UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

LABORATORIO DE ROBÓTICA INDUSTRIAL

PRÁCTICA # 1

MODELOS DE ROBOTS USANDO URDF

1. OBJETIVO GENERAL

Dar a conocer el formato unificado para la descripción de Robots (URDF), con la visualización en las Herramientas de simulación, para la implementación de estructuras robóticas de cadenas cinemáticas abiertas.

1.1 OBJETIVOS ESPECIFICOS

- Conocer el formato URDF y algunos de sus parámetros más importantes.
- Modelar un robot utilizando el formato URDF y visualizarlo en RViz

2. EQUIPMENT AND MATERIALS

Item	Description	quantity
1	Computer with Linux and ROS Kinetic	1

3. DESARROLLO DEL LABORATORIO

Para el desarrollo adecuado del laboratorio se debe tener en cuenta los siguientes aspectos.

- Las respuestas a las preguntas o tareas de esta sección deben ser desarrolladas en un documento separado utilizando cualquier editor de LaTex, como Overleaf.
- El reporte de laboratorio debe ser presentado solamente en formato pdf a través de Classroom hasta la fecha programada. Solo es necesario que un integrante de cada grupo presente el reporte.
- Se recomienda leer el apéndice A, el cual contiene información genérica del formato URDF, antes de comenzar el laboratorio.
- Para más información sobre el formato URDF, se puede consular su página wiki en: http://wiki.ros.org/urdf.

Para poder trabajar con ROS, se debe primero tener un espacio de trabajo (workspace), el cual contendrá todo aquello que será desarrollado. Para el laboratorio de este curso se creará un espacio de trabajo llamado `lab_ws` en la carpeta del usuario (`/home/user`, donde `user` es el nombre del usuario). Para ello, abrir un terminal e ingresar los siguientes comandos:

```
$ cd ~
$ mkdir -p lab_ws/src
```

Lo que estos comandos hacen es ir a la carpeta del usuario (cd) y allí crear una carpeta (`mkdir`) llamada `lab_ws` que a su vez contiene la carpeta `src`. Una vez creadas estas carpetas, es necesario ir a la carpeta `src`, que será la que contendrá todo el código fuente, e inicializar el nuevo espacio de trabajo utilizando el comando `catkin_init_workspace`, que es propio de ROS. Esto se realiza de la siguiente manera:

```
$ cd lab_ws/src
$ catkin_init_workspace
```

Seguidamente se debe compilar el espacio de trabajo, a pesar de que está (por el momento) vacío, ya que ello creará archivos que son necesarios. Para ello, se va a la carpeta del espacio de trabajo (`lab_ws`) y se usa el comando `catkin_make`

```
$ cd ~/lab_ws
$ catkin_make
```

Una vez ejecutado `catkin_make`, se generan automáticamente dos carpetas denominadas `build` y `devel`. Finalmente, se debe hacer que el espacio de trabajo creado esté visible para ROS. Esto se consigue ejecutando el script llamado setup.bash que se encuentra en la carpeta devel (se ejecuta usando el comando `source`). Para que esta ejecución se realice cada vez que se abre un nuevo terminal, se añade la instrucción de ejecutar este script al `. bashrc` (que es el programa que se ejecuta al abrir un nuevo terminal) del siguiente modo:

$ echo "source ~/lab_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc

Luego de esto, cerrar el terminal y abrir uno nuevo. Para verificar que todo marcha bien, al escribir el comando `roscd` se debe llegar a la dirección `~/lab ws/devel`. Se debe tener en cuenta que todo lo relacionado con código fuente (*source code*) que se vaya a utilizar debe estar en la carpeta `~/lab ws/src`, donde `src` proviene de la palabra *source*.

Luego para poder trabajar, es necesario descargar el repositorio de github que contiene los archivos necesarios (LabRobotica20221.git), utilizando el comando `git clone.` El paquete de ROS llamado lab1 es el que se utilizará durante este laboratorio

$ cd ~/lab_ws/src
$ git clone https://github.com/andresperez86/LabRobotica20221.git

Seguidamente se debe compilar el espacio de trabajo. Para ello, se va a la carpeta del espacio de trabajo (`lab_ws`) y se usa el comando `catkin_make`

3.1. ROBOT EN URDF

Dentro del paquete que se utilizará en este laboratorio (`lab1`) ir a la carpeta `urdf` y abrir el archivo llamado `robot.urdf`. Inspeccionar el contenido de este archivo, el cual contiene las etiquetas `robot, link y joint`, mencionadas en el apéndice A.

