UNIVERSITÉ DE TOULON

MASTER INGÉNIERIE DE SYSTÈMES COMPLEXES
PARCOURS ROBOTIQUE ET OBJETS CONNECTÉS

Projet tutoré
Démarche d'un robot quadrupède

Élèves :
Otmane ATTOU
Wajdi HELAWI
Avotra Ny Aina Mampionontsoa RAKOTONDRAVONY

Enseignant :
Nicolas BOIZOT

Table des matières

1 Introduction : 3

2 Modélisation robotique : 4
2.1 Présentation de la squelette du chat : . . . 4
2.2 Schéma cinématique simplifié de robot : . . . 4
2.2.1 Avec des noms de membres de squelette de chat : . . . 4
2.2.2 Avec des noms "humains" : . . . 5
2.3 Modèle simplifié de la patte d'avant : . . . 6
2.3.1 Définitions de repères : . . . 6
2.3.2 Équations cinématiques : . . . 6
2.3.3 Trajectoire souhaité de la patte d'avant : . . . 8
2.4 Modèle simplifié de la patte d'arrière : . . . 8
2.4.1 Définitions de repères : . . . 8
2.4.2 Équations cinématiques : . . . 9
2.4.3 Trajectoire souhaité de la patte d'avant : . . . 10
2.5 Type de marche de robot quadrupède : . . . 10
2.5.1 Vocabulaire : . . . 10
2.5.2 La marche ciblée dans le projet : . . . 11

3 Modélisation numérique : 12
3.1 Construction du modèle de base : . . . 12
3.1.1 code : . . . 12
3.1.2 Affichage : . . . 13
3.2 Simulation de la patte avant et arrière : . . . 14
3.2.1 Calcul de la position de la patte d'avant : . . . 14
3.2.2 Calcul de la position de la patte d'arrière : . . . 14
3.2.3 Simulation statique de la patte avant et arrière : . . . 15
3.2.4 Simulation de la trajectoire souhaité de la patte d'avant : . . . 16
3.2.5 Simulation de la trajectoire souhaité de la patte d'arrière : . . . 16
3.2.6 Affichage de cycloïde de deux pattes : . . . 17
3.2.7 Simulation animée de la patte avant et arrière : . . . 18

4 Expérimentation sur Arduino : 20
4.1 Principe : . . . 20
4.2 Code sur Arduino : . . . 21

5 Conclusion : 30

Table des figures

1 Robot quadrupède à LIS 3
2 Squelette d'un chat 4
3 Schéma cinématique du robot quadrupède 5
4 Schéma cinématique du robot quadrupède 5
5 Définitions de repères et d'angles 6
6 Définitions de repères et d'angles 8
7 Définitions de différentes positions de pattes 11
8 Chronogramme de marche de pattes 12
9 La représentation de la base de robot et la position initiale 14
10 Définitions de repères et d'angles 16
11 Les différents mouvement des pattes pour la marche de robot 20

1 Introduction :

Dans le cadre du projet de la première année de master ingénierie de systèmes complexes, à l'université de Toulon, encadré par Nicolas Boizot, nous avons travaillé sur le robot quadrupède.

Un robot quadrupède est un robot disposant quatre pattes lui permettant de marcher. Son intérêt principal est de pouvoir marcher sur des terrains irréguliers, voire accidentés. Cette promesse s'inscrit dans le cadre de faire sortir les robots des usines, où ils agissent dans un environnement parfaitement connu et peuvent représenter un danger pour les opérateurs humains, et de les amener dans d'autres cadres, dans lesquels le degré d'incertitude est plus élevé, afin de les faire travailler en collaboration avec les humains. On parle alors de cobotique.

Nous présentons le robot quadrupède, disposé dans le Laboratoire d'informatique et des Systèmes (LIS), conçu par Léo GUENIN étant un ancien étudiant à l'IUT GEII de Toulon, et retravaillé par Ornella Braun et Cyrille Gomez.

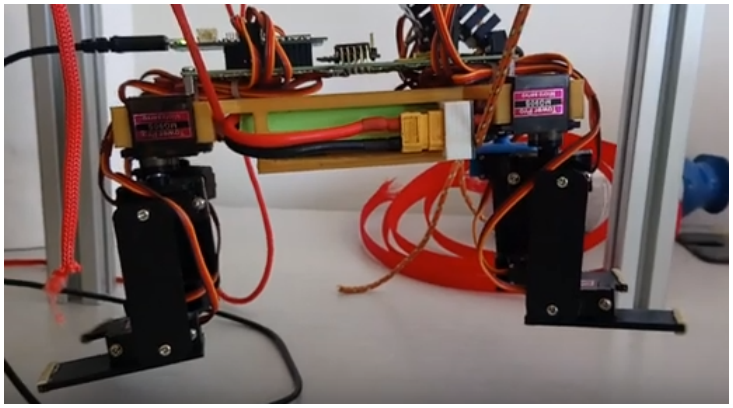


FIGURE 1 – Robot quadrupède à LIS

2 Modélisation robotique :

2.1 Présentation de la squelette du chat :

Afin de modéliser le robot, nous présentons la squelette d'un animal quadrupède, un chat.

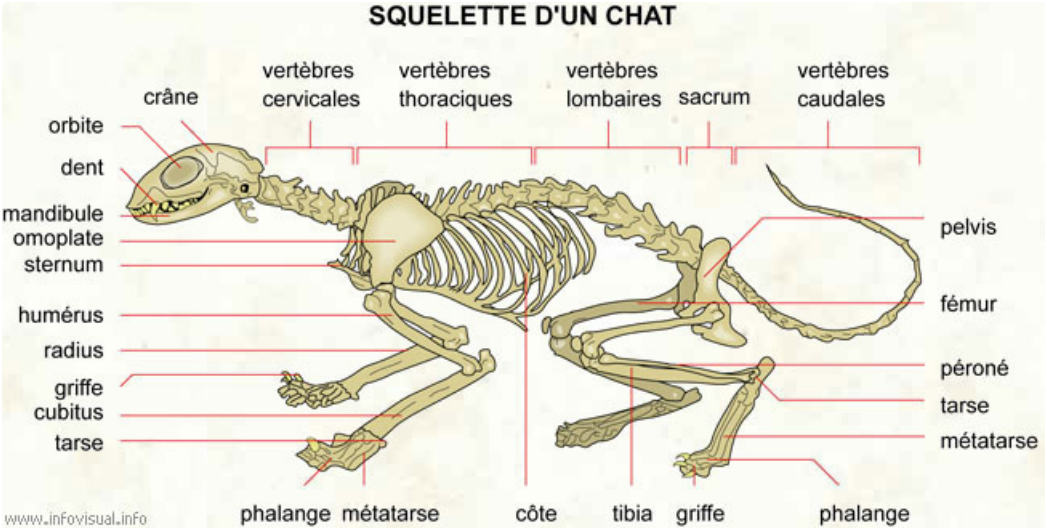


FIGURE 2 – Squelette d'un chat

2.2 Schéma cinématique simplifié de robot :

2.2.1 Avec des noms de membres de squelette de chat :

En s'inspirant de l'anatomie de chat, nous définissons le schéma cinématique du robot quadrupède (figure 2).

Les liaisons pivots 1, 2, 3, 4 permettent l'orientation du robot. Celles-ci ne seront pas être étudiées dans ce projet.

