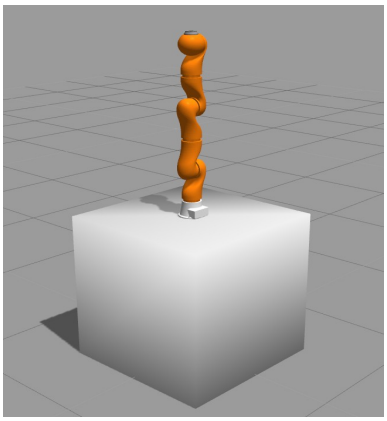
Laurent LEQUIEVRE – Ingénieur CNRS - Institut Pascal UMR6602

Equipe Maccs http://ip.univ-bpclermont.fr/index.php/fr/maccs

Juan Antonio CORRALES RAMON - Youcef MEZOUAR

Notions de Gazebo

Gazebo est un outil de simulation 3D permettant une communication avec ROS via des 'services' et des 'topics'.
Gazebo dispose d'une banque de données contenant des modèles de robots, capteurs ou encore objets 3D, que l'on peut ajouter dans le monde simulé (http://gazebosim.org https://bitbucket.org/osrf/gazebo_models/src). Il permet aussi de simuler le fonctionnement d'un robot spécifique (décrit avec le format URDF Universal Robotic Description Format ou bien dans le format SDF Simulation Description Format).



L'objectif est de simuler un bras **kuka lwr 4+** dans gazebo, de faire bouger ses articulations via des 'topics' ROS.

1- Créer un nouveau paquet nommé 'kuka_lwr_description'

A la racine de ce paquet créer les sous dossiers :
'config', 'launch', 'meshes', 'model', 'rviz', 'worlds'

2- La partie modélisation URDF du robot est fournie dans le dossier
'tp_ros_master_robotique/Ressources/kuka' :
- Copier les fichiers graphiques dans le dossier 'meshes'.
- Copier le fichier 'config.rviz' dans le dossier 'rviz'.
- Copier les fichiers 'kuka_lwr_utils.xacro',
'kuka_lwr_materials.xacro','kuka_lwr_gazebo.urdf.xacro', 'platform.urdf.xacro',
'only_kuka_lwr_gazebo.urdf.xacro' dans le dossier 'model'.

Regarder et comprendre en détail les fichiers 'xacro'.

Remarques :
- Le fichier ' kuka_lwr_gazebo.urdf.xacro' contient la macro 'kuka_lwr' qui décrit le robot ('link', 'joints' ...).
- Des balises de type **<inertial>** ont été ajoutées pour chaque 'link' (c'est spécifique à gazebo). Ceci permet de caractériser la valeur de leur masse, la position de son centre par rapport au 'link', ainsi que sa matrice d'inertie (qui est calculée via les macros '**cylinder_inertia_def**' et '**cuboid_inertia_def**' définie dans le fichier ' **kuka_lwr_utils.xacro**') :

```
<inertial>
  <mass value="${base_mass}"/>
  <origin xyz="0 0 0.055"  />
  <cylinder_inertia_def radius="0.06" length="0.11" mass="${base_mass}"/>
</inertial>
```

- De nouvelles propriétés dynamiques du 'joint' on été définies ('damping' amortissement, 'friction' frottement)

```
<dynamics friction="10.0" damping="1.0"/>
```

Pour les 'joints' de type 'revolute' → 'friction' en N.m , 'damping' en N.m.s/rad
- A chaque partie <**visual**> du 'link' a été associée une couleur spécifique via la balise **<material>**.

```
<material name="Kuka/Orange"/>
```

Ces couleurs on été définies dans le fichier '**kuka_lwr_materials.xacro**' avec un code 'rgba' spécifique :

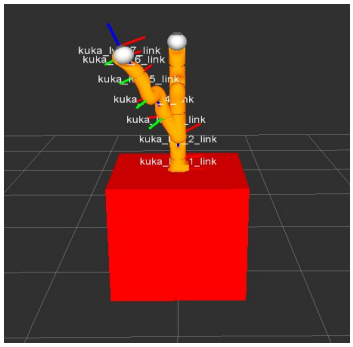
```
<material name="Kuka/Orange">
   <color rgba="1.0 0.487 0 1.0"/>
 </material>
```

- Le fichier '**platform.urdf.xacro**' contient la description complète de la plate-forme, c'est à dire le socle relié à la description du bras via l'appel de sa macro.
- Le fichier '**only_kuka_lwr_gazebo.urdf.xacro**' contient uniquement la description d'un bras (sans le socle) et servira plus tard.