Las etiquetas de tipo <`visual`> describen la apariencia del eslabón y son las que se mostrarán en la visualización y en la simulación del robot. Dentro de las etiquetas <`visual`> se define la geometría del robot como cilindros `(cylinder)`, cubos `(box)`, esferas `(sphere)` o mallas 3D `(mesh).` Es importante notar que las longitudes siempre se encuentran en metros. Además, se puede especificar el material a través de su color `(color)` y textura `(texture).`

Preguntas. Considerando el modelo robot.urdf, responder:

1. (0.5 pts) > ¿Cuál es el nombre del robot?
2. (0.5 pts) > ¿Cuántos eslabones tiene el robot y cuáles son sus nombres?
3. (0.5 pts) > ¿Cuántas articulaciones tiene, ¿cuáles son sus nombres, y de qué tipo son?
4. (0.5 pts) > Para cada articulación, ¿Cuáles son sus límites articulares y límites de velocidad?
5. (0.5 pts) > ¿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 lab1/urdf
$ check_urdf robot.urdf
```

Este programa realiza el "parsing" del modelo URDF y muestra OK, así como una estructura simplificada de los eslabones del robot, cuando ningún problema es detectado; de lo contrario, muestra un error.

Para el caso de robots más complejos resulta útil visualizar de manera gráfica la estructura de los eslabones del robot y sus articulaciones. Para ello, se puede utilizar el comando llamado `urdf_to_graphiz` de la siguiente forma:

```
$ roscd lab1/urdf
$ urdf_to_graphiz robot.urdf
```

Este comando crea un archivo en .pdf con el nombre del robot (que podría no ser igual al nombre del archivo urdf, como en este caso). El pdf generado puede ser abierto con cualquier programa que permita la lectura de pdfs (como "Acrobat" en Ubuntu). Abrir el pdf e inspeccionarlo.

Tareas. En el reporte del laboratorio incluir lo siguiente:

1. (0.25 pts) Una captura de pantalla del pdf obtenido al utilizar `urdf to graphiz`
2. (1 pts) Una pequeña descripción de los elementos que contiene este pdf

3.2. VISUALIZACIÓN DEL MODELO CON RVIZ

Luego de tener el modelo del robot en URDF, diseñado a partir del fabricante, este puede ser mostrado en RViz. Dentro de la carpeta `launch` del paquete `lab1` existe un archivo llamado `robot.launch` con el siguiente código :

robot.launch

