TD : Marche quadrupède

Introduction

L'objectif de ce TD est de se servir des différents éléments vus en cours pour contrôler un robot quadrupède. En utilisant une version analytique du modèle géométrique inverse, vous allez concevoir des trajectoires pour les extrêmités des pattes permettant au robot de se déplacer.

Ce TD contient trois fichiers :

simulation.py : L'interface permettant d'exécuter le code que vous allez écrire. Elle vous permet entre autre de choisir la primitive motrice à utiliser.

control.py : Le module permettant le contrôle du robot. Plusieurs fonctionnalités sont déjà implémentées pour vous permettre de vous concentrer uniquement sur la marche. N'hésitez pas à vous inspirez du code déjà présent pour ajouter de nouvelles fonctionnalités.

homogeneous_transform.py : Une implémentation de plusieurs fonctionnalités basiques pour les transformations homogènes.

En plus du fichier `control.py`, vous devrez rendre un fichier `README` contenant vos remarques et différentes réponses aux questions posées dans le TP.

Prise en main

Vous pouvez identifier l'indice des différentes pattes du robot et vérifier que les sens de rotation des différents éléments correspondent à ce que vous attendez en exécutant simplement le programme sans argument : `./simulation.py`

Dans ce TD, vous allez expérimentez plusieurs primitive motrice différentes. Elles héritent toutes de la classe abstraite `WalkEngine` qui définit entre autre la position des pattes lorsque le robot n'est pas en train d'effectuer un pas.

Avec la commande `./simulation.py --walkEngine Fixed`, vous pouvez tester différentes valeurs pour cette position de repos. Les valeurs par défaut sont intentionellement peu adaptées, vous devez donc en choisir de meilleures. Une fois que vous avez des valeurs convenables, modifier les valeurs par défaut qui sont stockées dans le fichier `control.py`, dans le dictionnaire `defaultParameters`.

Marcher avec des splines

Afin de faire se déplacer le robot, une des possibilités est de définir la trajectoire que vont effectuer les extrémités de la patte par rapport à leurs positions initiales respectives.

Afin d'assurer une certaine stabilité, il est plus simple de bouger une seule patte à la fois et donc d'effectuer le même mouvement de manière déphasée sur chaque patte. C'est l'approche choisie dans ce TD. Comme les trajectoires se répètent, on aura aussi recours à la notion d'affixe qui normalise le temps qui s'est écoulé en fonction de la durée totale de la trajectoire. Ainsi,

l'affixe sera toujours compris dans l'intervalle $[0, 1[$. Cette partie est commune à toutes les trajectoires et est donc définie dans la classe `WalkEngine` au sein des méthodes `getAffix` et `getLegAffix`.

Vous pouvez vous familiariser avec la méthode proposée et la comprendre en lisant le code de la classe `TrajectoryWalkEngine` et en lançant la commande suivante :
`./simulation.py --walkEngine Trajectory --splinePath walk_forward.json`

Écrivez d'autres splines offrant d'autres possibilités de déplacement :

walk_backward.json : Pour que le robot se déplace dans la direction opposée (x négatif dans l'orientation initiale).

walk_left.json : Pour que le robot se déplace vers la gauche (direction y positive dans l'orientation initiale).

walk_right.json : Pour que le robot se déplace vers la droite (direction y négative dans l'orientation initiale).

Il est plus difficile d'obtenir une marche permettant au robot de tourner sur lui-même simplement en modifiant la spline. Expliquez pourquoi la solution n'est pas adaptée dans votre `README`.

Des marches paramétrables

Si l'utilisation de spline proposée dans la version précédente permet de rapidement obtenir un résultat satisfaisant, elle ne permet pas de changer facilement des paramètres de la marche lors de l'exécution, c'est un élément critique pour contrôler la taille des différents pas par exemple.

Les splines présentées mettent en évidence quelques paramètres que l'on a très souvent envie de pouvoir changer dynamiquement. La classe `ParametricWalkEngine` considère les variables suivantes :

flyingRatio : La proportion du temps passé en l'air par chaque patte.

stepHeight : La hauteur maximale théorique à laquelle monte une patte.

stepX : La taille d'un pas en mètres selon l'axe 'x' dans le repère du robot. Une valeur négative indique que le robot recule.

stepY : La taille d'un pas en mètres selon l'axe 'y' dans le repère du robot.

stepDir : Le changement de direction du robot à chaque pas en radians.

