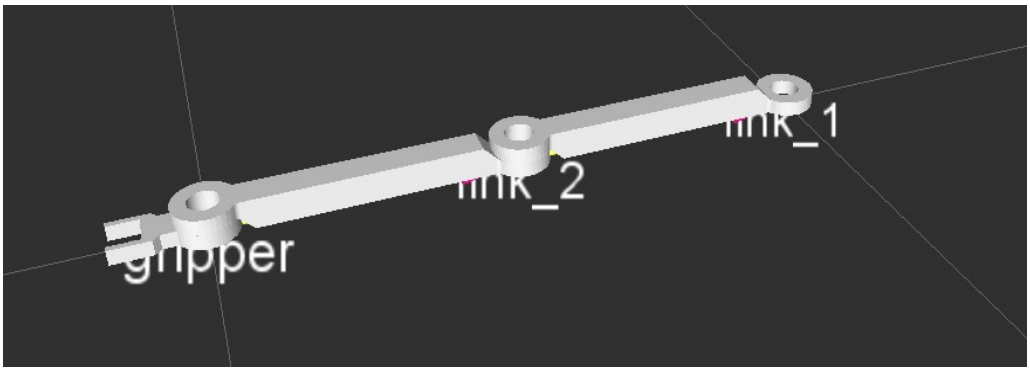
Laurent LEQUIEVRE – Ingénieur CNRS - Institut Pascal UMR6602

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

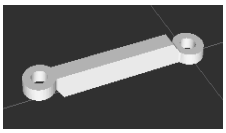
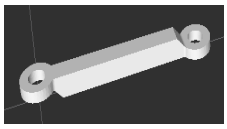
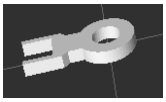
Juan Antonio CORRALES RAMON - Youcef MEZOUAR

Notions de URDF, xacro, rviz, tf

On souhaite modéliser ce type de robot, le visualiser et le bouger dans rviz :



Il est composé de 3 parties nommées 'link_1', 'link_2' et 'gripper' :

Pour ce faire, on va définir les caractéristiques de ces parties dans un fichier **URDF** (**U**nified **R**obot **D**escription **F**ormat http://wiki.ros.org/urdf). La syntaxe de ce type de fichier est composé de balises au format XML.

1- Créer un nouveau paquet nommé '**my_first_urdf**'

2- Créer un dossier 'launch', un dossier 'rviz', un dossier 'meshes', un dossier 'urdf' à la racine du paquet. Dans le dossier 'meshes' créer 2 sous dossiers 'collision' et 'visual'.
→ 'meshes' contiendra les fichiers de représentation graphique (au format 'stl' par exemple)
→ 'urdf' contiendra la description du robot (liens, joints ...)
→ 'launch' contiendra le script de lancement de visualisation du fichier URDF dans RVIZ

Copier les ressources graphiques contenues dans le dossier 'tp_ros_master_robotique/Ressources/gripper/meshes' dans les dossiers 'collision' et 'visual' de votre paquet.

3- Dans le dossier 'urdf' ajouter un nouveau fichier URDF nommé 'planar_3dof.urdf'.

Ajouter ce descriptif minimal du robot dans le fichier URDF :

```
<robot name="planar_3dof">
</robot>
```

Ici on spécifie que l'on souhaite définir un robot en lui donnant un nom.

Ajouter les 3 parties du robot que l'on nomme 'link' et un 'base_link' virtuel (expliqué plus tard):

```
<robot name="planar_3dof">
 <link name="base_link"/>

 <link name="link_1">
  <visual>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/visual/arm_link.stl"/>
   </geometry>
   <material name="grey">
    <color rgba="0.7 0.7 0.7 1.0"/>
   </material>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/collision/arm_link.stl"/>
   </geometry>
  </collision>
 </link>

 <link name="link_2">
  <visual>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/visual/arm_link.stl"/>
   </geometry>
   <material name="grey"/>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/collision/arm_link.stl"/>
   </geometry>
  </collision>
 </link>

 <link name="gripper">
  <visual>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/visual/gripper.stl"/>
   </geometry>
   <material name="grey"/>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://my_first_urdf/meshes/collision/gripper.stl"/>
   </geometry>
  </collision>
 </link>
</robot>
```

La balise **<visual>...</visual>** permet de définir la géométrie à afficher.
Voici un exemple simple avec un cylindre de 0.5 m de long et d'un rayon de 0.1 m, d'une couleur RGBA grise :

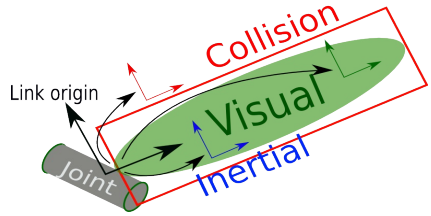
```
<visual>
  <geometry>
    <cylinder length="0.5" radius="0.1"/>
  </geometry>
  <material name="grey">
     <color rgba="0.7 0.7 0.7 1.0"/>
   </material>
</visual>
```

Dans notre modélisation, il s'agit d'un fichier graphique au format 'stl'.

