 Utilisation pendant la marche

La figure 4.11 nous montre les valeurs mesurées par le capteur sous les pieds du robot Sigmaban (Fabre *et al.* [2016]) au cours d'un cycle de marche. Comme on peut le constater, l'estimation du centre de pression latérale (y) nous indique clairement le pied de support du robot. Egalement, l'estimation de la force verticale (F_z) présente les chocs du pied avec le sol au moment du changement de support.

4.3.4 Stabilisation de la marche

Aujourd'hui, un grand nombre des contrôleurs présentés permettant de faire marcher les robots humanoïdes sont en boucle ouverte. La plupart d'entre eux sont d'ailleurs basés sur le critère du ZMP, calculant par exemple hors-ligne une trajectoire à suivre pour les moteurs par dynamique inverse.

De même, les robots de la RoboCup utilisent des contrôleurs essentiellement en boucle ouverte qui impliquent le réglage manuel et perpetuel de nombreux

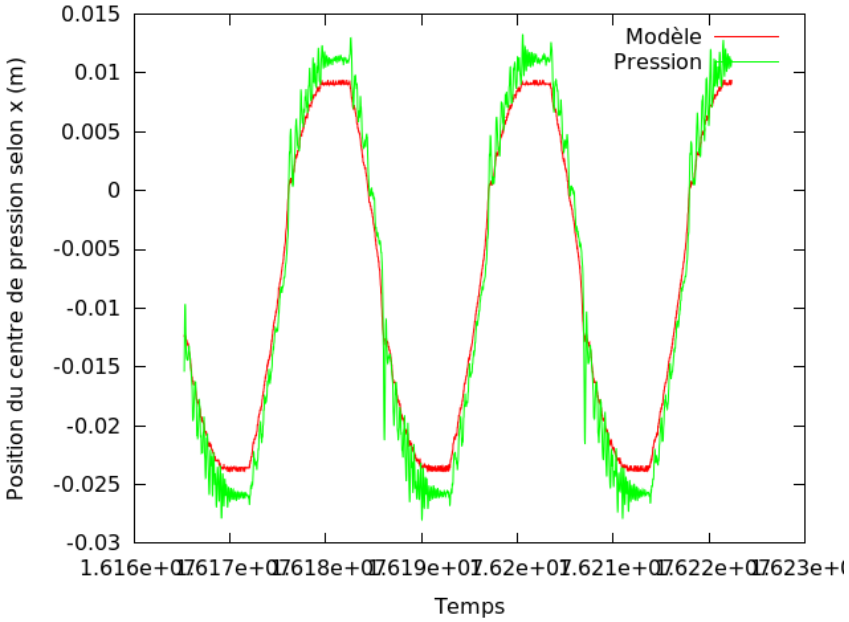


FIGURE 4.10 – Tracé comparatif du centre de masse selon l'axe x donné par le modèle et calculé avec le capteur de pression.

paramètres. Ces contrôleurs reposent sur une stabilité intrinsèque du robot et des ajustement experts, rendus possibles entre autre par l'aspect bas coût du matériel.

Notre contrôleur (présenté dans Rouxel *et al.* [2015]) est basé sur des splines dans le monde cartésien. Il est assez similaire à celui présenté dans le chapitre précédent, avec du swing qui est ici essentiel pour transférer le support du robot d'un pied à l'autre. La différence avec le contrôleur des robots à pattes est l'utilisation des trois degrés de libertés supplémentaires pour garder le pied parallèle au sol pendant la marche. De plus, la rotation de la hanche est utilisée pour tourner, le centre de masse du robot étant presque aligné avec le pied lors du support.

La présence des CPG dans les neurones suggère que la boucle ouverte existe dans la nature, et que la cyclicité de la marche d'un animal n'est pas auto-entretenue par une série d'actions et de réactions avec son environnement. L'expérience qui a démontré cela était la comparaison des ordres produits par les neurones pour les muscles *in vivo* et par le système nerveux *in vitro*, plongé dans une solution saline (Marder et Bucher [2001]). Dans ce cas se pose la question de la stabilisation, ou comment fermer la boucle pour obtenir une marche plus stable, voire réagir à des perturbations extérieures.

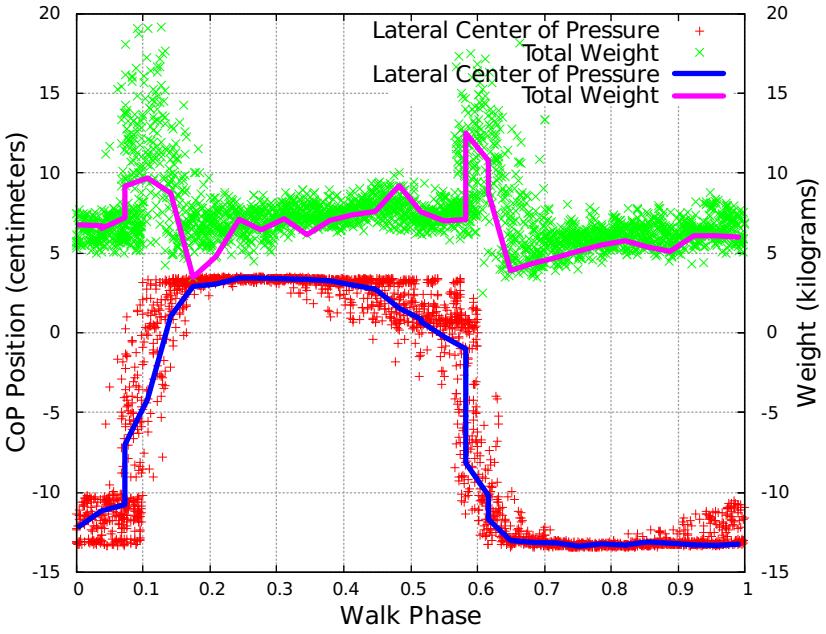


FIGURE 4.11 – Schéma des valeurs prises par le centre de pression latéral (y) et le poids total du robot (F_z) pendant plusieurs cycles de marche, en fonction de l'affixe dans le cycle (entre 0 et 1).

Ces perturbations peuvent faire sortir le robot du bassin attracteur sur lequel la stabilité intrinsèque repose et peuvent provoquer sa chute. Une d'entre elles est la perturbation latérale du robot, qui le pousse à rouler sur le côté du pied (figure 4.13). Dans cette situation, le robot est en déséquilibre et ses possibilités d'action sont réduites. Si cette perturbation est trop forte, un mouvement spécifique et complexe pourrait être nécessaire pour essayer de garder l'équilibre, un humain va typiquement croiser les jambes ou faire un petit saut. Sur un robot, on peut essayer de tomber avec le moins de dégâts possibles (Ruiz-del Solar *et al.* [2009]). En revanche, il est aussi possible que cette perturbation soit plus légère. On pourrait par exemple comparer le robot sur la figure 4.13 à une boîte qui aurait reçu une impulsion, et qui peut soit basculer, soit se soulever avant de retomber dans son état original.

