Le contrôleur du robot BIP2000

Christine Azevedo Coste, Roger Pissard-Gibollet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

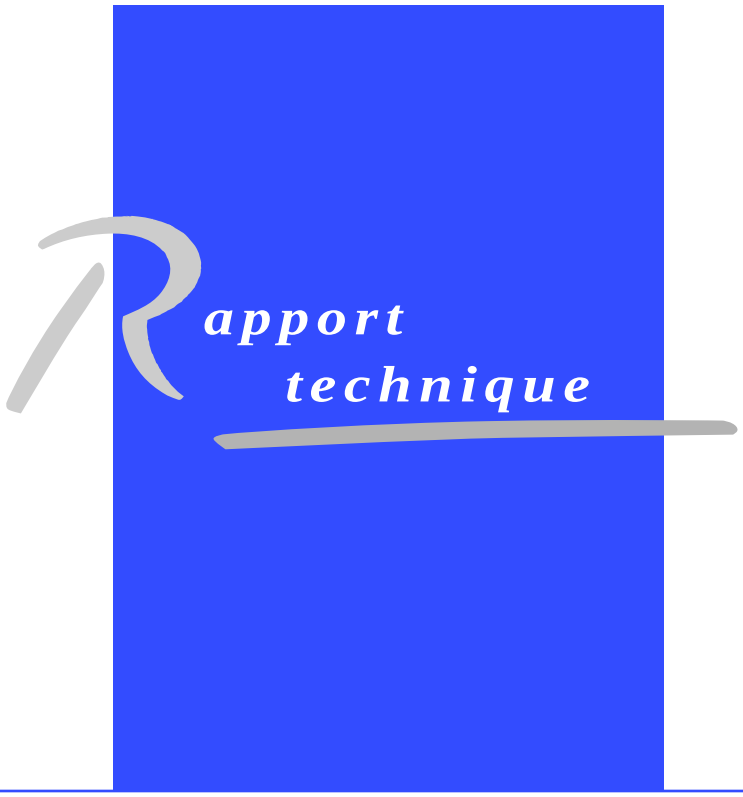
Le contrôleur du robot BIP2000

Christine Azevedo et Roger Pissard-Gibollet

N° 0249

Avril 2001

THÈME 4



ISSN 0249-0803

Le contrôleur du robot BIP2000

Christine Azevedo et Roger Pissard-Gibollet

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Bip et Service Robotique - INRIA

Rapport technique n° 0249 — Avril 2001 — 144 pages

Résumé : L'INRIA Rhône-Alpes et le Laboratoire de Mécanique des Solides de l'Université de Poitiers ont développé dans le cadre d'un projet commun, un robot anthropomorphe à quinze degrés de liberté capable de marcher dynamiquement ainsi que de gravir des escaliers. Deux prototypes ont été conçus et réalisés par le LMS. L'INRIA avait la responsabilité de la réalisation de l'armoire de commande et des câblages, ainsi que du développement d'un contrôleur pour le robot. Ce rapport présente la partie logicielle du contrôleur qui a été programmé et mis en œuvre sur la base de l'environnement temps-réel Orccad. Ce contrôleur a été utilisé dans le cadre d'expérimentations de marche statiquement stable dans le plan et de mouvements posturaux 3D statiquement stables. Ces résultats ont été présentés lors de l'Exposition Universelle de Hanovre 2000 et ont fait l'objet de plusieurs publications. Ce document fait suite au Rapport Technique 0243 qui présente les détails liés à l'armoire de commande et au câblage du robot.

Mots-clés : Robots bipèdes, ORCCAD, contrôle-commande

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

The Robot Bip2000 Controller

Abstract: INRIA Rhône-Alpes and LMS have developed an anthropomorphic walking robot with 15 degrees of freedom able to walk dynamically. Two prototypes have been designed and built by the LMS. INRIA was in charged of the design of a controller for the robot. This report presents the software part of the controller we have developed on the basis of the real-time environment Orccad. This controller has been used in the first experimentations of static 2D walking and static 3D postural movements. These results have been presented at the Hanover exposition EXPO2000 and in several publications. This document is a following of the Technical Report 0243.

Key-words: Biped robots, Orccad, control

Table des matières

Première partie

Généralités

Chapitre 1

Présentation générale

1.1 Le projet BIP2000

Le projet BIP2000 regroupe deux laboratoires français, l'*INRIA* Rhône-Alpes et le *LMS*[1]. L'objectif du projet est la réalisation d'un robot bipède anthropomorphe à 15 degrés de liberté (ddl) pouvant se déplacer dynamiquement en marchant et capable de franchir des escaliers.

1.1.1 Présentation des acteurs du projet

- Le *LMS* a la responsabilité de la partie mécanique du robot. Ce travail a abouti à la conception et à la construction de deux prototypes, ainsi qu'au développement d'un système de transmissions qui a été breveté.

- Le *Service Moyens Robotiques Vision et Réalité Virtuelle* de l'*INRIA Rhône-Alpes* a la charge de la partie électronique du robot. Ce travail correspond au câblage des différents moteurs et capteurs, ainsi qu'à la réalisation de l'armoire de commande et au développement des *drivers*. Ce travail a fait l'objet d'un rapport technique [MR00].

- Le projet *BIP* de l'*INRIA Rhône-Alpes* a la responsabilité de la partie contrôleur du robot. Ce travail a permis de développer la partie logicielle du système et d'implémenter une loi de commande permettant de poursuivre des trajectoires articulaires.

1. *Laboratoire de Mécanique des Solides de Poitiers* *http://www-lms.univ-poitiers.fr*

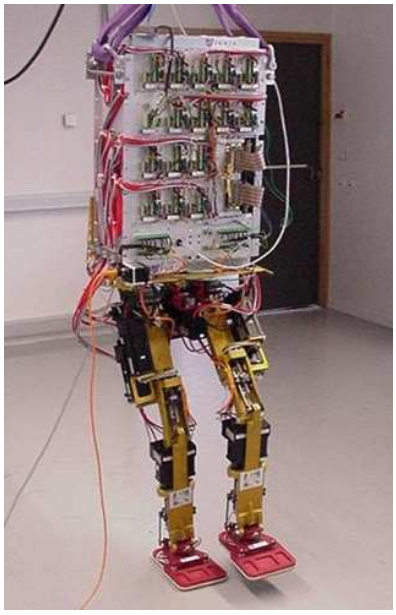


FIG. 1.1: *Robot* BIP2000

1.1.2 Réalisation de la partie mécanique des prototypes

Le *LMS* s'est inspiré des données cinématiques et des capacités dynamiques anthropomorphes pour la conception du robot. Pour plus d'informations le lecteur pourra se reporter à [SAR98].

Le calendrier de la réalisation mécanique des deux prototypes a été le suivant :

- Juillet 1999 : construction d'une première version comportant deux jambes (8 ddl) associées à un chariot à roues (fig.1.3).

- Janvier 2000 : construction d'un prototype comportant les jambes et le bassin (15 ddl).

- Septembre 2000 : rajout d'un bassin au premier prototype et obtention d'un second prototype à 15 ddl (fig.1.1).

FIG. 1.2: *Détail des jambes du robot*

Les différents paramètres du robot sont regroupés dans le document technique [SAR00].

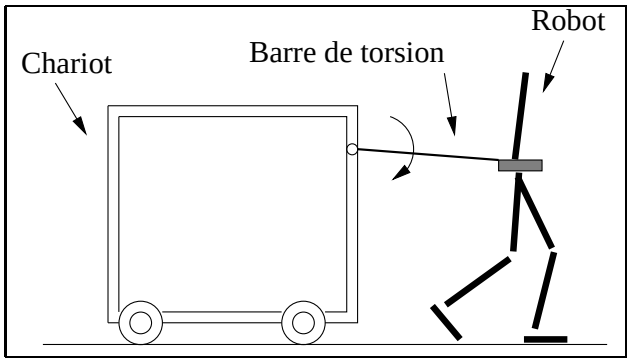


FIG. 1.3: *Version simplifiée du robot : les 2 jambes (8 ddl) sont associées à un chariot muni de 4 roues → 6 ddl plan*

1.1.3 Réalisation de la partie électronique des prototypes

Les différents prototypes ont été câblés au fur et à mesure de leur mise à disposition. La première version (8 ddl) était dotée d'une armoire de commande reliée par un cordon ombilical au robot (fig.1.4).

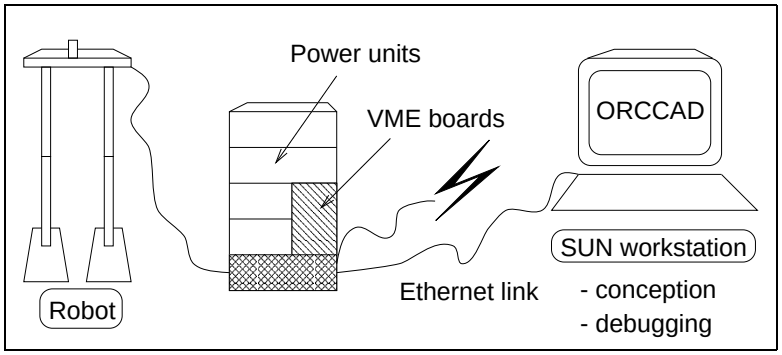


FIG. 1.4: *Architecture de la première version du robot (8 ddl)*

Les armoires de commandes des deux prototypes finaux sont intégrées à la structure mécanique et tiennent lieu de tronc (fig.1.5).

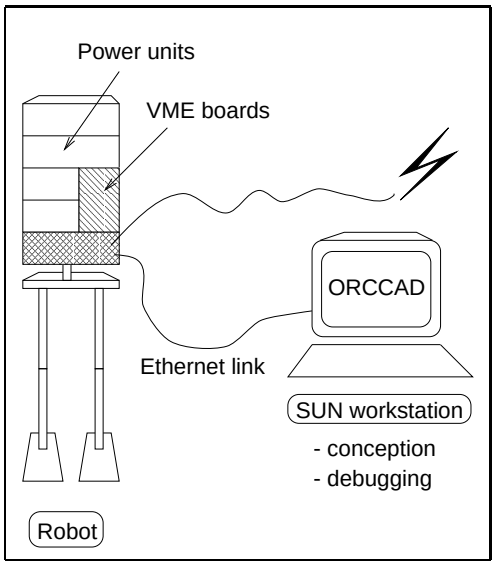


FIG. 1.5: *Architecture du robot Bip2000*

Ce travail est référencé dans le document [MR00]. Une fois ce travail réalisé, il ne restait plus qu'à compléter la plate-forme expérimentale en développant le contrôleur permettant de commander le robot.

1.1.4 Implémentation du contrôleur

Le contrôleur a été développé dans un premier temps pour le prototype à 8 ddl. Il a été complété et utilisé pour la commande du prototype complet à 15 ddl. Cette partie sera développée en détail dans ce rapport.

1.1.5 Objectifs

L'objectif final du projet est de parvenir à la réalisation de démarches dynamiquement stables dans l'espace. Tout en conservant cette donnée à l'esprit nous nous sommes penchés dans un premier temps sur le problème des mouvements statiques. Le contrôleur présenté ici devait permettre de poursuivre des trajectoires articulaires robot au sol. Les premiers résultats expérimentaux ont été présentés lors de l'Exposition Universelle 2000 à Hanovre [2].

Remarque importante: ce document fait suite au Rapport Technique [MR00]. Certaines informations concernant les drivers et les ressources physiques sont reprises dans notre document. Les informations du présent rapport devront être considérées comme référence.

2. http://www.inrialpes.fr/bip2000

1.2 Description du système robotique

Nous ne rappelons ici que les caractéristiques principales du robot, pour de plus amples informations, le lecteur pourra se reporter à [ESP00, SAR98]. Nous présentons exclusivement le robot complet à 15 ddl dans ce rapport.

1.2.1 Architecture mécanique

Le robot a été dimensionné pour la marche dynamique en 3 dimensions et pour la montée et la descente d'escaliers [ESP97].

Il peut se déplacer dans le plan grâce à la rotation de ses chevilles, genoux et hanches qui permettent la flexion/extension dans le plan sagittal (fig.1.7). Les changements de direction sont possibles par la rotation interne/externe des tronc, bassin et hanches. La rotation des chevilles, hanches et vertèbre lombaire permettent l'abduction/adduction dans le plan frontal (fig.1.7) et ainsi le maintien de l'équilibre latéral. Un degré de liberté (ddl) supplémentaire entre le tronc et le bassin rend les systèmes de déplacement et d'équilibre indépendants.

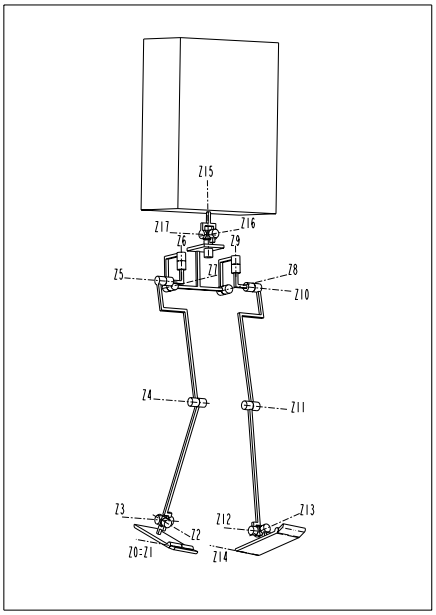


FIG. 1.6: *Cinématique du Robot*

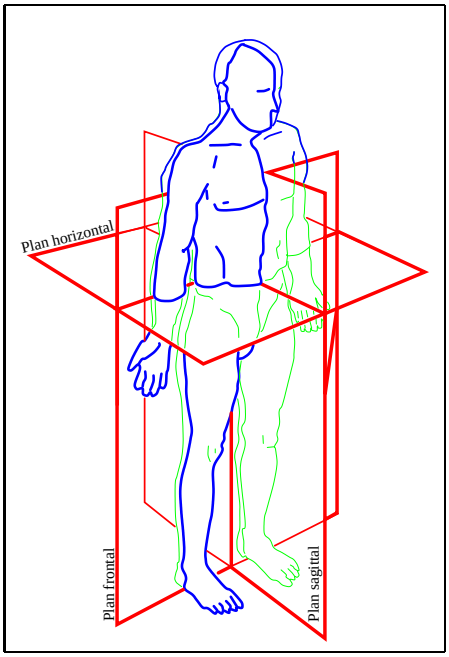


FIG. 1.7: *Plans de coupe du corps*

BIP2000 mesure 180 cm pour 105 kg. Les longueurs, masses, position des centres de masse et moments d'inertie des segments sont proches de ceux de l'être humain. Pour plus de détails se reporter à [SAR00].

1.2.2 Actionneurs et transmissions

Les actionneurs sont des moteurs à courant continu sans balais. Cinq articulations sont équipées de réducteurs *harmonic-drives* (rotations z6, z7, z8, z9, z15 fig.1.6). Les autres transmissions sont des systèmes de vis-écrou à rouleaux satellites avec biellettes (fig.1.8). Ces transmissions ont fait l'objet d'un brevet.

Les chevilles (fig.1.9) du robot et la liaison tronc/bassin (fig.1.10) sont constituées de deux ensembles moteur-transmission en parallèle. Ces ensembles bougent simultanément pour donner un mouvement sagittal ou/et frontal.

1.2.3 Capteurs

Les codeurs des moteurs-variateurs fournissent la position angulaire relative des axes moteurs. Connaissant la position initiale, on peut en déduire la position absolue.
Trois capteurs d'effort placés sur chaque pied permettent de mesurer la composante verticale

FIG. 1.8: *Système de transmission au niveau du genou*

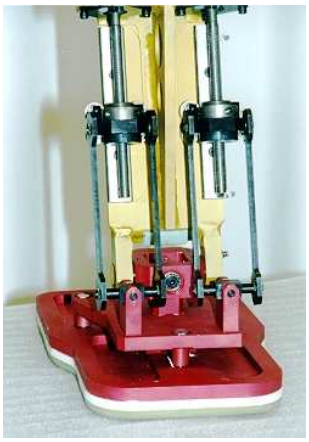


FIG. 1.9: *Robots parallèles de la cheville*

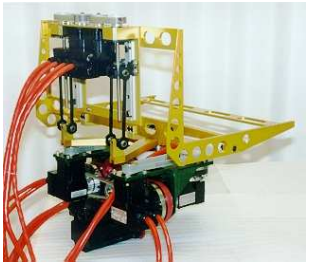


FIG. 1.10: *Robots parallèles du tronc*

de la force de réaction du sol (force de pression), les deux composantes du moment de cette force dans le plan horizontal et la position du centre de pression.
On utilise également un inclinomètre à deux axes pour connaître la direction du vecteur de gravité.
Un capteur à ultrasons sur les jambes permet la reconstruction du profil du sol.

1.2.4 Architecture informatique

Le tronc du robot contient une armoire qui comporte l'électronique de puissance et de commande. La commande en temps-réel du robot se fait sous le système d'exploitation *VxWorks* et l'environnement ORCCAD (p.18) depuis une station de travail (fig.1.5). Le robot est relié par un cordon ombilical à une source de tension et à la station de travail via un lien ethernet. À l'initialisation, les différents programmes élaborés sur ORCCAD sont transférés sur le processeur placé sur le tronc, la station ne sert ensuite qu'à la lecture des signaux entrée-sortie du robot.

1.3 Présentation de l'environnement *O*RCCAD

*O*RCCAD[3] (**O**pen **R**obot **C**ontroller **C**omputer **A**ided **D**esign) est un environnement logiciel qui permet de concevoir et de mettre en œuvre le contrôle et la commande d'un système robotique complexe. Il permet également de spécifier et valider des missions robotiques à réaliser par le système. Cet outil a été développé à l'*INRIA Rhône-Alpes* [ORC98].

1.3.1 Introduction

Un robot bipède est un système critique, il est indispensable de garantir un maximum de fiabilité avant l'exécution d'une action comme la marche par exemple. Il est nécessaire de s'assurer de la correction des spécifications définies, de la conformité de la programmation avec les spécifications, et de l'absence de problèmes liés aux contraintes temps réel. Deux aspects sont à distinguer :

- la commande pour les aspects continus,
- le contrôle pour les aspects discrets.

ORCCAD est une architecture de type contrôle-commande dont l'objectif est de faciliter l'implémentation de lois de commande et la programmation de missions robotiques complexes notamment dans des environnements peu structurés.

Nous donnons ici quelques notions relatives au fonctionnement de *O*RCCAD, pour plus d'informations le lecteur se reportera à [ARI99].

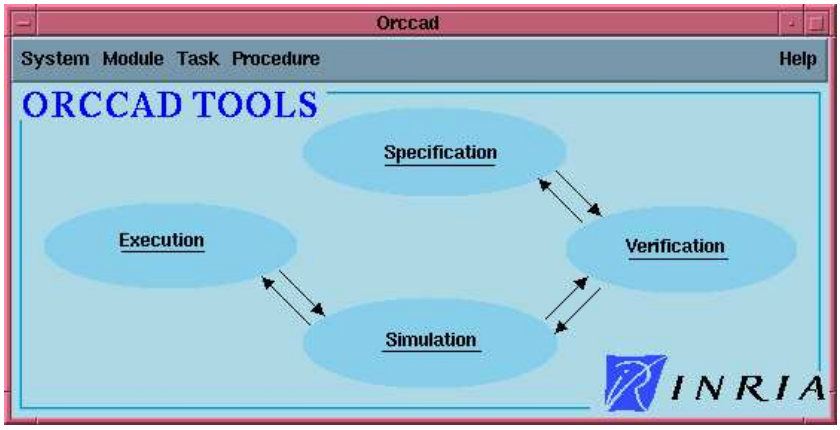


FIG. 1.11: *Interface d'accueil du logiciel* ORCCAD

3. *http://www.inrialpes.fr/iramr/Orccad/orccad-fra.html*

1.3.2 Méthodologie

La méthodologie d'ORCCAD est basée sur deux niveaux d'abstraction :

- le niveau fonctionnel,
- le niveau contrôle.

La communication entre ces deux niveaux d'abstraction se fait à l'aide d'événements discrets qui permettent de lancer des actions ou de les interrompre, et qui informent de l'état de l'exécution des actions.

Niveau fonctionnel

Ce niveau manipule une entité appelée *Tâche Robot* (*TR*). Une *TR* est la spécification complète et paramétrable d'une *Ressource Physique*, d'une *Loi de Commande*, et d'un *Comportement Logique*.

La *Ressource Physique*

La *Ressource Physique* représente le système mécanique que l'on souhaite commander. Elle contient les informations provenant du robot. Elle constitue l'interface entre la loi de commande et le robot. Elle permet l'envoi de consignes aux actionneurs et fournit les informations issues des capteurs.

La *Loi de Commande*

La *Loi de Commande* détermine la consigne à envoyer à chaque pas d'échantillonage aux actionneurs du robot en fonction des informations provenant de la *Ressource Physique*. La spécification de la *Loi de commande* se fait grâce à l'enchaînement de *Modules Algorithmiques* qui communiquent à travers des *Ports de Données*.

Le *Comportement Logique*

Le *Comportement Logique* est composé de signaux prédéfinis. Il gère le contrôle de la *TR*. L'utilisateur dispose de trois types de signaux :

1. les *Préconditions*, qui doivent être vérifiées avant que la *TR* ne démarre ;
2. les *Postconditions* qui représentent les événements émis par la *TR* lorsqu'elle se termine normalement ;
3. les *Exceptions*.

Les *Exceptions* sont divisées en trois catégories :

- *Exception de Type 1* : elles sont générées par la *TR* elle-même et signifient à la tâche courante qu'elle doit changer son mode de fonctionnement.

- *Exception de Type 2*: elles signifient à la tâche courante qu'elle doit arrêter son exécution. Le contrôle est alors transféré au niveau supérieur (niveau contrôle).
- *Exception de Type 3*: elles entraînent l'arrêt immédiat du système.

Niveau contrôle

Ce niveau manipule des entités appelées *Procédures Robot* (ou *PR*) qui permettent de composer logiquement et hiérarchiquement des *Tâches Robot* et d'autres *PR* pour spécifier des actions et des applications. Ce niveau manipule des événements discrets. Ce sont ces événements (signaux) qui permettent la communication entre les niveaux fonctionnel et de contrôle. Une application ORCCAD est définie en terme de séquence ou de mise en parallèle d'un ensemble de *TR* et de *PR* nécessaires pour la mise en œuvre de la mission robotique. La spécification sous forme de Procédure Robot se fait dans le langage ESTEREL[4].

1.3.3 Environnement de programmation

On distingue quatre étapes dans le cycle de programmation d'une application robotique (fig.1.11) :

1. la *Spécification*,
2. la *Vérification*,
3. la *Simulation*,
4. l'*Exécution*.

ORCCAD propose pour chacune de ces phases une interface graphique qui intègre des outils dédiés.

Spécification

Modules algorithmiques

Les fichiers de code associés à un module algorithmique sont :

- *inc.h* fichier include : contient les déclarations externes et les *include*.
- *var.c* fichier variables : contient la déclaration des variables internes utilisées dans les autres codes.
- *init.c* fichier initialisation : ce programme initialise les variables.
- *compute.c* fichier de calcul : programme l'algorithme qui produit les variables de sortie ou les évènements à partir des variables d'entrée.
- *end.c* fichier de fin : libère les allocations mémoire.

4. *http://www.esterel.org/*

Modules ressource physique

Les fichiers de code associés à un module ressource physique sont :

- *inc.h* fichier include : contient les déclarations externes et les *include*.
- *var.c* fichier variables : contient la déclaration des variables internes utilisées dans les autres codes.
- *init.c* fichier initialisation : ce programme initialise les moteurs ou les capteurs.
- *reinit.c* fichier de réinitialisation : ce fichier réinitialise les moteurs et capteurs (envoi de nouveaux paramètres).
- *end.c* fichier de fin : ce programme coupe les moteurs ou les capteurs.

Modules automate

Ce module spécifie le comportement logique de la *TR*.

Tâches Robots

La spécification des *TR* se fait graphiquement, sous forme de schémas-blocs. Chaque composant de la *TR* (*Ressource Physique, Module Algorithmique, Comportement Logique*) est identifié par un bloc dédié. Ces blocs communiquent grâce à des liaisons entre leurs *Ports*. On distingue :

- Le *Port de Données* qui relie les *Modules Algorithmiques* entre eux ou au *Port Driver* d'une *Ressource Physique*. Il sert également à relier le *Module Automate* avec les *Modules Algorithmiques* qui peuvent produire des événements.
- Le *Port Driver* qui permet de relier la *Ressource Physique* aux *Modules Algorithmiques*.
- Le *Port Événement* qui sert à relier le *Module Automate* et les *Modules Algorithmiques*.
- Le *Port Paramètre* qui n'apparaît que sur des *Modules Algorithmiques* et n'est relié à aucun port d'aucun bloc.

Pour le *Module Automate*, l'utilisateur doit spécifier pour chaque *Port Événement* quel est le type du signal qui lui est associé : *Précondition, Exception de Type 1*, *2* ou *3*, ou *Postcondition*. L'utilisateur doit également spécifier pour les *Ressources Physiques* et les *Modules Algorithmiques*, pour chacun des ports, un nom, un type et une nature. Il précise également le code qui implante la fonctionnalité de chaque bloc. La spécification d'une *TR* sera achevée lorsque l'utilisateur aura spécifié les contraintes temporelles au moyen d'une interface graphique spéciale, et cela pour chaque *Module Algorithmique* composant la *TR*.