Toutes les liaisons pivots sont actionnées par Des servomoteurs MG90S.

Les coussinets sont censées d'avoir un facteur de frottement élevé afin de pouvoir assurer le contact entre le robot et le le sol.

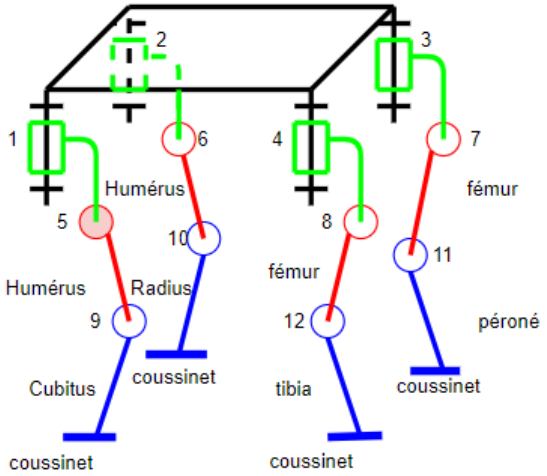


FIGURE 3 – Schéma cinématique du robot quadrupède

2.2.2 Avec des noms "humains" :

Afin de simplifier la présentation, nous avons fait le choix d'utiliser des termes de membres humains. La figure 4 illustre les noms utilisés.

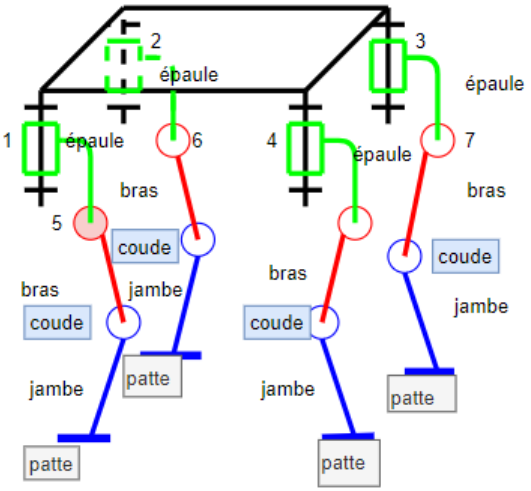


FIGURE 4 – Schéma cinématique du robot quadrupède

2.3 Modèle simplifié de la patte d'avant :

Du fait de la ressemblance de structure de deux patte d'avant, nous nous limitons notre étude que sur la trajectoire d'une seule patte.

2.3.1 Définitions de repères :

Nous définissons les repères associés à l'épaule, au bras et à la jambe selon le formalisme de Khalil et en respectant les notations de Denavit-Hartenberg :
- $(O_0, \vec{x_0}, \vec{y_0}, \vec{z_0})$ lié à l'épaule.
- $(O_1, \vec{x_1}, \vec{y_1}, \vec{z_1})$ lié au bras, avec $\overrightarrow{O_0O_1} = L_1\vec{x_1}$,$\vec{z_0} = \vec{z_1}$ et $(\vec{x_0}, \vec{x_1}) = q_1$.
- $(O_2, \vec{x_2}, \vec{y_2}, \vec{z_2})$ lié au jambe, avec $\overrightarrow{O_1O_2} = L_2\vec{x_2}$, $\vec{z_1} = \vec{z_2}$ et $(\vec{x_1}, \vec{x_2}) = q_2$.

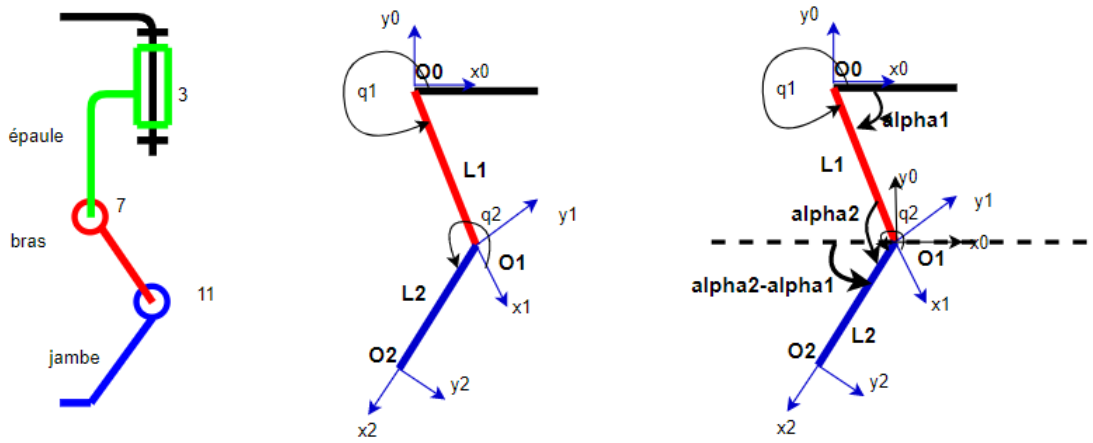


FIGURE 5 – Définitions de repères et d'angles

2.3.2 Équations cinématiques :

Nous considérons que la longueur du bras et de la jambe sont égale afin de simplifier le calcul.
Comme le système considéré ne contient que 2 liaisons pivot, nous pouvons utiliser les projections afin de déterminer les positions de coudesO_1 et de coussinets O_2. En se basant sur la figure (5), nous obtenons cela :

$$O_1 = O_0 + L * (cos(\pi + \alpha 1), sin(\pi + \alpha 1))$$
$$O_2 = O_1 + L * (cos(\alpha 2 - \alpha 1), -sin(\alpha 2 - \alpha 1))$$

Nous cherchons à exprimer les commandes α_1 et α_2 permettant de cibler un point dans l'espace de occupée par la patte de robot. Donc nous notons les coordonnées de O_2 par X et Y, alors :

$$X = -L * C_1 + L * C_{1-2}$$

$$Y = -L * S_1 + L * S_{1-2}$$

Nous calculons $X^2 + Y^2$, nous trouvons :

$$X^2 + Y^2 = 2 * L^2 - 2 * (C_1 C_{2-1} - S_1 S_{2-1})$$

Donc :

$$C_2 = 1 - \frac{X^2 + Y^2}{2L^2}$$

$$\boxed{alpha_2 = arccos(1 - \frac{X^2 + Y^2}{2L^2})}$$

Nous passons aux calcul de α_1. Pour cela, nous posons $A = C1$ et $B = S1$, donc les expressions de X et Y deviennent :

$$X = A(-L + LC_2) + LS_2B$$

$$Y = B(-L + LC_2) - LS_2A$$

L'équation sous la forme matricielle est la suivante :

$$\begin{pmatrix} -1 + C_2 & S_2 \\ -S_2 & -1 + C_2 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} X/L \\ Y/L \end{pmatrix}$$

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} -1 + C_2 & S_2 \\ -S_2 & -1 + C_2 \end{pmatrix}^{-1} \begin{pmatrix} X/L \\ Y/L \end{pmatrix}$$

Nous trouvons :

$$\begin{pmatrix} C_1 \\ S_1 \end{pmatrix} = \begin{pmatrix} \frac{(-1+C_2)*X/L - S_2 Y/L}{2-2C_2} \\ \frac{S_2 * X/L - (-1+C_2)Y/L}{2-2C_2} \end{pmatrix}$$

