TD: Modèle géométrique direct

1 Introduction

Ce TD sert entre autre de support pour vous familiariser avec des outils et une méthodologie de développement qui seront utilisés tout au long de cette UE. Pour bien prendre en main l'ensemble des outils, il peut être utile de suivre les tutoriels `Webots`[1]

Afin de pouvoir exécuter vos commandes il sera nécessaire d'installer en tant qu'utilisateur les bibliothèques python indispensables:

```
pip3 install -U --user scipy plotly seauval
```

L'archive fournie pour ce TD contient un monde `Webots`, un simulateur de robots open-source. Le contenu initial de l'archive devrait être le suivant:

```
|-- controllers
|   |-- motor_controller
|   |   |-- homogeneous_transform.py
|   |   |-- motor_controller.py
|   |   |-- robots.py
|   |   \-- test_homogeneous_transform.py
|   \-- supervisor
|       \-- supervisor.py
|-- plot.py
|-- protos
|   |-- AxisReferential.proto
|   |-- RobotRRR.proto
|   \-- RobotRT.proto
\-- worlds
    \-- td_introduction.wbt
```

Pour mener à bien ce td, vous allez devoir effectuer des modifications dans les trois fichiers suivants:

- `homogeneous_transform.py`: dans lequel vous implémenterez des fonctions de base pour travailler dans des coordonnées homogènes.
- `robots.py`: Ce fichier définit une classe abstraite `RobotModel`, au cours du TD, vous devrez implémenter les méthodes des classes `RTRobot` et `RRRRobot` pour visualiser correctement la position de l'outil de votre robot.
- `motor_controller.py`: Ce fichier définit la manière dont est contrôlé le robot, il sera également utilisé pour sauvegarder les données des capteurs.
- `supervisor.py`: Ce script est le chef d'orchestre de la simulation, il a accès à des informations qui ne sont pas disponibles pour les robots.

[1] Voir: `https://cyberbotics.com/doc/guide/tutorials`

1.1 Méthodologie de test

Au cours de ce TD, vous allez pouvoir expérimenter différentes manières de tester votre code. L'un des intérêts principaux est que vous réfléchissiez à quelles méthodes sont appropriées pour les différentes fonctionnalités que vous implémentez. Certaines d'entres elles sembleront sûrement peu adaptées pour ce TD, mais elles seront mises à profit au cours des prochains TD.

1. **Tests unitaires:** Un jeu de tests basé sur la bibliothèque `unittest` est déjà implémenté dans le fichier `test_homogeneous_transform.py`. Vous pouvez le lancer à l'aide de la commande `nosetests3`, ce qui devrait résulter sur 12 tests échouant tous puisque le code n'est pas encore implémenté.

   Vous pouvez parfaitement créer d'autres fichiers de test et/ou compléter celui-ci si cela vous semble adapté.

2. **Tests en simulation:** L'environnement mis à disposition vous permet de rapidement visualiser différentes configurations du robot, mais aussi de visualiser très rapidement certaines erreurs que vous pouvez commettre dans le code. Par la suite, l'environnement de simulation permettra également de tester les comportements de vos robots et de vérifier qu'ils correspondent bien à ce que vous attendez.

3. **Affichage des données:** Le fichier `plot.py` sera utilisé pour visualiser l'évolution de certaines variables au cours de l'exécution. C'est une phase particulièrement importante pour détecter des artefacts qui peuvent être imperceptibles lorsque l'on se contente d'observer visuellement le comportement d'un robot.

4. **Évaluation semi-automatisée:** Ces tests ne sont pas des éléments important de développement, ils seront uniquement utilisés dans un but d'évaluation dans le cadre de cette UE. Les avantages principaux étant la possibilité de pondérer différents tests et la possibilité de laisser l'enseignant juger de la pertinence de certains éléments difficilement automatisables (qualité du code, ...).

