Universidad de Ingeniería y Tecnología

FUNDAMENTOS DE ROBÓTICA

Prof. Oscar E. Ramos Ponce

Guía de Laboratorio 3:

Modelos de Robots usando URDF

Lima - Perú

2017 - 1

Laboratorio 3:
Modelos de Robots usando URDF

1. Objetivos

- Familiarizarse con el formato URDF y sus características
- Modelar un robot utilizando el formato URDF y visualizarlo en RViz

2. URDF

2.1. Introducción

ROS provee paquetes que pueden ser utilizados para construir modelos 3D y comunicarse con estos modelos. El formato URDF (*Unified Robot Description Format*) permite definir modelos de robots así como los sensores y el ambiente de trabajo. Sin embargo, solo aquellos robots que tienen eslabones rígidos conectados mediante articulaciones pueden ser descritos mediante modelos *URDF*. Estos modelos están representados mediante un archivo de tipo `XML`, y como tal, están compuestos por etiquetas de *XML* especiales que pueden ser leídas para extracción de información. A la lectura del archivo para extraer la información importante del modelo se denomina *parsing* y al programa o función que lo realiza, *parser*.
Algunos paquetes y herramientas útiles para la interacción con modelos URDF son los siguientes:

- `joint_state_publisher`. Este paquete contiene un nodo del mismo nombre que lee la descripción del modelo del robot, encuentra todas las articulaciones (*joints*) y publica los valore articulares para todas las articulaciones movibles usando *sliders*.
- `robot_state_publisher`. Este paquete lee el estado de las articulaciones del robot y publica las posiciones y orientaciones 3D de cada eslabón usando la cinemática obtenida a partir del *URDF*. La posición 3D del robot se publica como una transformación (`tf`) que define las relaciones entre los sistemas de coordenadas.
- `xacro`. Significa *Xml mACROs* y es un formato que permite utilizar variables y otros *add-ons* para la generación de modelos complejos en formato URDF. De esta forma los modelos pueden ser más fáciles de entender y más mantenibles.

2.2. Modelado de un robot con URDF

Un modelo *URDF* puede representar la descripción cinemática y dinámica de un robot, su representación visual, y el modelo de colisión. Al igual que cualquier archivo `xml` está basado en etiquetas. Las etiquetas comúnmente utilizadas en *URDF* son las siguientes.

1. `link`. Representa un solo eslabón del robot y permite especificar sus propiedades, como tamaño, forma, color o una malla 3D compleja importada. También se puede proveer propiedades dinámicas al eslabón como la matriz de inercia y propiedades de colisión. La sintaxis es la siguiente:

link

```
<link name="nombre del óeslabn">
   <inertial>...........</inertial>
   <visual> ............</visual>
   <collision>..........</collision>
</link>
```

La sección visual representa el eslabón real del robot y el área alrededor del eslabón es la sección de colisión que se utiliza para detectar colisiones antes de chocar con el eslabón real. La Figura 1 da una idea de esto.

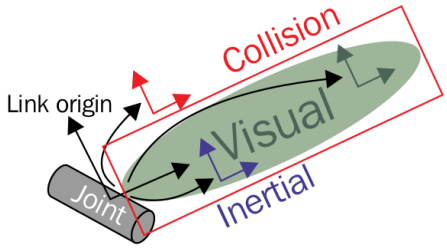

Figura 1: Representación de un eslabón en URDF

2. `joint`. Esta etiqueta representa las articulaciones del robot. Se puede especificar la cinemática y dinámica de la articulación, así como establecer los límites del movimiento articular y de su velocidad. Esta etiqueta soporta articulaciones de revolución, contínuas, prismáticas, fijas y flotantes. La sintaxis es la siguiente:

joint

```
<joint name="nombre de la articulacion">
   <parent link="link1"/>
   <child link="link2"/>
   <calibration .... />
   <dynamics damping ..../>
   <limit effort .... />
</joint>
```

Una articulación queda definida cuando se especifica los eslabones que une: el primero es llamado el padre (*parent*) y el segundo el hijo (*child*). Una ilustración de esta relación se muestra en la Figura 2

3. `robot`. Esta etiqueta encapsula todo el modelo del robot. Dentro de esta etiqueta se define el nombre del robot, los eslabones y las articulaciones del robot. La sintaxis es la siguiente:

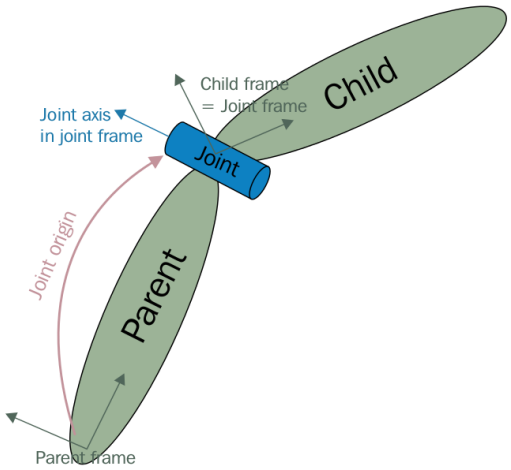

Figura 2: Representación de una articulación en URDF

robot

```
<robot name="<name of the robot>"
  <link> ..... </link>
  <link> ..... </link>
  <joint> ..... </joint>
  <joint> ........</joint>
</robot>
```

Un modelo de robot consiste de articulaciones y eslabones conectados. Un ejemplo se muestra en la Figura 3

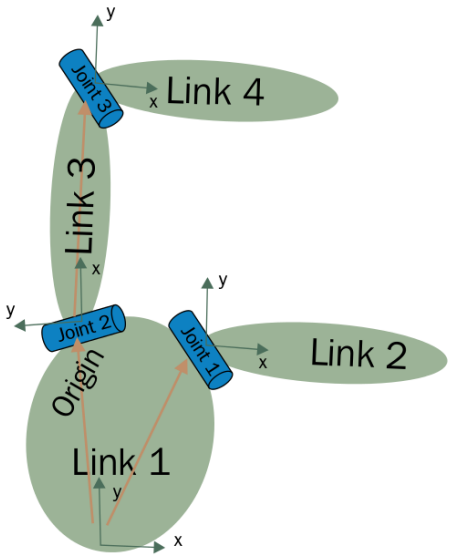

Figura 3: Representación de un eslabón en URDF

4. `gazebo`. Esta etiqueta se utiliza cuando se incluye parámetros de simulación para el simulador llamado *Gazebo*. Se puede usar esta etiqueta para incluir plugins de gazebo, material, propiedades, entre otros.

Se puede encontrar mucha más información sobre etiquetas de URDF en el wiki de URDF en ROS: `http://wiki.ros.org/urdf/XML`. De igual manera, más información sobre las etiquetas de los eslabones está disponible en `http://wiki.ros.org/urdf/XML/link`.

3. Robot en URDF

Dentro del espacio de trabajo `lab_ws` crear un paquete llamado `lab3` y dentro de este paquete crear una carpeta llamada `urdf`. Descargar el archivo llamado `robot1.urdf` dentro de la carpeta `urdf`. Abrir el archivo e inspeccionarlo. Las etiquetas que contiene son las mencionadas anteriormente: `robot`, `link` y `joint`.

Las etiquetas de tipo `<visual>` describen la apariencia del eslabón y son las que se mostrarán en la simulación del robot. Dentro de estas etiquetas se define la geometría del robot como cilindros (`cylinder`), cubos (`box`), esferas (`sphere`) o mallas 3D (`mesh`) creadas con programas de diseño mecánico (de preferencia en formato .stl). Además se puede especificar el material: color (`color`) y textura (`texture`).

Preguntas. Abrir el modelo urdf `robot1.urdf`, inspeccionarlo y responder.

1. ¿Cuál es el nombre del robot?
2. ¿Cuántos eslabones tiene el robot y cuáles son sus nombres?
3. ¿Cuántas articulaciones tiene, cuáles son sus nombres, y de qué tipo son?
4. Para cada articulación, ¿cuáles son sus límites articulares y límites de velocidad?
5. ¿Qué par de eslabones une cada articulación?

