Projet de semestre Été 2003
Modélisation d'un Robot Salamandre

EPFL
Logics Sytems Laboratory

Encadré par Pr. Auke J. Ijspeert

CAPERN Simon

9 juillet 2003

Table des matières

1 Introduction

Ce rapport présente le travail effectué tout au long du semestre d'été 2003 dans le cadre d'un projet de modélisation d'un robot bio-inspiré. Le but sous-jacent à ce travail est le développement d'un vrai robot basé sur la morphologie et le comportement d'une salamandre.

Ce projet a donc pour but d'étudier les différents paramètres de ce modèle et leur influence sur celui-ci et son comportement.

Pour que ces tests soient valides et aident au développement et à la conception d'un prototype, cette simulation doit être la plus réaliste possible. Elle doit donc intégrer les lois physiques qui régissent son futur environnement comme les forces gravitationnelles mais aussi les lois mécaniques qui contrôlent le robot, les couples et les forces mises en jeux lors de ses déplacements.

Cette "salamandre "devra pouvoir se mouvoir sur le sol de deux manières possibles : la locomotion dite *serpentine* empreintée au serpent qui n'utilise que l'ondulation du corps pour ce déplacer et une méthode plus conventionnelle : la marche, qui utilise alors les pattes du robot. Par manque de temps, nous n'avons pas pu modéliser la nage pour ce robot.

De plus on doit pouvoir modifier l'environnement pour tester l'aptitude de ce robot face aux obstacles et aux variations de terrains.

Cette modélisation doit donc nous aider à déterminer principalement deux choses : premièrement les fonctions optimales de contrôles et dans un deuxième temps la géométrie des pattes la plus adaptée.

2 Revue littéraire

Le domaine de la robotique est à l'heure actuelle en plain essor. En effet avec les progrès technologiques en miniaturisation les robots sont de plus en plus étudiés pour des raisons diverses : soit dans un but purement expérimental pour recréer par exemple des comportements d'autonomie ou de cognition, soit dans un but plus pratique comme les robots envoyés sur Mars ou ceux servant à l'exploration d'endroits inaccesibles à l'homme comme des pipelines ou des réacteurs nucléaires. Ces quelques robots ne sont qu'un petit exemple de l'étendu des idées et des concepts bio-inspirés. . Rien que pour le serpent on peut trouver une vingtaine de référence sur http ://ais.gmd.de/%7Eworst/snake-collection.html qui référence uniquement les robots sans pattes.

Robots bioinspirés et locomotion

Projet PIRIA cherche à mettre en place un robot-serpent autonome, reconfigurable dont les segments peuvent être remplacé en cas de panne. L'articulation et la motorisation de ce robot se fait au niveau des joints. Ce robot a développé un mode de déplacement basé sur la rotation de chaque segment, cela lui permet de se mouvoir sans friction au niveau du sol.(http ://www.sics.se/ mn/dragon.htm)

GMD-snake le but de ce projet est d'imiter au tant que possible le mouvment d'un serpent. Ce robot et compose de plusieurs sections cylindriques qui peuvent se déformer indépendamment les unes des autres. L'articulation se fait au niveau des segments et non au niveau des joints comme nous avons pu le voir précédemment.(http ://ais.gmd.de/BAR/snake.html)

RHEX Ce robot autonome est munis de six pattes élastiques qui lui permettent de marcher, courir ou sautiller. Ce robot se révèle être très agile et flexible. Ses pattes tournent à la manières de celles que l'on veut implémenter sauf que le premier membre de la patte est atrophié.(http ://www.rhex.net)

FIG. 1 – a)PIRIA b)GMD-Snake

3 ODE

Open Dynamics Engine[1] est une librairie libre de simulation physique développée par Russell Smith. Elle permet la simulation de corps dynamiques dans un environnement virtuel.

Elle implémente un moteur physique qui va permettre de gérer l'ensemble des forces de l'environnement en respectant les lois physiques connues.

De plus, *ODE* offre un moteur de détection de collision. On va pouvoir l'utiliser pour créer les forces de contacts entre les différents objets du monde.

3.1 Éléments d'*ODE*

- Body :
  Éléments de base dans *ODE* , ces "body" sont les objets dans le monde physique d'*ODE* . En effet, c'est à ces objets que l'on va appliquer des forces et qui vont servir à déterminer les relations physiques entre eux.
- Geom :
  Ces objets sont les formes géométriques qui vont servir au moteur de détection de collision. A chaque "body" créé on associe une forme géométrique qui va le représenter dans l'espace de collision.
- Joint :
  Ce sont les objets qui vont servir à relier les "body" entre eux. Ils offrent de nombreuses possibilités entre les joints actifs, passifs, motorisés, bloqués ou à différents degrés de liberté.

3.2 Physique

Outre le prise en compte des lois universelles de la dynamique, *ODE* est intéressante car elle propose plusieurs fonctionalitées comme la gestion des contacts. Pour cela elle simule des forces de frictions et des contraintes fonctionnelles comme la gestion des joints et des forces qui lui sont appliqué.

Mais *ODE* est obligé de faire appel à des approximations dans les calculs, pour des raisons de gains de temps de calculs, c'est pour cela que l'on doit définir des constantes de contraintes(CFM) et de réduction d'erreur(ERP).

- Forces de fiction : Ces forces sont basées sur le modèle de Coulomb :

$$|f_T| <= mu * |f_N|$$
$$fm = mu * |fN|$$

[1] http ://opende.sourceforge.net

f_N : force normale
f_T : force tangentielle
f_m : force maximale tangentielle
mu : coefficient de friction.

ODE en utilise une approximation pour des raisons d'efficacité.
mu ne représente pas le coefficient de friction mais la force tangentielle maximale.

$$f_m = mu$$

Cette force est donc indépendante de la force normale ce qui ne respecte pas les lois physiques mais permet une bonne approximation et un gain de calcul.

– Constraint force mixing (CFM) : Constante décrivant le comportement des contraintes au sein d'*ODE* . En effet, certaines contraintes peuvent être plus ou moins absolu pour nous : elles sont appelées *hard* ou $soft$. C'est donc la constante CFM fixé à l'initialisation d'*ODE* qui assure ou non le respect des contraintes.

$$\begin{array}{lcl} J*v & = & c \\ force & = & J^T*\lambda \\ J*v & = & c+CFM*\lambda \\ (JM^{-1}J^T+CFM/h)\lambda & = & c/h \end{array}$$

v : vitesse du corps
J : matrice jacobienne représentant les degrés de liberté d'un joint
c : vecteur
h : step size
CFM : matrice diagonale de contraintes

– Error reduction parameter (ERP) : Complémentaire de la constante CFM celle-ci assure la réduction d'erreur au niveau des joints. Elle autorise ou non la violation des contraintes des joints suivant l'état de la simulation.

$$ERP = h*k_p/(hk_p+k_d)$$

h :step size
k_p : constante d'élasticité
k_d : constante d'atténuation

3.3 Collision

ODE offre aussi une gestion des collisions. Pour cela on définit un espace de collision *space*, puis on y insère des objets géométriques correspondants aux objets physiques de la simulation.

ODE propose des primitives de base définies : les sphères, les cylindres, les "cappedcylinders", les "boxes", les plans et quelques autres. On peut aussi définir sa propre forme géométrique en composant ces primitives. L'espace de collision est lui aussi un *Geom* car on peut utiliser des espaces de collision imbriqués.

```
dSphereClass        Sphere
dBoxClass           Boîte
dCCylinderClass     ''Capped'' cylindre
dCylinderClass      Cylindre
dPlaneClass         Plan
dGeomTransformClass Forme composée
dRayClass           Rayon
dTriMeshClass       Triangle
dSimpleSpaceClass   Simple space
dHashSpaceClass     Hash table based space
```

A chaque étape de la simulation *ODE* va contrôler d'éventuels contacts entre les objets *geom*. On lance une routine qui va déterminer des paires de formes $(o1, o2)$ susceptibles de rentrer en collision : cette fonction recherche toutes les formes géométriques de l'espace *space* ayant éventuellement un contact entre elles.

```
dSpaceCollide (space,0,&nearCallback);
```

space : espace de collision.
nearCallback : fonction utilisateur appelée en cas de contact supposé.