Une expérience standard est d'ailleurs en général d'attacher une masse au bout d'une corde et de la lâcher sur le robot. Si un objet de masse m_a est lâché à la hauteur h_a, comme présenté sur la figure 4.12, si on suppose que l'énergie est conservée pendant le choc, le robot roulera autour du pied opposé. Si son centre de masse dépasse la verticale, il retombera de l'autre côté. Dans le cas contraire il retombera sur ses pieds. La condition pour que le robot ne tombe

pas peut alors s'exprimer en terme d'énergie :

$$(E_r = \frac{1}{2} m_r g \Delta h_r) < (E_a = \frac{1}{2} m_a g \Delta h_a)$$

Où E_r est l'énergie nécessaire au robot pour basculer (c'est à dire au pendule inverse pour atteindre le point de basculement) et E_a l'énergie fournie par le choc inélastique. Autrement dit :

$$m_r \Delta h_r < m_a \Delta h_a$$

En réalité, Δh_r peut s'exprimer géométriquement :

$$\Delta h_r = \sqrt{d^2 + h^2} - h$$

Où d est l'écart latéral entre le centre de masse du robot et h la hauteur du centre de masse (figure 4.12).

En faisant l'expérience dans la pratique, ce calcul n'est en réalité pas juste. La valeur ci-dessus n'est en effet qu'une borne inférieure de l'énergie nécessaire pour faire tomber le robot.

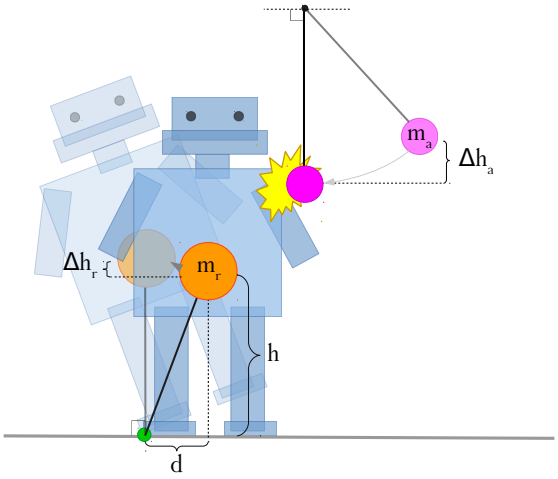


FIGURE 4.12 – Modèle d'une expérience de test de stabilité, un objet accroché au bout d'une corde est lâché sur le robot.

Si la perturbation permet au robot de retomber sur ses pieds, il subsiste en revanche un problème si ce dernier fonctionne en pure boucle ouverte. En effet, le pied sur lequel le robot roule doit rester plus longtemps au sol. Un problème typique observé sur ces robots après une telle perturbation est que, poursuivant le mouvement périodique de la boucle ouverte, le robot essaie de lever le pied sur lequel il roule, retombant par la suite sur le pied opposé qu'il

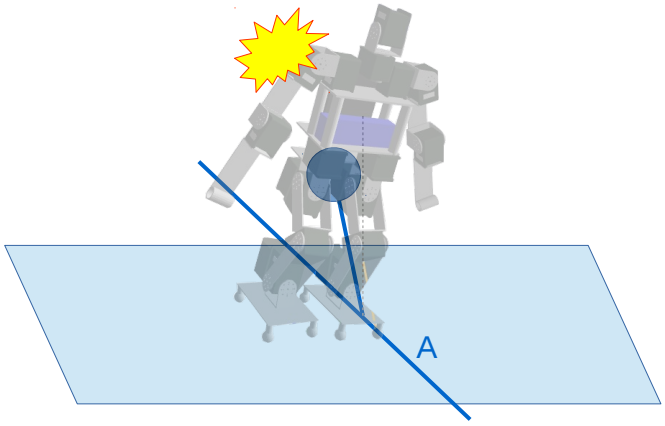


FIGURE 4.13 – Lorsque le robot est perturbé, il roule autour de l'arête d'un pied pendant un instant. Il perd temporairement un degré de liberté : ici, il ne peut plus appliquer de couple autour de l'axe A. Le contrôleur en boucle ouverte ne prend pas en compte cet état et risque d'effectuer des actions non souhaitables comme par exemple lever la jambe de support qui n'a pas pu changer.

essaiera à son tour de lever au mauvais moment, provoquant alors une oscillation qui entraînera une chute qui aurait pu être évitée (figure 4.14).

Ce comportement caractéristique est assez remarquable, car le robot se retrouve dans un état particulier qui est identifiable plusieurs pas avant de tomber.

Une manière de réduire ces instabilités latérales est d'introduire une phase de double support, qui permet de s'assurer naturellement que le robot retrouvera un équilibre latéral avant de changer de pied de support. Chez l'homme, cette phase de double support existe et cette phase de double support est d'environ 20% du cycle de marche.

Cependant, cette solution ralentit également la marche, pour les mêmes raisons que celles évoquées dans le chapitre précédent.

Graf et Röfer [2010], Missura et Behnke [2011] et Alcaraz-Jiménez *et al.* [2013] proposent de réaliser du *capture step* sur le robot. Ce dernier est approximé par un pendule (présenté dans Kajita *et al.* [2001]) et des équations analytiques donnent alors une trajectoire attendue du centre de masse. De cette manière, l'instant et l'endroit où le changement de pied aura lieu peut être prédit et le comportement en boucle ouverte peut être corrigé. L'intérêt de ces

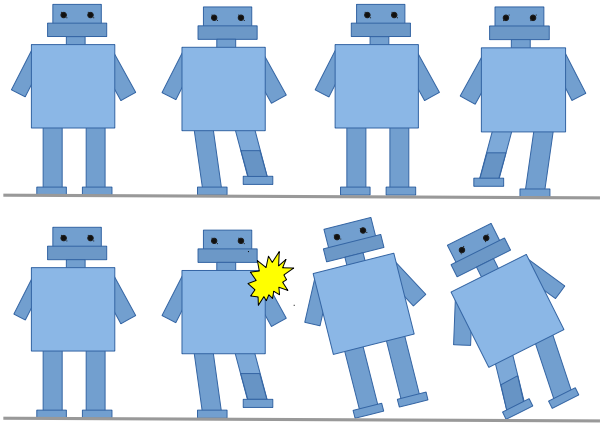


FIGURE 4.14 – Le robot lève les jambes successivement lorsqu'il marche en boucle ouverte. En cas de perturbation, un comportement risqué qui peut alors se produire est de lever la jambe sur laquelle le robot est effectivement supporté.

méthodes est qu'elles ne nécessitent pas d'aménagement sur les robots autres que les encodeurs des moteurs et la centrale inertielle, qui permettent d'estimer l'état du pendule qui est utilisé comme modèle. Un avantage de cette méthode est qu'il n'est pas nécessaire d'estimer l'accélération du centre de masse, mais sa position et sa vitesse, qui sont alors comparées à une trajectoire nominale. En revanche, les auteurs précisent qu'il faut malgré tout ajuster correctement le modèle dynamique du pendule afin d'obtenir des bonnes prédictions. Un autre problème est d'essayer de filtrer correctement le bruit et les délais liés à la mesure des capteurs, afin d'estimer position et vitesse du centre de masse.

Notre approche repose sur une verification de l'état du robot à certaines phases du mouvement de marche. En particulier, le moment où le robot transfère son poids d'un pied à l'autre (échange de support). C'est précisément à cet instant que le capteur de pression nous fournit le plus d'informations selon l'axe latéral. En réglant cet instant et un seuil, il est possible de s'assurer que cet échange de support a bien eu lieu et que le mouvement en boucle ouverte n'est pas perturbé. Dans le cas contraire, le mouvement est mis en pause. Cette pause, qui est en général très brève, évite au mouvement de balancier artificiel que le robot crée en appliquant des couples sur le sol d'être déphasé avec le mouvement naturel du pendule qui apparaît lorsqu'il roule sur le pied, et lui permet de reprendre son cycle de marche dès qu'il se rétablit (figure 4.16).