2 Implémentation

2.1 Prise en main de la simulation

Pour pouvoir exécuter `webots` au cremi, vous devez commencer par définir les deux variables d'environnement suivantes:

```
export WEBOTS_HOME=/usr/local/webots
export LD_LIBRARY_PATH=$WEBOTS_HOME/lib/webots:$LD_LIBRARY_PATH
```

Le plus simple étant de les ajouter dans votre fichier de configuration `.bashrc`. Ensuite, vous pouvez lancer votre première simulation en utilisant la commande suivante:

```
webots worlds/td_introduction.wbt
```

Vous pouvez choisir quel robot vous souhaitez utiliser ainsi que la durée après laquelle la simulation s'arrête automatiquement en modifiant les variables d'environnement `ROBOT_NAME` et `SIM_DURATION` avant d'exécuter votre programme.

Il est également possible de changer la cible des moteurs en modifiant les valeurs utilisées dans le fichier `motor_controller.py`.

2.2 Transformations homogènes

L'implémentation des fonctions `rot_x`, `rot_y`, `rot_z`, `translation` et `invert_transform` en utilisant la bibliothèque `numpy` et le contenu du cours ne devrait pas poser de problèmes majeurs. En cas de doute, les fichiers de tests devraient vous permettre de comprendre rapidement ce qui est attendu.

2.3 Modèles de robot et modèle géométrique direct

Dans le dossier `protos`, vous trouverez deux fichiers `proto`[2]: `RobotRRR.proto` et `RobotRT.proto`. Ces fichiers décrivent la structure des deux robots et vous pouvez les utiliser pour en déduire le modèle géométrique du robot.

À partir des fichiers `proto` décrivant les robots et en modifiant les cibles initiales vous devriez être capable d'écrire les matrices de transformation correspondant à des configurations *simples* du robot (par exemple en utilisant que des angles qui sont des multiples de $\frac{pi}{4}$.

Implémentez les méthodes `getBaseFromToolTransform` et vérifiez qu'elles donnent bien les résultats que vous attendiez.

2.3.1 Visualiser les erreurs

Outre des tests sur les valeurs, vous pouvez aussi visualiser vos matrices de transformation en dessinant des repères dans la simulation, voir Fig. 1. Pour obtenir cette visualisation, il est nécessaire que le `motor_controller` communique la pose qu'il a estimé pour l'outil au `supervisor` qui a le droit de déplacer des objets. Cette communication se fera à travers la variable `customData`, il faudra que vous réalisiez les éléments suivants:

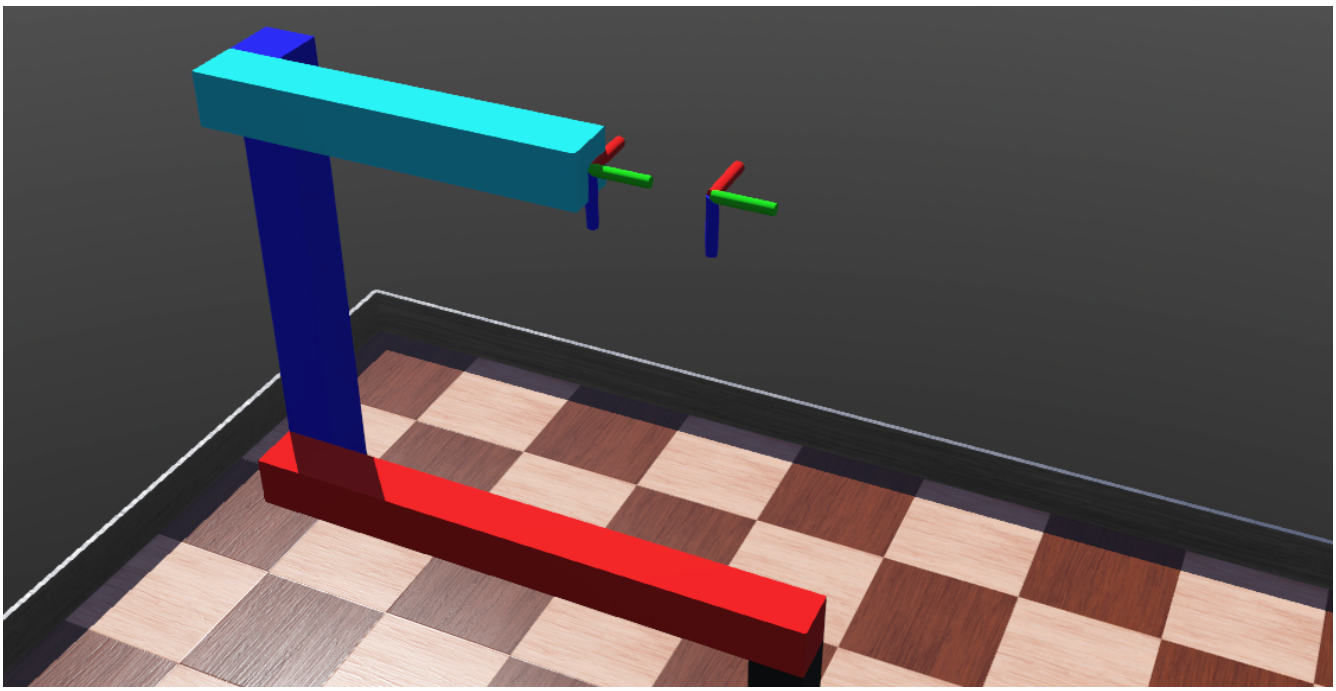

Figure 1: Visualisation d'erreur du modèles par représentation des repères.

- Dans le fichier `motor_controller.py`:

  1. Sérialiser la position et l'orientation de votre outil sous la forme d'une chaîne de caractère.

[2]Voir `https://cyberbotics.com/doc/reference/proto`

2. Affecter la chaîne de caractère obtenue à la variable `customData`[3].

- Dans le fichier `supervisor.py`, implémentez la fonction `updateBeliefToolTip`:

  1. Récupérez la chaîne de caractère en vous servant de la méthode `getField`[4]
  2. Redéfinir la valeur du champ `translation` pour positionner le repère correctement.
  3. Redéfinir la valeur du champ `rotation` pour que le repère soit bien orienté.

Pensez bien que vous pouvez:

- Commencer par prendre en compte uniquement la position.
- Forcer des valeurs dans `supervisor.py` pour vérifier que l'affichage correspond à ce que vous en attendez.
- Transmettre des valeurs intermédiaires (repère après la première articulation) pour valider des étapes une à une.

S'il n'y aucune erreur, le repère `belief_tooltip` qui se base sur votre implémentation devrait se confondre avec le repère `simulator_tooltip` (qui est à priori toujours bien placé et orienté).

2.4 Fichiers de logs et déboguage

Lorsque la simulation s'exécute, elle enregistre des données dans deux fichiers CSV différents qui partagent la même structure: `robot_data.csv` et `simulator_data.csv`. Ces fichiers contiennent les 5 colonnes suivantes:

t: Le temps qui s'est écoulé depuis le début de la simulation en secondes.

variable: Le nom de la variable qui est mesurée (nom de l'articulation, dimension pour la position, ...).

order: Peu utile pour l'instant, il permettra de distinguer position, vitesse et accélération(0,1 et 2).

source: D'où provient l'information? Est-ce la position désirée ou la position mesurée?

value: La valeur associée.

Le contenu initial vous permet d'observer l'évolution des trajectoires, par exemple en utilisant la procédure suivante:

```
# Lancement de la simulation avec fin automatique
SIM_AUTO_EXIT=1 SIM_DURATION=2 webots worlds/td_introduction.wbt
# Affichage des graphiques
python3 plot.py
```

En vous inspirant de la méthode `logData` présente dans `supervisor.py`, modifiez le fichier `motor_controller.py` pour enregistrer:

- Les positions articulaires obtenues à partir des capteurs
- La pose de l'outil

[3] Voir `https://cyberbotics.com/doc/reference/robot#wb_robot_set_custom_data`
[4] Voir `https://cyberbotics.com/doc/reference/supervisor#wb_supervisor_node_get_field`

3 Consignes de rendu

1. Éditez le contenu du fichier `to_pack.txt` pour qu'il contienne tous les fichiers demandés
2. Créer une archive en utilisant la commande suivante:

   ```
   python3 -m seauval.manager pack --group <group_id> .
   ```

3. Vérifiez le contenu de l'archive `tar tzf XXX.tar.gz`
4. Envoyer votre archive par mail avec le sujet suivant: "(4TSD904U) Rendu TD MGD". N'oubliez pas de mettre en copie tous les membres du groupe.