Pour chaque paire $(o1, o2)$ trouvée par *dSpaceCollide* la fonction utilisateur *nearCallback* est appelée :

```
static void nearCallback (void *data, dGeomID o1, dGeomID o2)
```

Cette fonction sert à préciser les contacts au niveau des deux formes géométriques $o1$ et $o2$. Pour cela, on fait un appel à *dCollide* dans *nearCallback* qui renvoie le nombre et la position de chaque contact entre $o1$ et $o2$:

```
n = dCollide (dGeomID o1, dGeomID o2, int flags,
              dContactGeom *contact, int skip);
```

flags : nombre maximum de contacts entre *o*1 et *o*2.
contact : structure définie en cas de contact.
skip : décalage dans le tableau *contact* pour trouver le prochain élément.

dCollide renvoie donc *n* le nombre de contacts entre *o*1 et *o*2 et stocke leurs propriétés dans le tableau *contact*.

Ce tableau est un tableau de structure *dContact* :

```
struct dContactGeom {
  dVector3 pos;       // contact position
  dVector3 normal;    // normal vector
  dReal depth;        // penetration depth
  dGeomID g1,g2;      // the colliding geoms
};

struct dContact {
  dSurfaceParameters surface;
  dContactGeom geom;
  dVector3 fdir1;
  dVector3 fdir2;
};
```

surface est une structure qui définit les propriétés du contact.

geom est donc définie par un appel à dCollide.

*fdir*1 est la première direction dans laquelle la force de frottement est appliquée.

*fdir*2 est la deuxième direction de frottement normale à *fdir*1 et à *geom.normal*.

La fonction utilisateur *nearCallback* sert à définir les paramètres de *surface*. Nous avons alors plusieurs paramètres à définir :

surface.mode : ce flag permet de préciser les options appliquées au contact.

- *dContactMu2 :* Utilise deux forces de frictions différentes *mu* et *mu*2.
- *dContactFDir1 :* Fdir1 est définie par l'utilisateur.
- *dContactBounce :* Surface rebondissante définie par *bounce*.
- *dContactSoftERP :* ERP est définie par l'utilisateur.
- *dContactSoftCFM :* CFM est définie par l'utilisateur.
- *dContactMotion1 :* La surface bouge indépendamment des "body" dans la direction Fdir1.

- *dContactMotion2 :* Idem mais dans la direction Fdir2.
- *dContactSlip1 :* Force de glissement dans la direction Fdir1.
- *dContactSlip2 :* Idem mais dans la direction Fdir2.
- *dContactApprox1_1 :* Utilisation de la pyramide de friction dans la direction Fdir1 au lieu de l'approximation faite sur *mu*.
- *dContactApprox1_2 :* Idem mais dans la direction Fdir2.
- *dContactApprox1 :* Utilisation des deux précédentes méthodes.

mu/mu2 : définissent les forces de frictions dans les directions Fdir1 et Fdir2. (0 : pas de frottements)

bounce : Élasticité de la surface (0 :dur ; 1 :très élastique)

soft_erp/soft_cfm : contrôle l'approximation d'erreur(0 :pas de correction, 1 :correction de toutes les erreurs de joints)/(0 :contraintes forte 1 :contraintes faibles)

motion1/motion2 : vitesse de la surface dans les directions Fdir1 et Fdir2.

slip1/slip2 : coefficient de glissement pour les forces de friction mu et mu2.

4 Implémentation

Nous allons donc nous attacher à implémenter un robot salamandre capable d'évoluer dans un monde virtuel soumis au même loi physique que le notre. La simulation physique de ce projet se voulant la plus précise possible, nous allons utiliser les unités universelles. De plus cela évite des conversions qui pourraient s'avérer source d'erreurs lors du passage au vrai robot :

- Time : Seconde (s)
- Masse : Kilogramme (kg)
- Longueur : Mètre (m)

Le système de coordonnées utilisé est un système à trois dimensions x,y,z avec z qui est l'axe orienté verticalement vers le haut. Les axes x et y définissent alors un plan horizontal.

4.1 Globale

Structure générale du code :

1. Initialisation d'*ODE*
2. Initialisation de l'environnement
3. Initialisation du robot
4. Boucle de simulation :
   - Définition des vitesses angulaires des joints :
     setMotors()
   - Détection de contacts :
     dSpaceCollide (...) création des joints de contacts.
   - Calcul des contraintes :
     dWorldStep (...) ;
   - Destruction des joints de contacts
     dJointGroupEmpty (contactgroup) ;
   - Çalcul de la vitesse
   - Écriture des données
   - Affichage
5. Destruction ressources

4.2 Initialisation d'*ODE*

Pour commencer la simulation il nous faut tout d'abord définir plusieurs éléments au niveau d'*ODE* . En effet, il est nécessaire de créer le monde physique

world pour pouvoir ensuite créer les objets formant le corps de la salamandre, puis l'espace de collision *space* qui va nous servir à détecter les contacts entre les objets.

```
world = dWorldCreate();
space = dHashSpaceCreate (0);
```

On doit également définir certaines contraintes internes d'*ODE* :

- **Error Reduction Parameter(ERP)** est la tolérance à l'erreur dans la gestion des forces. Pour un certain réalisme nous voulons que la tolérance soit faible et donc que le paramètre de réduction d'erreur soit élevé.
- **Constant Force Mixing (CFM)** est la force de contrainte des joints qui traduit la possibilité pour le joint de se dessouder. Dans notre simulation nous voulons que les joints soient indestructibles, nous fixons donc la contrainte à une très faible valeur.

```
dWorldSetERP(world,0.7);
dWorldSetCFM(world, 1e-5);
```

Une fois *ODE* initialisé, nous pouvons commencer à créer notre environnement.

4.3 Initialisation de l'environnement

Celui-ci est assez simple, il se compose d'un sol et d'une force de gravité.

Le sol va être créé en insérant un plan dans l'espace de collision *space*. Nous pouvons incliner l'angle de ce plan pour qu'il forme un angle *ground_angle* avec le plan d'équation $z = 0$. Cet angle peut être défini par l'utilisateur dans le fichier de configuration.

Quant à la gravité,on la simule grâce à une fonction d'*ODE* . On la fixe à $9.81 m.s^{-2}$ comme celle de la terre, et on l'oriente négativement suivant l'axe des z vertical.

```
ground = dCreatePlane (space,sin(ground_angle),0,cos(ground_angle),0);
dWorldSetGravity (world,0,0,-9.81);
```

4.4 Structure du robot

A présent, l'environnement d'*ODE* étant initialisé nous pouvons librement créer notre robot.

Le robot salamandre est composé de n_{seg} segments parallélépipédiques liés deux à deux par un joint. Les fixations que nous utilisons sont des joints à un degré de liberté suivant l'axe vertical des *z*.

A deux de ces segments $limbs_{attach1}$ et $limbs_{attach1}$ sont attachés les deux paires de pattes .

Pour les tests nous utiliserons 12 segments (soit $n_{seg} = 12$) et nous attacherons les pattes au troisième et au huitième segments (soit $limbs_{attach1} = 3$ et $limbs_{attach1} = 8$). Ces paramètres peuvent être modifié aisément dans le fichier de configuration s'il s'avère qu'ils ne sont pas optimaux.