Par conséquent, les valeurs de α_1 sont :

$$\boxed{\alpha_1 = arcos(\frac{(-1 + C_2) * X/L - S_2 Y/L}{2 - 2C_2}) si S_1 > 0}$$

$$\boxed{\alpha_1 = -arcos(\frac{(-1 + C_2) * X/L - S_2 Y/L}{2 - 2C_2}) si S_1 < 0}$$

2.3.3 Trajectoire souhaité de la patte d'avant :

En choisissant judicieusement les angles α_1 et α_2, nous pouvons déterminer toutes les positions possibles de la patte, à condition que ces dernières soit dans l'espace de travail de la patte.
Dans la partie de modélisation numérique, nous avons choisie de faire un cycloïde.
Nous rappelons que l'équation paramétrique d'un cycloïde est défini par :

$$\begin{cases} x(\theta) & = R(\theta - sin(\theta)) \\ y(\theta) & = R(1 - cos(\theta) \end{cases}$$

2.4 Modèle simplifié de la patte d'arrière :

2.4.1 Définitions de repères :

Nous définissons les repères associés à l'épaule, au bras et à la jambe selon le formalisme de Khalil et en respectant les notations de Denavit-Hartenberg :
- $(O_0, \vec{x_0}, \vec{y_0}, \vec{z_0})$ lié à l'épaule.
- $(O_1, \vec{x_1}, \vec{y_1}, \vec{z_1})$ lié au bras, avec $\overrightarrow{O_0O_1} = L_1\vec{x_1}$,$\vec{z_0} = \vec{z_1}$ et $(\vec{x_0}, \vec{x_1}) = q_1$.
- $(O_2, \vec{x_2}, \vec{y_2}, \vec{z_2})$ lié au jambe, avec $\overrightarrow{O_1O_2} = L_2\vec{x_2}$, $\vec{z_1} = \vec{z_2}$ et $(\vec{x_1}, \vec{x_2}) = q_2$.

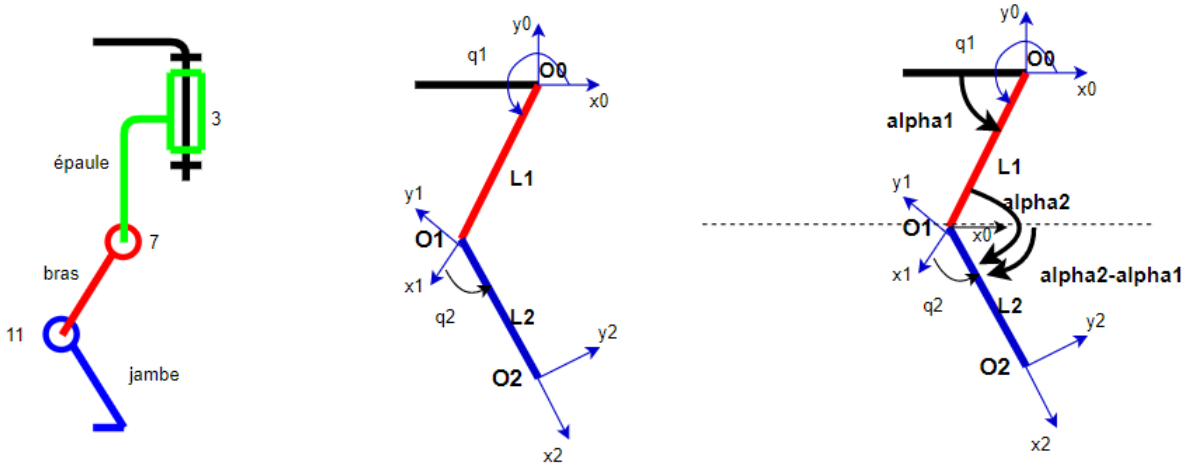


FIGURE 6 – Définitions de repères et d'angles

2.4.2 Équations cinématiques :

Nous considérons que la longueur du bras et de la jambe sont égale afin de simplifier le calcul.
Comme le système considéré ne contient que 2 liaisons pivot, nous pouvons utiliser les projections afin de déterminer les positions de coudesO_1 et de coussinets O_2. En se basant sur la figure (6), nous obtenons cela :

$$O_1 = O_0 + L * (cos(-\alpha_1), sin(-\alpha_1))$$
$$O_2 = O_1 + L * (cos(\pi + \alpha_2 - \alpha_1), -sin(\pi + \alpha_2 - \alpha_1))$$

Nous cherchons à exprimer les commandes α_1 et α_2 permettant de cibler un point dans l'espace de occupée par la patte de robot. Donc nous notons les coordonnées de O_2 par X et Y, alors :

$$X = L * C_1 + L * C_{\pi+2-1} = L * C_1 + L * C_{2-1}$$

$$Y = -L * S_1 + L * S_{\pi+2-1} = -L * S_1 - L * S_{2-1}$$

Nous calculons $X^2 + Y^2$, nous trouvons :

$$X^2 + Y^2 = 2 * L^2 * (1 + C_{2-1+\pi})$$

Donc :

$$\boxed{alpha_2 = arccos(1 - \frac{X^2 + Y^2}{2 * L^2})}$$

Nous passons aux calcul de α_1. Pour cela, nous posons $A = C1$ et $B = S1$, donc les expressions de X et Y deviennent :

$$X = A(L - LC_2) - LS_2B$$

$$Y = B(-L - LC_2) - LS_2A$$

L'équation sous la forme matricielle est la suivante :

$$\begin{pmatrix} 1 - C_2 & -S_2 \\ -S_2 & -1 + C_2 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} X/L \\ Y/L \end{pmatrix}$$

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 1 - C_2 & -S_2 \\ -S_2 & -1 + C_2 \end{pmatrix}^{-1} \begin{pmatrix} X/L \\ Y/L \end{pmatrix}$$

Nous trouvons :

$$\begin{pmatrix} C_1 \\ S_1 \end{pmatrix} = \begin{pmatrix} \frac{-S_2 * Y}{L*(C_2^2 - 2*C_2 + S2^2 + 1)} - \frac{X(C_2 - 1)}{L*(C_2^2 - 2*C_2 + S2^2 + 1)} \\ \frac{Y*(C_2 - 1)}{L*(C_2^2 - 2*C_2 + S2^2 + 1)} - \frac{S_2 * X}{L*(C_2^2 - 2*C_2 + S2^2 + 1)} \end{pmatrix}$$

Par conséquent, les valeurs de α_1 sont :

$$\boxed{\alpha_1 = arcos(\frac{-S_2 * Y}{L * (C_2^2 - 2 * C_2 + S2^2 + 1)} - \frac{X(C_2 - 1)}{L * (C_2^2 - 2 * C_2 + S2^2 + 1)}) si S_1 > 0}$$

$$\boxed{\alpha_1 = -arcos(\frac{Y * (C_2 - 1)}{L * (C_2^2 - 2 * C_2 + S2^2 + 1)} - \frac{S_2 * X}{L * (C_2^2 - 2 * C_2 + S2^2 + 1)}) si S_1 < 0}$$

