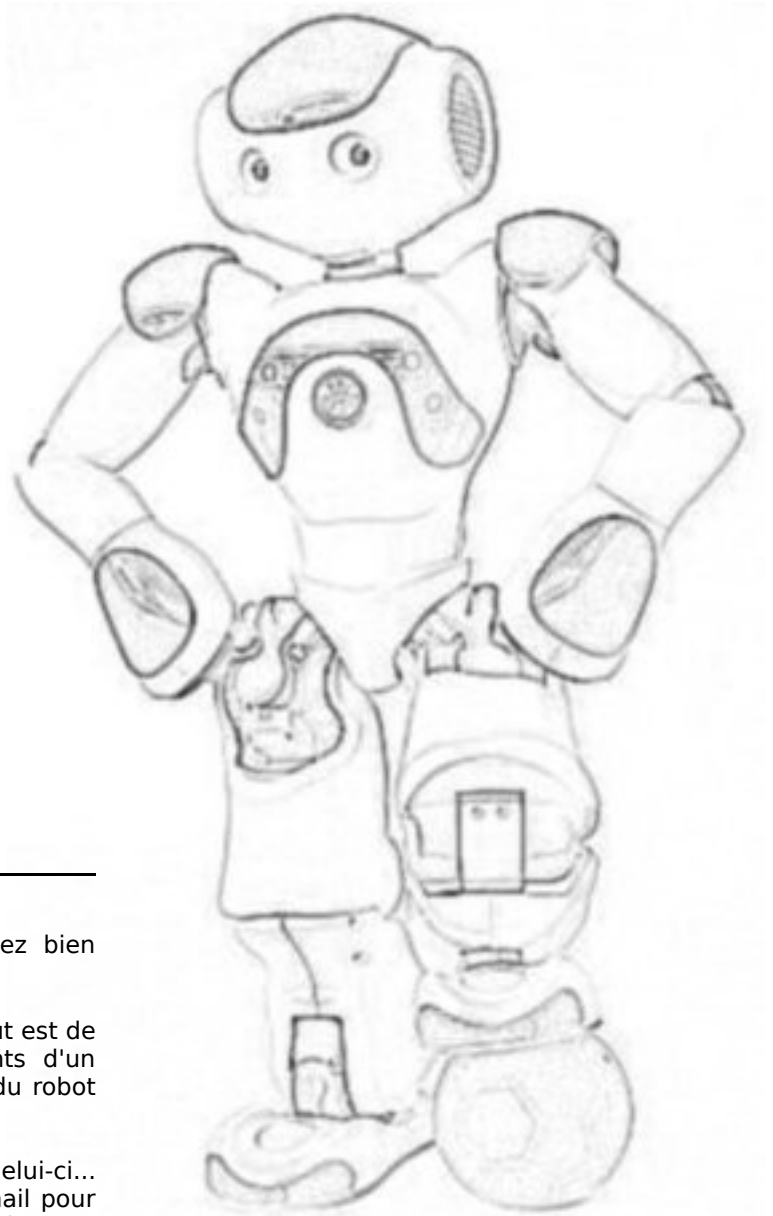


LibreBot Mag

Numéro 00.0 - 1,00 €

Sommaire

Edito_____	p1
Présentation du robot Nao_____	p2
Jouer avec un robot Nao virtuel_____	p3
Piloter le robot Nao avec URBI_____	p4-5
Programmation : flexions_____	p6
Premier programme URBI_____	p7
Mini Concours_____	p8



Edito

Bonjour, et merci de l'intérêt que vous avez bien voulu accorder à cette modeste publication.

Ceci est le numéro zéro d'un magazine dont le but est de vous apprendre à programmer les mouvements d'un robot humanoïde, et plus spécifiquement ceux du robot Nao.

Il n'est pas dit qu'il y ait d'autres numéros que celui-ci... Mais si le contenu vous intéresse, envoyer un mail pour réclamer la suite, et qui sait, il y aura peut être un numéro un :-)

En attendant, bonne lecture.

Rédaction : Bothari GANHIR
Mise en page : Bothari GANHIR
Relecture : C.J. / C.P.

Site Web : <http://librebot.free.fr>
Mail : librebot@free.fr

Présentation du robot Nao

Il est peu probable que vous ayez un robot Nao chez vous. Pour éviter que la lecture soit trop fastidieuse, nous allons faire comme ci vous me posez des questions, et moi, je vais faire comme si je connaissais les réponses ;),

A quoi ressemble le robot Nao ?