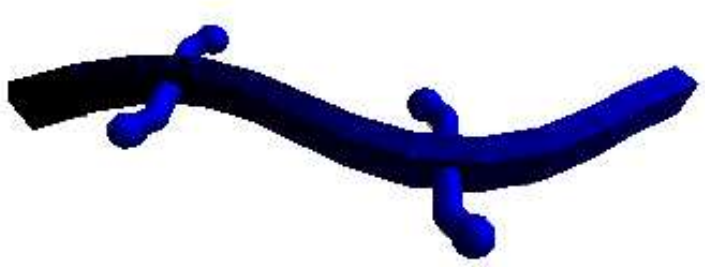

FIG. 2 – salamandre

4.4.1 Segment du Corps

Pour chaque segment on va créer deux choses au niveau d'*ODE* : un "body" pour la gestion physique des forces dans l'environnement *world* et un "Geom" qui est la forme géométrique correspondante au "body" dans l'espace de collision *space*.
On détermine alors pour chaque segment une taille, une masse, une position et des données propres à chaque partie du corps .

```
body[i]  = dBodyCreate (world);
dMassSetBox ( &m,body_dens ,body_size[0] ,body_size[1],body_size[2]);
dBodySetMass (body[i],&m);
```

```
dBodySetPosition (body[i], x , y , z);
dBodySetData(body[i] ,(void*)BODY_PART);
```

On crée alors la forme géométrique correspondante :

```
b_box[i] = dCreateBox  (space,body_size[0] ,body_size[1],body_size[2]);
```

et on affecte cette forme au “body”.

```
dGeomSetBody (b_box[i],body[i]);
```

Le corps est composé de *n* segments qui sont reliés les uns aux autres par des joints. Ceux-ci sont des *Hinge* dont l’axe de rotation est fixé selon l’axe *z*. Ce joint nous offre un seul degré de liberté. On peut alors éventuellement choisir d’avoir plus de souplesse le long du corps dans le plan vertical et remplacer un *Hinge* par un *Ball* qui offre trois degrés de liberté. On bloque alors un des axes pour conserver le réalisme.

```
dJointAttach        (b_joint[i], body[i],body[i+1]);
dJointSetHingeAnchor(b_joint[i], x , y , z);
dJointSetHingeAxis  (b_joint[i], 0 , 0 , 1);
dJointSetHingeParam (b_joint[i], dParamLoStop,-body_max_ang);
dJointSetHingeParam (b_joint[i], dParamHiStop, body_max_ang);
```

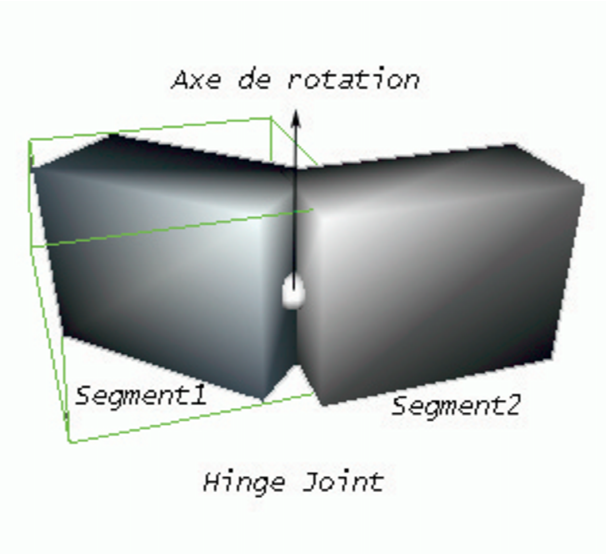

FIG. 3 – Joint inter-segment

4.4.2 Les Pattes

Les pattes de notre robot sont particulières car elles ne correspondent à aucun mouvement naturel. Elles sont entre la roue et la patte d’insecte. Notre patte bouge

en effectuant une rotation suivant l'axe de son premier segment. Ce qui lui procure un point d'appui au sol pendant une partie de la période de rotation.

Celles-ci sont donc entièrement paramétrables pour les besoins de la modélisation. En effet l'un des intérêts de cette simulation est l'étude de l'influence de la géométrie des pattes sur la vitesse du robot et son comportement face à des obstacles.

Chaque patte est composée des trois *CappedCylinder* (cylindre auquel on ajoute une demi-sphère à chaque extrémité) de longueur $lenght_i$ et de rayon $radius_i$. On peut aussi paramétrer l'angle de la patte avec le segment du corps $angle_1$et l'angle des deux premiers cylindres $angle_2$.

Chaque articulation de la patte est bloquée sauf la dernière qui peut être passive. Pour chaque articulation on va donc créer un joint *Hinge* mais nous fixons les paramètres d'angle min/max à 0.

```
legs[i].joint[j] = dJointCreateHinge (world,0);
dJointAttach
dJointSetHingeParam(legs[i].joint[j], dParamLoStop,0.0);
dJointSetHingeParam(legs[i].joint[j], dParamHiStop,0.0);
```

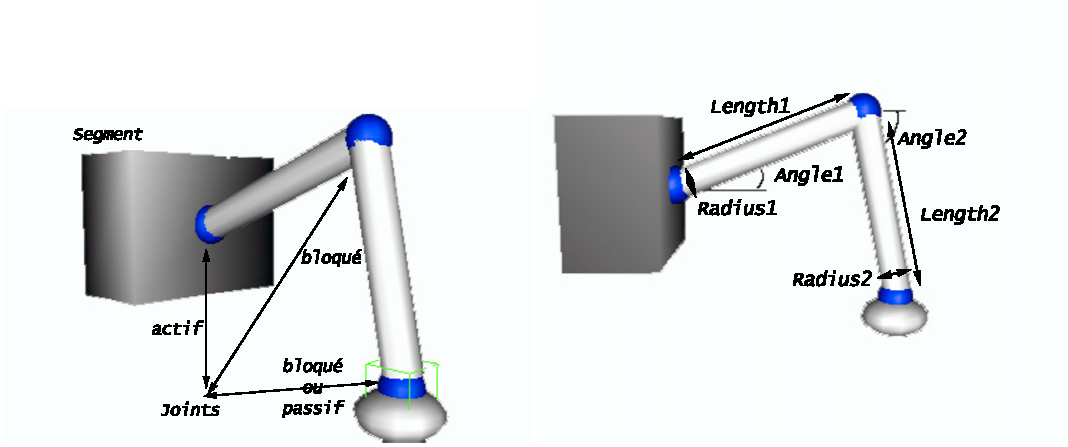

FIG. 4 – Pattes

4.5 Contrôleurs

Pour l'implémentation des contrôleurs nous utilisons des *PID*. Avec *ODE* nous ne pouvons pas définir directement et précisément l'angle de chaque joint

et nous ne pouvons travailler qu'avec les vitesses angulaires et les forces maximales. La force maximale est la force maxi applicable à un joint pour pouvoir obtenir la vitesse angulaire voulue. On fixe donc celle-ci à une constante assez grande (5 N par ex.).
$\theta =$ angle réel
$\tilde{\theta} =$ angle désiré

$$\delta\theta = A * (\tilde{\theta} - \theta)$$

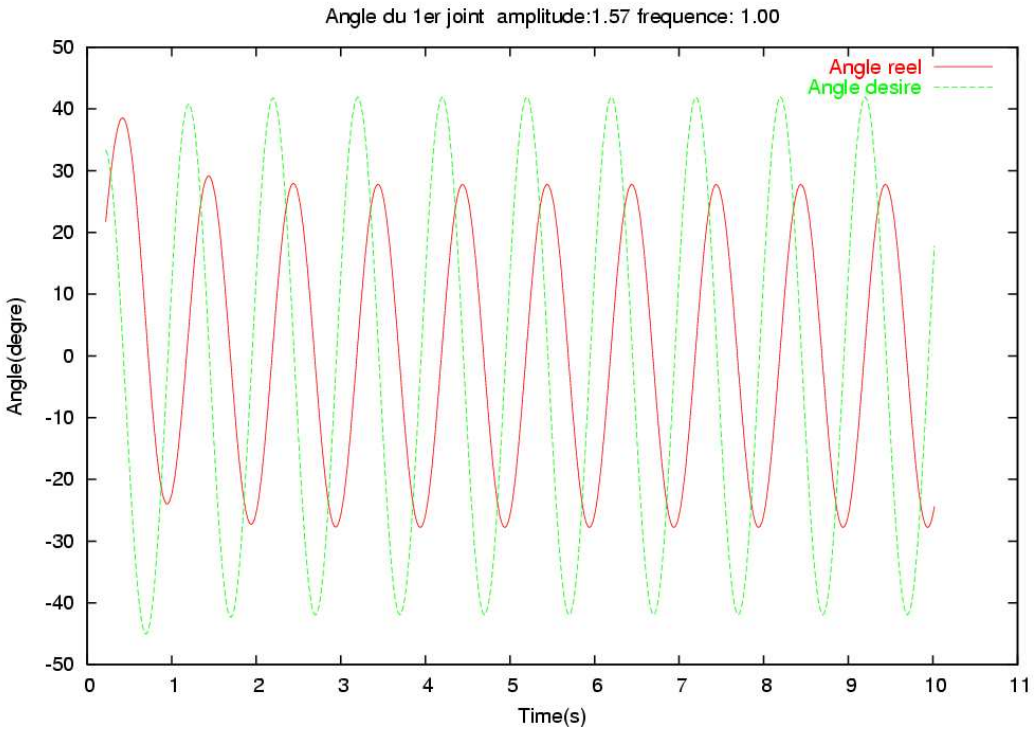

FIG. 5 – Décalage du PID

4.6 Fonctions de contrôle

Pour ce simulateur, nous allons utiliser deux modes : la marche et la locomotion serpentine. Chaque mode nécessite alors des fonctions de contrôles distinctes.