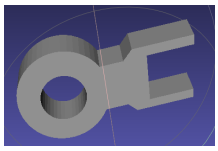
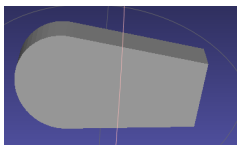
```
<mesh filename="package://my_first_urdf/meshes/visual/arm_link.stl"/>
```

Remarque :
Pour visualiser sous ubunu un fichier 'stl', il faut installer l'outil 'meshlab'.
sudo apt-get install meshlab

La balise **<collision>...</collision>** permet de définir la géométrie approximée autour du 'visual' nécessaire au calcul de collision.



Ici par exemple le 'gripper' 'visual' et 'collision'.

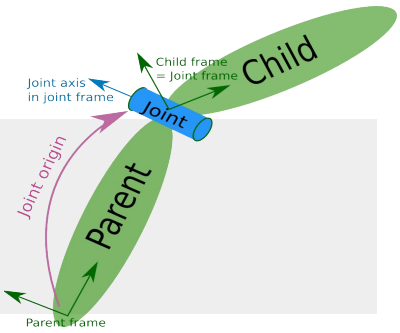



Remarque : Il est à noter que le repère d'un 'link' se trouve au pied de celui-ci.

Ajouter maintenant les articulations entre ces différents 'link' appelées 'joint':
→ permet de relier 2 'link' : entre un parent et un enfant.
→ permet de spécifier le type de mouvement articulaire (revolute, continuous, prismatic, fixed, floating, planar) et sur quel axe cela tourne.
→ permet de spécifier les limites articulaires et de vitesse.
→ permet de spécifier la position du 'joint' par rapport au 'link' parent.

Exemple :

```
<joint name="joint_2" type="revolute">
  <origin xyz="0.5 0 0" rpy="0 0 0"/>
  <parent link="link_1"/>
  <child link="link_2"/>
  <axis xyz="0 0 1"/>
  <limit lower="-1.57" upper="1.57" effort="0" velocity="0.5"/>
</joint>
```



Dans cet exemple, l'articulation '**joint_2**' est de type '**revolute**' (une articulation à charnière qui tourne autour d'un axe et possède des limites supérieures et inférieures).
Cette articulation relie le parent '**link_1**' et l'enfant '**link_2**'.
Elle ne tourne qu'autour de l'axe Z (**<axis xyz="0 0 1"/>**).
Elle se trouve par rapport au repère du parent à une position et rotation donnée (**<origin xyz="0.5 0 0" rpy="0 0 0"/>**), ici 0.5 m en X.
Elle possède des limites articulaires (en radians), d'effort max (ici en Nm) et de vitesse max (m/s) (**<limit lower="-1.57" upper="1.57" effort="0" velocity="0.5"/>**)

Voici les 'joints' à ajouter après les 'link' dans le fichier URDF du robot :

```
<joint name="joint_1" type="revolute">
   <origin xyz="0 0 0" rpy="0 0 0"/>
   <parent link="base_link"/>
   <child link="link_1"/>
   <axis xyz="0 0 1"/>
   <limit lower="-1.57" upper="1.57" effort="0" velocity="0.5"/>
 </joint>

 <joint name="joint_2" type="revolute">
   <origin xyz="0.5 0 0" rpy="0 0 0"/>
   <parent link="link_1"/>
   <child link="link_2"/>
   <axis xyz="0 0 1"/>
   <limit lower="-1.57" upper="1.57" effort="0" velocity="0.5"/>
 </joint>

 <joint name="joint_3" type="revolute">
   <origin xyz="0.5 0 0" rpy="0 0 0"/>
   <parent link="link_2"/>
   <child link="gripper"/>
   <axis xyz="0 0 1"/>
   <limit lower="-1.57" upper="1.57" effort="0" velocity="0.5"/>
 </joint>
```

Remarquer l'utilité du 'base_link' virtuel comme parent de 'joint_1'.

