ensil-ensci ÉCOLE D'INGÉNIEURS DE LIMOGES

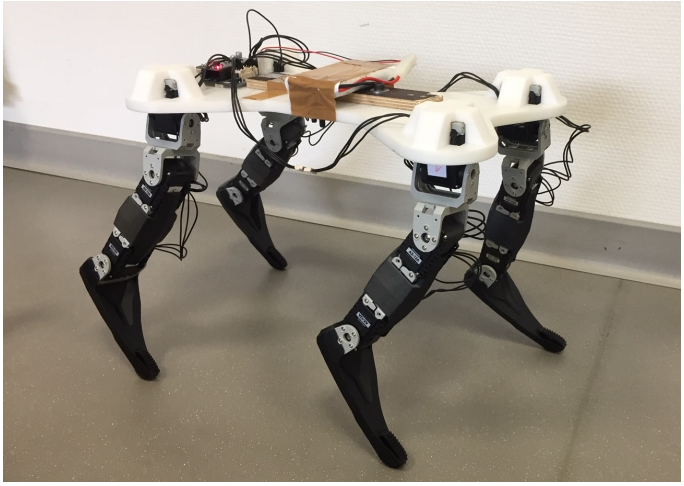
École nationale supérieure d'ingénieurs de Limoges

Rapport de Projet

Conception et réalisation d'un robot quadripode

Etudiants :
Thomas ARCHER
Géraud MARTIN-MONTCHALIN

Encadrant :
Stephane RENAULT



23 juin 2019

Table des matières

1 Introduction

Aujourd'hui, à l'heure ou la robotique industriel se développe prodigieusement dans certaines régions du globe, de nombreuses entreprises créent et présentent leurs prototypes de robots tétrapode. Parmi elles nous pouvons retrouver : *Spot mini* de Boston Dynamics, *Laikago* de Unitree Robotics, *Anymal* de Anybotics et bien d'autre. Tous ces modèles déjà très performant et optimisés ayant défriché le domaine d'étude nous permettrons ainsi d'avoir une idée précise des lignes à suivre pour modéliser et concevoir efficacement notre robot.

Un 1^{er} prototype de robot quadripode à déjà été réalisé lors de projets précédents à l'ENSIL-ENSCI. Il disposait de 8 articulations actionnables pour se déplacer et les servomoteurs ne semblaient pas assez performants. Nous nous épaulerons dessus et nous utiliserons en outre cette année un matériel plus performant qui nous a été fournis par notre tuteur Mr. Renault en début d'année.
Nous avons ainsi eu la chance de commencer le projet avec douze moteurs Dynamixels $AX-12A$, une carte $OpenCM9.04$, un étage de puissance $OpenCM$ 485 EXP et une batterie externe de 9.6V et 2000mAH. Cela nous a donné la possibilité dès lors de nous concentrer directement sur le coeur du projet.

Pour mener à bien ce projet, nous allons dans un premier temps réaliser une modélisation pièces par pièces du robot que nous voudrons assez satisfaisantes pour pouvoir réaliser la partie qui suit.
Une fois les pièces réalisées et le robot assemblé, nous allons pouvoir commencer la partie de définition de trajectoires et démarrer les tests. Une grande partie du projet serra donc des essais réalisés afin de valider la conception mécanique mais aussi la partie programmation du robot.

2 Modélisation/conception du robot

Dans cette partie nous allons discuter des choix et de l'évolution des parties mécaniques. Pour commencer, le choix de l'impression 3D c'est fait pour plusieurs raisons ; le poids, la flexibilité des formes mais aussi afin de pouvoir facilement apporter des modifications sur les pièces. Lors de l'évolution du projet, nous sommes passés par beaucoup de prototypes pour le robot. l'exposé de ceux ci se ferra pièces par pièces et non version par version.

En ce qui concerne la forme globale, elle est inspirée de l'existant, autant en termes de robots quadrupèdes évoqués en introduction que de l'animal. Comme à l'origine du projet, nous possédions 12 moteurs, nous avons pour chaque jambe 3 moteurs. Toutes les jambes sont identiques et le choix à été fait de les modéliser de cette façon.

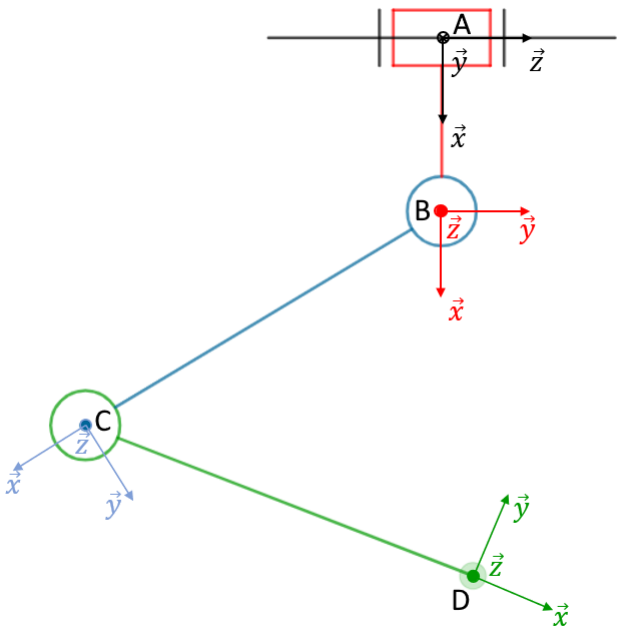


FIGURE 1 – modélisation du robot cinématique

Sur cette figure, la partie noire représente le corps, les trois autres pièces forment un "bras" et le point D est le point en contact avec le sol. Grâce à ces 3 orientations des articulations, la jambe peut se déplacer dans les 3 directions de l'espace. Cela lui laisse alors assez de liberté pour pouvoir marcher d'avant en arrière mais aussi pour pouvoir bouger ses pattes sur le coté ce qui lui permettra plus tard de tourner sur lui même. Les distances retenues entre les points on étés au départ choisies telles que chaque patte fasse 150mm mais après le montage et les premiers tests de marche, nous avons pris la décision de réduire cette distance (à 120mm). On se retrouve donc avec $[A, B] = 50mm, [B, C] = 120mm, [C, D] = 120mm$ Le robot se divise donc en $1 + 3 * 4 = 11$ parties différentes :

— 1* Le corps (partie noire sur la figure ci-dessus))
— 4* La partie verte entre A et B, appelée "épaule" par la suite
— 4* La partie bleue entre B et C, appelée "bras arrière"
— 4* La partie rouge entre C et D, appelée "bras avant".

Nous verrons pour chacune de ces pièces les choix, problèmes rencontrés, solutions et améliorations possibles. En plus des pièces que nous allons créer, nous avons 2 autres pièces qui données avec les moteurs nous permettent de mettre en place un intermédiaire entre nos pièces et ces derniers. Ces pièces sont appelées "FP04-F3" et "FP04-F2".

2.1 Épaule

la partie "épaule" est simplement faite à l'aide des pièces "FP04-F2" fournies avec les moteurs. La fixation est simple, faite avec les vis fournies elles aussi dans le pack des moteurs.

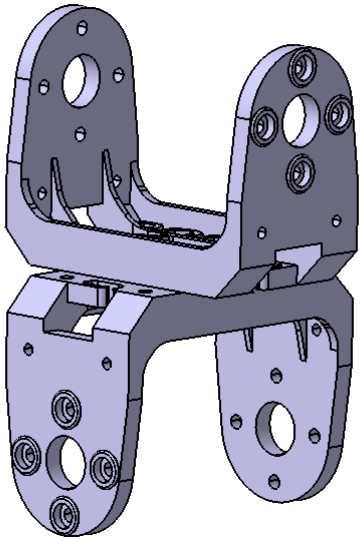


FIGURE 2 – pièce "épaule" du robot

Après réflexion, une solution plus simple pour la commande et la fixation de la jambe serais de modifier cette partie afin retourner le moteur du haut. Cela reviendrait à intégrer le moteur à l'épaule. En modifiant cela, il faudrait modifier quelques fonctions dans nos programmes mais le gros avantage serait la fixation au corps, avec la pièce "FP04-F3" qui laisse plus d'espace entre elle et le moteur et qui permet directement d'y placer une vis. Autrement, c'est une partie assez simple sur laquelle aucun changement n'a été fait.

2.2 Corps

Concernant le corps, il est composé de deux pièces identiques, qui s'emboîtent l'une dans l'autre pour former le corps en entier. Nous sommes restés sur un design très très simplistes, nous concentrant plus sur la marche que sur le design du robot. C'est pour cela que le corps reste très grandement améliorable.

