TP Gazebo

Rémi RIGAL - Joris TILLET

Janvier 2021

1 Introduction

Gazebo est un logiciel utilisé pour la simulation en robotique. Un bon simulateur permet de tester des architectures de robot ainsi que des algorithmes rapidement et sans risque. On peut également s'en servir pour entraîner des réseaux de neurones. Gazebo présente l'avantage d'être pensé pour la simulation de robots, plus ou moins complexes et nombreux, dans des environnements très différents. Enfin, on peut facilement le connecter avec ROS, et il existe déjà beaucoup de briques facilement utilisables dans un projet (modèles de robots, capteurs, plugins, etc).

2 Installation et configuration

2.1 Installation de ROS 2

Installer la version de ROS 2 correspondante à la version d'Ubuntu installée sur votre ordinateur:

- Ubuntu 18.04
  - *Dashing* (`https://index.ros.org/doc/ros2/Installation/Dashing/Linux-Install-Debians`)
  - *Eloquent* (`https://index.ros.org/doc/ros2/Installation/Eloquent/Linux-Install-Debians`)
- Ubuntu 20.04
  - *Foxy* (`https://index.ros.org/doc/ros2/Installation/Foxy/Linux-Install-Debians`)

Dans toute la suite du TP, les packages à installer sont donnés sous la forme *ros-XXX-package* où "*XXX*" doit être remplacé par la version de ROS installée sur votre ordinateur (par ex. *ros-XXX-gazebo-ros* devient *ros-foxy-gazebo-ros* si ROS Foxy est installé).

Quelle que soit votre version de ROS, choisir l'installation complète en installant le package *ros-XXX-desktop*.

2.2 Installation de *colcon*

Colcon permet de compiler des workspaces ROS, installer le avec la commande suivante:

```
sudo apt install python3-colcon-common-extensions
```

2.3 Création d'un workspace

Créer un workspace dédié au TP, par exemple *∼/ensta_ws/src*.

Pour la suite, toujours compiler le workspace avec l'option *–symlink-install* comme ci-après:

```
colcon build --symlink-install
```

Des liens symboliques sont ainsi créés pour tous les fichiers non-compilés (Python, XML, YAML, Rviz...), il n'est pas donc pas nécessaire de recompiler le workspace à chaque fois que l'un de ces fichiers est modifié. Il est cependant nécessaire de recompiler le workspace lorsque l'un de ces fichiers est crée, supprimé ou renommé.

3 Langages de description

Pour créer un robot dans Gazebo, on utilise un langage de description pour définir son architecture physique. On décrit ainsi les différentes parties rigides (*links*) du robot et les articulations (*joints*) qui existent entre elles. L'objectif de ce TP est de concevoir son robot à l'aide d'une représentation similaire à la Figure 1.

Pour cela, le langage `URDF` (*Unified Robot Description Format*) sera utilisé, il est compatible avec Rviz et Gazebo notamment. Ce langage est basé sur la syntaxe XML. Afin de simplifier la rédaction d'un fichier `URDF`, la syntaxe `Xacro` vient compléter ce langage. Il permet notamment de définir des macros, de répartir la description du robot en plusieurs fichiers ou encore d'évaluer des formules mathématiques. Attention cependant, un fichier utilisant les notations propres au `Xacro` doit être converti en `URDF` avant d'être utilisé, la conversion peut s'effectuer à l'aide du package *xacro*.

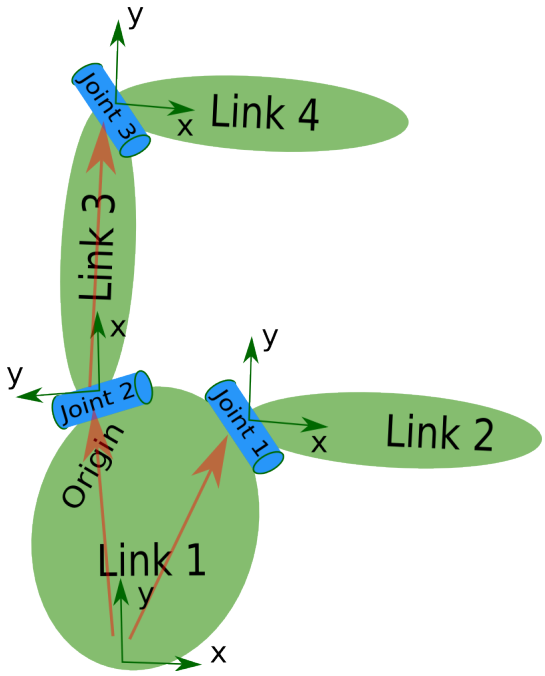

Figure 1: Exemple de decription d'un robot

Attention: Gazebo utilise nativement un format standard appelé le `SDF` (*Simulation Description Format*) sémantiquement très proche de l'`URDF` mais dont la syntaxe est différente.