Nous avons monté un banc de test dans lequel une masse de 1 kg était attachée au bout d'une corde d'environ 1.9 m. Nous avons alors lâché cette

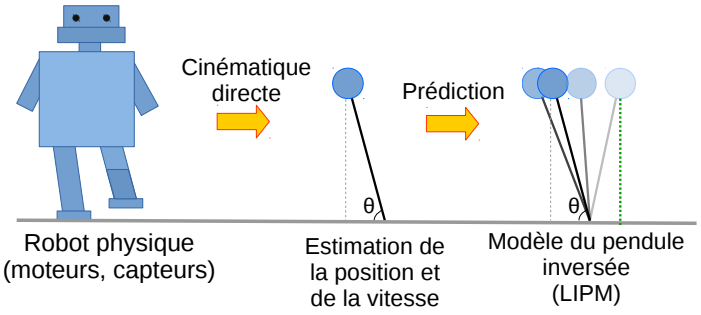


FIGURE 4.15 – Principe standard du *capture step*, les mesures des capteurs de position et de la centrale inertielle sont utilisées pour calculer la position et la vitesse du centre de masse du robot, et donc réaliser une prédiction sur le comportement du modèle du pendule.

masse sur le robot, en prenant soin qu'elle le perturbe latéralement pendant la marche (figure 4.17). Nous avons fait 20 essais avec et 20 essais sans le système de stabilisation (en pure boucle ouverte) en nous plaçant successivement à 50 puis à 60 cm du robot. Voici le nombre de chutes obtenues :

	Stabilisation activée		Stabilisation désactivée	
	Chute	Non-chute	Chute	Non-chute
50 cm	1	19	15	5
60 cm	9	11	14	6

En réalité, la phase de la marche au moment de l'impact semble avoir une importance sur la capacité du robot à se stabiliser. Quand la stabilisation est désactivée, le comportement du robot est caractéristique et il est facile de distinguer le cas d'une chute inévitable et franche du cas de la perturbation mal gérée par l'aspect boucle ouverte du mouvement.

Cette technique peut être vue en action dans la vidéo : https://youtu.be/avJI_cBuMm0.

Un des avantages de cette méthode est sa simplicité de mise en oeuvre à bord des robots, car elle est basée sur une mesure directe du centre de pression qui est au coeur de la détection des problèmes de stabilité. De plus, elle ne requiert pas d'estimer l'état complet du robot, c'est à dire de la position et de la vitesse du centre de masse, éliminant les difficultés liées à l'élimination du bruit dans ce dernier.

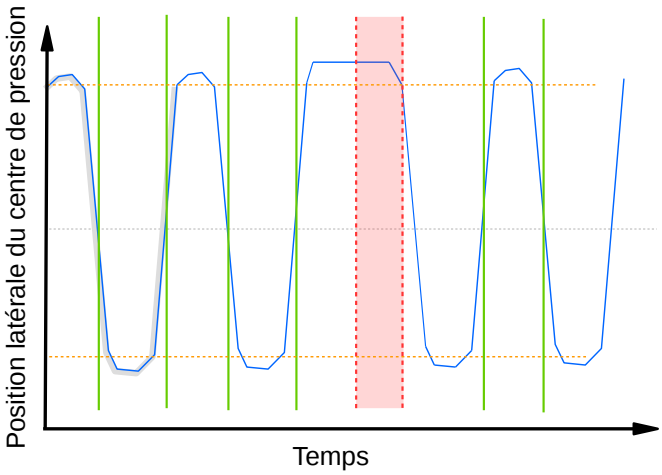


FIGURE 4.16 – Méthode de détection et de correction des instabilités de la marche dans le plan latéral. L'estimation du centre de pression latéral est la courbe bleue, les barres vertes sont les instants auxquels la vérification que le centre de pression est bien en dessous du seuil (en jaune). Si il ne l'est pas, le mouvement est mis en pause.

4.4 Cycle nominal

La méthode que nous avons présentée précédemment fonctionne bien dans la pratique, essentiellement grâce à sa simplicité, qui est due à la présence des capteurs de pression qui mesurent directement l'information importante nous permettant de détecter les perturbations.

Ce principe peut être généralisé. En effet, la méthode utilisée ici est une comparaison des capteurs avec un profil nominale, qui est observé dans le cas où le mouvement est stable et où le robot ne tombe pas. Cette idée a déjà été présentée dans Pastor *et al.* [2012], qui est appellée *associative skill memory*, ou mémoire associative d'habileté.

4.4.1 Profilage des capteurs

Nous présentons ici une approche plus générale, dans laquelle le profil des capteurs est appris statistiquement. Un ensemble de N capteurs représente l'état Q du robot. Nous appellerons t l'affixe du mouvement. Cet affixe est d'abord discrétisé en c classes $t_1, t_2, t_3 ... t_c$. L'état du robot mesuré à un affixe t_i peut être représenté par une distribution normale multidimensionelle $\mathcal{N}(\mu(t_i), \Sigma(t_i))$, avec comme paramètres la moyenne $\mu(t_i)$ et la matrice de covariance $\Sigma(t_i)$.

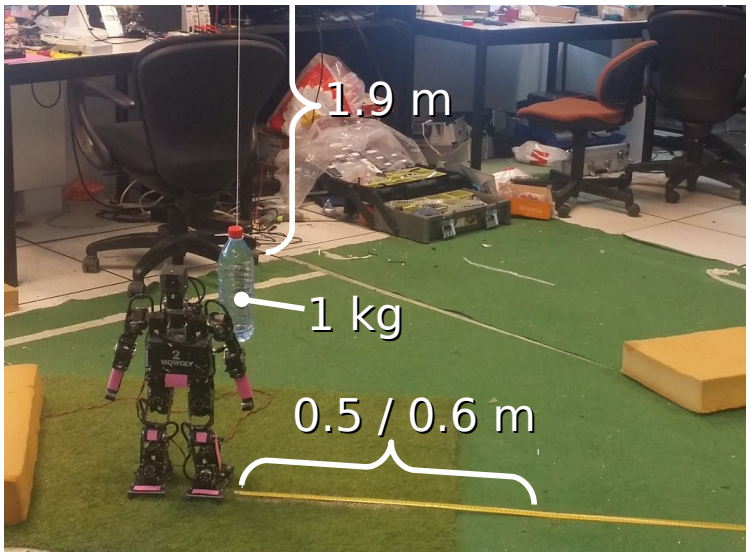


FIGURE 4.17 – Présentation du banc de test utilisé pour tester la stabilisation latérale du robot.

Ces paramètres peuvent être estimés empiriquement à partir d'un ensemble d'échantillons :

$$\mu(t_i) = \frac{1}{n_i} \sum_k Q_{i_k}$$

$$\Sigma(t_i) = \frac{1}{n_i - 1} \sum_k Q_{i_k} Q_{i_k}^T$$

Où Q_{i_k} est un des n_i échantillons observés dans la classe t_i. Ainsi, pendant l'exécution du mouvement, il est possible de tester la plausibilité que l'état Q_r soit correct à un affixe t_r donné à l'aide du test du χ^2. En effet, la valeur :

$$f(Q_r, t_r) = (Q_r - \mu(t_r))^T \Sigma(t_r)^{-1} (Q_r - \mu(t_r))$$

Suit une distribution de χ^2 à N degrés de libertés. La plausibilité peut donc se calculer par :

$$P(Q_r, t_r) = F_{\chi^2_N}(f(Q_r, t_r))$$

Où $F_{\chi^2_N}$ désigne la fonction de répartition de la loi du χ^2 à N degrés de liberté. Il est également possible de proposer l'affixe ayant le meilleur score au test, et donc une phase du mouvement qui ressemble plus à l'état effectif courant du robot :

$$t_m(Q_r) = \arg\min_{t_i} f(Q_r, t_i)$$

4.4.2 Méthode

La méthode que nous proposons se déroule donc en deux temps, tout d'abord la détection de la perturbation. Cette dernière se fait par le test du χ^2, et requiert un seuil qui désigne la tolérance par rapport aux échantillons enregistrés.