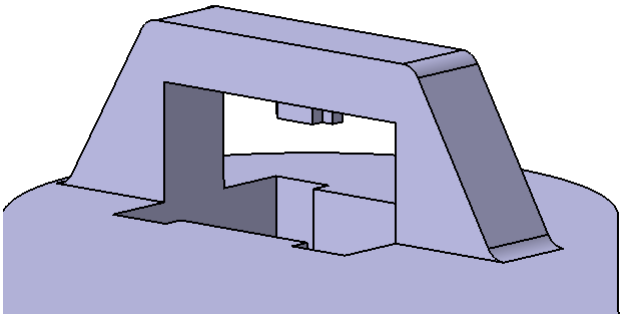


FIGURE 3 – Partie assurant la fixation de la jambe sur le corps

le modèle a beaucoup évolué, le point majeur sont les fixations (figure ci-dessus), au premier

prototype trop fines, elles cassaient notamment à l'intersection des couches créées par l'impression. Immédiatement les modifications ont été de renforcer cette partie. Cela c'est fait par un épaississement et une modification de la forme des angles qui étaient à l'origine de la cassure.

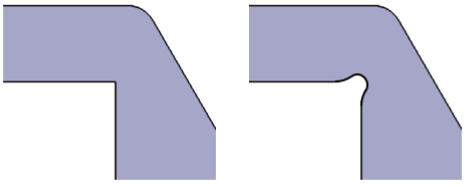


FIGURE 4 – Angles droits défectueux avant et après

Concernant les améliorations à apporter,

- la première chose à modifier est la fixation des jambes, actuellement c'est par serrage que les pattes sont fixées. cela a l'avantage d'être plus simple à monter/démonter mais dans l'objectif de faire un modèle fini, il serait préférable de le fixer avec une vis/écrou.
- Renforcer l'assemblage des deux parties du corps est plus que nécessaire. Par exemple prévoir un emplacement pour faire passer des barres de carbone, acier ou autre matériau bien plus solide que le plastique utilisé en impression 3D serait une bonne solution.
- Un emplacement pour la batterie et la partie électronique manquent aussi au corps. Cela pourra être ajouté à l'existant ou modifié directement sur la pièce.
- La Dernière modification serait donc le design de la pièce qui pour l'instant ressemble beaucoup à un banc d'essai.

2.3 Bras arrière

La fabrication des bras arrières n'a pas eu besoin de beaucoup de versions. Le maintien des moteurs se fait par serrage. Le seul changement c'est opéré sur la méthode d'impression. Nous obtenons aussi le même soucis qu'avec le bâtit, quand les couches de plastique sont alignés avec les angles droits de la pièces cela l'affaiblit. Pour résoudre simplement ce problème, nous avons appliqué une légère rotation à la pièce lors de l'impression. afin que les lignes d'impressions soient inclinées par rapport aux axes principaux de la pièce.

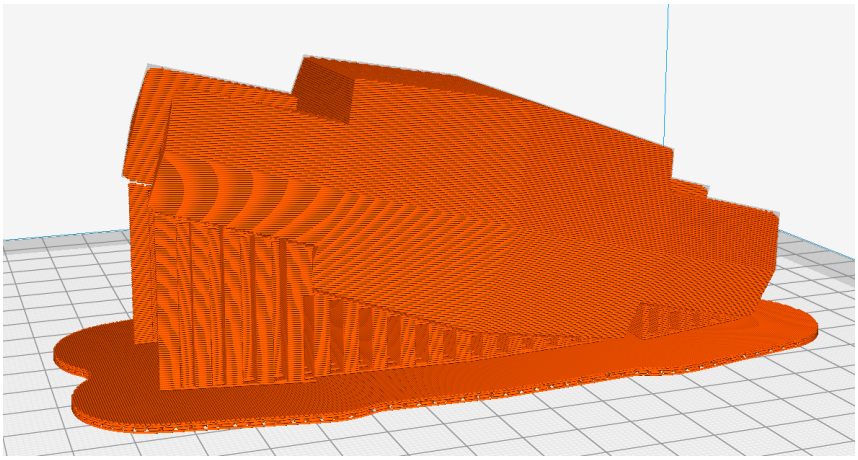


FIGURE 5 – Positionnement de la pièce pour l'impression

Avec ce simple petit changement, le problème de fragilité est résolu sans modifier la pièce.

2.4 Bras avant

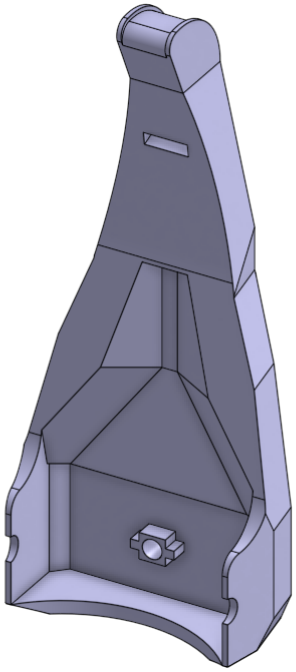


FIGURE 6 – Pièce jouant le rôle de l'avant bras

L'avant bras est la pièce ayant eu le plus de modification tout au long du projet, notamment au niveau des fixations. Au départ, la fixation était faite par serrage, puis rapidement la solution de la vis/écrou à été choisie. L'avant bras du robot à une forme particulière pour pouvoir plier le bras complètement au repos. comme sur la photo suivante :

FIGURE 7 – patte du robot en position couchée

La fixation de celui ci avec la pièce "FP04-F2" (sortie de moteur) est faite a l'aide d'une simple vis/écrou. La forme du bras avant épouse la pièce et donc sert au guidage pour la fixation.

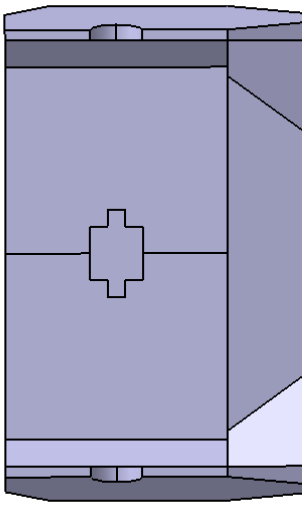


FIGURE 8 – Partie de l'avant bras permettant la fixation sur la pièce FP04-F2

L'adhésion au sol est réalisée par un antidérapant qui est glissé dans une poche le long du bout du pied du robot.

FIGURE 9 – Antidérapant autour du bout de l'avant bras

Cette pièce n'a pas d'améliorations vraiment utiles à atteindre.

3 Modélisation de la marche

Dans ce chapitre, nous allons suivre deux solution différentes permettant de réaliser la marche de notre robot.
La première solution consistera en l'interpolation polynomiale des angles des moteurs en fonction des coordonnées cartésiennes (X, Y, Z) du repère outil suivant la trajectoire voulu projeté dans le repère du bâtit. Ces interpolations nous permettrons d'être précis dans un rayon de quelques centimètres autour de la trajectoire choisit pour le repère outil. Enfin, la seconde solution consistera en l'utilisation d'un module de Matlab nommé *Robotics Toolbox*. Ce dernier nécessitera de définir la marche au préalable puis nous renverra un tableau correspondant aux valeurs des angles de chaque moteurs à chaque étape du pattern de marche.

3.1 Avec Géogebra

3.1.1 Modélisation d'une patte et de ses trajectoires

Les données avec lesquelles nous allons travailler ici sont uniquement :
— Les coordonnées de l'effecteur x,y et z (point P sur la figure suivante)
— Les angles correspondant à ces coordonnées.

La question de solutions multiples ne vas pas être abordé ici. Le problème est résolu par l'initialisation qui est faite dans la création de la jambe. Concernant cette création, il faut placer les points sur des cercles qui décrivent leur trajectoires (le troisième moteur tourne autour du premier sur un cercle de 12cm et l'effecteur tourne autour du troisième moteur de la même façon). Avec de la simple géométrie nous arrivons donc à créer une jambe de robot en 3D. Nous pouvons sur cette modélisation déplacer l'effecteur relativement à la base (haut de la jambe). Il suffit ensuite de créer un angle (entre les vecteurs qui nous intéressent) et sa valeur pourra être lisible.

La trajectoire que doit suivre une patte ne peut être contrôlé uniquement par la valeur des angles des moteurs. nous devons donc pour chaque position de l'effecteur connaître quelle est la configuration correspondante du robot et les valeurs de position à donner aux moteurs

Pour déterminer les angles moteurs dont nous avons besoin, nous allons passer par des fonctions qui prendrons en entrée la position du repère outil de la patte (ce qui nous permettra de définir la trajectoire facilement) et sortirons les angles correspondant des moteurs : θ_1, θ_2 et θ_3.