Un *link* est défini par son aspect visuel (*visual*), sa géométrie de collision (*collision*) et son inertie (*inertial*), qui peuvent tous les trois différer. Ces éléments peuvent notamment être décrits par des formes simples ou par des *meshes* (au format *dae* par exemple). Deux *links* sont reliés par un *joint*, dont on précise notamment les degrés de libertés.

Les liens suivants seront utiles pour la suite du TP, ils présentent les syntaxes de l'`URDF` et du `Xacro`:

- `http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch`
- `http://gazebosim.org/tutorials/?tut=ros_urdf`
- `http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File`

4 Construction d'un robot à 2 roues

4.1 Création du package de description

Se placer dans le dossier *src* du workspace et créer un package *nom_robot_description* (remplacer *nom_robot* par le nom de votre robot) à l'aide de la commande suivante:

```
ros2 pkg create nom_robot_description
```

Nous n'allons pas créer de scripts dans ce package, les dossiers *src* et *include* peuvent donc être supprimés.

Ce package contiendra la description du robot dans un dossier *urdf*, un fichier *launch* permettant de visualiser le robot avec Rviz ainsi qu'un fichier de configuration pour Rviz. La structure de ce package devra être la suivante:

```
nom_robot_description/
    config/
        display.rviz
    launch/
        display.launch.py
    urdf/
        nom_robot.urdf
    CMakeLists.txt
    package.xml
```

Les fichiers dont les extensions sont *rviz*, *py* ou *urdf* ne sont pas compilés, il est donc nécessaire d'indiquer au compilateur que ces fichiers doivent être copiés dans le dossier *share* du package. Pour cela, ajouter les lignes suivantes dans le fichier *CMakeLists.txt* avant la commande *ament_package()*:

```
install(DIRECTORY config DESTINATION share/${PROJECT_NAME})
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
install(DIRECTORY urdf DESTINATION share/${PROJECT_NAME})
```

4.2 Préparation du package

Installer les packages nécessaires à la suite du TP:

```
sudo apt install ros-XXX-rviz2 ros-XXX-urdf ros-XXX-xacro
↪  ros-XXX-robot-state-publisher
↪  ros-XXX-joint-state-publisher-gui
```

Ouvrir Rviz à l'aide de la commande *rviz2* et enregistrer la configuration (*File →Save Config As*) dans un fichier *display.rviz* comme mentionné dans la section 4.1.

Créer le fichier `URDF` *nom_robot.urdf* dans le dossier *urdf* du package. Rédiger la description d'un simple robot composé d'un unique *link* avec la géométrie visuelle de votre choix. Vérifier l'intégrité du fichier avec la commande suivante:

```
check_urdf robot.urdf
```

La mention *Successfully Parsed XML* doit apparaître.

4.3 Affichage du robot dans Rviz

Créer le fichier *display.launch.py*. Ce fichier doit lancer 3 noeuds différents:

- *robot_state_publisher*: Ce noeud publie la description du robot dans le topic */robot_description* ainsi que les translations/rotations de chaque joint non fixe dans le topic */tf* calculées à partir des informations du topic */joint_states*.
- *joint_state_publisher_gui*: Ce noeud publie dans le topic */joint_states* des angles fictifs pour chaque joint non fixe. Ces angles sont modifiables à l'aide d'une interface avec des curseurs, cela permet de vérifier le sens et la direction des axes de rotation de chaque joint.
- *rviz2*: Le noeud associé au logiciel Rviz, il doit être démarré avec la configuration présente dans le fichier *display.rviz*.

Quelques liens utiles pour vous aider:

- `https://index.ros.org/doc/ros2/Tutorials/Launch-Files/Creating-Launch-Files`
- `https://github.com/ros/robot_state_publisher/tree/foxy`
- `https://github.com/ros/joint_state_publisher/tree/foxy`
- `https://github.com/ros/urdf_tutorial/tree/ros2` (Ne pas utiliser le lien du README)

Note: Pour les liens Github, ne pas oublier de sélectionner la branche correspondant à votre version ROS.

Une fois le fichier *launch* terminé, compiler le workspace et l'exécuter:

```
colcon build --symlink-install
. install/setup.bash
ros2 launch nom_robot_description display.launch.py
```

Dans Rviz, ajouter la visualisation *TF* en cliquant sur le bouton *Add* de la barre latérale *Displays*. Un repère devrait apparaître pour chaque *link* du robot.

Ajouter également la visualisation *RobotModel* en configurant le topic pour la description à la valeur */robot_description*. Régler le paramètre *Reliability Policy* de ce topic sur *Best Effort*. Le robot devrait être visible au centre de l'écran.