4.6.1 Locomotion serpentine

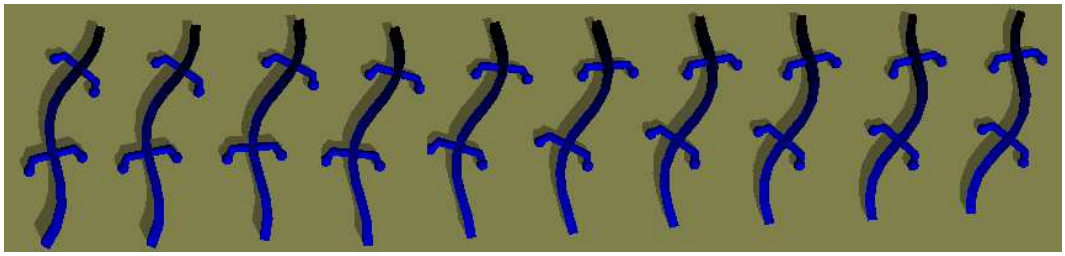

FIG. 6 – Locomotion Serpentine

Pour ce mode de déplacement, le corps du robot ondule suivant une fonction sinusoïdale et les pattes restent inactives. On va donc fixer l'angle des pattes de telle sorte qu'elles ne gênent pas l'ondulation du corps.

On fixe la valeur de l'angle alors à $PI/2$.

$$\theta_{legs}(j) = \pi/2$$

j : numéro du joint $\in [0 \dots 3]$

On veut pouvoir tester différentes formes de fonctions pour le contrôle du corps nous allons donc paramétrer une fonction sinus de telle manière à pouvoir la modifier facilement et à couvrir un gamme de mouvement assez grande. Nous aurons donc trois paramètres essentiels : l'amplitude, la fréquence et la longueur d'onde.

$$\theta_{body}(i) = A * sin(2\pi\nu t + 2\pi i/(n-1) * \phi_s)$$

i : numéro du joint $\in [0 \dots n-1]$
A : amplitude
ν : fréquence
t : temps
n : nombre de joints
ϕ_s : Longueur d'onde

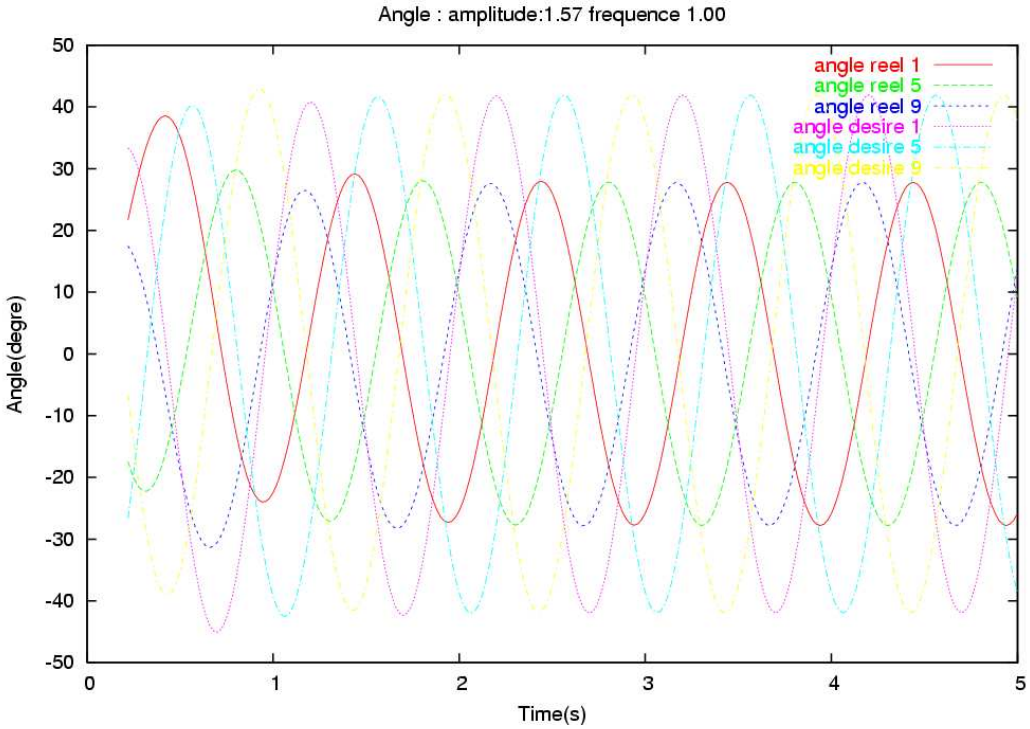

FIG. 7 – Valeurs réels et désirés des joints du corps

4.6.2 Marche

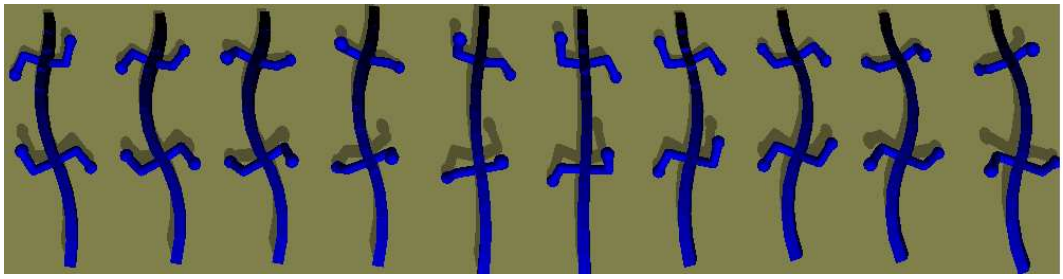

FIG. 8 – Marche

Pour la marche nous allons nous inspirer de la marche de la salamandre. Pour cela on sépare le corps en trois parties limitées par les segments d'attaches. Chaque partie est alors en opposition de phase avec ses voisines à tout moment t. Il ne nous reste donc plus que deux paramètres à tester pour cette fonction : l'amplitude et la fréquence.

$$\theta_{body}(i) = \begin{cases} A sin(2\pi\nu t + 2\pi) & \text{pour} \quad i >= segment\_attach1 \\ & \text{et} \quad i < segment\_attach2 \\ A sin(2\pi\nu t) & \text{sinon} \end{cases}$$

i : numéro du joint $\in [0 \ldots n-1]$
A : amplitude
ν : fréquence
t : temps

En revanche le contrôle de la rotation des pattes se complique un peu. Dans un premier temps cette rotation est linéaire de période $2\pi/\nu$. Mais nous allons aussi étudier des fonctions de rotation non linéaire($B \neq 0$). De plus nous introduisons un paramètre supplémentaire : le décalage de phase ϕ entre la rotation des pattes et l'ondulation du corps, pour voir la meilleure synchronisation entre les pattes et le corps.

$$\theta_{legs}(j) = 2\pi(t+\phi)\nu + B sin(2\pi(t+\phi))$$

j : numéro du joint $\in [0 \ldots 3]$
B : forme de la fonction. Linéaire si B=0 sinon deux phase une lente et une rapide $B \in [0 \ldots 1]$.
ϕ : phase entre la fonction de contrôle du corps et des pattes.