```
<?xml version="1.0"?>
<launch>
        <param name="robot_description" textfile="$(find lab1)/urdf/robot.urdf" />
        <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" />
        <param name="use_gui" value="true"/>
        <node name="rviz" pkg="rviz" type="rviz" args="-d $(find lab1)/config/robotconfig. rviz" re
        quired="true" />
</launch>
```

Este archivo permite visualizar el modelo del robot usando el comando `roslaunch` del siguiente modo:

$ roslaunch lab1 robot.launch

Por defecto, no existe ningún elemento de tipo robot agregado en RViz. Para poder visualizar el robot se debe seguir los siguientes pasos:

1. Agregar un modelo de robot: Add→RobotModel↲OK.

2. Especificar el sistema que se utilizará como sistema de referencia: en la lista desplegable correspondiente a `Fixed_Frame` → `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.`

Si se desea visualizar los sistemas de referencia asociados con cada uno de los eslabones, se puede añadir un elemento de tipo `TF: Add→TF→OK.`

Ejercicios y Preguntas. Utilizando como base el modelo robot.urdf anterior, crear un nuevo modelo llamado robot2.urdf, así como un archivo que permita su visualización, llamado robot2.launch.
Resuelva las siguientes preguntas con el archivo urdf creado (robot2.urdf).

1. (0.5 pts) Cambiar el nombre del robot (no el nombre del archivo) a `robotinutil.` 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. (1 pts)Añadir un eslabón llamado `linkcircular` 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 `linkcircular.`

3. (1 pts)Realizar las modificaciones necesarias en el modelo de tal modo que el modelo final luzca como el mostrado en la Figura 1 (nota: el color del eslabón final tiene los siguientes componentes RGB: 0.6, 0.2, 0.6). Además, la nueva longitud del eslabón 1 (`link1`) es 0.6 m.

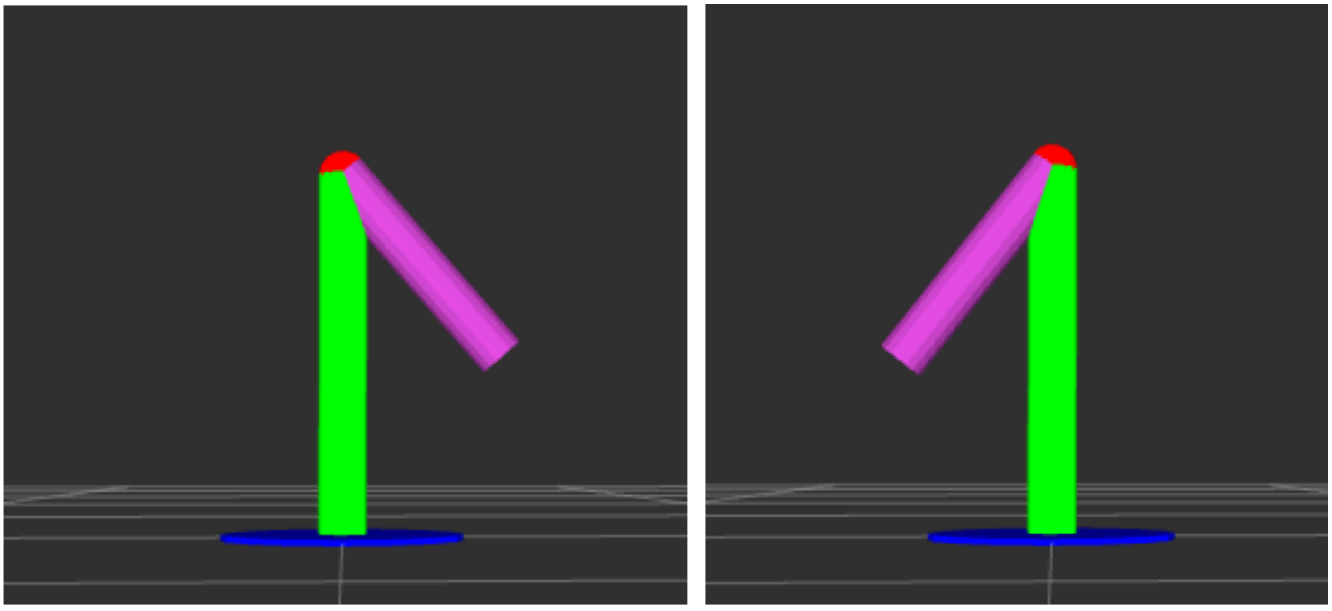

Figura 1

4. (3 pts) En el reporte, copiar el código de `robot2.urdf`, indicando claramente las modificaciones que han sido realizadas.

5. (1 pt) Añadir capturas de pantalla del modelo obtenido, en RViz, con ángulos diferentes a los mostrados en la figura anterior

3.3. XACRO PARA LA CREACIÓN DE MODELOS URDF

El formato URDF, solamente permite definir valores numéricos constantes para longitudes, masas, inercias, etc. Así, si se quiere, por ejemplo, ingresar la misma longitud para dos eslabones, es necesario especificar el mismo número dos veces, y si uno cambia, se debe manualmente cambiar el otro (en caso de que se desee siempre la misma longitud). De igual manera, si se desea especificar la mitad de un valor, es necesario ingresar dicho valor de forma manual, como una constante. Esto reduce la flexibilidad de URDF para trabajar con modelos complejos de robots.
Para solucionar estos problemas, se creó el formato xacro, el cual permite crear variables (llamadas propiedades), macros que pueden ser reutilizadas, así como algunas operaciones matemáticas. La siguiente es una breve descripción de estos elementos añadidos en xacro:

- *Propiedades:* Son variables asociadas a un valor y se definen de la siguiente manera: `<xacro:property name="nombre_var" value="0.5" />`. Luego, 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:* Se comportan como funciones, y admiten parámetros de referencia como entradas. Por ejemplo, la siguiente es una macro que toma como parámetro de entrada la masa`(mass)` y crea una etiqueta de valores inerciales con la masa ingresada y con un tensor de inercia predefinido.