Ce robot ressemble à celui présenté sur la couverture, et la particularité qui le distingue des autres, c'est qu'il a des jambes, et qu'on peut donc lui apprendre à marcher.

Car ce robot, c'est un peu comme un bébé, pour qu'il fasse quelque chose, il faut d'abord que quelqu'un lui ait appris à le faire.

Où peut-on en acheter un ?

Pour l'instant on ne peut pas, car il n'est pas encore en vente. Seuls quelques privilégiés en ont un, et ce n'est pas la version finale.

Combien coûte-t-il ?

On ne sait pas encore précisément, mais il coûte cher, sans doute trop cher pour que vous puissiez en acheter un sans l'aide de vos parents.

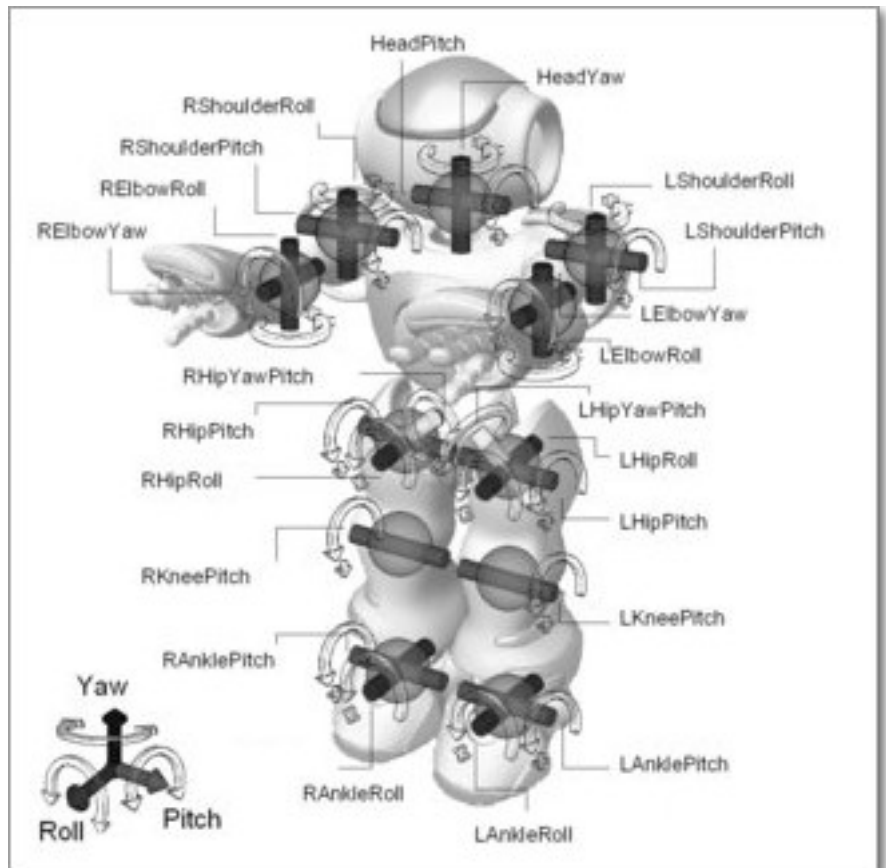


Fig. 1: Axe et orientation des articulations du robot Nao

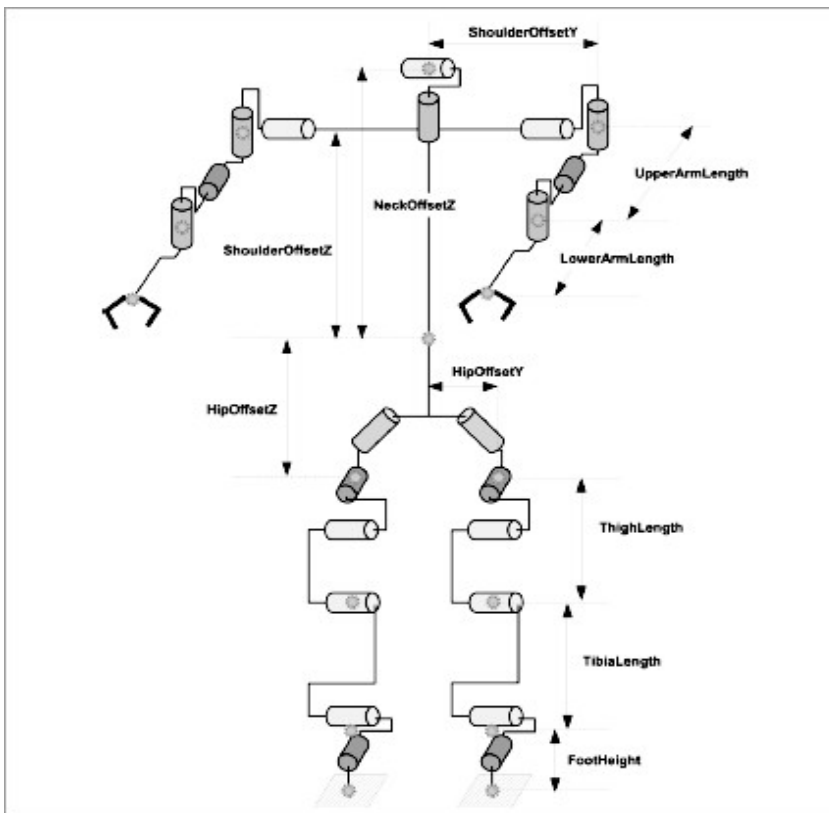


Fig. 2: Détail de la hiérarchie des axes

Que sait-il faire ?

En ce moment, les concepteurs de Nao lui apprennent à faire plein de choses, mais nous, nous allons faire comme si il ne savait rien faire, et on va tout lui apprendre nous même !

A quoi peut-on jouer avec lui ?

A plein de choses, mais dans ce magazine, on va juste jouer à lui apprendre à se déplacer, comme on apprend à un bébé à marcher.

Est-ce que le magazine pourra m'apprendre à lui faire faire d'autres choses ?

Oui :) Le but est de lui apprendre à jouer au foot, pour qu'il gagne la prochaine Robocup ! Mais pour ça, il faut commencer par lui apprendre à marcher, puis lui apprendre à comprendre ce qu'il voit.