Au final nous nous retrouvons avec un modèle comme ceci :

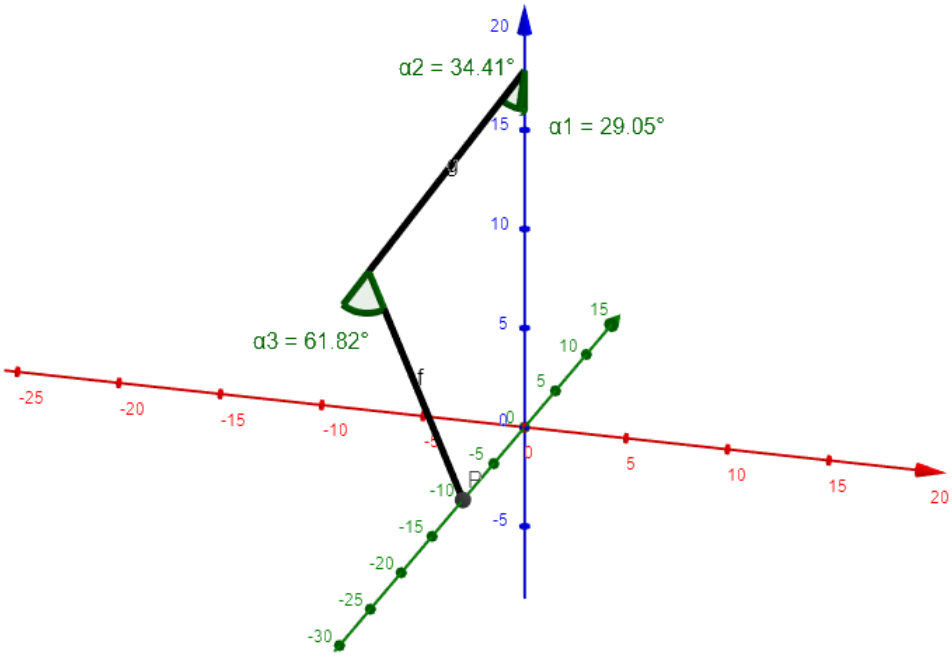


FIGURE 10 – modélisation du robot et visualisation des angles

3.1.2 Création des fonctions d'interpolation

Maintenant la modélisation faite, l'objectif est donc de prendre quelques points le long d'une trajectoire comme celle qui suis :

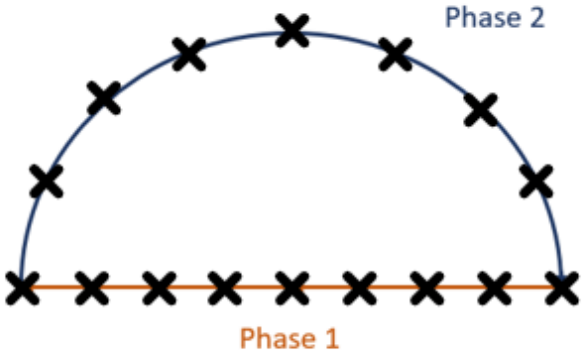


FIGURE 11 – Trajectoire désirée du repère outil

ensuite pour chacune des phases sur le logiciel nous réglons la jambe dans les configurations qui correspondent à ses points et relevons les valeurs des angles pour chaque configurations. Séparer les phases nous permet d'avoir deux fonctions séparées et apporte plus de flexibilité (voir partie programmation) pour le contrôle du mouvement et les fonctions sont plus justes.

Une fois tout les angles relevés, pour la Phase 1, nous pouvons maintenant créer un tableau comme celui-ci :

m	X	-0,08	-0,06	-0,04	-0,02	0	0,02	0,04	0,06	0,08
degres	1	0,23	0,23	0,23	0,23	0,23	0,23	0,23	0,23	0,23
	2	61,57	58,49	54,65	50,13	45	39,35	33,30	26,91	20,24
	3	81,82	85,41	87,96	89,49	90	89,49	87,96	85,41	81,82

Comme présenté sur la figure 11, on voit que durant la phase 1, seule la valeur en x change, ce qui correspond à la première ligne sur le tableau, les 3 autres lignes sont les valeurs relevées des angles $\theta 1$, $\theta 2$ et $\theta 3$. L'objectif est de pouvoir créer une fonction à implémenter dans le programme comme par exemple $move(x)$ qui change les angles des moteurs pour que la jambe soit à l'abscisse x.

pour cela il faut donc avoir des fonctions continues sur (dans notre exemple) $[-0.08; 0.08]$ qui renvoient les angles voulus. A l'aide du tableur Excel (voir le tableur $traitement\_numerique$) nous trouvons les interpolations polynomiales d'ordre 2 qui passent par ces points et donnent ainsi les angles des moteurs en fonction des coordonnées du repère outil projeté dans le repère du bâtit.

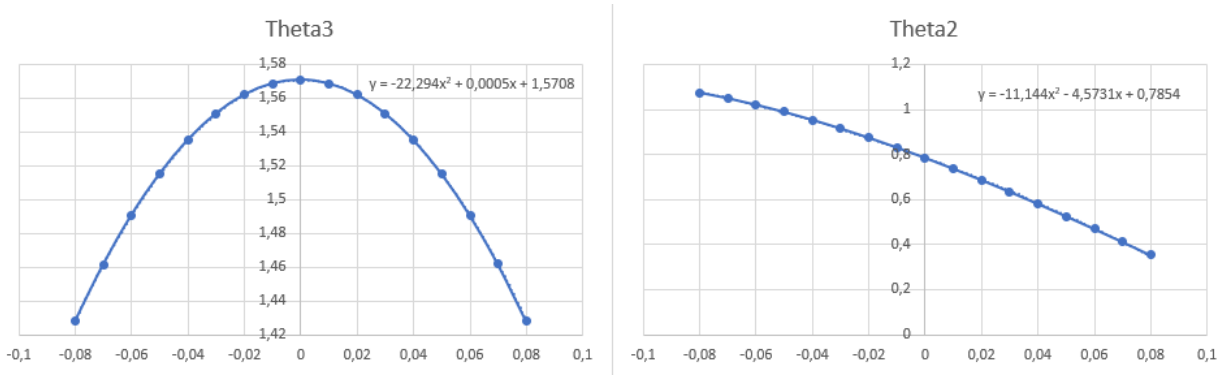


FIGURE 12 – Interpolation polynomiale des angles θ_1, θ_2 en fonction de l'abscisse du point D (x en m)

Ces graphes montrent les fonctions $Theta2(x)$ et $Theta3(x)$ qui renvoient $\theta 2$ et $\theta 3$ en fonction de x. Ce sont ces fonctions que nous utiliserons dans le programme.

Une fois ces dernières obtenues, il est alors très simple de positionner les repères outils de chaque patte du robot. Le problème cependant ici est que nous devons relever $3 * nb\_de\_points$ à chaque changement de trajectoire et tout ceci à la main ce qui reste peu pratique. D'autre part, les deux fonctions que nous obtenons ne sont valables que le long de trajectoires définies. Pour pouvoir s'étendre à la 3D il faudrais pouvoir interpoler une fonction de 3 variable.

3.2 Avec robotics toolbox

Cette partie à été réalisable suite aux TP de robotique de la période février-mars. Nous avons appris grâce au module *RoboticsToolbox* de Peter Corke à nous servir du tableau de Denavit-Hartenberg afin de piloter un bras robotique puis nous avons alors pu positionner quatre bras par rapport à un châssis afin de simuler au mieux notre robot tétrapode. A partir de cette base nous avons alors pu générer puis simuler des patterns de trajectoires.

3.2.1 Modélisation d'une patte et de ses trajectoires

A. Définition de la structure

On a, en noir le bâtit et à l'opposé en vert le repère outil (ici le bout de la patte qui vas toucher le sol). On place donc les repères d'articulation comme ceci :

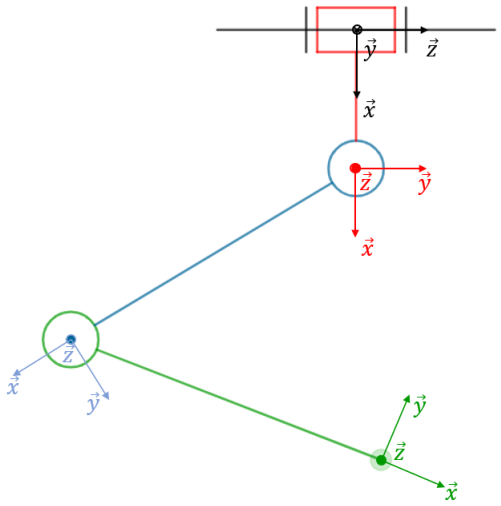


FIGURE 13 – Schéma cinématique de la jambe et des repères

Une fois les repères placés, nous pouvons alors implémenter les dimensions de notre patte dans le tableau de $Denavit - Hartenberg$.