2.4.3 Trajectoire souhaité de la patte d'avant :

En choisissant judicieusement les angles $alpha_1$ et $alpha_2$, nous pouvons déterminer toutes les positions possibles de la patte, à condition que ces dernières soit dans l'espace de travail de la patte.
Dans la partie de modélisation numérique, nous avons choisie de faire un cycloïde.
Nous rappelons que l'équation paramétrique d'un cycloïde est défini par :

$$\begin{cases} x(\theta) & = R(\theta - sin(\theta)) \\ y(\theta) & = R(1 - cos(\theta) \end{cases}$$

2.5 Type de marche de robot quadrupède :

2.5.1 Vocabulaire :

La démarche est la méthode de déplacement du robot. Elle comporte à la fois la séquence de dépôt des pattes et la posture du corps dans l'espace. Les robots de plus de quatre pattes peuvent tous utiliser des démarches très similaires.
Il est important de connaître les termes généraux afin de comprendre la démarche. Le cycle de chaque patte se sépare en deux phases, le balancement "SWING" et la posture "STANCE". Le balancement est le portion de la séquence où la patte est dans les airs. La posture est le moment où la patte est au sol. Ces deux phases composent le pas ou la foulée.
Le pas se définit de façon globale par quatre paramètres : la longueur, la période, le rapport de rendement (RR) et la hauteur. Ces quatre paramètres permettent de définir l'allure d'un cycle complet d'une patte.
La longueur et la hauteur du pas représentent sa forme dans l'espace.
La période et le rapport de rendement (RR) décrivent le mouvement dans le temps. La période est le temps nécessaire pour faire un pas.
Le RR représente la portion du cycle où la patte est en phase de posture (équation 1.2). Il est défini comme suit :

Les robots qui marchent de façon statique utilisent en général deux types de démarche. Ils rampent "crawl" ou se dandine "wave".

Les paramètres sont les mêmes d'une séquence à l'autre sauf l'ordre de levée de chaque patte. Elles se reproduisent dans le temps sans se modifier. Ce type de démarche ne s'adapte pas vraiment à des changements brusques d'environnement. Elles sont répétitives et ne considèrent pas de méthode afin modifier leur séquence pour considérer les changements.

2.5.2 La marche ciblée dans le projet :

La figure 7 définit la notations par chiffre des 4 pattes.

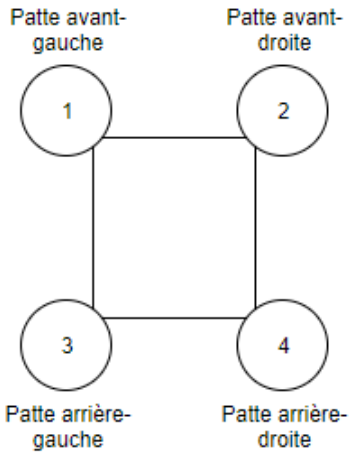


FIGURE 7 – Définitions de différentes positions de pattes

La figure 8 montre la chronogramme de marche de 4 pattes sur une période de 2T. En effet, deux pattes seront actionnées simultanément pendant chaque demi période.

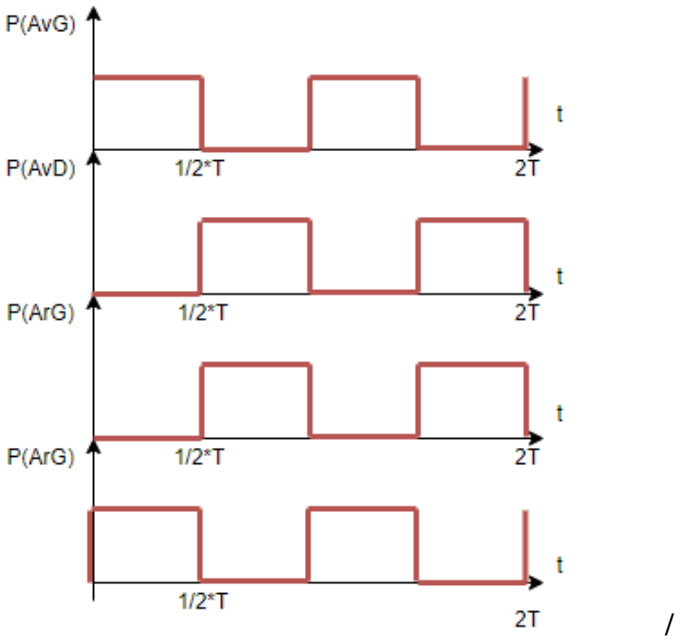


FIGURE 8 – Chronogramme de marche de pattes

3 Modélisation numérique :

3.1 Construction du modèle de base :

3.1.1 code :

La base constitue la partie bâti du robot. La figure (9) illustre la création de l'épaule d'avant et d'arrière. Dans cette étape, nous avons initialisé les angles α_1 (a1) et α_2 (a2).

```
L = 4; %Longueur d'une patte

Ef = [6;0]; %Position de l' paule  avant
Er = Ef - [2.5*L;0]; %Position de l' paule  arri re

Base = [Er,Ef]; %la base du robot

a1 = 1;
a2 = 2*a1;
a3 = 1;
a4 = 2*a3;
```

Les différentes positions sont :

```
Cf = [L*cos(a1+pi);L*sin(a1+pi)]+Ef; %position du coude d'avant
Pf = Cf+[L*cos(a1-a2);L*sin(a1-a2)]; %position d' paule  d'avant

Cr = [L*cos(-a1);L*sin(-a1)]+Er; %position du coude arri re
Pr = Cr+[L*cos(pi+a2-a1);L*sin(pi+a2-a1)];%position d' paule  arri re
```

3.1.2 Affichage :

Nous obtenons l'affichage avec le code suivant :

```
plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'k')
plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'ok') %repr sente le 'Ok' de l'
    articulation au niveau de la coude.

plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'k')
plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'ok') %repr sente le 'Ok' de l'
    articulation au niveau de la coude.


%% TRACE PATTE ARRIERE
plot([Er(1),Cr(1)],[Er(2),Cr(2)],'k')
plot([Er(1),Cr(1)],[Er(2),Cr(2)],'ok')


plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'k')
plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'ok')
hold off
```

La représentation de la base de robot et la position initiale obtenue est la suivante :

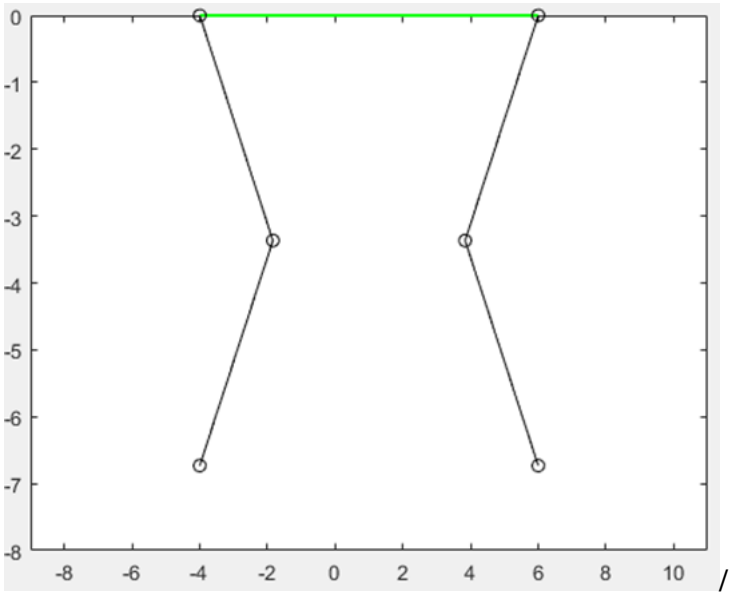


FIGURE 9 – La représentation de la base de robot et la position initiale

3.2 Simulation de la patte avant et arrière :

3.2.1 Calcul de la position de la patte d'avant :

```
[X1,Z2] = ginput(1); %Le point objectif dans le rep re de la base
A1 = (X1 - Ef(1))/L; % (mise    jour par rapport    X1 et Z1))
B1 = (Z2-Ef(2))/L;
alpha2_a_d1=acos(1-0.5*(A1^2+B1^2));
Mat1 =  [cos(alpha2_a_d1)-1   ,  -sin(alpha2_a_d1);
        - sin(alpha2_a_d1)    ,cos(alpha2_a_d1)+1];
Imat1 = inv(Mat1);
Result1 = Imat1*[A1; B1];
alpha1_a_d1=acos(Result1(1));
a11 = alpha1_a_d1;
a21 = alpha2_a_d1;
```

