TD: Locomotion à pattes

1 Introduction

Ce TD a pour sujet la locomotion de robots à pattes, il a pour principale vocation de vous permettre d'expérimenter des méthodes *simples* permettant d'aboutir à différentes possibilités de locomotion pour des robots à pattes.

Quelques éléments théoriques importants pour la locomotion sont brièvements introduits avant de décrire l'environnement logiciel fourni (et principalement les différences par rapport aux précédents TP).

Dans votre rendu, veillez à inclure l'ensemble des trajectoires 'intéressantes' que vous avez pu obtenir, en commentant leurs avantages, inconvénients et ce qu'elles permettent de mettre en évidence.

2 Théorie

La locomotion à pattes dynamique étant un sujet complexe qui dépassent de loin ce qui est demandé dans ce TP, elle ne sera pas traitée ici. Si vous voulez commencer à vous renseigner sur ce domaine voici deux pointeurs que je trouve intéressants :

— `https://scaron.info/teaching/point-de-non-basculement.html`
— `https://www.youtube.com/watch?v=qT9qzwCJjAk`

2.1 Base fixe et base flottante

Dans l'ensemble des cas vu jusqu'à présent, vous avez travaillé sur des robots à **base fixe**, c'est à dire que leur point d'ancrage était considéré comme parfaitement immobile et fixe dans le repère monde.

À l'opposé, les robots à **base flottante** peuvent se déplacer dans le repère monde, cela se traduit souvent par l'ajout de 6 degrés de liberté à un robot afin de spécifier la position et l'orientation de son tronc dans le repère monde. Dans le cadre de ce TD, le principal élément à garder en tête est le fait que les mouvements planifiés pour les jambes d'un robot ne peuvent pas ignorer les effets qu'ils ont sur la base flottante. Trivialement, si un robot fléchit les jambes, c'est son tronc qui descendra et pas ses pieds qui monteront.

2.2 Centre de masse

Parmi les éléments d'une importance primordiale pour la stabilité d'un robot, il y a la position de son centre de masse, notée x_{CoM}. Si elle est relativement facile à calculer à partir du moment où vous disposez de la position et de l'orientation de la base flottante, de la géométrie du robot et de la distribution des masses, elle est bien plus difficile à mesurer étant donné qu'elle va être affectée aussi bien par les erreurs de mesures sur la position des différents joints que par l'estimation de la base flottante.

Dans le cadre de ce TD, il ne sera pas nécessaire d'obtenir la position du centre de masse, en revanche le concept sera souvent nécessaire.

2.3 Surface de sustentation

La surface de sustentation (aka. *support polygon*) est définie comme étant l'enveloppe convexe des points en contact avec le sol, voir Figure 1.

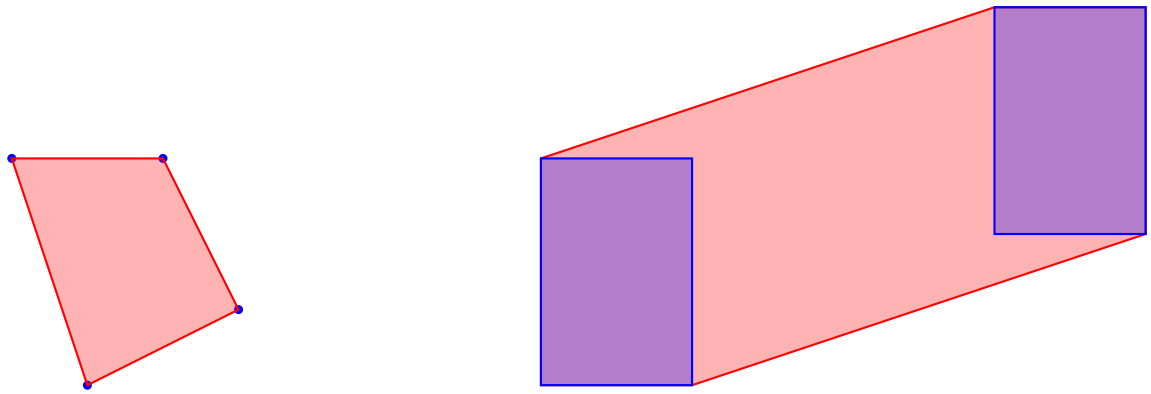

FIGURE 1 – Exemples de surfaces de sustentation (en rouge) associées aux contacts au sol (en bleu), que ceux-ci soient des points (à gauche) ou des surfaces (à droite).

2.4 Équilibre statique

Lorsqu'un robot a une vitesse et une accélération faible, il est considéré comme étant à l'équilibre statique si son centre de masse est à l'intérieur de la surface de sustentation. Cette propriété n'est ni nécessaire, ni suffisante pour le cas dynamique, mais elle a comme avantage de permettre de générer facilement des trajectoires très stables qui peuvent être *mise en pause* à tout moment sans risquer une chute du robot.