B. Modélisation de la patte

La prochaine étape est de réaliser la matrice de passage du repère monde (noir) au repère objet (vert). Pour cela on utilise la convention de Denavit-Hartenberg grâce aux fonction de RVCtoolbox suivantes :

Listing 1 – Création du tableau de Denavit-Hartenberg

```
%lignes relatives aux differentes articulations dans le tableau de Denavit-
    Hartenberg.
L(1)=Link([t1,0,L1,pi/2]);
L(2)=Link([t2,0,L2,0]);    %articulations
L(3)=Link([t3,0,L2,0]);

% affiche toutes les donnees relatives de la 'Jambe' que nous venons de creer.
Jambe=SerialLink(L,'name','Jambe')
```

Une fois la dernière fonction lancée, Matlab nous renvoie un affichage du tableau de Denavit-Hartenberg ainsi que quelques informations supplémentaires comme la matrice de la base, gravité etc.
Afin d'obtenir un affichage plus convenable, nous choisissons de jouer sur des paramètres de la jambe tel que la base, et quelques options d'affichage comme le niveau du sol. L'affichage de notre 'Jambe' se fait ensuite suivant cette fonction : `Jambe.plot([`$\alpha_1, \alpha_2, \alpha_3$`])`. On obtiens alors :

Listing 2 – Création du tableau de Denavit-Hartenberg

```
% On fait ici tourner le robot de pi/2 selon y et le 12*sqrt(2)+5 correspond
    la hauteur voulue depuis le sol de la premi re articulation.
Jambe.base = [0  0  1       0      ;
              0  1  0       0      ;
              -1 0  0  12*sqrt(2)+5;
              0  0  0       1      ]

% 'floorlevel', 0 place le sol de l'affichage a 0 et 'nobase' retire la barre
    noire qui relie le sol a la premiere articulation du robot. Les valeurs 0,-pi
    /4 et pi/2 sont des offset donnees aux angles pour l'exemple.
Jambe.plot([0,-pi/4,pi/2],'floorlevel',0,'nobase')
```

Tout cela nous permet alors de simuler la patte du robot ci-dessous :

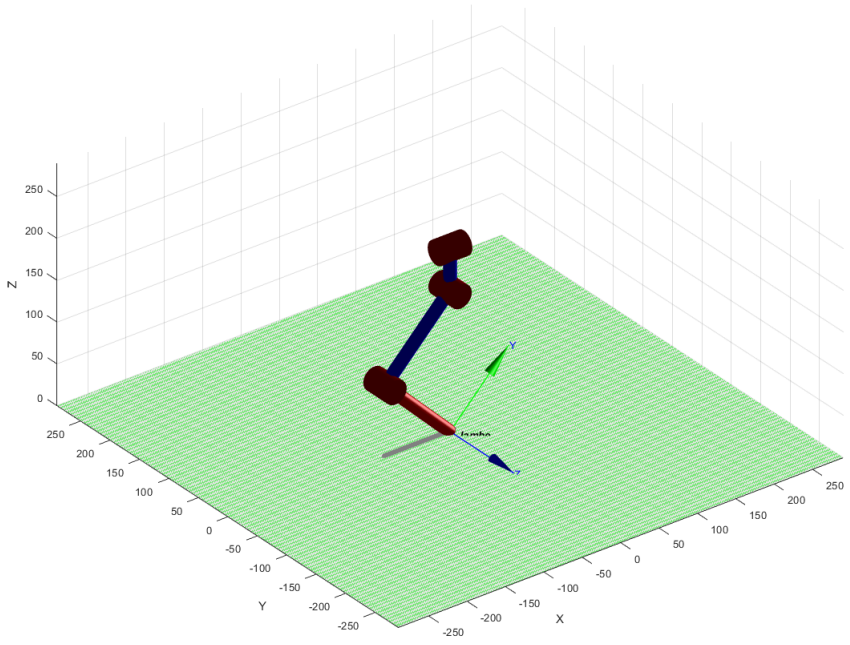


FIGURE 14 – Affichage de la jambe avec offset

A partir de cette simulation, nous allons pouvoir observer la réponses de la patte aux différentes trajectoires imaginées.

C. Trajectoires

L'objectif de cette partie vas être de créer une trajectoire du bout de la jambe du robot qui suit ce type de contour :

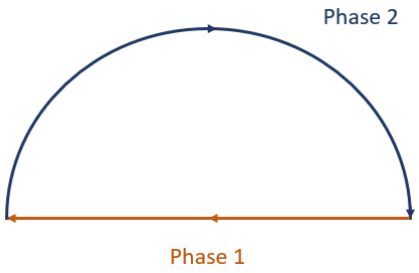


FIGURE 15 – Trajectoire désirée de la jambe

Pendant la phase 1, le bout de la patte recule ce qui reviens à pousser le robot vers l'avant. Dans la phase n°2, la patte se soulève du sol pour aller se poser plus loin et recommencer le cycle.

Tout d'abord, nous pouvons choisir de commander les mouvements articulations par articulations. c'est à dire que nous allons directement rentrer les valeurs des angles que nous voulons pour chaque moteurs. L'inconvénient de cette pratique est cependant qu'il faut se débrouiller soit même pour faire que la patte s'arrête à hauteur du sol. Pour créer une trajectoire de ce type, nous pouvons procéder comme ceci avec la fonction $jtraj()$.

Listing 3 – Définition de notre trajectoire en espace articulaire

```
%% -Creation de trajectoire-
%---En Espace articulaire---
M=jtraj([-pi/12 -pi/4 pi/2],[pi/12 -pi/6 pi/1.5],[0:99]/99);
```

Nous décidons cependant par la suite de passer en coordonnées cartésiennes dans un soucis de simplicité.
Tout d'abord, nous allons créer les 3 vecteurs $p1, p2, p3$ par lesquels nous voulons faire passer le bout de notre 'Jambe'.
Ensuite, grâce à la fonction $mstraj()$ nous créons cette trajectoire. Nous coupons ensuite les extrémités de cette courbe pour avoir une trajectoire cyclique et arrondie. Enfin, avec la méthode $ikine()$ nous obtenons les valeurs d'angle de chaque moteurs.

Listing 4 – Définition de notre trajectoire en espace cartésien

```
%% -Creation de trajectoire-
%---En Espace cart sien---
p1 = [-6 -2 0] ;
p2 = [0 -1  8];
p3 = [6 -2 0];

%creation de la trajectoire passant par les points 1,2 et 3
Mc2 = mstraj([p2;p3;p1;p2;p3],[],[10,10,20,10,10],p1,0.5,8);
```

```
9
10 Mc2 = Mc2(20:100,:);
11 Mc2 = transl(Mc2);
12
13 M=[1 1 1 0 0 0]; %permet de laisser les degres de liberte
14 Q = Jambe.ikine(Mc2,[0,-pi/4,pi/2],M); %calcul des valeurs des angles a partir de
       la trajectoire de l'effecteur
```

3.2.2 Génération d'un cycle de marche complet pour le robot

L'objectif ici vas être de coordonner plusieurs pattes en même temps pour avoir une simulation correcte du robot. Nous nous interrogerons alors à la fin sur la stabilité de celui-ci

A. Définition du cycle de marche

La trajectoire créée précédemment est la suivante :

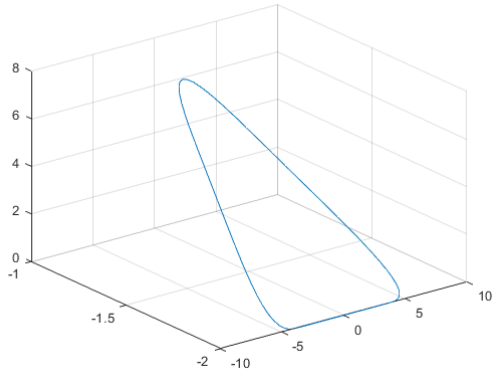


FIGURE 16 – Trajectoire d'une jambe

Sur ce graphe, nous voyons que la trajectoire correspond à ce que nous voulions avec aucun déplacement en dessous du niveau du sol.
Nous implémentons alors 4 jambes en donnant des trajectoires décalé d'un demi cycle par rapport aux deux autres pour les jambes avant gauche et arrière droite grâce à la fonction *gait*(). Cette dernière nous permettra aussi de faire la symétrie de l'angle du moteur au niveau du châssis pour les pattes de gauche.
Le code ci dessous nous permet ainsi l'affichage complet du robot en mouvement.