Procédures

Les *Procédures* correspondent à l'arrangement logique de *TR* et de *PR*. Pour la programmation en ESTEREL des *PR*, ORCCAD propose à l'utilisateur un éditeur qui facilite la programmation des applications. Cet éditeur permet l'insertion automatique du code généré pour chaque *TR* et chaque *PR*.

Vérification

ORCCAD utilise un outil qui consiste à analyser l'automate issu du contrôle logique de l'application. Cela permet d'assurer une sûreté de fonctionnement au niveau contrôle des *TR* et des *PR*.

Simulation

ORCCAD propose un outil qui permet de simuler le système robotique complet.

Exécution

L'*Éxécution* correspond à la traduction de la spécification de l'application en code *C++* temps-réel. ORCCAD dispose d'une interface qui permet de définir les informations nécessaires à l'exécution.

Chapitre 2

Organisation logicielle

L'ensemble des informations et des programmes liés au robot bipède est archivé dans le répertoire :

`/local/projets/robotique/Bipede`

Une copie est disponible sur le cdrom dans le répertoire Bipede.

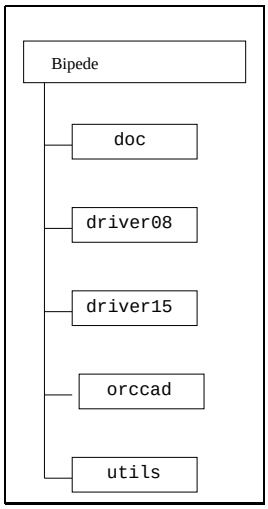


FIG. 2.1: *Arborescence du répertoire /Bipede*

2.1 Répertoire *doc*

Ce répertoire contient les différents documents concernant le robot BIP, notamment les rapports techniques et autres rapports de stage.

2.2 Répertoire *driver*15

Ce répertoire contient les sources des programmes des fonctions drivers du robot [MR00] dans sa version finale à 15 ddl. Ces fonctions seront sollicitées au niveau de la *Ressource Physique* des applications ORCCAD.

2.2.1 Programme *hardBip*

Définit les accès Hardware.

Fonction *bipHardInit*

1. Cette fonction initialise les structures logicielles et les registres des modules IP de l'armoire de commande.
2. Aucun paramètre n'est utilisé.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement.

Fonction *bipHardClose*

1. Cette fonction ferme les structures logicielles et met les modules IP dans le mode approprié.
2. Aucun paramètre n'est utilisé.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement.

Fonction *bipHardGetEncoders*

1. Cette fonction récupère les valeurs IP quadrature (codeurs de moteurs).
2. Les paramètres sont le numéro du canal et la valeur du codeur.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si le numéro de canal ne correspond pas à un module IP de type quadrature.

Fonction *bipHardDisableTopZ*

1. Cette fonction désactive le top zéro des IP quadrature (codeurs moteurs). Elle permet d'utiliser les codeurs en mono-tour.
2. Aucun paramètre n'est spécifié.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipHardEnableTopZ*

1. Cette fonction active le top zéro des IP quadrature (codeurs moteurs). Elle permet d'utiliser les codeurs en multi-tours. Le compte de quadrature est remis à zéro chaque fois que le top est détecté sur le tour moteur.
2. Aucun paramètre n'est spécifié.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipHardPutAnalog*

1. Cette fonction met la valeur ([-10.0...10.0]) dans le DAC. Elle permet d'utiliser les convertisseurs numériques-analogiques.
2. Les paramètres sont le numéro du canal et la valeur du codeur.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou si le numéro de canal ne correspond pas à un module IP de type DAC.

Fonction *bipHardPGetAnalog*

1. Cette fonction récupère la valeur ([-10.0...10.0]) sur module ADC. Elle lit les tensions sur les convertisseurs analogiques-numériques.
2. Les paramètres sont le numéro du canal et la valeur du codeur.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou si le numéro de canal ne correspond pas à un module IP de type ADC.

Fonction *bipHardGetDigit*

1. Cette fonction récupère la valeur (0 ou 1) sur module Unidig. Elle utilise des voies logiques en sortie.
2. Les paramètres sont le numéro du canal et la valeur du codeur.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou si le numéro de canal ne correspond pas à un module IP de type Unidig.

Fonction *bipHardPutDigit*

1. Cette fonction met la valeur (0 ou 1) sur module Unidig. Il s'agit de la lecture des voies logiques configurées en entrée.
2. Les paramètres sont le numéro du canal et la valeur du codeur.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou si le numéro de canal ne correspond pas à un module IP de type Unidig.

Fonction *bipKeywords*

1. Cette fonction analyse une ligne de format spécifique sur la console. Cette ligne est composé de texte et de 15 valeurs espacées. La fonction transforme la ligne en un vecteur contenant les 15 valeurs et une clef contenant le texte.
2. Les paramètres sont la clef et le vecteur.
3. La fonction retourne la valeur -1 si l'analyse a échoué et une valeur entière de 0 à 14 correspondant au nombre de valeurs de la ligne.

Fonction *bipHardMenu*

1. Cette fonction imprime à l'écran un panneau en format ascii pour utiliser les fonctions de *hardBip*.
2. La fonction n'utilise aucun paramètre et ne retourne aucune valeur.

2.2.2 Programme *drvBip*

Le programme *drvBip* décrit les accès bas niveau. Il définit différentes fonctions.

Fonction *bipOpen*

1. Cette fonction initialise le robot.
2. Elle dispose d'un paramètre *absolute* qui peut prendre la valeur $TRUE$ ou $FALSE$. La valeur $TRUE$ correspond au cas où l'on utilise les valeurs d'offset des articulations. Cette situation survient lorsqu'on positionne le robot dans une position connue pour laquelle on connaît les valeurs absolues des articulations avant de démarrer le système. La valeur $FALSE$ signifie que l'on travaille en mode relatif, les offset sont des valeurs nulles, on utilise les valeurs des potentiomètres pour connaître la position de départ du robot.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipInhibit*

1. Cette fonction inhibe les amplificateurs de puissance des moteurs.
2. Aucun paramètre n'est utilisé.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipClose*

1. Cette fonction termine l'accès hard du bipède.
2. Aucun paramètre n'est utilisé.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipGetEncoders*

1. Cette fonction récupère les positions angulaires (en radians) des codeurs *encoders* sur les axes moteurs.
2. Le paramètre est un vecteur contenant les 15 valeurs des différents codeurs.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipSetOffsetJoints*

1. Cette fonction définit les valeurs d'offset des positions angulaires des moteurs (rad). Ces valeurs permettent de déterminer la position absolue des moteurs connaissant la position relative. $\Theta_{abs} = \Theta_{rel} + offset$
2. Le paramètre est un vecteur contenant les 15 valeurs des valeurs d'offset des positions angulaires $offsetj$.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipEnableTopZEncoders*

1. Cette fonction définit le top zéro sur chaque position moteur. Chaque fois qu'un top zéro est détecté lorsque le moteur tourne, les valeurs codeurs sont effacées et les valeur des angles prennent les valeurs d'offset.
2. Le paramètre est un vecteur contenant 15 valeurs *enable*. Ces valeurs sont $TRUE$ si le top zéro est activé et $FALSE$ sinon.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipGetPotars*

1. Cette fonction lit les valeurs (V) fournies par les potentiomètres sur chaque articulation.
2. Le paramètre est un vecteur contenant 15 valeurs *potars*.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipGetCurrent*

1. Cette fonction lit les valeurs (V) sur chaque moteur.
2. Le paramètre est un vecteur contenant 15 valeurs *current*.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipPutCurrent*

1. Cette fonction écrit les valeurs de tension aux bornes de chaque moteur.
2. Le paramètre est un vecteur contenant 15 valeurs *current*.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou si les valeurs *current* dépassent les limites autorisées.

Fonction *bipStop*

1. Cette fonction envoie la valeur 0V aux moteurs.
2. Aucun paramètre n'est nécessaire.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipCheck*

1. Cette fonction recherche des erreurs hardware éventuelles.
2. Le paramètre *error* a pour valeur BIP_NO_ERROR si toutes les fonctions ont retourné la valeur OK. *error* a pour valeur $BIP\_ERROR\_LIMIT\_LEFT$ si une butée électrique a été déclenchée sur la jambe gauche. *error* a pour valeur $BIP\_ERROR\_LIMIT\_RIGHT$ si une butée électrique a été déclenchée sur la jambe droite.
   error a pour valeur $BIP\_ERROR\_EMERGENCY$ si le bouton arrêt d'urgence est enclenché.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau ou s'il y a une erreur au niveau du robot.

Fonction *bipSetOffsetForce*

1. Cette fonction définit les valeurs d'offset (N) des capteurs de force placés sur les pieds du robot.
2. Le paramètre est un vecteur regroupant les 6 valeurs d'offset des capteurs d'effort $offset$.
3. La fonction retourne la valeur OK si tout s'est déroulé normalement et $ERROR$ si une erreur est survenue dans l'accès bas niveau.

Fonction *bipGetFeetForce*

1. Cette fonction lit les pressions sur les pieds du robot (N).

2. Le paramètre est un vecteur regroupant les 6 valeurs de pression sur les capteurs d'effort *force*.

3. La fonction retourne la valeur *OK* si tout s'est déroulé normalement et *ERROR* si une erreur est survenue dans l'accès bas niveau.

2.2.3 Programme *menuBip*

Le programme *menuBip* définit le menu ascii qui permet d'effectuer les tests sur le driver et le hardware du robot. Ce menu est accessible par la commande *bipMenu* depuis *VxWorks* (p.141).

2.3 Répertoire *utils*

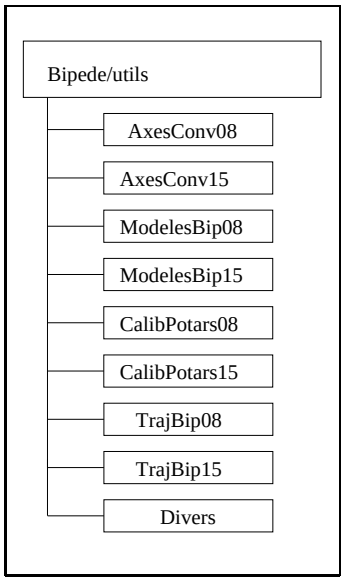


FIG. 2.2: *Arborescence du répertoire /Bipede/utils*

2.3.1 Description générale des répertoires

Le détail du contenu des différents répertoires sera détaillé plus en avant dans ce rapport. Des fichiers de documentation README sont disponibles.

Répertoires $AxesConv08$ **et** $AxesConv15$

Les répertoires $AxesConv*$ indicés 08 et 15 respectivement pour les versions 8 ddl et 15 ddl du robot contiennent les programmes qui permettent d'estimer les rapports de réduction des différentes transmissions articulaires et positions articulaires en fonction des informations des codeurs des moteurs.

Répertoires $ModelesBip08$ **et** $ModelesBip15$

Les répertoires $ModelesBip*$ indicés 08 et 15 respectivement pour les versions 8 ddl et 15 ddl du robot contiennent la définition des termes de l'équation dynamique du robot. Jusqu'ici seul le vecteur gravité est utilisé.

Répertoires $CalibPotars08$ **et** $CalibPotars15$

Les répertoires $CalibPotars*$ indicés 08 et 15 respectivement pour les versions 8 ddl et 15 ddl du robot contiennent les programmes nécessaires à la calibration des potentiomètres. La procédure de campagne de mesure est décomposée sous différents fichiers, et l'utilisation de ces informations est automatisée.

Répertoires $TrajBip08$ **et** $TrajBip15$

Les répertoires $TrajBip*$ indicés 08 et 15 respectivement pour les versions 8 ddl et 15 ddl du robot contiennent des programmes permettant de vérifier les trajectoires articulaires définies sous forme de points de passage d'un polynôme du cinquième degré.

Divers

Le répertoire */Bipede/utils* contient également la copie des différents outils utilisés autour du robot Bipède (simulateurs, scripts shell...).

2.4 Répertoire *orccad*

Ce répertoire regroupe les différentes applications développées sous le logiciel ORCCAD.

2.4.1 Répertoire *sys*

Le répertoire *sys* contient les informations et les spécifications des *Ressources Physiques* utilisables par l'ensemble des utilisateurs.

2.4.2 Répertoires *user*08 et *user*15

Les répertoires *user*∗ indicés 08 ou 15 respectivement pour la version 8ddl et 15 ddl du robot, sont les répertoires utilisateurs qui contiennent les spécifications */user/Spec/* des différents *Modules*, *Tâches Robot* et *Procédures Robot*, ainsi que le code généré à l'éxécution sous ORCCAD */user/Exec*.

Chapitre 3

Conventions utiles

Les deux versions du robot fonctionnent sur le même principe. Nous considèrerons désormais dans ce document uniquement la version 15 ddl du robot.

3.1 Mouvements et articulations

Les conventions choisies pour les appellations, signes et références sont données cidessous. Ce sont ces normes que nous utilisons dans tous les programmes du contrôleur. Les différentes figures sont également disponibles page 102.

3.1.1 Position zéro

La position "zéro" est la position du robot dans laquelle les 15 valeurs des positions articulaires sont nulles. Il s'agit de la position où le corps est tendu à la verticale et les pieds horizontaux (fig.3.1).

Cette position peut se retrouver grâce à des marques blanches qui ont été placées au niveau de chaque articulation.

Cette posture sert de référence pour calculer la position absolue du robot à partir des informations fournies par les codeurs relatifs (dans le cas ou les potentiomètres ne sont pas utilisés).

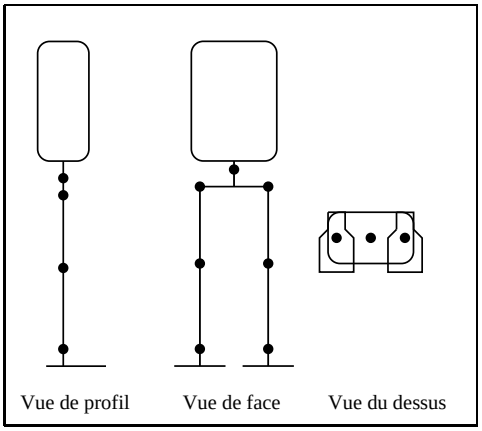


FIG. 3.1: *Position zéro du robot*

3.1.2 Mouvements

On distingue trois types de mouvements, les mouvements dans le plan sagittal, les mouvements dans le plan vertical et les mouvements dans le plan frontal (fig.1.7).

Mouvements sagittaux

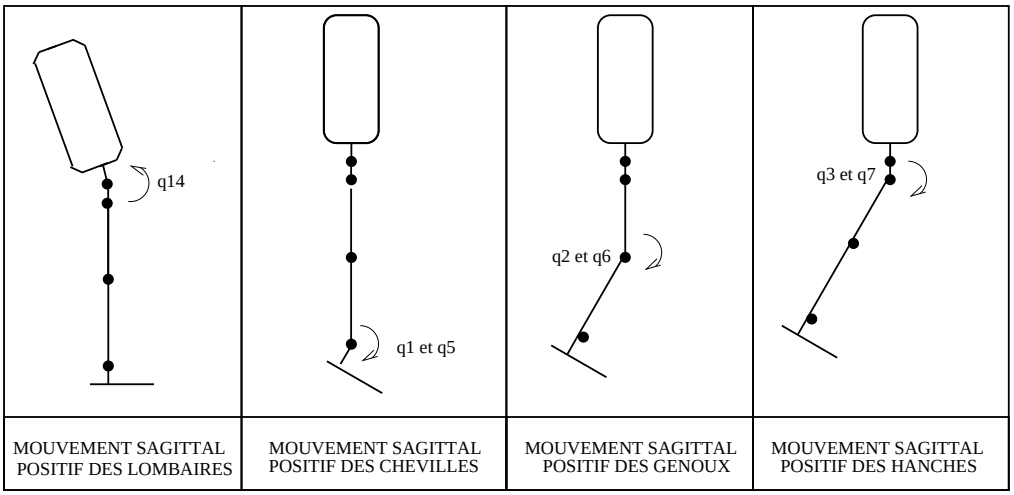


FIG. 3.2: *Mouvements du robot dans le plan sagittal appelés mouvements sagittaux*

Le robot peut se déplacer dans le plan grâce à la rotation de ses deux chevilles, deux genoux et deux hanches qui permettent la flexion/extension dans le plan sagittal (fig.3.2). La figure 3.2 représente les différents mouvements autorisés sur le robot ainsi que la convention choisie pour le sens positif de ces mouvements.

Mouvements verticaux

Au cours de la marche, les changements de direction sont possibles par la rotation interne/externe des tronc, bassin et hanches.

La figure 3.3 représente les différents mouvements autorisés sur le robot ainsi que la convention choisie pour le sens positif de ces mouvements.

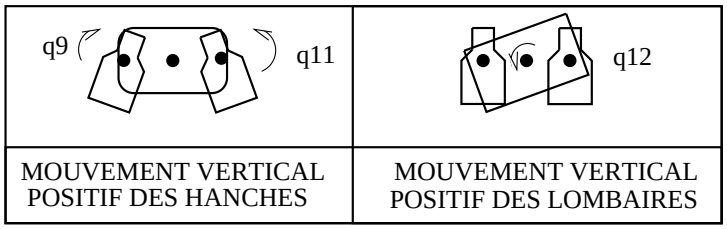


FIG. 3.3: *Mouvements du robot dans le plan horizontal appelés mouvements verticaux*

Mouvements frontaux

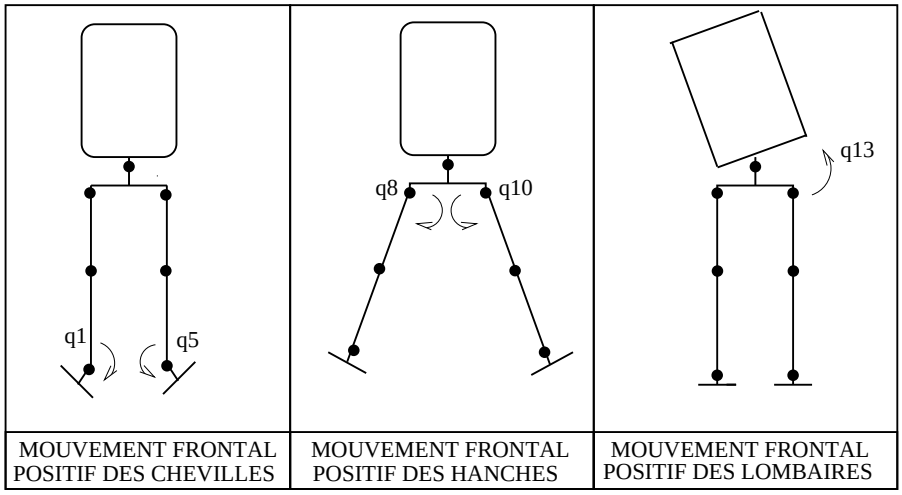


FIG. 3.4: *Mouvements du robot dans le plan frontal appelés mouvements frontaux*

La rotation des chevilles, hanches et vertèbre lombaire permettent l'abduction/adduction dans le plan frontal et ainsi le maintien de l'équilibre latéral. Un degré de liberté (ddl) supplémentaire entre le tronc et le bassin rend les systèmes de déplacement et d'équilibre indépendants. La figure 3.4 représente les différents mouvements autorisés sur le robot ainsi que la convention choisie pour le sens positif de ces mouvements.

Le tableau suivant récapitule les différents types de mouvements dont est capable le robot ainsi que l'appellation des articulations qui permettent ces mouvements.

Mouvement	**Articulation associée**
Rotation interne-externe du pied par rapport à la jambe	Chevilles axe frontal
Flexion et extension du pied par rapport à la jambe	Chevilles axe sagittal
Flexion et extension du tibia par rapport au fémur	Genou axe sagittal
Flexion et extension du fémur par rapport au tronc	Hanche axe sagittal
Abduction et adduction du fémur	Hanche axe vertical
Rotation interne et externe de la hanche	Hanche axe frontal
Rotation interne externe des lombaires	Tronc axe vertical
Inflexion latérale des lombaires	Tronc axe frontal
Flexion et extension des lombaires	Tronc axe sagittal

TAB. 3.1: *Relation entre mouvements et articulations*

3.1.3 Paramétrage des articulations

Les différentes définitions et conventions sont définies dans le fichier :

`/include/utilBip15.h`

Une copie est également disponible :

`/Bipede/utils/AxesConv15/utilBip.h`

Référençage des articulations

Dans les différents programmes développés sous ORCCAD, les articulations sont numérotées de 0 à 15 et les positions articulaires forment un vecteur q de dimension 15. L'unité de mesure des positions articulaires est le *radian*.
Pour la version 8 ddl plane du robot nous avions opté pour une numérotation des articulations. Nous avons complété cette numérotation pour les articulations supplémentaires de la version à 15 ddl.

	Articulation	**Situation**	**Position**
0	Cheville axe frontal	jambe droite	q[0]
1	Cheville axe sagittal	jambe droite	q[1]
2	Genou axe sagittal	jambe droite	q[2]
3	Hanche axe sagittal	jambe droite	q[3]
4	Cheville axe frontal	jambe gauche	q[4]
5	Cheville axe sagittal	jambe gauche	q[5]
6	Genou axe sagittal	jambe gauche	q[6]
7	Hanche axe sagittal	jambe gauche	q[7]
8	Hanche axe vertical	jambe droite	q[8]
9	Hanche axe frontal	jambe droite	q[9]
10	Hanche axe vertical	jambe gauche	q[10]
11	Hanche axe frontal	jambe gauche	q[11]
12	Tronc axe vertical	bassin	q[12]
13	Tronc axe frontal	lombaires	q[13]
14	Tronc axe sagittal	lombaires	q[14]

TAB. 3.2: *Numérotation des articulations et notation des positions articulaires associées*

Remarque : Lorsqu'on utilisera des logiciels tels que SCILAB la numérotation des éléments du vecteur se fera de 1 à 15 !

3.1.4 Conventions

Pour pallier aux inconvénients de la numérotation (peu intuitive) nous avons introduit des conventions pour l'appellation des articulations. La désignation de chaque articulation se fait par 4 sous-désignations *ArtPartieSymétrie_ Axe* qui se déclinent comme suit.

Partie	**Symétrie**	**Axe**
Che = Cheville	D = Droite ou G = Gauche	F = Frontal ou S = Sagittal
Gen = Genou	D = Droite ou G = Gauche	S = Sagittal
Han = Hanche	D = Droite ou G = Gauche	F = Frontal ou S = Sagittal ou V = Vertical
Lomb = Lombaire		F = Frontal ou S = Sagittal ou V = Vertical

TAB. 3.3: *Convention pour l'appellation des articulations* **Art=Articulation**

Ainsi la désignation de l'articulation de la hanche de la jambe gauche pour la flexion dans le plan sagittal est ArtHanG_S.

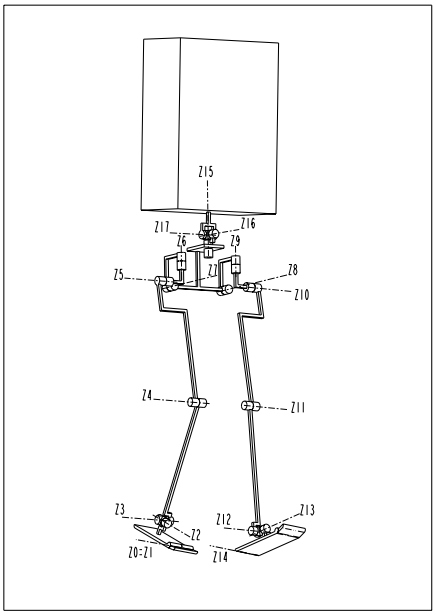


FIG. 3.5: *Cinématique du robot bipède*

Articulation	**Position articulaire**
cheville droite axe frontal	q[0] ou q[ArtCheD_F]
cheville droite axe sagittal	q[1] ou q[ArtCheD_S]
genou droit axe sagittal	q[2] ou q[ArtGenD_S]
hanche droite axe sagittal	q[3] ou q[ArtHanD_S]
cheville gauche axe frontal	q[4] ou q[ArtCheG_F]
cheville gauche axe sagittal	q[5] ou q[ArtCheG_S]
genou gauche axe sagittal	q[6] ou q[ArtGenG_S]
hanche gauche axe sagittal	q[7] ou q[ArtHanG_S]
hanche droite axe frontal	q[8] ou q[ArtHanD_F]
hanche droite axe vertical	q[9] ou q[ArtHanD_V]
hanche gauche axe frontal	q[10] ou q[ArtHanG_F]
hanche gauche axe vertical	q[11] ou q[ArtHanG_V]
lombaire axe vertical	q[12] ou q[ArtLomb_V]
lombaire axe frontal	q[13] ou q[ArtLomb_F]
lombaire axe sagittal	q[14] ou q[ArtLomb_S]

TAB. 3.4: *Conventions retenues pour la notation des positions articulaires*

3.2 Mouvements et moteurs

	Moteur	**Situation**	**Position**
0	Cheville droite	extérieur	th[0]
1	Cheville droite	intérieur	th[1]
2	Genou droit		th[2]
3	Hanche droit sagittale		th[3]
4	Cheville gauche	intérieur	th[4]
5	Cheville gauche	extérieur	th[5]
6	Genou gauche		th[6]
7	Hanche gauche sagittale		th[7]
8	Hanche droite frontale		th[8]
9	Hanche droite verticale		th[9]
10	Hanche gauche frontale		th[10]
11	Hanche gauche verticale		th[11]
12	Bassin vertical		th[12]
13	Tronc	droit	th[13]
14	Tronc	gauche	th[14]

TAB. 3.5: *Numérotation des moteurs et notation pour la position des moteurs*

Il y a un moteur par articulation, sauf dans le cas des chevilles et du tronc où les mouvements frontal/sagittal sont dus aux effets du mouvement combiné de deux moteurs montés en parallèle. Dans les différents programmes développés sous ORCCAD les positions moteurs forment un vecteur *th* de dimension 15. L'unité de mesure des positions moteur est le *radian*. Pour pallier aux inconvénients de la numérotation des moteurs nous avons introduit des conventions pour leur désignation. Pour désigner les moteurs, si le moteur correspond exactement à la motorisation de l'articulation, la même désignation est utilisée en utilisant *MotPartieSymétrie_Axe*. Sinon, comme dans le cas des lombaires et des chevilles, la désignation est faite par *MotPartieSymétrie_Moteur*.

