Partie 4 : Avec le vrai robot

Implémenter `MoveIt!` en simulation est pratique, mais le mieux est de le réaliser sur un robot réel !

Fichiers créés ou modifiés dans cette partie :

- `state_publisher_student.py`
- `scara_cpe_bringup.launch`
- `emagnet_student.cpp`
- `CMakeLists.txt`
- `moveCoins.py`
- `moveCoinsPlanningScene.py`

4.1 Real robot dynamixel interface

Pour que le robot réel puisse communiquer avec `MoveIt!`, il est d'abord nécessaire de créer un **publisher** (`state_publisher_student.py`) qui récupère les positions des joints et les renvoie dans un unique message.

Enfin, un *launch file* permet ensuite de lancer directement tous les noeuds qui permettent la gestion du robot réel : `scara_cpe_bringup.launch`. Il peut être exécuter grâce à la commande suivante :

```
$ roslaunch scara_cpe_robot scara_cpe_bringup.launch
```

Cette commande ouvre ainsi `RViz`, et il est alors possible de visualiser la position du robot réel.

4.2 Implement the effector (electro-magnet)

Le bras `Scara` utilisé possède un électro-aimant, qui peut ainsi servir pour détecter mais aussi saisir des pièces et les déplacer. Pour savoir si une pièce est sous l'électro-aimant, un message est reçu depuis la carte `Arduino` qui le pilote. Un noeud est ainsi créé pour transformer ce message en topic : `/detect_metal_pub`.

Pour lancer le noeud qui s'occupe de la détection :

```
$ rosrun scara_cpe_robot emagnet_student
```

Le topic créé permet ainsi de savoir quand une pièce vient d'être détecté ou vient de sortir de la zone détectée par l'électro-aimant.

4.3 Moveit and real robot

Lorsque l'interface entre le robot réel et `ROS` est créé, il devient possible de contrôler le bras grâce à `MoveIt!`.

Pour cela, il est nécessaire de modifier plusieurs fichiers pour adapter `MoveIt!` à utiliser le `controller` du robot réel.

Lorsque tout fonctionne correctement, un *launch file* permet de lancer tout ce qui est nécessaire pour utiliser le bras réel avec `MoveIt!` :

```
$ roslaunch scara_cpe_robot moveit_bringup.launch
```

Il devient alors possible de contrôler le bras réel grâce à l'interface de `RViz` en générant et exécutant le chemin pour atteindre la 'balle bleue'.

4.4 Play with coins

L'objectif de cette partie est de déplacer une pièce d'une **position A** à une **position B**.

Pour cela, un programme `Python` a été créé mais n'a pas pu être testé : `moveCoins.py`.

4.5 Planning scene

Lors des déplacement du bras, `MoveIt!` est capable de réaliser de l'évitement d'obstacles en prenant en compte ce que le bras tient avec lui. Pour cela, il est possible d'ajouter des éléments dans ce qui s'appelle le `Planning Scene`. C'est cet outil qui gère toute la détection et d'adapter les déplacements pour éviter des collisions.

(Partie non finalisée)

Conclusion

Le contrôle d'un bras robotique peut devenir très facilement complexe pour réaliser des mouvements tout en prenant en compte l'environnement. Ainsi, un outil comme `MoveIt!` est très efficace pour gérer les déplacements de toutes formes de bras et intègre tous les composants qui simplifient l'implémentation d'un système de contrôle d'un bras. Un fichier `URDF` (ou `xacro`) est alors utilisé pour définir la forme et la structure d'un bras afin que `MoveIt!` connaissent tous ses paramètres. Il est alors possible de facilement tester le bras dans un simulateur avant de l'intégrer sur un modèle réel. `MoveIt!` gère donc automatiquement tous les joints entre les différentes parties du bras pour savoir comment déplacer l'ensemble sans générer des mouvements erronés. Pour finir, `MoveIt!` apparaît comme un bon outil pour rapidement et efficacement implémenter un bras robotique.