Il est alors possible de proposer l'affixe le plus plausible, et de modifier le déroulement du mouvement pour l'adapter à l'état effectif observé sur le robot (figure 4.18).

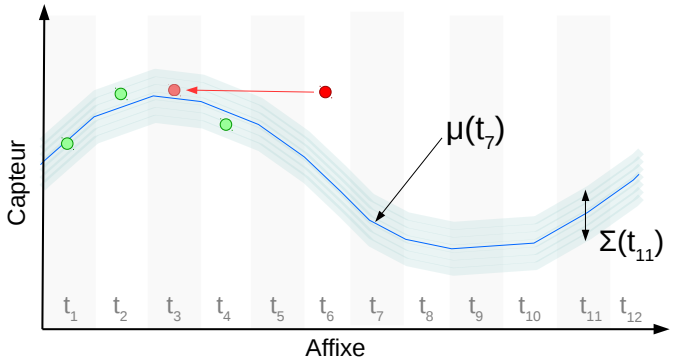


FIGURE 4.18 – Présentation de la méthode de suivi de profil nominal sur un capteur, les colonnes représentent les classes d'affixe discrétisées, la valeur présentée en rouge est peu plausible compte tenu du profil nominal connu, un meilleur candidat d'affixe peut être proposé.

4.4.3 Mise en oeuvre

Nous avons mis en oeuvre cette méthode et l'avons testé sur le robot Sigmaban. Nous avons choisi comme capteurs quatre dimensions, l'estimation du centre de pression latérale, le gyromètre en roulis, l'accéléromètre latéral et l'estimation du roulis filtré par la centrale inertielle. Nous avons créé l'estimation statistiques en prenant des échantillons de ces capteurs à bord du robot, en enregistrant l'affixe de la marche et leurs valeurs en pilotant le robot à la manette pendant sa marche, en pure boucle ouverte. L'affixe a été discrétisé en 100 classes.

La méthode précédente a alors été testée sur le robot. Dans le cas où le test du χ^2 révélait une perturbation, nous avons modifié l'affixe du mouvement pour qu'il corresponde à celui qui était, d'après ce test le plus plausible. Afin d'éviter les possibles sauts d'affixe, et d'utiliser le contexte (la détermination de l'affixe la plus plausible étant acontextuelle), nous avons limité la variation qui peut s'appliquer sur l'affixe (qui est un paramètre de la méthode).

Ce travail est toujours en cours, mais des premiers résultats ont révélés des comportements plus originaux, le robot étant capable d'adapter son allure à tout moment.

Un test est par exemple de baisser la fréquence de marche robot (à environ 1 Hz). Dans ce cas, les mouvements du bassin sont plus amples et plus lents, et le robot roule sur le côté du pied quasiment à chaque pas, nécessitant un ajustement permanent. En pure boucle ouverte, tout comme avec la méthode experte basée sur des seuils manuels, le robot est très instable (il est d'ailleurs nécessaire de l'aider à démarrer), le bassin attracteur est plus petit. Avec la méthode basée sur le suivi du cycle nominal, le robot est stable et résiste même à des petites perturbations.

Cette technique est toujours en cours de développement, les résultats préliminaires peuvent être observés en vidéo : https://youtu.be/Ja5ANewZnwA

4.4.4 Avantages et limites

Cette méthode est pûrement statistique et a pour avantages d'être indépendante de tout modèle. Elle est également complètement boîte noire et pourrait s'appliquer à d'autres mouvements. Son fonctionnement intuitif est de replacer un mouvement en boucle ouverte perturbé dans son bassin attracteur.

Elle requiert peu de réglages, puisque le principal paramètre est le seuil de détéction des anomalies, et éventuellement des paramètres permettant de filtrer les variations d'affixe comme expliqué dans le cas de Sigmaban.

Cependant, il est nécessaire de faire l'acquisition de données pour construire le modèle statistique. De plus, cette méthode ne créera pas de mouvement original, puisqu'elle ne fait qu'ajuster la phase d'un mouvement.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un capteur basé sur des composants standard et bas coût permettant d'estimer les forces de réactions avec le sol

plusieurs points sous les pieds du robot.

Ce capteur a été construit et testé sous les pieds d'un petit robot humanoïde, fournissant alors des informations supplémentaires sur l'état du robot qui peuvent être utilisées pour déterminer le pied de support ou estimer la position du centre de pression.

De plus, nous avons proposé une méthode qui permet d'améliorer la stabilité latérale du robot, qui a elle aussi été testée au cours d'expériences, et pendant la compétition de la RoboCup 2015 et 2016 (Allali *et al.* [2016]). Cette méthode qui se base sur l'estimation du centre de pression latérale, est basée sur une mesure qui permet de détecter les phases dans lesquelles le robot roule sur le pied, atteignant des positions dangereuses pour son équilibre au cours du mouvement périodique de la marche.

Nous avons également proposé une méthode plus générale basée sur le suivi du profil nominal des capteurs du robot dont le but est de détecter les perturbations mais aussi de réagir, en essayant de ramener le robot dans le bassin attracteur intrinsèquement stable du mouvement en boucle ouverte.

Perspectives

4.6 Robots paramétriques

4.6.1 Famille de robots

L'ambition du travail sur les robots paramétriques est la synthèse rapide d'une locomotion sur un robot quelconque. Au cours de cette thèse, nous avons proposé un contrôleur et, après une série d'expérience avons proposé une méthode afin de déterminer rapidement des allures pour un robot donné à partir de simulations géométriques.

Cependant, nous n'avons traîté que d'une famille particulière de robots, et certains problèmes restent encore non résolus dans le but de permettre à un utilisateur non expert de créer un robot capable de marcher.

Les ambiguïtés cinématiques sont une première question, qui n'est pas abordée dans cette thèse. Comme expliqué, trois degrés de libertés permettent d'atteindre des positions (x, y, z) dans une partie de l'espace opérationnel, l'ajout de degrés de libertés crée donc un problème sous-contraint. La réponse qui nous semble la plus naturelle est d'imposer des contraintes supplémentaires, dont notamment l'orientation du bout de la patte par rapport au sol. Cette approche permet de tirer profit des degrés de libertés supplémentaires pour, par exemple, garder un pied plat sur le sol et garantir une meilleure adhérence ainsi qu'une meilleure surface de sustentation. C'est d'ailleurs ce que nous faisons sur le robot Sigmaban. Cela revient à imposer aussi le tangage, roulis et lacet du bout du pied (α, β, γ).

Une approche standard de ce genre de problème est l'utilisation d'un algorithme tel que Levenberg-Marquardt (Moré [1978]), car les équations cinématiques du robot sont connues, mais surtout dérivables, ce qui permet de converger plus vite.

De plus, dans cette thèse, nous avons considéré les robots comme des "étoiles", où chaque patte était une chaîne cinématique rattachée au corps.