Il est important de noter que lorsqu'il n'y que deux points de contacts avec le sol, le centre de masse doit être parfaitement aligné sur la droite reliant les deux points pour être stable [1].

Dans le cadre de ce TD, cette notion jouera un rôle crucial pour comprendre la difficulté de générer des trajectoires stables.

2.5 Primitives motrices périodiques

Si le calcul de trajectoires dynamiques est un problème complexe qui n'est pas réellement abordé dans ce TD, cela n'empêche pas d'obtenir des résultats fonctionnels assez facilement [2] en définissant des trajectoires périodiques et en réglant leurs paramètres [Beh06].

C'est principalement cette idée qui sera poursuivie dans ce TD. Afin de la mettre en place, il est important de décrire certains des éléments qui la compose et des principes à suivre :

- Les primitives motrices considérées ici ont une **période** ω qui définit la durée d'un cycle. Afin de permettre de varier la période facilement, toutes les trajectoires sont définies à partir d'une **phase** $\phi \in [0, 1]$.
- Les trajectoires se basent sur une **position de repos** à laquelle sont ajoutées des modifications, par exemple : lever et avancer la jambe. On considère donc ici le fait d'additionner plusieurs trajectoires.
- Si certains mouvements sont synchronisés entre toutes les pattes [3], d'autres doivent impérativement être déphasés, comme le fait de soulever les pattes. C'est pour cette raison que chaque jambe peut avoir sa propre phase pour certaines composantes de la trajectoire.
- L'une des conditions importantes (et difficile) à assurer est le fait que les pattes qui sont en contact avec le sol ne luttent pas les unes contre les autres. Il faut donc assurer une certaine cohérence entre elles, quels que soient les paramètres.

1. Ce qui est difficilement réalisable en pratique...
2. La notion de *facilité* étant toujours relative
3. Par exemple, le transfert du poids de droite à gauche.

3 Environnement logiciel

La première chose à noter sur le code qui vous est fourni est qu'il a été grandement simplifié, en retirant par exemple les notions de différence entre espace de planification et espace des cibles pour les trajectoires. Cette différence n'est pas vraiment pertinente pour le problème actuel étant donné que pour la marche, on souhaite généralement obtenir des trajectoires dans l'espace opérationnel. Il n'y a pas vraiment de garanties de rétro-compatibilité avec le contenu des TD précédents puisqu'il n'y a pas de tests rigoureux à propos de celles-ci.

Vous pourrez notez l'apparition de trois nouveaux robots :

`quadruped_ypp` Un quadrupède dont chaque patte est composée d'une succession de joints angulaires Yaw-Pitch-Pitch.

`quadruped_rpp` Un quadrupède dont chaque patte est composée d'une succession de joints angulaires Roll-Pitch-Pitch.

`hexapod_rpp` Un quadrupède dont chaque patte est composée d'une succession de joints angulaires Roll-Pitch-Pitch.

Il est à noter que les deux derniers robots ne diffèrent que par leurs nombre de pattes étant donné qu'ils gardent la même structure interne pour les pattes.

Dans le module `model.py`, vous trouverez les classes correspondants à ces différents robots implémentées afin de vous permettre de gagner un peu de temps. Attention cependant, certains éléments devront probablement être améliorés au cours du TD. Vous noterez également l'apparition de la classe `LeggedRobot` qui permet de grandement simplifier l'écriture du code des différents robots.

Le module `trajectories.py` contient la classe `WalkEngine` qui implémente déjà une version extrêmement simpliste d'une primitive motrice convenant au robot `hexapod`.

4 Travaux pratiques

4.1 Problèmes de stabilités

Comme d'habitude, vous pouvez manipuler les joints des robots si vous ne spécifiez pas de trajectoire. Par exemple en exécutant la commande suivante.

```
./simulation.py --robot quadruped_rpp
```

Vous devriez rapidement vous apercevoir que l'équilibre du robot est précaire et qu'il est difficile de lui faire soulever une patte sans le faire tomber. Vous pouvez essayer de le manipuler dans l'espace opérationnel pour voir si vous avez plus de succès.

```
./simulation.py @scenarios/quadruped_rpp_analytical.txt
```

Cette instabilité s'explique facilement par le fait que le centre de masse est initialement sur la bordure du triangle de support obtenu une fois qu'une des pattes n'est plus en contact avec le sol.

Si vous essayez de manipuler le robot `hexapod` en utilisant des commandes similaires, vous verrez que c'est tout de suite plus facile. C'est donc naturellement sur ce la meilleure robot que se focalisera ce début de TD.

4.2 Paramètrage de la marche

Le fichier `hexapod_rpp_walk.json` permet de définir des valeurs initiales pour la primitive motrice utilisée lorsque vous exécuter le scénario associé. Exécuter la commande suivante :

```
./simulation.py @scenarios/quadruped_rpp_walk.txt
```

Initialement, le robot ne se déplacera pas du tout car la hauteur des pas et leur longueur sont toutes deux égales à 0. Ajustez les valeurs des paramètres à l'aide des curseurs et expérimentez leurs effets. Essayez de trouver une configuration qui vous semble satisfaisante et sauvegardez les valeurs dans le fichier `Json`.