Sauvegarder la configuration de Rviz (*File* →*Save Config*). Le package est à présent configuré pour afficher le robot décrit par le fichier *robot.urdf*.

4.4 Construction d'un robot à 2 roues

Après avoir identifié les *links* et les *joints* nécessaires, modifier le fichier `URDF` afin de modéliser le robot détaillé dans la figure 2. Seule la géométrie visuelle est importante ici, les couleurs n'ont pas d'importance. La roue folle du robot doit être modélisée à l'aide d'une sphère fixe.

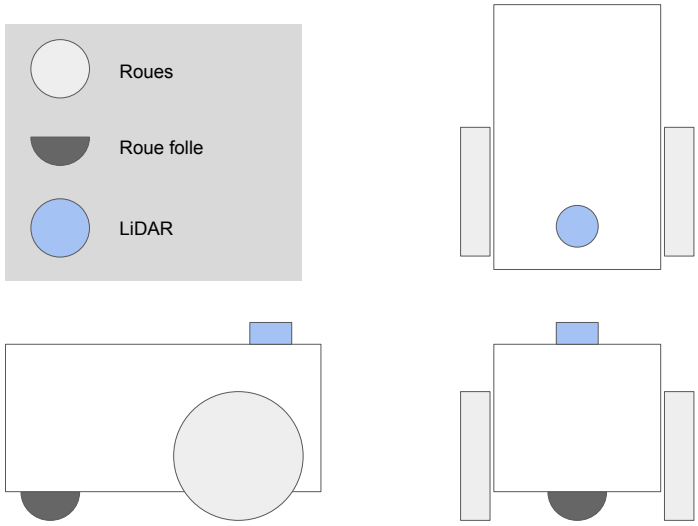

Figure 2: Schéma du robot à concevoir

5 Simulation du robot à 2 roues

5.1 Création du package de simulation

Dans cette partie nous allons importer le robot modélisé dans la section 4 dans *Gazebo* afin de simuler son comportement.

Se placer dans le dossier *src* du workspace et créer un package *nom_robot_gazebo* (remplacer *nom_robot* par le nom de votre robot) à l'aide de la commande suivante:

```
ros2 pkg create nom_robot_gazebo
```

Nous n'allons pas créer de scripts dans ce package, les dossiers *src* et *include* peuvent donc être supprimés.

Ce package contiendra un fichier *launch* permettant de démarrer la simulation du robot. La structure de ce package devra être la suivante:

```
nom_robot_gazebo/
    launch/
        gazebo.launch.py
    CMakeLists.txt
    package.xml
```

De même que pour le package *nom_robot_description*, ajouter la ligne suivante au fichier *CMakeLists.txt* avant la commande *ament_package()*:

```
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
```

5.2 Préparation du package

Installer les packages nécessaires à la suite du TP:

```
sudo apt install ros-XXX-gazebo-dev ros-XXX-gazebo-plugins
↪  ros-XXX-gazebo-ros ros-XXX-rqt-robot-steering
```

5.3 Simulation du robot dans Gazebo

Créer le fichier *gazebo.launch.py*. Ce fichier doit lancer 3 noeuds différents:

- *robot_state_publisher*: Ce noeud publie la description du robot dans le topic */robot_description* ainsi que les translations/rotations de chaque joint non fixe dans le topic */tf* calculées à partir des informations du topic */joint_states*.
- *gazebo*: Le noeud associé au simulateur *Gazebo*. *Gazebo* doit être démarré en incluant le fichier *gazebo.launch.py* du package *gazebo_ros* dans ce fichier *launch*.
- *gazebo_spawn_entity*: Ce noeud doit utiliser le script *spawn_entity.py* du package *gazebo_ros* afin de faire apparaître le robot du package *nom_robot_description* dans *Gazebo*.

Quelques liens utiles:

- `https://github.com/ros2/launch/blob/master/launch/doc/source/architecture.rst#id23`
- `https://github.com/ros-simulation/gazebo_ros_pkgs/blob/foxy/gazebo_ros/launch/spawn_entity_demo.launch.py`

Une fois le fichier *launch* terminé, compiler le workspace et l'exécuter:

```
colcon build --symlink-install
. install/setup.bash
ros2 launch nom_robot_gazebo gazebo.launch.py
```

Vérifier que Gazebo se lance et qu'il n'y a pas d'erreurs dans le terminal. Il est normal que le robot n'apparaissent pas dans la fenêtre principale de *Gazebo* à ce stade. Il doit cependant apparaître dans la liste *Models* sur la gauche de l'interface, le modèle *ground_plane* doit également être présent, si c'est le cas passer à la suite du TP.