Cela ne permet pas de représenter par exemple un robot avec une colonne vertébrale.

4.6.2 Découverte embarquée de morphologie

Un autre problème connexe à celui de la locomotion est celui de la découverte automatique du robot de sa propre morphologie à bord.

Bongard *et al.* [2006] présente un robot quadrupède qui est continuellement en train de se modéliser lui-même. Parfois, il réalise des actions aléatoires et tente de trouver des modèles qui expliquent les informations qu'il récupère depuis ses capteurs.

Cully *et al.* [2015] a proposé une méthode sans modèle dont le but est d'adapter la locomotion du robot à partir d'une cartographie performance/-confiance qui permet de mettre en relation les différentes manières de se déplacer avec un indice de confiance qui évolue au cours du temps, et permet donc de s'adapter en cas de modifications mécanique.

La découverte de la morphologie du robot est une brique intéréssante qui permettrait de faire marcher un robot sans même connaître son modèle à l'avance.

4.6.3 Caractère holonome

Un autre aspect qui peut être intéréssant d'étudier sur ce type de robots est leur caractère holonome. Dans nos expériences, notamment celles des points de contrôle, nous avons piloté les robots en les faisant tourner et avancer droit vers leur cible.

Une autre propriété qu'il serait intéréssant de vérifier est le caractère holonome des robots, ou leur capacité à avancer selon un vecteur quelconque. Bien que notre contrôleur le permette techniquement, cela n'a pas été évalué en tant que critère (dans aucune expérience les robots ne marchaient selon des vecteurs différents que celui de leur "avant" déterminé).

Dans la plupart des trajectoires, les animaux et les humains avancent selon un avant prédéfini, qui est lié à la morphologie de leurs jambes, mais aussi à leurs mouvements de tête. Mombaur *et al.* [2010] a par exemple proposé des critères pour essayer d'expliquer le comportement des humains pendant leurs déplacements.

Cependant, pour atteindre des cibles proches, ou éviter des obstacles, la locomotion holonome est utile. Il est par exemple difficile de jouer au foot avec un robot à deux roues, car des petits mouvements latéraux peuvent s'avérer très pratiques pour se réaligner avec une balle. De plus, l'holonomie peut être appréciée dans certaines tâches, comme par exemple tourner autour d'une cible tout en maintenant une caméra face à elle.

Le test que nous avons effectué en changeant l'orientation du vecteur de direction du robot lors des simulations géométriques nous a par exemple suggéré qu'un robot pentapode pourrait avoir besoin de changer d'allure selon sa direction pour rester le plus stable possible.

Une expérience pourrait donc être conçue pour essayer de tester à quel point un robot à pattes est capable de suivre un vecteur de translation arbitraire, et donc évaluer son holonomie.

4.6.4 Environnement open-source

L'environnement (Metabot Studio) a également pour but de permettre à d'autre contributeurs de pouvoir par exemple tester leurs algorithmes sur une famille de robots paramétriques en simulation.

Ce dernier est déjà open-source, bien qu'actuellement peu documenté. Il a déjà été utilisé au cours d'enseignement en découverte de la robotique avec des élèves-ingénieurs, afin de leur proposer de s'initier à la création de mouvements sur des robots à pattes. Le but de cet enseignement était de choisir un robot parmi plusieurs proposés et d'écrire le code qui permet de le faire marcher, permettant d'introduire des notions telles que le modèle géométrique inverse, les splines ou les CPG.

Un tel environnement pourrait être utile à des expériences dans lesquelles des algorithmes, par exemple génétiques ou d'apprentissage seraient évalués sur un corpus de robot.

4.7 Cycle nominal

Nos travaux sur la stabilisation de la marche qui correspondent à l'établissement d'un cycle nominal de référence appris statistiquement sont le début d'une piste permettant d'atteindre une marche stable et robuste à partir d'un contrôleur en boucle ouverte.

Cette méthode souffre néanmoins pourrait être améliorée, tout d'abord en

améliorant la manière dont les informations sont filtrées, pour éviter les accoups liés au bruit des capteurs.

De plus, une amélioration possible serait de parvenir à réaliser un apprentissage en ligne, qui permettrait d'ajuster la trajectoire nominale, et qui permettrait une meilleure adaptation par exemple aux changements de paramètres tels que la posture du robot pendant la marche.

Enfin, la même méthode pourrait être testée sur d'autres mouvements, tels que le relevage ou le tir, afin d'essayer d'établir si le mouvement s'est bien passé (par exemple, le pied a t-il touché la balle ? ou encore le robot s'est-il bien relevé ?), mais aussi réagir.

4.8 Synthèse de mouvements robustes

Un sujet dont nous n'avons pas discuté dans cette thèse, mais qui fait partie de nos préoccupations, essentiellement dans le cadre de la RoboCup, est le fait de créer des mouvements robustes. Les mouvements robustes, malgré le fait d'être en boucle ouverte, ont une stabilité intrinsèque qui les rend plus adéquats.

Un exemple, à la fois parlant et pragmatique, est le relevage du robot Sigmaban, qui consiste à passer d'une position couchée (sur le ventre ou le dos) à debout, par une série de déplacement des actuateurs. Ces mouvements sont essentiels pour permettre au robot de reprendre sa locomotion après une chute.

Nous avons développé de tels mouvements sur le robot Sigmaban, en modifiant manuellement des points de contrôles dans des splines. Dans la pratique, ces mouvements demandent des grands déplacements dans l'espace articulaire, une des difficultés étant d'éviter les auto-collisions du robot. Avec de nombreuses heures investies, nous avons obtenu un mouvement rapide et efficace [3].

Nous avons également synthétisé ce mouvement, en utilisant un modèle dynamique du robot et en réalisant une optimisation du mouvement directement sur les mêmes splines à l'aide d'une fonction de score indiquant à quelle hauteur le robot a réussi à amener sa tête [4].

Une première remarque est que la méthode utilisée pour l'optimisation automatique est très naïve, comparée à l'approche experte. Un opérateur humain

3. https://youtu.be/6rsLzqLDM8w, mouvement de relevage créé manuellement
4. https://youtu.be/9SrHuwC3-3Y, mouvement de relevage découvert entièrement en simulation.

qui conçoit ce mouvement va fonctionner par "étapes", en amenant d'abord les bras à toucher le sol, puis en levant le corps, en déplaçant la masse au dessus des pieds et en relevant le buste. Ces *a priori* ne sont pas pris en compte dans l'optimisation brute de la spline. L'espace de recherche est donc gigantesque avec cette méthode et il faut de nombreuses heures de calcul pour obtenir une convergence vers un mouvement cohérent. Il existe également des corrélations claires pour un opérateur humain. Comment implémenter un "guidage" humain, dans l'optimisation de ce type de mouvements ?

D'un autre côté, nous pensons que le mouvement que nous avons généré est également moins robuste que le mouvement créé manuellement. Un exemple de robustesse est la robustesse au temps, la version manuelle du mouvement peut être jouée plus ou moins vite, menant au même résultat. Un autre exemple est la tolérance aux changements du modèle, comme par exemple le changement de la batterie qui modifie la masse du torse. Comment générer des mouvements intrinsèquement robustes ?