3.2.2 Calcul de la position de la patte d'arrière :

```
[Xc,Zc] = ginput(1);
A = (Xc - Ef(1))/L;
B = (Zc-Ef(2))/L;
alpha2_a_d=acos(1-0.5*(A^2+B^2));
```

```
Mat =  [cos(alpha2_a_d)-1   ,   sin(alpha2_a_d);
        - sin(alpha2_a_d)     ,cos(alpha2_a_d)-1];
Imat = inv(Mat);
Result = Imat*[A; B];
alpha1_a_d=acos(Result(1));
a1 = alpha1_a_d;
a2 = alpha2_a_d;
```

3.2.3 Simulation statique de la patte avant et arrière :

Code :

```
Cf = [L*cos(a1+pi);L*sin(a1+pi)]+Ef; %position du coude

Pf = Cf+[L*cos(a1-a2);L*sin(a1-a2)];%position d'epaule

Cr = [L*cos(-a1);L*sin(-a1)]+Er; %position du coude rear
Pr = Cr+[L*cos(pi+a2-a1);L*sin(pi+a2-a1)]; %position d'epaule rear \\
```

```
hold on\\

plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'k');
plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'ok');


plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'k');
plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'ok');

plot([Er(1),Cr(1)],[Er(2),Cr(2)],'k');
plot([Er(1),Cr(1)],[Er(2),Cr(2)],'ok');


plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'k');
plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'ok');


hold off
```

Affichage :

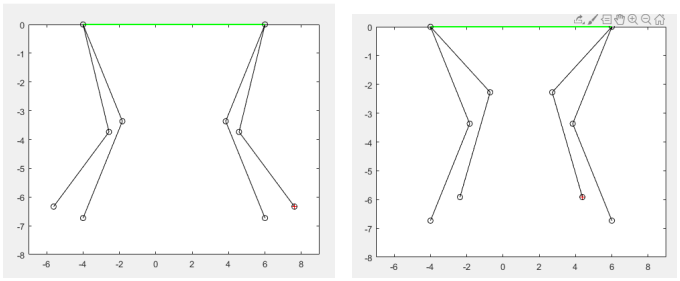


FIGURE 10 – Définitions de repères et d'angles

3.2.4 Simulation de la trajectoire souhaité de la patte d'avant :

Code :

```
X1=[3 , 3.36 , 4 , 5 , 6 , 7 , 8 , 8.64 , 9 ,7.5 ,6 ,5 ,4.5 ,3.5 ,3 ];%le cycoilde choisi repere X
Z1=[-7, -6.5 , -6.076 , -5.8 , -5.8 , -5.8 , -6.076 , -6.5 , -7 ,-7 ,-7,-7,-7 ,-7,-7 ]; %Le cycloide choisi reper Y
l=length(X1);
A1 = (X1(:)- Ef(1))/L;
B1 = (Z1(:)-Ef(2))/L;
alpha2_a_d1=acos(1-0.5*(A1(:).^2+B1(:).^2));%calcule du l'ongle alpha2
for j=1:l
    a=cos(alpha2_a_d1(j))-1;
    b=sin(alpha2_a_d1(j));
    c=-sin(alpha2_a_d1(j));
    d=cos(alpha2_a_d1(j))-1;
    M=[a,b;c,d];

    Imat1 = inv(M);
    Result1 = Imat1*[A1(j); B1(j)];
    alpha1_a_d1(j)=acos(Result1(1));
    a1(j) = alpha1_a_d1(j); %calcule du l'ongle alpha1
    a2(j) = alpha2_a_d1(j); %calcule du l'ongle alpha2
end
```

3.2.5 Simulation de la trajectoire souhaité de la patte d'arrière :

Code :

```
Xr1=[-7 , -6.64 , -6 , - 5 , - 4 , -3 ,
-2 , -1.36 , -1 ,-2.5 ,-3,-4.5 ,-5.5
,-6.5,-7 ];
Zr1=[-7, -6.5 , -6.076 , -5.8 , -5.8 , -5.8 , -6.076
, -6.5 , -7 ,-7 ,-7 ,-7 ,-7,-7,-7];
Ar = (Xr1(:) - Er(1))/L;
Br = (Zr1(:)-Er(2))/L;
alpha2r=acos(1-0.5*(Ar(:).^2+Br(:).^2));
for j=1:1
    a=-cos(alpha2r(j))+1;
    b=-sin(alpha2r(j));
    c= -sin(alpha2r(j));
    d=cos(alpha2r(j))-1;
    Mr=[a,b;c,d]
    Imatr = inv(Mr);
    Resultr = Imatr*[Ar(j); Br(j)];
    alpha1r(j)=acos(Resultr(1));

    ar1(j) =alpha1r(j);
    ar2 (j)= alpha2r(j);
end
```

3.2.6 Affichage de cycloïde de deux pattes :

Code :

```
%%%affichage
j=[1:1:1];
Cf = [L*cos(a1(j)+pi);L*sin(a1(j)+pi)]+Ef; %position du coude
Pf = Cf+[L*cos(a1(j)-a2(j));L*sin(a1(j)-a2(j))];%position d'epaule

Cr = [L*cos(-ar1(j));L*sin(-ar1(j))]+Er; %position du coude rear
Pr = Cr+[L*cos(pi+ar2(j)-ar1(j));L*sin(pi+ar2(j)-ar1(j))];%position d'
    epaule rear
```

```
%%affichage
figure(2);

plot(Base(1,:),Base(2,:),'Color','Green','LineWidth',1.5);
axis([Base(1,1)-5, Base(1,2)+5, -8,0]);
```

```
hold on

```

```
for i=1:2:((l*2)-1)
    %tracahge du patte avant
    plot([Ef(1),Cf(i)],[Ef(2),Cf(i+1)],'k') (7)
    plot([Ef(1),Cf(i)],[Ef(2),Cf(i+1)],'ok') (8)

    hold on

    plot([Cf(i),Pf(i)],[Cf(i+1),Pf(i+1)],'k') (6)
    plot([Cf(i),Pf(i)],[Cf(i+1),Pf(i+1)],'or') (5)

    %--tracage du patte arriere

    plot([Er(1),Cr(i)],[Er(2),Cr(i+1)],'k') (4)
    plot([Er(1),Cr(i)],[Ef(2),Cr(i+1)],'ok') (3)

     hold on

    plot([Cr(i),Pr(i)],[Cr(i+1),Pr(i+1)],'k') (2)
    plot([Cr(i),Pr(i)],[Cr(i+1),Pr(i+1)],'or') (1)

end
```