Partie	**Symétrie**	**Moteur**
Che = Cheville	D = Droite ou G = Gauche	I = Intérieur ou E = Extérieur
Lomb = Lombaire	D = Droit ou G - Gauche	

TAB. 3.6: *Convention pour l'appellation des moteurs* **Mot=Moteur**

Moteur	Position du moteur
moteur extérieur cheville droite	th[0] ou th[MotCheD_E]
moteur intérieur cheville droite	th[1] ou th[MotCheD_I]
moteur genou droit mouvement sagittal	th[2] ou th[MotGenD_S]
moteur hanche droite mouvement sagittal	th[3] ou th[MothanD_S]
moteur intérieur cheville gauche	th[4] ou th[MotCheG_I]
moteur extérieur cheville gauche	th[5] ou th[MotCheG_E]
moteur genou gauche mouvement sagittal	th[6] ou th[MotGenG_S]
moteur hanche gauche mouvement sagittal	th[7] ou th[MothanG_S]
moteur hanche droite mouvement frontal	th[8] ou th[MothanD_F]
moteur hanche droite mouvement vertical	th[9] ou th[MotHanD_V]
moteur hanche gauche mouvement frontal	th[10] ou th[MotHanG_F]
moteur hanche gauche mouvement vertical	th[11] ou th[MotHanG_V]
moteur lombaire mouvement vertical	th[12] ou th[MotLomb_V]
moteur droit lombaire mouvement du tronc	th[13] ou th[MotLomb_D]
moteur gauche lombaire mouvement du tronc	th[14] ou th[MotLomb_G]

TAB. 3.7: *Conventions retenues pour la notation des positions moteurs*

3.3 Rapports de réduction

Le mouvement des moteurs est transformé en mouvement des articulations grâce aux systèmes de transmission. Comme nous l'avons vu à l'exception des *Harmonic Drives Gears*, les autres transmissions (vis-écrou à rouleaux satellites) n'ont pas des rapports de réduction constants. Dans les différents programmes développés sous ORCCAD les rapports de réduction (ratios) forment un vecteur *ratio* de dimension 21. On désigne les rapports de réduction par une nomenclature semblable à celle utilisée pour les articulations et les moteurs (tab.3.8).

3.4 Limitations des mouvements

Les mouvement des articulations sont limités d'un point de vue mécanique, au delà d'un certain débattement l'articulation amène les segments qu'elle relie en contact. Une sécurité électrique a été rajoutée pour arrêter le mouvement avant le choc. Enfin, si le robot évolue sur le sol et que les butées électriques sont atteintes, la puissance est coupée et le robot s'effondre. Nous avons donc ajouté une sécurité logicielle pour éviter l'enclenchement des sécurités électriques.

	Rapport de réduction	**Notation**
0	cheville droite : moteur extérieur / mouvement frontal	ratio[0] ou ratio[RatCheD_FE]
1	cheville droite : moteur extérieur / mouvement sagittal	ratio[1] ou ratio[RatCheD_SE]
2	genou droit : moteur / mouvement sagittal	ratio[2] ou ratio[RatGenD_S]
3	hanche droite : moteur / mouvement sagittal	ratio[3] ou ratio[RatHanD_S]
4	cheville gauche : moteur intérieur / mouvement frontal	ratio[4] ou ratio[RatCheG_FI]
5	cheville gauche : moteur intérieur / mouvement sagittal	ratio[5] ou ratio[RatCheG_SI]
6	genou gauche : moteur / mouvement sagittal	ratio[6] ou ratio[RatGenG_S]
7	hanche gauche : moteur / mouvement sagittal	ratio[7] ou ratio[RatHanG_S]
8	hanche droite : moteur / mouvement frontal	ratio[8] ou ratio[RatHanD_F]
9	hanche droite : moteur / mouvement vertical	ratio[9] ou ratio[RatHanD_V]
10	hanche gauche : moteur / mouvement frontal	ratio[10] ou ratio[RatHanG_F]
11	hanche gauche : moteur / mouvement vertical	ratio[11] ou ratio[RatHanG_V]
12	lombaire : moteur / mouvement vertical	ratio[12] ou ratio[RatLomb_V]
13	lombaire : moteur droit / mouvement frontal	ratio[13] ou ratio[RatLomb_FD]
14	lombaire : moteur droit / mouvement sagittal	ratio[14] ou ratio[RatLomb_SD]
15	cheville droite : moteur intérieur / mouvement frontal	ratio[15] ou ratio[RatCheD_FI]
16	cheville droite : moteur intérieur / mouvement sagittal	ratio[16] ou ratio[RatCheD_SI]
17	cheville gauche : moteur extérieur / mouvement frontal	ratio[17] ou ratio[RatCheG_FE]
18	cheville gauche : moteur extérieur / mouvement sagittal	ratio[18] ou ratio[RatCheG_SE]
19	lombaire : moteur gauche / mouvement frontal	ratio[19] ou ratio[RatLomb_FG]
20	lombaire : moteur gauche / mouvement sagittal	ratio[20] ou ratio[RatLomb_SG]

TAB. 3.8: *Conventions retenues pour la numérotation et l'appellation des rapports de réduction des transmissions moteur-articulation*

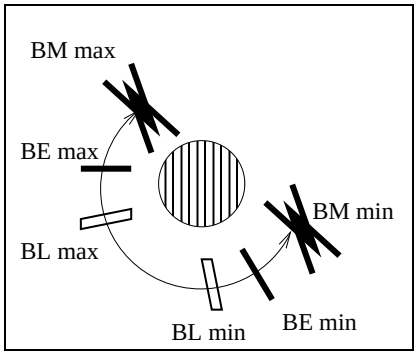


FIG. 3.6: *Position des différentes butées sur le débattement d'une articulation*

Les différentes valeurs numériques sont répertoriées en annexe page 105.

3.4.1 Butées mécaniques du robot

Les différentes butées du robot sont déclarées dans :
*/include/utilBip*15*.h* dont une copie est disponible :
*/Bipede/utils/AxesConv*15*/utilBip.h*

Ces butées correspondent au débattement de l'articulation entre deux contacts mécaniques sur une des pièces du robot (tab.2.1). L'origine correspondant à la position zéro (fig.3.1).

3.4.2 Butées électriques

Ces butées correspondent au débattement autorisé avant que l'articulation ne déclenche un capteur de fin de course. Bien entendu ces capteurs se déclenchent avant que l'articulation n'arrive en butée mécanique, ils servent de sécurité. Lorsqu'ils sont déclenchés ils provoquent un arrêt d'urgence qui coupe les moteurs. Nous donnons (tab.2.2) l'emplacement des butées électriques théoriques, en réalité ces positions ne sont pas rigoureusement effectives, il conviendrait de les évaluer expérimentalement.

3.4.3 Butées logicielles

Il est évident que lorsque le robot arrive en butée électrique les moteurs étant coupés la chute du robot est inévitable. Nous avons donc placé des butées logicielles pour éviter que le robot n'atteigne les butées électriques (tab.2.3) .

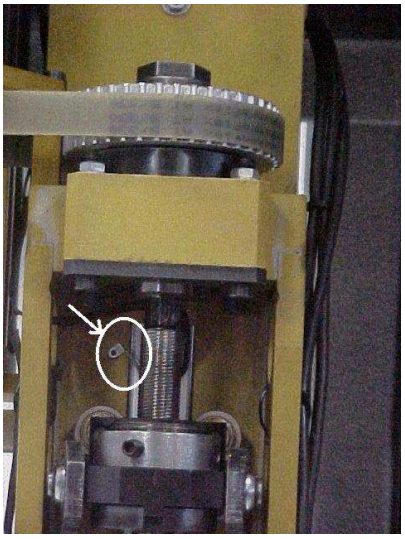


FIG. 3.7: *Capteur de fin de course*

Chapitre 4

Modélisation du système

Ce chapitre décrit la méthode utilisée pour obtenir le modèle dynamique du robot.

4.1 Introduction

D'un point de vue mécanique, le robot en phase de locomotion est un système mécanique de corps rigides soumis à des contraintes unilatérales. Ces contraintes sont des inégalités qui traduisent la non-interpénétrabilité des corps rigides en présence de frottements secs. On peut isoler quatre configurations principales pour le robot, en fonction du nombre de pieds en contact avec le sol et selon le pied en contact avec le sol.

4.1.1 Robot suspendu au dessus du sol

Dans les phases de réglage et d'ajustement des applications, le robot n'est pas posé sur le sol. De même, jusqu'ici l'initialisation du robot, c'est à dire l'étape pendant laquelle on évalue la position absolue du robot, se fait robot suspendu (fig.4.1). Dans ces deux cas, la caisse qui sert de tronc est soutenue par le haut par des sangles accrochées à un palan mécanique que l'on peut actionner par une télécommande pour descendre ou monter le robot (fig.4.2).

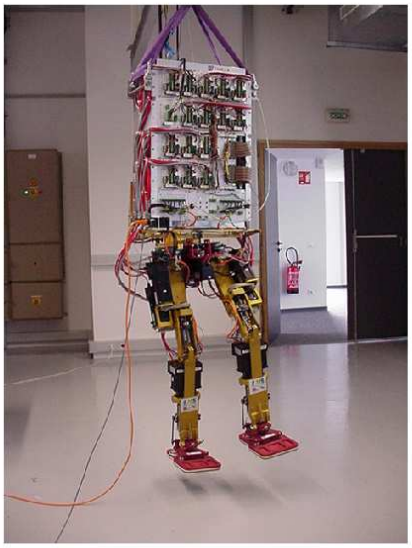


FIG. 4.1: *Robot maintenu au dessus du sol, lors de l'initialisation, du rodage et des réglages de paramètres*

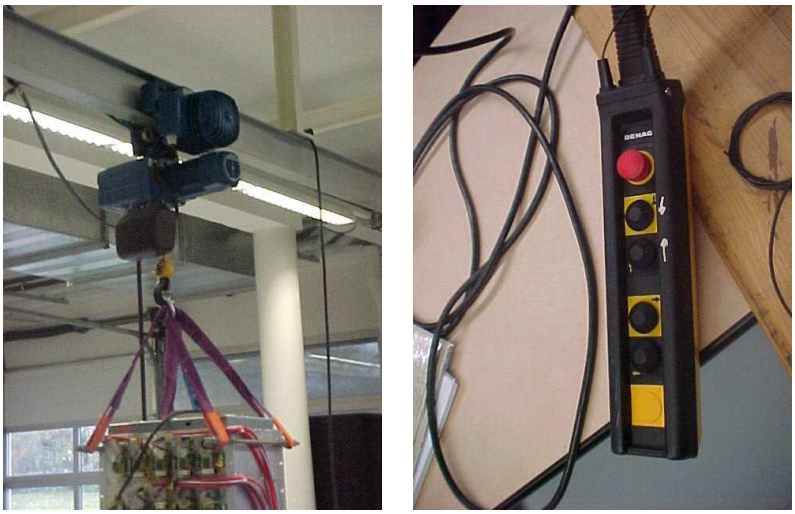
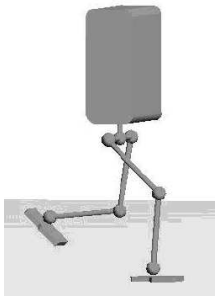


FIG. 4.2: *Système de maintien au dessus du sol*

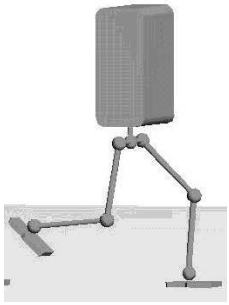
4.1.2 Robot en mouvement sur le sol

Lorsque le bipède se déplace au sol, il peut avoir un seul de ses pieds au sol (*simple support*) ou les deux en même temps (*double support*).
On a donc trois configurations différentes possibles (fig.4.3). :

1. simple support pied droit
2. simple support pied gauche
3. double support pied droit ou pied gauche en avant

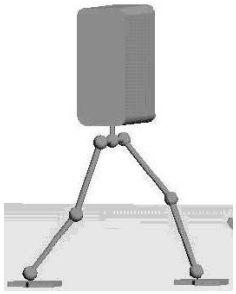


Robot en phase de simple support droit



Robot en phase de simple support gauche

Robot en phase de double support pied droit en avant



Robot en phase de double support pied gauche en avant

FIG. 4.3: *Configurations du robot évoluant sur le sol*

4.2 Méthodologie

Afin de simplifier le problème nous avons choisi de modéliser le robot dans ses différentes phases d'appui par autant de modèles de robots manipulateurs. On considère donc que le robot est rivé au sol excluant par conséquent tout glissement du pied sur le sol.

4.2.1 Phases de simple support et phase de non support

Dans un premier temps nous allons exclure les phases de double support et ainsi nous affranchir de la modélisation des contraintes. Classiquement, l'équation de la dynamique du robot peut alors s'exprimer dans chacune des différentes phases d'appui sous la forme Lagrangienne :

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \Gamma \tag{4.1}$$

où :

- q est l'ensemble des positions articulaires,
- $M(q)$ est la matrice d'inertie,
- $C(q,\dot{q})$ est le vecteur de Coriolis et des forces centrifuges,
- $G(q)$ est le vecteur de gravité,
- Γ est le vecteur des couples articulaires.

On notera tout au long de ce document : $\dot{a}$ la dérivée première de la composante a et $\ddot{a}$ sa dérivée seconde.

Remarque : nous verrons par la suite comment contourner le problème du double-support pratiquement.

4.2.2 Résolution

Afin de déterminer chaque terme (M,C,G) de l'équation (4.1), nous avons utilisé un outil de génération automatique des dynamiques de Lagrange de robots arborescents flottants rigides avec référence variable : RobotDyn qui s'appuie sur les logiciels Scilab [1] et Maple. Cet outil a été développé par F. Génot [GEN98] et modifié par P.B. Wieber [WIE01]. Cet outil utilise la représentation de Khalil-Kleinfinger et génère automatiquement son modèle dynamique.

4.3 Formalisation de Khalil Kleinfinger

La formalisation dite de Khalil-Kleinfinger [KHA86] est une modification de la représentation de Denavit-Hartenberg afin de faciliter son application aux chaînes cinématiques fermées [GOG96].

1. http://www-rocq.inria.fr/scilab

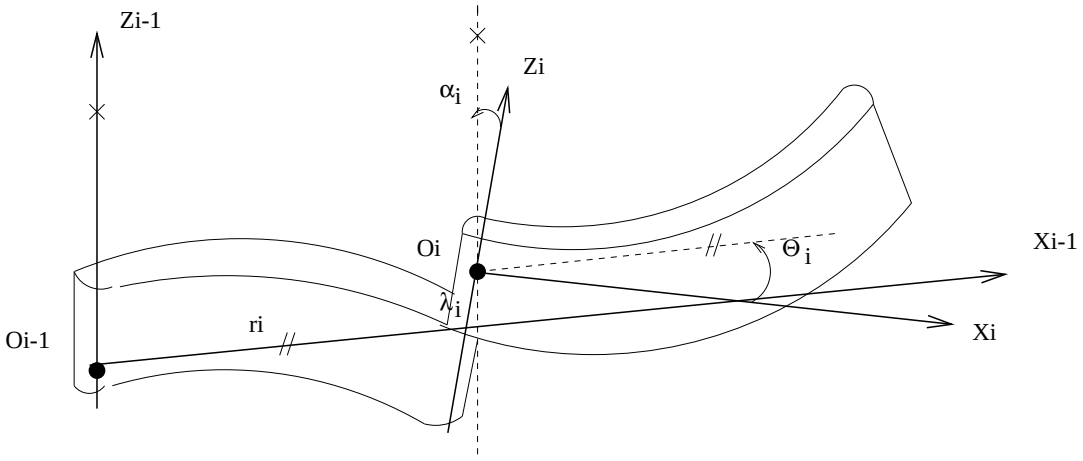


FIG. 4.4: *Le paramétrage de Khalil-Kleinfinger décrit le passage du repère $i-1$ au repère i en effectuant une translation r_i et une rotation α_i le long de l'axe $X_{ref_k} x_i$, puis une translation λ_i et une rotation θ_i le long de l'axe z_i.*

La représentation de Khalil Kleinfinger classique pour une chaîne de n segments successifs est donnée par une liste de vecteurs de la forme $\{(r_i, \lambda_i, \alpha_i, \Theta_i)\}_{i=1..n}$. On associe à l'élément i un système de coordonnées $O_i x_i y_i z_i$, l'axe z_i coïncidant avec l'axe de la paire cinématique. L'origine O_i étant située sur l'axe de la paire i. La paire i fait la liaison entre les éléments $i-1$ et i. L'axe x_i est superposé à la normale commune aux axes z_i et z_{i+1}. Pour définir la position et l'orientation du système de coordonnées i par rapport au système $i-1$ on utilise les paramètres :

- r_i : la distance entre O_{i-1} et z_i,
- λ_i : la distance entre O_i et x_{i-1},
- α_i : l'angle de rotation autour de l'axe x_{i-1}, mesuré de l'axe z_{i-1} vers l'axe z_i,
- Θ_i : l'angle de rotation autour de l'axe z_I mesuré de l'axe x_{i-1} vers l'axe x_i.

4.4 Résolution du modèle dynamique

Le logiciel ROBOTDYN se base sur deux fichiers *cinematique.maple* et *dynamique.maple* pour générer automatiquement les éléments de l'équation de la dynamique. Ces fichiers contiennent les différentes informations sur la structure cinématique et dynamique du robot. Sur la base du document technique [SAR00], nous avons complété les différents champs nécessaires.

La paramétrisation du robot selon Khalil-Kleinfinger va dépendre de la configuration considérée car on choisit la base de référence au niveau du pied en contact avec le sol.

Remarque : Les différents fichiers relatifs au modèle se trouvent dans :

`/Bipede/utils/RobotDyn`

4.5 Robot en simple support droit

On considère la situation où le robot a son pied droit posé à plat sur le sol, et l'autre pied constamment au-dessus du sol. La base de référence choisie pour le paramétrage est l'extrémité du pied droit. Il est inutile de calculer le modèle pour le simple support gauche, on le retrouve facilement à partir du modèle du simple support droit.

4.5.1 Fichier *cinematique.maple*

Les paramètres de Khalil-Kleinfinger prennent les valeurs données dans le tableau suivant.

	Solides	r_i	λ_i	α_i	Θ_i
1	Pied droit	0	0	0	$\pi/2$
2	Cardan droit	0,083	-0,170	$\pi/2$	$q[0+1]$
3	Tibia droit	0	0	$-\pi/2$	$q[1+1]$
4	Cuisse droite	0,410	0	0	$q[2+1]$
5	Levier hanche droite	0,410	0	0	$\pi/2+q[3+1]$
6	Équerre hanche droite	0	0	$\pi/2$	$-\pi/2+q[9+1]$
7	Bassin	0	0	$\pi/2$	$q[8+1]$
8	Équerre hanche gauche	0,220	0	0	$q[10+1]$
9	Levier hanche gauche	0	0	$-\pi/2$	$\pi/2+q[11+1]$
10	Cuisse gauche	0	0	$-\pi/2$	$\pi/2-q[7+1]$
11	Tibia gauche	0,410	0	0	$-q[6+1]$
12	Cardan gauche	0,410	0	0	$-q[5+1]$
13	Pied gauche	0	-0,120	$-\pi/2$	$q[4+1]$
14	Ex-orteils	0	0	0	0
15	Levier lombaires	0,110	0,128	$-\pi/2$	$q[12+1]$
16	Cardan armoire	0	0	$\pi/2$	$\pi/2+q[13+1]$
17	Support + armoire	0	0	$-\pi/2$	$q[14+1]$
18	Rotation Y	0	0	$-\pi/2$	$\pi/2+q[19+1]$
19	Rotation X	0	0	$\pi/2$	$\pi/2+q[18+1]$
20	Rotation Z	0	0	$-\pi/2$	$-\pi/2+q[20+1]$

TAB. 4.1: *Paramétrage de Khalil Kleinfinger pour le cas robot en simple-support droit*

Remarque : la numérotation des vecteurs sous SCILAB commence à 1 !

Une copie du listing du fichier est fourni en annexe p.110. Le fichier original se trouve dans : */Bipede/utils/RobotDyn/*15*ddl/*

4.5.2 Fichier *dynamique.maple*

Soit le solide i de masse m_i, muni d'une articulation de centre O_i selon l'axe Z_i (fig.4.5).
La position du centre de masse ou centre d'inertie du solide est : $G_i = (x_i; y_i; z_i) = (O_i\vec{G}_i\vec{X}_i; O_i\vec{G}_i\vec{Y}_i; O_i\vec{G}_i\vec{Z}_i)$
La matrice d'inertie exprimée en O_i est notée I_{O_i}.
Pour plus de détails le lecteur est invité à consulter [SAR00].

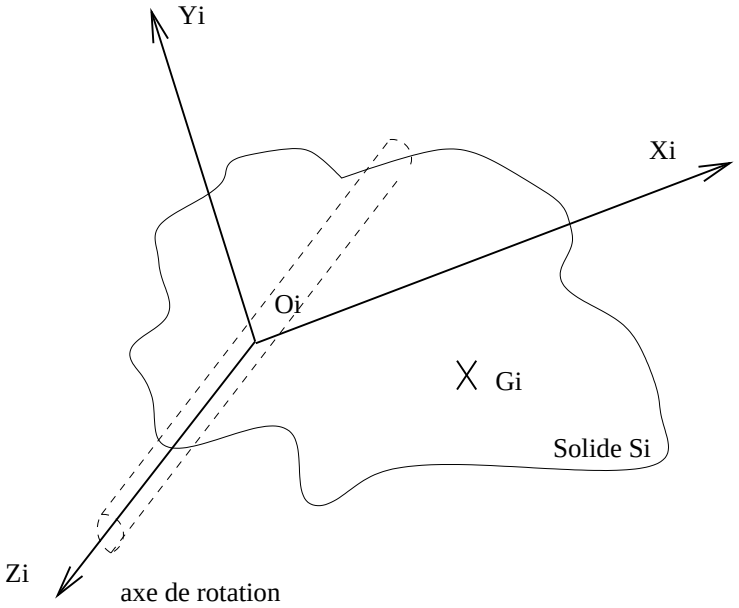


FIG. 4.5: *Paramétrage inertiel d'un solide*

Les valeurs des masses (kg), les positions des centre de masse (m) et inertie ($kg.m^2$) des segments, sont données ci-après.

		m_i	$G_i.10^{-3}$	$I_{O_i}.10^{-2}$
1	Pied de droit	2,34	$[24, 151, 0]$	$([7, 0, 0], [0, 0, 0], [0, 0, 7])$
2	Cardan jambe droit	0,18	$[0, 0, 0]$	$([0, 0, 0], [0, 0, 0], [0, 0, 0])$
3	Tibia jambe droit	5,93	$[258, 28, 0]$	$([2, -4, 0], [-4, 47, 0], [0, 0, 47])$
4	Cuisse jambe droit	10,9	$[250, 5, 45]$	$([6, -1, -15], [-1, 82, 0], [-15, 0, 80])$
5	Levier hanche droit	0,7	$[2, -34, 0]$	$([0, 0, 0], [0, 0, 0], [0, 0, 0])$
6	Équerre hanche droit	3,2	$[-5, 107, 29]$	$([6, 0, 0], [0, 0, 0], [0, 0, 5])$
7	Bassin	8,8	$[110, -12, -68]$	$([13, 1, 7], [1, 24, -2], [7, -2, 15])$
8	Équerre hanche libre	3,2	$[5, 29, -107]$	$([6, 0, 0], [0, 5, 0], [0, 0, 0])$
9	Levier hanche libre	0,7	$[2, 0, 34]$	$([0, 0, 0], [0, 0, 0], [0, 0, 0])$
10	Cuisse jambe libre	10,9	$[160, -5, -45]$	$([6, 0, 5], [0, 42, 0], [5, 0, 39])$
11	Tibia jambe libre	5,93	$[152, -28, 0]$	$([2, 3, 0], [3, 21, 0], [0, 0, 21])$
12	Cardan jambe libre	0,18	$[0, 0, 0]$	$([0, 0, 0], [0, 0, 0], [0, 0, 0])$
13	Pied libre	2,34	$[59, 0, 139]$	$([6, 0, -2], [0, 7, 0], [-2, 0, 1])$
15	Levier lombaires	1,05	$[0, 47, -25]$	$([0, 0, 0], [0, 0, 0], [0, 0, 1])$
16	Cardan lombaires	0,46	$[0, 0, 46]$	$([1, 0, 0], [0, 0, 0], [0, 0, 1])$
17	Tronc =support+armoire	48	$[405, 13, 9]$	$([126, 6, -6], [6, 1075, 0], [-6, 0, 1026])$

TAB. 4.2: *Données dynamiques dans le cas du simple support droit*

Une copie du listing du fichier est fourni en annexe p.114. Le fichier original se trouve dans : */Bipede/utils/RobotDyn/15ddl/*

4.6 Robot suspendu

On considère le cas où le robot est suspendu (initialisation, réglages). Un palan soutient la caisse du robot par le dessus. Le tronc est bloqué en position verticale. La base de référence pour la paramétrisation est le solide armoire+support.