Et comment on... ?

STOP ! Fini pour les questions, il est temps de passer aux choses sérieuses.

Jouer avec un robot Nao virtuel

Comme nous n'avons pas de robot Nao, nous allons utiliser le logiciel « Webots » qui propose une version virtuelle de Nao.

Téléchargement et installation

Nous allons utiliser la version d'évaluation gratuite de Webots. Il vous suffit donc de chercher « webots download » sur google, et vous devriez arriver sur la page permettant de le télécharger. Il y a des versions pour Linux, Windows et Mac. Évidemment, comme j'utilise Linux (distribution Ubuntu), il faudra peut être adapter mes explications à votre propre système.

Une fois que vous avez chargé le logiciel Webots, il suffit de l'installer. Sous linux, il faut extraire le contenu de l'archive « tar.bz2 » où l'on veut. Cela créé un répertoire « webots », dans lequel il y a le programme webots que l'on souhaite lancer.

Lancement de Webots

Ça y est, vous avez installé webots ? Dans ce cas, il est temps de l'utiliser. Lancer l'application et regarder ce qu'il se passe :)

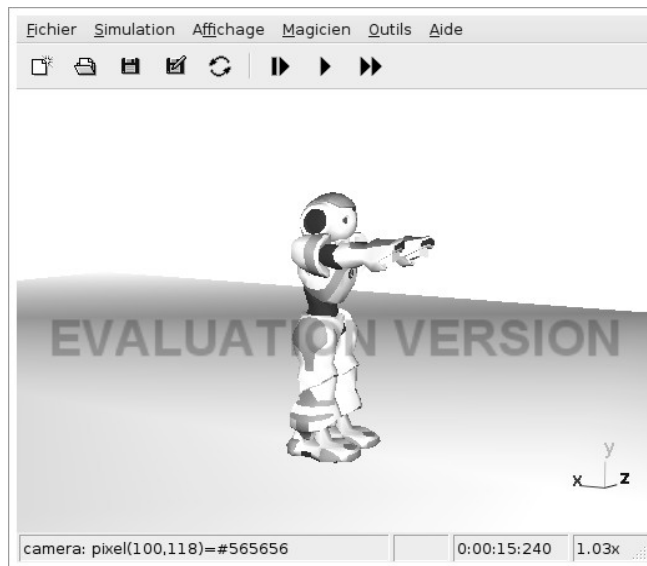
Webots permet d'ouvrir des fichiers de type « *.wbt ». Par défaut, je crois que « webots » ouvre le fichier « Nao2.wbt » qui se trouve dans le répertoire « webots / projects / contests / nao_robotcup / worlds ». Avec le menu de webots, vous pouvez essayer d'ouvrir d'autres fichiers « wbt » (il y en a plein dans les sous-répertoires de « webots/projects »).