Pour vérifier la validité du fichier URDF, on peut installer un outil 'check_urdf' via les paquets ubuntu :

```
sudo apt-get install liburdfdom-tools
```

Et faire un 'check' du fichier :

```
check_urdf  planar_3dof.urdf
```

4- Créer un 'launch' pour visualiser le fichier URDF dans RVIZ.

Pour simplifier l'affichage de cet URDF dans RVIZ on a besoin de créer un fichier 'launch' que l'on nommera 'display_simple.launch' et que l'on sauvegardera dans le dossier 'launch' du paquet.
Voici son contenu :

```
<launch>
        <arg name="model" default="planar_3dof.urdf"/>
        <param name="robot_description" command="$(find xacro)/xacro.py '$(find my_first_urdf)/urdf/$(arg model)'" />
        <param name="use_gui" value="true"/>
        <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="" required="true" />
</launch>
```

Ce 'launch' positionne la variable 'rosparam' 'robot_description' qui contiendra le contenu du fichier URDF. C'est une convention, on doit toujours la nommée 'robot_description'.
Le paramètre 'use_gui' nous permettra d'avoir une interface à base de 'sliders' pour modifier la valeur articulaire des joints.
Les 'nodes' 'joint_state_publisher' (http://wiki.ros.org/joint_state_publisher) publiera les valeurs articulaires sur le topic '/joint_states' et 'robot_state_publisher' (http://wiki.ros.org/robot_state_publisher) publiera les transformations entre 'link' sur le topic '/tf'.

Lancer le fichier 'launch' :

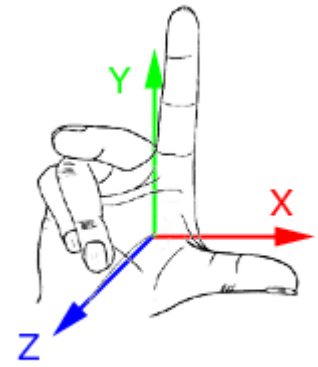
```
roslaunch my_first_urdf display_simple.launch
```

5- Modifier la 'config' de RVIZ.

Rviz se lance et on doit faire quelques ajouts/modifications à l'interface par défaut :
→ Modifier le paramètre 'Fixed Frame' à 'base_link'
→ Ajouter les outils 'Robot State' et 'Robot Model' en cliquant sur le bouton 'Add' en bas à gauche. (http://wiki.ros.org/rviz/DisplayTypes/RobotModel)
On peut à ce stade faire bouger les 'sliders' de l'interface 'Joint State Publisher' lancée séparément de rviz.

Axes de RVIZ :
→ Rouge (Red) = X
→ Vert (Green) = Y
→ Bleu (Blue) = Z

RPY = Rotations (valeurs exprimées en radians) :
→ **R**oll = Autour de l'axe X
→ **P**itch = Autour de l'axe Y
→ **Y**aw = Autour de l'axe Z



Repère de la main droite

Sauvegarder la 'config' actuelle de rviz (Menu File/Save Config As) dans un fichier nommé 'config.rviz' dans le dossier 'rviz' du paquet.

Ajouter le fichier 'config.rviz' dans le fichier launch 'display_simple.launch' :

```
<launch>
	<arg name="model" default="planar_3dof.urdf"/>
	<param name="robot_description" command="$(find xacro)/xacro.py '$(find my_first_urdf)/urdf/$(arg model)'" />
	<param name="use_gui" value="true"/>
	<arg name="rvizconfig" default="$(find my_first_urdf)/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>
```

6- Utiliser TF.

Observer le contenu des topics '/tf' et '/joint_states' pendant que le robot bouge.

Observer la translation et rotation entre deux 'frames tf' (ici une 'frame' est équivalent à un 'link') :

```
rosrun tf tf_echo <source_frame> <target_frame>
rosrun tf tf_echo base_link link_1
rosrun tf tf_echo base_link link_2
```

Voir graphiquement les relations entre 'frames' :

```
rosrun tf view_frames
```

(Ceci va générer dans le dossier courant le fichier 'frames.pdf')

Puis lire le fichier 'frames.pdf' généré précédemment :

```
evince frames.pdf
```

7- Utiliser TF en C++ dans un 'node'.