Notre environnement (Metabot Studio), qui permet de traiter de robots paramétriques, nous a permis de réaliser un modèle complet de Sigmaban. Une possibilité intéréssante serait donc d'injecter volontairement des petites erreurs sur son modèle, afin d'évaluer la robustesse d'un mouvement de synthèse. Nous pensons que ce genre d'approche peut également permettre de réduire le biais de la simulation, proposant des mouvements plus faciles à transporter dans la réalité (par exemple, le mouvement créé manuellement fonctionne très bien en simulation).

Bibliographie

Alcaraz-Jiménez, Juan José, Missura, Marcell, Martínez-Barberá, Humberto et Behnke, Sven, 2013. Lateral disturbance rejection for the nao robot. Dans *RoboCup 2012 : Robot Soccer World Cup XVI*, pages 1–12. Springer.

Alexander, R McN, 1984. The gaits of bipedal and quadrupedal animals. *The International Journal of Robotics Research*, 3(2) :49–59.

Alexander, RMcN et Jayes, AS, 1983. A dynamic similarity hypothesis for the gaits of quadrupedal mammals. *Journal of zoology*, 201(1) :135–152.

Allali, Julien, Deguillaume, Louis, Fabre, Rémi, Gondry, Loic, Hofer, Ludovic, Ly, Olivier, N'Guyen, Steve, Passault, Grégoire, Pirrone, Antoine et Rouxel, Quentin, 2016. Rhoban football club : Robocup humanoid kid-size 2016 champion team paper. Dans *RoboCup 2016 : Robot Soccer World Cup XX*. Springer.

Altendorfer, Richard, Moore, Ned, Komsuoglu, Haldun, Buehler, Martin, Brown Jr, H Benjamin, McMordie, Dave, Saranli, Uluc, Full, Robert et Koditschek, Daniel E, 2001. Rhex : a biologically inspired hexapod runner. *Autonomous Robots*, 11(3) :207–213.

Angle, Colin, 1989. *Genghis, a six legged autonomous walking robot.* Thèse de doctorat, Massachusetts Institute of Technology.

Auerbach, Joshua, Aydin, Deniz, Maesani, Andrea, Kornatowski, Przemyslaw, Cieslewski, Titus, Heitz, Grégoire, Fernando, Pradeep, Loshchilov, Ilya, Daler, Ludovic et Floreano, Dario, 2014. Robogen : Robot generation through artificial evolution. Dans *Artificial Life 14 : Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, EPFL-CONF-200995, pages 136–137. The MIT Press.

Baillie, J-C, 2005. Urbi : Towards a universal robotic low-level programming language. Dans *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 820–825. IEEE.

Baltazar, P., de la Hogue, T. et Desainte-Catherine, M., 2014. *i-score, an Interactive Sequencer for the Intermedia Arts*. Athena (Greece).

Benazera, Emmanuel, 2014. libcmaes : Multithreaded c++11 implementation of cma-es family for optimization of nonlinear non-convex blackbox functions. https://github.com/beniz/libcmaes/.

Berman, Barry, 2012. 3-d printing : The new industrial revolution. *Business horizons*, 55(2) :155–162.

Blickhan, Reinhard et Full, RJ, 1993. Similarity in multilegged locomotion : bouncing like a monopode. *Journal of Comparative Physiology A*, 173(5) :509–517.

Bongard, Josh, Zykov, Victor et Lipson, Hod, 2006. Resilient machines through continuous self-modeling. *Science*, 314(5802) :1118–1121.

Brooks, Rodney A, 1989. A robot that walks ; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2) :253–262.

Brooks, Rodney A et Flynn, Anita M, 1989. Fast, cheap and out of control. Rapport technique, DTIC Document.

Catto, Erin, 2005. Iterative dynamics with temporal coherence. Dans *Game Developer Conference*, tome 2, page 5.

Coulom, Rémi, 2011. Clop : Confident local optimization for noisy black-box parameter tuning. Dans *Advances in Computer Games*, pages 146–157. Springer.

Coumans, Erwin, 2003. Bullet : Real-time physics simulation. http://www.bulletphysics.org/.

Coumans, Erwin *et al.*, 2013. Bullet physics library. *Open source : bulletphysics. org*, 15.

Cully, Antoine, Clune, Jeff, Tarapore, Danesh et Mouret, Jean-Baptiste, 2015. Robots that can adapt like animals. *Nature*, 521(7553) :503–507.

Dekker, MHP, 2009. Zero-moment point method for stable biped walking. *Eindhoven University of Technology*.

Dynamics, Boston, 2013. Atlas - the agile anthropomorphic robot. http://www.bostondynamics.com/robot_Atlas.html.

EDUCATION, 2015. Bulletin officiel spécial n°11 du 26 novembre 2015. http://www.education.gouv.fr/pid285/bulletin_officiel.html?cid_bo=94753.

ENGLSBERGER, Johannes, WERNER, Alexander, OTT, Christian, HENZE, Bernd, ROA, Maximo A, GAROFALO, Gianluca, BURGER, Robert, BEYER, Alexander, EIBERGER, Oliver, SCHMID, Korbinian *et al.*, 2014. Overview of the torque-controlled humanoid robot toro. Dans *IEEE-RAS International Conference on Humanoid Robots*, pages 916–923.

FABRE, Rémi, GIMBERT, Hugo, GONDRY, Loic, HOFER, Ludovic, LY, Olivier, N'GUYEN, Steve, PASSAULT, Grégoire et ROUXEL, Quentin, 2016. Rhoban football club team - description paper.

FABRE, Rémi, ROUXEL, Quentin, PASSAULT, Grégoire, N'GUYEN, Steve et LY, Olivier, Accepted. Dynaban, an open-source alternative firmware for dynamixel servo-motors. Dans *Symposium RoboCup 2016 : Robot World Cup XX*.

FUDAL, Paul, GIMBERT, Hugo, GONDRY, Loic, HOFER, Ludovic, LY, Olivier et PASSAULT, Grégoire, 2013. An experiment of low cost entertainment robotics. Dans *2013 IEEE RO-MAN*, pages 820–825. IEEE.

FUJITA, Masahiro et KITANO, Hiroaki, 1998. Development of an autonomous quadruped robot for robot entertainment. *Autonomous Robots*, 5(1) :7–18.

FULL, RJ, 1989. Mechanics and energetics of terrestrial locomotion : bipeds to polypeds. *Energy Transformations in Cells and Animals*, pages 175–182.

FULL, Robert J et KODITSCHEK, Daniel E, 1999. Templates and anchors : neuromechanical hypotheses of legged locomotion on land. *Journal of Experimental Biology*, 202(23) :3325–3332.

GEIJTENBEEK, Thomas, VAN DE PANNE, Michiel et VAN DER STAPPEN, A Frank, 2013. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6) :206.

GOOGLE, 2012. Blockly : a library for building visual programming editors. https://developers.google.com/blockly/.

GOUAILLIER, David, HUGEL, Vincent, BLAZEVIC, Pierre, KILNER, Chris, MONCEAUX, Jérôme, LAFOURCADE, Pascal, MARNIER, Brice, SERRE, Julien et MAISONNIER, Bruno, 2009. Mechatronic design of nao humanoid. Dans *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 769–774. IEEE.

GRAF, Colin et RÖFER, Thomas, 2010. A closed-loop 3d-lipm gait for the robocup standard platform league humanoid. Dans *Proceedings of the Fifth Workshop on Humanoid Soccer Robots in conjunction with the*.

HA, Inyong, TAMURA, Yusuke, ASAMA, Hajime, HAN, Jeakweon et HONG, Dennis W, 2011. Development of open humanoid platform darwin-op. Dans *SICE Annual Conference (SICE), 2011 Proceedings of*, pages 2178–2181. IEEE.