3.2.7 Simulation animée de la patte avant et arrière :

Code :

```
%affichage anim
F1 = figure(1);

%tracahge model du base
plot(Base(1,:),Base(2,:),'Color','Green','LineWidth',1.5) (9)
axis([Base(1,1)-5, Base(1,2)+5, -8,0]);
hold on

plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'k')
plot([Ef(1),Cf(1)],[Ef(2),Cf(2)],'ok')


plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'k')
plot([Cf(1),Pf(1)],[Cf(2),Pf(2)],'ok')


%TRACE PATTE ARRIERE
plot([Er(1),Cr(1)],[Er(2),Cr(2)],'k')
plot([Er(1),Cr(1)],[Er(2),Cr(2)],'ok')


plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'k')
plot([Cr(1),Pr(1)],[Cr(2),Pr(2)],'ok')
```

```
%affichage anim
F1 = figure(1)
hold off
Axe = F1.Children(1);

Socle1 =Axe.Children(9);
AvantBras_r = Axe.Children(4);
JointsAvantBras_r = Axe.Children(3);

Bras_r = Axe.Children(2);
JointsBras_r = Axe.Children(1);

%bras avant

AvantBras = Axe.Children(8);
JointsAvantBras = Axe.Children(7);

Bras = Axe.Children(6);\\
JointsBras = Axe.Children(5);


for k= 1:3
    j=17;
    for i=1:2:((1*2)-1)v
        AvantBras.XData = [Ef(1),Cf(j)];
        JointsAvantBras.XData = [Ef(1),Cf(j)];

        AvantBras.YData = [Ef(2),Cf(j+1)];
        JointsAvantBras.YData = [Ef(2),Cf(j+1)];

        Bras.XData = [Cf(j),Pf(j)];
        JointsBras.XData = [Cf(j),Pf(j)];

        Bras.YData = [Cf(j+1),Pf(j+1)];
        JointsBras.YData = [Cf(j+1),Pf(j+1)];

        AvantBras_r.XData = [Er(1),Cr(i)];
        JointsAvantBras_r.XData = [Er(1),Cr(i)];

        AvantBras_r.YData = [Er(2),Cr(i+1)];
        JointsAvantBras_r.YData = [Er(2),Cr(i+1)];

        Bras_r.XData = [Cr(i),Pr(i)];
        JointsBras_r.XData = [Cr(i),Pr(i)];

        Bras_r.YData = [Cr(i+1),Pr(i+1)];
        JointsBras_r.YData = [Cr(i+1),Pr(i+1)];

        if j==((1*2)-1)
            j=1;
        else
            j=j+2;
        end
      %patte rear
```

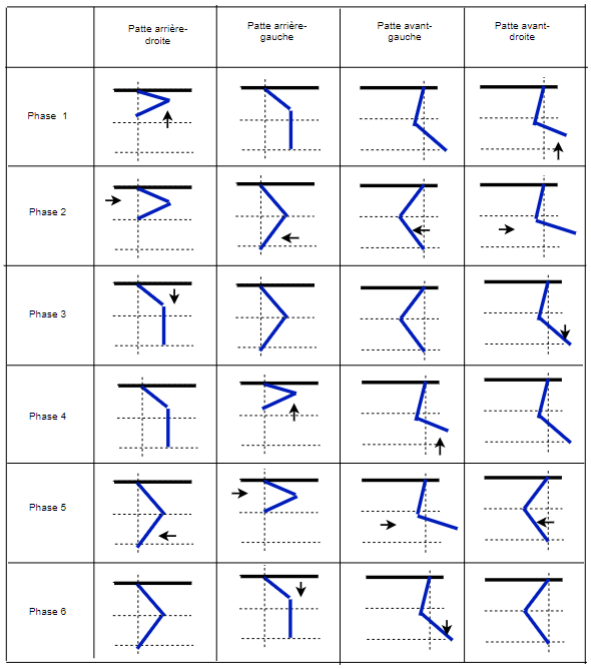
```
%avant

    pause (1)
    figure(F1)
end
```

Affichage :

4 Expérimentation sur Arduino :

4.1 Principe :

La figure (11) illustre le type de marche choisi pour permettre au robot de marcher.



/

FIGURE 11 – Les différents mouvement des pattes pour la marche de robot

Les commentaires marquées sur le code Arduino permettent de bien comprendre comment le code a été construit.