4.6.1 Fichier *cinematique.maple*

Les paramètres de Khalil-Kleinfinger prennent les valeurs données dans le tableau suivant.

	Solide	r_i	λ_i	α_i	Θ_i
17	Support + armoire	0	0	0	$\pi/2$
18	Repère intermédiaire	0	0	0	$-q[14+1]$
16	Cardan armoire	0	0	0	0
19	Repère intermédiaire	0	0	0	$-\pi/2-q[13+1]$
15	Levier lombaires	0	0	$-\pi/2$	0
20	Repère intermédiaire	0	-0,128	0	$-q[12+1]$
14	Ex-orteils	0	0	0	0
7	Bassin	-0,110	0	0	0
6	Équerre hanche droite	0	0	0	$-q[8+1]$
5	Levier hanche droite	0	0	$-\pi/2$	$\pi/2-q[9+1]$
4	Cuisse droite	0	0	$-\pi/2$	$\pi/2-q[3+1]$
3	Tibia droit	0,410	0	0	$-q[2+1]$
2	Cardan droit	0,410	0	0	$-q[1+1]$
1	Pied droit	0	-0,120	$-\pi/2$	$-q[0+1]$
8	Équerre hanche gauche	0,220	0	0	$q[10+1]$
9	Levier hanche gauche	0	0	$-\pi/2$	$\pi/2+q[11+1]$
10	Cuisse gauche	0	0	$-\pi/2$	$\pi/2-q[7+1]$
11	Tibia gauche	0,410	0	0	$-q[6+1]$
12	Cardan gauche	0,410	0	0	$-q[5+1]$
13	Pied gauche	0	-0,120	$-\pi/2$	$q[4+1]$

TAB. 4.3: *Données cinématiques dans le cas du robot suspendu*

Une copie du listing du fichier est fourni en annexe p.117. Le fichier original se trouve dans :

`/Bipede/utils/RobotDyn/15ddlPendu/`

4.6.2 Fichier *dynamique.maple*

Les valeurs des masses (kg), les positions des centre de masse (m) et inertie ($kg.m^2$) des segments, sont données ci-après.

		m_i	$G_i 10^{-3}$	$I_{O_i} 10^{-2}$
1	Pied de support	2,34	$(59, 0, 139)$	$([0,0,0],[0,0,0],[0,0,7])$
2	Cardan jambe support	0,18	$(0, 0, 0)$	$([0,0,0],[0,0,0],[0,0,0])$
3	Tibia jambe support	5,93	$(152, -28, 0)$	$([2,-4,0],[-4,47,0],[0,0,47])$
4	Cuisse jambe support	10,9	$(160, -5, 45)$	$([6,-1,-15],[-1,82,0],[-15,0,80])$
5	Levier hanche support	0,7	$(2, 34, 0)$	$([0,0,0],[0,0,0],[0,0,0])$
6	Équerre hanche support	3,2	$(-5, 29, -107)$	$([6,0,0],[0,0,0],[0,0,5])$
7	Bassin	8,8	$(110, -12, -68)$	$([13,1,7],[1,24,-2],[7,-2,15])$
8	Équerre hanche libre	3,2	$(5, 29, -107)$	$([6,0,0],[0,5,0],[0,0,0])$
9	Levier hanche libre	0,7	$(2, 0, 34)$	$([0,0,0],[0,0,0],[0,0,0])$
10	Cuisse jambe libre	10,9	$(160, -5, -45)$	$([6,0,5],[0,42,0],[5,0,39])$
11	Tibia jambe libre	5,93	$(152, -28, 0)$	$([2,3,0],[3,21,0],[0,0,21])$
12	Cardan jambe libre	0,18	$(0, 0, 0)$	$([0,0,0],[0,0,0],[0,0,0])$
13	Pied libre	2,34	$(59, 0, 139)$	$([6,0,-2],[0,7,0],[-2,0,1])$
15	Levier lombaires	1,05	$(0, 47, -25)$	$([0,0,0],[0,0,0],[0,0,1])$
16	Cardan lombaires	0,46	$(0, 0, 46)$	$([1,0,0],[0,0,0],[0,0,1])$
17	Tronc =support+armoire	48	$(405, 13, 9)$	$([126,6,-6],[6,1075,0],[-6,0,1026])$

TAB. 4.4: *Données dynamiques dans le cas du robot suspendu*

Une copie du listing du fichier est fourni en annexe p.122. Le fichier original se trouve dans :

`/Bipede/utils/RobotDyn/15ddlPendu/`

Deuxième partie

Description du contrôleur

Chapitre 1

Présentation

1.1 Objectif du contrôleur

L'objectif est de réaliser un contrôleur qui permette au robot de poursuivre des trajectoires articulaires. Le robot devra pouvoir évoluer sur le sol et réaliser des mouvements pré-définis avec le simulateur.

1.2 Méthode

Notre travail a commencé dans un premier temps par l'étude d'une version simplifiée ne comportant que les deux jambes du robot (fig.1.3). La robustesse et l'efficacité de notre contrôleur a été éprouvée par la réalisation de différents types de démarches plane statiquement stable [RR00]. Nous nous sommes ensuite penchés sur le problème de la réalisation de mouvements posturaux pour le robot complet. Ces mouvements seront faits dans un premier temps avec le robot en simple support droit uniquement.
Le contrôleur a été implémenté à l'aide du logiciel ORCCAD, tous les programmes sources se trouvent dans :

`robotique/Bipede/orccad/`

1.3 Description

Nous distinguerons trois parties essentielles dans la description du contrôleur.

1. Communication avec le système,
2. Procédure d'initialisation,
3. Poursuite de trajectoire.

Chapitre 2

Communication avec le système

Le contrôleur doit pouvoir agir sur le robot en envoyant des signaux de commande. Il doit aussi être informé de l'état du robot par les capteurs.

Sous ORCCAD, la communication avec le système se fait par l'intermédiaire de modules qui sont spécifiques au robot, on les trouve dans le répertoire :

`/Bipede/orccad/sys`

Ces modules seront utilisés dans toutes les tâches robot *TR* pour communiquer avec le robot.

2.1 Communication avec les actionneurs du robot

Les actionneurs sont constitués de moteurs CC et de réducteurs de type Harmonic Drive ou transmissions spéciales (p.15). L'asservissement en position des articulations se fait par une commande en tension des moteurs. Il est donc nécessaire de disposer du modèle de la chaîne de transmission pour avoir la relation entre tensions moteurs et positions articulaires.

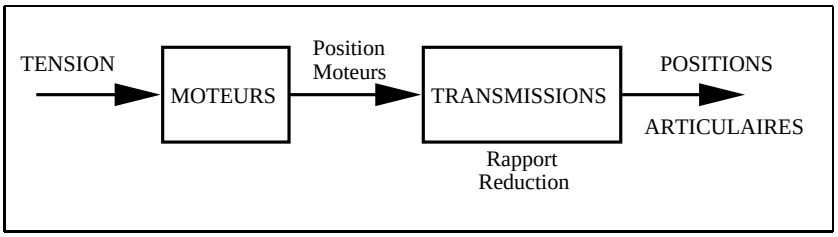


FIG. 2.1: *Chaîne de transmission reliant les tensions moteurs aux positions articulaires*

Deux modules ORCCAD représentent les actionneurs du robot :

1. *Bip15* qui représente les moteurs du robot,
2. *bipJoints15* qui représente les transmissions du robot.

2.1.1 Ressource Physique *Bip15*

Ce Module a été décrit partiellement dans [MR00]. Il permet de relier la tâche robot aux drivers du robot.

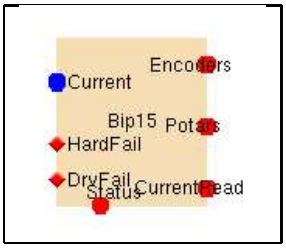


FIG. 2.2: *Ressource physique* Bip15

Remarque : D'une manière générale on notera xxxI le vecteur associé au port d'entrée xxx, et yyyO le vecteur associé au port de sortie yyy. Les vecteurs sont numérotés selon les conventions vues dans ce rapport.

Les valeurs *CurrentI* des tensions de commande arrivent sur le port d'entrée *Current*. Ces tensions sont envoyées aux 15 moteurs correspondants.

Les codeurs incrémentaux exploitent les valeurs délivrées par les résolveurs des moteurs. Les codeurs ont une précision de 4096 pas (tops) par tour. Un tour de moteur représente un déplacement de 2 à 4 degrés de l'articulation. La fonction *bipGetEncoders* calcule la position absolue du moteur (radians) à partir de la position relative en top codeurs et de la position de départ.

La fonction des potentiomètres est l'obtention de la position absolue des articulations à la mise sous tension. Ces potentiomètres sont fixés sur les articulations et alimentés en 10V. Les 15 valeurs disponibles sur le port de sortie *Potars* sont les tensions présentes au point milieu des potentiomètres. Ces tensions peuvent être utilisées pour déterminer la position absolue des articulations correspondantes. Le manque de précision des potentiomètres explique que ceux-ci ne sont utilisés que lors de l'initialisation du robot afin de connaître sa position de départ. Des problèmes techniques liés au collage des potentiomètres rendent actuellement peu sûre leur utilisation.

Le vecteur *CurrentReadO* du port *CurrentRead* fournit les tensions réellement récupérées en sortie des moteurs.

Port	**Vecteur**	**Description**
Entrée	CurrentI[15]	Tensions envoyées aux moteurs
Sortie	EncodersO[15]	Positions des moteurs
Sortie	PotarsO[15]	Tensions au point milieu des potentiomètres
Sortie	CurrentReadO[15]	Tensions en sortie des moteurs

TAB. 2.1: *Description des ports du module* Bip15

2.1.2 Module algorithmique *bipJoints15*

Le module algorithmique *bipJoints15* permet de calculer les positions articulaires et les rapports de réduction à partir des positions moteurs. Ce module fait appel à la fonction *CalcPosRatio* définie dans *utilBip15.h*.

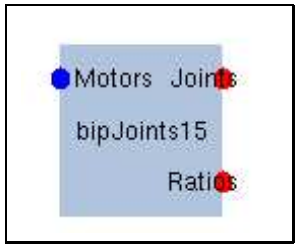


FIG. 2.3: *Module algorithmique* bipJoints15

Les sources des calculs utilisés se trouvent dans :

`/Bipede/drivers/AxesConv15`

Une copie documentée se trouve dans

`/Bipede/utils/AxesConv15`

Les programmes ont été implémentés par P.B. Wieber sur la base du travail de J.J. Parmentier [PAR99]. La commande MAPLE *CalcTabloPos.mpl* crée automatiquement le fichier *TabloPos.h* qui contient un tableau par articulation (de type vis à rouleau) qui relie les positions moteurs aux positions articulaires. Dans le cas des transmissions parallèles la position articulaire est reliée à la somme ou à la différence des positions moteurs.

- La fonction *CalcPos* effectue une interpolation sur la base du tableau *TabloPos.h* pour déterminer une approximation de la position articulaire à partir de la position moteur.

- La fonction *CalcRatio* effectue une interpolation pour déterminer une approximation du rapport de réduction à partir de la position moteur en utilisant *TabloPos.h*.
- La fonction *CalcPosRatio* effectue une interpolation sur la base du tableau *TabloPos.h* pour déterminer une approximation de la position articulaire et du rapport de réduction.

Port	**Vecteur**	**Description**
Entrée	MotorsI[15]	Position moteur
Sortie	RatiosO[21]	Rapports de réductions
Sortie	JointsO[15]	Positions articulaires absolues

TAB. 2.2: *Description des ports du module* bipJoint15

2.2 Communication avec les capteurs du robot

Le module ressource physique *Sensors15* permet de lire les informations sur les différents capteurs disponibles (capteurs de force, inclinomètre, capteur ultrason).

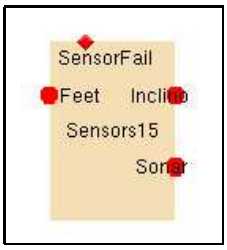


FIG. 2.4: *Ressource physique* Sensors15

Port	**Vecteur**	**Description**
Sortie	FeetO[6]	Valeurs en Newtons des informations des capteurs de force
Sortie	InclinoO[2]	Valeurs en degrés des informations de l'inclinomètre
Sortie	SonarO[1]	Valeurs en volts lues sur le capteur ultrason

TAB. 2.3: *Description des ports du module* Sensors15

Chapitre 3

Procédure d'initialisation

3.1 Présentation

L'initialisation du système fait l'objet d'une procédure robot ORCCAD *ProcInit* (fig.3.1). Cette procédure est définie dans :

`robotique/Bipede/orccad/user15/Spec/RobotProcedure/ProcInit.PC`

```
[
 % Component bipInit
 [
  % call bipInit_fileparameter()(?bipInit_Start, "filename");
   call bipInit_parameter()(?bipInit_Start );
   call bipInit_controler()(?bipInit_Start);
   emit START_bipInit;
   await
     case BF_bipInit
     case T3_bipInit do exit T3_controle_ProcInit
   end await
  ]
]
```

FIG. 3.1: *Description Esterell de la procédure* ORCCAD **ProcInit**

Cette procédure met en jeu une seule tâche robot *bipInit* :

`robotique/Bipede/orccad/user15/Spec/RobotTasks/bipInit`

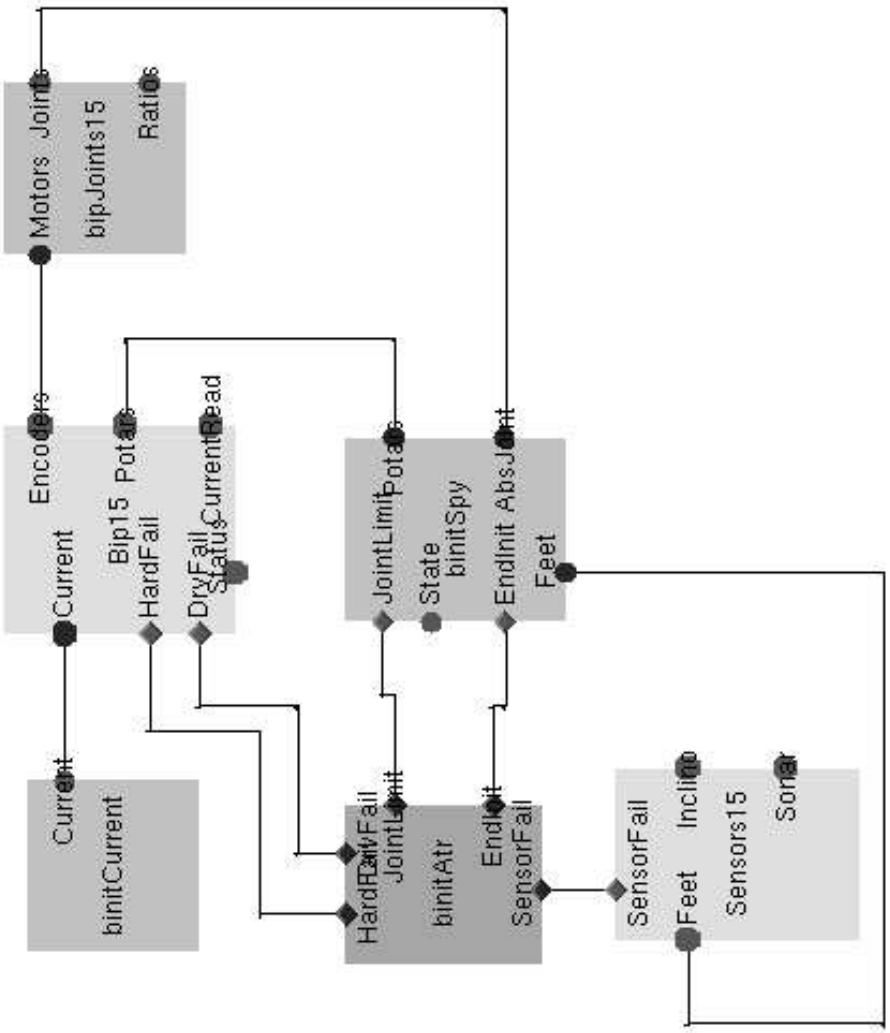


FIG. 3.2: *Tâche robot* ORCCAD **bipInit**

Les modules intervenant dans la *TR* se trouvent dans le répertoire :

`robotique/Bipede/orccad/user15/Spec/Modules`

L'objectif de l'initialisation, outre le démarrage du système est de déterminer la position de départ absolue du robot. Deux solutions sont possibles, l'utilisation des potentiomètres ou le positionnement du robot dans une position connue.

3.1.1 Initialisation automatique

L'initialisation automatique consiste à estimer la position du robot au démarrage du système en utilisant les informations fournies par les potentiomètres. La méthode présentée ici a été utilisée pour la version 8 ddl du robot mais jamais pour la version 15 ddl.

La méthode consiste à approximer la courbe reliant les positions articulaires aux tensions des potentiomètres par une droite Δ_i. On utilise ensuite la pente de la droite et l'offset des capteurs pour déduire une position articulaire à partir des tensions lues.

Le répertoire :

`robotique/Bipede/utils/CalibPotars15`

contient les outils logiciels pour la calibration des potentiomètres du robot. Un guide est disponible sur le site privé :

`http://www.inrialpes.fr/iramr/private`

La procédure de calibration devra être faite régulièrement pour s'assurer qu'il n'y a pas de problème au niveau des potentiomètres. Les différentes calibrations font l'objet d'un historique :

`robotique/Bipede/utils/CalibPotars15/Mesures/MesDDMMYY`

La procédure consiste à récupèrer expérimentalement les valeurs des positions articulaires et des tensions potentiomètres sur plusieurs aller retour de l'articulation considérée sur un mouvement d'amplitude la plus importante possible. P.B. Wieber a développé des programmes MAPLE :

`robotique/Bipede/utils/CalibPotars15/Mesures/Controle/potarsi.mws`

permettant de traiter les données en calculant par la méthode des moindres carrés la pente et l'offset de la droite Δ_i. Le script *Calibration.sh* récupère l'offset et la pente pour chaque articulation et les regroupe dans le fichier *Potars15.h* et *BipPotars.h* :

`robotique/Bipede/utils/CalibPotars15/Mesures/Potars15.h`
`robotique/include/BipPotars15.h`

3.1.2 Initialisation manuelle

Le but de l'initialisation est de déterminer la position du robot au démarrage pour en déduire la position absolue en fonction des informations des codeurs des moteurs. L'initialisation manuelle consiste à positionner dans une position parfaitement connue, par exemple la position zéro (p.33).

3.2 Description des modules intervenant dans la *TR*

3.2.1 Module algorithmique *binitCurrent*

Durant la phase d'initialisation le robot ne doit pas bouger. Le module *binitCurrent* envoie un vecteur *CurrentO* ne contenant que des zéros sur le port *Current* de la ressource physique *bip15*.

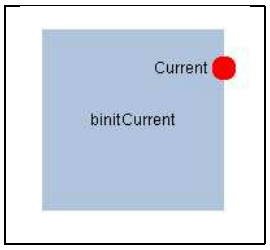


FIG. 3.3: *Module algorithmique* binitCurrent

Port	**Vecteur**	**Description**
Sortie	CurrentO[15]	Valeurs de tension moteurs nulles

TAB. 3.1: *Description des ports du module* binitCurrent

3.2.2 Module algorithmique *binitSpy*

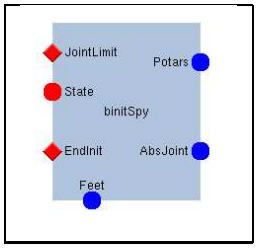


FIG. 3.4: *Module algorithmique* binitSpy

L'initialisation a une durée T_3 et comporte différentes étapes successivtes.

Étape $t < T_1$

La première partie de l'initialisation est consacrée à la mesure des informations des potentiomètres et des capteurs de force. Ces mesures sont ajoutées les unes aux autres.

Étape $t = T_1$

Le module demande ensuite à l'utilisateur de choisir entre init manuel et init automatique. Nous rappelons que l'init automatique n'est pas encore utilisable sur la version 15 ddl, nous en présentons néanmoins le principe.
Dans le cas de l'init automatique, on utilise la moyenne des mesures effectuées sur les potentiomètres *PotarsI* (en volts). Le module utilise les valeurs de la pente et de l'offset de la droite Δ disponibles dans *BipPotars.h*. La pente permet de convertir la tension en position articulaire qui ajoutée à l'offset du potentiomètre fournit une estimation de la position de l'articulation *StateO*. On la traduit également en position moteur *offset_joint* à l'aide de la fonction *bipUtilCalcMotors*.
Dans le cas de l'init manuel, nous l'avons dit, le robot est positionné dans une position déterminée *offset_joint*. Nous avons choisi de placer le robot dans la position zéro. Dans ce cas, le robot est en butée mécanique au niveau des genoux. Un message avertit l'utilisateur que la procédure de calcul de la position absolue est terminée et l'utilisateur doit retirer manuellement les genoux des butées avant de poursuivre la procédure.
On calcule dans les deux cas la moyenne des valeurs relevées sur les capteurs d'effort, cette moyenne *offset_feet* servira d'offset pour les capteurs.

Étape $t = T_2$

On affiche ensuite les valeurs potentiomètres, positions absolues et valeurs capteurs de force.

Étape t = T_3

Puis on fait successivement des vérifications de non dépassement des butées mécaniques. Si tout s'est déroulé correctement on termine l'init.

Par la suite, les valeurs *offset_joint* seront récupérées dans les différentes tâches robot pour le calcul des positions absolues à partir des données des codeurs.

Port	Vecteur	Description
Entrée	PotarsI[15]	Mesures en volts des potentiomètres
Entrée	FeetI[6]	Mesures en Newtons des capteurs de force
Entrée	AbsJointI[15]	Positions relatives des articulations
Sortie	StateO[15]	positions absolues des articulations

TAB. 3.2: *Description des ports du module* binitSpy

3.2.3 Module automate *binitAtr*

Le module *binitAtr* reçoit les évènements générés au niveau des autres modules et en déduit la conduite à tenir (selon le type d'événement).

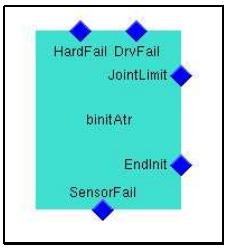


FIG. 3.5: *Module automate* binitAtr

Évènement	Type	Description
HardFail	T3	Problème matériel sur la ressource physique
JointLimit	T3	Butée logicielle atteinte
SensorFail	T3	Problème sur un capteur
EndInit	Post-Condition	Fin de la tâche d'initialisation

TAB. 3.3: *Événements associés au module* binitAtr

Chapitre 4

Procédures de poursuite de trajectoires

4.1 Procédure ProcPts

La procédure*ProcPts* met en jeu deux tâches robot *bipInit* et *bipPts* qui s'enchaînent (fig.4.1). Nous avons déjà décrit la phase d'initialisation, nous ne décrirons ici que la tâche robot *bipPts* qui se trouve :

```
orccad/user/Spec/RobotTasks/bipPts
```

Cette tâche commence lorsque la post-condition associée à la *TR* d'initialisation est satisfaite. Les modules associés (fig.4.2) se trouvent dans le répertoire :

```
orccad/user/Spec/Modules
```

4.1.1 Communication avec le système

Cette partie a été développée dans le chapitre précédent. Le module *Bip15* met à disposition la position absolue des moteurs. Cette position est calculée à partir des informations relatives des codeurs et de la connaissance de la position de départ qui a été déterminée à l'initialisation. Le module *Sensors15* fournit toutes les informations disponibles au niveau des capteurs du système. Le module *bipJoint15* permet de déterminer les rapports de réduction relatifs à une position donnée, ainsi que la position articulaire absolue de chaque articulation à partir de la position absolue des moteurs.