HANSEN, Nikolaus et OSTERMEIER, Andreas, 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2) :159–195.

HENGST, Bernhard, IBBOTSON, Darren, PHAM, Son Bao et SAMMUT, Claude, 2001. Omnidirectional locomotion for quadruped robots. Dans *RoboCup 2001 : Robot Soccer World Cup V*, pages 368–373. Springer.

HOBBELEN, Daan, DE BOER, Tomas et WISSE, Martijn, 2008. System overview of bipedal robots flame and tulip : Tailor-made for limit cycle walking. Dans *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2486–2491. IEEE.

HOLLINGER, Avrum et WANDERLEY, Marcelo M, 2006. Evaluation of commercial force-sensing resistors. Dans *Proceedings of International Conference on New Interfaces for Musical Expression*. Citeseer.

HORMANN, Kai et AGATHOS, Alexander, 2001. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3) :131–144.

HUGEL, Vincent, BLAZEVIC, Pierre, STASSE, Olivier et BONNIN, Patrick, 2003. Trot gait design details for quadrupeds. Dans *Robot Soccer World Cup*, pages 495–502. Springer.

JANSEN, Theo, 2007. *The great pretender*. 010 Publishers.

JONES, Rhys, HAUFE, Patrick, SELLS, Edward, IRAVANI, Pejman, OLLIVER, Vik, PALMER, Chris et BOWYER, Adrian, 2011. Reprap–the replicating rapid prototyper. *Robotica*, 29(01) :177–191.

JOUANDEAU, Nicolas et HUGEL, Vincent, 2013. Simultaneous evolution of leg morphology and walking skills to build the best humanoid walker. Dans *IEEE-RAS Int. Conf. on Humanoid Robots*.

KAJITA, Shuuji, KANEHIRO, Fumio, KANEKO, Kenji, FUJIWARA, Kiyoshi, HARADA, Kensuke, YOKOI, Kazuhito et HIRUKAWA, Hirohisa, 2003. Biped walking pattern generation by using preview control of zero-moment point.

Dans *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, tome 2, pages 1620–1626. IEEE.

KAJITA, Shuuji, KANEHIRO, Fumio, KANEKO, Kenji, YOKOI, Kazuhito et HIRUKAWA, Hirohisa, 2001. The 3d linear inverted pendulum mode : A simple modeling for a biped walking pattern generation. Dans *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, tome 1, pages 239–246. IEEE.

KANEHIRA, Noriyuki, KAWASAKI, TU, OHTA, Shigehiko, ISMUMI, T, KAWADA, Tadahiro, KANEHIRO, Fumio, KAJITA, Shuuji et KANEKO, Kenji, 2002. Design and experiments of advanced leg module (hrp-2l) for humanoid robot (hrp-2) development. Dans *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, tome 3, pages 2455–2460. IEEE.

KANEKO, Kenji, KANEHIRO, Fumio, MORISAWA, Mitsuharu, AKACHI, Kazuhiko, MIYAMORI, Go, HAYASHI, Atsushi et KANEHIRA, Noriyuki, 2011. Humanoid robot hrp-4-humanoid robotics platform with lightweight and slim body. Dans *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4400–4407. IEEE.

KANG, Dong-Oh, LEE, Yun-Jung, LEE, Seung-Ha, HONG, Yeh Sun et BIEN, Zeungnam, 1997. A study on an adaptive gait for a quadruped walking robot under external forces. Dans *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, tome 4, pages 2777–2782. IEEE.

KATO, Ichiro, 1973. The wabot-1. *Bulletin of Science and engineering Research Laboratory, Waseda University*, (62).

KIMURA, Hajime, YAMASHITA, Toru et KOBAYASHI, Shigenobu, 2001. Reinforcement learning of walking behavior for a four-legged robot. Dans *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, tome 1, pages 411–416. IEEE.

KITANO, Hiroaki, ASADA, Minoru, KUNIYOSHI, Yasuo, NODA, Itsuki et OSAWA, Eiichi, 1997. Robocup : The robot world cup initiative. Dans *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM.

KOENIG, Nathan et HOWARD, Andrew, 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. Dans *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, tome 3, pages 2149–2154. IEEE.

KOHL, Nate et STONE, Peter, 2004. Policy gradient reinforcement learning for fast quadrupedal locomotion. Dans *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, tome 3, pages 2619–2624. IEEE.

KURAZUME, Ryo, YONEDA, Kan et HIROSE, Shigeo, 2002. Feedforward and feedback dynamic trot gait control for quadruped walking vehicle. *Autonomous Robots*, 12(2) :157–172.

LEE, T-T, LIAO, C-M et CHEN, Ting-Kou, 1988. On the stability properties of hexapod tripod gait. *IEEE Journal on Robotics and Automation*, 4(4) :427–434.

LEWIS, M Anthony, FAGG, Andrew H et SOLIDUM, Alan, 1992. Genetic programming approach to the construction of a neural network for control of a walking robot. Dans *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2618–2623. IEEE.

MAES, Pattie et BROOKS, Rodney A, 1990. Learning to coordinate behaviors. Dans *AAAI*, pages 796–802.

MAGNENAT, Stéphane, RÉTORNAZ, Philippe, BONANI, Michael, LONGCHAMP, Valentin et MONDADA, Francesco, 2011. Aseba : A modular architecture for event-based control of complex robots. *Mechatronics, IEEE/ASME Transactions on*, 16(2) :321–329.

MARDER, Eve et BUCHER, Dirk, 2001. Central pattern generators and the control of rhythmic movements. *Current biology*, 11(23) :R986–R996.

MCGEER, Tad, 1990. Passive dynamic walking. *The international journal of robotics research*, 9(2) :62–82.

MCGHEE, Robert B, 1968. Some finite state aspects of legged locomotion. *Mathematical Biosciences*, 2(1-2) :67–84.

MCGHEE, Robert B et FRANK, Andrew A, 1968a. On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, 3 :331–351.

MCGHEE, Robert B et FRANK, Andrew Alfonso, 1968b. Optimum quadruped creeping gaits. Rapport technique, DTIC Document.

MEGARO, Vittorio, THOMASZEWSKI, Bernhard, NITTI, Maurizio, HILLIGES, Otmar, GROSS, Markus et COROS, Stelian, 2015. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6) :216.

MISSURA, Marcell et BEHNKE, Sven, 2011. Lateral capture steps for bipedal walking. Dans *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 401–408. IEEE.

MOMBAUR, Katja, TRUONG, Anh et LAUMOND, Jean-Paul, 2010. From human to humanoid locomotion—an inverse optimal control approach. *Autonomous robots*, 28(3) :369–383.

MORÉ, Jorge J, 1978. The levenberg-marquardt algorithm : implementation and theory. Dans *Numerical analysis*, pages 105–116. Springer.

MUYBRIDGE, Eadweard, 1899. *Animals in motion*. Courier Corporation.

NEUHAUS, Peter D, PRATT, Jerry E et JOHNSON, Matthew J, 2011. Comprehensive summary of the institute for human and machine cognition's experience with littledog. *The International Journal of Robotics Research*, 30(2) :216–235.

OGURA, Yu, AIKAWA, Hiroyuki, SHIMOMURA, Kazushi, KONDO, Hideki, MORISHIMA, Akitoshi, LIM, Hun-ok et TAKANISHI, Atsuo, 2006. Development of a new humanoid robot wabian-2. Dans *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 76–81. IEEE.