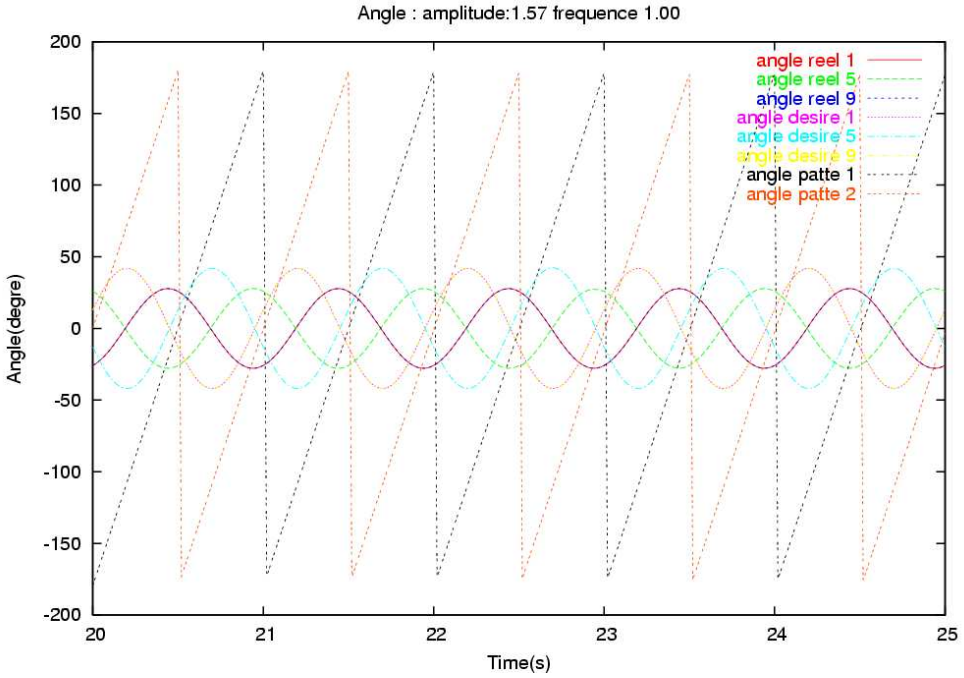

FIG. 9 – Valeurs réels et désirés des joints du corps et des pattes

4.7 Configuration

Nous utilisons un fichier de configuration pour fixer les paramètres du robot. Ce fichier définit tous les paramètres structurels du modèle. On peut modifier les dimensions des segments du corps, la taille et la structure des pattes et l'environnement comme le sol et ses obstacles.

De plus pour modifier aisément les paramètres des fonctions de contrôle on peut les passer en ligne de commande.

4.7.1 Fichier de configuration

Ce fichier par default est "model.conf" situé dans le repertoire de lancement du programme. Ou il peut-être manuellement défini en passant comme paramètre à l'exécutable $-c$ "nom_de_fichier".
Ce fichier peux être modifié à volonté, pour décrire et modifier le robot sans que l'on ai besoin de recompiler le programme. En effet toutes ces données sont lues à l'exécution et le modèle est alors modifier en conséquence.

Paramètres modifiables :

body_num : nombre de segments pour le corps du robot

legs_attach1 : premier segment d'attache des pattes

legs_attach2 : deuxième segment d'attache des pattes

body_dens : densité du corps 1000.0

legs_dens : densité des pattes

body_size1 : longueur d'un segment

body_size2 : largeur d'un segment

body_size3 : hauteur d'un segment

ground_num : nombre de cube d'obstacle

ground_size : dimension des cotés du cube

ground_angle : angle du sol avec le plan $z = 0$

ground_rand : hauteur maximale d'un cube

ground_draw : affichage ou non des obstacles

$length_i$: longueur du i^{ieme} segment de la patte ($i \in [1 \dots 3]$).

$radius_i$: rayon du i^{ieme} segment de la patte ($i \in [1 \dots 3]$).

legs_angle1 : angle des pattes avec le corps par rapport à y

legs_angle2 : angle de la première articulation par rapport à y

4.7.2 Ligne de commande :

Pour pouvoir effectuer de nombreux tests sur les fonctions de contrôles nous devons pouvoir les fixer facilement. C'est pourquoi nous pouvons passer différentes options au programme :

```
usage: main.exe [-g] [-c name] [-a amp] [-p phase] [-f freq] [-t profil] [-s long]
 g : graphic simulation

 c name   : Nom du fichier de configuration du robot
 a amp    : Amplitude de la fonction sinus du corps
 f freq   : Fréquence de la fonction sinus du corps
 p phase  : Phase entre le corps et les pattes
 s long   : Longueur d'onde de la fonction sinus du corps
 t profil : Type de fonction du contrôle des pattes
```

5 Expérimentation

La phase d'expérimentation va nous permettre de déterminer plusieurs points. Tout d'abord nous allons pouvoir tester différentes fonctions de contrôle du corps et des pattes. Ensuite, nous nous intéresserons à la structure des pattes, dimensions, orientations...

Notre critère de validation d'un jeu de paramètres sera la vitesse du robot.

Il faut tout de même préciser que ces valeurs de vitesses ne sont que relatives. En effet, dans cette simulation nous ne connaissons pas les forces de frottements au niveau du sol en contact avec le robot. Nous allons donc fixer arbitrairement ces forces de frictions (modélisées par *mu* et *mu*2). Les résultats que l'on obtiendra seront quand même exploitables si l'on garde à l'esprit qu'il ne faut pas s'attacher aux valeurs mais à leur répartition.

Pour obtenir un déplacement lors de la locomotion serpentine il faut définir des forces de frictions asymétriques telles que $mu \leq mu2$.

5.1 Optimisation des contrôleurs

5.1.1 Locomotion serpentine

La locomotion serpentine est un mode de déplacement largement utilisé chez les reptiles notamment les serpents. Il s'agit d'une ondulation du corps de l'animal qui provoque un déplacement grâce à des appuis asymétriques au niveau de la zone de contact entre le sol et le corps. Le mouvement étant périodique et ondulatoire, nous allons utiliser une fonction de contrôle de type "sinus".

Nous allons nous intéresser à la fonction de contrôle des articulations du corps, enfin plus précisément à ses paramètres. Dans cette expérimentation nous allons utiliser une fonction sinusoïdale de type qui permet d'obtenir un mouvement très similaire à celui d'un serpent :

$$Angle_{corps}(i) = A * sin(2\pi t * \nu + 2\pi i/(n-1) * \phi_s)$$

n :nombre de joints
i :numéro du joint $\in [0 \dots n-1]$
ν :fréquence
A :amplitude
ϕ_s:Longueur d'onde

Nous allons alors nous attacher à montrer l'influence de l'amplitude, de la fréquence et de la longueur d'onde sur la vitesse.
Pour cette série de tests nous allons faire varier l'amplitude (degré), la fréquence

(Hz) et la longueur d'onde dans des intervalles respectifs $[30\ldots90]$, $[1\ldots2]$ et $[1\ldots2]$.

Nous obtenons les résultats suivants :

Influence de la fréquence

a	30	40	50	60	70	80	90
1.00	0.37	0.49	0.53	0.54	0.54	0.52	0.51
1.50	0.53	0.70	0.75	0.75	0.75	0.74	0.73
2.00	0.64	0.83	0.90	0.90	0.91	0.91	0.90

TAB. 1 – Vitesse du robot en fonction de l'amplitude et de la fréquence

On voit que plus la fréquence est grande plus la vitesse augmente mais ce n'est pas non plus proportionnel à cause de l'influence des forces de frictions au niveau du corps.

Il est intéressant de regarder l'influence de l'amplitude. En effet celle-ci ne fait pas augmenter la vitesse de façon monotone. On voit que l'on a un maximum pour une amplitude comprise entre 50 et 80.