4.2 Code sur Arduino :

```
//include des librairies
#include <Servo.h> // librairie servomoteurs
#include <SPI.h> // librairie du doute
#include <time.h> // librairie temps
//**********definition des differentes parties des pattes du robot
#define coudeavantdroite 0
#define coudeavantgauche 1
#define coudearrieregauche 2
#define coudearrieredroite 3
#define epauleavantdroite 4
#define epauleavantgauche 5
#define epaulearrieregauche 6
#define epaulearrieredroite 7
#define jambehautdroite 8
#define jambehautgauche 9
#define jambebasgauche 10
#define jambebasdroite 11
//**********definition des positions des pattes
#define avantdroite 20
#define avantgauche 21
#define arrieredroite 22
#define arrieregauche 23
//**********definition des constantes
const float pi = 3.1415;
const float L0 = 16.4; //longueur de la base
const float L1 = 5.1;//longueur de
const float L2 = 5; // longueur de
// **************angles utilis  pour les calcules******************
float angle[]={0,0};
float angle1[]={0,0};
float angle2[] = {0,0};
typedef struct Patte Patte;
//********************************CREATION STRUCTURE PATTE
    *********************************
struct Patte
{
  int id;
  int coude;
  int epaule;
  float angleepaule;
  float anglecoude;
};
//****************************** declaration des pattes
    ******************************************
Patte patteavantgauche;
Patte patteavantdroite;
Patte pattearrieregauche;
Patte pattearrieredroite;
//************************************VARIABLES SERVOMOTEUR
    ****************************************
Servo servo0;
```

```
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
Servo servo6;
Servo servo7;
Servo servo8;
Servo servo9;
Servo servo10;
Servo servo11;

//**************************************************SETUP
    ***********************************************/
void setup() {

  delay(200); // Pause de 200 millisecondes

  Serial.begin(9600); //Initialisation de la communication s rie

  // Assignation des ports servomteurs (Vue de face : fil de
     communication vers nous, l   o   il y a les yeux)
  servo0.attach(0); // coude avant droit
  servo1.attach(1); // coude avant gauche
  servo2.attach(2); // coude arri re gauche
  servo3.attach(3); // coude arri re droit
  servo4.attach(4); //   paule   avant droit
  servo5.attach(5); //   paule   avant gauche
  servo6.attach(6); //   paule   arri re gauche
  servo7.attach(7); //   paule   arri re droite
  servo8.attach(18); // jambe haut droite
  servo9.attach(10); // jambe haut gauche
  servo10.attach(8); // jambe bas gauche
  servo11.attach(19); // jambe bas droite
// definir les attributs des pattes
patteavantgauche.id = avantgauche;
patteavantgauche.coude = coudeavantgauche;
patteavantgauche.epaule = epauleavantgauche;


patteavantdroite.id = avantdroite;
patteavantdroite.coude = coudeavantdroite;
patteavantdroite.epaule = epauleavantdroite;


pattearrieregauche.id = arrieregauche;
pattearrieregauche.coude = coudearrieregauche;
pattearrieregauche.epaule = epaulearrieregauche;


pattearrieredroite.id = arrieredroite;
pattearrieredroite.coude = coudearrieredroite;
pattearrieredroite.epaule = epaulearrieredroite;
delay(5000);
// mettre le robot dans sa position initiale( debout)
```

```
102   initialiser();
103   delay(2000);
104 }
105 //**********************************BOUCLE PRINCIPALE
        ************************
106 void loop() {
107   assis();
108   delay(5000);
109   initialiser();
110   delay(5000);
111   lever(patteavantgauche,pattearrieredroite);
112   delay(5000);
113   marcher();
114 }
115 //***********fonction utilis  pour calibrer les servo**********
116 void calibrer(int moteur,int initiale)
117 {
118    CommandeMoteur(moteur,180-initiale);
119    delay(2000);
120    for (int i=0; i<180; i+=5)
121    {
122       CommandeMoteur(moteur,180-(initiale+i));
123       delay(3000);
124    }
125 }
126 //*********************BOUGER UNE PATTE
        *********************************
127 void bougerpatte(Patte* patte, float angle[])
128 {
129   if(patte->id == avantgauche || patte->id == arrieredroite)
130   {
131     patte->angleepaule=angle[0];
132     patte->anglecoude=angle[1];
133     CommandeMoteur(patte->epaule,angle[0]);
134     CommandeMoteur(patte->coude,angle[1]);
135   }
136   else
137    {
138     patte->angleepaule=angle[0];
139     patte->anglecoude=angle[1];
140     CommandeMoteur(patte->epaule,180-angle[0]);
141     CommandeMoteur(patte->coude,180-angle[1]);
142   }
143 }
144 //*************************COMMANDE UNE SERVOMOTEUR
        ************************
145 void CommandeMoteur(int moteur, int angle){
146
147   Servo servo;
148
149   switch (moteur){
150     case 0:
151       servo = servo0;
152       break;
153     case 1:
```

```
184         servo = servo1;
185         break;
186       case 2:
187         servo = servo2;
188         break;
189       case 3:
190         servo = servo3;
191         break;
192       case 4:
193         servo = servo4;
194         break;
195       case 5:
196         servo = servo5;
197         break;
198       case 6:
199         servo = servo6;
200         break;
201       case 7:
202         servo = servo7;
203         break;
204       case 8:
205         servo = servo8;
206         break;
207       case 9:
208         servo = servo9;
209         break;
210       case 10:
211         servo = servo10;
212         break;
213       case 11:
214         servo = servo11;
215         break;
216       }
217
218       Pente(servo,angle);
219       //servo.write(angle);
220     }
221   //**************************REGLER VITESSE SERVOMOTEUR
        ***************************
222   void Pente(Servo servo, int consigne_angle){
223
224     int previous_angle = servo.read();
225     int angle = previous_angle;
226     int interval = 50;
227
228     if (previous_angle < consigne_angle){
229       while(angle < consigne_angle){
230         delay(interval);
231         angle += 5;
232         angle = constrain(angle,previous_angle,consigne_angle);
233         servo.write(angle);
234       }
235     }
236     else{
237       while(angle > consigne_angle){
```

```
208         delay(interval);
209         angle -= 5;
210         angle = constrain(angle,consigne_angle,previous_angle);
211         servo.write(angle);
212       }
213     }
214
215   }
216
217
218   //*******************************CONVERSION DES ANGLES
          *******************************
219   float radtodeg(float angle)
220   {
221     return (angle*180)/pi;
222   }
223   float degtorad(float angle)
224   {
225     return (angle*pi)/180;
226   }
227
228   //*******************************AVANCER UNE PATTE
          *********************************
229   void avancer (Patte patte)
230   {
231     angle[0] = patte.angleepaule;
232     angle[1] = patte.anglecoude;
233     float d_base_sol;
234     float d_leve;
235     d_base_sol = L1*sin(degtorad(angle[0]))+L2*sin(degtorad(angle[1]-angle
          [0]));
236     d_leve = L1*sin(degtorad(angle[0]));
237
238   //   angle[1]=radtodeg(pi-(asin((d_leve-L1*sin(degtorad(angle[0])))/L2)+
        degtorad(angle[0])));// lever la patte
239     if ( patte.id==arrieregauche||patte.id==arrieredroite)
240     angle[1]=45;
241     else
242     angle[1]= 20;
243     bougerpatte(&patte,angle);
244
245   //   delay(1000);
246     if (patte.id==arrieregauche || patte.id == arrieredroite)
247       angle[0]-=5*5;
248     else
249       angle[0]+=5*5;
250   //angle[1]=radtodeg(pi-(asin((d_leve-L1*sin(degtorad(angle[0])))/L2)+
        degtorad(angle[0])));
251   // avancer la patte en le maintenant lev
252     bougerpatte(&patte,angle);
253
254   //   delay (1000);
255   if(patte.id == arrieregauche||patte.id == arrieredroite)
256   {
257     angle[0] = radtodeg(asin((d_base_sol-L2)/L1))+10;
```

```
258     angle[1] = 90+angle[0];
259   }
260
261   else
262     angle[1] = radtodeg(pi-(asin((d_base_sol-L1*sin(degtorad(angle[0])))/L2
          )+degtorad(angle[0])));
263     bougerpatte(&patte,angle); //baisser la patte
264   //  delay (1000);
265   }
266
267   //*********************************ASSIS
          ****************************************