3- Créer un fichier nommé 'display_rviz.launch' dans le dossier 'launch' permettant de voir dans un premier temps, cette modélisation dans 'rviz'.

```
<launch>
            <arg name="model" default="platform.urdf.xacro"/>
            <param name="robot_description" command="$(find xacro)/xacro.py '$(find kuka_lwr_description)/model/$(arg
model)'" />
            <param name="use_gui" value="true"/>
           <arg name="rvizconfig" default="$(find kuka_lwr_description)/rviz/config.rviz" />
            <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
            <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
            <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />
</launch>
```

Lancer le 'launch' et bouger les articulations via les 'sliders'.



4- Afficher le robot dans gazebo

Pour commencer, il faut envoyer à gazebo un 'monde vide' via le paquet '**gazebo_ros**' et un fichier 'launch' paramétrable '**empty_world.launch**'.

Essayer cette commande :

```
roslaunch gazebo_ros empty_world.launch
```

Ensuite, pour afficher le modèle (URDF) du robot dans gazebo, il faut utiliser le service **gazebo/spawn_model** du paquet '**gazebo_ros**' (http://wiki.ros.org/gazebo_ros).
Un 'node/script' nommé '**spawn_model**' du paquet '**gazebo_ros**' permet d'utiliser ce service.

Ajouter un 'model' en commande dans un terminal :
Se mettre dans le dossier 'tp_ros_master_robotique/Ressources/gazebo_models' et lancer cette commande (il faut laisser gazebo ouvert) :

```
rosrun gazebo_ros spawn_model -file coke_can/model.sdf -gazebo -model my_can_coke -z 0
rosrun gazebo_ros spawn_model -file beer/model.sdf -gazebo -model my_beer -z 0 -x 1 -y 1
```

Pour supprimer le 'model' 'my_can_coke', on peut utiliser le 'service' gazebo 'delete_model' :

```
rosservice list
rosservice call /gazebo/delete_model my_can_coke
```

Ajouter un 'model' avec un fichier 'launch' :

```
<node name="spawn_trunk_urdf" pkg="gazebo_ros" type="spawn_model" args="-param robot_description -urdf
-model $(arg robot_name)" respawn="false" output="screen" />
```

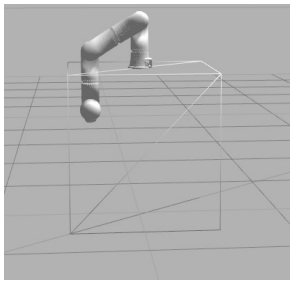
Ici les arguments fournis à '**spawn_model**' sont '**-param**' pour indiquer que le contenu 'urdf' du modèle se trouve dans la variable 'robot_description' et '**-model**' pour donner un nom au modèle dans gazebo.

Créer un fichier 'launch' nommé 'display_gazebo.launch' dans le dossier 'launch' du paquet :

```
<launch>
	<!-- set some ros tools -->
	<arg name="use_rviz" default="false"/>
	<arg name="rvizconfig" default="$(find kuka_lwr_description)/rviz/config.rviz" />

	<!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
	<include file="$(find gazebo_ros)/launch/empty_world.launch" />

	<arg name="robot_name" default="platform"/>
	<param name="robot_description" command="$(find xacro)/xacro.py '$(find kuka_lwr_description)/model/$(arg
robot_name).urdf.xacro'" />
	<node name="spawn_trunk_urdf" pkg="gazebo_ros" type="spawn_model" args="-param robot_description -urdf
-model $(arg robot_name)" respawn="false" output="screen" />

	<group if="$(arg use_rviz)">
		<node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />
	</group>
</launch>
```

Lancer le fichier 'launch'
Remarque :
Le fichier 'empty_world.launch' que l'on inclut, lance automatiquement le client et le serveur de gazebo avec un monde vide.

Observer que le 'robot' est de couleur 'grise' et surtout que ses 'link' on tendance à retomber (du à la masse définie + gravité). Il faut donc définir les couleurs des 'link' pour gazebo et surtout définir un contrôleur/moteur pour les articulations.



Remarque :
Dans la cas ou 'gazebo' ne s'affiche pas, il se peut que c'est uniquement sa partie cliente qui pose problème. Vous pouvez essayer de lancer la partie cliente de 'gazebo' dans un autre terminal avec cette commande :

```
rosrun gazebo_ros gzclient
```

Vous pouvez vous amuser à lancer 'gazebo + roscore' à la main en lançant dans des fenêtres séparées :

```
roscore
rosrun gazebo_ros gzserver
rosrun gazebo_ros gzclient
```

5- Définir le type de 'plugin' gazebo à utiliser.