Pourquoi la valeur de 0.25 pour le *flyingRatio* pour cette variable a-t-elle une signification particulière ? Expliquez quel est l'impact d'utiliser une valeur supérieure à 0.25 dans votre `README`.

LegLifter

La première étape de la marche est de lever successivement chaque patte. Il vous suffit d'implémenter la méthode `getLegHeight` de la classe `ParametricWalkEngine` pour obtenir le comportement souhaité. Pour un affix dans $]0, \text{flyingRatio}[$, la valeur retournée doit être comprise dans $]0, \text{stepHeight}[$. Dans les autres cas la valeur retournée doit être 0. Vous pouvez tester votre code avec la commande suivante : `./simulation.py --walkEngine LegLifter`

Vérifiez que vous avez bien le résultat attendu en modifiant les variables `stepHeight` et `flyingRatio`.

CartesianWalk

En vous basant sur le contenu des différentes splines que vous avez créées précédement et sur les autres classes, Vous devriez facilement pouvoir compléter le code de la classe `CartesianWalkEngine` pour obtenir une marche permettant de se déplacer en 'x' et en 'y' simultanément avec une taille de pas contrôlable en temps réel. Cette marche n'a pas besoin de prendre en compte les ordres en rotation et pourra être testée avec :
`./simulation.py --walkEngine Cartesian`

RotationWalk

La classe `RotationWalkEngine` doit permettre au robot de tourner sur lui même, sans nécessiter qu'il soit capable d'effectuer des translations. L'une des difficultés est de s'assurer que toutes les pattes qui sont en contact avec le sol se déplacent de manière cohérente pour éviter qu'elles exercent des forces qui s'opposent. Est-ce possible ? Répondez à cette question de manière justifiée dans votre `README`. Si c'est possible expliquez comment vous faîtes pour assurer cette contrainte, dans le cas contraire, essayez de minimiser ces forces et expliquez la stratégie que vous avez utilisée.
`./simulation.py --walkEngine Rotation`

Pour aller plus loin

Cette partie est plus difficile et requiert d'avoir bien assimilé les points précédents. Elle n'est pas indispensable mais comporte plusieurs problèmes qui font appel à l'ensemble des notions vues en cours ou les approfondissent.

Marche omni-directionnelle

Implémentez la classe `OmniDirectionalWalkEngine` de manière à supporter simultanément des ordres de translation en cartésien et des ordres en rotation. Le travail que vous avez effectué pour le robot holonome à roues pourra servir de base à votre méthode.

Stabilisation statique

Pour cette partie, créez un dossier différent afin de vous assurer que vos modifications n'impactent pas vos résultats pour les autres exercices.

Vous avez sûrement pu remarquer que le robot n'est pas stable sur ses trois pattes lorsqu'il bouge la quatrième. Les conséquences sont particulièrement visibles lorsque la vitesse d'exécution du mouvement de marche est lente. Effectivement, il arrive que le centre de masse du robot sorte du polygone de sustentation défini par l'enveloppe convexe des points en contact avec le sol, ce qui fait basculer le robot.

Afin d'éviter ce problème, ajoutez un mouvement au torse afin de stabiliser le robot lorsque la trajectoire est jouée à vitesse faible.

Évaluation

Les documents seront à déposer sur Moodle dans une archive nommée : `nom_prenom.tar.gz`
L'exécution de `tar -xzf nom_prenom.tar.gz` ne devra pas afficher d'erreur.

L'archive devra contenir un dossier `nom_prenom` dans lequel vous mettrez :

— Votre fichier `control.py`

— D'autres fichiers python si votre `control.py` si besoin
— Un fichier `README` récapitulant le travail effectué et les différentes réponses aux questions posées.
— Si vous avez avancé dans la partie 'pour aller plus loin', ajoutez aussi un dossier nommé `stabilisation` contenant le code modifié.