Listing 5 – Affichage du Robot Complet

```
1 figure(1);
2 patch(X,Y,Z,'b')% Affiche le chassis
3 for i=1:80
4     Q1 = gait(Q,i,0,0);
5     Jambe1.plot(Q1,'floorlevel',0,'nobase','fps',120,'trail','k');hold on
6     Q2=gait(Q,i,40,1);
7     Jambe2.plot(Q2,'floorlevel',0,'nobase','fps',120,'trail','k');hold on
```

```
    Q3 = gait(Q,i,0,1);
    Jambe3.plot(Q3,'floorlevel',0,'nobase','fps',120,'trail','k');hold on
    Q4=gait(Q,i,40,0);
    Jambe4.plot(Q4,'floorlevel',0,'nobase','fps',120,'trail','k');hold on
end
```

Nous obtenons ainsi une marche en 2 temps avec toujours au moins deux jambes sur le sol comme nous pouvons le voir sur la figure ci-dessous.

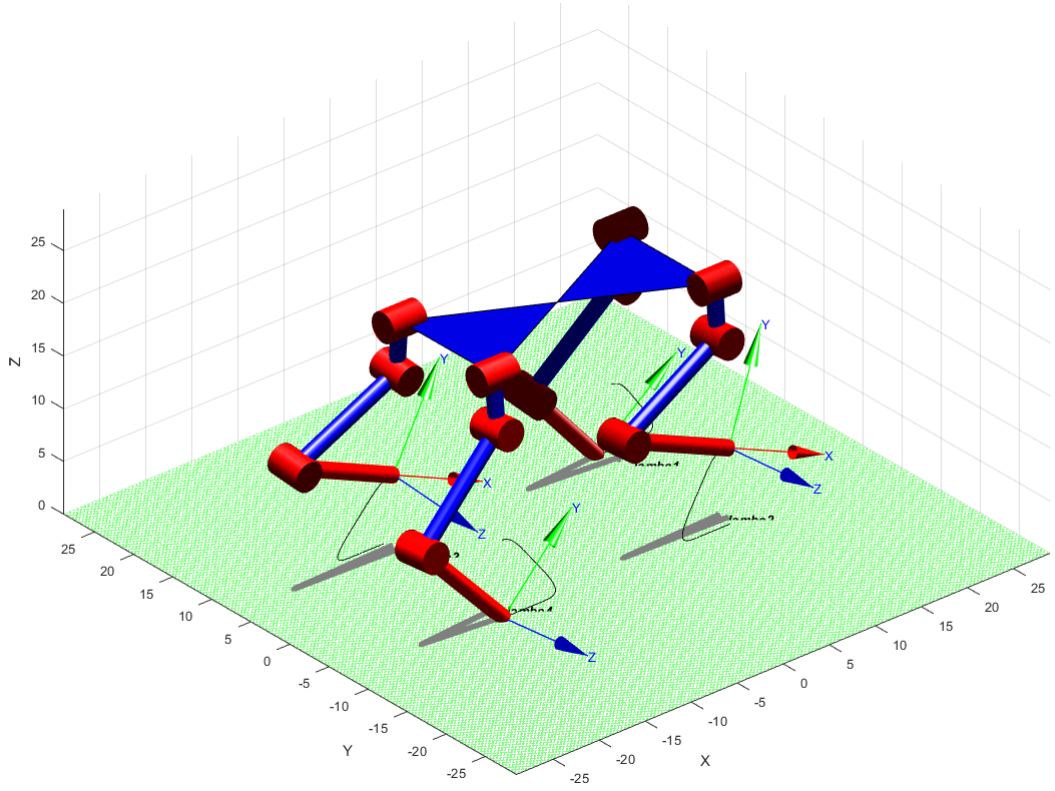


FIGURE 17 – Cycle de marche

B. Simulation et analyse de la stabilité

Suite à la partie précédente, nous obtenons une marche satisfaisante du robot. Seule la vitesse d'affichage est a critiquer. On modifie donc le processus comme ci-dessous :

Listing 6 – Affichage du Robot Complet

```
%init
figure (1);

Q1 = gait(Q,1,0,0);
Jambe1.plot(Q1,'floorlevel',0,'nobase','fps',120,'trail','k','noname');hold on;
Q2=gait(Q,1,25,1);
Jambe2.plot(Q2,'floorlevel',0,'nobase','fps',120,'trail','k','noname');hold on;
Q3 = gait(Q,1,50,0);
Jambe3.plot(Q3,'floorlevel',0,'nobase','fps',120,'trail','k','noname');hold on;
Q4 = gait(Q,1,75,1);
Jambe4.plot(Q4,'floorlevel',0,'nobase','fps',120,'trail','k','noname');hold on;
```

```
12
13 %mouvement
14 for i=1:300
15     Q1 = gait(Q,i,0,1);
16     Jambe1.animate(Q1);hold on %animate permet d'accelerer l'animation du robot
17     Q2=gait(Q,i,50,1);
18     Jambe2.animate(Q2);hold on
19     Q3 = gait(Q,i,25,1);
20     Jambe3.animate(Q3);hold on
21     Q4=gait(Q,i,75,1);
22     Jambe4.animate(Q4);hold on
23 end
24 %cet affichage correspond a la marche en 4 temps qui est expliquee par la suite
```

Pour étudier la stabilité du robot, nous utilisons un affichage supplémentaire. C'est dans le script principal, la partie intitulée : `affichage des hauteurs de patte`. Le but est ici d'afficher la hauteur z de chaque patte en parallèle. Pour notre robot, le résultat est le suivant :

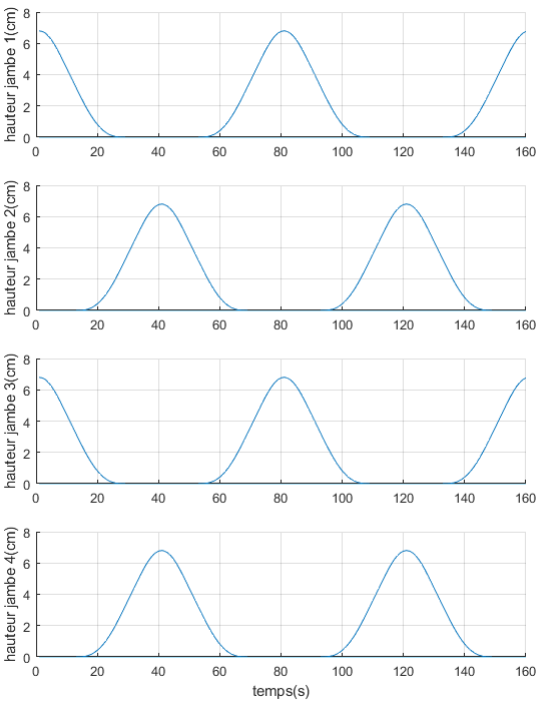


FIGURE 18 – hauteur des différentes pattes pour la marche en 2 temps

On constate ci-dessus que seules 2 jambes sont en contact du sol à chaque instant. La stabilité n'est donc pas tout a fait assurée et, afin de l'améliorer nous avons changé la séquence de marche pour que le robot puisse marcher avec en permanence 3 pattes sur le sol. Pour cela on augmente le nombre de points pour la phase 1 de la marche, et on change les valeurs de déphasage pour les `gait()` afin les placer à 1/4, 1/2 et 3/4 de la séquence.

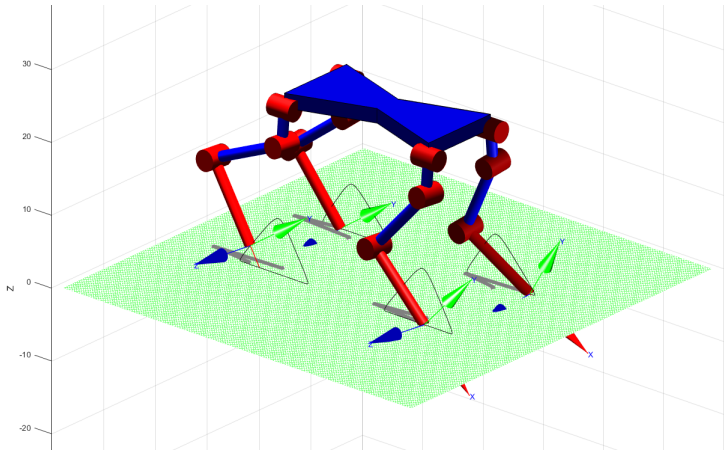


FIGURE 19 – Robot final modélisé

Avec tout ces changements on obtiens un robot qui soulève ses pattes les unes après les autres et nous obtenons donc les courbes suivantes :

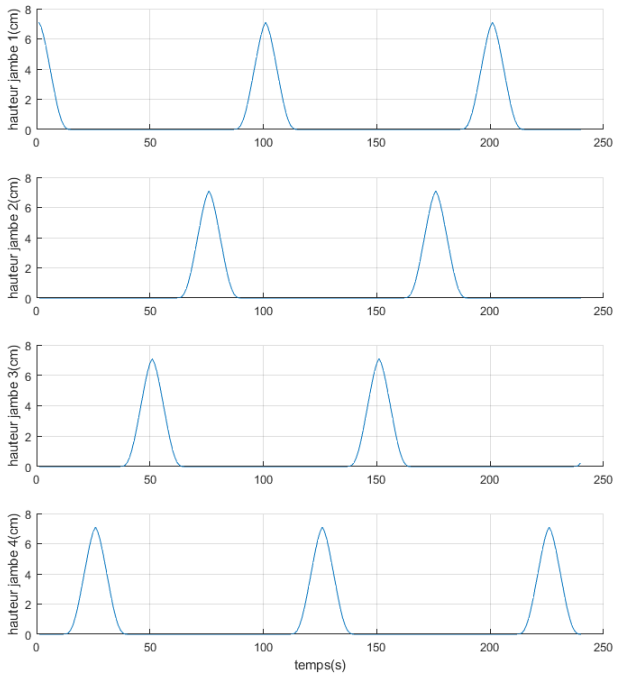


FIGURE 20 – hauteur des différentes pattes pour la marche en 4 temps

On observe ainsi grâce à ce graphique que nous n'avons désormais jamais plus d'une patte est en l'aire tout au long du pattern. Cela devrait en théorie augmenter la stabilité du robot.