268   void assis()
269   {
270     angle[0] = 0;
271     angle[1] = 0;
272     synchroniser(&patteavantgauche,&patteavantdroite,angle,angle,10);
273   }
274
275   //*********************************INITIALISER LE ROBOT
          **********************************
276   void initialiser()
277   {
278      angle[0]=45;
279      angle[1]=90;
280      CommandeMoteur(jambehautdroite,80);
281      CommandeMoteur(jambehautgauche,95);
282      CommandeMoteur(jambebasdroite,82.5);
283      CommandeMoteur(jambebasgauche,85);
284
285      synchroniser(&patteavantgauche,&pattearrieredroite,angle,angle,25);
286      synchroniser(&pattearrieregauche,&patteavantdroite,angle,angle,25);
287   }
288   //*********************************MARCHER**********************
289   void marcher()
290   {
291   //  lever(patteavantdroite,pattearrieregauche,patteavantgauche,
          pattearrieredroite);
292     lever(patteavantdroite,pattearrieregauche);
293     pousser(patteavantgauche,pattearrieredroite);
294     poser(patteavantdroite,pattearrieregauche);
295     delay(200);
296   //  lever(patteavantgauche,pattearrieredroite,patteavantdroite,
          pattearrieregauche);
297     lever(patteavantgauche,pattearrieredroite);
298     pousser(patteavantdroite,pattearrieregauche);
299     poser(patteavantgauche,pattearrieredroite);
300     delay(200);
301   }
302   //*********************************LEVER DEUX PATTES EN MEME TEMPS
          *************************
303   void lever(Patte patteAv, Patte patteAr)
304   {
305     angle[0] = patteAv.angleepaule;
306     angle1[0] = patteAr.angleepaule;
```

```
307     angle1[1]=45;
308     angle[1]=20;
309     synchroniser(&patteAv,&patteAr,angle,angle1,10);
310   }
311   //*******************************AVANCE DEUX PATTES EN MEME TEMPS
          *************************
312   void avancesynchrone(Patte patteAv, Patte patteAr)
313   {
314     angle[0] = patteAv.angleepaule;
315     angle[1] = patteAv.anglecoude;
316     angle1[0] = patteAr.angleepaule;
317     angle1[1] = patteAr.anglecoude;
318     float d_base_sol;
319     float d_leve;
320     d_base_sol = L1*sin(degtorad(angle[0]))+L2*sin(degtorad(angle[1]-angle
            [0]));
321     d_leve = L1*sin(degtorad(angle[0]));
322
323     angle1[1]=45;
324     angle[1]= 20;
325     synchroniser(&patteAv, &patteAr,angle,angle1,5);
326
327     angle1[0]-=5*5;
328     angle[0]+=5*5;
329     synchroniser(&patteAv, &patteAr,angle,angle1,5);
330
331
332     angle1[0] = radtodeg(asin((d_base_sol-L2)/L1));
333     angle1[1] = 90+angle1[0];
334     angle1[0]+=5;
335     angle[1] = radtodeg(pi-(asin((d_base_sol-L1*sin(degtorad(angle[0])))/L2
            )+degtorad(angle[0])))+5;
336     synchroniser(&patteAv, &patteAr,angle,angle1,5);
337   }
338   //*********************************AVANCER ET POSER DEUX PATTES LEVES
          PRECEDEMENT*************************
339   void poser(Patte patteAv, Patte patteAr)
340   {
341     angle[1] = patteAv.anglecoude;
342     angle1[1] = patteAr.anglecoude;
343     float d_base_sol;
344     d_base_sol = L1*sin(degtorad(45))+L2*sin(degtorad(45));
345
346     angle1[0]=patteAr.angleepaule-5*5;
347     angle[0]=patteAv.angleepaule+5*5;
348     synchroniser(&patteAv, &patteAr,angle,angle1,5);
349
350     angle1[0] = radtodeg(asin((d_base_sol-L2)/L1));
351     angle1[1] = 90+angle1[0];
352     angle1[0]+=10;
353     angle[1] = radtodeg(pi-(asin((d_base_sol-L1*sin(degtorad(angle[0])))/L2
            )+degtorad(angle[0])))+20;
354     synchroniser(&patteAv, &patteAr,angle,angle1,5);
355   }
356   //*********************************POUSSER LE ROBOT EN AVANT AVEC DEUX
```

```
    PATTES*****************************
void pousser(Patte patteAv, Patte patteAr)
{
  angle[0]=45;
  angle[1]=90;
  angle1[0]=45;
  angle1[1]=90;
  synchroniser(&patteAv, &patteAr,angle,angle1,10);
}
//*****************************FAIRE BOUGER DEUX PATTES EN MEME TEMPS
    *****************************
void synchroniser(Patte* patte1,Patte* patte2,float angle[],float angle1
    [],int interval)
{
  Servo coude1 = choisirservo(patte1->coude);
  Servo epaule1= choisirservo(patte1->epaule);
  Servo coude2= choisirservo(patte2->coude);
  Servo epaule2= choisirservo(patte2->epaule);
  float previous_coude2 = coude2.read();
  float previous_coude1 = coude1.read();
  float previous_epaule2 = epaule2.read();
  float previous_epaule1 = epaule1.read();

  patte1->angleepaule = angle[0];
  patte2->angleepaule = angle1[0];
  patte1->anglecoude = angle[1];
  patte2->anglecoude = angle1[1];

  float consigne_angle[] = { angle[0], angle[1], angle1[0], angle1[1]};

  if (patte1->id == avantdroite || patte1->id == arrieregauche)
  {
     consigne_angle[0] = 180-patte1->angleepaule;
     consigne_angle[1] = 180-patte1->anglecoude;
  }
  if (patte2->id == avantdroite || patte2->id == arrieregauche)
  {
     consigne_angle[2] = 180-patte2->angleepaule;
     consigne_angle[3] = 180-patte2->anglecoude;
  }

  float previous_angle[] = {previous_epaule1,previous_coude1,
      previous_epaule2,previous_coude2};
  Servo servo[] = {epaule1,coude1,epaule2,coude2} ;
  float Angle[] = {previous_epaule1,previous_coude1,previous_epaule2,
      previous_coude2};

  int i = 0;
  while(Angle[0]!=consigne_angle[0] || Angle[1] != consigne_angle[1] ||
      Angle[2] != consigne_angle[2] || Angle[3] != consigne_angle[3])
  {
    for (i=0; i!=4; i++)
    {
       if (previous_angle[i] < consigne_angle[i])
       {
```

```
406             delay(interval);
407             Angle[i] += 5;
408             Angle[i] = constrain(Angle[i],previous_angle[i],consigne_angle[
                    i]);
409             servo[i].write(Angle[i]);
410           }
411           else
412           {
413             delay(interval);
414             Angle[i] -= 5;
415             Angle[i] = constrain(Angle[i],consigne_angle[i],previous_angle[
                    i]);
416             servo[i].write(Angle[i]);
417           }
418       }
419     }
420   }
421   //*****************************IDENTIIER UN SERVO PAR LE NOM DE L'
          ARTICULATION************************
422   Servo choisirservo(int moteur)
423   {
424     Servo servo;
425
426     switch (moteur){
427       case 0:
428         servo = servo0;
429         break;
430       case 1:
431         servo = servo1;
432         break;
433       case 2:
434         servo = servo2;
435         break;
436       case 3:
437         servo = servo3;
438         break;
439       case 4:
440         servo = servo4;
441         break;
442       case 5:
443         servo = servo5;
444         break;
445       case 6:
446         servo = servo6;
447         break;
448       case 7:
449         servo = servo7;
450         break;
451       case 8:
452         servo = servo8;
453         break;
454       case 9:
455         servo = servo9;
456         break;
457       case 10:
```

```
458         servo = servo10;
459         break;
460       case 11:
461         servo = servo11;
462         break;
463       }
464       return servo;
465   }
```

5 Conclusion :

Le travail réalisé dans ce projet nous a permis de surpasser dans la modélisation d'un système mécanique, la commande, l'implantation dans un algorithme de simulation et l'exploitation en conditions réels.
Les prochaines équipes qui souhaiteront s'intéresser à ce projet pourront contribuer à l'élaboration d'autres postures pour ce robot, penser à faire marcher autrement et peut-être faire des choses attrayantes avec le code donné. Nous pouvons améliorer les performances physiques de ce dernier (nouveaux servomoteurs, de nouvelles pièces, modifier le matériaux qu'il y entre les pattes et le sol par un autre d'un facteur de frottement plus important et essayer de mettre aux pattes de robot des surfaces sphériques).
Une extension à ce dernier serait de le paramétrer, de faire une modélisation en trois dimensions du robot de manière à ce qu'il puisse changer de direction dans l'espace et aussi d'optimiser l'algorithme de marche en prenant compte la distance entre le robot et le sol et la distance des pattes par rapport au centre simultanément.