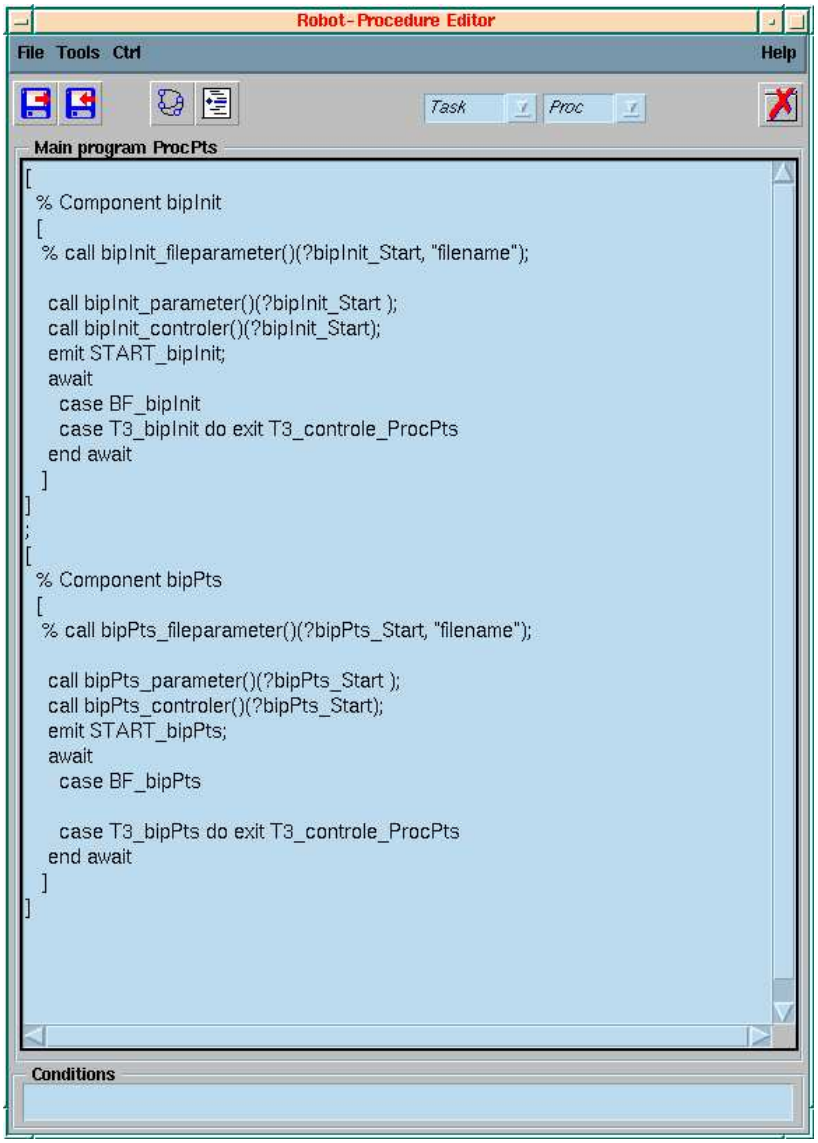


FIG. 4.1: *Description Esterell de la procédure* ORCCAD **ProcPts**

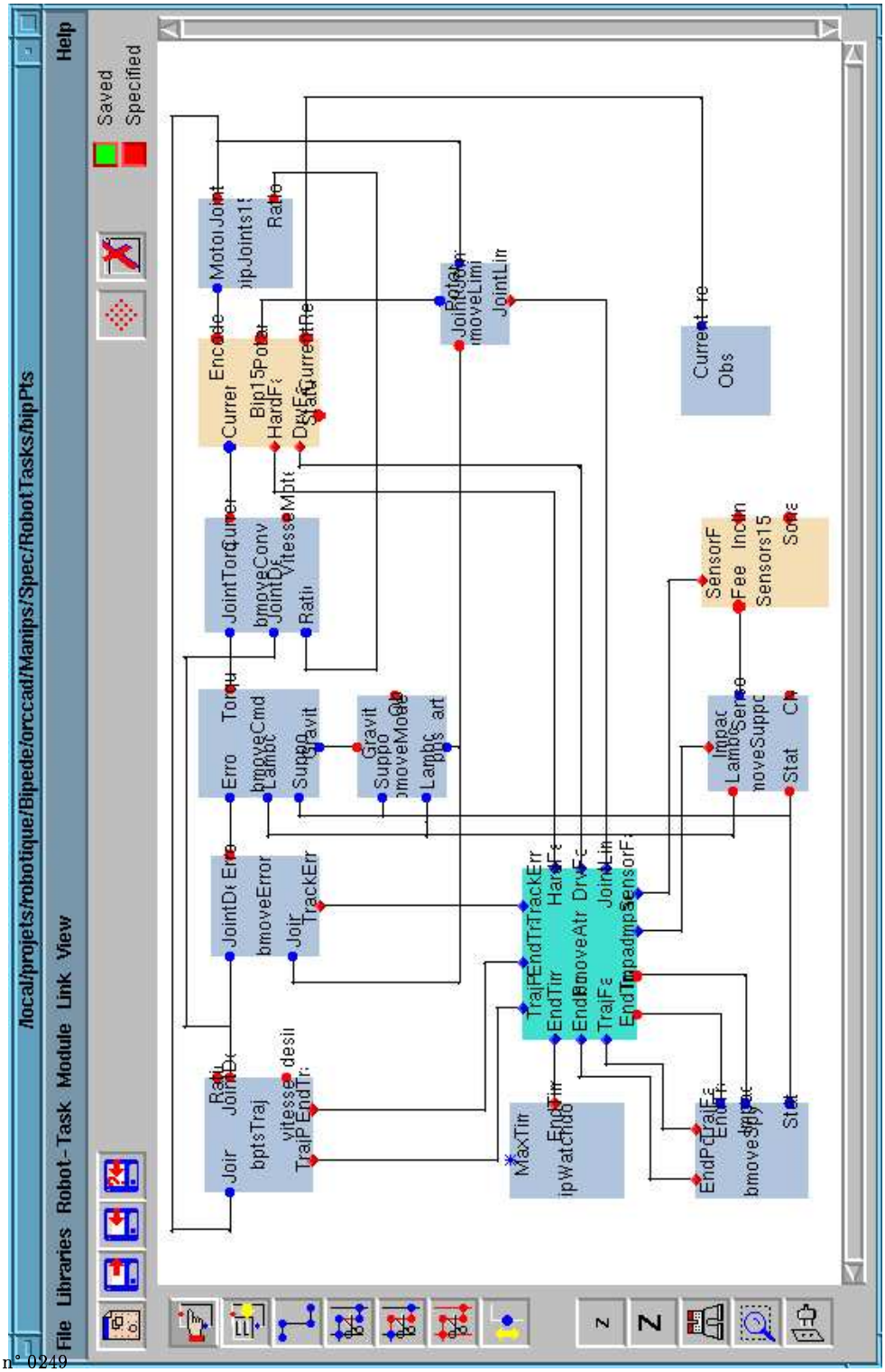


FIG. 4.2: *Tâche robot* ORCCAD **bipPts**

4.1.2 Détection de support

Lorsque le robot évolue sur le sol, la détection de support joue un rôle très important. Elle permet de savoir dans quelle phase de support se trouve le robot et de détecter les éventuels impacts avec le sol ou un obstacle.
De plus on peut également observer les déplacements du centre de pression et vérifier que l'équilibre est conservé.
La détection de support s'appuie sur les informations fournies par les capteurs d'effort par l'intermédiaire du module *Sensors15*.
Les phases de support considérées sont : le double support, le simple support droit, le simple support gauche, les transitions. Les transitions correspondent aux cas ou la somme des forces de pression est inférieure au poids total du robot, c'est-à-dire lorsque le robot n'est que partiellement posé sur sol (en cours de dépose ou en cours de levage).

Méthode

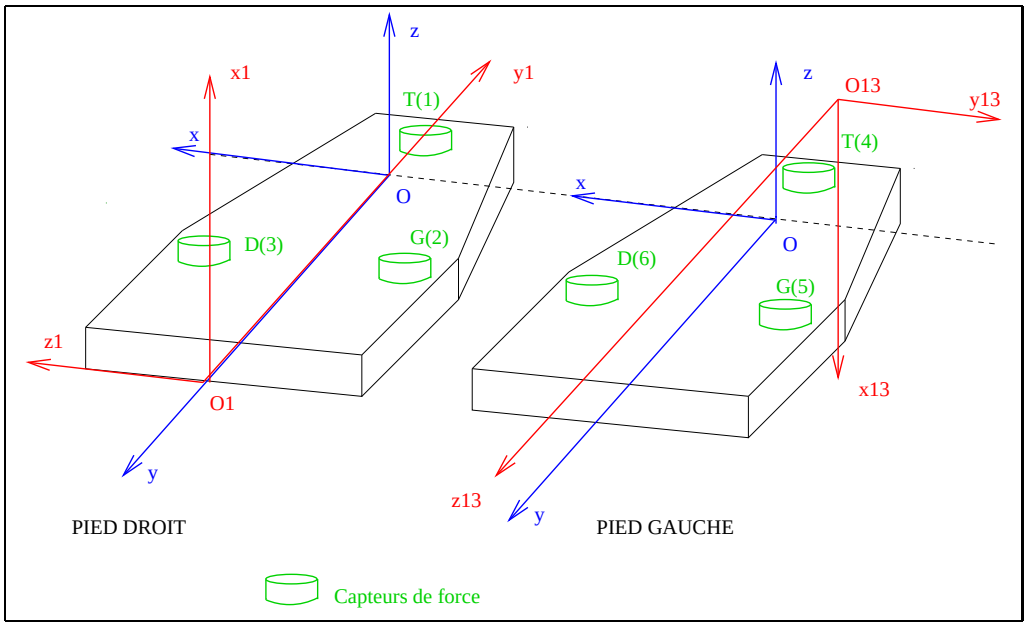


FIG. 4.3: *Position des capteurs de force, axes associés et numérotation*

Sur chaque pied, les capteurs sont situés au talon, à l'avant droit et à l'avant gauche (fig.4.3), ils sont numérotés de 1 à 6. Chaque capteur délivre une valeur qui correspond à la force exercée par le tibia sur le capteur.

On dispose donc pour chaque pied de trois valeurs :

- F_T la force exercée par le tibia sur le capteur du talon,
- F_G la force exercée par le tibia sur le capteur avant droit,
- F_D la force exercée par le tibia sur le capteur avant gauche.

Les positions des capteurs de chaque pied sont exprimées dans un repère centré en la projection O de la cheville sur la semelle et d'axes tels que représentés sur la figure 4.3.
on notera les coordonnées des capteurs :

- $C_T = (l_T, L_T)$ pour le capteur du talon,
- $C_D = (l_D, L_D)$ pour le capteur avant droit,
- $C_G = (l_G, L_G)$ pour le capteur avant gauche.

Nous faisons l'hypothèse que les semelles se posent toujours à plat sur un sol horizontal afin de simplifier les calculs. Les calculs complets sont fournis en annexe page 135.

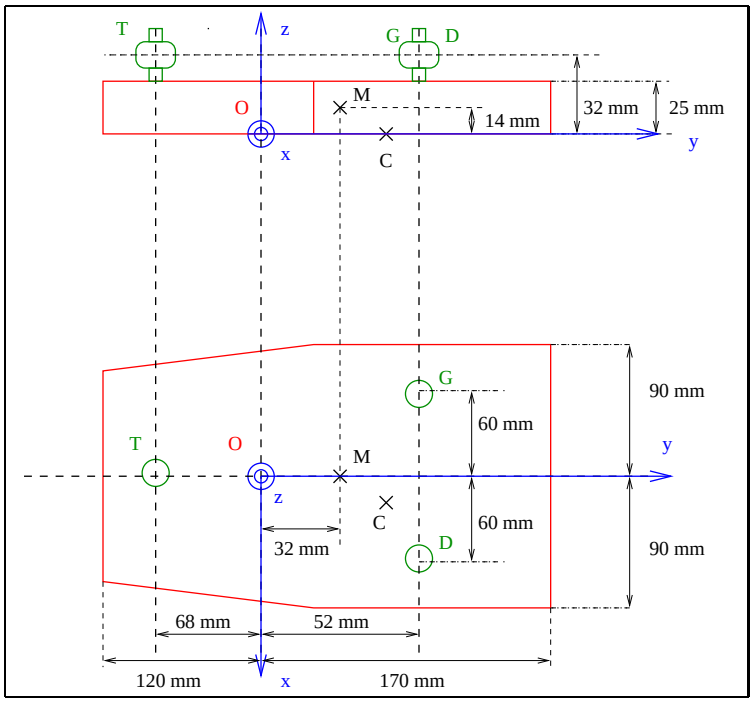


FIG. 4.4: *Mesures relatives aux pieds du robot*

Force de réaction au sol

Nous pouvons calculer la composante verticale F_P de la force de pression qui s'applique sur le pied de support. Elle permet de savoir dans quelle phase de support de la marche on se trouve.

$$Fp = -F_T - F_G - F_D \tag{4.1}$$

Moment de la force de réaction

On peut également déterminer la projection suivant les axes longitudinaux et latéraux du moment de pression (fig.4.5).

$$\begin{cases} Mp_l = -L_T \times F_T - L_G \times F_G - L_D \times F_D \\ Mp_L = -l_T \times F_T - l_G \times F_G - l_D \times F_D \end{cases} \tag{4.2}$$

Centre de pression

Le centre de pression est le point en lequel le moment des efforts de pression est nul.

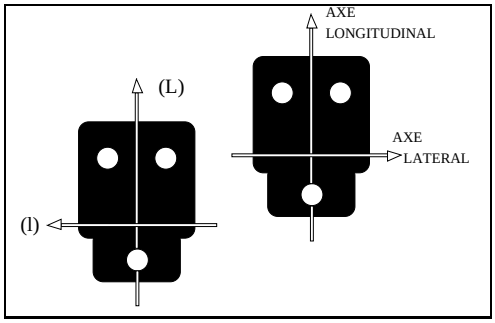


FIG. 4.5: *Axes pour le repérage du centre de pression*

La position du centre de pression s'obtient aisément :

$$Cp = \begin{pmatrix} Cp_l \\ Cp_L \end{pmatrix} = \begin{pmatrix} -\frac{Mp_L}{Fp} \\ \frac{Mp_l}{Fp} \end{pmatrix} \tag{4.3}$$

Module algorithmique *bmoveSupport*

Le module *bmoveSupport* calcule la force de pression sur chaque pied, ainsi que la position CM du centre de pression.

Nous utilisons une fonction de détection d'impacts pour chaque pied SD et SG. Cette fonction correspond à la somme cumulée de la force de pression sur un pied donné, corrigée du bruit moyen.

$$F_d = force_{pression} - \frac{\Delta\mu}{2,0}$$

Lorsque le pied est au dessus du sol, les informations des capteurs sont très bruitées, en revanche la somme cumulée de la force de pression est une droite peu perturbée.
On observe cette droite au cours du temps, si l'on détecte une modification de la pente, et si cette modification est significative on conclut à un impact.
Lorsqu'un impact est détecté, on observe les valeurs des forces de pression sur chaque pied pour déterminer dans quelle phase de support on se trouve connaissant la phase de support précédent l'impact. On génère également l'événement impact. En cas de phase de transition ou de double support on calcule la valeur $LambdaO_0$ correspondant au rapport entre la force de pression sur le pied droit et le poids total du robot et $LambdaO_1$ correspondant au rapport entre la force de pression sur le pied gauche et le poids total du robot.

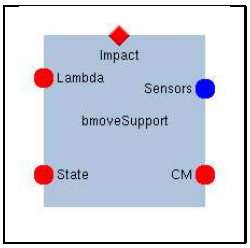


FIG. 4.6: *Module algorithmique* bmoveSupport

Port	**Vecteur**	**Description**
Entrée	SensorsI[6]	Valeurs des capteurs de force
Sortie	StateO[1]	Support en cours
Sortie	LambdaO[1]	Poids supporté par le pied droit
Sortie	CMO[2]	Position latérale et longitudinale du centre de pression

TAB. 4.1: *Description des ports du module* bmoveSupport

Les valeurs de $StateO$ sont :

- 0 : aucun pied au sol,
- 1 : simple support droit,
- 2 : simple support gauche,

- 3 : double support,
- 4 : robot en cours de dépose ou de levage.

Ces valeurs sont modifiées suite à la détection d'impact uniquement.

4.1.3 Trajectoires de référence et consigne

Méthode

Les fichiers utilisés par le module contiennent des points de passage souhaités pour les différentes articulations. On calcule ensuite un polynôme de degré 5 reliant les points de passage et assurant la vitesse désirée à chaque point de passage. On peut ainsi définir des mouvements de façon très simple et rapide pour le robot lorsqu'il est suspendu (réglages de gains, rodage...). Le format des fichiers est :

```
# Trajectoire
LOOP X Y Z
TIME t
POSR a b c d
VELR 0 0 0 0
POSL e f g h
VELL 0 0 0 0
POSB i j k l m
VELB 0 0 0 0 0
POST n o
VELT 0 0
```

Les trois valeurs qui suivent le mot clef LOOP sont : le nombre de répétitions de la trajectoire, le point de la trajectoire sur lequel on recommence les cycles et la durée de ralliement du premier point d'un cycle à partir du dernier point du cycle précédent.
Les différentes valeurs ne doivent pas dépasser les limites articulaires logicielles. Le mot clef TIME est suivi de la durée de ralliement d'un point à partir du point précédent. Les mots clefs POSR, POSL, POSB, POST sont suivis respectivements des valeurs désirées pour les positions articulaires de la jambe droite, de la jambe gauche, du bassin et du tronc. Les mots clefs VELR, VELL, VELB, VELT sont suivis des valeurs désirées de vitesses pour les différentes positions désirées. Cette fonctionnalité n'ayant pas été testée, les valeurs des vitesses seront toutes nulles.
Par exemple on souhaite que la hanche de la jambe droite partant de la position initiale se place dans la position -1,0 radians en 5 secondes, revienne à 0,5 radians en 2 secondes puis à nouveau à sa position nulle en 3 secondes, et ce, 4 fois. Les autres articulations demeurant immobiles.

```
# Trajectoire essai
LOOP 4 2 5.0
TIME 5.0
POSR 0.0 0.0 0.2 -1.0
VELR 0.0 0.0 0.0 0.0
POSL 0.0 0.0 0.2 0.0
VELL 0.0 0.0 0.0 0.0
POSB 0.0 0.0 0.0 0.0 0.0
VELB 0.0 0.0 0.0 0.0 0.0
POST 0.0 0.0
VELT 0.0 0.0
TIME 2.0
POSR 0.0 0.0 0.2 0.5
VELR 0.0 0.0 0.0 0.0
POSL 0.0 0.0 0.2 0.0
VELL 0.0 0.0 0.0 0.0
POSB 0.0 0.0 0.0 0.0 0.0
VELB 0.0 0.0 0.0 0.0 0.0
POST 0.0 0.0
VELT 0.0 0.0
TIME 3.0
POSR 0.0 0.0 0.2 0.0
VELR 0.0 0.0 0.0 0.0
POSL 0.0 0.0 0.2 0.0
VELL 0.0 0.0 0.0 0.0
POSB 0.0 0.0 0.0 0.0 0.0
VELB 0.0 0.0 0.0 0.0 0.0
POST 0.0 0.0
VELT 0.0 0.0
```

4.1.4 Module algorithmique *bptsTraj*

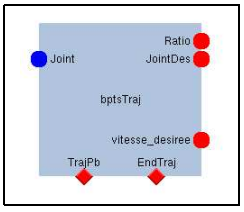


FIG. 4.7: *Module algorithmique* bptsTraj

Port	Vecteur	Description
Entrée	JointI[15]	positions articulaires réelles
Sortie	JointDesO[15]	positions articulaires désirées

TAB. 4.2: *Description des ports du module* bptsTraj

4.1.5 Contrôle-commande

Le premier prototype de BIP2000 dispose d'une puissance de calcul limitée (fig.4.8). Afin de maintenir une période d'échantillonage suffisamment faible (8 ms) nous avons utilisé des lois de commande les moins coûteuses possibles en temps de calcul.

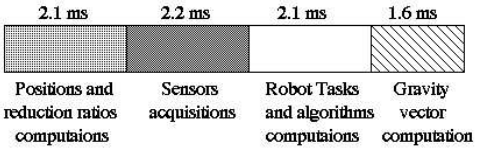


FIG. 4.8: *Durées de calculs relatives*

Méthode

Nous souhaitons poursuivre des trajectoires articulaires q_d. Ces trajectoires pourront être calculées a priori ou en-ligne. Pour cela, nous utilisons une loi de commande de type proportionnel-dérivé associé à une compensation de gravité et une compensation de frottement. On calcule dans un premier temps la partie PD+Gravité pour déterminer les couples articulaires Γ :

$$\Gamma = K_p(q_d - q) + K_v(\dot{q}_d - \dot{q}) + \hat{G}(q) \tag{4.4}$$

avec K_p et K_v des vecteurs de constantes positives, q_d la trajectoire désirée que l'on poursuit et $\hat{G}$ une estimation du vecteur gravité.

Les moteurs que nous utilisons sont commandés en tension U. Nous avons transformé Γ en vecteur de tensions U' grâce aux modèles des moteurs [PAR99] et nous avons introduit une compensation des frottements :

$$U = U' + F.sign(\dot{\theta}_m) \tag{4.5}$$

où F est le vecteur des constantes de frottements et $\dot{\theta}_m$ le vecteur des vitesses moteurs.

Compensation des frottements secs

Nous avons évalué expérimentalement les frottements secs en bout de transmission de chaque moteur (fig.7). Le membre en aval du moteur considéré est lâché, il s'arrête dans une

position où le frottement est égal à la gravité Kf_i. On peut également envoyer progressivement des couples à l'articulation jusqu'à ce qu'elle bouge.

Pour plus d'informations sur l'architecture et les algorithmes de la commande du robot, se reporter à [AZE00].

Module algorithmique *bmoveError*

On a besoin de l'erreur entre position désirée et position réelle ainsi que vitesse désirée et vitesse réelle.

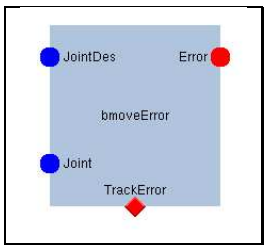


FIG. 4.9: *Module algorithmique* bmoveError

Le module effectue la différence entre positions articulaires désirées q_d et réelles q et la met à disposition sur son port de sortie *Error*. Si les erreurs sont supérieures aux limites acceptables que l'on s'est fixées on génère l'événement *TrackError*.

Port	**Vecteur**	**Description**
Entrée	JointDesI[30]	Positions articulaires désirées suivies des vitesses désirées
Entrée	JointI[15]	Positions articulaires absolues réelles
Sortie	ErrorO[15]	Erreur en position du robot

TAB. 4.3: *Description des ports du module* bmoveError

Module algorithmique *bmoveModel*

Dans notre commande on doit également utiliser le vecteur gravité du modèle. Comme on l'a vu (p.45), le modèle dépend de la phase de support considérée. On dispose du modèle pour le robot suspendu :

`robotique/Bipede/utils/ModelesBip15/gravite_pendu.*`

et pour le simple support droit :

`robotique/Bipede/utils/ModelesBip15/gravite.*`

En fonction de la valeur de *SupportI* on connaît la phase de support et on en déduit une estimation du vecteur gravité $\hat{G}$.

- Dans le cas du simple support droit, $\hat{G}$=*gravite*.
- Dans le cas du robot suspendu, $\hat{G}$=*gravite_pendu*.
- Dans le cas du simple support gauche on récupère *gravite* et on réécrit les lignes du vecteur pour se ramener au cas pied gauche au sol (non implémenté pour la version 15 ddl du robot).

Les démarches expérimentées sur la version 8 ddl étaient lentes, symétriques et présentaient des phases de double-support très courtes. Lors du double-support, on avait un transfert linéaire du poids d'un pied sur l'autre. On suppose dans le cas où l'on est en train de soulever ou de poser le robot que le poids arrive de manière linéaire sur le pied au sol. Cela revient à dire que l'on passe linéairement d'un modèle à l'autre. On écrit donc que :

$$\hat{G} = Lambda0 * G0 + Lambda1 * G1$$

où Lambda0 correspond au rapport entre le poids supporté par le pied droit et le poids total du robot. Et Lambda1 = 1-Lambda0.

Remarque : Attention tous les algorithmes de la version 8 ddl n'ont pas été implémentés sur la version 15 ddl qui n'était censé faire que de la posture sur son pied droit

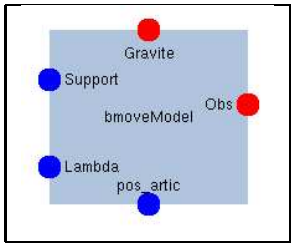


FIG. 4.10: *Module algorithmique* bmoveModel

Port	Vecteur	Description
Entrée	SupportI[1]	Phase de support
Entrée	LambdaI[2]	Ratios des poids sur pieds / poids total
Entrée	pos_articI[15]	Positions articulaires absolues
Sortie	GraviteO[15]	15 composantes du vecteur de gravité

TAB. 4.4: *Description des ports du module* bmoveModel

Module algorithmique *bmoveCmd*

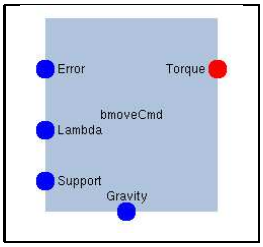


FIG. 4.11: *Module algorithmique* bmoveCmd

On dispose de jeux de gains : un jeu pour le cas du robot suspendu et un jeu pour le cas du robot au sol. En fonction de la phase de support *SupportI* on choisit un jeu. Dans le cas de la transition on utilise une combinaison linéaire des deux jeux de gain en fonction du poids qui se déplace sur le pied de support *LambdaI*.

On calcule l'erreur en vitesse en fonction de l'erreur en position *ErrorI* par un filtre.

$$\dot{e}(t) = 0,4.\dot{e}(t-1) + 0,6.\frac{(e(t) - e(t-1))}{dt}$$

On dispose alors de tous les éléments pour déterminer le couple de commande *TorqueO* relatif au PD+compensation de gravité *GravityI*.