Le logiciel webots permet de simuler un monde en 3D. Une fois que vous avez ouvert un fichier « wbt », vous verrez les différents éléments qui ont été mis dans ce monde, et vous pourrez lancer ou suspendre la simulation.

En général, il y a dans chaque monde un robot qui bouge tout seul, car ses actions sont contrôlées par un programme.

Contrôler soi même le robot Nao

Pour vous apprendre à contrôler votre Nao, j'ai préparé un fichier « librebot_nao.wbt » aussi simple que possible. Vous pouvez le trouver dans l'archive qui se trouve à cette adresse :



http://librebot.free.fr/mag/librebot_webots_00.zip

Chargez cette archive et décompressez la. Cela devrait créer un répertoire « librebot_webots ». Ouvrez avec webots le fichier « worlds/librebot_nao.wbt » de cette archive.

Il s'agit d'un monde avec juste un robot Nao. Ce robot n'est contrôlé par aucun programme, ce qui va nous permettre de lui faire faire nous même ce que l'on veut pour mieux comprendre comment s'y prendre.

Lancer la Simulation

Une fois que vous aurez ouvert le monde librebot_nao.wbt, il faut lancer la simulation (icône de lecture « Run » en forme de triangle). L'icône de lecture devient carré, et en bas à droite de la fenêtre, le temps de simulation se met à défilier.

Vous pouvez maintenant double cliquer sur le robot Nao. Cela devrait ouvrir une fenêtre avec des curseurs. Chacun de ces curseurs contrôle un axe de rotation d'une articulation du robot. En bougeant les curseurs, vous faites bouger le robot !

Utilisation de Webots

Si vous cliquez dans la fenêtre de simulation et déplacez la souris en maintenant le clic, vous pouvez déplacer votre point de vue, et ainsi voir le robot sous un autre angle. Cliquez gauche, cliquez droit, molette, chacun des boutons de la souris à une action particulière. Ainsi la molette

permet de zoomer plus ou moins.

Vous pouvez cliquer sur l'icône « Revenir » pour retrouver le monde et le robot tels qu'ils étaient en début de simulation.

Il y a dans la fenêtre de simulation une petite zone rectangulaire. Elle affiche ce que voit le robot. Si vous bougez la tête du robot, le robot regardera ailleurs, et le contenu de la fenêtre sera modifié en conséquence.

La suite ?

Vous allez vite vous rendre compte que la fenêtre de curseur est pratique pour se familiariser de façon interactive avec les articulations du robot, mais que contrôler chaque axe un par un à la main est laborieux et ne permet pas de faire grand chose au robot, à part le faire tomber...

Pour aller plus loin, nous allons faire de la programmation !

Piloter le robot Nao avec le langage URBI

Pour faire faire des choses plus évoluées au robot, nous allons utiliser le langage de programmation URBI.

Normalement, vous n'avez rien de plus à installer, car le langage URBI est intégré à la version de Webots que vous avez installé.

Grâce à URBI, nous allons pouvoir nous connecter au robot, et le piloter en tapant des commandes.

Lancer URBI

Ouvrez le monde « librebot_nao.wbt », et lancez la simulation webots (bouton « Run »). Une fois que la simulation est en cours, ouvrez une fenêtre de terminal (fenêtre DOS sous Windows), et lancez la commande suivante :

```
telnet localhost 54001
```

Cette commande permet de se connecter par telnet au robot de la simulation Webots.

Vous pouvez aussi vous connecter au robot de la simulation *webots* en utilisant le programme « urbiRemote » (aussi appelé urbilab). Une version gratuite de ce programme peut être téléchargée à l'adresse : <http://www.gostai.com/urbilab.html>

Pour installer le logiciel, il suffit sous Linux de charger le programme, de décompresser l'archive et de lancer le programme « UrbiRemotePro.sh » contenu dans le répertoire « urbiRemote » qui a été créé.

Une fois *UrbiRemote* lancé, il faut cliquer sur l'icône de connexion, mettre « localhost » comme serveur, et « 54001 » pour le port.

Si *webots* ou la simulation n'a pas été lancé, vous obtiendrez une erreur disant :

```
telnet: Unable to connect to
remote host: Connection
refused
```

Ou avec UrbiRemote :

```
Connexion Error
Can't connect to `localhost`
on `54001`.
Error code: -1
```

Cela arrive aussi après 5 minutes de simulation (cf la version d'évaluation de Webots ne permet que 5 minutes de simulation, après quoi il faut la relancer depuis le début).

