UNIVERSIDAD CARLOS III DE MADRID

CREACIÓN DE MODELO URDF DEL ROBOT MANFRED

PROYECTO FIN DE CARRERA INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

ALUMNO: RAÚL MERINO LÓPEZ
DIRECTOR: JAVIER V. GÓMEZ GONZÁLEZ
TUTOR: DAVID ÁLVAREZ SÁNCHEZ

Título: Creación de modelo URDF del robot MANFRED
Autor: Raúl Merino López
Director: Javier V. Gómez González
Tutor: David Álvarez Sánchez

EL TRIBUNAL

Presidente: _

Vocal: _

Secretario: _

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día _ _ _ de _ _ _ _ _ _ _ _ _ _ _ _ de 201_ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

RESUMEN

En estas últimas décadas, tanto la informática y la robótica están creciendo rápidamente. Hoy en día se emplean robots y manipuladores asistidos tanto para fines industriales, como para ayudar a personas enfermas. Por este motivo, se están desarrollando nuevas tecnologías para lograr controlar el robot según desee el usuario.

Una tecnología que se está usando cada vez más es ROS (Robot Operating System). Es un sistema para la escritura de software robótico de desarrollo libre. Por lo tanto, el usuario puede personalizar los parámetros y características del robot como desee. Una vez definido el software, puede comprobar y realizar simulaciones en el entorno que desee, para después ejecutarlo en tiempo real.

Este proyecto se basa en elaborar software en ROS para el manipulador móvil MANFRED de la Universidad Carlos III. Para ello, hay que conseguir los archivos en 3D de cada pieza del robot, y escribir el código URDF. Después se harán las pruebas para comprobar el correcto funcionamiento del mismo.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

En la actualidad, el constante desarrollo de la electrónica y la informática, ha motivado que todos los campos de la vida humana se están automatizando; por ejemplo: las comunicaciones, el hogar, los transportes, la industria, etc. En todo ese proceso de automatización, los controladores y microcontroladores juegan un papel de vital importancia. Este desarrollo ha provocado que la forma de controlar cualquier máquina sea digital. Por este motivo ha sido necesario desarrollar unos sistemas de adquisición de datos, capaces de obtener las señales emitidas por los sensores y transformarlas en señales digitales. De esta manera el microprocesador que controla el funcionamiento, los puede interpretar y actuar en función de los datos obtenidos y el entorno. En las últimas décadas, todo este desarrollo ha sido muy útil para la realización de autómatas; máquinas capaces de interactuar con el entorno y modificar sus pautas de comportamiento dependiendo de él.

El primer paso en el camino de la robótica para salir del entorno industrial incorporándose a todo tipo de tareas y espacios de trabajo, es la necesidad de incorporar en un mismo sistema la movilidad y la capacidad para manipular objetos que se aúnan en los manipuladores móviles. Son máquinas capaces de moverse en entornos diseñados para humanos, interactuar y colaborar con personas y que, simultáneamente, cumplan unos criterios de robustez operacional y seguridad física elevados. Por este motivo, el diseño del manipulador móvil tiene que cumplir esas características integrando tanto el diseño hardware del sistema (mecánico, eléctrico, sensorial….) como la arquitectura software (que presenta un alto grado de robustez y capacidad para detectar y poder recuperar fallos en todos los niveles operacionales).

El grupo de investigación del Roboticslab de la Universidad Carlos III de Madrid, ha desarrollado un amplio trabajo en el campo de la robótica móvil y, más recientemente, en la línea de los manipuladores móviles. Es fundamentalmente en este área dónde la experiencia acumulada en proyectos

previos nos ha permitido extraer conclusiones que se están aplicando en el desarrollo del robot MANFRED que aquí presentamos.

Figura 1: Manipulador Manfred

El objetivo a lograr en este proyecto es conseguir un simulador gráfico 3D del manipulador Manfred. Un simulador en robótica aporta grandes ventajas:

- Ahorro de tiempo en el diseño de aplicaciones.
- Aumentar en gran medida el nivel de seguridad, asociado a los equipos robóticos desde varios escenarios, ya que pueden ser probados antes de que el sistema esté activado.

Las características de estas aplicaciones son tan avanzadas que permiten definir todos los parámetros de los movimientos del robot**.** El concepto de simulación es posible gracias a la programación del sistema mediante el software específico. El software empleado será Robot Operating System (ROS), el cual está creciendo enormemente puesto que es software libre y permite realizar simulaciones en bastantes entornos conocidos, como por ejemplo moveit! o Gazebo.

1.1 Objetivos

El objetivo principal es conseguir describir el URDF (Unified Robot Description Format) que contenga toda la información del robot manipulador Manfred. La descripción de un robot consiste en un conjunto eslabones y de articulaciones que conectan los eslabones juntos. Para comprobar que el archivo URDF está editado correctamente, será necesario utilizar la herramienta de ROS llamada rviz, que es un visualizador con la capacidad de mover todas las articulaciones. Debido a que el URDF del robot Manfred será muy extenso, hay que dar un enfoque más general con XACRO. Un XACRO es un tipo de URDF macro que se emplea para generar un URDF. Con XACRO es muy sencillo después personalizar las partes deseadas y generar el URDF personalizado de todo el robot. Como hay que construir tres tipos de Manfred distintos, hay que definir sus partes por separado y después unirlas. Los tres manipuladores Manfred son:

- Manfred1: Base, brazo LWR-UC3M-1 y Gifuhand III.
- Manfred2: Base y brazo LWR-UC3M-1 con el primer eslabón cortado.
- Manfred3: Base, brazo Manfred3 y Gifuhand III.

Después de comprobar con la herramienta rviz que tanto los eslabones como las articulaciones están correctamente, significa que el archivo URDF está bien definido. Por último, se configurará cada archivo y se comprobará en Moveit!.

2. El manipulador móvil MANFRED

Teniendo en cuenta que los requisitos que tiene que cumplir el manipulador son bajo peso y control coordinado para todos los grados de libertad, el manipulador móvil avanzado que se propone Figura 1 se estructura en los siguientes elementos:

2.1 Base móvil

Esta base está constituida por dos plataformas de acero de base octogonal con 2 grados de libertad. Tres son ruedas de apoyo y las otras dos son ruedas motrices asociadas a dos motores tipo Brushless, con sus correspondientes servoamplificadores. Las dos ruedas motrices dan lugar al movimiento tipo diferencial, lo que permite al robot girar sobre sí mismo.

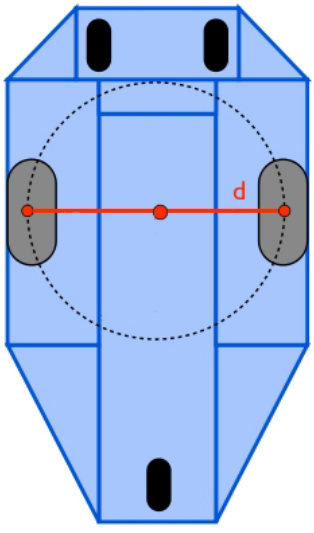

Figura 2: Base de Manfred.

2.2 Brazo manipulador UC3M-LWR-1

El brazo manipulador UC3M-LWR-1 constituye el elemento fundamental del robot manipulador MANFRED. Está compuesto por elementos rígidos conectados por medio de articulaciones de revolución. Cada articulación conforma un grado de libertad, hasta alcanzar un total de 6 g.d.l. Está diseñado para dotar a MANFRED de una notable flexibilidad para realizar tareas de agarre y desplazamiento de objetos, combinando los diversos grados de libertad de los que dispone.

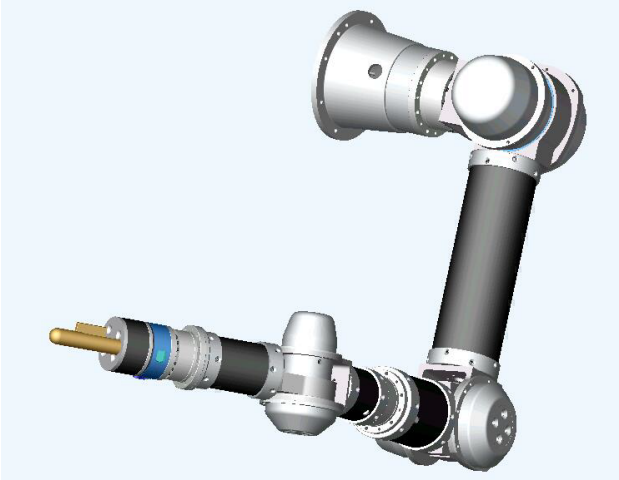

Figura 3: Brazo manipulador UC3M-LWR-1.

2.3 El torso

Es el cuerpo del manipulador (figura 1), en el que van situados:

- El brazo manipulador UC3M-LWR-1.
- La plataforma de visión.
- La telemetría láser 3D.

2.4 El sistema sensorial

Permite obtener información del entorno de trabajo que es utilizada para generar los datos que permitan dotar de capacidad de movimiento, tanto al brazo manipulador como a la base móvil. Para ello, se utilizará simultáneamente un sensor láser Hokuyo y el detector Kinect.

2.4.1 Láser Hokuyo

Este láser tiene una gran resolución angular y velocidad de barrido, y tiene un rango angular de 270º . Para conseguir puntos de todo el espacio tridimensional, se hará girar el láser con el motor Dynamixel. Este motor posee una gran robustez y velocidad variable, ideal para mover el láser.

Figura 4: Motor Dynamixel.

Figura 5: Láser Hokuyo.

2.4.2 Kinect

El aparato electrónico kinect es un controlador de juego desarrollado por Mircrosoft para la videoconsola Xbox 360. Está compuesto por un sensor CMOS para RGB, un sensor CMOS para infrarrojo de Clase 1, un dispositivo láser que genera un patrón de puntos y segmentos, un micrófono de múltiples matrices y el chip PrimeSensor que se encarga de todos los controles del sistema (captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y capacidades de reconocimiento de voz). Este dispositivo va conectado al manipulador Manfred, obteniendo la capacidad de detectar el entorno y reconocer los objetos.

Figura 6: Kinect.

2.5 Brazo manipulador MANFRED 3.

Basado en el diseño del brazo manipulador anterior, el equipo de RoboticsLab ha desarrollado un nuevo diseño, que está enfocado en:

- Reducir su anterior peso propio (de 18 kg a 14 kg).
- Añadir una DoF adicional (de 6 a 7 con el fin de resolver el problemas asociados con la no redundancia: singularidades y espacios de trabajo obstruido).
- Distribuir de forma independiente un control de la fuerza y la posición en cada articulación (para evaluar adecuadamente el rendimiento del brazo).
- Incluir todos los sistemas de cable en el interior del brazo (junto con su electrónica), en busca de una mejora de la seguridad para los usuarios potenciales y reducir cualquier riesgo asociado al estrangulamiento de los componentes.

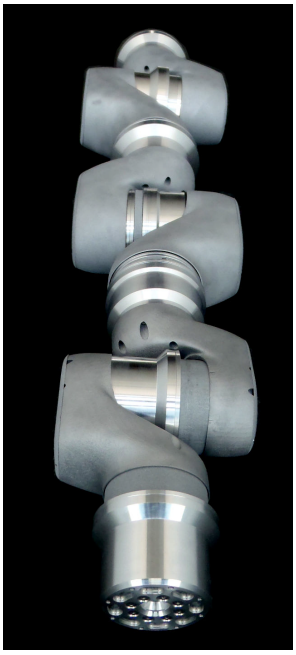

Figura 7: Brazo manipulador Manfred 3.

2.6 Gifu hand III

Una fusión de avance de las tecnologías y una intensa investigación dio lugar a Gifu hand III. La mano se asemeja a la mano humana en tamaño y forma, así como en su capacidad para la manipulación. Gifu Hand ha sido diseñada para ser utilizado como una plataforma de manos de un robot para la investigación robótica. Presenta muy buena reacción en los puntos de transmisión, oponibilidad del pulgar, y el espacio movilidad de los dedos. El pulgar tiene 4 articulaciones con 4 grados de libertad (g.d.l), y cada uno de los otros cuatro dedos tiene 4 articulaciones con 3 g.d.l. Por lo tanto, Gifu hand III presenta 16 g.d.l., lo cual le otorga una gran destreza para poder agarrar cualquier objeto. Está equipada con servomotores de corriente continua, construidos en los componentes de los dedos y la palma de la mano. El uso de engranajes de satélite en la transmisión le otorga un alto nivel de rigidez y unas tasas de holgura muy bajas.

La mano Gifu hand III irá colocada en el extremo del brazo, tanto en el UC3M-LWR-1 como en el brazo manipulador MANFRED 3.