Port	**Vecteur**	**Description**
Entrée	SupportI[1]	Phase de support
Entrée	LambdaI[2]	Ratios des poids sur pieds / poids total
Entrée	ErrorI[15]	Erreurs entre positions désirées et réelles
Entrée	GravityI[15]	Vecteur gravité estimé
Sortie	TorqueO[15]	Couples articulaires

TAB. 4.5: *Description des ports du module* bmoveCmd

Module algorithmique *bmoveConv*

Le module *bmoveConv* convertit les couples commande *JointTorqueI* en tensions moteurs à partir des rapports de réduction des transmissions *RatioI* et des constantes moteurs fournies par les constructeurs. On rappelle les relations entre positions moteurs Θ et positions articulaires q, ainsi que la relation entre couples moteurs C_m et articulaires C_a :

$$J\dot{\Theta} = \dot{q}$$
$$C_m^T\dot{\Theta} = C_a^T J\dot{\Theta}$$

D'où :

$$C_m = J^T C_a$$

où J est la matrice des rapports de réduction. Dans le cas des transmissions de type parallèles :

$$J = \begin{pmatrix} ratio_{frontal-exterieur} & ratio_{frontal-interieur} \\ ratio_{sagittal-exterieur} & ratio_{sagittal-interieur} \end{pmatrix}$$

On ajoute alors la compensation de frottements, on obtient la tension de commande des moteurs *CurrentO*. La compensation utilise le signe de la vitesse moteur désirée *VitesseMoteurO* qui est calculée à partir des positions articulaires *JointDes* désirées à l'aide d'un filtre identique à celui utilisé pour le calcul de l'erreur en vitesse.

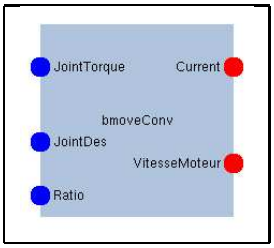


FIG. 4.12: *Module algorithmique* bmoveConv

Port	**Vecteur**	**Description**
Entrée	JointTorqueI[15]	Couples articulaires
Entrée	JointDesI[15]	Positions articulaires désirées
Entrée	RatioI[21]	Rapports de réduction
Sortie	CurrentO[15]	Tensions moteurs
Sortie	VitesseMoteurO[15]	Vitesses moteurs

TAB. 4.6: *Description des ports du module* bmoveConv

4.1.6 Surveillance

Module algorithmique *bipWatchdog*

Le module *bipWatchdog* génère l'événement *EndTime* lorsque le temps courant atteint la valeur *MaxTime*. Il s'agit d'une sécurité qui garantit que la procédure aura une fin.

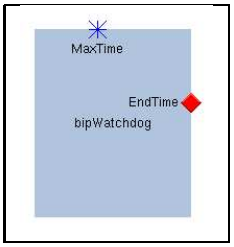


FIG. 4.13: *Module algorithmique* bipWatchdog

Le code correspondant au module est disponible :

`orccad/sys/Spec/Modules/bipWatchdog`

Module algorithmique *bmoveLimit*

Le module *bmoveLimit* génère l'évènement *JointLimit* dans le cas où une position articulaire courant atteint sa butée logicielle. De plus l'appel à la fonction externe *bipInhibit* permet d'inhiber les amplificateurs de puissance. On rappelle que les butées sont définies dans utilBip15.h.

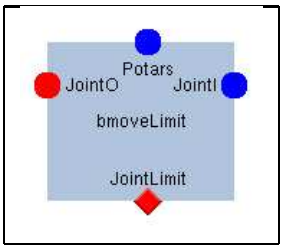


FIG. 4.14: *Module algorithmique* bmoveLimit

Port	**Vecteur**	**Description**
Entrée	JointI[15]	Positions articulaires courantes
Entrée	PotarsI[15]	Tensions des potentiomètres
Sortie	JointO[15]	Positions articulaires courantes

TAB. 4.7: *Description des ports du module* bmoveLimit

Module algorithmique *bmoveSpy*

Le module *bmoveSpy* sert de relais aux événements de type T1. Un certain nombre de commandes ont été préparées mais non implémentées et validées. Notamment, on souhaite modifier le code pour que le robot ne commence sa phase de poursuite de trajectoires que lorsqu'il est parfaitement posé sur le sol. De même on souhaite arrêter la *TR* en cas d'impact détecté. Actuellement ce module n'agit pas sur le comportement de la *TR*.

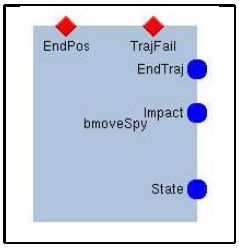


FIG. 4.15: *Module algorithmique* bmoveSpy

Port	Vecteur	Description
Entrée	StateI[1]	Phase de support courante
Entrée	ImpactI[1]	Événement impact détecté
Entrée	EndTrajI[1]	Événement fin de trajectoire détecté

TAB. 4.8: *Description des ports du module* bmoveSpy

Les vecteurs *ImpactI* et *EndTrajI* indiquent si les événements de type T1 *Impact* et *EndTraj* ont été ou non générés.

Module algorithmique *Obs*

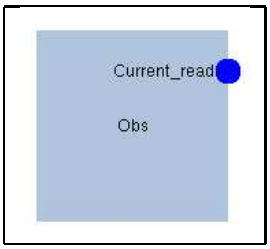


FIG. 4.16: *Module algorithmique* Obs

Le module Obs sert uniquement à récupérer les tensions en sortie des moteurs.

Module automate *bmoveAtr*

Le module automate gère les événements survenant au cours du déroulement de la procédure.

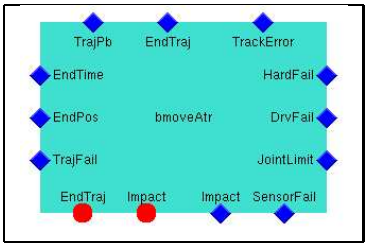


FIG. 4.17: *Module automate* bmoveAtr

Événement	**Type**	**Description**
HardFail	T3	Problème matériel sur la ressource physique
SensorFail	T3	Problème sur un capteur
DrvFail	T3	Problème de driver
EndTime	T3	Délai MaxTime atteint
JointLimit	T3	Butée logicielle atteinte
TrackError	T3	Erreur de position maximale atteinte
TrajPb	T3	Problème de trajectoire
TrajFail	T3	Problème de trajectoire
Impact	T1	Impact détecté
EndTraj	T1	Fin de trajectoire
EndPos	T3	Trajectoire désirée terminée
EndInit	Post-Condition	Fin de la tâche d'initialisation

TAB. 4.9: *Événements associés au module* bmoveAtr

4.2 Procédure ProcMove

Cette procédure met en jeu deux tâches robot *bipInit* et *bipMove* qui s'enchaînent (fig.4.18). Le seul module différent de cette procédure par rapport à *ProcPts* est le module trajectoires.

```
% Component bipInit
[
 % call bipInit_fileparameter()(?bipInit_Start, "filename");
  call bipInit_parameter()(?bipInit_Start );
  call bipInit_controler()(?bipInit_Start);
  emit START_bipInit;
  await
    case BF_bipInit
    case T3_bipInit do exit T3_controle_ProcMove
  end await
]


% Component bipMove
[
 % call bipMove_fileparameter()(?bipMove_Start, "filename");
  call bipMove_parameter()(?bipMove_Start );
  call bipMove_controler()(?bipMove_Start);
  emit START_bipMove;
  await
    case BF_bipMove
    case T3_bipMove do exit T3_controle_ProcMove
  end await
]
```

FIG. 4.18: *Description Esterell de la procédure* ORCCAD **ProcMove**

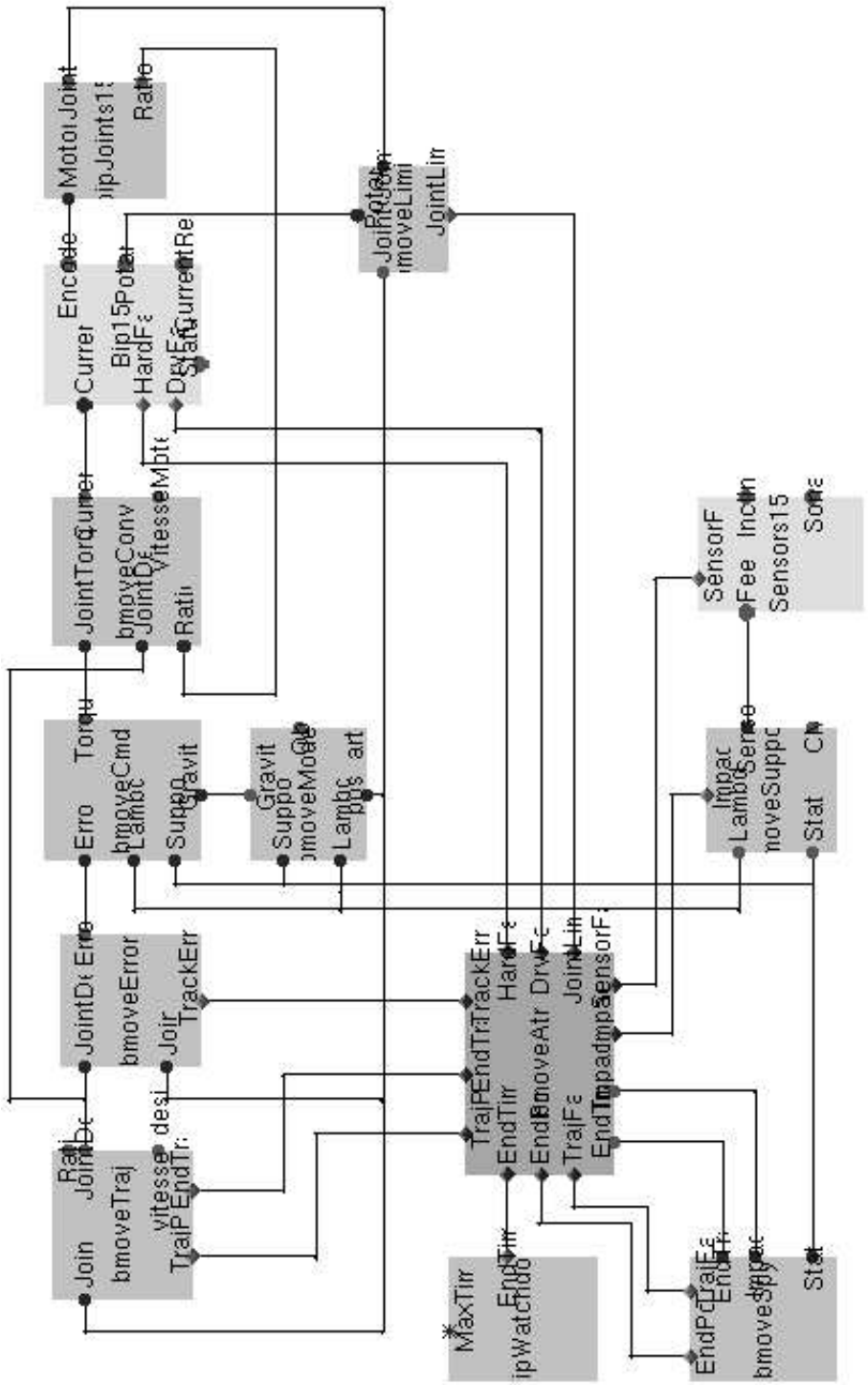


FIG. 4.19: *Tâche robot* ORCCAD **bipmove**

4.2.1 Trajectoires de référence et consigne

Méthode

On utilise deux formats de fichiers de trajectoires : V0 et V1. Les formats de type V0 contiennent les positions articulaires désirées à chaque instant alors que les fichiers V1 contiennent des points de passage entre lesquels on interpole pour trouver les positions à chaque instant.

Trajectoires polynômiales V1

Le format des fichiers est :

```
# Format
V1
#    NbArt
15
# NbPts
5
# Cycle
4
# Rebouclage
2
# Temps
5.0
#
# Position Repos
#
Temps 5.0
Jambe_D 0.0 -0.1 0.2 0.04
Jambe_G 0.0 -0.1 0.2 0.04
Bassin 0.0 0.0 0.0 0.0
Lombaires 0.0 0.0 0.0
#
# Genou droit plie
#
Temps 5.0
Jambe_D 0.0 -0.1 0.2 0.04
Jambe_G 0.0 -0.1 0.2 0.04
Bassin 0.0 0.0 0.0 0.0
Lombaires 0.0 0.07 0.0
```

où :

- **Format** : déclare le format de fichier,
- **NbArts** : déclare le nombre d'articulations du robot considéré,
- **NbPts** : déclare le nombre de points de passages définis dans le fichier,
- **Cycle** : déclare le nombre de fois que la trajectoire sera rejouée,
- **Rebouclage** : spécifie sur quel point de passage on reboucle la trajectoire pour un nouveau cycle lorsque le point de passage final a été atteint.
- **Temps** : déclare le temps alloué au robot pour atteindre le premier point de passage à partir de sa position initiale.
- **Temps xx** : déclare le temps alloué au robot pour atteindre le point de passage considéré à partir du point précédent.
- **Jambe_D** : liste les positions articulaires des cheville frontale, cheville sagittale, genou, hanche sagittale de la jambe droite,
- **Jambe_G** : liste les positions articulaires des cheville frontale, cheville sagittale, genou, hanche sagittale de la jambe gauche,
- **Bassin** : liste les positions articulaires des hanches verticale et frontale jambe droite et hanches verticale et frontale de la jambe gauche,
- **Lombaires** : liste les positions articulaires des lombaires verticale, frontale et sagittale.

On calcule ensuite un polynôme de degré 5 reliant les points de passage et assurant une vitesse nulle à chaque point de passage. On peut ainsi définir des mouvements de façon très simple et rapide pour le robot lorsqu'il est suspendu (réglages de gains, rodage...).

Trajectoires V0 Nous ne détaillerons pas ici la méthode de génération de trajectoires, se reporter à [RR00] et [WIE01] pour plus de détails.

Le principe consiste à décrire des postures à l'aide de fonctions de sortie, à interpoler entre ces postures pour obtenir un mouvement puis à convertir ces trajectoires en trajectoires articulaires. Les mouvements sont générés de manière à assurer l'équilibre statique du robot au sol.

Cette méthode permet de générer l'ensemble des positions articulaires à chaque instant pour le robot.

```
# Format
V0
# NbArt
15
# NbPts
20
# Cycle
300
# Rebouclage
1
# Temps
12
0.1 0 0 0.1 0 0 0.3 0 0 0.1 0 0 0.0 0 0 0.0 0 0 0.6 0 0 0.6 0 0 0.1 0
0 -0.1 0 0 0.8 0 0 0 0 0 0 0 0 0.8 0 0 0.1 0 0.1
0.1 0 0 0.1 0 0 0.3 0 0 0.1 0 0 0.0 0 0 0.0 0 0 0.6 0 0 0.6 0 0 0.1 0
0 -0.1 0 0 0.8 0 0 0 0 0 0 0 0 0.8 0 0 0.1 0 0.1
0.1 0 0 0.1 0 0 0.3 0 0 0.1 0 0 0.0 0 0 0.0 0 0 0.6 0 0 0.6 0 0 0.1 0
0 -0.1 0 0 0.8 0 0 0 0 0 0 0 0 0.8 0 0 0.1 0 0.1
```

On a le même type d'en-tête que pour les fichiers V1. On a ensuite une liste de valeurs rangées dans l'ordre de numérotation des articulations et regroupées par 3 : position articulaire, vitesse articulaire, accélération articulaire désirées. Vitesse et accélération ne sont pas utilisées dans la version actuellement implémentée. Les trajectoires sont déclarées dans :

`/home/chimay/vxworks/moinette/Exec/Sessions/current/`

4.2.2 Module algorithmique *bmoveTraj*

On distingue cinq étapes dans chaque application du robot :

1. *PINIT* : le robot est suspendu et atteint la posture de départ du mouvement (position initiale de la trajectoire désirée)
2. *PWAIT* : le robot reste immobile pendant un temps fixé à l'avance pour être posé sur le sol,
3. *PMANIP* : le robot effectue le mouvement désiré (poursuit la trajectoire désirée),
4. *PWAITR* : le robot reste immobile pendant un temps fixé à l'avance pour être suspendu,
5. *PRETOUR* : le robot revient dans sa position initiale.

Le retour en position initiale (PRETOUR) est utile car le mauvais fonctionnement des potentiomètres nous a obligés à fonctionner en init manuel et donc à positionner manuellement le robot dans sa position zéro. Il est plus simple d'arrêter une manipulation près de la position zéro pour diminuer la tâche de l'utilisateur.

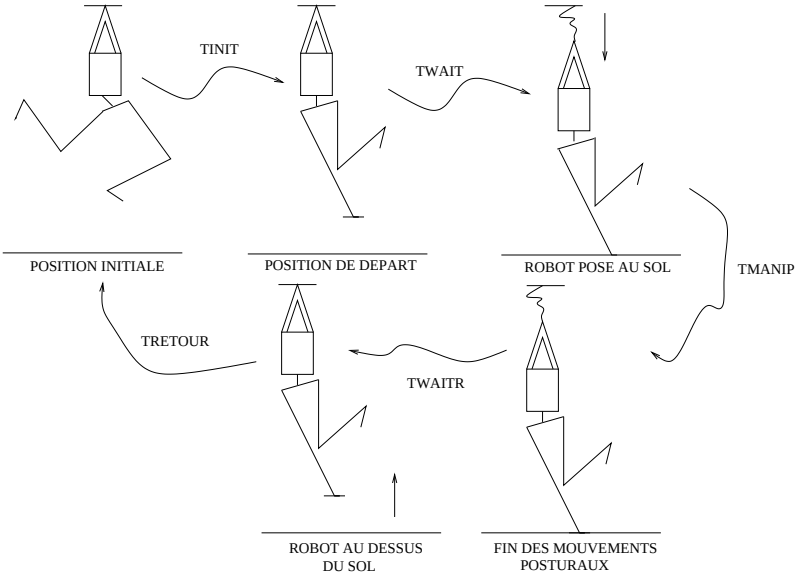


FIG. 4.20: *Différentes étapes de la* TR bipMove

L'ensemble des étapes est décrit dans le corps du module *bmoveTraj*.

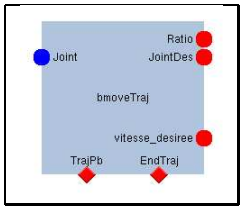


FIG. 4.21: *Module algorithmique* bmoveTraj

Port	**Vecteur**	**Description**
Entrée	JointI[15]	positions articulaires réelles
Sortie	JointDesO[15]	positions articulaires désirées

TAB. 4.10: *Description des ports du module* bmoveTraj

Chapitre 5

Trajectoires validées

Les différentes trajectoires jouées par le robot se trouvent dans le répertoire :

`/home/chimay/vxworks/moinette/Exec/Sessions/current/`

La plus grande prudence est nécessaire avant de jouer une nouvelle trajectoire robot au sol.

5.1 Trajectoires V1

Les trajectoires de type V1 ne doivent être jouées que lorsque le robot est suspendu. Elles permettent de spécifier simplement des mouvements.

5.2 Trajectoires V0

Les trajectoires V0 sont générées à l'aide du simulateur et garantissent la stabilité statique si le robot est posé à plat sur un sol plat.

5.2.1 Postures

Les postures disponibles sont listées ci-dessous.

p0.traj	posture initiale
p1.traj	posture pied légèrement sur le coté
p2.traj	posture pied sur le coté
p5.traj	posture pied en avant
p6.traj	posture flamand rose

TAB. 5.1: *Postures disponibles*

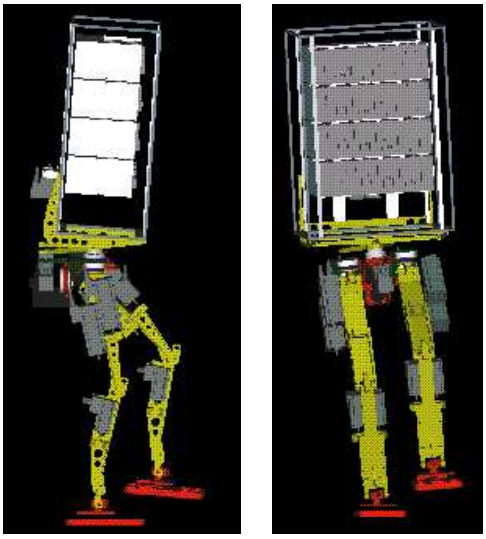


FIG. 5.1: *Postion p0 : position initiale*

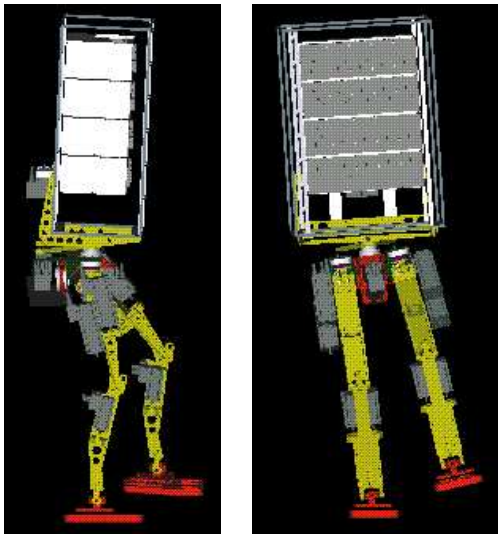


FIG. 5.2: *Postion p1 : pied petit écart côté*

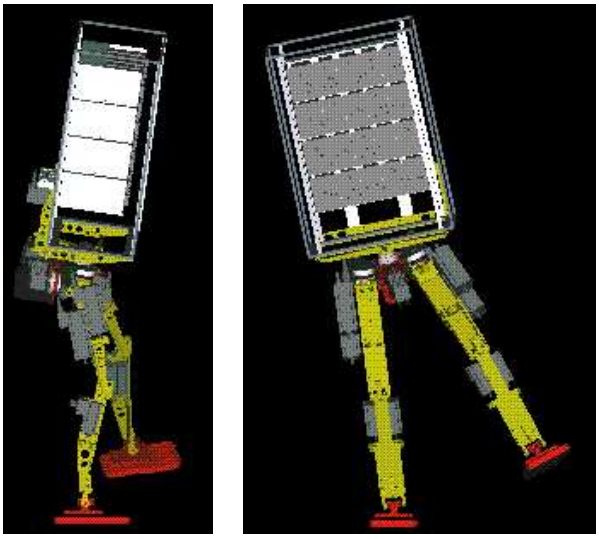


FIG. 5.3: *Postion p2 : pied grand écart côté*

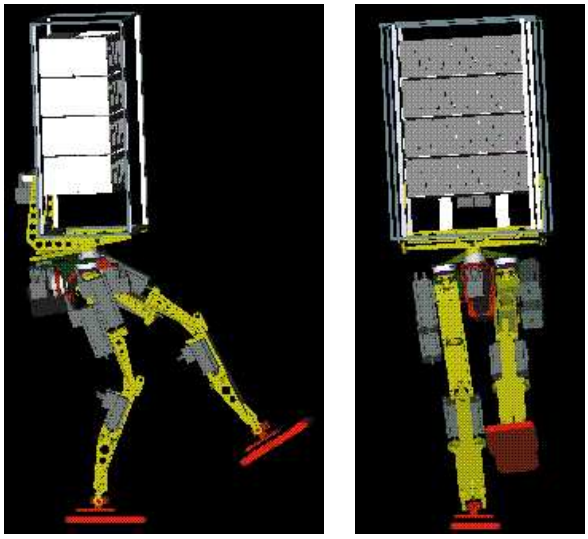


FIG. 5.4: *Postion p5 : pied en avant*

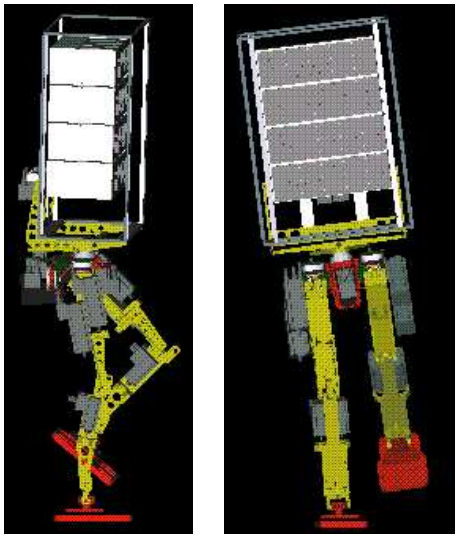


FIG. 5.5: *Postion p6 : pied flamand rose*

5.2.2 Mouvements Posturaux

Les mouvements posturaux disponibles sont listés ci-dessous. La nomenclature pXYZ.traj signifie que le mouvement relie la posture X à la posture Y puis la posture Y à la posture Z.

p01.traj	petit balayage horizontal
p02.traj	grand balayage horizontal
p65.traj	
p62.traj	
p062.traj	

TAB. 5.2: *Mouvements posturaux disponibles*

Chapitre 6

Conclusions

La version simplifiée du robot à 6 ddl est parvenue à la réalisation de différentes démarches statiquement stables dans le plan. Les algorithmes de cette version ont été partiellement transposés dans la version 15 ddl du prototype. Nous sommes ainsi parvenus à l'expérimentation de mouvements posturaux statiquement stables en appui sur le pied droit.

Le prototype de Hanovre devra être remis en route et il faudra refaire à nouveau les mouvements posturaux dont il était capable avant son départ. Les algorithmes devront être transposés également sur le second prototype du robot. Les programmes développés pour la version simplifiée du robot devront être complètement implémentées sur la version 15 ddl (détection de support).
Le double support est un point problématique, en effet, dans cette configuration, le robot forme une chaîne fermée soumise aux habituels problèmes d'hyperstaticité. Ne disposant pas de l'information force tangentielle nous n'étions pas capables de vérifier l'absence de problème à ce niveau. Nous avons donc généré des démarches pour lesquelles le robot ne bougeait pas dans les phases de double support.