```
<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>
```

Una vez declarada la macro, esta puede ser utilizada desde el archivo de definición del modelo, que podría ser el mismo archivo donde se definió la macro, u otro archivo.
En este ejemplo, la macro puede ser instanciada como:
`<xacro:inertial_matrix mass="1"/>.`

Luego, a partir del formato xacro es posible obtener un archivo con formato urdf como se verá más adelante.

Preguntas. Abrir el archivo `robotx.xacro` que se encuentra en la carpeta `urdf` del paquete `lab1`, analizarlo y responder lo siguiente.

1. (0.75 pt) Indicar todas las diferencias entre el modelo xacro (`robotx.xacro`) y el modelo URDF usado anteriormente (`robot.urdf`)
2. (1 pts) Indicar qué ventajas podría tener el uso del formato *xacro* en lugar del formato *urdf* directamente.

Conversión de xacro a URDF

Los archivos con formato de tipo xacro pueden ser fácilmente convertidos a formato URDF utilizando el script llamado `xacro.py`. Por ejemplo, para convertir a urdf el archivo xacro `robotx.xacro`, se puede utilizar la siguiente secuencia de comandos:

```
$ roscd lab1/urdf
$ rosrun xacro xacro --inorder robot1b.xacro > robot1b.urdf
```

Pregunta. (1 pt) Abrir el archivo resultante `robot1b.urdf` y compararlo con el archivo xacro original `robot1b.xacro`. ¿Qué diferencias hay entre ambos archivos?

Visualización en RViz

Dentro de la carpeta `launch` se encuentra el archivo `robotx.launch` con el contenido mostrado a continuación. Este archivo será utilizado para visualizar el modelo.

robotx.launch

```
<?xml version="1.0"?>
<launch>
        <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
        lab1)/urdf/robotx.xacro" />
        <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" />
        <param name="use_gui" value="true"/>
        <node name="rviz" pkg="rviz" type="rviz" args="-d $(find lab1)/config/robotxconfig.
        rviz" required="true" />
</launch>
```

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

Pregunta. (1 pt) > ¿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?

3.4. MODELO SIMULADO EN GAZEBO

Mientras que RViz se utiliza solo para visualizar 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 incluir, además, de la información puramente geométrica (usada en RViz), también, es necesario agregar la información relacionada con las propiedades dinámicas de cada elemento. Algunos de los elementos que es necesario adicionar para la simulación dinámica se describen en el apéndice B.

3.4.1 Visualización del Modelo en Gazebo

El archivo `urdf/robot gazebo.urdf` contiene las etiquetas mencionadas en el apéndice B, Además, de algunas otras etiquetas adicionales necesarias para el funcionamiento adecuado de Gazebo. Asimismo, el archivo `launch/robot gazebo.launch` contiene el archivo de lanzamiento necesario para ejecutar gazebo y visualizar el modelo urdf dentro de este entorno de simulación. Visualizar el modelo en Gazebo utilizando los siguientes comandos:

$ roslaunch lab1 robot_gazebo.launch

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

Pregunta. (1 pt) Identicar y mencionar las diferencias principales entre el archivo de gazebo `robot_gazebo.urdf` y el modelo puramente geométrico `robot.urdf`. Es decir, indicar aquellos elementos nuevos que aparecen en el modelo de Gazebo.

3.4.2 Movimiento del Robot en Gazebo

Para realizar el movimiento de cada articulación, se añade un controlador de ROS, el cual está indicado en la etiqueta `transmition`. 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 interfaz de hardware. La función principal de la interfaz 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 código puede accionar un robot simulado o un robot real.

Para realizar la interfaz entre los controladores se crea un archivo de tipo `yaml` (que es un estándar de archivos con un formato específico). Dentro de la carpeta `config,` se encuentra el archivo llamado `robot_control.yaml` el cual contiene lo siguiente

Robot_control.yaml

```
robot:
   # 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 define el tipo el de controlador a usar por cada articulación, al igual que las constantes proporcional, integral y derivativa de cada uno de los controladores de cada articulación en un bajo nivel.

