Le génie pour l'industrie

Implémentation du Whole Body Operational Space Control (W.B.O.S.C)

Par

Vincent Girardeau

GIRV15099604

Rapport de projet spécial

Montréal, le 2019-08-11

École de technologie supérieure

TABLE DES MATIÈRES

1. INTRODUCTION

Le projet consiste à implémenter les mathématiques et la logique inspirées d'un article scientifique dans le code du club Naova, qui est un club scientifique programmant des robots nommés Nao. Chaque année, le club participe au Championnat du monde de robotique, la RoboCup. La division à laquelle participe le club est la Standard Platform League (SPL), la division où des robots Nao jouent au soccer. Il est strictement interdit de modifier le robot à l'exception de son code.

Afin de se qualifier, l'équipe doit soumettre une nouveauté, un article scientifique. C'est pourquoi cette année, nous avons proposé d'implémenter un article scientifique sur une boucle fermée aussi appelé « closed-loop ». Le projet spécial portera donc sur l'implémentation d'une boucle fermée afin de contrôler les moteurs des Nao avec un courant électrique.

2. REMERCIEMENT

Tout abord, j'aimerais remercier le département de génie électrique de l'école de technologie supérieure d'avoir mis à sa disposition plusieurs ressources afin d'aboutir à ce projet. Notamment à Saad Maarouf, Kali Yassine et Fortin Jonathan. Ils ont fourni un support et un effort important pour que ce projet se termine dans les délais

Aussi, un remerciement particulier à April Alain pour avoir supervisé le projet.

Ensuite, je tiens à remercier les membres du club Naova d'avoir offert leur support sur l'implémentation de l'article dans leur code.

Pour finir je remercie toutes les autres personnes impliquées de près ou de loin au projet.

3. UNE BOUCLE FERMÉE DANS UN ROBOT NAO

3.1. Qu'est-ce qu'une boucle fermée

Avant d'aller plus loin dans l'explication du projet, il est nécessaire de définir ce qu'est une boucle fermée.

Nous pouvons retrouver plusieurs définitions de ce qu'est une boucle fermée, voici celle que je vais donner.

> Boucle fermée : Système de contrôle automatisé dans lequel une opération, un processus ou un mécanisme utilise le retour d'information (en anglais feedback) de ce même processus afin de réduire les erreurs.

En d'autres termes, nous utilisons une boucle fermée dans un robot afin de corriger la sortie d'une fonction à l'aide du retour d'information de cette dernière.

Actuellement, le code du robot Nao utilise seulement des boucles ouvertes, c'est-à-dire sans utiliser le retour d'information.

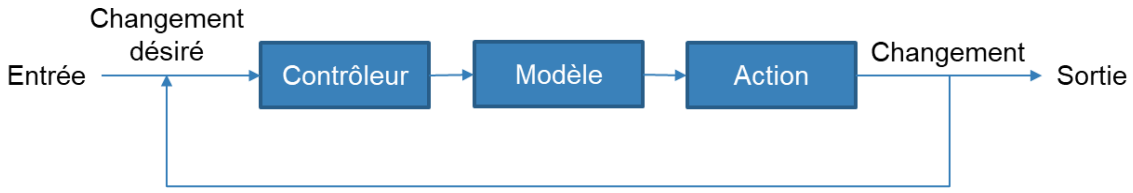

Figure 1 : Schéma d'une fonction à boucle ouverte

Sur la *Figure 1*, on peut voir le contrôleur recevoir un changement désiré. Il va ensuite faire toutes les actions nécessaires pour faire ce changement aux modèles puis va envoyer un message de sortie lorsqu'il aura fini ses actions. Ici, l'entrée ne récupère aucune information sur l'exécution du cycle précèdent. La plupart des fonctions d'un programme fonctionnent de cette manière.

Une boucle fermée fonctionnera de la même manière à l'exception que l'on va utiliser le résultat en sortie pour l'exécution suivante.