Créer un fichier 'kuka_lwr.gazebo.xacro' dans le dossier 'model', qui définira un 'plugin' gazebo.

Un 'plugin gazebo' est du code C++ compilé en bibliothèque dynamique (fichier '.so') qui sera chargé dans 'gazebo' et permettra d'interagir avec toutes ses fonctionnalités.

Plusieurs types de 'plugin' existent :
'**world plugin**' : on peut changer le moteur physique, les lumières, et autres propriétés du monde simulé.
'**model plugin**' : on peut interagir avec le modèle chargé dans 'gazebo', contrôler par exemple les 'joints' d'un robot.
'**sensor plugin**' : sert à modéliser des capteurs, comme une caméra par exemple.
'**system plugin**' : permet de contrôler toutes les fonctions système de 'gazebo'.
'**visual plugin**' : permet par exemple d'ajouter des objets graphiques dans le monde simulé.

Dans notre cas, nous avons besoin d'un 'plugin' de type 'model plugin' pour interagir avec les articulations du robot afin de les faire bouger.
Ce 'plugin' devra aussi communiquer avec un contrôleur ROS. Ce dernier sera chargé à chaque mise à jour de la simulation 'gazebo' de calculer la bonne commande à envoyer au robot simulé.
Le concept de contrôleur se nomme : 'ROS control (http://gazebosim.org/tutorials?tut=ros_control)'.

Il existe un 'plugin' clé en main permettant de contrôler les articulations d'un robot dans 'gazebo' via 'ros_control', ce 'plugin' se nomme '**libgazebo_ros_control.so**' et s'ajoute comme ceci dans l'urdf :

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
        <robotNamespace>/ns_kuka_lwr</robotNamespace>
```

```
        <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>
</gazebo>
```

La balise <**robotSimType**> permet d'indiquer le type d'interface que l'on souhaite utiliser entre 'gazebo' et le contrôleur manager (ros_control).
Ici il s'agit d'une classe C++ '**DefaultRobotHWSim'** qui fait partie du 'namespace' C++ **'gazebo_ros_control'**. Pour info : Cette classe hérite de **'gazebo_ros_control::RobotHWSim'.**
Ici un 'namespace **/ns_kuka_lwr**' a aussi été défini pour le robot, par commodité. Il permettra notamment par la suite, de travailler avec plusieurs robots dans des 'namespace' différents.

Le contenu du fichier 'kuka_lwr.gazebo.xacro' sera une macro nommée 'kuka_lwr_gazebo' :

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">

	<xacro:macro name="kuka_lwr_gazebo" params="name namespace">

		<gazebo>
			<plugin name="${name}_gazebo_ros_control" filename="libgazebo_ros_control.so">
				<robotNamespace>${namespace}</robotNamespace>
				<robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
			</plugin>
		</gazebo>

		<gazebo reference="${name}_base_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_1_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_2_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_3_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_4_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_5_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_6_link">
			<gravity>true</gravity>
			<material>Gazebo/Orange</material>
		</gazebo>

		<gazebo reference="${name}_7_link">
			<gravity>true</gravity>
			<material>Gazebo/Grey</material>
		</gazebo>

	</xacro:macro>

</robot>
```

On en profite pour définir aussi toutes les couleurs de type 'gazebo' pour chaque 'link'. Cela permet de convertir les couleurs définies dans la partie 'visual' du 'link' dans l'urdf en couleur supportées par gazebo (les couleurs 'material gazebo' commencent toutes par 'Gazebo/color'). Enfin, on positionne aussi la propriété de gravité à 'true' pour chaque 'link'.
On n'oublie pas de passer en paramètre de cette macro le 'namespace' **${namespace}** et le nom du robot **${name}**.

Attention : Le plugin '**libgazebo_ros_control.so'** cherche à lire les descriptions des 'joints' du bras dans un paramètre nommé par défaut 'robot_description'. Compte tenu que ce plugin se trouve dans un 'namespace', il faudra donc définir le paramètre 'nom_namespace/robot_description'. Ce paramètre contiendra uniquement la modélisation du bras (contenu dans le fichier '**only_kuka_lwr_gazebo.urdf.xacro**').

Ajouter ceci dans le fichier 'display_gazebo.launch' :

```
<param name="ns_kuka_lwr/robot_description" command="$(find xacro)/xacro.py '$(find
kuka_lwr_description)/model/only_kuka_lwr_gazebo.urdf.xacro'" />
```

Le choix du namespace '**ns_kuka_lwr**' sera appliqué pour toute la modélisation qui suit.