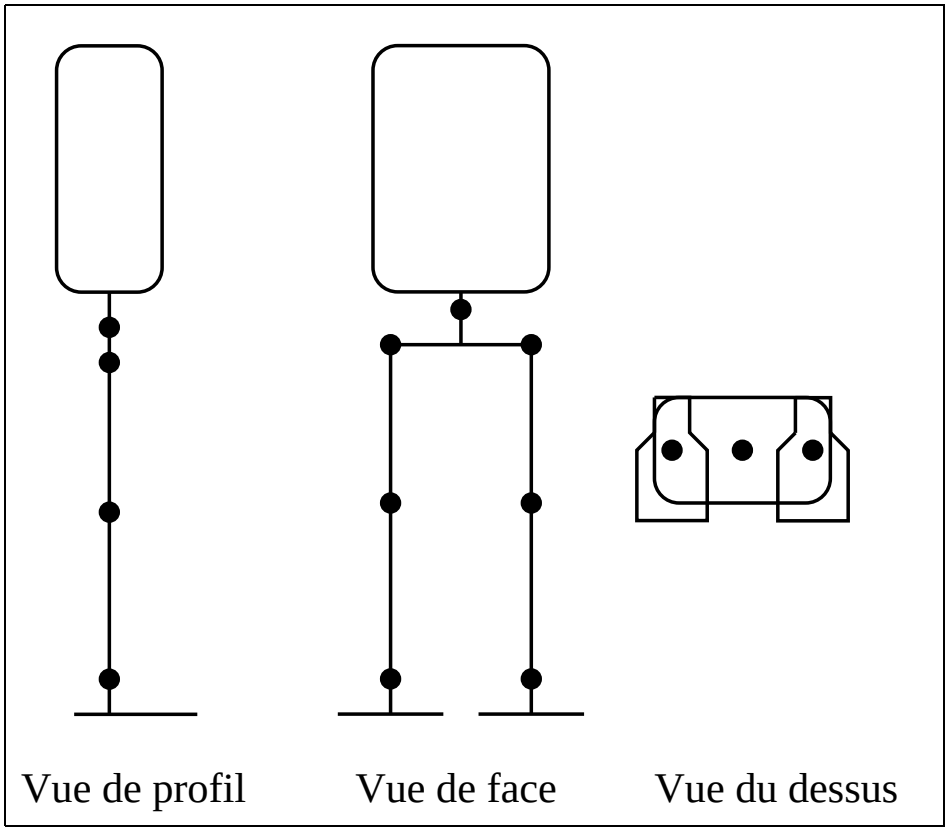
Troisième partie

Annexes

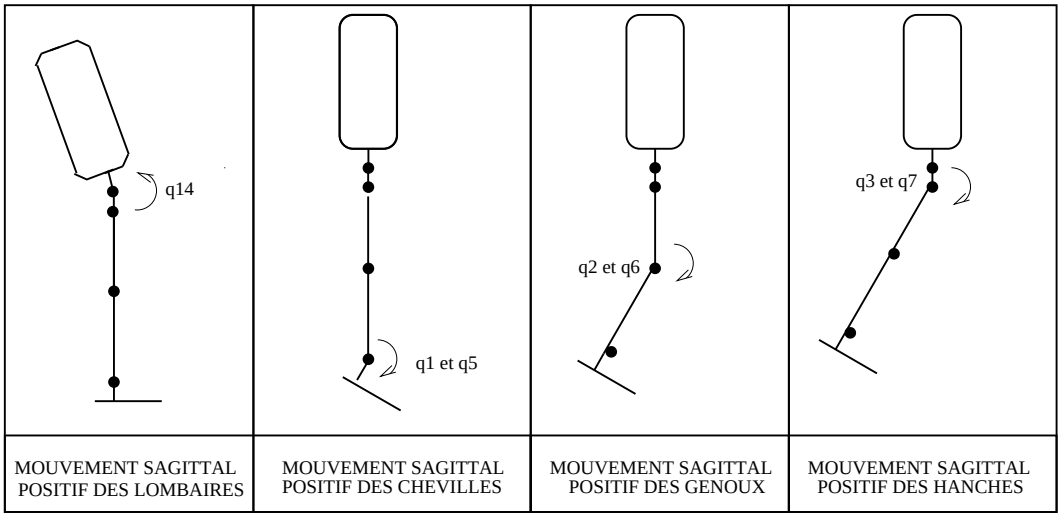
Chapitre 1

Annexes relatives aux mouvements du robot

1.1 Position zéro

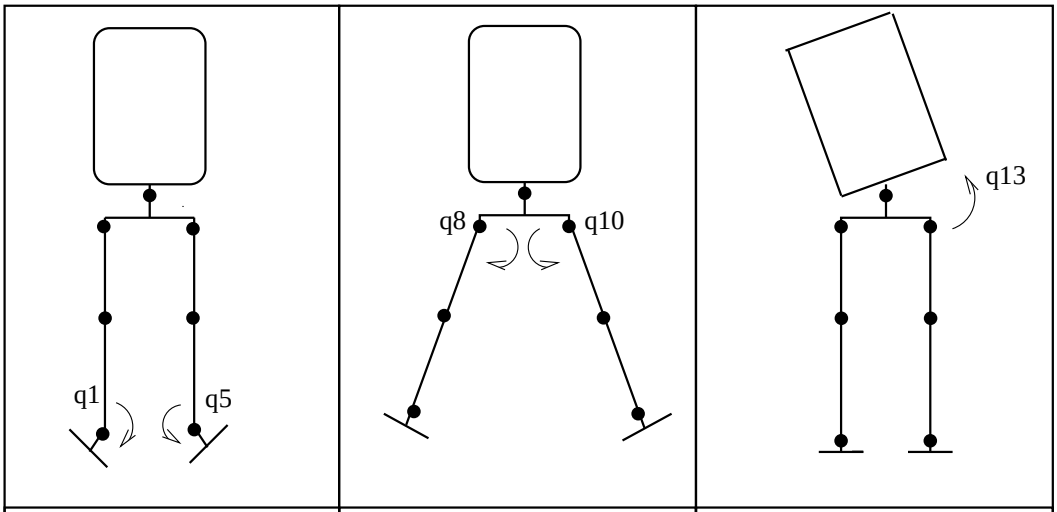


1.2 Mouvements sagittaux



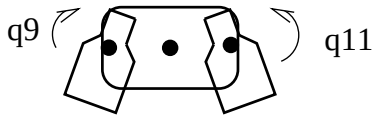
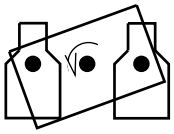
MOUVEMENT SAGITTAL POSITIF DES LOMBAIRES	MOUVEMENT SAGITTAL POSITIF DES CHEVILLES	MOUVEMENT SAGITTAL POSITIF DES GENOUX	MOUVEMENT SAGITTAL POSITIF DES HANCHES

1.3 Mouvements frontaux



MOUVEMENT FRONTAL POSITIF DES CHEVILLES	MOUVEMENT FRONTAL POSITIF DES HANCHES	MOUVEMENT FRONTAL POSITIF DES LOMBAIRES

1.4 Mouvements verticaux

	
MOUVEMENT VERTICAL POSITIF DES HANCHES	MOUVEMENT VERTICAL POSITIF DES LOMBAIRES

Chapitre 2

Limitations des mouvements

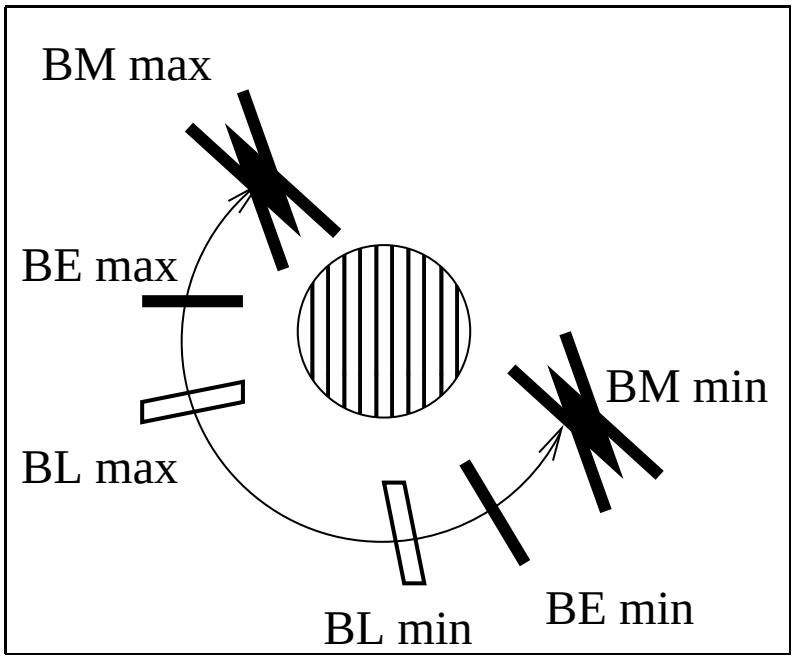


FIG. 2.1: *Position des différentes butées sur le débattement d'une articulation*

2.1 Butées mécaniques du robot

Désignation	**Butée (rad)**
MECH_MIN_ANKLE_FRON	-0.3490658504
MECH_MAX_ANKLE_FRON	0.3490658504
MECH_MIN_ANKLE_SAGI	-0.7853981635
MECH_MAX_ANKLE_SAGI	0.7853981635
MECH_MIN_KNEE	0.0
MECH_MAX_KNEE	1.570796327
MECH_MIN_HIP_FLEX	-1.308996939
MECH_MAX_HIP_FLEX	0.3490658504
MECH_MIN_HIP_SUBD	-0.2617993878
MECH_MAX_HIP_SUBD	0.2617993878
MECH_MIN_HIP_VERT	-0.2617993878
MECH_MAX_HIP_VERT	0.2617993878
MECH_MIN_TRUNC_VERT	-0.2617993878
MECH_MAX_TRUNC_VERT	0.2617993878
MECH_MIN_TRUNC_FRON	-0.17453293
MECH_MAX_TRUNC_FRON	0.17453293
MECH_MIN_TRUNC_SAGI	-0.3490658504
MECH_MAX_TRUNC_SAGI	0.17453293

TAB. 2.1: *Butées mécaniques des articulations*

2.2 Butées électriques

Désignation	**Butée (rad)**
ELEC_MIN_ANKLE_FRON	-0.2792527
ELEC_MAX_ANKLE_FRON	0.2792527
ELEC_MIN_ANKLE_SAGI	-0.715585
ELEC_MAX_ANKLE_SAGI	0.715585
ELEC_MIN_KNEE	0.0872665
ELEC_MAX_KNEE	1.4835299
ELEC_MIN_HIP_FLEX	-1.2391838
ELEC_MAX_HIP_FLEX	0.2792527
ELEC_MIN_HIP_SUBD	-0.2094395
ELEC_MAX_HIP_SUBD	0.2094395
ELEC_MIN_HIP_VERT	-0.2094395
ELEC_MAX_HIP_VERT	0.2094395
ELEC_MIN_TRUNC_VERT	-0.2094395
ELEC_MAX_TRUNC_VERT	0.2094395
ELEC_MIN_TRUNC_FRON	-0.1047198
ELEC_MAX_TRUNC_FRON	0.1047198
ELEC_MIN_TRUNC_SAGI	-0.2792527
ELEC_MAX_TRUNC_SAGI	0.1047198

TAB. 2.2: *Butées électriques des articulations*

2.3 Butées logicielles

Désignation	Butées (rad)
MinCheD_F = MinCheG_F	-0.24
MaxCheD_F = MaxCheG_F	0.24
MinCheD_S = MinCheG_S	-0.68
MaxCheD_S = MaxCheG_S	0.68
MinGenD_S = MinGenG_S	0.10
MaxGenD_S = MaxGenG_S	1.45
MinHanD_S = MinHanG_S	-1.20
MaxHanD_S = MaxHanG_S	0.24
MinHanD_F = MinHanG_F	-0.19
MaxHanD_F = MaxHanG_F	0.19
MinHanD_V = MinHanG_V	-0.19
MaxHanD_V = MaxHanG_V	0.19
MinLomb_V	-0.21
MaxLomb_V	0.21
MinLomb_F	-0.09
MaxLomb_F	0.09
MinLomb_S	-0.24
MaxLomb_S	0.09

TAB. 2.3: *Butées logicielles des articulations*

Chapitre 3

Annexes relatives à la modélisation du robot

Nous avons regroupé les listings des fichiers *cinematique.maple* et *dynamique.maple* pour les configurations robot suspendu et robot en simple support droit.

Remarque : Les différents fichiers relatifs au modèle se trouvent dans :

`robotique/Bipede/utils/RobotDyn`

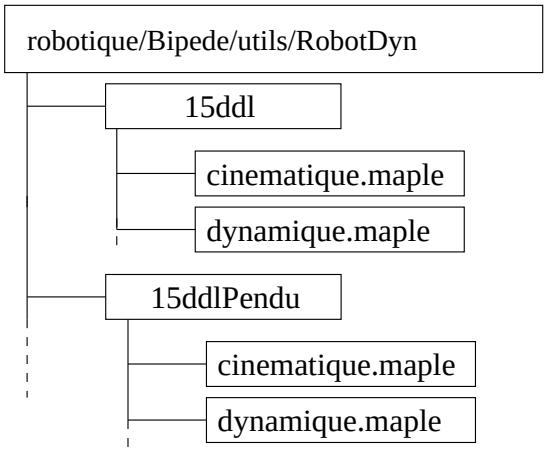


FIG. 3.1: *Organisation du répertoire* RobotDyn

3.1 Robot en simple support droit

3.1.1 Simple support droit *cinematique.maple*

```
# Cinematique du Robot BIP2000
# Paramètres de DENAVIT-HARTENBERG
# MODIFIÉ (KHALIL-KLEINFINGER)

# Nombre de solides
NSOL := 17+3:

# Nombre de degrés de liberté
NDDL := 15+6:

# Définition des coordonnées et
# vitesses généralisées
q := vector(NDDL):
qdot := vector(NDDL):

# Coordonnees base de reference
base:=vector([q[16],q[17],q[18]]):

# Repère 1 : pied droit
ref_1    := 20:
r_1      := 0:
lambda_1 := 0:
alpha_1  := 0:
theta_1  := Pi/2:

# Repère 2 : cardan droit
ref_2    := 1:
r_2      := 0.083:
lambda_2 := -0.170:
alpha_2  := Pi/2:
theta_2  := q[1]:

# Repère 3 : tibia droit
ref_3    := 2:
r_3      := 0:
lambda_3 := 0:
alpha_3  := -Pi/2:
theta_3  := q[2]:
```

```
# Repère 4 : cuisse droite
ref_4    := 3:
r_4      := 0.410:
lambda_4 := 0:
alpha_4  := 0:
theta_4  := q[3]:

# Repère 5 : levier hanche droite
ref_5    := 4:
r_5      := 0.410:
lambda_5 := 0:
alpha_5  := 0:
theta_5  := Pi/2+q[4]:

# Repère 6 : equerre hanche droite
ref_6    := 5:
r_6      := 0:
lambda_6 := 0:
alpha_6  := Pi/2:
theta_6  := -Pi/2+q[10]:

# Repère 7 : pelvis
ref_7    := 6:
r_7      := 0:
lambda_7 := 0:
alpha_7  := Pi/2:
theta_7  := q[9]:

# Repère 8 : equerre hanche gauche
ref_8    := 7:
r_8      := 0.220:
lambda_8 := 0:
alpha_8  := 0:
theta_8  := q[11]:

# Repère 9 : levier hanche gauche
ref_9    := 8:
r_9      := 0:
lambda_9 := 0:
alpha_9  := -Pi/2:
theta_9  := Pi/2+q[12]:
```

```
# Repère 10 : cuisse gauche
ref_10    := 9:
r_10      := 0:
lambda_10 := 0:
alpha_10  := -Pi/2:
theta_10  := Pi/2-q[8]:
# signes theta <-> q opposes

# Repère 11 : tibia gauche
ref_11    := 10:
r_11      := 0.410:
lambda_11 := 0:
alpha_11  := 0:
theta_11  := -q[7]:
# signes theta <-> q opposes

# Repère 12 : cardan gauche
ref_12    := 11:
r_12      := 0.410:
lambda_12 := 0:
alpha_12  := 0:
theta_12  := -q[6]:
# signes theta <-> q opposes

# Repère 13 : pied gauche
ref_13    := 12:
r_13      := 0:
lambda_13 := -0.120:
alpha_13  := -Pi/2:
theta_13  := q[5]:

# Repère 14 : (inexistant)
ref_14    := 0:
r_14      := 0:
lambda_14 := 0:
alpha_14  := 0:
theta_14  := 0:

# Repère 15 : levier lomb.
ref_15    := 7:
r_15      := 0.110:
lambda_15 := 0.128:
```

```
alpha_15  := -Pi/2:
theta_15  := q[13]:

# Repère 16 : cardan armoire
ref_16    := 15:
r_16      := 0:
lambda_16 := 0:
alpha_16  := Pi/2:
theta_16  := Pi/2+q[14]:

# Repère 17 : support+armoire
ref_17    := 16:
r_17      := 0:
lambda_17 := 0:
alpha_17  := -Pi/2:
theta_17  := q[15]:

# Repère 18 : rotation Y
ref_18    := 0:
r_18      := 0:
lambda_18 := 0:
alpha_18  := -Pi/2:
theta_18  := Pi/2+q[20]:

# Repère 19 : rotation X
ref_19    := 18:
r_19      := 0:
lambda_19 := 0:
alpha_19  := Pi/2:
theta_19  := Pi/2+q[19]:

# Repère 20 : rotation Z
ref_20    := 19:
r_20      := 0:
lambda_20 := 0:
alpha_20  := -Pi/2:
theta_20  := -Pi/2+q[21]:
```

3.1.2 Simple support droit *dynamique.maple*

```
#
# Dynamique du Robot BIP2000
#
#

# Informations dynamiques sur les
# différents solides du robot

# Vecteur gravité
Gravite := vector([0,-9.81,0]):

# Solide 1 : pied droit
m_1  := 2.34:
G_1  := vector([24,151,0]*1e-3):
IO_1 := matrix([[7,0,0],[0,0,0],[0,0,7]]*1e-2):

# Solide 2 : cardan droit
m_2  := 0.18:
G_2  := vector([0,0,0]*1e-3):
IO_2 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 3 : tibia droit
m_3  := 5.93:
G_3  := vector([258,28,0]*1e-3):
IO_3 := matrix([[2,-4,0],[-4,47,0],[0,0,47]]*1e-2):

# Solide 4 : cuisse droite
m_4  := 10.9:
G_4  := vector([250,5,45]*1e-3):
IO_4 := matrix([[6,-1,-15],[-1,82,0],[-15,0,80]]*1e-2):

# Solide 5 : levier hanche droite
m_5  := 0.7:
G_5  := vector([2,-34,0]*1e-3):
IO_5 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 6 : equerre hanche droite
m_6  := 3.2:
G_6  := vector([-5,107,29]*1e-3):
IO_6 := matrix([[6,0,0],[0,0,0],[0,0,5]]*1e-2):
```

```
# Solide 7 : pelvis
m_7  := 8.8:
G_7  := vector([110,-12,-68]*1e-3):
IO_7 := matrix([[13,1,7],[1,24,-2],[7,-2,15]]*1e-2):

# Solide 8 : equerre hanche gauche
m_8  := 3.2:
G_8  := vector([5,29,-107]*1e-3):
IO_8 := matrix([[6,0,0],[0,5,0],[0,0,0]]*1e-2):

# Solide 9 : levier hanche gauche
m_9  := 0.7:
G_9  := vector([2,0,34]*1e-3):
IO_9 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 10 : cuisse gauche
m_10  := 10.9:
G_10  := vector([160,-5,-45]*1e-3):
IO_10 := matrix([[6,0,5],[0,42,0],[5,0,39]]*1e-2):

# Solide 11 : tibia gauche
m_11  := 5.93:
G_11  := vector([152,-28,0]*1e-3):
IO_11 := matrix([[2,3,0],[3,21,0],[0,0,21]]*1e-2):

# Solide 12 : cardan gauche
m_12  := 0.18:
G_12  := vector([0,0,0]*1e-3):
IO_12 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 13 : pied gauche
m_13  := 2.34:
G_13  := vector([59,0,139]*1e-3):
IO_13 := matrix([[6,0,-2],[0,7,0],[-2,0,1]]*1e-2):

# Solide 14 : (inexistant)
m_14  := 0:
G_14  := vector([0,0,0]*1e-3):
IO_14 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 15 : levier lombaires
m_15  := 1.05:
```

```
G_15  := vector([0,47,-25]*1e-3):
IO_15 := matrix([[0,0,0],[0,0,0],[0,0,1]]*1e-2):

# Solide 16 : cardan armoire
m_16  := 0.46:
G_16  := vector([0,0,46]*1e-3):
IO_16 := matrix([[1,0,0],[0,0,0],[0,0,1]]*1e-2):

# Solide 17 : support armoire + armoire
m_17  := 48:
G_17  := vector([405,13,9]*1e-3):
IO_17 := matrix([[126,6,-6],[6,1075,0],[-6,0,1026]]*1e-2):

# Solide 18 : (inexistant)
m_18  := 0:
G_18  := vector([0,0,0]*1e-3):
IO_18 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 19 : (inexistant)
m_19  := 0:
G_19  := vector([0,0,0]*1e-3):
IO_19 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 20 : (inexistant)
m_20  := 0:
G_20  := vector([0,0,0]*1e-3):
IO_20 := matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):
}
```

3.2 Robot suspendu

3.2.1 Robot suspendu *cinematique.maple*

```
#
# Cinematique du Robot BIP2000
#
# Paramètres de DENAVIT-HARTENBERG MODIFIÉ (KHALIL-KLEINFINGER)

# Nombre de solides
NSOL := 20:

# Nombre de degrés de liberté
NDDL := 15:

# Définition des coordonnées et vitesses généralisées
q := vector(NDDL):
qdot := vector(NDDL):

# Coordonnees de la base de reference
base := vector([0,0,0]):
# NB := x vers l'avant, y vers le haut, z vers la gauche

ArtCheD_F := 0:
ArtCheD_S := 1:
ArtGenD_S := 2:
ArtHanD_S := 3:
ArtCheG_F := 4:
ArtCheG_S := 5:
ArtGenG_S := 6:
ArtHanG_S := 7:
ArtHanD_F := 8:
ArtHanD_V := 9:
ArtHanG_F := 10:
ArtHanG_V := 11:
ArtLomb_V := 12:
ArtLomb_F := 13:
ArtLomb_S := 14:


# Repère 17 : support armoire + armoire
#
#     FIXE
```

```
#
ref_17    := 0:
r_17      := 0:
lambda_17 := 0:
alpha_17  := 0:
theta_17  := Pi/2:

# Repere 18 : intermediaire
ref_18    := 17:
r_18   := 0:
lambda_18 := 0:
alpha_18  := 0:
theta_18  := -q[ArtLomb_S+1]:

# Repère 16 : cardan armoire
ref_16    := 18:
r_16      := 0:
lambda_16 := 0:
alpha_16  := Pi/2:
theta_16  := 0:

# Repere 19 : repere intermediaire
ref_19    := 16:
r_19      := 0:
lambda_19 := 0:
alpha_19  := 0:
theta_19  := -Pi/2-q[ArtLomb_F+1]:

# Repère 15 : levier lombaires
ref_15    := 19:
r_15      := 0:
lambda_15 := 0:
alpha_15  := -Pi/2:
theta_15  := 0:

# Repere 20 : intermediaire
ref_20    := 15:
r_20      := 0:
lambda_20 := -0.128:
alpha_20  := 0:
theta_20  := -q[ArtLomb_V+1]:
```

```
# Inversion de la chaine articulee bassin->armoire
# en armoire->bassin
# Repère 14 : Semelle gauche #
ref_14    := 13: #
r_14      := 0: #
lambda_14 := 0: #
alpha_14  := 0: #
theta_14  := 0: #

# Repère 7 : pelvis
ref_7    := 20:
r_7      := -0.110:
lambda_7 := 0:
alpha_7  := Pi/2:
theta_7  := 0:

##### Jambe droite recopie de jambe gauche a partir du levier
##### NB: dynamique.maple a modifier
# Repère 6 : equerre hanche droite
ref_6    := 7:
r_6      := 0:
lambda_6 := 0:
alpha_6  := 0:
theta_6  := -q[ArtHanD_F+1]:

# Repère 5 : levier hanche droite
ref_5    := 6:
r_5      := 0:
lambda_5 := 0:
alpha_5  := -Pi/2:
theta_5  := Pi/2-q[ArtHanD_V+1]:

# Repère 4 : Cuisse droite
ref_4    := 5:
r_4      := 0:
lambda_4 := 0:
alpha_4  := -Pi/2:
theta_4  := Pi/2-q[ArtHanD_S+1]:

# Repère 3 : Tibia droit
ref_3    := 4:
r_3      := 0.410:
```

```
lambda_3 := 0:
alpha_3  := 0:
theta_3  := -q[ArtGenD_S+1]:

# Repère 2 : Cardan droit
ref_2    := 3:
r_2      := 0.410:
lambda_2 := 0:
alpha_2  := 0:
theta_2  := -q[ArtCheD_S+1]:

# Repère 1 : Pied droit
ref_1  := 2:
r_1  := 0:
lambda_1 := -0.120:
alpha_1  := -Pi/2:
theta_1  := -q[ArtCheD_F+1]:

####### Jambe gauche identique au 15ddl pied fixe
# Repère 8 : equerre hanche gauche
ref_8    := 7:
r_8      := 0.220:
lambda_8 := 0:
alpha_8  := 0:
theta_8  := q[ArtHanG_F+1]:

# Repère 9 : levier hanche gauche
ref_9    := 8:
r_9      := 0:
lambda_9 := 0:
alpha_9  := -Pi/2:
theta_9  := Pi/2+q[ArtHanG_V+1]:

# Repère 10 : cuisse gauche
ref_10    := 9:
r_10      := 0:
lambda_10 := 0:
alpha_10  := -Pi/2:
theta_10  := Pi/2-q[ArtHanG_S+1]: # signes theta <-> q opposes

# Repère 11 : tibia gauche
ref_11    := 10:
```

```
r_11      := 0.410:
lambda_11 := 0:
alpha_11  := 0:
theta_11  := -q[ArtGenG_S+1]: # signes theta <-> q opposes

# Repère 12 : cardan gauche
ref_12    := 11:
r_12      := 0.410:
lambda_12 := 0:
alpha_12  := 0:
theta_12  := -q[ArtCheG_S+1]: # signes theta <-> q opposes

# Repère 13 : pied gauche
ref_13    := 12:
r_13      := 0:
lambda_13 := -0.120:
alpha_13  := -Pi/2:
theta_13  := q[ArtCheG_F+1]:
```