Para efectuar la simulación dinámica del modelo construido en Gazebo, correr el archivo `robot_gazebo_control.launch` que se encuentra en la carpeta `launch` del paquete. Se debe notar que aparece el robot en Gazebo.

$rostopic list

Luego, se puede mover las articulaciones con posiciones arbitrarias. Para realizar una prueba de esto, utilizar el comando `rostopic pub`, el cual realiza la publicación de un mensaje en un tópico determinado. En este caso, el tópico será el controlador de la primera articulación:

$ rostopic pub /robot/joint1_position_controller/command std_msgs/Float64 "0.5"
$ rostopic pub /robot/joint1_position_controller/command std_msgs/Float64 "data: 0.5"

Del mismo modo, 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 el envío desde Python utilizando el archivo `send_joints` que se encuentra en la carpeta src. Ejecutar el programa (mientras esté corriendo la simulación de Gazebo) como:

$rosrun lab1 send_joints

Preguntas.
1.(2 pt) Explicar con detalle qué hace el programa send joints
2.(1 pts) Modificar el programa `send joints` para que el robot se mueva a distintas configuraciones articulares. Incluir en el reporte lo medicado y algunas capturas de pantalla de lo obtenido.

4. DESAFÍO

1.(1pts) Diseñar la estructura cinemática de un robot de por lo menos 3 grados de libertad en papel, indicando las longitudes de los eslabones, y cómo están orientadas las articulaciones.
2.(3 pts) 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.(0.5 pts) Visualizar el robot en RViz
4.(0.5 pts) Visualizar el robot en Gazebo, añadiendo la información necesaria al modelo URDF.
5.(2 pts) Controlar en Gazebo las articulaciones del robot diseñaado a partir de un

archivo Python.

5. CONCLUSIONES(5 pts)

Elaborar almenos tres conclusiones sobre el laboratorio desarrollado e indicarlas en el informe del laboratorio

References

[1] Tutoriales de ROS: http://wiki.ros.org/ROS/Tutorials

[2] Tutoriales de URDF: http://wiki.ros.org/urdf/Tutorials

Tutorial de control en Gazebo: http://gazebosim.org/tutorials/?tut=ros control
[3] Tutoriales de ROS: http://wiki.ros.org/ROS/Tutorials

6. Apéndices A.

URDF

A.1 Introducción

ROS (Robot Operating System) provee paquetes que pueden ser utilizados tanto para construir modelos 3D de robots como para comunicarse con estos modelos a través de señales que comanden su movimiento. El formato URDF (Unified Robot Description Format) permite deescribir modelos de robots a través de la especificación de cada eslabón, con sus características físicas (como masa, tensor de inercia, longitudes, etc.), y de cada articulación. Además, se pueden de describir actuadores, diversos sensores, y objetos para un entorno de trabajo. Sin embargo, actualmente el formato URDF está restringido a robots que poseen eslabones rígidos.

Los modelos del formato URDF siguen la convención de los archivos de tipo XML (eXtensible Markup Language), y como tal, están compuestos por etiquetas especiales de XML que pueden ser leídas para la extracción de la información que contienen. A la lectura del archivo para extraer la información importante del modelo se denomina “parsing” y al programa o función que lo realiza, parser. ROS provee herramientas para realizar la extracción de información de un modelo URDF, pero estas herramientas

hacen uso principalmente de C++. 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 (en formato URDF), encuentra todas las articulaciones (joints) y publica los valores articulares (q) para todas las articulaciones móviles usando (sliders.)

robot state publisher. Este paquete lee el estado de las articulaciones del robot (q) y publica las posiciones y orientaciones 3D de cada eslabón (xi) usando la cinemática directa obtenida a partir del modelo URDF. La posición 3D del robot se publica como una transformación (tf) que define las relaciones entre los sistemas de coordenadas.

xacro. Signica 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.