Si la connexion se passe bien, vous devriez voir s'afficher des lignes précisant la version de URBI.

Commandes URBI

Une fois que vous êtes connecté au robot, vous pouvez taper des commandes dans cette même fenêtre de terminal (ou dans la ligne de saisie en bas de la fenêtre de *UrbiRemote*), vos commandes seront alors envoyées au robot qui les exécutera.

Avec telnet, pour quitter URBI, il faut taper le caractère '^]' (le ^ signifie qu'il faut maintenir la touche CTRL appuyée, puis taper le caractère indiqué, en l'occurrence le crochet fermé, qui nécessite sur mon clavier d'utiliser aussi la touche 'alt gr'). Cela quitte URBI, et vous ramène au mode telnet, qui se quitte en tapant « quit » puis la touche entrée.

Avant toute chose, une première règle à ne pas oublier : quand vous taper une commande URBI, il faut à la fin taper un point-virgule « ; », sinon la commande ne sera pas exécutée.

Hello World !

La coutume veut que la première chose que l'on fasse quand on apprend un nouveau langage est de faire afficher « Bonjour le monde ! » (en anglais « Hello world ! »).

```
echo("Hello World !");
[00108636] *** Hello World !
```

Cela va juste afficher la phrase dans votre fenêtre de commande URBI. Évidemment, cela n'a aucun impact sur le robot Nao, donc ce n'est pas très intéressant. Mais au moins, la coutume est respectée !)

Commande d'un moteur

Dans URBI, chaque élément du robot a un nom. Ainsi, le moteur de l'articulation du cou qui permet de faire tourner la tête du robot à gauche ou à droite s'appelle 'HeadYaw' (attention à bien respecter les majuscules et minuscules).

Pour faire tourner la tête du robot, il suffit d'indiquer quelle valeur d'angle le moteur doit atteindre. Pour cela, il suffit de taper dans la fenêtre URBI la commande suivante (attention à ne pas oublier le point-virgule à la fin) :

```
HeadYaw.val = 30;
```

Normalement, après avoir tapé puis validé cette ligne en appuyant sur la touche entrée du clavier, vous devriez voir la tête du robot bouger. Comme la valeur est positive, la tête a du tourner vers la gauche. Essayer une valeur négative, et elle devrait tourner vers la droite :

```
HeadYaw.val = -50;
```

La valeur que l'on met correspond à un angle en degré. Essayons de tordre le cou du robot :

```
HeadYaw.val = -180;
```

On voit qu'il n'y a pas de risque de tordre le cou du robot. Si on demande un angle trop important, le mouvement va commencer, puis s'arrêter une fois atteinte la limite de rotation du cou (en l'occurrence 60°). Donc toutes les valeurs supérieures à 60 degrés (ou inférieures à -60°) feront tourner la tête du robot à 60 (ou -60) degrés seulement.

Lecture de l'angle d'une articulation

De la même façon qu'on peut indiquer à un moteur quel angle il doit prendre, on peut demander au robot quel angle a un moteur donné. Double cliquer sur le robot pour ouvrir la fenêtre des curseurs, et faite bouger la tête du robot en déplaçant le curseur HeadYaw.

Maintenant, vous pouvez faire afficher l'angle du cou en tapant dans la fenêtre URBI la commande :

```
echo(HeadYaw.val);
[00099999] 19.99999
```

On peut afficher un message un peu plus explicite de cette façon :

```
echo("angle=" + HeadYaw.val);
[00099999] angle=19.99999
```

Mouvements lents

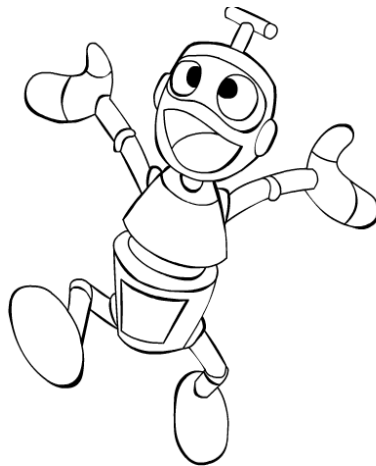
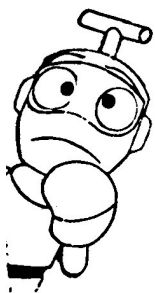
Les commandes précédentes entraînent un mouvement aussi rapide que possible.

Si on veut, on peut spécifier la durée que doit prendre le mouvement, ce qui permet d'obtenir des mouvements plus lents.