En implémentant les différents types de marche créés ici, directement dans notre robot on observera que les mouvements obtenus sont exactement ceux demandé. Cependant pour que cela soit réellement satisfaisant dans la pratique, il nous aurait fallu rajouter un environnement physique aux simulations. En effet, étant donné que nous sommes en boucles ouverte ici, plus la simulation sera proche de la réalité plus nous serons aptes à avoir dans la pratique un robot se comportant comme nous le désirons. La meilleur solution restant toujours d'utiliser une boucle fermée mais cela impliquerai d'utiliser des capteurs que nous n'avons pas tout en implémentant une régulation somme toute assez complexe.

3.3 Comparaison des deux méthodes

A priori, aucune des deux solution proposée ne semble se démarquer positivement de l'autre, tout du moins, pour le type d'usage que nous en avons aujourd'hui. Les deux méthodes ont en effet chacune leurs points forts et leurs points faibles.
La première méthode utilisée est intéressante en ce qu'elle nous permet une liberté total autour du point de fonctionnement choisit et donc elle serait très utile si le robot devait par lui même définir ses mouvements dans le repère du bâtit. Cependant, elle nous demande aussi de définir à la main chaque point par lequel nous voulons que le repère outil passe, ce qui peut être coûteux en temps et ne pas forcement mener aux bons choix.
La seconde méthode utilisée est avant tout pratique car tout d'abord, nous avons une meilleur visibilité de ce qui sera fait à l'aide des simulations Matlab mais aussi car il nous suffit de définir seulement quelques points de passage et un type de trajectoire pour changer totalement le pattern de marche. De plus le principe de la marche "step by step" nous permet en posant des laps de temps constants entre chaque étape, d'obtenir une marche fluide que l'on peut accélérer ou décélérer très simplement. Cependant ses majeurs problèmes sont d'une part la place importante qu'elle peut prendre dans la mémoire et le fait qu'une fois définit, on ne peut plus revenir dessus. Le problème étant ainsi que, si nous voulions réguler la marche du robot avec une boucle fermé pour plus de stabilité, ce type de solution ne nous permettrai pas de faire varier le pattern de marche.

4 Programmation

Dans ce chapitre, nous allons tout d'abord comprendre le fonctionnement des moteurs puis nous verrons ensuite comment nous avons conçu la mise en route du robot et comment nous avons implémenté les patterns de marche du robot selon les différentes solution apporté dans le chapitre 3.
Dans cette partie, il sera, encore une fois, seulement question d'automatisation. Le programme n'ayant aucun retour extérieur, il sera défini et figé avant la mise en mouvement du robot.

4.1 Implémentions de base d'un moteur

Dans cette sous-partie, nous allons tenter de résumer le plus simplement possible les grandes lignes de la documentation des servo-moteurs dynamixels qui nous à été fournie en début d'année.

4.1.1 Commande du moteur

Pour commencer, la documentation $AX-12$ *User Guide* nous offre le tableau des variables internes des servo-moteurs dont voici un extrait ci-dessous :

Area	Address (Hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Lowest byte of model number	R	12 (0X0C)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	1 (0X01)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	70 (0X46)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)

FIGURE 21 – Tableau des variables internes d'un moteur Dynamixel

Ce tableau nous permet alors de connaître l'adresse de chaque variable ou constante encrée dans le moteur. Certaines ne peuvent pas être modifiées(Access : R pour Read) mais la plus part le peuvent(Access RW pour Read Write). Enfin, certaines données nécessite deux adresse. En effet chaque adresse contient 8 bits mais certaines variables en on besoin de 16. C'est le cas par exemple pour la vitesse du moteur qui contient une adresse "Mooving Speed(L)" qui correspond au 8 bits de poids faible (i.e. de 0 à 255) et une autre adresse "Mooving Speed(H)" qui correspond aux bits de poids forts qui permettent d'atteindre les valeurs de 0 à 65532. Pour le coté pratique cependant il est possible de donner directement à l'adresse (L) une valeur appartenant à $[0\ ,\ 2^{16}-1]$ et les valeurs des deux adresses s'en trouveront alors modifié comme il le faut.
Il est nécessaire cependant de prendre conscience qu'une fois que les variables ayant une adresse comprise entre 0 et 18 sont modifié, elle le sont jusqu'à la prochaine modification car les moteurs possèdent leur propre mémoire morte.

4.1.2 Initialisation et mise en mouvement du moteur

Pour mettre le moteur en mouvement, il faut tout d'abord comprendre comment lui donner des ordres puis ensuite, modifier les variables nécessaire pour qu'il soit utilisable dans la configuration que nous désirons.
Ainsi, les trois fonctions qui suivent sont sûrement les plus importantes :

```
int readByte(int ID, int adress);
void writeByte(int ID, int adress, int value);
void writeWord(int ID, int adress, int value);
```

Comme cela est pré-supposable, la fonction *readByte*() permet de lire la valeur de la variable à l'adresse désirée. Si l'identifiant du moteur n'est pas changé, sa valeur par défaut sera 1. Il sera cependant nécessaire de la changer par la suite car chaque moteur utilisé doit avoir son propre identifiant pour pouvoir être différencié des autres. En effet, ces derniers pouvant être branché en série, nous ne pourrons pas les différencier par les pins à partir desquels ils sont connectés sur la carte de contrôle.
Les fonctions *writeByte*() et *writeWord*() quant à elle, permettent de changer la valeur des variables internes d'un moteur à une adresse donnée. Leur différence vient du fait que la première peut seulement envoyer une variable d'un octet tandis que la deuxième peut envoyer une variable de deux octet ce qui permet de changer les adresses (L) et (H) d'une même variable en une fois comme vu dans la section précédente avec l'exemple de la variable "Mooving Speed".

Une fois cela en tête, nous pouvons alors modifier les variables de la mémoire morte de chaque moteur afin que ces derniers soient dans la configuration qui nous convient au mieux.
Tout d'abord nous avons changés L'ID de chaque moteur (Adresse N°3) pour qu'aucun d'entre eux n'ai la même valeur. Ensuite, nous sommes passés en *Joint mode* à l'instar du *Wheel mode*. Pour cela nous avons du donner des limites d'angles aux moteurs à l'aide des adresses comprises entre 6 et 9. Nous avons fait en sorte de fixer ces limites de façon à ce que les moteurs ne se retrouvent jamais à forcer sur le bâtit. Les valeurs données sont explicités sur le schéma ci-dessous :

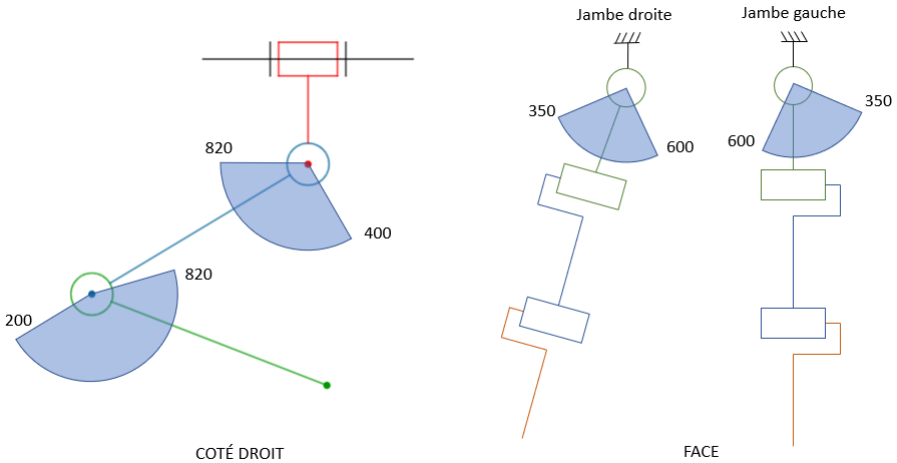


FIGURE 22 – Valeurs limites imposés aux différents moteurs

Les valeurs limites étant initialement 0 et 1023(en théorie, car en pratique tant que nous sommes en *Wheel mode*, il n'y a pas de limites), nous les avons ainsi modifié comme proposé sur le schéma. Ce graphique permet donc d'avoir une idée plus précise de l'amplitude possible des mouvements du robots.