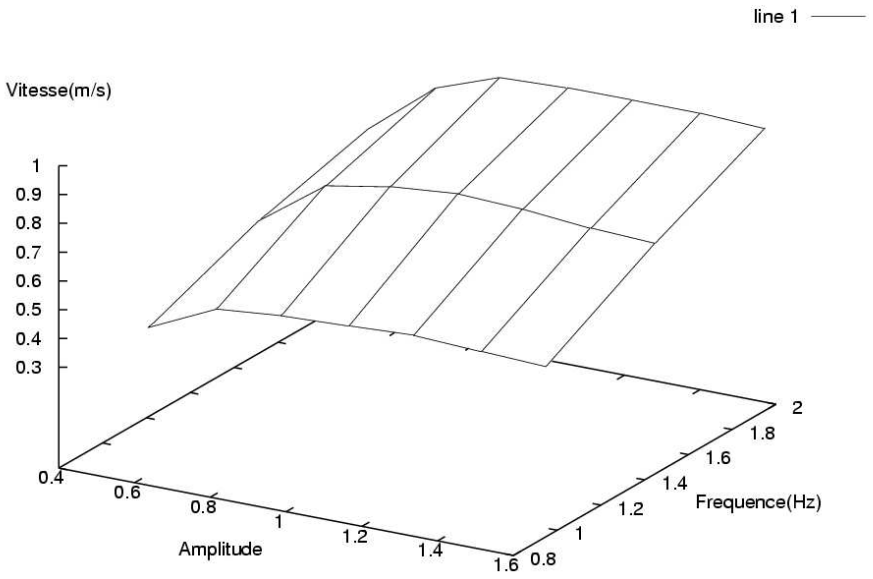

FIG. 10 – Vitesse en fonction de la fréquence et de l'amplitude.

Influence de la longueur d'onde

On fixe la fréquence à 1.

a	30	40	50	60	70	80	90
1.00	0.37	0.49	0.53	0.54	0.54	0.52	0.51
1.50	0.53	0.70	0.75	0.75	0.75	0.74	0.73
2.00	0.64	0.83	0.90	0.90	0.91	0.91	0.90

TAB. 2 – Vitesse du robot en fonction de l'amplitude et de la longueur d'onde

On observe toujours une influence quasi-proportionnelle de la fréquence sur la vitesse, en revanche l'augmentation de la longueur d'onde semble réduire sensiblement la vitesse.

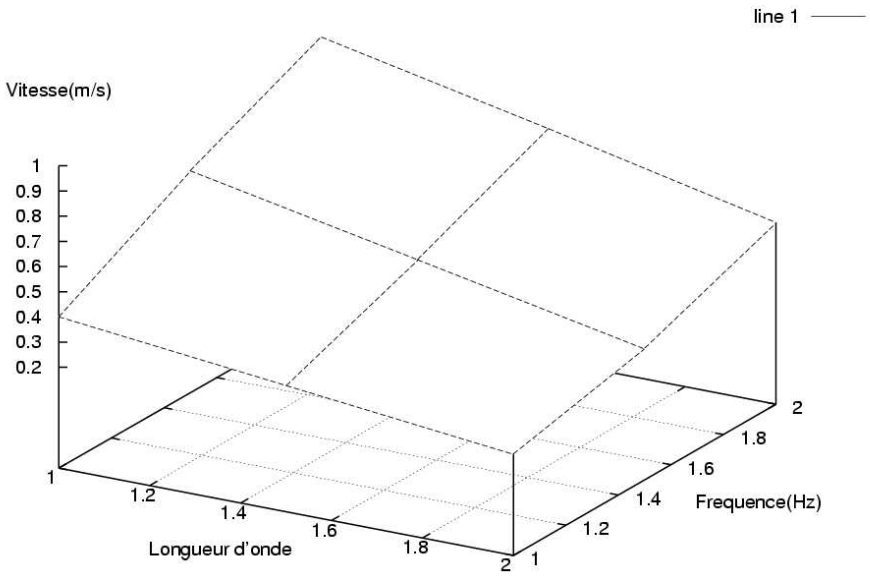

FIG. 11 – Vitesse en fonction de la fréquence et de la longueur d'onde.

5.1.2 Marche

Dans cette série de tests, nous allons nous intéresser à la marche. Pour la marche beaucoup plus de facteurs rentrent en considération. En plus de ceux déjà testés : l'amplitude A, la fréquence ν et la longueur d'onde ϕ_s , nous testerons l'influence de la phase ϕ et la forme de la fonction de contrôle des pattes sur la rapidité du robot.

On effectue le test avec une fonction de contrôle des joints du corps identique sauf au niveau de la phase de chaque segment. On inverse la phase entre les segments d'attache des pattes.

$$Angle_{joint}(i) = \begin{cases} A * sin(2\pi t * \nu + 2\pi) & \text{pour} \quad i >= segment\_attach1 \\ & \text{et} \quad i < segment\_attach2 \\ A * sin(2\pi t * \nu) & \text{sinon} \end{cases}$$

Pour le contrôle des pattes,nous allons évaluer plusieurs types de fonction.

– Type 1 : Fonction linéaire

$$Angle_{pattes}(i) = 2\pi(t + \phi) * \nu$$

– Type 2 : Fonction non-linéaire

$$Angle_{pattes}(i) = 2\pi(t + \phi) * \nu + A * sin(2\pi(t + \phi))$$

Pour tous ces types de fonction nous chercherons à optimiser la vitesse en fonction de la phase.

Évaluation de l'influence de la fréquence

L'augmentation de la fréquence montre de fortes instabilités au niveau du comportement de la salamandre. En effet les impulsions des pattes au niveau du sol étant effectué plus rapidement le robot a tendance à décoller et à se retourner. On effectuera donc la suite des tests avec une fréquence égale à 1.

Évaluation de l'influence de la phase et de l'amplitude

D'après ces résultats nous pouvons constater que la vitesse a un maximum pour une phase proche de 0.6 période par rapport à la tête du robot.

De plus on peut vérifier que la vitesse en fonction de ϕ est bien périodique.

En revanche, on observe pas tout à fait le même comportement que pour la locomotion serpentine par rapport à l'amplitude. Celle-ci fait augmenter la vitesse en grandissant, mais en regardant de plus près les simulations nous constatons que celles-ci sont plus instables si l'amplitude est trop élevé. Le robot lorsqu'il se contracte met en jeux des forces trop grandes qui ont tendance à faire basculer le robot.

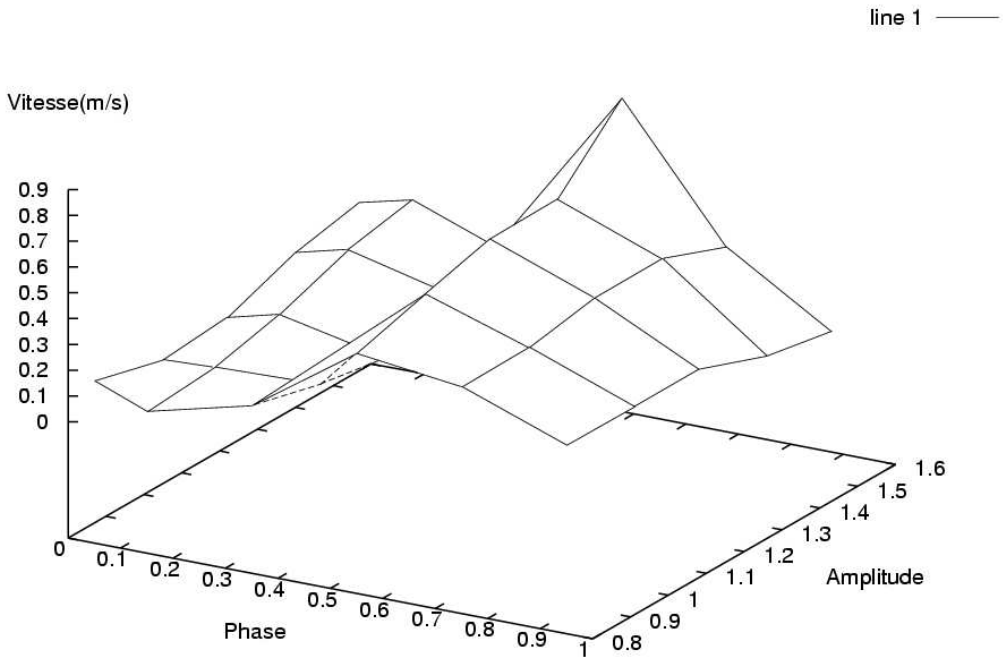

FIG. 12 – Vitesse de marche en fonction de la phase et de l'amplitude.