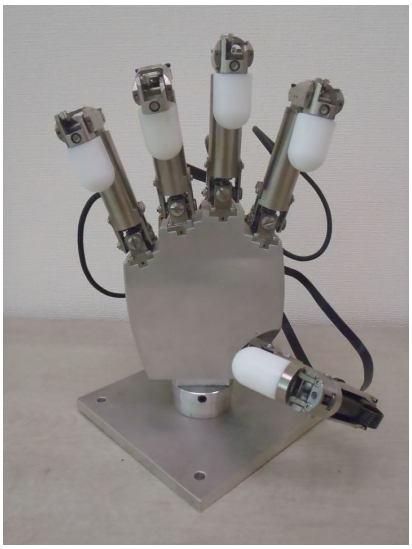

Figura 8: Gifu Hand III.

3. Robot Operating System

3.1 ¿Qué es ROS?

Robot operating system (ROS) es un marco flexible para la escritura de software robótico. Se trata de una colección de herramientas, librerías y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento complejo y robusto en una amplia variedad de plataformas robóticas.

Al crear un diseño altamente robusto, el software robótico tiene una arquitectura muy complicada. Desde la perspectiva del robot, los problemas que parecen triviales a los seres humanos a menudo varían enormemente entre las instancias de tareas y entornos. Hacer frente a estas variaciones es tan fuerte que ningún individuo puede aspirar a hacerlo por su cuenta.

Como resultado, ROS fue construido desde cero para fomentar la robótica de colaboración de desarrollo de software. ROS fue diseñado específicamente para que grupos puedan colaborar y construir sobre el trabajo del otro , y definir así una arquitectura robótica totalmente sólida y funcional. Por ello, ROS es un proyecto de código abierto, y el código dentro de ella es el resultado de los esfuerzos combinados de una comunidad internacional.

El ecosistema de ROS se compone actualmente de decenas de miles de usuarios en todo el mundo , que trabajan en ámbitos que van desde pequeños proyectos de aficionados a los sistemas de automatización industrial de gran tamaño.

ROS fue diseñado para ser lo más distribuida y modular como sea posible, de modo que los usuarios pueden usar poco o mucho según sus necesidades. La modularidad de ROS permite que el usuario pueda seleccionar y elegir qué partes son útiles y qué partes prefiere implementar. La naturaleza distribuida de ROS también fomenta una comunidad grande proporcionando paquetes que le dan un gran valor en la parte superior del sistema de ROS principal. Estos paquetes varían en la fidelidad que abarca todo, desde implementaciones con nuevos algoritmos para los controladores (drivers) a las capacidades en la calidad industrial . La comunidad de usuarios de ROS definen una infraestructura común para proporcionar un punto de integración que ofrece el acceso a los controladores de hardware, las capacidades de robots genéricos, herramientas de desarrollo, librerías externas útiles, y más.

ROS posee una red peer-to-peer de procesos ejecutándose a la vez. Los conceptos básicos son:

- Nodes: Son los procesos que realizan el cálculo.
- Master: Proporciona la información necesaria para que todos los nodos sean identificados y se comuniquen.
- Parameter server: Forma parte del Master, y permite guardar los datos de forma centralizada.
- Messages: Estos mensajes son usados por los nodos para que puedan comunicarse entre ellos.
- Topic: Es un nombre que es usado para identificar el contenido del mensaje
- Services: Es el modelo de comunicación cliente-servidor.
- Bags: Son un formato para guardar y reproducir datos de mensajes ROS.

3.2 Componentes Principales

3.2.1 Infraestructura de comunicaciones

Un sistema de comunicación es a menudo una de las primeras necesidades que surgen en la implementación de una nueva aplicación robótica. El sistema de mensajería integrado de ROS le permite ahorrar tiempo mediante la gestión de los detalles de la comunicación entre los nodos distribuidos a través de publicaciones anónimas / método de suscripción. Otra ventaja de utilizar un sistema de de mensajes es que obliga a implementar interfaces claras entre los nodos del sistema, mejorando así la encapsulación y la promoción de la reutilización de código . La estructura de estas interfaces de mensaje se define en el mensaje de IDL (Interface Description Language).

3.2.2 Características robóticas específicas

ROS proporciona librerías robóticas específicas comunes y herramientas para que el robot funcione rápidamente. Estas son sólo algunas de las características robóticas específicas que ROS ofrece.

3.2.2.1 Mensajes estándar de robot

Años de discusión y desarrollo de la comunidad han dado lugar a un conjunto de formatos de mensaje estándar que cubren la mayor parte de los casos de uso común en la robótica. Hay definiciones de mensajes de conceptos geométricos como poses, transformadas y vectores; para sensores como cámaras, IMU y láser; y para los datos de navegación como odometría, rutas y mapas; entre muchos otros. Estos mensajes también incrementan la interoperabilidad con las herramientas y características existentes en el ecosistema de ROS.

3.2.2.2 Librería geométrica de robot

Un reto común en muchos proyectos de robótica es hacer el seguimiento de distintas partes del robot y dónde se localizan con respecto a la otra . Por ejemplo, si desea combinar datos de una cámara con los datos de un láser, lo que necesita saber dónde está cada sensor, en algún marco de referencia común . Este problema es especialmente importante para los robots humanoides con muchas partes móviles . Abordamos este problema de ROS con la librería tf (transformada), que mantendrá el seguimiento de todo que está en el sistema robótico.

Diseñado pensando en la eficacia, la librería tf se ha utilizado para administrar las transformadas de los datos para robots con más de un centenar de grados de libertad y las tasas de actualización de cientos de Hertzios. La librería tf permite definir las transformaciones estáticas, tales como una cámara que se fija a una base móvil, y las transformaciones dinámicas, como una articulación de un brazo robótico.

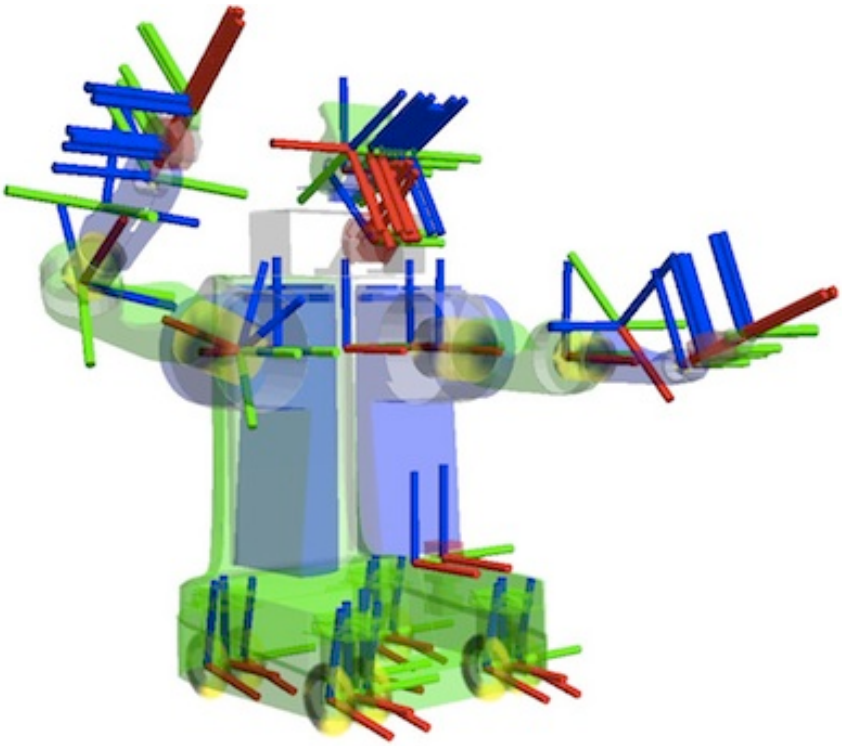

Figura 9: Geometría de Robot en ROS.

3.2.2.3 Descripción del lenguaje robot

ROS proporciona un conjunto de herramientas para describir y modelar el robot para que pueda ser comprendido por el resto de su sistema de ROS, incluyendo tf, robot_state_publisher y rviz. El formato para la descripción de su robot de ROS es URDF (Unified Robot Descripción Format), que consiste en un documento XML en el que se describen las propiedades físicas de su robot, desde la longitud de las extremidades y tamaños de ruedas a la ubicación de los sensores y la apariencia visual de cada parte del robot.

Una vez que se define de esta manera, el robot se puede utilizar fácilmente con la librería tf, representada en las tres dimensiones con visualizaciones detalladas, y se utiliza con los simuladores y los planificadores de movimiento.

3.2.2.4 Diagnósticos

ROS proporciona una forma estándar para producir y recopilar diagnósticos globales sobre el robot para que, a simple vista, se pueda ver rápidamente el estado del robot y determinar la forma de abordar los problemas que se presenten.

3.3 Herramientas

Uno de los rasgos más característicos de ROS es el poderoso conjunto de herramientas de desarrollo. Estas herramientas ayudan con la introspección, la depuración, el trazado, y visualización del estado en el desarrollo del sistema. Las herramientas de ROS se aprovechan de la capacidad de introspección a través de una extensa colección de utilidades de línea de comandos y gráficos que simplifican el desarrollo y depuración. Las herramientas más usadas son las siguientes:

3.3.1 rviz

Tal vez la herramienta más conocida en ROS, rviz proporciona la visualización tridimensional de muchos tipos de datos de los sensores y cualquier robot descrito por URDF. La herramienta rviz puede visualizar muchos de los tipos de mensajes comunes previstos en ROS, como escaneos láser, las nubes de puntos tridimensionales, y las imágenes de cámaras. También utiliza la información de la librería tf para mostrar todos los datos de los sensores en un sistema de coordenadas común de su elección, así como una representación tridimensional del robot. Visualizar todos los datos en la misma aplicación no sólo se ve impresionante, sino que también le permite ver rápidamente lo que ve el robot, e identificar problemas tales como errores de alineación del sensor o imprecisiones del modelo robot.

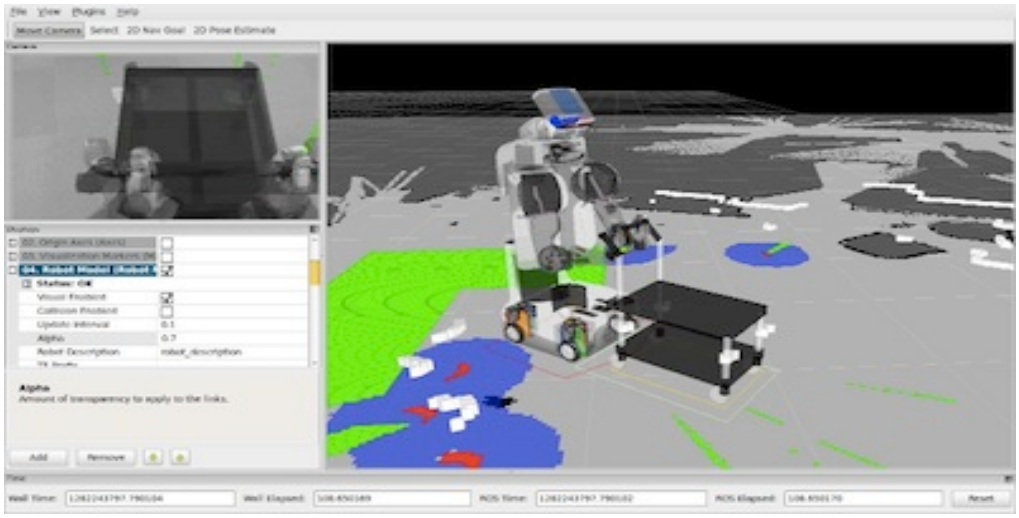

Figura 10: Ejemplo de la herramienta rviz.

3.3.2 rqt

ROS proporciona rqt, un marco basado en Qt para el desarrollo de interfaces gráficas para el robot. Se pueden crear interfaces personalizadas y configurar la extensa librería de plugins rqt integradas en pestañas, en pantalla dividida, y otros diseños. También se pueden introducir y modificar los propios plugins rqt.

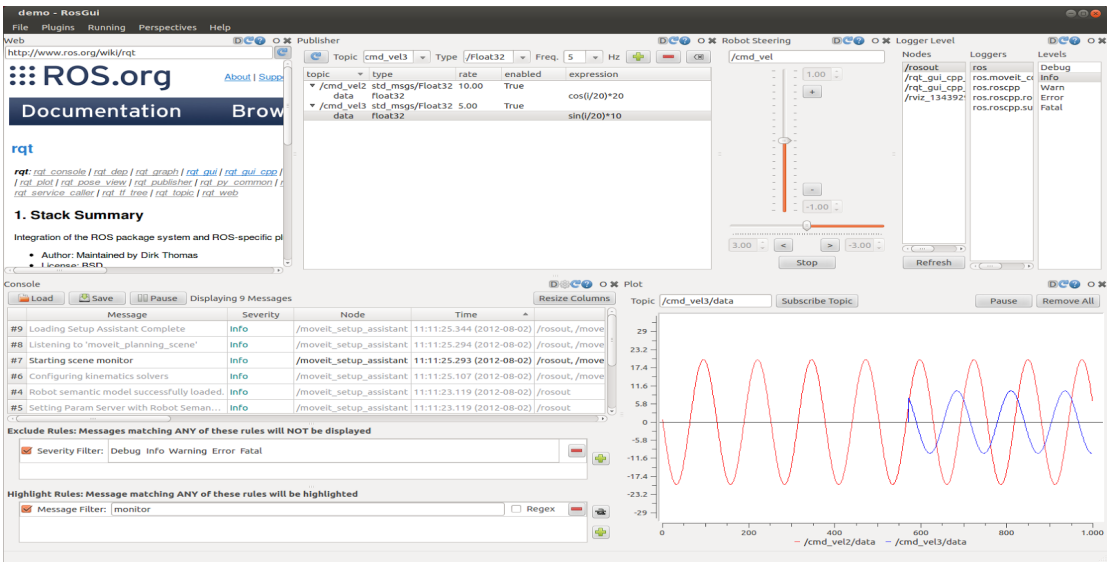

Figura 11: Interfaz de rqt.