Cuando se tiene un urdf nuevo, es posible verificar si el mismo está correctamente especificado o si presenta algún problema de sintaxis usando el comando `check_urdf`. Para verificar el modelo `robot1.urdf` ejecutar los siguientes comandos desde un terminal:

```
$ roscd lab3/urdf
$ check_urdf robot1.urdf
```

Este programa realiza el *parsing* del modelo URDF y muestra `OK` si ningún problema es detectado. Para el caso de robots más complejos resulta útil ver la estructura de los eslabones del robot y sus articulaciones de manera gráfica. Para ello, se puede utilizar el comando llamado `urdf_to_graphiz`:

```
$ roscd lab3/urdf
$ urdf_to_graphiz robot1.urdf
```

Este comando genera un pdf con el nombre del robot (que puede no ser igual al nombre del archivo urdf, como en este caso). El pdf generado puede ser abierto con cualquier programa. Abrir el pdf e inspeccionarlo. Para mayor información sobre URDFs se puede ver `http://wiki.ros.org/urdf`

4. Visualización del Modelo con RViz

Luego de diseñar el modelo del robot en URDF, éste puede ser visualizado en RViz. Crear una carpeta llamada `launch` y una carpeta llamada `config` dentro del paquete `lab3`. Dentro de la carpeta `launch` crear un archivo llamado `robot1.launch` con el siguiente contenido:

robot1.launch

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

  <arg name="model" />
  <param name="robot_description" textfile="$(find lab3)/urdf/robot1.urdf" />
  <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="-d $(find lab3)/config/
      robot1config.rviz" required="true" />

</launch>
```

Luego de creado este archivo, visualizar el modelo del robot usando el comando (pero tener en cuenta que no se mostrará nada, aún):

```
$ roslaunch lab3 robot1.launch
```

Por defecto, no hay ningún robot agregado en RViz. Para visualizar el robot adecuadamente seguir los siguientes pasos:

1. Agregar un modelo de robot: `Add` → `RobotModel` → `OK`.
2. Especificar el sistema base de referencia: en la lista desplegable correspondiente a `Fixed Frame` escoger `base_link`.

Verificar usando los 2 *sliders* que los límites articulares sean los especificados en el modelo URDF y verificar que los nombres de las articulaciones (en los *sliders*) coincida con los especificados en el modelo. Luego, para guardar la configuración actual hacer clic en: `File` → `Save Config`. Para ver los sistemas de referencia, se puede añadir `Add` → `TF` → `OK`.

Ejercicios y Preguntas. Crear un nuevo modelo URDF llamado `robot2.urdf` y un archivo de lanzamiento `robot2.launch` similares a los anteriores. En esta parte se trabajará con el archivo urdf creado (`robot2.urdf`).

1. Cambiar el nombre del robot (no el nombre del archivo) a `mirobot2`. Cambiar los nombres de las articulaciones a `joint1` y `joint2`. Dejar el eslabón `base_link` con ese mismo nombre pero cambiar el nombre de los otros eslabones a `link1` y `link2`. Consecuentemente, modificar también los nombres de los eslabones dentro de las definiciones de las articulaciones.

2. Añadir un eslabón llamado `linkcirc` de tipo `sphere` (en el cual solo se especifica el radio [`radius`]). Además, añadir una articulación llamada `joint3` de tipo `fixed` que una los eslabones `link2` con `linkcirc`.

3. Realizar las modificaciones necesarias en el modelo de tal modo que el modelo final luzca como el mostrado en la Figura 4 (nota: el color del eslabón final tiene los siguientes componentes RGB: 0.6, 0.2, 0.6). La nueva longitud del eslabón 1 es 0.6.

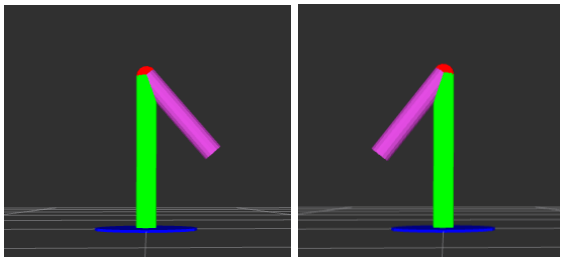

Figura 4: Modelo deseado de robot2.urdf

4. Indicar las modificaciones que han sido realizadas. Adjuntar el código de `robot2.urdf` o copiarlo en el reporte.

5. Añadir capturas de pantalla del modelo obtenido.

5. Uso de xacro para Crear Modelos URDF

La flexibilidad de URDF se reduce al trabajar con modelos complejos de robots. La descripción del modelo usando xacro permite crear macros que pueden ser reutilizadas, variables y algunas operaciones matemáticas.

- *Propiedades*: Son variables asociadas a un valor y se definen de la siguiente manera: `<xacro:property name="nombre_var"value="0.5/>`. Luego de definida esta propiedad, para recuperar su valor se utiliza: `${nombre_var}`.

- *Operaciones Matemáticas*: Se puede realizar algunas operaciones matemáticas elementales; por ejemplo: `${nombre_var+0.5}`.

- *Macros*: Son como funciones que pueden tener parámetros de referencia. La siguiente es una macro para definir valores inerciales