L'objectif est de créer un 'node' nommé 'tf_listener' qui souscrit automatiquement au topic '/tf' et qui calcule la transformation entre 2 'link' passés en paramètres ('argv[1]' et 'argv[2]') et affiche le résultat.

Créer un nouveau paquet nommé 'my_first_tf' qui dépendra des paquets 'roscpp', 'geometry_msgs' et 'tf' :

```
catkin_create_pkg my_first_tf roscpp geometry_msgs tf
```

Créer un fichier C++ nommé 'tf_listener.cpp' contenant ceci :

```
#include <ros/ros.h>
#include <tf/tf.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/TransformStamped.h>

int main(int argc, char **argv) {

   //Set up the node and nodehandle.
   ros::init(argc, argv, "tf_listener");
   ros::NodeHandle nh;
   ROS_INFO_STREAM("Started node tf_listener.");

   std::string target_frame, source_frame;
   tf::TransformListener tf_listener;
   tf::StampedTransform transform;
   ros::Rate rate(1.); //used to throttle execution

   if (argc < 3) {//if no arguments are given, use base_link and part as the default frame names
      target_frame = "/base_link";
      source_frame = "/gripper";
   }
   else {//if there are at least 2 arguments given, use them as frame names
      target_frame = argv[1];
      source_frame = argv[2];
   }

   while (ros::ok()) {
      try {
         //find transform from target TO source (naming is a bit confusing),
         //at the current time (ros::Time()), and assign result to transform.
         tf_listener.lookupTransform(target_frame, source_frame, ros::Time(), transform);

         //copy transform to a msg type to utilize stringstream properties
         //show transform, and distance between two transforms
         geometry_msgs::Transform buffer;
         tf::transformTFToMsg(transform, buffer);
         ROS_INFO_STREAM("Transform from " << target_frame << " to " << source_frame << ": " << std::endl << buffer);

         // Get 'origin' of Vector Translation of tansform
         tf::Vector3 vect_origin = transform.getOrigin();

         // Get length of Vector translation
         double length = vect_origin.length();
         ROS_INFO_STREAM("Distance between transforms: " << length << " meters.");

        // Get Roll, Pitch, Yaw from transform
        tf::Matrix3x3 matRot = transform.getBasis();
        double roll, pitch, yaw;
        matRot.getRPY(roll, pitch, yaw);
        ROS_INFO_STREAM("Roll : " << roll << ", Pitch : " << pitch << ", Yaw : " << yaw);

      }
```

```
    catch (...) {//assume that the exception thrown is because transform is not available yet
      ROS_WARN_STREAM("Waiting for transform from " << target_frame << " to " << source_frame);
    }
    rate.sleep();   //throttle execution (1 second default)
  }
  return 0;
}
```

Ici l'objet '**tf::TransformListener**' permet automatiquement de souscrire au topic '**/tf**'.
La méthode '**lookupTransform**' permet de calculer la transformation entre 2 'tf_frames' à un instant donné, le résultat étant stocké dans un objet '**tf::StampedTransform**'.
La méthode '**transformTF'ToMsg**' est uniquement utilisée pour convertir la transformation obtenue dans un message ros, afin d'afficher son contenu via un 'ros stream'.
Le message 'Transform' de 'geometry_msgs' contient 2 valeurs :

```
Vector3 translation
Quaternion rotation
```

Modifier le fichier 'CMakelists.txt' afin de prendre en compte ce fichier.

Compiler et exécuter ce 'node' avec différents paramètres :

```
rosrun my_first_tf tf_listener
rosrun my_first_tf tf_listener /base_link /link_2
rosrun my_first_tf tf_listener /link_1 /gripper
```

8- Utiliser 'xacro'.

Afin de simplifier la création de gros robots demandant plusieurs fichiers 'Urdf', dont plusieurs parties peuvent être communes (décrire un bras gauche et un bras droit), il est possible d'utiliser des 'macros XML' qui se nomment 'Xacro'.

On peut définir des macros, mais aussi des propriétés sur lesquels on peut faire de simples calculs mathématiques.

```
<!-- use xacro property -->
<xacro:property name="width" value=".2" />
<xacro:property name="bodylen" value=".6" />
<link name="my_link">
        <visual>
                <geometry>
                        <cylinder radius="${width*2}" length="${bodylen+0.7}"/>
                </geometry>
        </visual>
</link>

<!-- basic macro -->
<xacro:macro name="default_origin">
        <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:macro>

<!-- use macro -->
<xacro:default_origin />
```

Les fichiers 'xacro' se terminent par l'extension '.urdf.xacro' et sont convertit en fichier 'urdf' par le programme python 'xacro.py' du paquet 'xacro' (cf la commande dans les 'launch' précédents).

```
rosrun xacro xacro.py model.xacro > model.urdf
```

Notre objectif est de créer une macro 'xacro' pour définir notre robot 'planar'. Ce qui nous permettra de placer dans 'rviz' par exemple : 2 robots 'planar' ou on le souhaite, sans avoir à doubler le code 'urdf'.