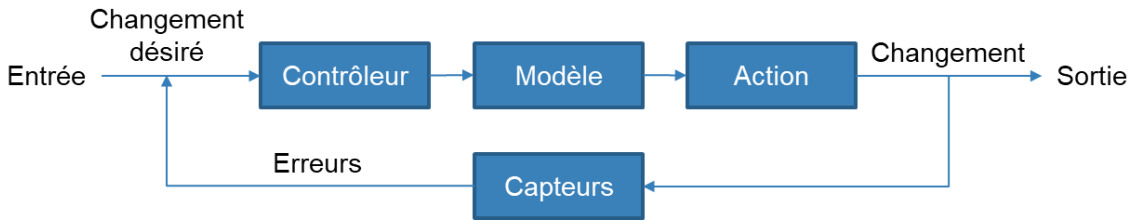

Figure 2 : Schéma d'une fonction à boucle fermée

Sur la *Figure 2*, il est possible d'apercevoir que la sortie est récupérée et est analysée ensuite par d'autres fonctions, ou bien dans notre cas par des capteurs. On produit ensuite des erreurs que l'on va transmettre au prochain cycle afin de les corriger dans la prochaine exécution. Cela montre qu'avec une boucle fermée, nous pouvons être plus précis dans nos exécutions.

Avec une fonction de ce type, il sera possible de faire beaucoup de choses non permises de base par un robot Nao.

3.2. Pourquoi implémenter une boucle fermée

Premièrement, l'un des principaux problèmes des robots Nao est la limitation par le constructeur sur les contrôles possibles.

Dans notre cas, implémenter l'article permettra au robot de se déplacer plus rapidement. Pour ce faire, il est nécessaire de contrôler les moteurs avec un courant électrique, car actuellement nous utilisons des positions avec des angles en degré. Le constructeur a décidé de bloquer le contrôle par courant électrique. Le moyen d'y remédier est de passer par une boucle fermée afin de simuler un robot virtuel.

Envoyer un courant plutôt qu'une position en degré permet d'avoir plus de précision sur l'angle du moteur. En effet, il n'est pas nécessaire de faire des conversions pour envoyer un courant à un moteur électrique. En revanche, envoyer une position, il faudra faire plusieurs calculs pour convertir cet angle désiré en courant électrique désiré.

4. FONCTIONNEMENT DU CODE DE NAOVA

4.1. La représentation

Le code de Naova est divisé en plusieurs parties. L'une des plus importante est la représentation. Il faut voir la représentation comme une interface dans les langages orientés objet. C'est-à-dire quelle va définir les méthodes et variables du module.

Chaque module peut en appeler un autre à l'aide de méthode définie dans la représentation de ce dernier.