Évaluation de l'influence de la forme de la fonction de rotation des pattes

Pour cette série de tests nous allons utiliser une fonction de rotation des pattes non-linéaire de type 2 vue plus haut. Nous prendrons comme paramètre $A = 0.9$ et nous ferons varier l'amplitude entre 40 et 80 degrés(limite de stabilité) et la phase entre 0 et 1.

Nous voyons à présent que la période de rotation ce divise en 2 phases : l'une lente et l'autre plus rapide. Nous cherchons à créer un mouvement qui va être rapide pendant que la pattes n'est pas en contact avec le sol et lent sinon, afin de favoriser un maximum les points d'appuis au sol. Ces variations entraînent des impulsions qui devraient favorisé la vitesse.

Au vue de ces résultats on remarque que la variation de l'amplitude se fait moins sentir avec ce type de marche. De plus, si on augmente l'amplitude au-dessus de 80 la marche devient alors plus instable et il y a des risques de retournement pour le robot, dus à des impulsions au niveau des pattes plus rapides.

On remarque toujours un maximum de vitesse pour une phase au alentour

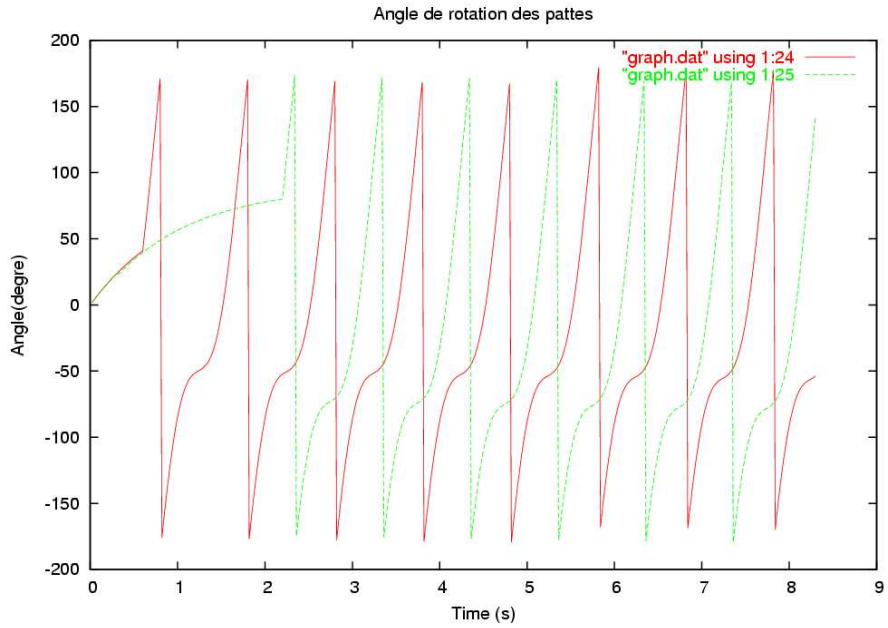

FIG. 13 – Forme de la fonction de rotation.

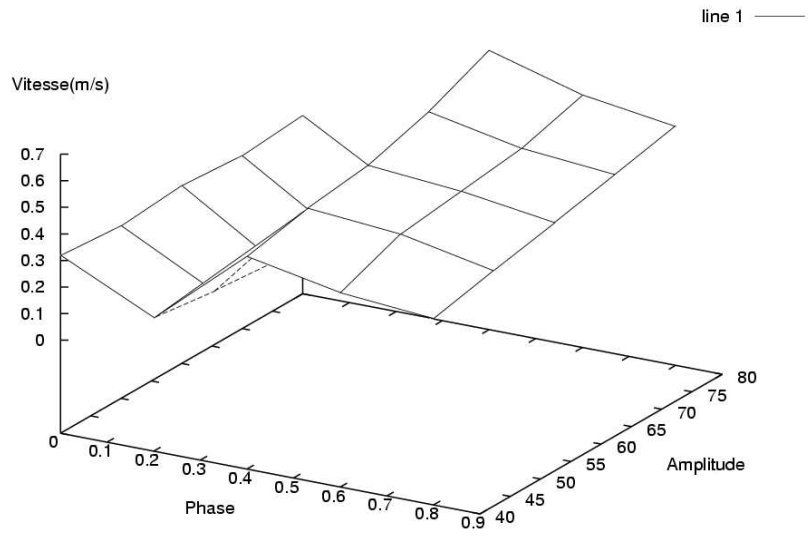

FIG. 14 – Vitesse en mode de marche non-linéaire.

de 0.5 mais cette fois-c plutôt vers 0.4. Différence s'expliquant grâce à la forme de la fonction de rotation. En effet dans ce cas là la vitesse de rotation est plus grande lorsque la patte ne touche pas le sol, elle génère alors un petit décalage de phase avec le corps.

5.2 Géométrie des pattes

Dans un deuxième temps nous allons nous intéresser à la structure des pattes de la salamandre. En effet on peut penser que celle-ci risque d'influencer la vitesse de marche de notre robot. Nous allons étudier l'influence de ces différents paramètres sur la vitesse :

*length*1 : la longueur du premier segment qui influe sur la distance du point d'appui au corps que l'on fait varier entre 0.03 et 0.01 mètres.

*length*2 : la longueur du deuxième segment qui influe sur la hauteur du corps par rapport au sol que l'on fait varier entre 0.03 et 0.12 mètres.

*Angle*1 : l'angle du premier segment avec le corps. Cette angle modifie l'axe de rotation de la patte. En effet, celle-ci doit tourner dans l'axe de ce segment sinon le genou toucherait le sol lors de la rotation. On testera ici deux positions l'une ou l'angle est nul c-à-d que la patte est orthogonale au segment d'attache et l'autre de types insectoïdes avec un angle de 30 degrés.

*Angle*2 : l'angle du "genou". On testera ici deux valeurs : 90 et 70 degrés pour que le point d'appui soit plu éloigné du corps.

*Radius*1/*Radius*2 : le diamètre des cylindres. A priori la largeur des cylindres n'influent pas sur le comportement du robot en marche. Mais évidement plus les pattes sont larges, plus le poids est grand et le couple augmente au niveau du joint.

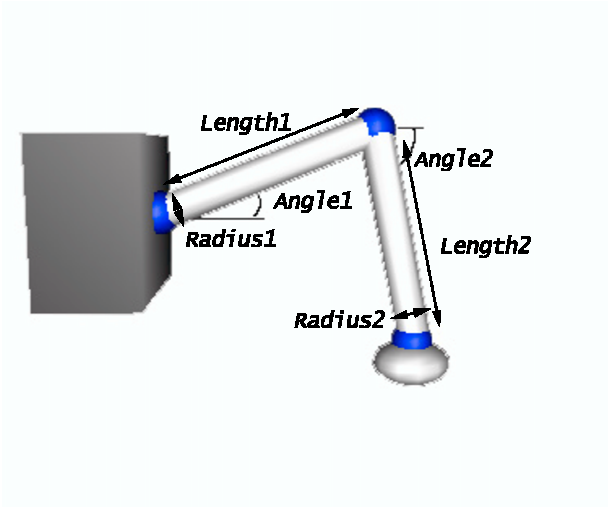

FIG. 15 – Pattes

Résultats :
D'après ces graphiques nous pouvons constater plusieurs choses :

Il semble qu'il ait une longueur optimal pour la deuxième partie de la patte se situant entre 0.06 et 0.07 m. Cette longueur correspond à la hauteur du robot. On peut penser que trop haut le robot est moins stable et par conséquent moins rapide.

La position des pattes avec un angle de 30 degrés confère une bonne stabilité car elle abaisse le centre de gravité de la salamandre.

Des pattes très large ($l1 \geq 0.1$ m) mais peu haute (($l1 \simeq 0.04$ m) offre aussi un centre de gravité très bas et semble en plus augmenter la vitesse sensiblement car les points d'appui sont très éloigné du corps et joue pleinement leur rôle de pivot lorsque le corps se contracte.

Il faut faire attention que les pattes touchent bien le sol. En effet si l'on a des pattes de formes insectoïdes il faut que le deuxième segment soit grand sinon les pattes tournent dans le vide.