OPENSCAD, 2010. Openscad : The programmers solid 3d cad modeller. http://www.openscad.org/.

PASSAULT, Grégoire, ROUXEL, Quentin, FABRE, Remi, N'GUYEN, Steve et LY, Olivier, 2016a. Optimizing morphology and locomotion on a corpus of parametric legged robots. Dans *Conference on Biomimetic and Biohybrid Systems*, pages 227–238. Springer.

PASSAULT, Grégoire, ROUXEL, Quentin, HOFER, Ludovic, N'GUYEN, Steve et LY, Olivier, 2015. Low-cost force sensors for small size humanoid robot. Dans *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on (Video Contribution)*, pages 1148–1148. IEEE.
URL https://youtu.be/_d7Phe0qois

PASSAULT, Grégoire, ROUXEL, Quentin, PETIT, François et LY, Olivier, 2016b. Metabot : a low-cost legged robotics platform for education. Dans *Autonomous Robot Systems and Competitions (ICARSC), 2016 IEEE International Conference on*. IEEE.

PASTOR, Peter, KALAKRISHNAN, Mrinal, RIGHETTI, Ludovic et SCHAAL, Stefan, 2012. Towards associative skill memories. Dans *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 309–315. IEEE.

Pollack, Jordan B et Lipson, Hod, 2000. The golem project : Evolving hardware bodies and brains. Dans *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, pages 37–42. IEEE.

Pot, Emmanuel, Monceaux, Jérôme, Gelin, Rodolphe et Maisonnier, Bruno, 2009. Choregraphe : a graphical tool for humanoid robot programming. Dans *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 46–51. IEEE.

Pratt, Gill A et Williamson, Matthew M, 1995. Series elastic actuators. Dans *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, tome 1, pages 399–406. IEEE.

Quigley, Morgan, Conley, Ken, Gerkey, Brian, Faust, Josh, Foote, Tully, Leibs, Jeremy, Wheeler, Rob et Ng, Andrew Y, 2009. Ros : an open-source robot operating system. Dans *ICRA workshop on open source software*, tome 3, page 5. Kobe, Japan.

Raibert, Marc, Blankespoor, Kevin, Nelson, Gabriel, Playter, Rob et Team, TB, 2008. Bigdog, the rough-terrain quadruped robot. Dans *Proceedings of the 17th World Congress*, tome 17, pages 10822–10825. Proceedings Seoul, Korea.

Raibert, Marc, Chepponis, Michael et Brown, HBJR, 1986. Running on four legs as though they were one. *IEEE Journal on Robotics and Automation*, 2(2) :70–82.

Resnick, Mitchel, Maloney, John, Monroy-Hernández, Andrés, Rusk, Natalie, Eastmond, Evelyn, Brennan, Karen, Millner, Amon, Rosenbaum, Eric, Silver, Jay, Silverman, Brian *et al.*, 2009. Scratch : programming for all. *Communications of the ACM*, 52(11) :60–67.

Riedo, Fanny, Chevalier, Morgane, Magnenat, Stéphane et Mondada, Francesco, 2013. Thymio ii, a robot that grows wiser with children. Dans *Advanced Robotics and its Social Impacts (ARSO), 2013 IEEE Workshop on*, pages 187–193. IEEE.

Robotics, Aldebaran, 2010. Projet romeo. http://projetromeo.com/.

Rohmer, Eric, Singh, Surya PN et Freese, Marc, 2013. V-rep : A versatile and scalable robot simulation framework. Dans *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE.

Rouxel, Quentin, Passault, Gregoire, Hofer, Ludovic, N'Guyen, Steve et Ly, Olivier, Accepted. Learning the odometry on a small humanoid robot.

Dans *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE.

Rouxel, Quentin, Passault, Grégoire, Hofer, Ludovic, N'Guyen, Steve et Ly, Olivier, 2015. Rhoban hardware and software open source contributions for robocup humanoids. Dans *Proceedings of 10th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conference on Humanoid Robots, Seoul, Korea*.

Ruiz-del Solar, Javier, Palma-Amestoy, Rodrigo, Marchant, Román, Parra-Tsunekawa, Isao et Zegers, Pablo, 2009. Learning to fall : Designing low damage fall sequences for humanoid soccer robots. *Robotics and Autonomous Systems*, 57(8) :796–807.

Sakagami, Yoshiaki, Watanabe, Ryujin, Aoyama, Chiaki, Matsunaga, Shinichi, Higaki, Nobuo et Fujimura, Kikuo, 2002. The intelligent asimo : System overview and integration. Dans *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, tome 3, pages 2478–2483. IEEE.

Samuelsen, Eivind et Glette, Kyrre, 2015. Real-world reproduction of evolved robot morphologies : Automated categorization and evaluation. Dans *Applications of Evolutionary Computation*, pages 771–782. Springer.

Saranli, Uluc, Buehler, Martin et Koditschek, Daniel E, 2001. Rhex : A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7) :616–631.

Sardain, Philippe et Bessonnet, Guy, 2004. Forces acting on a biped robot. center of pressure-zero moment point. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 34(5) :630–637.

Shkolnik, Alexander, Levashov, Michael, Manchester, Ian R et Tedrake, Russ, 2010. Bounding on rough terrain with the littledog robot. *The International Journal of Robotics Research*, page 0278364910388315.

Sims, Karl, 1994. Evolving virtual creatures. Dans *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM.

Song, Shin-Min, 1984. *Kinematic optimal design of a six-legged walking machine*.

Ting, LH, Blickhan, Reinhard et Full, Robert J, 1994. Dynamic and static stability in hexapedal runners. *Journal of Experimental Biology*, 197(1) :251–269.

Traversaro, Silvio, Pucci, Daniele et Nori, Francesco, 2015. In situ calibration of six-axis force-torque sensors using accelerometer measurements. Dans *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2111–2116. IEEE.

Tsagarakis, Nikolaos G, Metta, Giorgio, Sandini, Giulio, Vernon, David, Beira, Ricardo, Becchi, Francesco, Righetti, Ludovic, Santos-Victor, Jose, Ijspeert, Auke Jan, Carrozza, Maria Chiara *et al.*, 2007. icub : the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10) :1151–1175.

Tsujita, Katsuyoshi, Tsuchiya, Kazuo et Onat, Ahmet, 2001. Adaptive gait pattern control of a quadruped locomotion robot. Dans *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, tome 4, pages 2318–2325. IEEE.

Upton, Eben et Halfacree, Gareth, 2012. *Meet the Raspberry Pi*. John Wiley & Sons.

Vukobratović, Miomir et Borovac, Branislav, 2004. Zero-moment point—thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01) :157–173.

Vukobratović, Miomir, Borovac, Branislav et Šurdilović, Dragoljub, 2001. Zero-moment point—propoer interpretation and new apprications. Dans *Proc. of The Second IEEE-RAS International Conference on Humanoid Robots, CD-ROM*.

Walter-Herrmann, Julia et Büching, Corinne, 2014. *FabLab : Of machines, makers and inventors*. transcript Verlag.

Wisse, M et Hobbelen, DGE, 2007. *Limit Cycle Walking*. I-Tech Education and Publishing, Vienna, Austria.

Wisspeintner, Thomas, Van Der Zant, Tijn, Iocchi, Luca et Schiffer, Stefan, 2009. Robocup@ home : Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, 10(3) :392–426.

Zakai, Alon, 2011. Emscripten : an llvm-to-javascript compiler. Dans *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312. ACM.