Pour cela il faut taper la valeur à atteindre (la consigne), suivi de «time:» et la durée que doit prendre le mouvement :

```
HeadYaw.val = -60 time:10s;
```

Avec ce type de commande, dès que la tête a atteint l'angle voulu, le mouvement s'arrête.



Mouvements continus

Avec URBI, on peut aussi définir des commandes qui tournent en continu.

Essayez ceci (attention, pour les commandes qui tournent en boucle, il faut mettre une virgule à la fin de la commande au lieu d'un point-virgule, sinon on ne pourra plus taper d'autres commandes) :

```
HeadYaw.val = 0 sin:5s
ampli:40,
```

Ceci demande au robot de bouger la tête de façon sinusoïdale autour de la position centrale 0, et avec une amplitude max de chaque côté de 40°.

Les étiquettes

Le problème de ce type de commandes en continu, c'est qu'elles ne s'arrêtent jamais ...

Heureusement, on peut donner une étiquette à nos commandes, puis faire un stop sur une étiquette donnée.

Si on reprend le même exemple (relancer la simulation pour qu'il n'y ait plus de mouvement), on peut lancer la commande avec l'étiquette « test ».

```
test: HeadYaw.val = 0 sin:5s
ampli:40,
```

Du coup, dès qu'on en a marre de voir la tête du robot osciller de gauche à droite, on peut taper la commande « stop » comme ci-dessous, ce qui va arrêter le

mouvement associé à l'étiquette spécifiée :

```
stop test;
```

Normalement, le mouvement de tête du robot devrait s'arrêter.

Voilà, il ne vous reste plus qu'à essayer tout cela avec d'autres articulations pour vous familiariser avec URBI et le robot Nao.

Programme

Comme taper des commandes est assez laborieux, on peut les écrire dans un fichier et les sauver. C'est tout simplement ce type de séquence de commandes qui constitue un programme.

Au lancement de la simulation, le robot exécute automatiquement les commandes contenues dans le fichier « nao.u » (jusqu'à maintenant il était vide, donc le robot ne faisait rien).

Ouvrez le fichier « nao.u », et taper les commandes suivantes :

```
# Mouvement continu de la tête
HeadYaw.val = 0 sin:10s ampli:60,
HeadPitch.val = 0 sin:500ms ampli:20,

# Mouvement continu des bras
RShoulderPitch.val = 0 sin:2s ampli:120,
LShoulderPitch.val = 0 cos:2s ampli:120,

RShoulderRoll.val = -40 sin:3.5s ampli:55,
LShoulderRoll.val = 40 cos:3.5s ampli:55,

RElbowRoll.val = 45 sin:1.5s ampli:45,
LElbowRoll.val = -45 cos:1.5s ampli:45,

RElbowYaw.val = 0 sin:2.5s ampli:90,
LElbowYaw.val = 0 cos:2.5s ampli:90,
```

Sauver le fichier. Relancer la simulation. Et voilà, le robot bouge sa tête et ses bras !

On peut même dire que vous venez de programmer votre robot pour lui faire danser la tecktonik ! Pour un début c'est pas mal ;-)

Maintenant que vous savez écrire un programme, il est temps d'essayer de faire bouger le robot sur ses jambes sans qu'il ne tombe !

Et pour ça, il faut commencer par un peu de théorie :-)

Programmation : flexions

Le premier mouvement que nous allons apprendre à notre robot sera assez simple : faire des flexions. C'est à dire que l'on va le faire se plier sur ses jambes, puis se relever.

Pour cela, il faut lui faire plier les genoux. On va donc apprendre à contrôler la jambe du robot.

La Jambe

Pour pouvoir contrôler la jambe, il faut commencer par bien en comprendre la structure.

Il y a trois articulations :

- ▶ la hanche (« Hip » en anglais),
- ▶ le genou (« Knee »),
- ▶ et la cheville (« Ankle »),

Sur la figure, les articulations sont repérées par une lettre correspondant à l'initiale de leur nom anglais, à savoir : H, K et A.

La hanche est reliée au torse du robot (point O), et la cheville au pied du robot (point F, « foot »).