El plugin rqt_graph ofrece la introspección y la visualización de un sistema de ROS en vivo, que muestra los nodos y las conexiones entre ellos, lo que le permite fácilmente depurar y entender su sistema de funcionamiento y cómo se estructura.

El plugin rqt_plot monitoriza frecuencias , tensiones, o cualquier cosa que se pueda representar como un número que varía con el tiempo. El plugin rqt_plot le permite elegir el trazado que mejor se adapte a las necesidades.

Con el plugin rqt_topic se puede supervisar cualquier número de temas que se publican en el sistema. El plugin rqt_publisher permite publicar mensajes a cualquier tema, lo que facilita la experimentación con el sistema.

Para el registro de datos y la reproducción, ROS utiliza el formato de bolsa (Bag). Los archivos Bag se pueden crear y acceder de forma gráfica a través del plugin de rqt_bag. Este plugin puede grabar datos en bolsas, reproducir temas seleccionados de una bolsa, y visualizar el contenido de una bolsa, incluyendo la visualización de las imágenes y el trazado de los valores numéricos en el tiempo.

3.4 Integración con otras librerías

ROS proporciona una integración perfecta con programas y aplicaciones muy populares, como Gazebo, moveit!, OpenCV y otros proyectos de código abierto. ROS tiene un sistema de paso de mensajes que le permite cambiar fácilmente a diferentes fuentes de datos, desde sensores en vivo a archivos de registro.

3.4.1 Gazebo

Gazebo es un simulador 3D multi-robot, con propiedades dinámicas y cinemáticas completas. La integración entre ROS y Gazebo es proporcionada por un conjunto de plugins de Gazebo que apoyan muchos robots y sensores existentes. Debido a que los plugins presentan la misma interfaz de mensaje que el resto del ecosistema ROS, se pueden escribir nodos ROS que son compatibles con la simulación, los datos registrados y el hardware de los robots.

3.4.2 Moveit!

MoveIt! es una librería de planificación de movimientos que ofrece implementaciones eficientes y bien probados de los algoritmos de planificación que se han utilizado en una amplia variedad de robots, desde plataformas con ruedas sencillas de humanoides para caminar. Si se desea aplicar un algoritmo existente o desarrollar otro propio enfoque, MoveIt! es perfecto para la planificación de movimiento. A través de su integración con los ROS, MoveIt! se puede utilizar con cualquier robot apoyado por ROS. Los datos de planificación se pueden visualizar con rviz y plugins rqt, y se pueden ejecutar a través del sistema de control de ROS.

4. Lenguajes y aplicaciones utilizadas

4.1 Lenguaje de programación XML

XML, siglas en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos de forma legible. XML permite definir etiquetas personalizadas para la descripción y la organización de datos [4].

Además permite una utilización efectiva en Internet para sus diferentes terminales de una manera segura, fiable y fácil. Se puede usar en bases de datos, editores de texto, hojas de cálculo, etc.

Ventajas del XML:

- Es ampliable: Se puede extender fácilmente el código XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin ninguna complicación.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se acelera el desarrollo de aplicaciones.
- Al ser de estructura jerárquica, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.
- Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.

Estructura de un documento XML

La tecnología XML mantiene la información estructurada de la forma más abstracta y reutilizable posible. Esto quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Por esta razón, presenta una estructura jerárquica. A esas partes se las llaman *elementos*, y se las señala mediante etiquetas. Una etiqueta es una marca hecha en el documento, que señala un fragmento de éste como un elemento. Las etiquetas tienen la forma <nombre>, donde *nombre* es el nombre del elemento que se está señalando.

Partes de un documento XML:

• Prólogo

Los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas. No es obligatorio ponerlo.

Ejemplo: <?xml version="1.0" encoding="UTF-8"?>

• Cuerpo

El cuerpo tiene que contener sólo un elemento raíz, característica obligatoria para que el documento esté bien formado. Sin embargo es necesaria la adquisición de datos para su buen funcionamiento. Ejemplo:

<Edit_Mensaje> (...) </ Edit_Mensaje>

• Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

•Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

• Entidades predefinidas

Entidades para representar caracteres especiales para que no sean interpretados como marcado en el procesador XML.

Ejemplo: entidad predefinida: *& carácter: &*.

• Comentarios

Son aclaraciones que pueden ser escritas para informar al lector, y son ignorados por el procesado. Los comentarios en XML mantienen el siguiente formato:

<!--- Esto es un comentario --->

4.2 XML Robot Description Format (URDF)

Unified Robot Descripción Format (URDF) es una especificación XML para describir un robot. Tratamos de mantener esta especificación lo más general posible, pero, obviamente, la especificación no puede describir todos los robots. La principal limitación en este punto es que sólo las estructuras de árbol pueden ser representados, descartando todos los robots paralelos. Además, la especificación asume que el robot consiste en eslabones rígidos conectados por articulaciones; elementos flexibles no son compatibles. La especificación incluye:

- Descripción cinemática y dinámica del robot.
- Representación visual del robot.
- Modelo de colisiones del robot.

La descripción de un robot consiste en un conjunto de elementos de eslabones (links) y de un conjunto de elementos de unión (joints) que conectan los eslabones juntos.

Véase la siguiente estructura de árbol para poder editar su URDF:

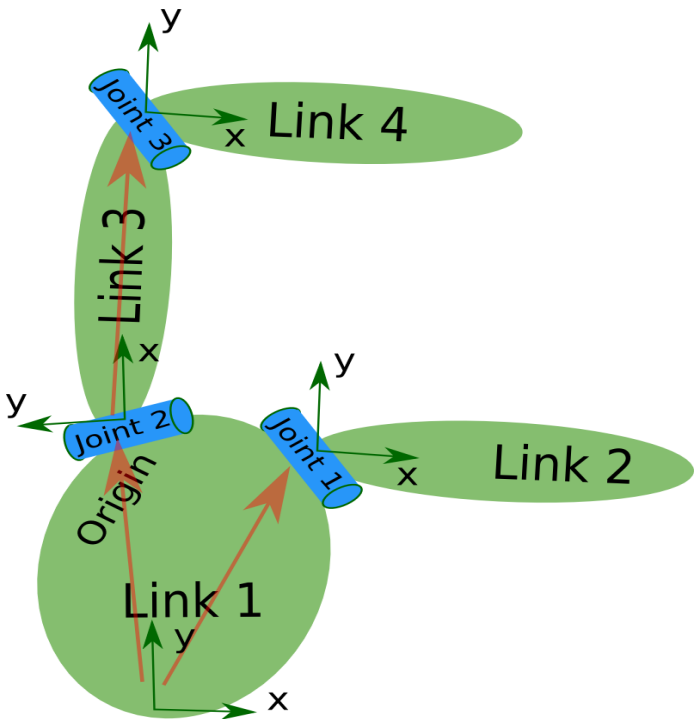

Figura 12: Ejemplo de estructura robótica.

Así que la descripción típica de este robot será:

```
<robot name="pr2">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint>  ....  </joint>
  <joint>  ....  </joint>
  <joint>  ....  </joint>
</robot>
```

Se puede ver que el elemento raíz del formato URDF es el elemento <robot>. Hay que definir ahora el elemento link y el elemento joint, y las etiquetas y atributos de cada uno de ellos.

4.2.1 Elemento link

El elemento link describe un cuerpo rígido con una inercia y características visuales:

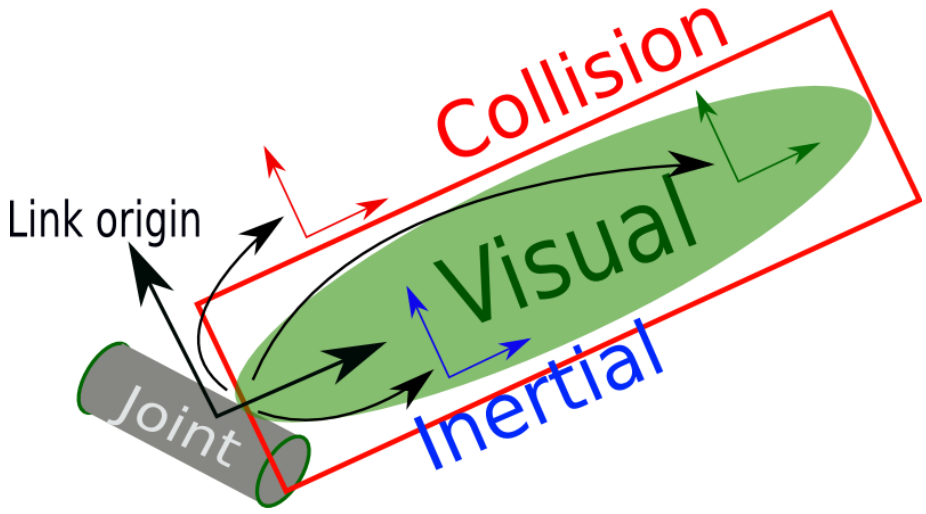

Figura 13: Elemento link.

Atributos:

Name (necesario)
El nombre del eslabón (link).

Elementos:

<inertial> (opcional)
Las propiedades inerciales del eslabón (link).

<origin> (opcional: por defecto a la identidad si no se especifica).
Esta es la posición del marco de referencia inercial, relativo con el marco de referencia del eslabón. El origen del sistema de referencia inercial tiene que estar en el centro de gravedad. Sus ejes no necesitan estar alineados con los ejes principales de la inercia.
xyz (opcional: por defecto el vector cero)
Representa el desplazamiento en x, y y z.

rpy (opcional: por defecto a la identidad si no se especifica)

Representa los ángulos de giro sobre los ejes x, y y z.

<mass>

Es el valor de la masa del eslabón.

<inertia>

La matriz rotacional inercial (3x3), representada en el marco inercial.Debido a que la matriz rotacional inercial es simétrica, solo 6 elementos de esta matriz son especificados, utilizando los atributos ixx, ixy, ixz, iyy, iyz, izz.

<visual> (opcional)

Las propiedades visuales del eslabón (link). Este elemento especifica la forma del objeto (caja, cilindro, esfera, etc.) para propósitos de visualización. NOTA: Varias peticiones de etiquetas <visual> pueden existir para el mismo eslabón. La unión de estas peticiones constituye la representación visual del eslabón.

name (opcional).

Especifica un nombre para una parte de la geometría del eslabón. Esto es útil para poder especificar trozos de la geometría.

<origin> (opcional: por defecto a la identidad si no se especifica).

El marco de referencia del elemento visual con respecto al marco de referencia del eslabón.

xyz (opcional: por defecto el vector cero)

Representa el desplazamiento en x, y y z.

rpy (opcional: por defecto a la identidad si no se especifica)

Representa los ángulos de giro sobre los ejes x, y y z.

<geometry> (necesario)

La forma del objeto visual. Puede ser una de las siguientes:

<box>

El atributo Size representa la longitud de los tres lados del cubo. El origen del cubo está en su centro.

<cylinder>

Con radius y lenght se especifica su radio y altura. El origen del cilindro está en su centro.

<sphere>

Especifica el radio con radius. El origen de la esfera está en su centro.

<mesh>

Una figura tridimensional especificada por un archivo en filename, y un opcional scale que escala el tamaño de la figura. El formato recomendado para la mejor textura y color es el archivo Collada .dae, aunque también soporta archivos .stl.

<material> (opcional)

Es el material del elemento visual. Está permitido especificar un elemento material en el elemento 'robot' (nivel de arriba). Desde el eslabón se puede referirse el material por un nombre en name.

<color> (opcional)

rgba El color de un material es definido por un conjunto de cuatro números representando rojo/verde/azul/opacidad, cada uno entre un rango de 0 a 1.

<texture> (opcional)

La textura del material es definida por un archivo, filename.

<collision> (opcional)

Las propiedades colisionales del eslabón (link). Puede ser diferente de las propiedades visuales de un eslabón, por ejemplo, los modelos de colisión más simples son usados para reducir el tiempo de computación. NOTA: Varias peticiones de etiquetas <collision> pueden existir para el mismo eslabón. La unión de la geometría que definen constituye la representación visual del eslabón.

name (opcional).