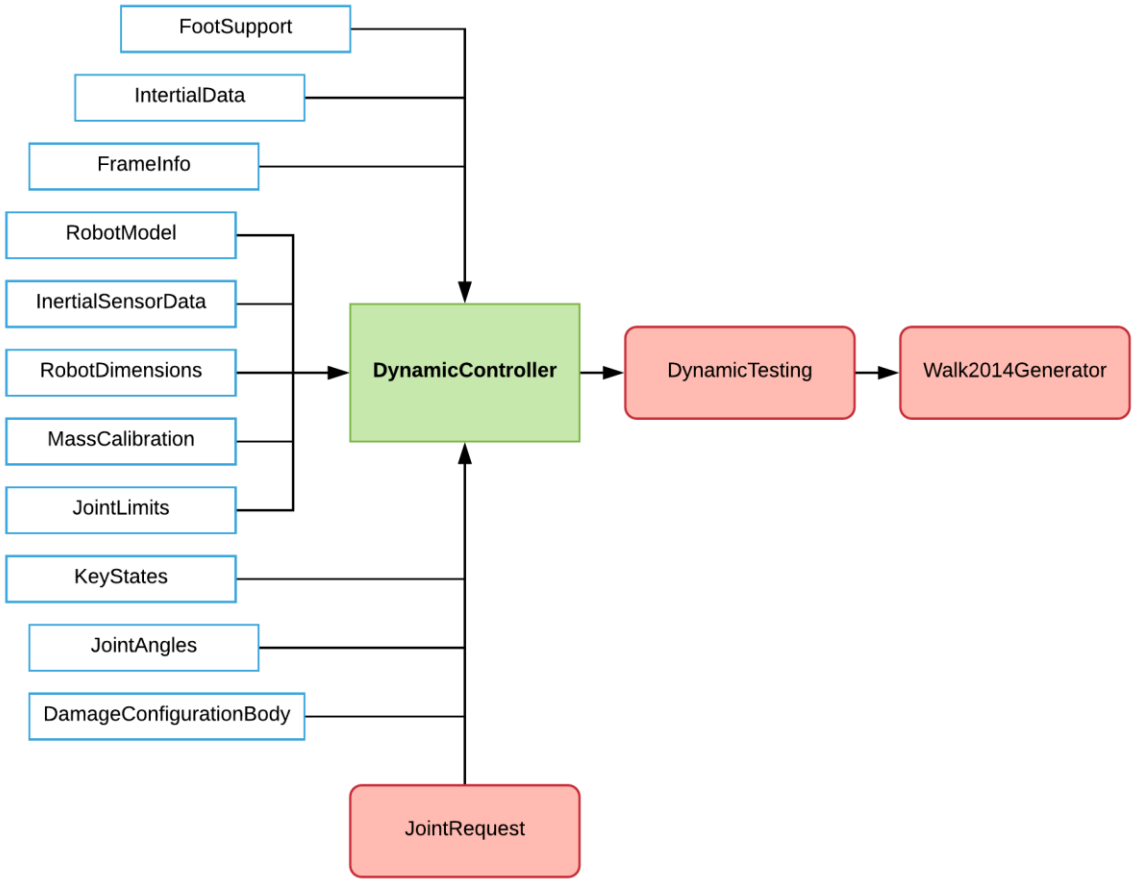

Figure 3 : Schéma des appels fait entre le module à implémenter et les autres [1]

Sur la *Figure 3*, on peut apercevoir des modules ainsi que des représentations. Les représentations sont de couleur rouge.

Dans le cas de notre implémentation, nous avons nommé la représentation de notre module DynamicTesting.

4.2. Le module

Il faut voir le module comme une classe. C'est ici que nous aurons toutes les logiques et les calculs. Nous devons implémenter les méthodes définies dans la représentation, mais nous pouvons aussi en ajouter d'autre, qui seront privées aux autres modules.

Il va aussi appeler d'autres modules afin de récupérer certaines informations.

Un module comporte au moins une méthode nommée *update()*. Elle sera appelée à chaque cycle d'exécution. Elle est obligatoire dans chaque module et permet de mettre à jour les informations stockées.

Dans notre cas, nous avons défini une deuxième méthode, nommée *init()* permettant d'initialiser notre module.

Dans la *Figure 3,* les modules se trouvent en vert et bleu. Celui que nous avons créé s'appelle DynamicController. On peut voir qu'il appelle plusieurs représentations et autres modules afin de récupérer les informations nécessaires à son fonctionnement.

4.3. La configuration

Afin de faciliter le changement de paramètre sans recompiler le code, un système de configuration est en place dans le code. Voici un exemple de fichier dans le cadre de notre projet.

dynamicController.cfg 81 Bytes

```
beta = 150;
Kp = -0.4;
Kd = 0.1;
K1 = 500;
K2 = 2100;
A = 1e-05;
theta0 = 1e-05;
```

Figure 4 : Exemple d'un fichier configuration du module à implémenter

Sur la *Figure 4,* nous pouvons voir que nous avons passé tous les paramètres que nous devrons modifier pour stabiliser le robot ou augmenter la vitesse de la marche.

Chaque module ne peut être lié qu'avec un fichier configuration. Pour permettre d'en gérer plusieurs, une hiérarchie bien spécifique a été mise en place. Voici une liste détaillée de comment sont lu les fichiers de configuration :

- *Dans le dossier d'un robot :* Permets d'avoir le fichier spécifique pour le robot. Cela peut être utile par exemple sur les fichiers de calibration des moteurs.

- *Dans le dossier d'une localisation :* Permets d'avoir le fichier spécifique pour un terrain précis. Si le fichier n'existe pas pour le robot, il va aller chercher dans ce dossier.
- *Dans le dossier d'un scénario :* Permets d'avoir un fichier spécifique à chaque scénario. Par exemple si on veut un scénario bien précis sur n'importe quel terrain et avec chaque robot, c'est ici qu'il faut mettre le fichier. Comme pour le dossier précédent, si le fichier existe dans celui de location ou de robot, il va utiliser celui-ci.

Le schéma en Annexe 1 montre le processus de sélection détaillé pour un fichier configuration.