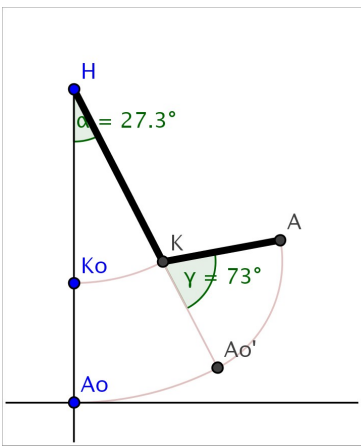


Fig. 3: Jambe du robot

Chacune des articulations peut avoir un ou plusieurs axes de rotation. Le genou a par exemple un seul axe de rotation, alors que la cheville en a 2, et la hanche 3.

Pour commencer, on va s'intéresser uniquement à un seul de ces axes de rotation : l'axe « pitch ».

Flexion max et min

Si le robot ne plie pas le genou, il sera à sa hauteur maximale. Quand le genou sera plié au maximum, le robot sera aussi bas qu'il peut l'être.

Dans la suite, on va appeler « c » la distance HK (la Cuisse), et « t » la distance KA (le Tibia).

La hauteur Y_{max} du robot est donc obtenue pour un angle α_K du genou de 0° :

$$Y_{max} = Y_{\alpha_K=0^\circ} = HK + KA = c + t$$

Mais que vaut Y_{min} ? Et plus généralement, que vaut Y pour un angle α_K donné ?

Pour le savoir il faut utiliser le théorème de Pythagore, et les formules de trigonométrie (cf calcul détaillé du cadre « Formule 1 »). On obtient :

$$HA_{\alpha_K} = \sqrt{c^2 + t^2 + 2 \cdot c \cdot t \cdot \cos(\alpha_K)}$$

Grâce à cette formule, on peut déduire l'angle du genou pour avoir une flexion donnée (c'est-à-dire une distance HA donnée) :

$$\alpha_K = \arccos\left(\frac{HA^2 - (c^2 + t^2)}{2 \cdot c \cdot t}\right)$$

Cas du Nao

La jambe du robot Nao a les caractéristiques suivantes :

$c = 120 \text{ mm}$

$t = 100 \text{ mm}$

α_K dans l'intervalle $[0^\circ; 130^\circ]$

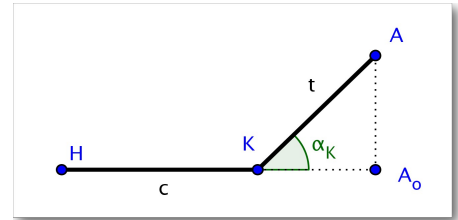
D'après le calcul que l'on vient de faire, les hauteurs de flexions sont donc :

$Y_{max} = 220 \text{ mm}$

$Y_{min} = 95 \text{ mm}$

Angle de la hanche et de la cheville

Si on ne contrôle pas comme il faut l'angle de la hanche et de la cheville, le pied du robot ne sera pas à plat sur le sol (et il aura donc de fortes chances de tomber), ou alors il ne gardera pas le torse droit (ce qui a aussi de fortes chances de le déséquilibrer).



Il faut donc calculer ces angles en fonction de la longueur HA que l'on souhaite donner à la jambe.

Dans notre cas, il s'agit en fait de trouver tous les angles du triangle HKA, sachant qu'on connaît la longueur de chacun des cotés de ce triangle.

On peut comme précédemment trouver ces angles en appliquant les formules de trigonométrie de base.

En fait, il y a plus simple et plus rapide, car les formules mathématiques de ce cas de figure sont connues.

Chercher « résolution d'un triangle » sur Google, le premier lien vous amènera à la page Wikipedia qui répertorie ce genre de formules (dont le cas où l'on cherche les angles d'un triangle dont on connaît la longueur des 3 côtés).

Il ne reste plus qu'à coder tout cela en URBI !

$$HA^2 = HA_0^2 + A A_0^2$$

$$HA_0 = c + t \cdot \cos(\alpha_K)$$

$$A A_0 = t \cdot \sin(\alpha_K)$$

$$HA^2 = c^2 + 2 \cdot c \cdot t \cdot \cos(\alpha_K) + t^2 \cdot \cos^2(\alpha_K) + t^2 \cdot \sin^2(\alpha_K)$$

$$HA^2 = c^2 + 2 \cdot c \cdot t \cdot \cos(\alpha_K) + t^2$$

$$HA = \sqrt{c^2 + t^2 + 2 \cdot c \cdot t \cdot \cos(\alpha_K)}$$

Formule 1: Calcul de HA en fonction de l'angle du genou.

Un premier vrai programme URBI

Comme on l'a vu précédemment, l'archive « librebot_webots_00.zip » que vous avez chargé et décompressé à créé un répertoire « librebot_webots ».