Especifica un nombre para una parte de la geometría del eslabón. Esto es útil para poder especificar trozos de la geometría.

<origin> (opcional: por defecto a la identidad si no se especifica).

Véase la descripción <origin> en el elemento visual más arriba.

<geometry>

Véase la descripción <geometry> en el elemento visual más arriba.

Aquí se muestra un ejemplo con todos los atributos anteriormente mencionados:

```
<link name="my_link">
  <inertial>
      <origin xyz="0 0 0.5" rpy="0 0 0"/>
      <mass value="1"/>
      <inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>
  <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
      <box size="1 1 1" />
      </geometry>
      <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
      </material>
  </visual>
  <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
      <cylinder radius="1" length="0.5"/>
      </geometry>
  </collision>
</link>
```

4.2.2 Elemento joint

El elemento joint (articulación) define las propiedades cinemáticas y dinámicas, además de los límites de seguridad (safety limits) de la articulación:

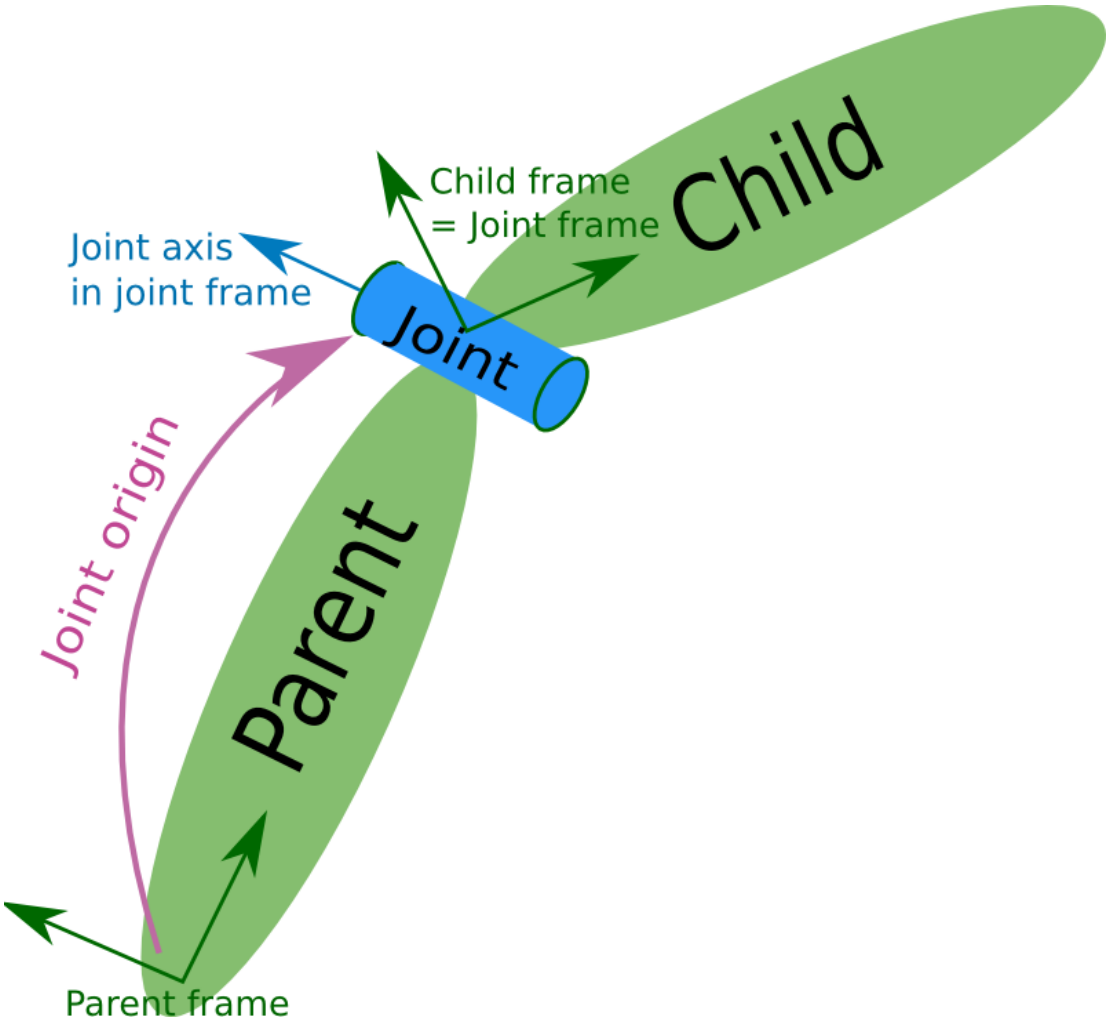

Figura 14: Elemento joint.

Atributos:

Name (necesario): Es el nombre de su articulación (joint).
Type (necesario): Establece el tipo de articulación, los cuales pueden ser uno de los siguientes:

• **revolute**: una articulación de bisagra que gira a lo largo del eje y tiene un rango limitado especificado por los límites superior e inferior.
• **continuous**: una articulación en bisagra continua que gira en torno al eje y no tiene límites superior e inferior.
• **prismatic**: una junta deslizante que se desliza a lo largo del eje, y tiene un rango limitado especificado por los límites superior e inferior.
• **fixed**: Esto no es realmente una articulación, ya que no puede moverse. Todos los grados de libertad están bloqueados. Este tipo de unión no requiere ejes, la calibración, la dinámica, límites o controlador de seguridad.
• **floating**: Esta articulación permite el movimiento de los 6 grados de libertad.
• **planar**: Esta articulación permite el movimiento en un plano perpendicular al eje.

Elementos:

<origin> (opcional, por defecto la identidad si no se especifica)

Esta es la transformada desde el eslabón del padre con el eslabón del hijo. La articulación se encuentra en el origen del eslabón de hijo, como se muestra en la figura anterior.

xyz (opcional: por defecto el vector cero)

Representa el desplazamiento en x, y y z.

rpy (opcional: por defecto el vector cero si no se especifica)

Representa la rotación en torno al eje fijo x, y y z. Todos los ángulos se especifican en radianes.

<parent> (necesario)

El nombre del eslabón padre (obligatorio):

link

El nombre del eslabón que es el padre de este eslabón en la estructura de árbol del robot.

<child> (necesario)

El nombre del eslabón hijo (obligatorio).

link

El nombre del eslabón que es el del hijo.

<axis> (opcional: por defecto (1,0,0))

Este es el eje de rotación de articulaciones de giro, el eje de traslación para juntas prismáticas, y la superficie normal para juntas planas. El eje se especifica en el marco común de referencia. Articulaciones fijas y flotantes no utilizan éste atributo.

xyz (necesario)

Representa el componente de un vector x, y, z. El vector debe estar normalizado.

<calibration> (opcional)

La posición de referencia de la articulación, usada para calibrar la posición absoluta de la articulación.

• **rising** (opcional)

Cuando la articulación se mueve en una dirección positiva, esta posición de referencia dará lugar a un flanco ascendente.

• **falling** (opcional)

Cuando la articulación se mueve en una dirección positiva, esta posición de referencia dará lugar a un flanco descendente.

<dynamics> (opcional)

Un elemento que especifica las propiedades físicas de la articulación. Estos valores se utilizan para definir las propiedades de modelado de la articulación, especialmente útil para la simulación.

• **damping** (opcional, por defecto 0)

El valor de amortiguación física de la articulación ($\frac{N*s}{m}$ para articulaciones prismáticas, $\frac{N*m*s}{rad}$ para articulaciones de revolución).

• **friction** (opcional, por defecto 0)

El valor de fricción estática física de la articulación (N para articulaciones prismáticas, $N*s$ para articulaciones de revolución).

<limit> (únicamente para articulaciones prismáticas y de revoluto)

Un elemento que puede contener los siguientes atributos:

• **lower** (opcional, por defecto es 0)

Un atributo que especifica el límite inferior de unión (radianes para juntas de revolución, metros en las articulaciones prismáticas). En articulaciones contínuas no se especifica este campo.

• **upper** (opcional, por defecto es 0)

Un atributo que especifica el límite superior conjunta (radianes para juntas de revolución, metros en las articulaciones prismáticas). En articulaciones contínuas no se especifica este campo.

• **effort** (requerido)

Un atributo para el máximo esfuerzo de la articulación (|esfuerzo aplicado | <|esfuerzo|).

• **velocity** (requerido)

Define la velocidad máxima de la articulación.

<mimic> (opcional)

Esta etiqueta se utiliza para especificar esta articulación imita a otra articulación existent. El valor de este conjunto puede calcularse como valor = multiplicador * other_joint_value + offset. Va seguido de los siguientes atributos:

joint (requerido)

Especifica el nombre de la articulación a imitar.

multiplier (opcional)

Especifica el factor multiplicativo en la fórmula anterior. El valor por defecto es 1.

offset (opcional)

Especifica el desplazamiento para agregar en la fórmula anterior. Por defecto es 0.

<safety_controller> (opcional)

Un elemento que puede contener los siguientes atributos:

• **soft_lower_limit** (opcional, por defecto es 0)

Un atributo que especifica el límite inferior de la articulación donde el controlador de seguridad comienza a limitar la posición de la articulación. Este límite debe ser mayor que el límite inferior de la articulación.

• **soft_upper_limit** (opcional, por defecto es 0)

Un atributo que especifica el límite superior de la articulación donde el controlador de seguridad comienza a limitar la posición de la articulación. Este límite debe ser menor que el límite superior de la articulación.

•**k_position** (opcional, por defecto es 0)

Un atributo que especifica la relación entre los límites de posición y velocidad.

•**k_velocity** (requerido)

Un atributo que especifica la relación entre el esfuerzo y los límites de velocidad.

A continuación se muestra un ejemplo con todos los atributos y elementos anteriormente mencionados:

```
<joint name="my_joint" type="floating">
        <origin xyz="0 0 1" rpy="0 0 3.1416"/>
        <parent link="link1"/>
        <child link="link2"/>
        <calibration rising="0.0"/>
        <dynamics damping="0.0" friction="0.0"/>
        <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
        <safety_controller k_velocity="10" k_position="15"
soft_lower_limit="-2.0" soft_upper_limit="0.5" />
</joint>
```

4.2.3 Ejemplo de una estructura robótica

A continuación se mostrará cómo crear el código en URDF del robot que se muestra en la siguiente figura:

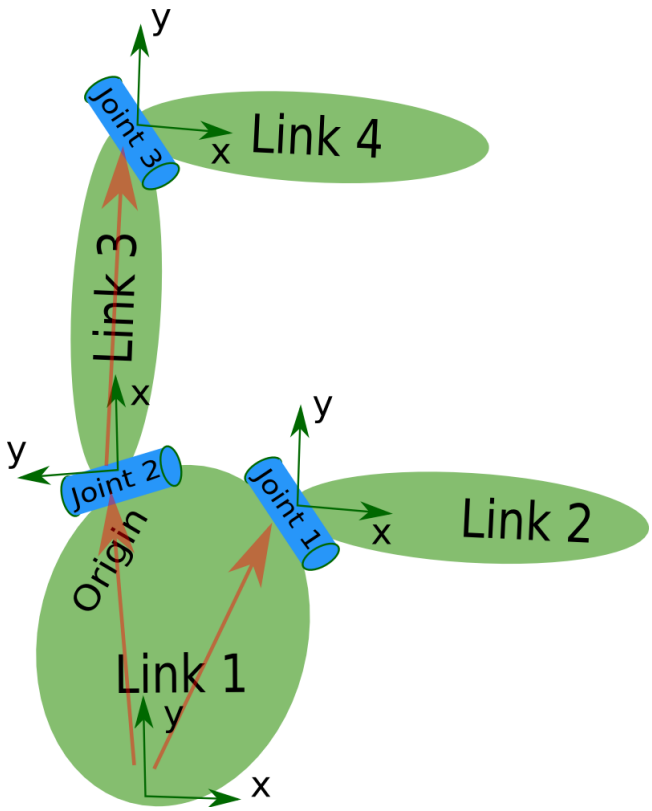

Figura 15: Estructura robótica.

El robot en de ésta imagen mantiene una estructura de árbol. Hay que empezar por lo más simple, y se creará el código de esta estructura, sin importar sobre las dimensiones, etcétera. Con el editor de texto, hay que crear un archivo llamado my_robot.urdf. Dentro hay que escribir lo siguiente, teniendo en cuenta la estructura de la imagen:

<robot name="test_robot">
<link name="link1" />
<link name="link2" />
<link name="link3" />

<link name="link4" />

<joint name="joint1" type="continuous">
<parent link="link1"/>
<child link="link2"/>
</joint>

<joint name="joint2" type="continuous">
<parent link="link1"/>
<child link="link3"/>
</joint>

<joint name="joint3" type="continuous">
<parent link="link3"/>
<child link="link4"/>
</joint>
</robot>

Esta estructura es bastante simple. Tiene cuatro eslabones (links), y entre dos de ellas siempre hay una articulación (joint), por lo que en este caso hay tres. Para comprobar que la estructura esté correctamente montada, hay que ejecutar la siguiente instrucción en el terminal:

check_urdf my_robot.urdf

Con ésta línea de código, el sistema analizará el programa de my_robot.urdf y mostrará por pantalla lo siguiente:

robot name is: test_robot
---------- Successfully Parsed XML ----------
root Link: link1 has 2 child(ren)
child(1): link2
child(2): link3
child(1): link4

Se puede observar que mantiene la estructura diseñada con el robot de la imagen de arriba.