```
<xacro:macro name="inertial_matrix" params="mass">
<inertial>
    <mass value="${mass}" />
        <inertia ixx="0.5" ixy="0.0" ixz="0.0"
                 iyy="0.5" iyz="0.0" izz="0.5" />
    </inertial>
</xacro:macro>
```

  Y luego de declarada la macro, ésta puede ser utilizada desde el archivo de definición del modelo como: `<xacro:inertial_matrix mass="1/>`

Preguntas. Descargar el modelo xacro `robot1b.xacro` y guardarlo dentro de la carpeta `urdf` del paquete `lab3`. Abrir el archivo xacro, analizarlo y responder.

1. Indicar todas las diferencias entre el modelo xacro (`robot1b.xacro`) y el modelo URDF usado anteriormente (`robot1.urdf`)

2. En este caso, ¿sería más conveniente utilizar *xacro* o *urdf* directamente? Mencionar el (los) motivo(s).

5.1. Conversión de xacro a URDF

Los archivos xacro pueden ser convertidos en URDF utilizando el script llamado xacro.py de la siguiente manera:

```
$ roscd lab3/urdf
$ rosrun xacro xacro.py robot1b.xacro > robot1b.urdf
```

Pregunta. Abrir el archivo resultante `robot1b.urdf` y compararlo con el archivo xacro original `robot1b.xacro`. ¿Qué diferencias hay entre ambos archivos?

5.2. Visualización en RViz

Dentro de la carpeta `launch` crear un archivo llamado `robot1b.launch` con el contenido mostrado a continuación. Este archivo será utilizado para visualizar el modelo.

robot1b.launch

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

<launch>
  <arg name="model" />
  <param name="robot_description" command="$(find xacro)/xacro.py $(find lab3)/
      urdf/robot1b.xacro" />
  <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="-d $(find lab3)/config/
      robot1bconfig.rviz" required="true" />
</launch>
```

Ejecutar el archivo creado usando `roslaunch` como en el caso anterior pero, obviamente, usando `robot1b.launch` en lugar de `robot1.launch`.

Pregunta. ¿Cuál es la diferencia principal en los archivos de tipo *launch* cuando se usa directamente el modelo *urdf* y cuando se usa el modelo especificado en formato *xacro*?

6. Modelo en Gazebo

Mientras que RViz es utilizado solo para mostrar el modelo del robot, Gazebo es un programa que permite la simulación dinámica del robot. Debido a ello, para que un modelo pueda ser simulado usando Gazebo es necesario añadir, además de la información puramente geométrica (usada en RViz), información relacionada con las propiedades dinámicas de cada elemento.

6.1. Algunos Elementos de Gazebo

Algunos de estos elementos adicionales que deben ser añadidos son los siguientes.

Color. El color en Gazebo no está relacionado con el color en RViz y debe ser especificado por etiquetas especiales. Por ejemplo:

```
<gazebo reference="base_link">
  <material>Gazebo/White</material>
</gazebo>
```

Transmisión. Para accionar el robot utilizando controladores de ROS, se debe definir el elemento `transmission` que relacionará los actuadores con las articulaciones. Esto se realiza de la siguiente manera:

```
<transmission name="tran1">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint1">
        <hardwareInterface>PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor1">
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

En este archivo, `joint name` denota la articulación a la cual se le asocia un actuador. La etiqueta `type` hace referencia al tipo de transmisión, en este caso es una transmisión simple (`SimpleTransmission`). El elemento `hardwareInterface` es el tipo de interface de hardware que se utiliza en el robot y puede ser de posición, velocidad o fuerza (position, velocity, effort). Esta interface de hardware es cargada desde el plugin de *gazebo_ros_control*. Es posible además especificar la reducción mecánica.

Plugin de Control. El control se especifica mediante el plugin de `gazebo_ros_control`, el cual asigna las interfaces de hardware necesarias y el controlador. El plugin se añade de la siguiente manera:

```
<gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
        <robotNamespace>/robot1</robotNamespace>
    </plugin>
</gazebo>
```

6.2. Visualización del Modelo en Gazebo

Descargar el archivo `robot1gazebo.urdf` el cual contiene las etiquetas antes mencionadas para gazebo además de otras etiquetas adicionales. Este modelo debe estar dentro de la carpeta `urdf`. Descargar además el archivo `robot1gazebo.launch` dentro de la carpeta `launch`. Visualizar el modelo con los comandos:

```
$ roslaunch lab3 robot1gazebo.launch
```

Notar que el modelo puede tender a moverse hacia algún lado debido a los parámetros dinámicos asignados a cada elemento.

Pregunta. Identificar y mencionar las diferencias principales entre el archivo de gazebo `robot1gazebo.urdf` y el modelo puramente geométrico `robot1.urdf`. Es decir, indicar aquellos elementos `nuevos` que aparecen en el modelo de Gazebo.

6.3. Movimiento del Robot en Gazebo

Para mover cada articulación se añade un controlador de ROS, el cual está indicado en la etiqueta `transmission`. Este controlador consiste principalmente de un mecanismo realimentado (normalmente un bucle PID), el cual recibe una referencia y controla la salida usando la realimentación de los actuadores. El controlador de ROS se comunica con la *interface de hardware*. La función principal de la interface de hardware es actuar como sistema mediador entre los controladores de ROS y el hardware real o el simulador. De este modo, el mismo controlador y el mismo código puede accionar un robot simulado o un robot real.

Para realizar la interface entre los controladores se define un archivo de tipo `yaml` (que es un estándar de archivos con un formato específico). Dentro de la carpeta `config`, crear un archivo llamado `robot1control.yaml` y copiar el siguiente contenido:

robot1control.yaml

```
robot1:
  # Publish all joint states -----------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------
  joint1_position_controller:
    type: position_controllers/JointPositionController
    joint: pan_joint
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: position_controllers/JointPositionController
    joint: tilt_joint
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

En este archivo `yaml` se especifica el tipo de controlador que se utilizará para cada articulación, así como los valores proporcional, integral y derivativo de cada controlador de bajo nivel.
Para realizar la simulación dinámica del modelo creado en Gazebo, descargar el archivo `robot1gazebo_control.launch` en la carpeta `launch` y ejecutarlo. Se debe observar el robot en Gazebo. Inspeccionar los tópicos existentes con:

```
$ rostopic list
```

Se puede luego mover las articulaciones con posiciones arbitrarias. Para ello, ingresar el siguiente comando:

```
$ rostopic pub /robot1/joint1_position_controller/command std_msgs/Float64 "0.5"
```

De igual manera, se puede enviar un comando articular a la otra articulación. Además, se puede utilizar un programa de Python para enviar el valor deseado a cada articulación. Se puede probar esto creando una carpeta llamada `src` y descargando allí el archivo `send_joints`. No olvidar hacer ejecutable el archivo (con `chmod a+x send_joints`). Se puede ejecutar el programa como

```
$ rosrun lab3 send_joints
```

7. Ejercicio

1. Diseñar la estructura cinemática de un robot de por lo menos 5 grados de libertad en papel, indicando las longitudes de los eslabones, y cómo están orientadas las articulaciones.

2. Implementar un modelo URDF (o xacro) del robot diseñado. Adjuntar un pdf con las especificaciones del modelo (obtenido a partir del modelo urdf con urdf_to_graphiz).
3. Visualizar el robot en RViz
4. Añadir información dinámica al modelo del robot. En el informe de laboratorio mencionar los valores de estos elementos dinámicos para cada eslabón.
5. Visualizar el modelo dinámico del robot en Gazebo.
6. Enviar distintos valores a las articulaciones a partir de un archivo Python.
7. *Opcional*: Realizar el análisis cinemático directo del robot y determinar su cinemática inversa de manera numérica.

8. Conclusiones

Elaborar algunas conclusiones sobre el laboratorio desarrollado e indicarlas en el informe del laboratorio.

Referencias

- Joseph, Lentin. *Mastering ROS for Robotics Programming*. Packt Publishing Ltd. 2015.
- Tutoriales de ROS: `http://wiki.ros.org/ROS/Tutorials`
- Tutoriales de URDF: `http://wiki.ros.org/urdf/Tutorials`