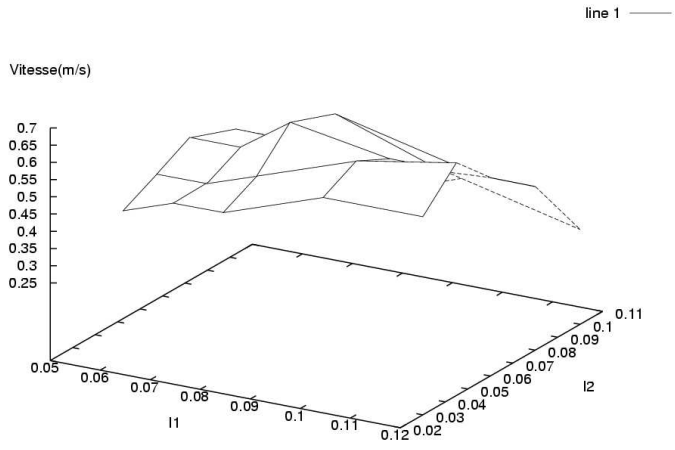

FIG. 16 – Vitesse en fonction des longueurs $l1$ et $l2$ avec $Angle_1 = 0$

De tous ces tests sur la structure géométrique de la patte, on peut déduire plusieurs choses.

Tout d'abord il semble que l'un des facteurs non négligeable d'une locomotion rapide et régulière soit une certaine stabilité du mouvement dû à un centre de gravité bas. Il y a alors plusieurs manières d'appréhender le problème. On peut faire des pattes peu haute (de l'ordre de 0.03-0.04m) mais longue pour compenser le manque d'amplitude du mouvement de celle-ci. Ou alors choisir de grandes pattes insectoïdes dont chaque segment tourne autour de $0.1m$ avec $Angle_1 \simeq -40$

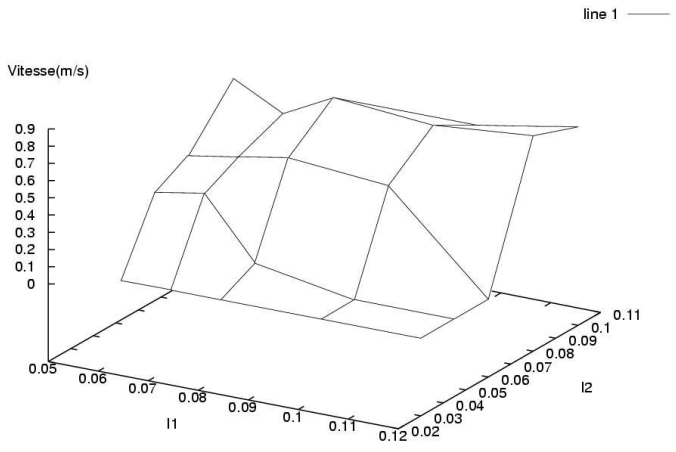

FIG. 17 – Vitesse en fonction des longueurs $l1$ et $l2$ avec $Angle_1 = -30$

et $Angle_1 \simeq 50$. On a alors le corps très près du sol et cela permet de stabiliser la marche.

On remarquera aussi qu'il faut faire attention à des pattes trop longue qui risqueraient de s'emmêler quand il y a une forte amplitude au niveau du corps.

5.3 Environnement

Pour cette série de tests nous allons voir dans quelle mesure notre robot est capable de franchir des obstacles. Pour cela on dispose d'un damier de cube de différentes hauteurs. On peut modifier le nombre, la taille et la hauteur de ces obstacles. On va alors s'intéresser à la structure des pattes et voir quelle géométrie permet les franchissements les plus aisés.

Malheureusement ces tests ne peuvent s'effectuer quand mode graphique pour vérifier le comportement du robot. Les résultats ne seront donc qu'une appréciation de l'utilisateur sur les différentes configurations testées.

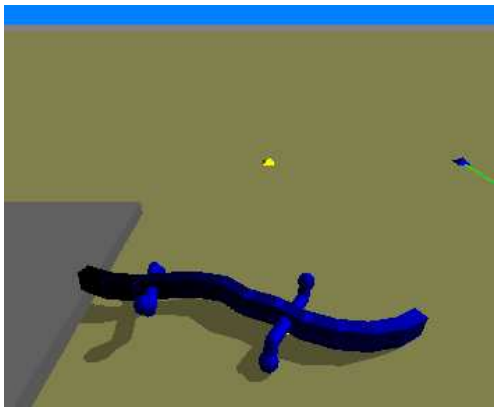

FIG. 18 – Franchissement.

La position des pattes est telle que la première partie du corps du robot a passer sur l'obstacle est la tête. En effet celle-ci se trouve bien en avant par rapport à l'extrémité des pattes. Cela implique que l'aisance avec laquelle le robot pourra franchir un obstacle dépend essentiellement de la géométrie du corps.

Il est clair que pour de petites pattes le franchissement des obstacles va être très dur voir impossible. En effet, pour l'instant, le corps n'a qu'un seul degré de liberté selon l'axe des *z*. ce qui ne lui permet pas d'augmenter la hauteur de sa tête par rapport au sol. On aura donc des problèmes pour nos pattes larges que l'on a vu dans la partie précédente.

Les résultats montre qu'il n'y a vraiment que la hauteur des pattes qui comptent dans le succès du passage. On peut donc considérer que plus la deuxième partie de la patte est longue $length2$ plus le passage sur l'obstacle sera facile.

Mais ces résultats montrent aussi qu'il faudra sûrement articuler le corps lui-même et ajouté un degré de liberté à au moins un joints pour permettre l'élévation de la tête au-dessus de l'obstacle.

6 Conclusion

Pour ce projet, j'ai développé une simulation de robot inspiré de la salamandre. Ce simulateur devrait permettre les tests de différents paramètres encore non-déterminés au niveau de la conception du vrai robot. Celui-ci permet de fixer la géométrie des pattes, les fonctions de contrôles de celles-ci et du corps mais aussi de contrôler l'influence de contraintes externes, et d'en évaluer les conséquences par une appréciation tant graphique que numérique.

Cette modélisation s'avère assez exploitable. En effet, nous avons pu déterminer l'influence de certains paramètres -du robot et extérieur- et trouver des valeurs optimales comme pour la phase ou la fréquence des contrôleurs du corps. Plus précisément, pour la locomotion serpentine il s'avère que plus la fréquence augmente plus la vitesse est grande si la friction au sol est suffisante. On a pu ainsi fixer des valeurs optimales et limites à l'amplitude dans le cas de la locomotion serpentine et de la marche.

J'ai essayer de faire une recherche exhaustive sur la structure des pattes, mais en réalité n'ayant que peu de contraintes pour celle-ci, il est très dur de trouver une configuration optimale au vue du nombre de paramètres des pattes. Mais nous avons vu quand même que plus le robot est stable, favorisé par certaines catégories de pattes, plus il avance vite et régulièrement. Mais l'inconvénient de pattes basses, est une difficulté accrue à passer des obstacles si petit soit-il. Pour l'instant il faudrait donc mener des tests plus poussés pour essayer de déterminer un compromis entre stabilité et habilité à franchir les obstacles. Les pattes longues et insectoïdes offre un peu ce compromis mais perdent vite en stabilité.

C'est en ce sens qu'il faudrait réfléchir au fait d'ajouter des degrés de libertés au niveau du corps pour compenser ce manque de hauteur du robot. On pourrait aussi réfléchir à utiliser des joints plus souple au niveau des attaches dans le but de contre-balancer les forces exercées par les pattes et ainsi stabilisé le robot.

Mais il reste aussi un vaste domaine a étudier pour que cette simulation du robot salamandre soit complète, il aurait fallu s'intéresser à son comportement dans un environnement aquatique. Malheureusement cette partie du projet n'a pu se réalisé par manque de temps et d'outils adaptés pour simulé un milieu aquatique. Malgré tout, cette simulation pourrait être très intéressante dans la mesure où elle permettrait de voir le passage du monde marin à terrestre et de voir l'influence des pattes dans la stabilité du robot en flottaison.

Cette simulation je l'espère pourra être complétée et affinée pour des résultats encore plus probant et donner encore plus de renseignements pour la conception de ce futur robot.