5.4 Description physique du robot

La raison pour laquelle le robot n'apparaît pas dans *Gazebo* alors qu'il apparaît dans Rviz est qu'aucune inertie ne lui a été associé. En effet *Gazebo* est un simulateur physique, il a donc besoin qu'au moins un *link* du robot ait une masse et une matrice d'inertie afin de pouvoir calculer les équations physiques de ce dernier. Des géométries de collisions sont également nécessaires afin d'empêcher le robot de traverser le sol ou des obstacles.

Compléter la description `URDF/Xacro` du robot en ajoutant les moments d'inertie et la masse de chaque *link*. Ajouter également une géométrie de collision à chaque *link*, cette géométrie pourra être identique à la géométrie visuelle. Vous pouvez vous aider du lien suivant pour les moments d'inertie: `https://fr.wikipedia.org/wiki/Moment_d%27inertie#Moments_d'inertie_particuliers`.

5.5 Spécification des matériaux

Afin que les interactions entre les composants soient au plus proche de la réalité, il est nécessaire de spécifier le matériau de chaque *link*.
Quelques liens utiles pour configurer les matériaux:

- `http://gazebosim.org/tutorials?cat=physics`
- `https://www.engineeringtoolbox.com/friction-coefficients-d_778.html`

Assigner le matériau suivant aux deux roues du robot:

```
<mu1>1.2</mu1>
<mu2>1.2</mu2>
<kp>500000.0</kp>
<kd>10.0</kd>
<minDepth>0.001</minDepth>
<maxVel>0.1</maxVel>
<material>Gazebo/FlatBlack</material>
```

Note: le tag *material* n'est utile que pour la couleur, il peut cependant vous servir d'indication: si les roues deviennent noires, c'est que les paramètres ont bien été pris en compte par *Gazebo*.

Assigner ensuite un matériau "glissant" à la roue folle afin de réduire ses frottements avec le sol.

6 Ajout d'un contrôleur moteur

Ajouter un contrôleur moteur de type char afin de permettre le contrôle simultané des deux roues du robot.

Ce contrôleur doit être paramétré pour publier l'odométrie du robot sur le topic */odom* et doit être capable de faire avancer et tourner le robot lorsque des ordres sont publiés dans le topic */cmd_vel*.

Il doit également être configuré pour publier dans le topic */joint_states* l'état des joints des roues afin de permettre au *Robot State Publisher* de reconstituer la TF du robot (les translations et rotations entre les différents *links*).

Le plugin *libgazebo_ros_diff_drive* présent dans *Gazebo* offre toutes ces fonctionnalités. S'aider de l'exemple suivant afin d'intégrer ce contrôleur au robot: `https://github.com/ros-simulation/gazebo_ros_pkgs/blob/dashing/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world`.

Il est également possible de s'aider du guide de migration de ROS 1 à ROS 2, seule documentation disponible, attention cependant de ne pas confondre les deux syntaxes: `https://github.com/ros-simulation/gazebo_ros_pkgs/wiki/ROS-2-Migration:-Diff-drive`.

Tester le fonctionnement du contrôleur en utilisant le plugin *Robot Steering* de RQT, en vérifiant que les roues tournent dans le bon sens pendant les rotations (c'est mieux `;)`).

7 Ajout de capteurs

On s'intéresse maintenant à la simulation des différents capteurs du robot qui nous permettrons de tester des algorithmes de cartographie, de navigation, de traitement d'images etc...

Créer un world Gazebo avec des formes ou utiliser le world *willowgarage.world* pour la suite des travaux, c'est ce monde dont on se servira pour valider les informations retournées par les capteurs.

7.1 LiDAR

Utiliser le *link* du LiDAR précédemment créé, il doit être fixe par rapport au chassis du robot. Il est important que ce dernier soit au dessus des roues afin de préserver un champ de vision à 360°. L'apparence visuelle du LiDAR n'est pas importante mais il doit être visible.

Gazebo est capable de simuler la sortie d'un LiDAR classique, pour cela il faut déclarer un *sensor* de type *ray* dans la description `URDF/Xacro`. S'aider de l'exemple disponible sur Github: `https://github.com/ros-simulation/gazebo_ros_pkgs/blob/dashing/gazebo_plugins/worlds/gazebo_ros_ray_sensor_demo.world`.

De même que pour le contrôleur, le guide de migration de ROS 1 vers ROS 2 peut être utile: `https://github.com/ros-simulation/gazebo_ros_pkgs/wiki/ROS-2-Migration:-Ray-sensors`.

Une fois le capteur créé, faire afficher les impacts du LiDAR sur Rviz à l'aide d'une visualisation *LaserScan*.

7.2 Caméra

Ajouter un *link* caméra à l'avant du robot et lui associer un capteur caméra. Afficher le retour vidéo dans Rviz.