Definir las dimensiones:

Ahora que está diseñada la estructura básica, hay que añadir sus correspondientes dimensiones. Observando la imagen del robot, el marco de referencia de cada pieza (en verde) está localizada en la parte inferior del link, que es el mismo marco de referencia de la articulación. Así que para añadir las dimensiones a el robot, todo lo que hay que hacer es especificar el offset desde la pieza a la articulación de sus hijos. Para realizar esto, hay que añadir el campo <origen> para cada una de las articulaciones.

Ahora hay que observar la segunda articulación. Joint2 esta desplazado (offset) en la dirección Y desde el link1, un pequeño offset en dirección X negativa desde link1, y es girada 90º sobre el eje Z. Por lo tanto, hay que añadir el siguiente elemento <origin> :
<origin xyz="-2 5 0" rpy="0 0 1.57" />

Al repetir esto para todos los elementos de la estructura del robot, el URDF quedaría de la siguiente manera:

```
<robot name="test_robot">
<link name="link1" />
<link name="link2" />
<link name="link3" />
<link name="link4" />

<joint name="joint1" type="continuous">
<parent link="link1"/>
<child link="link2"/>
<origin xyz="5 3 0" rpy="0 0 0" />
</joint>

<joint name="joint2" type="continuous">
```

<parent link="link1"/>
<child link="link3"/>
<origin xyz="-2 5 0" rpy="0 0 1.57" />
</joint>

<joint name="joint3" type="continuous">
<parent link="link3"/>
<child link="link4"/>
<origin xyz="5 0 0" rpy="0 0 -1.57" />
</joint>
</robot>

Completar las cinemáticas:

Todavía no se ha definido sobre qué eje gira cada una de las articulaciones del robot. Cuando estén añadidas, el modelo de cinemáticas estará completo. Para definirlas, hay que añadir el elemento <axis> a cada articulación (joint). El campo axis especifica el eje de rotación en el marco local.
Observando joint2, se aprecia que gira sobre el eje Y positivo. Entonces, simplemente hay que añadir el siguiente xml en el elemento joint.
<axis xyz="0 1 0" />
De forma similar, joint1 rota alrededor de los siguientes ejes:
<axis xyz="-0.707 0.707 0" />
Es importante normalizar los ejes.
Del mismo modo ocurre con joint3, rota alrededor de los siguientes ejes:
<axis xyz="-0.707 0.707 0" />

4.3 Archivos y programas necesarios

4.3.1 Programa MESHLAB

MeshLab es un software avanzado de procesamiento de diseño en 3D. Será utilizado para poder obtener los datos de inercia de cada eslabón y para conseguir el modelo 3D de las colisiones.

Obtener datos de inercia:

Para conseguir estos datos no es necesario calcularlo manualmente (muy complicado y llevaría mucho tiempo, a la vez de poco preciso). Para ello hay que utilizar el programa meshlab. Después de cargar la figura 3D, hay que seleccionar: *view → Show Layer Dialog* y se abrirá a la derecha un panel con información (de momento en blanco). Para hallar los valores inerciáles, *Quality measure and computations → Compute geometric measures.* Ahora en el recuadro inferior de *Layer Dialog* aparecerá varia información pero sólo dos son las necesarias:

Center of Mass: Corresponde con el elemento **<origin>**, el cual tiene que ser el centro de gravedad del eslabón.

Inertia Tensor: Corresponde con el elemento **<inertia>**, que es la matriz rotacional inercial (3x3), representada en el marco inercial. Debido a que la matriz rotacional inercial es simétrica, sólo hay que dar los valores de ixx, ixy, ixz, iyy, iyz, izz.

NOTA: Hay que prestar atención a las unidades de la figura cargada. Normalmente suelen estar en metros, pero hay otras que se dan en milímetros. Para saber en qué unidades se encuentra, simplemente hay que seleccionar el icono de metro (Measuring Tool) y medir la figura. Si corresponde con la de la realidad estará en metros, pero si no estará en milímetros (x1000). En este caso; las unidades de medida para el cálculo del centro de gravedad estará multiplicado

por mil (hay que dividir por mil para obtener la correcta) y la matriz inercial estará multiplicada por 10^9 (hay que dividir por 10^9 para obtener la correcta).

Modelo 3D de colisiones:

Como ya se vió en los tutoriales de URDF, los eslabones (links) tienen dos tipos de modelos tridimensionales, el visual y de colisión.

Un **modelo visual** se utiliza sólo con fines de visualización. Estos modelos pueden ser muy detallados, ya que no se utilizan para el control de colisiones. Los modelos visuales pueden ser exportados directamente desde una aplicación de diseño. Hay que asegurarse de que sean STL binarios, por eso es recomendable usar meshconv.

El **modelo de colisión** no debe tener un alto nivel de detalle. Cuanto más detallados sean, más tiempo se tardará en realizar las comprobaciones de colisión. Los modelos de colisión deben ser simples, sin embargo abarcar toda la geometría del eslabón. Por ello, la forma más recomendable de conseguir estos modelos es creandolos a partir del STL visual de esa pieza. Con el programa Meshlab se puede conseguir esto, siguiendo las siguientes instrucciones:
Filters → Remeshing, Simplification and Reconstruction → Convex Hull y después marcar la casilla Re-orient all faces coherently. Ejemplo de un modelo de colisión a partir de su modelo visual:

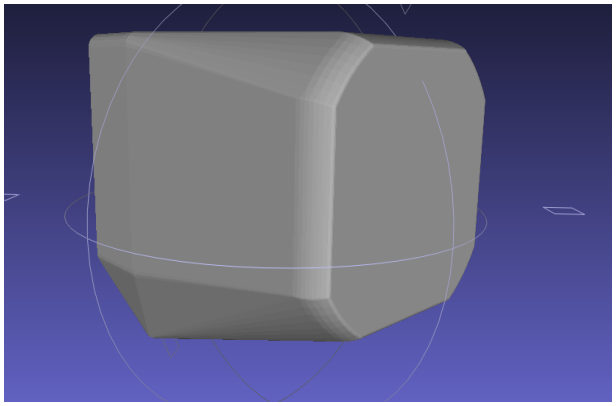

Figura 16: laserPhysics.stl (62kb).

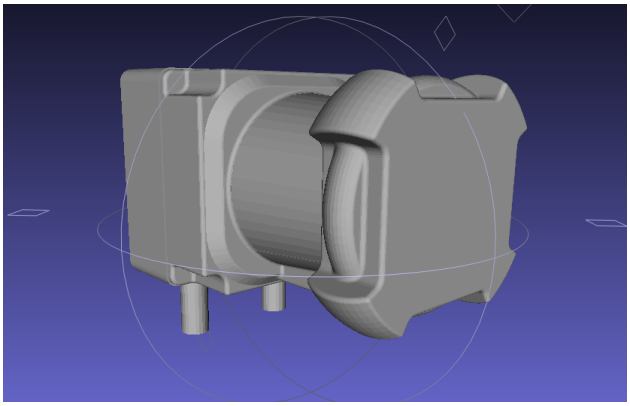

Figura 17: laser.stl (1.2 mb).

4.3.2 Modelo de las figuras 3D

En el caso de MANFRED, las figuras de cada parte del robot están en formato .wrl (virtual reality Modeling Language). Éste tipo de archivo no podrá ser leído por las librerías de ROS, ya que como se mencionó anteriormente, sólo puede leer archivos .dae (COLLADA) y .stl (Stereo Lithography). Lo más sencillo será transformar los archivos wrl en stl.

Convertir el formato WRL a STL:

Hay muchos programas (ya sea en Windows o Linux) que son capaces de leer archivos 3D, como AutoCAD, eDrawing, Inventor Fusion, Freecad o Meshlab…. y convertirlo en cualquier otro formato. Paro utilizarlo en URDF, el archivo se tiene que estar en formato ASCII binario STL para cargarlo y leerlo perfectamente el URDF. Para lograr esto, un método sencillo es emplear el programa meshconv.

Hay que introducir la siguiente instrucción para convertir un archivo .wrl a .stl:
/meshconv -c stl -tri nombrearchivo.wrl

Convertir el formato IV a STL:

Todos los archivos tridimensionales de la mano están en formato .iv (Inventor). Para poder convertirlo a WRL, hay que abrir el archivo con el editor de textos, y cambiar la primera línea que pone:
#Inventor V2.1 ascii por *#VRML V1.0 ascii*
Una vez que está en formato .wrl, hay que emplear meshconv para convertirlo a stl.

4.3.3 Definir los colores de los eslabones

Hay dos formas posibles para colorear las partes del robot:

Archivos de color .png

Consiste en una imagen que tiene el color y forma deseado para la pieza. Para conseguir este archivo hay que editarlo manualmente y llevaría mucho tiempo para lograr que quede bien.

Asignando los valores rgba

Es una forma sencilla de colorear la pieza. Simplemente hay que dar los valores deseados en rojo, verde, azul y la opacidad, que tienen un valor de 0 a 1. Para conseguir el color deseado y saber su valor, hay muchas páginas en internet que proporcionan esta información. Por ejemplo, si se desea conseguir el siguiente color azul:

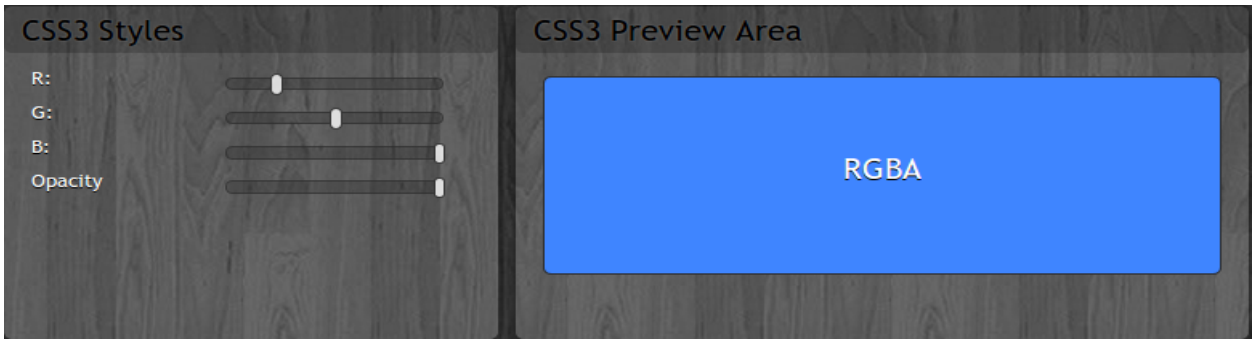

Figura 18: Panel RGBA, ejemplo azul.

En este caso: r=0.2, g=0.5, b=1 y a=1.

NOTA: Todas estas páginas webs proporcionan los valores rgb entre 0 y 255, puesto que es para el lenguaje HTML. Corresponde exactamente en XML a los valores de 0 a 1. El valor de opacidad si coinciden, 1 es totalmente visible y 0 es transparente (invisible).

4.3.4 Archivos XACRO MACRO

Con la razón de que el URDF del robot Manfred será muy extenso, hay que dar un enfoque más general con XACRO. Un XACRO es un tipo de URDF macro que se puede utilizar para generar un URDF. Con XACRO es muy sencillo después personalizar las partes deseadas y generar el URDF personalizado de todo el robot.

Una de la gran ventaja de utilizar XACRO es que si hay un error en la especificación XML, en el momento de generar el URDF, el terminal indica la línea donde se encuentra dicho error. Si el archivo macro no posee ningún error, lo generará satisfactoriamente y cuando se ejecute el URDF no mostrará ninguna anomalía.

Ejemplo de XACRO:

Se quiere modelar un robot a partir de su base y su brazo. Para este caso, los archivos que se necesitarán serán:

base.xacro base_macro.xacro brazo.xacro brazo_macro.xacro union.xacro

En *base_macro.xacro y brazo_macro.xacro* estará todo el código del modelado de esa parte.

En *base.xacro y brazo.xacro* se encuentra la referencia al archivo macro de cada parte, es decir:

```
<xacro:include filename="$(find robot)/urdf/base_macro.xacro"/>

<xacro:include filename="$(find robot)/urdf/brazo_macro.xacro"/>
```

En *union.xacro* se encuentra la referencia a estos dos archivos, y la articulación (joint) que une ambas partes.