4.3 Visualisation des trajectoires

Vous pouvez parfaitement vous servir de l'option `--log` et du script `plot.py` pour afficher les trajectoires cibles et mesurées. Vous remarquerez rapidement qu'au vu du nombre de degrés de libertés, afficher toutes les données sur un seul graphique ne permet plus vraiment de comprendre ce qui se passe.

Pour remédier à cette situation, vous avez deux possibilités : La première consiste à utiliser l'option `--var` qui a été ajoutée au script. Pour afficher uniquement la trajectoire de la patte avant gauche vous pourrez par exemple utiliser la commande suivante :

```
./plot.py --var left_front_x left_front_y left_front_z -- <data.csv>
```

4.4 Plus de paramètres dynamiques

Vous aurez sûrement remarqué que certains paramètres présents dans le fichier `Json` ne peuvent pas être modifiés dynamiquement durant la simulation. Il s'agit des paramètres `period` et `leg_phase_offsets`, pour les ajouter, il vous faudra modifier les 4 fonctions suivantes : `getParameters`, `setParameters`, `getParametersNames` et `getParametersLimits` dans la classe `WalkEngine`.

Vraisemblablement, votre première implémentation devrait mener à un comportement semblant erratique lors de la mise à jour du paramètre `period`. Effectivement, un léger changement de `period` peut entraîner un retour en arrière de la phase de votre marche alors que vous voulez certainement vous contenter d'accélérer ou de ralentir la vitesse d'exécution de la trajectoire. Réfléchissez à différentes manières de faire face à ce problème, en comparant les avantages et les inconvénients attendus. Implémenter la méthode qui vous semble la plus adaptée et vérifiez qu'elle donne des résultats satisfaisants.

Faîtes varier la période et observez les différents résultats obtenus. On s'attend à ce que le robot avance plus vite lorsque la période diminue, est-ce tout le temps vrai ?

4.5 Déplacement omni-directionnel

La primitive motrice fournie permet au robot d'avancer, mais elle ne lui permet pas du tout de se déplacer sur le côté où de contrôler son orientation.

Modifier la classe `WalkEngine` pour permettre au robot de se déplacer dans toutes les directions. À la fin il devra donc vous permettre de spécifier la vitesse désirée pour le robot à travers le contrôle de 3 paramètres clés :

`step_length` La distance parcourue par une patte en un pas selon l'axe x (dans le référentiel du robot)

`step_lateral` La distance parcourue par une patte en un pas selon l'axe y (dans le référentiel du robot)

`step_theta` Le changement d'orientation du robot à chaque pas.

Attention, contrôler l'orientation du robot tout en évitant que les pattes luttent les unes contres les autres est beaucoup plus dur que d'implémenter les pas latéraux. Si vous bloquez sur ce point, n'hésitez pas à passer aux suivants.

4.6 Robots quadrupèdes

Maintenant que vous êtes plus familiers avec le système de locomotion, essayez d'obtenir des résultats satisfaisants avec les robots quadrupèdes. Afin de pouvoir lever une patte sans que le robot ne perdre l'équilibre, vous pourrez par exemple ajouter une oscillation globale du corps selon l'axe y, cela permettra de déplacer le centre de masse pour qu'il reste dans la surface de sustentation lorsque le robot lève une patte. Notez que dans le fichier fourni, aucun décalage de phase n'est choisi pour les différentes pattes, il faudra donc que vous décidiez d'une stratégie à ce sujet.

À priori, vous devriez faire face à des difficultés importantes pour le robot `quadruped_ypp`. Si c'est le cas, essayez de comprendre pourquoi et décrivez le problème. Est-il possible des trouver des solutions logicielles ? Si oui, lesquelles ? Si non, pourquoi ?

5 Consignes de rendu

1. Éditez le contenu du fichier `group.csv` pour qu'il contienne tous les membres du groupe.
2. Éditez le contenu du fichier `to_pack.txt` pour qu'il contienne tous les fichiers demandés (sans oublier votre compte-rendu)
3. Créer une archive en utilisant la dernière version du script `group_work_packer.py`, présente dans l'archive du TD.
4. Vérifiez le contenu de l'archive `tar tzf XXX.tar.gz`
5. Envoyer votre archive par mail avec le sujet suivant : "(4TSD904U) Rendu TD Marche". N'oubliez pas de mettre en copie tous les membres du groupe.

Références

[Beh06] Sven Behnke. "Online trajectory generation for omnidirectional biped walking". In : *Proceedings - IEEE International Conference on Robotics and Automation* 2006.May (2006), p. 1597-1603. issn : 10504729. doi : 10.1109/ROBOT.2006.1641935.