A.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*. Cada etiqueta de este tipo representa un solo eslabón del robot y permite especificar sus propiedades, como tamaño, forma, color, o una malla 3D compleja importada (típicamente en formato .dae o .stl). Cada eslabón puede además contener propiedades dinámicas como su masa o su matriz de inercia, así como diversas propiedades de colisión (normalmente estructuras geométricas simples). La sintaxis de esta etiqueta es la siguiente:

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

La Figura 2 brinda una idea de lo que se entiende por sección visual y de colisión. La sección visual representa el eslabón real del robot y puede estar dado por una malla compleja con bastante detalle que visualmente hace que el modelo sea similar al robot

real. La sección de colisión es el área alrededor de la parte visual, típicamente muy simplificada, que se utiliza para detectar colisiones entre eslabones.

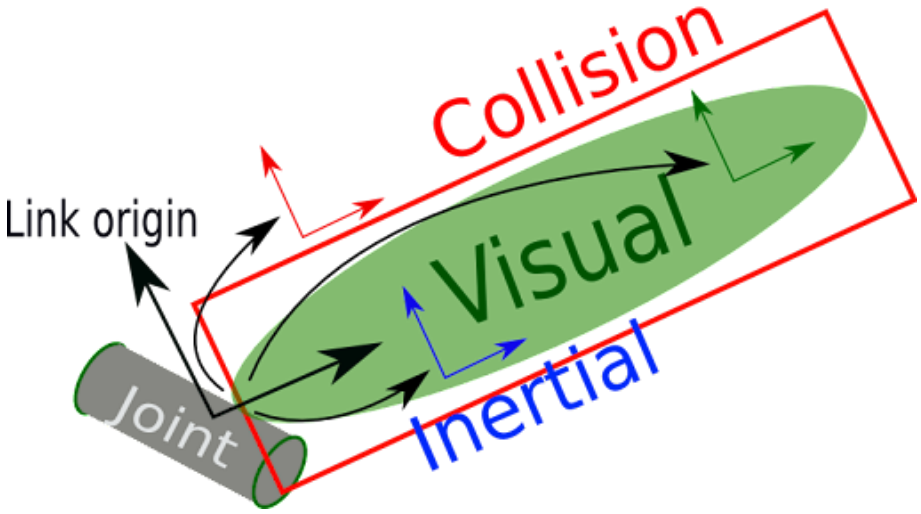

Figura 2 Representación de un eslabón en URDF

2. *joint.* Esta etiqueta representa cada una de las articulaciones del robot. Se puede especificar parámetros relacionados con la cinemática y la dinámica de cada articulación, así como establecer los límites del movimiento articular, de su velocidad, y del máximo esfuerzo mecánico que pueden aplicar (torque máximo). Esta etiqueta soporta articulaciones de revolución, continúas, prismáticas, fijas y flotantes. Las articulaciones fijas estrictamente no son articulaciones ya que no permiten el movimiento, pero se utilizan como un elemento que permite unir de manera fija dos eslabones entre sí. Algunos parámetros de la sintaxis de una articulación se muestran a continuación:

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

La Figura 3 muestra un esquema donde se puede ver lo que se conoce como el eslabón padre y el hijo. Cada articulación une dos eslabones y queda completamente unida solo cuando se especifica ambos eslabones que los une: el primero es llamado el padre

(parent) y el segundo el hijo (child). Como se observa en la figura, el origen de la articulación se especifica con respecto al sistema de referencia del eslabón padre.

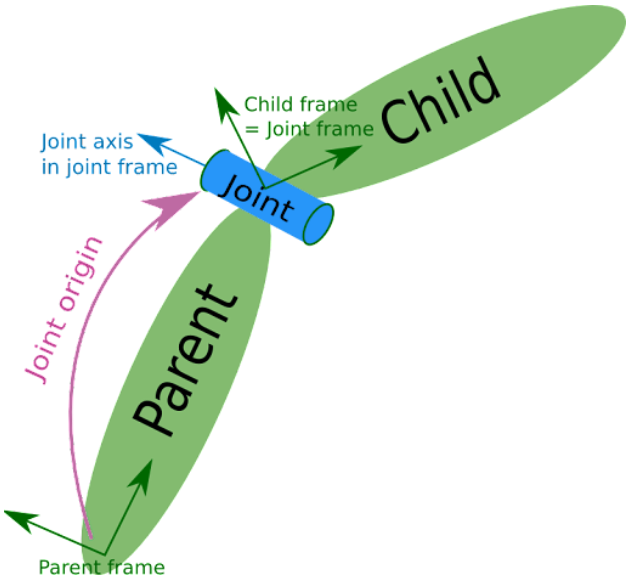

Figura 3 Representación de una articulación en URDF

3. Robot. Esta etiqueta encapsula todo el modelo del robot. Dentro de esta etiqueta se define el nombre del robot, todos sus eslabones y todas sus articulaciones. La sintaxis es la siguiente:

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

Donde pueden existir tantas etiquetas de tipo link y joint como sean necesarias. Un modelo de robot consiste en una serie de articulaciones y eslabones conectados. Un ejemplo se muestra en la Figura4.