5. IMPLÉMENTATION DE LA BOUCLE FERMÉE

5.1. Objectifs d'implémentation

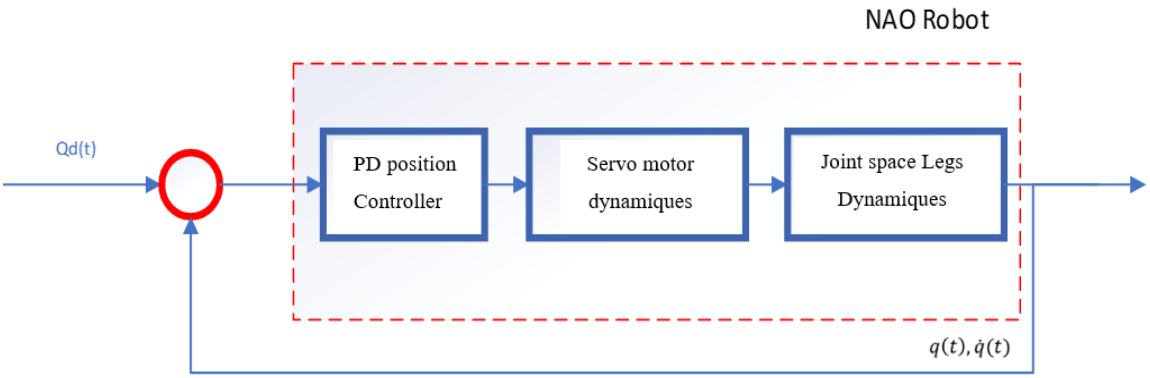

Figure 5 : Schéma du fonctionnement du module avec la boucle infinie [1]

La *Figure 5* montre comment fonctionne une boucle fermée au niveau des moteurs dans un robot Nao. Nous pouvons voir que le système envoie une position désirée au contrôleur gérant la marche. Ensuite, une fois l'opération exécutée, nous nous retrouvons avec une position réelle que nous allons récupérer pour l'exécution suivante.

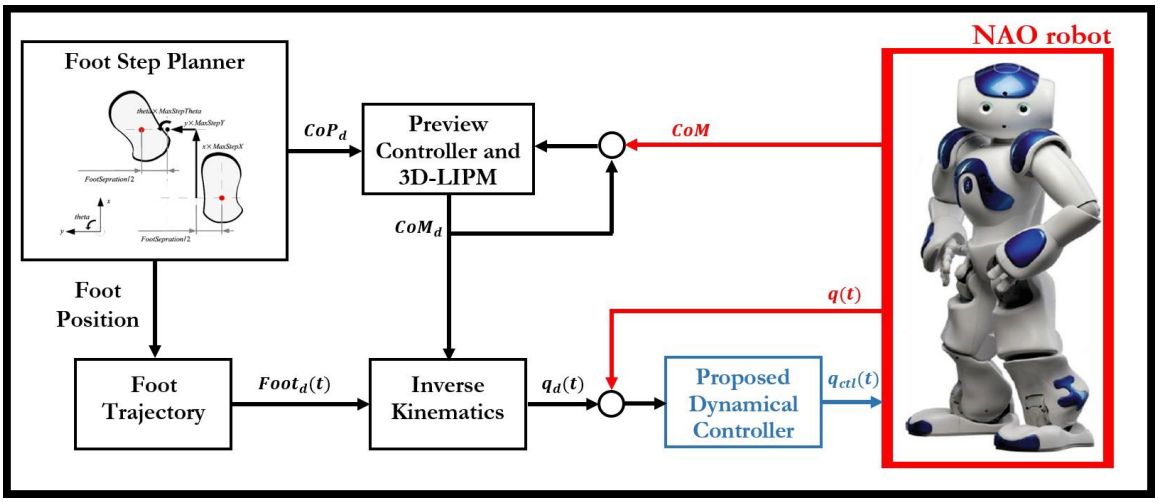

Figure 6 : Schéma du fonctionnement du WalkEngine avec notre module en plus [2]

La *Figure 6* est une vue bien plus détaillée de ce qu'on peut retrouver sur la *Figure 5*. Elle indique aussi spécifiquement comment la WalkEngine fonctionne dans le code du club Naova et comment implémenter notre boucle fermée.

5.2. Quelques formules mathématiques importantes

Plusieurs fonctions mathématiques sont nécessaires à l'implémentation et il est important de les comprendre.