Para generar el URDF de este robot, hay que poner la siguiente instrucción:

rosrun xacro xacro.py union.xacro robot.urdf

Donde robot.urdf es el programa, que contendrá la especificación XML de las XACROS, pero de una forma más “limpia y ordenada”. No se recomienda hacer muchos cambios o elaborar directamente el URDF puesto que si después hay algún error en el código será muy difícil detectarlo (el terminal no señala la línea errónea del programa).

Es muy importante que la información dentro del URDF sea exacta. Específicamente, los límites de las articulaciones y orientaciones deben ser correctos. Para visualizar el robot hay que ejecutar lo siguiente:

roslaunch urdf_tutorial display.launch model:=robot.urdf gui:=True

Se abrirá el rviz con el archivo robot.urdf.

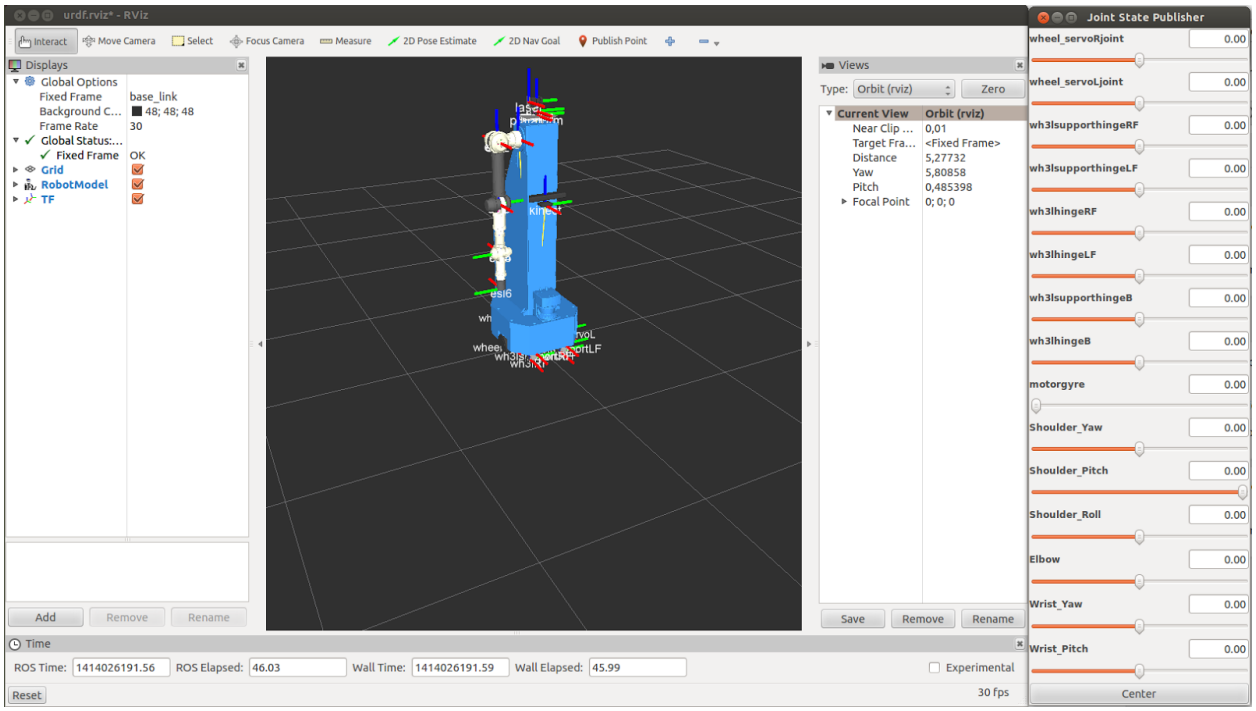

Figura 19: Robot.urf abierto en rviz.

5. Características particulares en el caso de Manfred

5.1 Aclaraciones del código

Lo primero a tener en cuenta es que hay que realizarlo con XACRO macro y después generarlo para conseguir el URDF. Hay que pensar bien las partes de cada xacro, y hacerlo de una forma limpia y poder cambiar si es posible las distintas partes para conseguir el MANFRED personalizado (es decir, cambiar brazos, poner o no la mano, configuraciones en articulaciones…). Por tanto, hay que diseñar la base, los distintos brazos y la mano por separado. Cada parte contendrá dos archivos (como se mencionó anteriormente), el xacro y el macro.xacro.

5.1.1 Base Manfred

Los datos de inertial son calculados con meshlab, y el valor de la masa es el real, en kilogramos. Visual y collision coinciden en posición y en orientación, los archivos de visual son más detallados y los colores son distintos para poder visualizarlo después.

Parte inferior de la base:

La posición de base_link está desplazada 0.0781 en el eje vertical z para que cuando se coloquen las ruedas, la parte inferior de estas toquen con el suelo.
En la figura 20 se muestra la parte inferior de la base (donde están colocadas las ruedas):

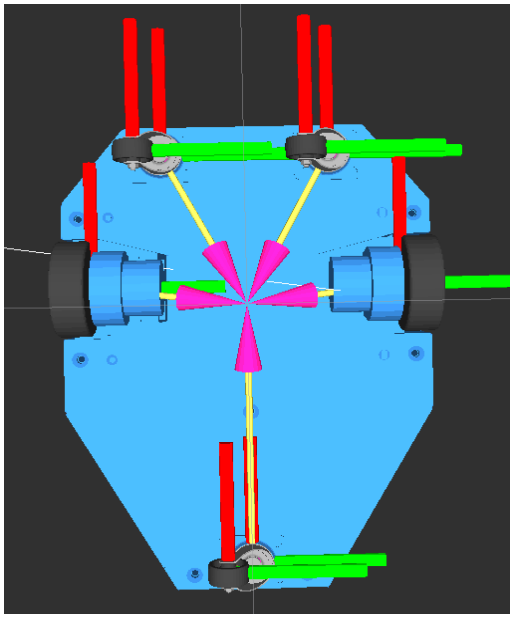

Figura 20: Base inferior Manfred en rviz.

El eje X es el rojo, el eje Y es el verde y el eje Z el azul (este no se aprecia en la imagen). Las flechas amarillas y rosas determinan la distancia de cada hijo a su padre. La posición de las ruedas y sus soportes están definidas con las articulaciones (joints). La rueda derecha lateral wheel_servoR (el hijo) está colocada respecto a base_link (el padre) a una distancia de:
xyz=" 0.02776 -0.2317 0.0763". De igual manera la rueda izquierda estará situada a: xyz=" 0.02776 0.2317 0.0763". Ambas ruedas al girar sobre sí mismas serán de tipo continuo (no tienen límites) y lo harán sobre el eje Y. La velocidad máxima angular viene dada en 1.57 radianes. De forma análoga se colocarán los tres soportes de las ruedas auxiliares (wh3lsupporthinge). Estos soportes giran sobre sí mismos en el eje Z, para que gire hacia los lados y controle la dirección de giro. Para colocar las ruedas auxiliares, hay que hacerlo respecto a sus padres, los soportes. Por tanto, como los tres soportes y rueda son exactamente iguales, la posición de la articulación será: xyz="0 -0.035 -0.05". Para determinar su dirección de giro, hay que apreciar el eje que atraviesa transversalmente la rueda, que es el eje x; por tanto: axis xyz=" 1 0 0".

Parte superior de la base:

En la figura 21 se muestra la base superior, que está compuesta por el aparato kinect y el sistema del láser Hokuyo.

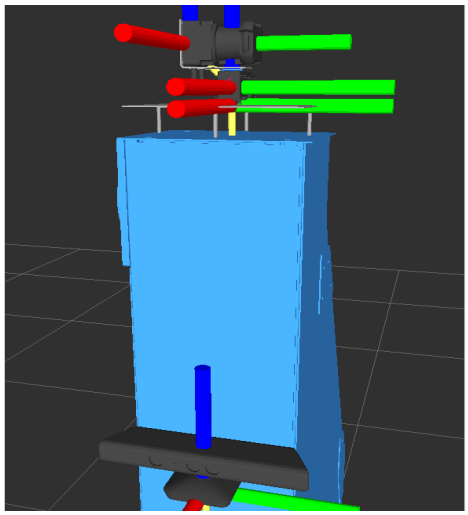

Figura 21: Base superior Manfred en rviz.

Para colocar el kinect, hay que tener en cuenta la altura (eje Z) y que quede junto al lateral de manfred (eje X). Por ello, su posición será: xyz= “0.16 0 1.12”. Al ser una articulación fija (no se mueve) hay que introducir el atributo fixed. Si se desea variar la inclinación del kinect, simplemente hay que introducir el valor deseado (en radianes) en rpy=” 0 valor 0”.

La plataforma estará situada en la parte superior de base_link, anclada mediantes cuatro tornillos. La plataforma se creará como si fuese una caja de 0.18x0.22 con un espesor de 0.002 en metros. Los tornillos se crearán como cilindros de radio 0.0025 y una longitud de 0.004 metros. Los tornillos se colocarán directamente a partir de la figura creada por la plataforma (box). La altura en z será de -0.02, es decir, en la parte inferior de la plataforma a una distancia de la mitad de su longitud. Al ser cuatro, y colocándolos en la forma que indica la imagen superior:

Tornillo 1: y = 0.1 (la mitad de 0.22 - 0.01).
Tornillo 2: x = -0.08 análogo a tornillo 3.
Tornillo 3: x = 0.08 (la mitad de 0.18 - 0.01).
Tornillo 4: y = -0.1 análogo a tornillo 1.

Hay que tener en cuenta que el láser estará situado al final de la siguiente estructura de árbol:

base_link
→ plataforma
→ motor
→ disco
→ chapa
→ láser

•plataforma: Colocada a xyz= "0.03 0 1.68" respecto del origen de la base. La articulación plataformsupport es fija (fixed, no se mueve).

•motor: Este link ha de orientarse de forma tumbada, para ello rpy="0 0 -1.57" (en link list).Está a 2.5 cm en el eje X de su padre la plataforma. El motor no gira, lo hace el disco, por eso esta articulación (motorsnag) está fija.

•disco: Al haber sido recortada esta pieza a partir de la original del motor, tiene su offset respecto al motor, así que no hay que moverla; xyz= "0 0 0". La articulación motorgyre es de tipo revolución sobre el eje Z, hasta 270º= 4.71 rad.

•chapa: Su posición esta recalibrada en el eslabón chapa (puesto que fue creada manualmente, tiene offsets inusuales) y además ha de estar orientada para que encaje con la pieza disco y laser (rpy="1.57 3.14 0"). Al estar correctamente colocada en link list, no hay que definir su posición en la articulación discochapasnag (fixed).

•láser: Ubicado a xyz= “0 -0.054 0.06” de su padre la chapa. También tipo fixed porque no se mueve. Este eslabón también tiene que orientarse como aparece en la imagen, por tanto rpy=”0 -1.57 0”, en link list.

NOTA: Todas las articulaciones son rotacionales. Por este motivo, se ha establecido que la velocidad máxima (limit velocity) es de 1.57 rad/s.

5.1.2 Brazo Manfred1

En la figura 22 se muestra el brazo UC3M-LWR-1, donde se aprecian los siete eslabones que lo componen:

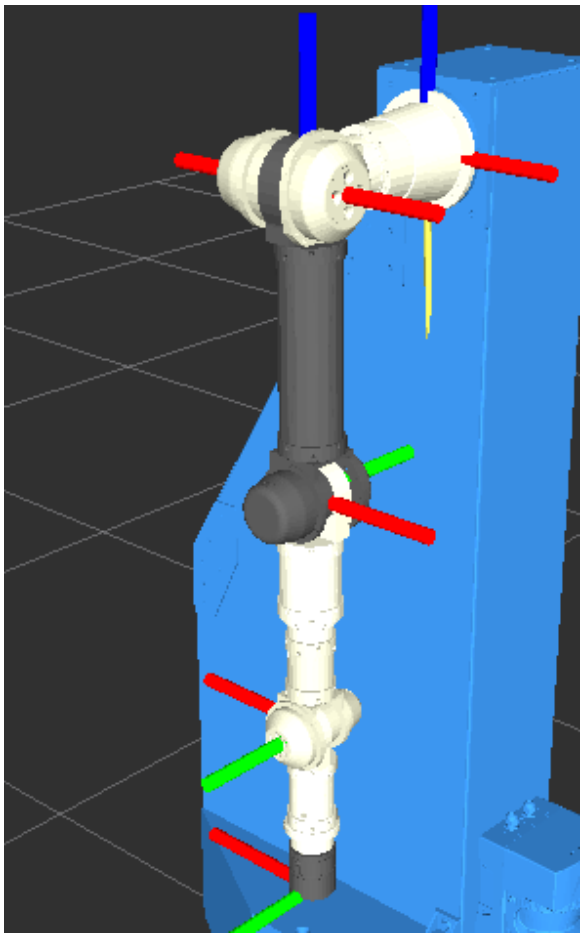

Figura 22: Brazo Manfred1 en rviz.