4.2 Implémentation des Patterns de marche

Dans cette section, nous allons nous concentrer sur l'implantation de la marche suivant les deux solutions apportés dans la partie 3.2.
Tout d'abord, cependant, nous allons nous attarder un peut sur le process prenant place avant le début de la marche. En effet, à la mise sous tension du robot, la fonction *setup*() se lance et, avec elle, un ensemble de trois fonction permettant au Tétrapode de se mettre en position initiale de marche :

- *asleep*() est la première à être lancée ; elle permet de mettre le robot en position couché, c'est à dire avec ses quatre pattes repliées sur elles mêmes. Cette dernière à été conçu pour arriver dans cette état final quelque soit la position initial du robot. Ce n'est pas toujours le cas cependant, par exemple si le robot part dans des conditions initiales vraiment trop complexes, mais cela fonctionne dans la plus grande majorité des cas.

- $upBeforeWalk()$ vient en seconde position et permet au robot de se lever sur ses quatre pattes en alignant les centres outils de ces dernières avec leur épaule respective.

- enfin $initWalk()$ positionne les pattes du robot de façon à ce que ces dernières soient en position pour la première étape de marche du robot. Même si elle peut sembler futile, cette fonction est nécessaire car sans elle le robot entamerai sa marche en déplaçant trop vite ces quatre pattes pour les positionner comme demandé et cela engendrerai alors une instabilité conséquente.

4.2.1 Pattern de marche à partir de Géogébra

La programmation de la marche à partir des interpolations estimé à l'aide de *Gogebra* nous laissant totalement libre de faire ce que nous voulons, cela nous rend finalement l'implémentation assez hasardeuse.Au cours de ce semestre, nous avons créés de nombreux pattern de marche en suivant cette solution. Nous allons vous présenter celui qui nous semble aujourd'hui le plus satisfaisant. Il s'agit d'une marche symétrique en 4 temps dont voici une représentation ci-dessous :

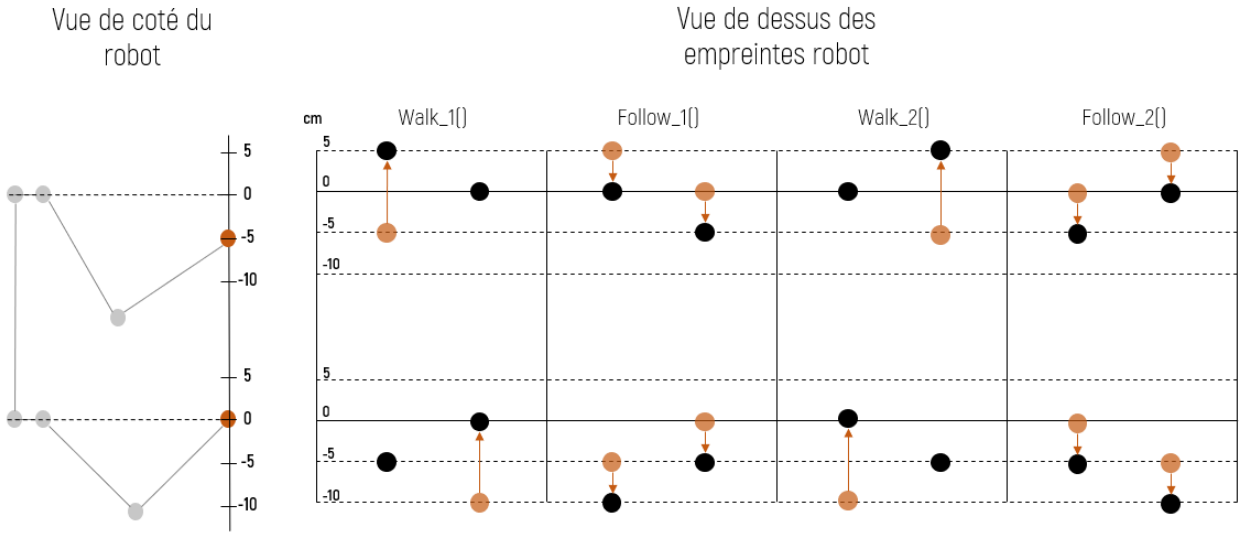


FIGURE 23 – Pattern de marche vu au travers des empreintes du robot sur le sol

Comme nous pouvons le voir sur ce schéma, la marche est symétrique en ce que $walk\_1()$ et $follow\_1()$ sont les symétriques de $walk\_2()$ et $follow\_2()$.

Nous ne rentrerons pas ici dans les détails de la programmation car les fonctions sont explicitées dans le programme $Demo\_MarcheLente\_MarcheRapide$ cependant, nous nous efforcerons d'énoncer au mieux les différents problèmes que nous avons rencontré lors de son écriture.
Le problème majeur étant l'instabilité créée par la marche. En effet, n'ayant aucun moyen de réguler l'équilibre du robot, nous avons du tenter d'équilibrer au mieux la démarche. Ainsi, seulement les pattes opposés par rapport au centre de gravité du robot se lèvent en même temps mais cela ne l'empêche cependant pas de pencher d'un coté ou de l'autre. Nous avons donc du augmenter la vitesse d'action sur ces étapes (ce sont les fonctions $walk\_i()$). Cependant en l'augmentant trop, le robot finis par taper dans le sol du coté ou il penche de façon trop violente ce qui le fait glisser et finis même par ne plus le faire avancer du tout. (Il est intéressant de noter qu'avec de très mauvais paramètres, nous avons même réussit à faire reculer le robot avec cette même marche). Ensuite, dans les étapes deux et quatre quand le robot avance son corps principal en jouant sur la friction des pattes par rapport au sol(fonctions $follow\_i()$), la vitesse aussi est un enjeux important car, si ce dernier est trop rapide, il engendre un moment d'inertie qui finira par le faire basculer vers l'avant (expliquant en partie pourquoi nous avons choisis de diminuer la hauteur des pattes). Enfin, vient le problème du nombre d'étape pour la réalisation de chaque phases (cf figure 2). Si nous avons trop de pas, cela ralenti le process et peut déstabiliser le robot quand il est dans la phase 1 (i.e. fonctions $walk\_i()$) et peut aussi amener des mouvements saccadé. Seulement, si nous n'avons pas assez d'étapes, le mouvement se désagrège et, surtout dans la phase 2(fonctions $follow\_i()$), cela peut mener à des variation de hauteur non négligeable.
Ainsi, en suivant cette logique, nous avons décidé de faire procéder les fonctions $walk()$ en 2 étapes et les fonctions $follow()$ en 10 étapes ; le tout en réglant au final empiriquement les vitesse des moteurs pour chacune des fonctions.

La fonction dans laquelle nous avons implémenté ce processus de marche s'appelle $walk\_follow()$ et est implémenté dans le programme $Demo\_MarcheLente\_MarcheRapide$

4.2.2 Pattern de marche à partir de Robotics Tools

Dans cette partie, le travail à réaliser à été beaucoup plus simple que dans la partie précédente. Avec du recul, il nous semble même évident que nous aurions du commencer avec ce type de simulation.
La simplicité d'implantation de cette algorithme viens du fait que nous pouvons d'abord tester le robot en simulation puis analyser son comportement avant de l'éprouver dans la réalité. Les problème soulevé en fin de partie 3.2.2 sur le manque de réalisme physique reste cependant toujours d'actualité.
La synchronisation des pattes entre elles étant résolu en simulation, nous pouvons alors passer sur une marche plus dynamique ou les étapes 1 et 2 puis 3 et 4 de la partie précédentes sont confondu. Ainsi, dans la marche que nous avons retenu pour cette partie, le robot à seulement deux étapes symétriques comme montré ci-dessous :