Créer dans le dossier 'urdf' du paquet 'my_first_urdf' un fichier 'xacro' nommé 'planar_3dof.urdf.xacro' :

```
<?xml version="1.0"?>
<robot name="planar_3dof" xmlns:xacro="http://www.ros.org/wiki/xacro">

 <!-- define variables -->
 <xacro:property name="pi" value="3.1415926535897931" />
 <xacro:property name="grey_rgba" value="0.7 0.7 0.7 1.0"/>
 <xacro:property name="velocity" value="0.5"/>
 <xacro:property name="effort" value="0"/>
 <xacro:property name="link_length" value="0.5"/>

 <xacro:macro name="a_planar" params="parent name *origin">

            <link name="${name}_link_1">
               <visual>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/visual/arm_link.stl"/>
                 </geometry>
                 <material name="grey">
                        <color rgba="${grey_rgba}"/>
                 </material>
               </visual>
               <collision>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/collision/arm_link.stl"/>
                 </geometry>
               </collision>
            </link>

             <link name="${name}_link_2">
               <visual>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/visual/arm_link.stl"/>
                 </geometry>
                 <material name="grey"/>
               </visual>
               <collision>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/collision/arm_link.stl"/>
                 </geometry>
               </collision>
             </link>

             <link name="${name}_gripper">
               <visual>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/visual/gripper.stl"/>
                 </geometry>
                 <material name="grey"/>
               </visual>
               <collision>
                 <geometry>
                        <mesh filename="package://my_first_urdf/meshes/collision/gripper.stl"/>
                 </geometry>
               </collision>
             </link>
```

```
            <joint name="${name}_joint_1" type="revolute">
              <insert_block name="origin"/>
              <origin xyz="0 0 0" rpy="0 0 0"/>
              <parent link="${parent}"/>
              <child link="${name}_link_1"/>
              <axis xyz="0 0 1"/>
              <limit lower="${-pi/2.0}" upper="${pi/2.0}" effort="${effort}" velocity="${velocity}"/>
            </joint>

            <joint name="${name}_joint_2" type="revolute">
              <origin xyz="${link_length} 0 0" rpy="0 0 0"/>
              <parent link="${name}_link_1"/>
              <child link="${name}_link_2"/>
              <axis xyz="0 0 1"/>
              <limit lower="${-pi/2.0}" upper="${pi/2.0}" effort="${effort}" velocity="${velocity}"/>
            </joint>

            <joint name="${name}_joint_3" type="revolute">
              <origin xyz="${link_length} 0 0" rpy="0 0 0"/>
              <parent link="${name}_link_2"/>
              <child link="${name}_gripper"/>
              <axis xyz="0 0 1"/>
              <limit lower="${-pi/2.0}" upper="${pi/2.0}" effort="${effort}" velocity="${velocity}"/>
            </joint>


 </xacro:macro>

</robot>
```

Ici on définit une macro '**a_planar**' à laquelle on passe 2 paramètres simple 'parent' et 'name' et 1 paramètre 'origin' plus compliqué (d'où le '*').
'name' = nom du robot (concaténé aux 'link' et 'joint' permet de les rendre unique).
'parent' = nom du 'link' virtuel : ce sera le fameux 'base_link'
'origin' = origin du robot (sera composé d'une translation et rotation).

Créer un fichier 'double_planar_3dof.urdf.xacro' qui utilisera la macro 'planar' et définira plusieurs robots :

```
<?xml version="1.0"?>
<robot name="double_planar" xmlns:xacro="http://www.ros.org/wiki/xacro">
        <link name="base_link"/>

        <!-- Include planar model -->
        <xacro:include filename="$(find my_first_urdf)/urdf/planar_3dof.urdf.xacro"/>

        <!-- using the models -->
        <xacro:a_planar parent="base_link" name="planar_left">
                <origin xyz="-0.5 0 0" rpy="0 -${pi/2.0} 0"/>
        </xacro:a_planar>

        <xacro:a_planar parent="base_link" name="planar_right">
                <origin xyz="0.5 0 0" rpy="0 -${pi/2.0} 0"/>
        </xacro:a_planar>

</robot>
```

Ici on définit 2 robots 'planar_left' et 'planar_right' avec des poses différentes.