Équation 1 : Calcul du sliding mode

$$\sigma(t) = \dot{\tilde{\mathrm{q}}}(t) + \beta\tilde{\mathrm{q}}(t)$$

Ce calcul utilise l'erreur de la vitesse et l'erreur de position calculée précédemment afin de s'assurer que les positions du système atteignent leurs positions désirées dans un temps finit.

Équation 2 : Fonction sign

$$sign\big(\sigma_i(t)\big) = \begin{cases} \text{-1}, & if\ \sigma_i(t) < 0 \\ 0, & if\ \sigma_i(t) = 0 \\ 1, & if\ \sigma_i(t) > 0 \end{cases}$$

Équation 3 : Calcul du TDE[1]

$$\widehat{\mathrm{H}}\big(q(t), \dot{\mathrm{q}}(t), \ddot{\mathrm{q}}(t)\big) \cong H\big(q(t\text{-}\Delta t), \dot{\mathrm{q}}(t\text{-}\Delta t), \ddot{\mathrm{q}}(t\text{-}\Delta t)\big) = \tau_M(t - \Delta t) - \overline{\mathrm{A}}\ddot{\mathrm{q}}(t - \Delta t)$$

Le calcul du TDE permet de reconstruire toute la dynamique utilisant notamment les états du système et les entrées de torque calculé.

Équation 4 : Calcul du torque avec TDE et SMC[2]

$$\tau_M(t) = \bar{A}\big[\ddot{q}_d(t) - \beta\dot{\tilde{q}}(t) - K_1\big(\sigma(t)\big)\sigma(t) - K_2 D\big(\sigma(t)\big)sign\big(\sigma(t)\big)\big] + \widehat{\mathrm{H}}\big(q(t), \dot{\mathrm{q}}(t), \ddot{\mathrm{q}}(t)\big)$$

Le calcul du torque se fait en utilisant le TDE et le SMC précédemment calculé. Nous utilisons aussi la fonction sign.

Équation 5 : Transformation du torque en position

$$qctl(t) = q(t) + K_p^{(-1)}\big(K_d\dot{\mathrm{q}}(t) - \tau_{\mathrm{M}}(t)\big)$$

La transformation du torque que nous venons de calculer se fait avec une image inversée du système linéaire des moteurs.

[1] TDE: Time Delay Estimation
[2] SMC: Sliding Mode Control

5.3. Pseudo-code

Voici le pseudo-code de ce qui a été implémenté.

Data: Define: β, $\bar{A}$, K_1, K_2, K_p, K_d, σ_0, d_i, l_i, $\tau_M(0)$, Δt, N % number of samples
initialization j==0;
while $j!=N$ **do**
  define: $q_d(j)$, $\dot{q}_d(j)$, $\ddot{q}_d(j)$;
  read $q(j)$, $\dot{q}(j)$;
  compute acceleration: $\ddot{q}(j) = (\dot{q}(j) - \dot{q}(j-1))/(t(j) - t(j-1))$;
  compute error of position: $\tilde{q}(j) = q(j) - q_d(j)$;
  compute error of velocity: $\dot{\tilde{q}}(j) = \dot{q}(j) - \dot{q}_d(j)$;
  define sliding surface: $\sigma(j) = \dot{\tilde{q}}(j) + \beta\tilde{q}(j)$;
  compute torque $\tau_M(j) =$;
  TDE: $H(j+1) = \tau_M(j) - \bar{A}\ddot{q}(j)$;
  torque to position transformer: $q_{ctl}(j) = q(j) + K_p^{-1}\left(K_d\dot{q}(j) - \tau_M(j)\right)$;
  j++;
end

Figure 7 : Pseudo-code du module [1]

On peut apercevoir dans la *Figure 7* notre boucle fermée. En effet, nous récupérons les informations à t-1 afin de corriger l'erreur à l'instant t. En réalité, pour calculer l'accélération, nous utilisons l'information sur les moteurs à t-2 et t-1.

Nous pouvons voir au début, des constantes définies, comme par exemple d_i et l_i. Dans le but d'accélérer l'implémentation et dû à la difficulté de les définir, nous nous sommes permit des les déclarer à 0.

Omettre ces valeurs n'aura pas un grand impact sur le robot, d_i et l_i sont multipliées dans un calcul, corrigeant un peu plus l'angle désiré. Elles permettent seulement à stabiliser encore plus le robot.