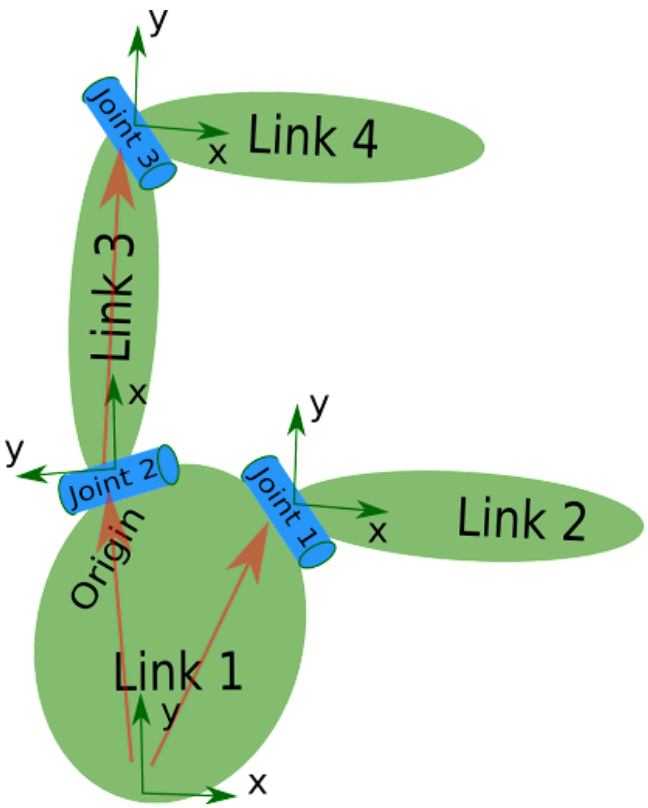

Figura 4 Ejemplo del modelo de un robot en URDF

4. Gazebo. Esta etiqueta se utiliza cuando se incluye parámetros de simulación para el simulador llamado Gazebo, el cual es utilizado por defecto en ROS para simulaciones dinámicas. Se puede usar esta etiqueta para incluir plugins específicos de gazebo, material (color, textura) que se mostraría en el simulador, así como otras propiedades específicas.

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.

B Algunos Elementos de Gazebo

Para la simulación dinámica en Gazebo, es necesario añadir algunos elementos adicionales al URDF. Algunos de estos elementos son los siguientes.

Color. El color en Gazebo no está relacionado con el color en RViz y debe ser especificado por etiquetas especiales de tipo material. Por ejemplo, el siguiente código añade color blanco al eslabón de la base:

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

Transmisión. Para accionar el robot utilizando controladores de ROS, en simulación, se debe definir el elemento llamado `transmission`, el cual se encarga de establecer la relación de cada actuador con su respectiva articulación, y así permite el movimiento de las articulaciones.

Esto se realiza de la siguiente manera:

```
<transmission name="trasmision1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint1">
      <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor1">
      <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

En esta etiqueta, `joint name` representa el nombre de la articulación a la cual se le asocia un actuador. La etiqueta `type` hace referencia al tipo de transmisión; en este caso se trata de una transmisión simple (SimpleTransmission). El elemento hardware Interfaz es el tipo de interfaz de hardware que se utiliza en el robot y puede ser de posición, velocidad o fuerza (position, velocity, effort). Esta interfaz de hardware es cargada desde el plugin de *gazebo ros control*. Es posible además especificar la reducción mecánica, que en el ejemplo mostrado es igual a la unidad.

Plugin de Control. El control se especifica mediante el plugin 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>/robot</robotNamespace>
     </plugin>
</gazebo>
```

En este caso, el *namespace* que se utiliza se denomina robot.