6- Définir les informations sur les transmissions reliant les 'joints' et actionneurs.

Pour utiliser le 'plugin **gazebo_ros_control**', il faut ajouter des informations supplémentaires dans la modélisation 'URDF', notamment des informations sur le type de transmission **<transmission ...>** reliant un 'joint' **<joint..>** et un actionneur **<actuator ...>**. Un 'joint' par définition est relié mécaniquement à un moteur via une transmission, et c'est cette transmission qui est définie :

```
<transmission name="trans0">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="kuka_lwr_0_joint">
                <hardwareInterface>PositionJointInterface</hardwareInterface>
        </joint>
        <actuator name="kuka_lwr_0_motor">
                <mechanicalReduction>1</mechanicalReduction> s
        </actuator>
</transmission>
```

Ici pas de 'réduction mécanique' pour la transmission (le paramètre est de **1**). Les infos de cette transmission de type '**transmission_interface/SimpleTransmission**' (une simple transmission avec un facteur de réduction, la seule supportée actuellement) concernent le joint '**kuka_lwr_0_joint**' avec une interface 'hardware' de type '**PositionJointInterface**'.

Il existe plusieurs type d'interfaces :
- **Effort** Joint Interface
- **Velocity** Joint Interface
- **Position** Joint Interface

qui respectivement permettent de lire l'état du 'joint' mais aussi de le commander en 'effort', 'vitesse', 'position'.

Le 'plugin **gazebo_ros_control**' analysera toutes les informations de transmissions et chargera les interfaces spécifiées.

Créer un fichier nommé 'kuka_lwr.transmission.xacro' dans le dossier 'model'.

Voici le contenu du fichier :

```
<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

	<xacro:property name="InterfacePosition" value="PositionJointInterface"/>
	<xacro:property name="InterfaceEffort" value="EffortJointInterface"/>

	<xacro:macro name="kuka_lwr_transmission" params="name">

		<transmission name="${name}_0_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_0_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_0_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

		<transmission name="${name}_1_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_1_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_1_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

		<transmission name="${name}_2_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_2_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_2_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

		<transmission name="${name}_3_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_3_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_3_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

		<transmission name="${name}_4_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_4_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_4_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>
```

```
		<transmission name="${name}_5_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_5_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface>  -->
			</joint>
			<actuator name="${name}_5_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

		<transmission name="${name}_6_trans">
			<type>transmission_interface/SimpleTransmission</type>
			<joint name="${name}_6_joint">
				<hardwareInterface>${InterfacePosition}</hardwareInterface>
				<!--<hardwareInterface>${InterfaceEffort}</hardwareInterface> -->
			</joint>
			<actuator name="${name}_6_motor">
				<mechanicalReduction>1</mechanicalReduction>
			</actuator>
		</transmission>

	</xacro:macro>

</robot>
```

On définit une macro 'kuka_lwr_transmission' qui ajoutera pour chaque 'joint' du bras kuka lwr les informations de transmission.

7- Inclure les fichiers de 'plugin' et 'transmission' dans la macro de définition du bras kuka.

Modifier le fichier 'kuka_lwr_gazebo.urdf.xacro' en ajoutant après les 'include' des 'utils' ceci :

```
<!-- gazebo and transmission models /-->
<xacro:include filename="$(find kuka_lwr_description)/model/kuka_lwr.gazebo.xacro"/>
<xacro:include filename="$(find kuka_lwr_description)/model/kuka_lwr.transmission.xacro"/>
```

Ajouter l'appel de ces macros définies dans les fichiers inclus précédemment. Ceci juste avant la fin de la définition de la macro du bras kuka :

```
<!-- URDF model /-->
<xacro:macro name="kuka_lwr" params="parent name namespace *origin">
	….
	<xacro:kuka_lwr_gazebo name="${name}" namespace="${namespace}"/>
	<xacro:kuka_lwr_transmission name="${name}"/>

</xacro:macro>
```

Lancer le 'launch' 'display_gazebo' pour observer les changements.

Les couleurs sont prises en compte dans gazebo, mais il manque dans le 'launch' le lancement du '**controller_manager**', dont son rôle sera de charger le ou les contrôleurs du robot.

8- Ajouter le '**controller_manager**' pour charger les contrôleurs.

Dans notre cas, on va utiliser des 'contrôleurs' fournis par ROS, mais on peut tout à fait développer son propre contrôleur.

La liste des contrôleurs et leurs paramètres vont être chargés via un fichier YAML dans le serveur de paramètres ROS avec la commande 'rosparam'.