5.4. Résultats

5.4.1. Changements sur les moteurs

On peut voir sur la *Figure 4*, les paramètres utilisés pour notre module.

Il faut savoir que ces paramètres sont extrêmement importants. En effet, nous avons débuté avec des valeurs très faibles, ce qui a causé des problèmes de stabilité au robot, il n'arrivait plus à se déplacer tout droit.

Une fois les paramètres bien configurés et le robot stable, nous avons récupéré des résultats sur les moteurs. Nous avons enregistré les positions des moteurs désirées avant puis après la correction de notre module.

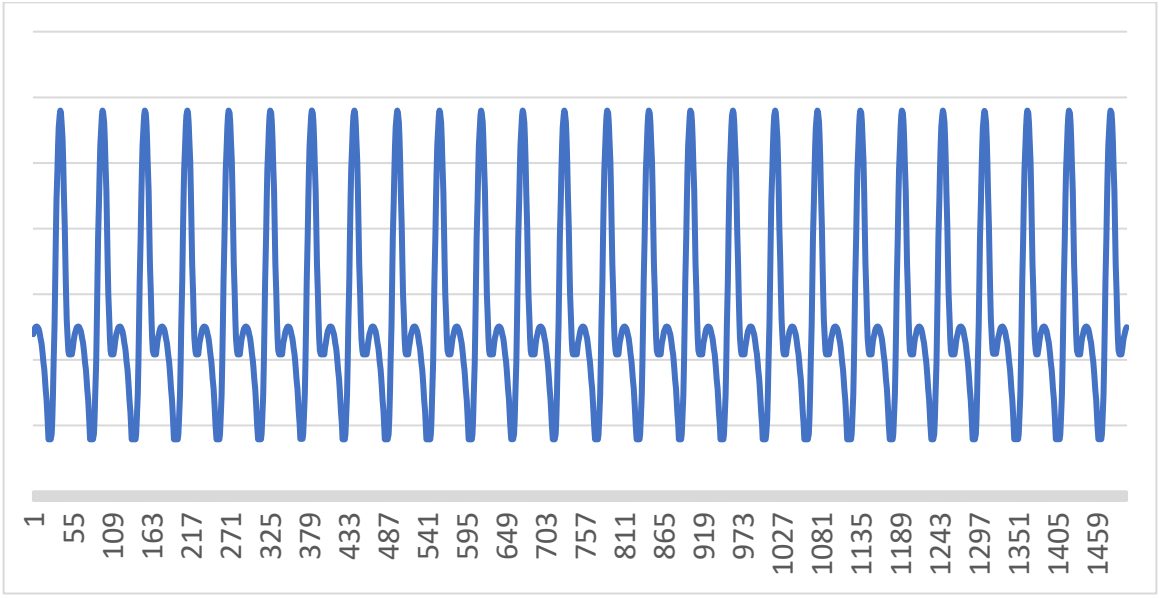

Figure 8 : Graphique de l'angle désiré avant correction pour la cheville droite [2]

La *Figure 8* représente l'angle désiré reçu par le module InverseKinematic, c'est-à-dire avant la modification de l'angle par notre module