La estructura del árbol del brazo es el siguiente:

esl0

→ esl1

→ esl2

→ esl3

→ esl4

→ esl5

→ esl6

Todos los eslabones son colocados directamente en las articulaciones, por lo tanto en la sección de eslabones(link list) no hace falta definir su posición (o bien forzandolo a 0) en los elementos visuales y colisionales. En estas dos últimas lo único que difiere son los archivos tridimensionales y sus colores. Para completar la sección inercial es necesario meshlab para hallar sus valores, como se explicó anteriormente.

Shoulder_Yaw (entre esl0 y esl1): Hay que definir rpy=”0 3.14 0” (es decir, girarlo media vuelta), puesto que el brazo está orientado hacia el suelo. esl0 tiene una longitud de 25 cm, por tanto xyz=” 0 -0.25 0”, en el eje Y negativo. Apreciando el sistema de coordenadas de esta articulación (que es la misma de esl1) para lograr un movimiento lateral, simplemente tiene que rotar sobre el eje Y.

Shoulder_Pitch (entre esl1 y esl2). Como en la articulación anterior hay que reorientar su posición con rpy=”0 3.14 0”. Su posición prácticamente coincide con esl1, pero tiene un pequeño margen de apenas 2 cm, para dejarlo más ajustado. Esta articulación tiene un movimiento “hacia adelante”, por tanto hay que girarlo sobre el eje X.

Shoulder_Roll (entre esl2 y esl3). Ubicado a 40 cm desde esl2. Apreciando la figura, tiene que ser xyz=”0 0 -0.4”. Esta articulación ya está bien orientada. El movimiento es análogo al de Shoulder_Yaw, por tanto tiene que girar sobre Y.

Elbow (entre esl3 y esl4). Ubicado a 34.5 cm desde esl3. Apreciando la figura, tiene que ser xyz=”0 0 -0.345”. Es necesario reorientar la articulación para que esté boca abajo (rpy=”0 3.14 0”). Tiene el movimiento parecido al de la muñeca, es decir tiene que girar sobre Z (entre -90 y 90).

Wrist_Yaw (entre esl4 y esl5). De forma similar a esl1 y esl2, la posición de esl5 está en esl4. Ya está bien orientado, por lo tanto:
xyz= “0 0 0” rpy=”0 0 0”. Su movimiento es lateral, entonces deberá de girar sobre Y.

Wrist_Pitch (entre esl5 y esl6).El eslabón esl6 está colocado a 26.5 cm (que es la longitud de esl5). Al estar bien orientada, su posición queda definida por xyz= “0 0 -0.265” rpy=”0 0 0”. Se mueve con el eje Z.

NOTA: Los datos de cada eslabón están recogidos en la tabla 1 de las especificaciones del brazo de Manfred UC3M-LWR-1 [1].

Name	Mass (kg)	Rotation Range	Angular Velocity(º/s)	Repeatable Peak Torque(Nm)
esl0	3.79	-	-	-
esl1	4.19	± 90	131	176
esl2	3.25	- 90	131	176
esl3	1.67	± 90	131	92
esl4	1.66	± 90	180	54
esl5	1.47	± 90	180	54
esl6	0.58	± 90	210	54

Tabla 1: Datos del brazo UC3M-LWR-1 (manfred1).

5.1.3 Mano Gifuhand III

A continuación se muestra la figura 23 de la mano completa. Se aprecian los sistemas de coordenadas de la palma, el dedo pulgar, el meñique y dos en el dedo medio.

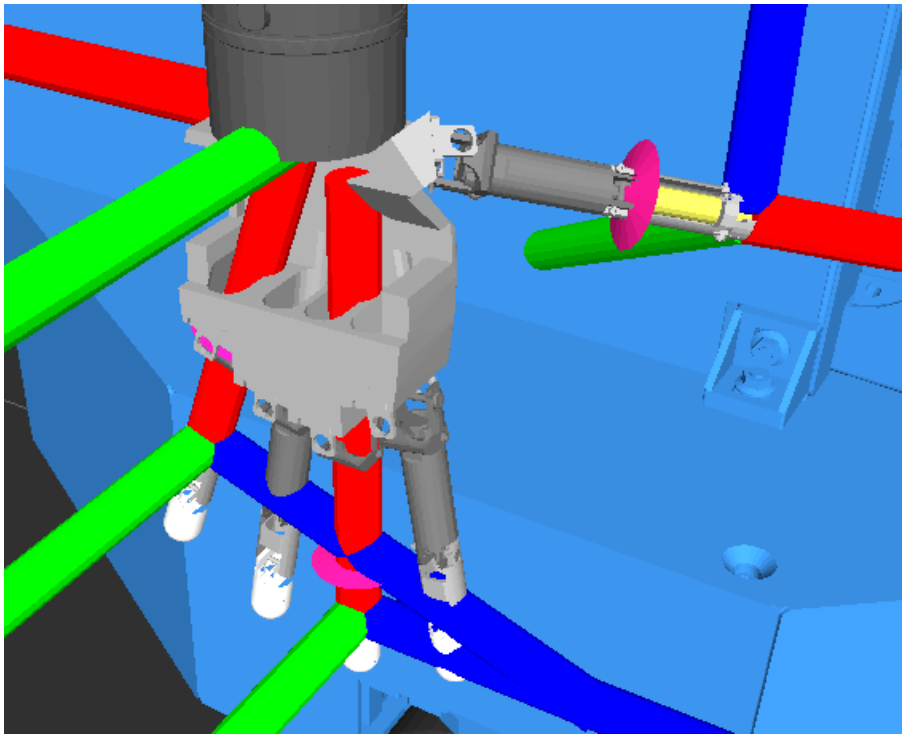

Figura 23: Mano Gifuhand III en rviz.

Masa de la mano:

En la tabla 2 se muestra la masa de la mano, extraída de la hoja de características de Gifuhand III [10].

Weight	Thumb	0.25 [kgf]
	Finger	0.20 [kgf]
	Total	1.4 [kgf]

Tabla 2: Masa de gifuhand3.

La mano entera (con dedos incluidos) tiene que pesar 1400 gramos. La palma 350 g, el pulgar entero 250 g y el resto de dedos 200 g cada uno. Cada dedo está compuesto por tres falanges. Las dos primeras falanges del pulgar pesa 100 g y la tercera 50 g. Para los otros cuatro dedos, la primera pesa 100 g y las otras dos 50 g.

<!-- link list >:

Primero hay que comprobar que los modelados de los eslabones (links) estén bien orientados. Como tanto la palma como las falanges no lo están, hay que orientarlos dándole valores a rpy. Para el caso de la palma, será rpy="1.41 2.2 -0.85"; de esta manera la palma está colocada de la forma que aparece en la figura 24. En el caso de las falanges, hay que voltearlas. Para ello, hay que ponerlas paralelas al plano del suelo, por tanto rpy=" 3.14 3.14 0". En el caso del pulgar, hay que poner rpy="3.14 3.14 3.14" para darle media vuelta y que su ranura coincida con la de la palma (en el punto tconexion).

Hay que fijarse también que los archivos 3D obtenidos están a una escala de 1:1000. Por ello, hay que darle su valor real introduciendo scale="0.001 0.001 0.001" dentro del elemento mesh.

Cada dedos tiene dos articulaciones en el lugar donde se une con la palma. Es decir, los dedos se abren y se cierran lateralmente (en abanico); y también se mueven hacia adelante para cerrar el puño. Por esta razón, hay que utilizar un eslabón adicional para este propósito. Será el link conexion, uno para cada dedo. Este eslabón es una esfera con 10 picómetros de radio, por lo tanto se despreciarán las propiedades inerciales.

Como los archivos 3D de la mano son pequeños (200 kb a 50 kb) no hace falta crear los archivos para el colisionado. Se usarán los mismos tanto para visual como para colisionado.

<u><!-- joint list >:</u>

Para facilitar la comprensión de las articulaciones, se muestran la siguientes imágenes:

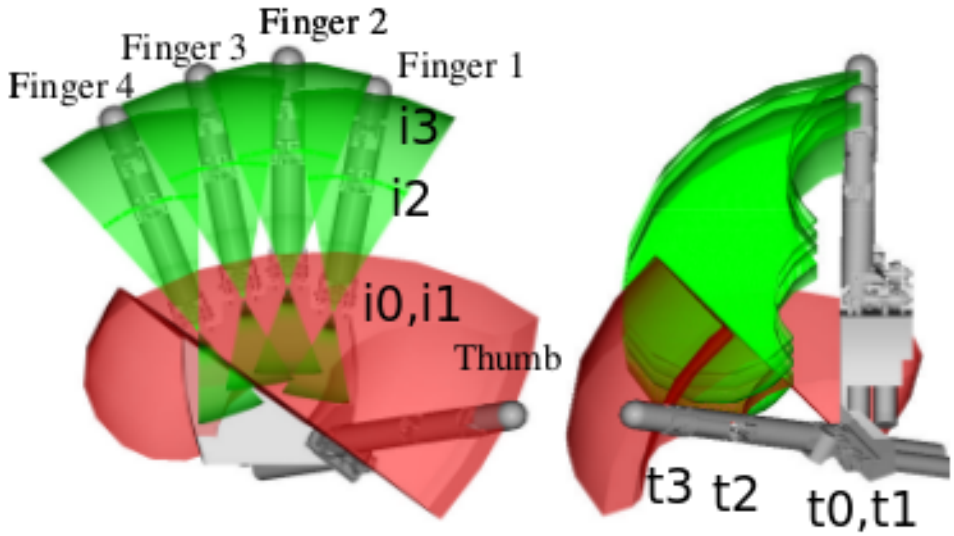

Figura 24: La movilidad de las articulaciones de Gifuhand III.

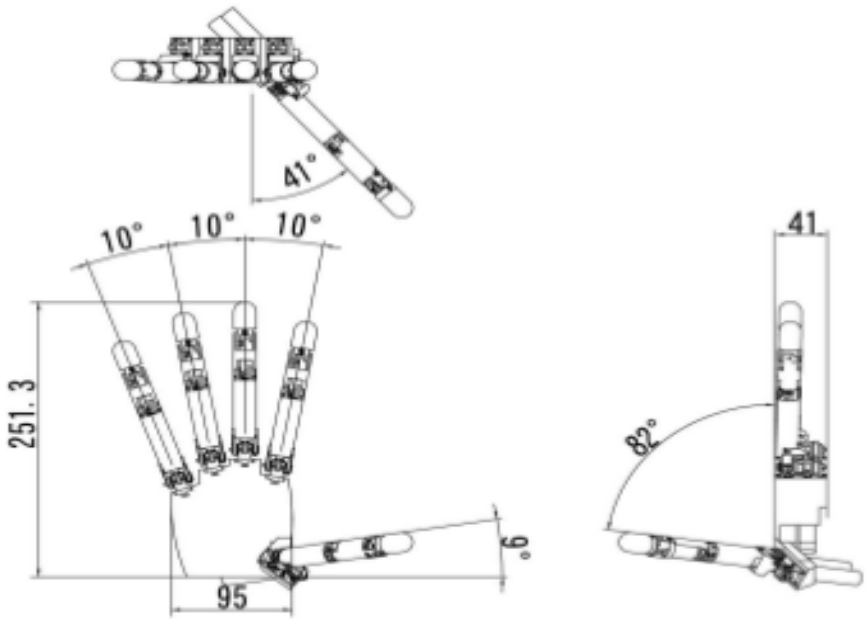

Figura 25: La posición en reposo de GifuHand III.

•Pulgar:

En primer lugar, hay que localizar la posición en la que se encontrará t0. Para ello es aconsejable utilizar la herramienta Measure de rviz, e ir midiendo desde el origen de la palma a ese punto en cada eje. En este caso xyz=”-0.039 -0.02 -0.012”. Ahora hay que orientarlo, para ello hay que fijarse bien en la figura 25. Una forma parecida a esta posición es con rpy=”2.61 2.88 0.7”. Como el movimiento de t0 es lateral, tiene que rotar sobre el eje Y.

Ahora hay que fijarse en la figura 24 para saber donde colocar t1, que es la misma posición de t0, por tanto no hay que moverlo. Observando la figura 23, el eje Z es el necesario para que el pulgar gire hacia la palma.

Para colocar t2, simplemente hay que fijarse en la figura 23, ver que el eje X (el rojo) está colocado en el dedo, y medir la distancia de thumb 02, que es 7.05 cm. No hay que orientarlo puesto que ya está orientado correctamente. Se mueve alrededor del eje Z como en t1.

Por último, t3 hay que colocarlo de forma similar a t2 y tampoco hay que orientarlo. Por ello xyz=”0 0 0” y rpy=”0 0 0”. Al igual que t1 y t2, t3 también gira en Z.