3.2.2 Robot suspendu *dynamique.maple*

```
#
# Dynamique du Robot BIP2000

# Informations dynamiques sur les différents solides du robot

# Vecteur gravité
Gravite: vector([0,-9.81,0]):

# Solide 1 : pied droit
m_1:=2.34:
G_1:=vector([59,0,139]*1e-3):
IO_1:=matrix([[7,0,0],[0,0,0],[0,0,7]]*1e-2):

# Solide 2 : cardan droit
m_2:=0.18:
G_2:=vector([0,0,0]*1e-3):
IO_2:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 3 : tibia droit
m_3:=5.93:
G_3:=vector([152,-28,0]*1e-3):
IO_3:=matrix([[2,-4,0],[-4,47,0],[0,0,47]]*1e-2):

# Solide 4 : cuisse droite
# G_4_z inverse pour tenir compte de la symetrie
m_4:=10.9:
G_4:=vector([160,-5,45]*1e-3):
IO_4:=matrix([[6,-1,-15],[-1,82,0],[-15,0,80]]*1e-2):

# Solide 5 : levierhanchedroite
m_5:=0.7:
G_5:=vector([2,0,34]*1e-3):
IO_5:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 6 : equerrehanchedroite
#G_6_xinversepourtenircomptedelasymetrie
m_6:=3.2:
G_6:=vector([-5,29,-107]*1e-3):
IO_6:=matrix([[6,0,0],[0,0,0],[0,0,5]]*1e-2):

# Solide 7 : pelvis
```

```
m_7:=8.8:
G_7:=vector([110,-12,-68]*1e-3):
IO_7:=matrix([[13,1,7],[1,24,-2],[7,-2,15]]*1e-2):

# Solide 8 : equerrehanchegauche
m_8:=3.2:
G_8:=vector([5,29,-107]*1e-3):
IO_8:=matrix([[6,0,0],[0,5,0],[0,0,0]]*1e-2):

# Solide 9 : levierhanchegauche
m_9:=0.7:
G_9:=vector([2,0,34]*1e-3):
IO_9:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 10 : cuissegauche
m_10:=10.9:
G_10:=vector([160,-5,-45]*1e-3):
IO_10:=matrix([[6,0,5],[0,42,0],[5,0,39]]*1e-2):

# Solide 11 : tibiagauche
m_11:=5.93:
G_11:=vector([152,-28,0]*1e-3):
IO_11:=matrix([[2,3,0],[3,21,0],[0,0,21]]*1e-2):

# Solide 12 : cardangauche
m_12:=0.18:
G_12:=vector([0,0,0]*1e-3):
IO_12:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 13 : piedgauche
m_13:=2.34:
G_13:=vector([59,0,139]*1e-3):
IO_13:=matrix([[6,0,-2],[0,7,0],[-2,0,1]]*1e-2):

# Solide 14 : SOL
m_14:=0:
G_14:=vector([0,0,0]*1e-3):
IO_14:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 15 : levierlombaires
m_15:=1.05:
G_15:=vector([0,47,-25]*1e-3):
```

```
IO_15:=matrix([[0,0,0],[0,0,0],[0,0,1]]*1e-2):

# Solide 16 : cardanarmoire
m_16:=0.46:
G_16:=vector([0,0,46]*1e-3):
IO_16:=matrix([[1,0,0],[0,0,0],[0,0,1]]*1e-2):

# Solide 17 : supportarmoire+armoire
m_17:=48:
G_17:=vector([405,13,9]*1e-3):
IO_17:=matrix([[126,6,-6],[6,1075,0],[-6,0,1026]]*1e-2):

# Solide 18 : intermediaire
m_18:=0:
G_18:=vector([0,0,0]*1e-3):
IO_18:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 19 : intermediaire
m_19:=0:
G_19:=vector([0,0,0]*1e-3):
IO_19:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):

# Solide 20 : intermediaire
m_20:=0:
G_20:=vector([0,0,0]*1e-3):
IO_20:=matrix([[0,0,0],[0,0,0],[0,0,0]]*1e-2):
```

Chapitre 4

Annexes relatives à l'environnement logiciel

```
-----------------------------------------------------------------

          Environnement pour l'utilisation des robots bipèdes
            mis à jour le 15 mars 2000
                        Christine Azevedo

-----------------------------------------------------------------


1. Répertoire Bipede
----------------------

Chaque utilisateur doit posséder son propre répertoire de travail:
Bipede.

Des liens lui permettent d'exporter ses propres trajectoires dans les
répertoires de travail du robot : current.

  mkdir ~/Bipede
  cd Bipede
  mkdir datas
  ln -s /home/chimay/vxworks/moinette/Exec/Sessions/current current
```

```
2. Raccourcis
----------------

Créer les alias moinette et bip.

alias moinette 'xterm -T moinette -n moinette -e windsh
moinette@chimay -s ~/.wind/windsh-moinette.vx'
alias bip 'moinette'

Ces alias permettent de lancer la connection avec le robot
et les fenêtres de travail correspondantes.


3. Répertoire .wind
---------------------

Créer un répertoire .wind contenant le fichier windsh-moinette.vx.

# Launch the virtual Console
?source /local/projets/robotique/wind/.wind/xxVio.tcl
?xxConsoleCreate 4
?xxConsoleStd 4
# Debut de session
ld < /robotique/lib/vxworks/moduleIP.l
ld < /robotique/lib/vxworks/t501drv.vxo
ld < /robotique/lib/vxworks/libRunidig.vxo
ld < /robotique/lib/vxworks/drvBip15.vxo
ld < /robotique/lib/vxworks/utilBip15.vxo
ld < /robotique/Bipede/orccad/user15/Exec/Bin/vxworks/PROCNAME

où  PROCNAME est la procédure {\SCSHAPE ORCCAD} qui sera chargée
automatiquement en lançant bip15.


4. Fichier .login
--------------------

Rajouter si nécessaire dans le fichier ~/.login

les lignes suivantes disponibles sur

     http://www.inrialpes.fr/iramr/local/ordi.html
```

```
# Moyens robotiques
# -----------------
setenv ROBOTIQUE /local/projets/robotique
#
# Tornado : cross-developpement VxWorks 5.3
#-------------------------------------------
setenv WIND_BASE $ROBOTIQUE/wind
setenv WIND_INCLUDE ${WIND_BASE}/target/h
setenv WIND_REGISTRY chimay
setenv WIND_LMHOST chimay

#
# Orccad
#-------------------------------------------
setenv ORCCAD_USER robotique/Bipede/orccad/user15
setenv ORCCAD_SYS  robotique/Bipede/orccad/sys
setenv ILVPATH $ORCCAD_SYS/Lib/data
setenv ILVHOME /usr/ilog/views24

Il faut également rajouter si nécessaire, les PATH :
${WIND_BASE}/host${WIND_HOST_TYPE}/bin:${ROBOTIQUE}/bin/solaris


5. Fichier .datfilter
-----------------------

Il est possible de sauvegarder et d'exploiter des données au cours
 d'une manipulation. Créer le fichier :

$HOME/.datfilter.cfg contenant les lignes suivantes :

# Fichier d'extraction de données.
#
# Format :
# Path directory for input data files coming from {\Scshape Orccad}
# Path directory for output data files
# FileInputDataName FileOutputDataName nbcol
$HOME/Bipede/current
$HOME/Bipede/datas
NomModule1_IdModule1_NomPort1 x1 8
```

```
NomModule2_IdModule2_NomPort2 x2 8
NomModule3_IdModule3_NomPort3 x3 8
NomModule4_IdModule4_NomPort4 x4 8

Lorsqu'on exécute une procédure-robot il est possible de sauvegarder
 et/ou d'afficher des données dans le répertoire
de sauvegarde des données des bipèdes :

/home/chimay/vxworks/moinette/Exec/Sessions/current

Un script shell est appelé par la commande datfilter,
il permet de diviser les colonnes du fichier de données en autant
de fichiers unidimensionnels pour permettre leur exploitation sous
Maple, Xprism, Matlab ou Gnuplot, ces fichiers sont
sauvegardés chez l'utilisateur :

/Bipede/datas
```

Chapitre 5

Annexes relatives aux manipulations

5.1 Exécution de la procédure ProcPts

```
-------------------------------------------------------------------
                  Utilisation des robots bipèdes
               Procedure ProcPts
                       ( ROBOT SUSPENDU )
                       Christine Azevedo
-------------------------------------------------------------------

1. Préparation de la station
-----------------------------

Lancer la commande : bip
Deux fenêtres s'affichent :
   -  moinette qui donne accés au robot
   -  vio qui affiche les divers printf et scanf.


2. Exécution d'une procédure
-----------------------------

  1- enclencher le bouton arrêt d'urgence, maintenir le robot
     au dessus du sol,
```

2- mettre le robot et l'armoire de commande sous tension,

3- charger la procédure souhaitée dans la fenetre moinette :

ld < robotique/Bipede/orccad/user15/Exec/Bin/vxworks/ProcPts

4- initialiser le robot (marques blanches) et genoux en butée

5- dans la fenêtre moinette éxécuter la commande

orcProcPtsGo

6- lorsque le message sur l'init automatique apparait retirer les genoux des butées,

7- relever le bouton arrêt d'urgence,

8- taper n à la suite de la question vio sur l'initialisation automatique,

9- une question sur le nom de la trajectoire à charger vous est posée, donner le nom du fichier a jouer (exemple demo.traj)

10- lorsque la manip est terminee enclencher le bouton arret d'urgence

11- Pour pouvoir lancer une nouvelle manip. taper reboot dans la fenêtre moinette.

5.2 Exécution de la procédure ProcMove Version 1

```
-----------------------------------------------------------------
                  Utilisation des robots bipèdes
            Procedure ProcMove version 1
                        ( ROBOT SUSPENDU )
                        Christine Azevedo
-----------------------------------------------------------------

1. Préparation de la station
-----------------------------

Lancer la commande : bip
Deux fenêtres s'affichent :
   -  moinette qui donne accés au robot
   -  vio qui affiche les divers printf et scanf.


2. Exécution d'une procédure
-----------------------------

  1-  enclencher le bouton arrêt d'urgence, maintenir le robot
      au dessus du sol,

  2-  mettre le robot et l'armoire de commande sous tension,

  3-  charger la procédure ProcMove :
      ld < robotique/Bipede/orccad/user15/Exec/Bin/vxworks/ProcMove
      ld < robotique/lib/vxworks/demoBip15.vxo

  4-  initialiser le robot manuellement (marques blanches)
      et genoux en butée

  5-  dans la fenêtre moinette éxécuter la commande
                              go15

  6- à la question sur le nombre de traj à charger répondre : 1

  7- à la question sur le nom de la trajectoire : air1.traj

  8- lorsque la question sur l'init automatique apparait
     retirer les genoux des butées,
```

```
 9-  relever le bouton arrêt d'urgence,

10-  à la question sur l'initialisation automatique répondre : n

11-  lorsque l'experimentation est terminee, enclencher l'arrêt
     d'urgence.

12- Pour pouvoir lancer une nouvelle manip. Taper reboot dans
    la fenêtre moinette.
```

5.3 Exécution de la procédure ProcMove Version 0

```
-----------------------------------------------------------------
                  Utilisation des robots bipèdes
            Procedure ProcMove version 0
                        ( ROBOT AU SOL )
                        Christine Azevedo
-----------------------------------------------------------------

1. Préparation de la station
-----------------------------

Lancer la commande : bip
Deux fenêtres s'affichent :
   -  moinette qui donne accés au robot
   -  vio qui affiche les divers printf et scanf.


2. Exécution d'une procédure
-----------------------------

  1-  enclencher le bouton arrêt d'urgence,

  2-  mettre le robot et l'armoire de commande sous tension,

  3-  charger la procédure ProcMove :
      ld < robotique/Bipede/orccad/user15/Exec/Bin/vxworks/ProcMove
      ld < robotique/lib/vxworks/demoBip15.vxo

  4-  initialiser le robot manuellement (marques blanches)
      et genoux en butée

  5-  dans la fenêtre moinette éxécuter la commande
                              go15

  6- à la question sur le nbre de traj à charger répondre : 1

  7- à la question sur le nom de la trajectoire : p01.traj

  8- lorsque la question sur l'init automatique apparait
     retirer les genoux des butées,
```

9- relever le bouton arrêt d'urgence,

10- à la question sur l'initialisation automatique répondre : n

11- attendre que la phase d'init soit terminée (voir l'écran)

12- pendant la phase ''pose au sol'' descendre doucement le robot à l'aide du palan et le poser bien à plat sur sol pied droit. Relâcher la courroie du palan.

ATTENTION : SI LE ROBOT N'EST PAS POSÉ CORRECTEMENT ARRÊTER LA MANIP.

13- le robot effectue ses mouvements pendant la phase ''manips au sol''

14- pendant la phase de ''remontée du robot'' remonter doucement à l'aide du palan.

12- pendant la phase ''retour à l'init'' le robot se remet dans sa position de départ.

13- enclencher l'arrêt d'urgence.

14- Pour pouvoir lancer une nouvelle manip. Taper reboot dans la fenêtre moinette.

Si un problème survient, enclencher l'arret d'urgence tout en relevant rapidement le robot au dessus du sol à l'aide du palan.

Conseils : 3 personnes sont nécessaires pour réaliser ces expérimentations.
- 1 pour actionner le palan,
- 1 pour aider au positionnement du robot sur son pied d'appui,
- 1 pour tenir l'arrêt d'urgence et vérifier à distance le positionnement du pied d'appui.

Chapitre 6

Annexes relatives à l'exploitation des capteurs de force

6.1 Définitions

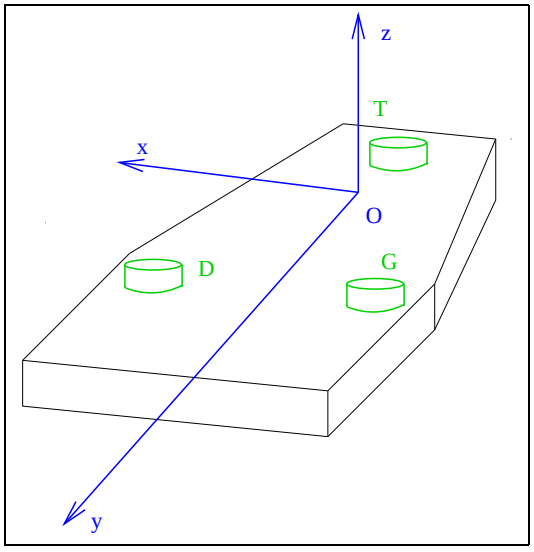


FIG. 6.1: *Représentation d'un pied du robot*

Ce chapitre reprend les calculs présentés dans le document [SAR00].

6.1.1 Repère R

Le repère $R = (O, x, y, z)$ lié à la semelle du pied (fig.6.1) se définit comme suit :

- O : projection de la cheville sur la semelle,
- x : axe latéral, vers la droite,
- y : axe latéral, vers l'avant,
- z : axe normal à la semelle, vers le haut.

6.1.2 Position des capteurs dans R

- Capteur du talon $\vec{OT} = (x_T, y_T, z_T)$
- Capteur gauche $\vec{OG} = (x_G, y_G, z_G)$
- Capteur droit $\vec{OD} = (x_D, y_D, z_D)$

6.1.3 Masse et gravité

- Masse de la semelle : m,
- Accélération de la pesanteur : g,
- Centre de masse M dans le repère R : $\vec{OM} = (x_M, y_M, z_M)$,
- Verticale ascendante exprimée dans le repère R : $\vec{VA} = (VA_x, VA_y, VA_z)$,

6.1.4 Informations des capteurs

- F positive en traction : 2500N pour 5V,
- F négative en compression : -2500N pour -5V,

6.2 Bilan des forces exercées sur la semelle

6.2.1 Action des forces de pression

$\vec{F_p} = (0, 0, F_p)$
$\vec{M_p} = (M_{px}, M_{py}, 0)$ exprimé au point O

6.2.2 Action des forces tangentielles de frottement

$\vec{F_t} = (F_{tx}, F_{ty}, 0)$
$\vec{M_t} = (0, 0, M_{tz})$ exprimé au point O

6.2.3 Poids

$\vec{F_M} = (-mgVA_x, -mgVA_y, -mgVA_z)$
$\vec{M_M} = (0,0,0)$ exprimé au point M

6.2.4 Actions du tibia sur les capteurs

Les actions du tibia sur les trois capteurs sont les forces et moments suivants :

$\vec{F_T} = (F_{Tx}, F_{Ty}, F_T)$
$\vec{M_T} = (0,0,0)$ exprimé au point T
$\vec{F_G} = (F_{Gx}, F_{Gy}, F_G)$
$\vec{M_G} = (0,0,0)$ exprimé au point G
$\vec{F_D} = (F_{Dx}, F_{Dy}, F_D)$
$\vec{M_D} = (0,0,0)$ exprimé au point D

F_{Tx}, F_{Ty}, F_{Gx}, F_{Gy}, F_{Dx} et F_{Dy} non mesurées

6.3 Équations d'équilibre

6.3.1 Somme des forces nulle

$$\vec{F_p} + \vec{F_t} + \vec{F_M} + \vec{F_T} + \vec{F_G} + \vec{F_D} = \vec{0} \quad (1)$$

6.3.2 Somme des moments en O nulle

$$\vec{M_p} + \vec{M_t} + \vec{OM}\vec{F_M} + \vec{OT}\vec{F_T} + \vec{OG}\vec{F_G} + \vec{OD}\vec{F_D} = \vec{0} \quad (2)$$

On projette (1) sur l'axe z et (2) sur les axes x et y.

$$F_p - mgVA_z + F_T + F_G + F_D = 0$$

$$M_{px} - mg(y_M VA_z - z_M VA_y) + (y_T F_T - z_T F_{Ty}) + (y_G F_G) + (y_D F_D - z_D F_{Dy}) = 0$$

$$M_{py} - mg(z_M VA_x - x_M VA_z) + (z_T F_{Tx} - x_T F_T) + (z_G F_{Gx} - x_G F_G) + (z_D F_{Dx} - x_D F_D) = 0$$

En faisant l'hypothèse que les produits zF_x et zF_y sont négligeables, les actions de pression sont représentées par le torseur suivant exprimé au point O :

$$F_p = mgVA_z - F_T - F_G - F_D$$
$$M_{px} = mg(y_M VA_z - z_M VA_y) - y_T F_T - y_G F_G - y_D F_D$$
$$M_{py} = mg(z_M VA_x - x_M VA_z) + x_T F_T + x_G F_G + x_D F_D$$

6.3.3 Centre de pression

Soit $C = (x_C, y_C, 0)$ le centre de pression sur le pied considéré :

$$x_C = -\frac{M_{Py}}{F_p}$$
$$y_C = \frac{M_{Px}}{F_p}$$

6.4 Cas particulier de la semelle horizontale

$VA = (0, 0, 1)$,

$$F_p = mg - F_T - F_G - F_D$$
$$M_{px} = mgy_M - y_T F_T - y_G F_G - y_D F_D$$
$$M_{py} = -mgx_M + x_T F_T + x_G F_G + x_D F_D$$

6.5 Valeurs numériques

6.5.1 Positions des capteurs

$$x_T = 0$$
$$y_T = -68$$
$$z_T = 32$$
$$x_G = -60$$
$$y_G = 52$$
$$z_G = 32$$
$$x_D = 60$$
$$y_D = 52$$
$$z_D = 32$$

6.5.2 Masse et centre de masse

$$m = 1.9kg$$
$$x_M = 0$$
$$y_M = 32$$
$$z_m = 14$$
$$g = 9.81m.s^{-2}$$

Chapitre 7

Annexes relatives à l'évaluation des frottements

Moteur	**Tension mesurée** en volts
MotCheD_E	0,9
MotCheD_I	0,45
MotGenD_S	0,1
MothanD_S	0,15
MotCheG_I	0,66
MotCheG_E	0,85
MotGenG_S	0,1
MothanG_S	0,15
MothanD_F	0,6
MotHanD_V	0,6
MotHanG_F	0,6
MotHanG_V	0,6
MotLomb_V	0,7
MotLomb_D	1,7
MotLomb_G	1,9

TAB. 7.1: *Évaluation des frottements*

```
-----------------------------------------------------------------

     Procédure pour la mesure des frottements
                        Christine Azevedo
-----------------------------------------------------------------


Après avoir lancé la commande : bip

Deux fenêtres s'affichent :

   -  moinette qui donne accés au robot

   -  vio qui affiche les divers printf et scanf.


1. Accès au menu
-----------------

1- exécuter la commande : bipMenu dans la fenêtre moinette

2- initialiser en exécutant la commande i dans la fenêtre vio
   (ne pas se préoccuper du double echo)

Pour l'aide : h



2. Commande en tension
-----------------------

1- sélectionner le menu driver : d dans dans la fenêtre vio

2- choisir une partie du robot : b suivi de 0 pour la jambe
   droite, 1 pour la jambe gauche...

3- la commande c suivie des tensions permet d'actionner les
   moteurs concernés. La commande g permet de lire les
   tensions
```

```
3.  Exemple : pour évaluer les frottements sur le genou droit
-------------------------------------------------------------

1- Enclencher l'arret d'urgence

2- Lancer le menu :                          bipMenu
3- Initialiser et lancer le driver :         i
4- Choisir la jambe droite comme cible :     b 0

5- Replier manuellement le genou et le relâcher
   Rechercher la tension a envoyer au moteur pour
   que le genou quitte cette position d'équilibre.

c 0.0 0.0 -0.01 0.0
c 0.0 0.0 -0.05 0.0
c 0.0 0.0 -0.1  0.0


ATTENTION AU SIGNE DES TENSIONS !!!
```

Bibliographie

[AZE00] C.Azevedo and the Bip team. Control Architecture and Alghorithms of the Anthropomorphic Biped Bip2000. In *Proc. of International Symposium on Mobile, Climbing and Walking Robots - CLAWAR'2000*, 2000.

[ESP00] B.Espiau and P.Sardain. The Anthropomorphic Biped Robot BIP2000. In *IEEE, International Conference on Robotics and Automation, ICRA'00*, 2000.

[ESP97] B.Espiau and the BIP team. BIP: A Joint Project for the Development of an Anthropomorphic Biped Robot. In *International Conference on Advanced Robotics, ICAR'97*, 1997.

[GEN98] F. Génot. Contributions à la modélisation et à la commande des systèmes mécaniques de corps rigides avec contraintes unilatérales. In *Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble - France*, 1998.

[WIE01] P.B. Wieber. Modélisation et Commande d'un Robot Marcheur. In *Thèse de Doctorat, - France*, 2000.

[MR00] G.Baille P.Di Giacomo H.Mathieu and R.Pissard-Gibollet. L'armoire de commande du robot bipède bip2000. In *Rapport technique de l'INRIA*, 2000, n.0243.

[RR00] C. Azevedo, N. Andreff. Étude expérimentale des premières démarches du robot BIP2000. In *Rapport de recherche de l'INRIA*, 2000, n.4017.

[ARI99] S.Arias. Formalisation et Intégration en Vision par Ordinateur Temps Réel. In *Thèse de Doctorat, Université de Nice Sophia Antipolis*, 1999.

[ORC98] The ORCCAD Team. The ORCCAD Architecture. In *Robotics Research, Special issues on Integrated Architectures for Robot Control and Programming*, 1998, vol.17-4, pp338-359.

[SAR00] P.Sardain. Parametres de BIP2000 Doc A et B. In *Laboratoire de Mécanique des Solides*, 2000.

[SAR98] P.Sardain, M.Rostami and G.Bessonnet. An Anthropomorphic Biped Robot: Dynamic Concepts and Technological Design. In *IEEE, Trans. on Syst. Man and Cybernetix*, 1998, vol.28a, pp823-838.

[PAR99] J.J.Parmentier. Contribution à la commande d'un robot bipède. In *Rapport de stage, École Polytechnique - Paris*, 1999.

[KHA86] W. Khalil and J.F. Kleinfinger. A new geometric notation for open and closed-loop robots. In *IEEE, International Conference on Robotics and Automation*, 1986, pp1174-1180.

[GOG96] G. Gogu, P. Coiffet, A. Barraco. Représentation des déplacements des robots. In *Hermes*, 1996.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803