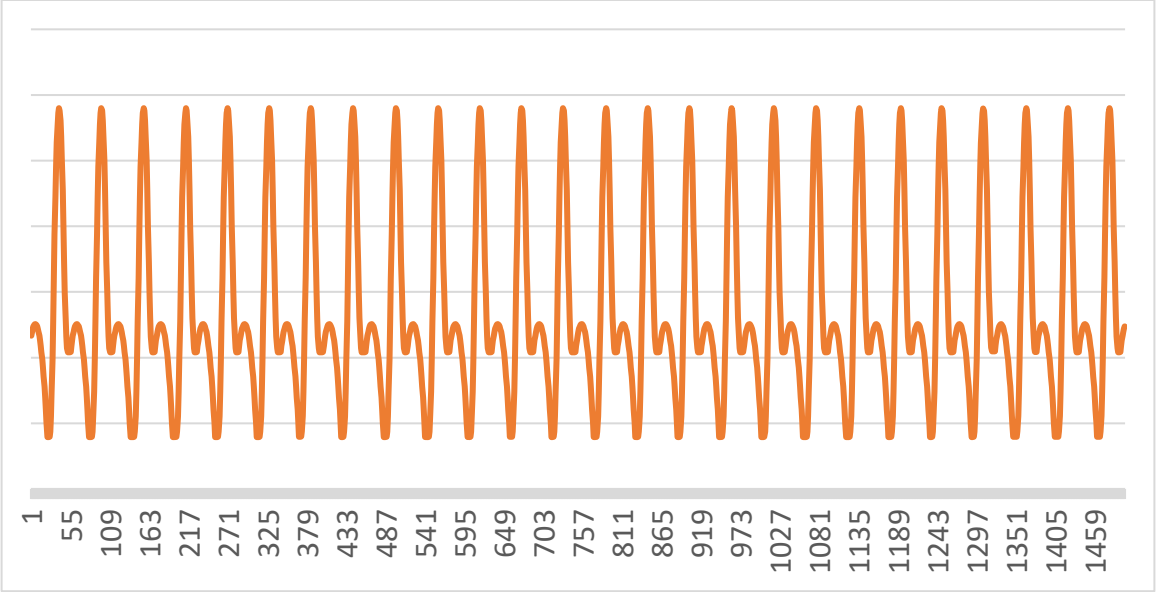

Figure 9 : Graphique de l'angle désiré après correction pour la cheville droite [2]

La *Figure 9* représente l'angle désiré modifié par notre module.

On aperçoit en comparant la *Figure 8* et la *Figure 9* que la position envoyée aux moteurs suit le même mouvement. Cela montre notamment que notre module n'affecte pas énormément l'angle, ce qui est bon signe et indique qu'il n'y a pas de grosse erreur d'implémentation.

Pour être sûrs que la position change bien, nous avons prélevé une dernière information.

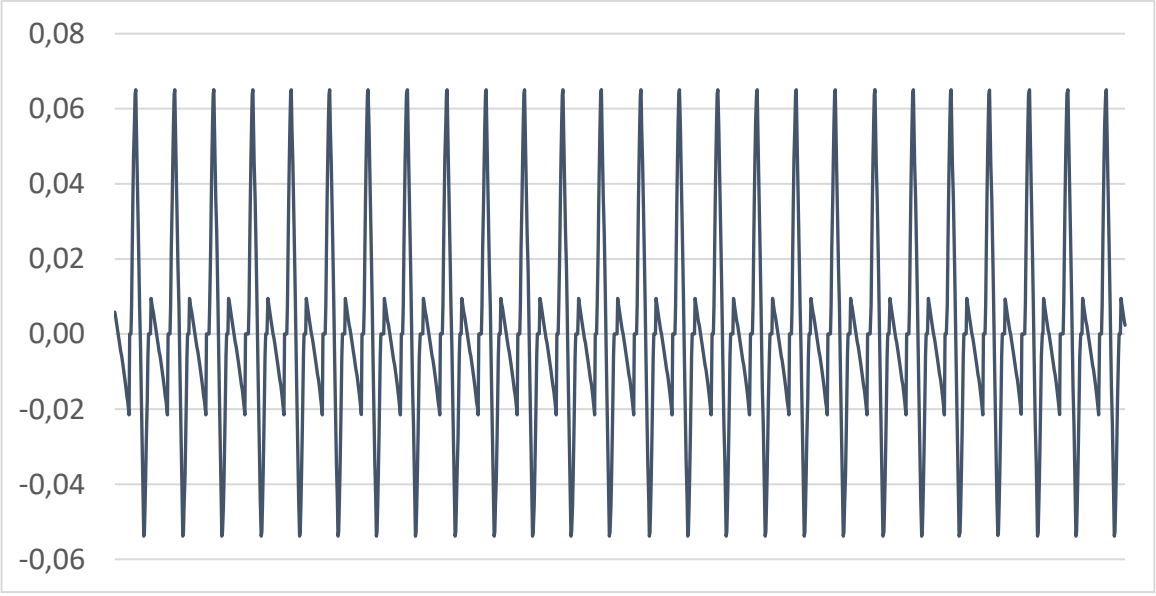

Figure 10 : Graphique de la différence entre l'angle avant et après correction

Sur la *Figure 10*, nous pouvons voir la différence entre l'angle désiré avant notre modification et celui après notre modification. L'axe vertical est en radian et l'axe horizontal en seconde et utilise la même échelle que la *Figure 8* et la *Figure 9*.