Modifier le fichier 'launch' 'display.launch' pour prendre en compte le fichier 'double_planar_3dof.urdf.xacro'.

Lancer le 'launch'. Observer le contenu du topic '/joint_states'.

9- Utiliser les 'namespace' pour séparer les '/joint_states' des 2 robots.

Notre objectif est d'avoir un topic '**/left/joint_states**' pour le robot '**planer_left**' et un topic '**/right/joint_states**' pour le robot '**planar_right**'. De même on souhaite avoir deux fenêtres différentes pour les 'sliders'.

Séparer les définitions 'urdf' des 2 robots, dans 2 fichiers nommés 'planar_left.urdf.xacro' et 'planar_right.urdf.xacro' :

```
<?xml version="1.0"?>
<robot name="robot_planar_left" xmlns:xacro="http://www.ros.org/wiki/xacro">
	<link name="base_link"/>

	<!-- Include planar model -->
	<xacro:include filename="$(find my_first_urdf)/urdf/planar_3dof.urdf.xacro"/>

	<xacro:a_planar parent="base_link" name="planar_left">
		<origin xyz="-0.5 0 0" rpy="0 -${pi/2.0} 0"/>
	</xacro:a_planar>
</robot>
```

```
<?xml version="1.0"?>
<robot name="robot_planar_right" xmlns:xacro="http://www.ros.org/wiki/xacro">
	<link name="base_link"/>

	<!-- Include planar model -->
	<xacro:include filename="$(find my_first_urdf)/urdf/planar_3dof.urdf.xacro"/>

	<xacro:a_planar parent="base_link" name="planar_right">
		<origin xyz="0.5 0 0" rpy="0 -${pi/2.0} 0"/>
	</xacro:a_planar>
</robot>
```

Créer un nouveau fichier 'launch' nommé 'display_double_namespace.launch' :

```
<launch>
  <arg name="model_left" default="planar_left.urdf.xacro"/>
  <arg name="model_right" default="planar_right.urdf.xacro"/>
  <arg name="rvizconfig" default="$(find my_first_urdf)/rviz/config.rviz" />

  <param name="robot_description" command="$(find xacro)/xacro.py '$(find
my_first_urdf)/urdf/double_planar_3dof.urdf.xacro'" />
  <param name="left/robot_description" command="$(find xacro)/xacro.py '$(find my_first_urdf)/urdf/$(arg model_left)'" />
  <param name="right/robot_description" command="$(find xacro)/xacro.py '$(find my_first_urdf)/urdf/$(arg model_right)'" />

  <group ns="left">
	<param name="use_gui" value="true"/>
	<node name="joint_state_publisher_left" pkg="joint_state_publisher" type="joint_state_publisher" />
	<node name="robot_state_publisher_left" pkg="robot_state_publisher" type="state_publisher" />
  </group>

  <group ns="right">
	<param name="use_gui" value="true"/>
	<node name="joint_state_publisher_right" pkg="joint_state_publisher" type="joint_state_publisher" />
	<node name="robot_state_publisher_right" pkg="robot_state_publisher" type="state_publisher" />
  </group>

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

Ici le contenu complet des 2 robots ('planar_left' et 'planar_right') est chargé dans la variable '**robot_description**' qui sera stockée dans 'rosparam' et utilisée par défaut dans 'rviz'.

Les 'nodes' 'joint_state_publisher' et 'robot_state_publisher' utilisent par défaut la variable nommée '**robot_description**' pour connaître la chaîne cinématique d'un robot.

Par conséquent, l'utilisation d'un 'namespace' différent pour le 'planar_left' et le 'planar_right' implique la création des 2 variables '**left/robot_description**' et '**right/robot_description**' stockées dans 'rosparam'.

Exemple : le 'node' 'joint_state_publisher' associé au 'namespace' 'left' cherchera par défaut à lire la variable 'robot_description' associée au même namespace 'left', c'est à dire la variable 'left/robot_description'.

Enfin, pour simplifier l'utilisation du même 'namespace' pour le 'joint_state_publisher' et 'robot_state_publisher', on utilise la notion de 'group' avec le paramètre 'ns' correspondant au nom du 'namespace' .

Lancer le 'launch' et observer les 'topics'.