Le sous-répertoire « controllers » contient les programmes des différents robots, et le programme du robot Nao vert que l'on voit quand on lance la simulation, c'est celui qui s'appelle « nao ».

Il y a dans ce répertoire « librebot_webots/controllers/nao/ » deux fichiers :

- URBI.ini (fichier lu par URBI quand on lance la simulation Webots)
- nao.u (programme qui pilote notre robot).

Voici le contenu du fichier *URBI.ini* :

```
load("nao.ini");
load("nao.u");
```

On voit que le programme *URBI.ini* se contente juste de charger (et exécuter) ce qui est contenu dans les fichiers *nao.ini* et *nao.u*.

Le fichier *nao.ini* contient en fait l'initialisation de l'ensemble des éléments du robot. On ne touchera pas à ce fichier.

On va par contre pouvoir modifier le contenu du fichier *nao.u*. Ouvrez le, puis tapez le code URBI que vous voulez. Ensuite sauvez le fichier.

Si vous relancer la simulation Webots (icône de rechargement), le nouveau code sera alors exécuté.

Le code qui suit peut-être tapé dans le fichier *nao.u*.

Il fait bouger la tête et les bras du robot en continu, et pour ce qui est des jambes, il fait faire la boucle infinie suivante :

- attente d'une seconde,
- flexion des jambes du robot,
- attente d'une seconde,
- extension des jambes du robot.

La flexion correspond à une consigne de longueur des jambes (cf partie flexion). Les angles des hanches, genoux et chevilles nécessaires pour obtenir cette flexion sont actualisés à intervalle de temps régulier (toutes les 40 millisecondes).

Maintenant, c'est à vous de modifier tout cela pour voir ce que cela donne !

```
# On groupe les moteurs similaires des deux jambes
# (pour pouvoir commander les deux en une seule affectation)
group Knees { RKneePitch, LKneePitch };
group Hips { RHipPitch, LHipPitch };
group Ankles { RAnklePitch, LAnklePitch };

# Définition de variables
var cuisse = 120; #longueur de la cuisse
var tibia = 100; #longueur du tibia
var yy = 220; #consigne de longueur de jambe
var tt = 1s; #vitesse du mouvement

# Boucle infinie modifiant la consigne
loop {
    wait(1s);
    yy = 95 time:tt;
    wait(1s);
    yy = 220 time:tt;
},

# Commande des moteurs
every (40ms) {
    Knees.val = acos( (yy**2 - (cuisse**2 + tibia**2)) / (2*cuisse*tibia) ) / pi * 180;
    Hips.val = - acos( (yy**2 + cuisse**2 - tibia**2) / (2*yy*cuisse) ) / pi * 180;
    Ankles.val = - acos( (yy**2 + tibia**2 - cuisse**2) / (2*yy*tibia) ) / pi * 180;
},

# Mouvement continu de la tête
HeadYaw.val = 0 sin:10s ampli:60,
HeadPitch.val = 0 sin:500ms ampli:20,

# Mouvement continu des bras
RShoulderPitch.val = 0 sin:2s ampli:120,
LShoulderPitch.val = 0 sin:2s ampli:120,

RShoulderRoll.val = -40 sin:3.5s ampli:55,
LShoulderRoll.val = 40 cos:3.5s ampli:55,

RElbowRoll.val = 45 sin:1.5s ampli:45,
LElbowRoll.val = -45 cos:1.5s ampli:45,

RElbowYaw.val = 0 sin:2.5s ampli:90,
LElbowYaw.val = 0 cos:2.5s ampli:90,
```

Mini Concours

Si vous avez suivi toutes les étapes du magazine jusqu'ici, vous êtes peut-être arrivé à faire faire des mouvements sympas à votre robot.

Dans ce cas, vous pouvez envoyer votre code au magazine, je me chargerai de rajouter votre robot dans la simulation « Nao Dancing Battle ». Il n'y a rien à gagner, si ce n'est le plaisir d'être celui qui a

programmé les mouvements de danse les plus sympas, bref, d'avoir le meilleur robot danseur :)

La vidéo de la piste de danse avec tous les robots sera mise en ligne sur « Dailymotion ». Vous pouvez d'ailleurs jeter un coup d'œil à ce que ça donne, cherchez juste « librebot » sur le site « Dailymotion », ou tapez

directement l'adresse suivante : <http://www.dailymotion.com/LibreBot>

Voilà, si ça vous tente de rejoindre ce « danse floor », envoyez votre code URBI par mail à l'adresse librebot@free.fr.

Bonnes créations.



FIN