Nous apercevons que notre module effectue bien des modifications sur l'angle. La correction varie entre environ 0,06 rad et -0,06 rad.

En analysant tous les graphiques, nous avons pu déterminer que selon les moteurs la variation minimum est de 0,007 rad et la variation maximum est de 0,15 rad. La moyenne de la variation est de 0.0702 rad.

5.4.2. Problèmes rencontrés

Pour commencer, le principal problème rencontré au début du projet est la compréhension des formules mathématiques. Mais à l'aide des PhD, nous avons pu nous assoir afin d'avoir beaucoup plus d'explication.

En revanche, nous avons rencontré un gros problème, qui malheureusement ne pourra être corrigé tout de suite. En effet, nous avons remarqué que l'algorithme ne prenait pas en compte les tremblements au niveau de la tête causée par une marche plus rapide du robot. C'est pourquoi, le robot avait du problème à visualiser les objets, et par conséquent, à voir la balle.

Ensuite, l'autre problème est celui pour trouver les paramètres. En effet, nous n'avions pas de robot avant de partir en compétition, ce qui nous a empêchés de tester et trouver les bons éléments avant la compétition.

Le dernier problème est le mauvais paramétrage du module WalkEngine. Nous avons remarqué que ses paramètres étaient très importants et pouvaient comporter un risque au robot. Nous avons d'ailleurs brulé plusieurs moteurs avec des paramètres trop élevés.

5.4.3. Récapitulatif

Voici un récapitulatif des points positifs et négatifs de l'implémentation de ce module dans sa version actuelle.

Les points positifs	Les points négatifs
- Une marche plus rapide	- La tête n'est pas gérée dans l'algorithme
- Naova était l'équipe avec les robots les plus rapides durant la compétition	- Manque de stabilité si les paramètres de marche ne sont pas bien configurés
- La possibilité de contourner les limitations imposées par le constructeur pour les commandes des moteurs	- Possibilité de briser ou de bruler des composants du robot, notamment les engrenages et moteurs.

6. AMÉLIORATION POSSIBLE

Plusieurs améliorations sont possibles. Pour commencer, il est possible d'implémenter les paramètres omis (d_i et l_i) afin de stabiliser encore plus le robot. Cette amélioration est d'ailleurs déjà en cours de réalisation.

Ensuite, il serait préférable de prendre en compte la caméra dans les mouvements effectués. En effet, le fait que le robot ait plus de vibration au niveau de la tête cause un problème de vision qui peut être corrigé par un algorithme ou par la modification des formules mathématiques déjà implémentées.

Pour finir, maintenant que la base de la boucle fermée est présente dans le code, il est possible d'ajouter de nouvelles commandes torque sur les moteurs des jambes afin d'améliorer la marche du robot ou bien tout simplement implémenter une boucle fermée sur d'autres parties du corps.

7. CONCLUSION

Pour conclure, nous avons vu différents aspects du projet comme l'importance d'une telle implémentation ou bien les dangers de mal configurer des paramètres.

Nous avons vu comment implémenter un nouveau module dans le code source de Naova et comment fonctionne la hiérarchie dans ce dernier.

Le projet a pu aboutir à sa fin, ouvrant la porte à la suite de la modification complète du WalkEngine actuel.

Pour finir, nous avons repoussé les limites du robot et celles imposées par le constructeur afin de le rendre plus rapide dans ses mouvements.

Il est aussi important de noter que grâce à cette implémentation les robots du club Naova étaient les plus rapides durant la RoboCup 2019.

8. RÉFÉRENCES

[1] D. Kim, S. Jens Jorgensen et P. S. Stone, «Dynamic Behaviors on the NAO Robot With Closed-Loop Whole Body Operational Space Control,» Austin, 2016.

[2] J. Fortin, « Une approche différente pour rendre plus rapide et plus stable le robot bipède Nao, » Montréal.

[3] Y. Kali, B. Brahmi, J. Fortin et V. Girardeau, «Walking Task Space Control using Time Delay Estimation based Sliding Mode of Position Controlled NAO Biped Robot,» Montréal.

9. ANNEXES

Annexe 1 : Processus de sélection d'un fichier configuration