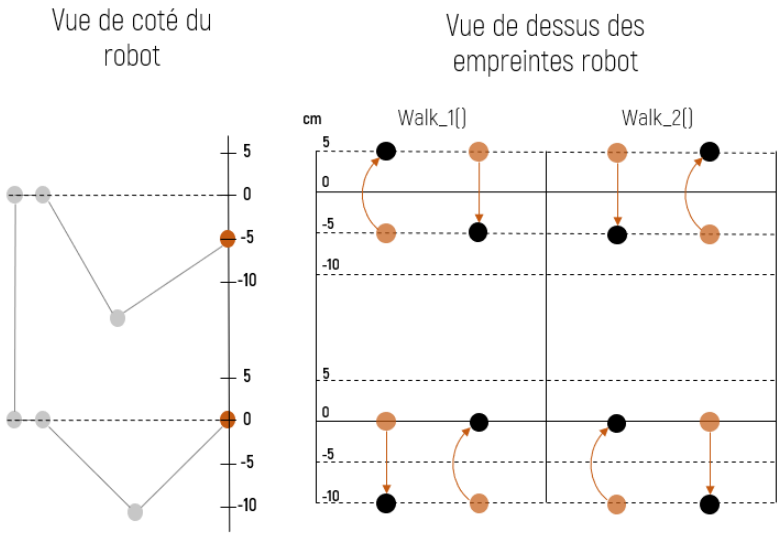


FIGURE 24 – Pattern de marche vu au travers des empreintes du robot sur le sol

Ici, le repère indiqué en cm est la différence selon la coordonnée X du repère outil par rapport au repère de l'épaule pour chaque bras.
Nous avions tenté d'implémenter cette marche avec *Gogebra* mais cela avait été un échec à cause du ralentissement du robot créé par la synchronisation des phases que nous tentions d'avoir à la main entre les 4 pattes.
Ainsi, il ne nous reste plus désormais que deux variables à modifier que sont le délai entre chaque étape et la vitesse des moteurs. Étant donné que cette marche est fortement instable du fait de sa plus grande complexité, nous avons du fixer haute la vitesse des moteurs et bas le délai entre chaque étape.
Ainsi le robot avance assez vite pour contrer la gravité qui à la fâcheuse tendance de le déstabiliser.

La fonction dans laquelle nous avons implémenté ce processus de marche s'appelle $matlabWalk()$ et est implémenté dans le programme $Demo\_MarcheLente\_MarcheRapide$.

Ainsi, comme proposé précédemment, il aurait été préférable de commencer par l'implémentation de la marche suivant la solution apporté par *Robotics Tools* car cette dernière nous permet une meilleur compréhension global du fonctionnement de la marche et nous rend les choses plus simples pour l'implantation de cette dernière dans le programme. De plus le faite de par la suite, nous repencher sur de l'interpolation pour estimer les fonctions des angles directement à partir des coordonnées du repère outil nous apporte un regard nouveau qui pourrait in fine nous permettre de nous lancer sur le chemin de la régulation et ainsi, plus qu'un simple automate, créer un vrai robot mécatronique.

5 Conclusion

5.1 Conception actuelle et future

En cette fin d'année, nous nous retrouvons donc avec un robot tout à fait opérationnel. des sacrifices on cependant du être fait sur le design de certaines pièces pour plutôt se concentrer sur l'obtention d'une marche plus convaincante.
Concernant la modélisation et la conception, comme vu dans la partie 3, le corps est à refaire en priorité. Par rapport aux modifications potentielles à effectuer, la pièce qui joue le rôle de l'épaule, pourrait être agencée différemment pour offrir plus de liberté et de simplicité vis a vis des fixations. Cette solution n'a pas été envisagée car l'idée est apparue trop tard et demandait de revoir nos équations et programmes de marche.
Ceci dis, avec le robot dans l'état actuel, nous avons pu mener beaucoup de tests, avec différents programmes et différents paramètres. Au final, le robot reste un automate qui avance grâce à un cycle prédéfini. Une méthode, celle de *Robotics Toolbox*, s'est avérée plus simple. A l'inverse, la méthode venant à créer des fonctions est plus adaptée à une régulation car la transformation trajectoire -> suite d'angles est faite à l'intérieur même du robot.
Le problème majeur auquel nous nous sommes heurtés est l'inertie de notre robot. En boucle ouverte, gérer l'inertie du robot du à l'accélération de chacune de ses pattes est très complexe. Nous avons pu minimiser ce problème avec différents paramètres et pratique de programmation mais cela reste quand même une limite à laquelle il faut faire attention.

5.2 Des idées non abouties

De l'idée à l'aboutissement, le chemin est parfois long. Voici des exemples d'idées que nous aurions aimé mettre en oeuvre mais qui ne se sont pas réalisées pour une raison ou une autre.

Nous avons tout d'abord durant ce projet cherché à implémenter un capteur infrarouge afin de commander le robot à distance à l'aide d'une petite télécommande. Cependant, la librairie nécessaire à cette réalisation n'était pas implémentable sur *OpenCM* contrairement à *Arduino* et même en tentant de modifier cette dernière, nous n'avons pas réussit à la faire s'exécuter. Nous avons bien entendu trouvé d'autre librairie en ligne mais aucune d'entre elles ne correspondaient au type de télécommande que nous possédions.
En fin d'année, nous avons aussi implémenté un capteur ultrason sur le devant du robot. Le but recherché était de s'en servir pour contourner les gros obstacles cependant ce dernier s'est trouvé très peut fiable. Même en faisant une moyenne des dernières valeurs de distance mesurées, le résultat était bien trop souvent aberrant pour¨être pris en compte. Peut être serait il intéressant d'ajouter un capteur lidar sur le dos du robot pour avoir une vision à 360°et ainsi faire de la détection d'obstacle ou de la régulation de trajectoire.
Une autre idée que nous aurions adorer essayer de mettre en oeuvre aurait été de faire de la reconnaissance d'image. Cela aurait alors permis au robot d'atteindre des cibles ou de suivre des personnes par exemple.
Enfin, tout au long de l'année, nous avons appris à créer en simulation différents types d'Intelligence Artificielle comme la $Neuro-volution$(mélange de programmation génétique et de réseau neuronaux) ou le $Deep\ Reinforcement\ Learning$(outil utilisé notamment par OpenAI dans la création de l'IA excellant à DOTA 2). Cela aurait pu permettre à notre robot d'apprendre à se lever tout seul voir à marcher en ligne droite. cette IA nous aurait donné la possibilité d'atteindre une sorte de régulation en rajoutant seulement quelques capteurs de pression et un gyroscope par exemple. Malheureusement le timing étant trop juste nous avons laissé cette idée de coté pour un prochain projet.

5.3 Ce que ce projet nous a apporté

Pour conclure, grâce à ce projet, nous avons pu largement étendre notre compréhension et notre pratique de la conception, modélisation et de l'utilisation de divers logiciels en créant un objet qui regroupe un certains nombre de nos connaissances mécatronique. Il est intéressant en outre de constater que, par exemple, la modélisation de la marche que nous pensions achevable en quelques semaines nous à finalement pris de long mois et n'est pas, encore aujourd'hui, au niveau que nous aurions espérés atteindre.
Pour finir, ce projet nous a délivré un avant goût assez concret de ce que peuvent être la conception et la modélisation dans le milieu de l'ingénierie et cela nous aura donné l'occasion à tout les deux d'avoir une perception plus fine de nos différents centres d'intérêts.

6 Synopsis

6.1 Français

Dans ce projet nous allons concevoir et réaliser un robot quadripode capable de marcher en ligne droite. Ce robot sera constitué d'un bâtit d'une trentaine de cm d'envergure et de quatre bras de 30 cm aussi environ contenant chacun trois servomoteurs. Pour cela nous allons concevoir une modélisation de chaque pièce nécessaire à l'aide du logiciel *Catia* que nous allons par la suite imprimer en 3D. Une fois ces pièces assemblés aux 12 moteurs dynamixel, nous programmerons le tout à l'aide d'une carte OpenCM équivalente à une carte Arduino. La programmation de la marche se fera alors en suivant deux solution différentes. La première consistera en l'interpolation polynomiale des angles des moteurs en fonction de la position des repères outils (en bout de bras). La deuxième passera par la création d'une simulation du robot à l'aide du module *Robotics Toolbox* sur Matlab.

Mots clés : Conception, Modélisation, Simulation, Robotique, Automatisme.

6.2 English

In this project we will design and build a quadripod robot able of walking in a straight line. This robot will consist of a 30 cm long body and four 30 cm arms containing three servomotors each. To do this, we will design a model of each necessary part using the *Catia* software that we will then print in 3D. Once these parts are assembled to the 12 dynamixel motors, we will do the programation using an OpenCM card which is similar to an Arduino card. The programming of the operation will be done by following two different solutions. The first one will consist of a polynomial interpolation of the motor angles according to the position of the tool marks (at the end of the arm). The second will involve to create a simulation of the robot using the *Robotics Toolbox* module on Matlab.

Key words : Design, Modeling, Simulation, Robotics, Automation.