Créer un fichier 'kuka_lwr_control.yaml' dans le dossier 'config' du paquet 'kuka_lwr_description' :

```
ns_kuka_lwr:
 # CONTROLLERS USED IN THE EXAMLE
 joint_state_controller:
   type: joint_state_controller/JointStateController
   publish_rate: 100

 # Position Controllers ---------------------------------------
 joint0_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_0_joint

 joint1_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_1_joint

 joint2_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_2_joint

 joint3_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_3_joint

 joint4_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_4_joint

 joint5_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_5_joint

 joint6_position_controller:
   type: position_controllers/JointPositionController
   joint: kuka_lwr_6_joint
```

Ici on énumère dans le 'namespace' '**ns_kuka_lwr**' la liste des contrôleurs que l'on souhaite charger. Un nom est donné à chaque contrôleur (par exemple : 'joint5_position_controller') suivi de leurs caractéristiques et paramètres.
Le contrôleur de type ' joint_state_controller/JointStateController' permettra de publier la position des 'joint' qui sera nécessaire au 'robot_state_publisher'.
Le contrôleur de type 'position_controllers/JointPositionController' permettra de contrôler le 'joint' par position angulaire.
(http://docs.ros.org/indigo/api/position_controllers/html/namespaceposition__controllers.html)

Attention :
Le copier/coller des données ci-dessus, peut entraîner des tabulations au lieu d'espaces. **Le fichier 'yaml' NE DOIT PAS CONTENIR DE TABULATIONS**.

Les contrôleurs peuvent se charger via un '**controller manager**' avec le script '**spawner**' qui fait partie du paquet '**controller_manager**'.

Modifier le fichier 'display_gazebo.launch' en ajoutant ceci :

```
<!-- load all controller configurations to rosparam server -->
<rosparam file="$(find kuka_lwr_description)/config/kuka_lwr_control.yaml" command="load"/>
	<arg name="controllers" default="joint0_position_controller joint1_position_controller
joint2_position_controller joint3_position_controller
	joint4_position_controller joint5_position_controller joint6_position_controller" />

	<!-- spawn controllers in namespace 'ns_kuka_lwr' -->
	<group ns="ns_kuka_lwr">
		<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="true"
output="screen" args="joint_state_controller $(arg controllers)"/>
	</group>

	<!-- publish joint states -->
	<group ns="ns_kuka_lwr">
		<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
respawn="false" output="screen">
			<remap from="/joint_states" to="/ns_kuka_lwr/joint_states" />
		</node>
	</group>
```

Ici on retrouve bien les noms des contrôleurs dans la variable '**controllers**'.
Remarquer l'utilisation du 'namespace' '**/ns_kuka_lwr**' pour le 'controller_manager'.

On en profite aussi pour ajouter (toujours dans le namespace 'ns_kuka_lwr') le 'node robot_state_publisher' qui publiera la pose 3D du robot, utilisée par tf.
Pour calculer cette pose, le 'robot_state_publisher' a besoin des valeurs articulaires, c'est à dire les 'joint_states' publiés par le 'joint_state_controller' (définit dans le fichier ' kuka_lwr_control.yaml').
Il faut donc penser à remapper le 'topic' par défaut '/joint_states' en '/ns_kuka_lwr/joint_states', puisqu'on a définit le 'joint_state_controller' dans le namespace 'ns_kuka_lwr'.

Relancer le fichier 'display_gazebo.launch'

Observer la liste des 'topics' disponibles pour chaque 'joint'.

```
rostopic list
…
/ns_kuka_lwr/joint0_position_controller/command
/ns_kuka_lwr/joint1_position_controller/command
/ns_kuka_lwr/joint2_position_controller/command
...
```

Le 'topic command' nous permettra de commander en radians le 'joint' comme ceci :

```
rostopic pub -1 /ns_kuka_lwr/joint0_position_controller/command std_msgs/Float64 '{ data : -0.5}'
```

ou bien comme ceci :

```
rostopic pub -1 /ns_kuka_lwr/joint1_position_controller/command std_msgs/Float64 -- 0.5
```

On peut maintenant obtenir l'état des 'joints' avec :

```
rostopic echo /ns_kuka_lwr/joint_states
```

Ou encore calculer les 'tf' entre 'joints'.

9- On souhaite maintenant modéliser 2 bras Kuka (un bras droit et un bras gauche) reliés sur un tronc commun (fichier 'platform.stl').
On souhaite pouvoir les afficher dans 'gazebo' et les bouger via des 'controllers' en position.
Attention : les bras seront définis dans 2 'namespace' différents ('ns_kuka_lwr_left' et 'ns_kuka_lwr_right') …. A vous de jouer !