En el manual de Gifuhand3 [10] aparecen las especificaciones requeridas. Ahí aparecen los valores de los límites de giro y su par de cada falange.
Tabla de los límites de los ángulos:

Operating angle of joints	1st joint	-28 ~ 28 [deg] (Thumb) -20 ~ 20 [deg] (Finger)
	2nd joint	-10 ~ 90 [deg]
	3rd joint	-10 ~ 90 [deg]
	4th joint	-10 ~ 90 [deg]

Tabla 3: Límites de giro de Gifuhand III.

Output torque of the thumb	1st joint	1.76 [Nm] [1], 1.03 [Nm] [2]
	2nd joint	1.27 [Nm] [1], 0.58 [Nm] [2]
	3rd joint	0.33 [Nm] [1], 0.25 [Nm] [2]
	4th joint	0.04 [Nm] [1], 0.02 [Nm] [2]

Tabla 4: Par máximo de las articulaciones de Gifuhand III.

•Dedo mediano

Hay que colocar m0 (que es la misma posición que mconexion) en el lugar que indica la figura 24. Se empleará el mismo procedimiento que para colocar t0. Ahora hay que orientarlo, de forma que su posición de reposo según indica la figura 23. De esta forma queda xyz=”-0.015 -0.0045 -0.1164” y rpy=”0 -1.57 0”. Para conseguir el movimiento lateral (al igual que con t0) hay que moverlo en el eje Y.

Las siguientes articulaciones se definen de forma similar a las del pulgar (cambia la distancia de la 3er falange a 3.2 cm en m3). Según la descripción de Gifuhand3, la última falange de estos cuatro dedos tiene que imitar a la anterior. Es decir, el movimiento de m3 es igual al movimiento de m2 (m3 no se mueve voluntariamente). Por ello, hay que añadir el siguiente elemento en la articulación m3:

<mimic joint=”i2” multiplier=”1” offset=”0”/>

•Dedo índice

Fijándose en la figura 25, se puede observar que el ángulo que forma entre cada uno de los cuatro dedos es 10º (aprox 0.17 rad). Para orientar i0, hay que girar 0.17 en sentido positivo. Por tanto rpy=”0 -1.4 0”. Para obtener la posición de xyz hay que hacerlo como se hizo con t0.

•Dedo anular

Para orientar r0, hay que girar 0.17 en sentido negativo respecto de m0, de tal forma quedará rpy=”0 -1.74 0”. Su posición y la descripción del resto de articulaciones es análogo a los dedos anteriores.

•Dedo meñique

Para orientar l0, hay que girar 0.17 en sentido negativo respecto de r0, de tal forma quedará rpy=”0 -1.74 0”. Su posición y la descripción del resto de articulaciones es análogo a los dedos anteriores.

5.1.4 Unión Manfred1

Tiene que hacer referencia a los tres macro.xacro de las partes, para poder utilizarlas. Las uniones son articulaciones, por tanto, hay que definirlas. Union es la articulación que hay entre base y brazo, como no se mueve, es tipo fixed. Está ubicada a esa distancia desde la base. La articulación unionmano es la que une el brazo con la mano, también fixed puesto que no se mueve (se mueve el eslabón esl6 y entonces mueve la mano).Como el sistema de coordenadas de esl6 coincide con el de palm, como se muestra en la siguiente imagen, no hay que definir su posición.

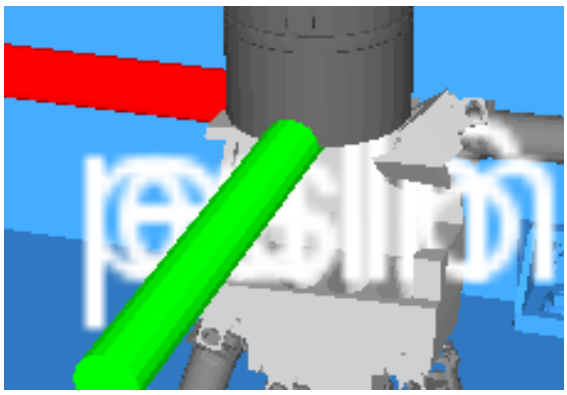

Figura 26: Unión de la mano rviz.

Para generar el archivo URDF completo, hay que introducir la siguiente instrucción en el terminal:
rosrun xacro xacro.py manfredunion1.xacro > manfred1.xacro
Si todo el código ha sido escrito correctamente, se creará el archivo URDF en esa carpeta. En caso que hubiese algún error, el terminal señalará que archivo falla y en qué línea de código.

5.1.5 Manfred2

El primer eslabón del brazo UC3M-LWR-1 tiene la posibilidad de ser seccionado (se puede desmontar un trozo de eslabón de 9 cm de largo). Por ello, hay que crear el eslabon0 recortado, para ello hay que utilizar meshlab y quitar ese trozo. El eslabon0 modificado se llamará esl0c, y tendrá otras propiedades inerciales, por lo que hay que calcularlas (con meshlab). Al ser 9 cm más corto, esl1 tiene que estar colocado a -16 cm (en lugar de -25 cm).El código es igual al de manfredunion1 pero sin la mano, ya que al colocar la mano, esta colisionaría con la base de Manfred y no podría operar. Además, tiene que mencionar a la macro.xacro de manfredbrazo2. El nuevo manfred2.urdf quedará de la siguiente manera:

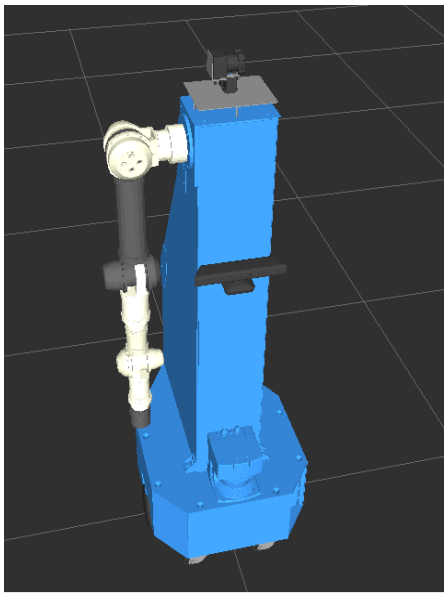

Figura 27: Imagen de Manfred2 en rviz.

5.1.6 Manfred3

En la sección 2.2.5, se describió otro brazo para Manfred totalmente distinto y que presenta 7 grados de libertad. El nuevo brazo extendido queda de la siguiente manera:

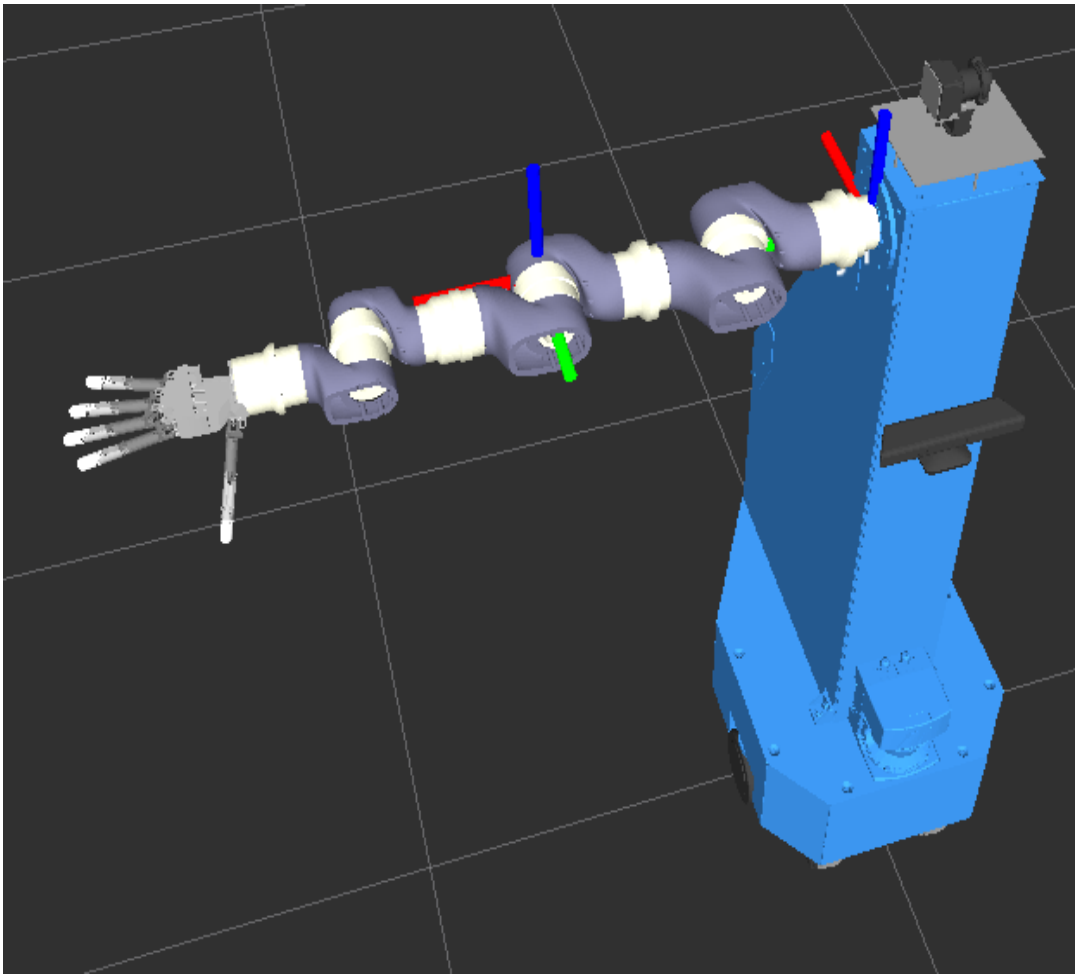

Figura 28: Imagen de Manfred3 en rviz.

Presenta siete eslabones que giran (los blancos, correspondientes a j1.1, j1.2, j….) y seis eslabones fijos (los morados, correspondientes a e1.1, e1.2, e…). Por tanto, presenta siete grados de libertad.

A continuación se muestra toda la información proporcionada para poder elaborar definir la configuración del brazo Manfred 3 [2]:

Articulación Hombro	J1.1	1,9
Eslabón tipo 1	E1.1	0,74
Articulación Hombro	J1.2	1,9
Eslabón tipo 1	E1.2	0,74
Articulación Hombro	J1.3	1,9
Eslabón tipo 2	E2.1	0,532
Articulación Codo	J2.1	1,4
Eslabón tipo 2	E2.2	0,532
Articulación Codo	J2.1	1,4
Eslabón tipo 3	E3.1	0,235
Articulación Muñeca	J3.1	1,1
Eslabón tipo 3	E3.2	0,235
Articulación Muñeca	J3.2	1,1

Tabla 5: Enumeración de eslabones y masas de Manfred3.

Name	Mass (kg)	Rotation Range	Angular Velocity (°/s)	Repeatable Peak Torque (Nm)
Shoulder (DOF 1)	1,9	± 170°	82,5	123
Shoulder (DOF2)		± 100°		
Shoulder (DOF3)		± 170°		
Elbow (DOF4)	1,4	± 100°	131,25	64
Elbow (DOF5)		± 170°		
Wrist (DOF 6)	1,1	± 100°	180	29
Wrist (DOF 7)		± 170°		

Tabla 6: Límites de giro y par máximo de Manfred3.

<!-- link list >:

Lo primero que hay que hacer es mover cada pieza sobre su eje para que coincida y quede centrada la pieza. Por ejemplo, con el eslabón j11 hay que ajustarlo con: origin xyz=”-0.058 0 -0.058”. Hay que hacer esto con todos los eslabones del brazo para evitar que el brazo realice movimientos no deseados. También hay que tener en cuenta que todos los archivos 3D están a escala 1:1000, por lo que hay que dividir entre 1000 para obtener el valor real.

<!-- joint list >:

Del mismo modo que ocurría con Manfred1 y Manfred2, la primera articulación estará definida en el archivo manfredunion3.xacro, que es shoulder1 (j1.1). La siguiente articulación se mantendrá fija (A) puesto que corresponde con e.1.1, y la siguiente que se moverá será shoulder2 (j1.2). La siguiente se mantendrá fija (B) y la siguiente girará, y lo mismo ocurre con las siguientes, hasta llegar al final. Por ello, todas las articulaciones giran sobre el eje Y, por tanto hay que definir axis <xyz= “0 1 0”/>. En cuanto a los valores de velocidad, límites y de fuerza, hay que ir completando según lo indicado en la tabla 6.

Orientación de los eslabones:

•e1.1, e2.1 y e3.1 tienen la misma orientación, por tanto: rpy=”0 0 0”.
•e1.2, e2.2 y e3.2 van colocados en sentido opuesto a los eslabones anteriores, entonces: rpy=”0 0 1.57”.

Orientación de las articulaciones:

•j1.1, j1.3, j2.2 y j3.2 están colocados en la misma dirección, cuyo valor es rpy=”0 0 1.57”.
•j1.2, j2.1 y j3.1 van orientados de la misma posición, que es rpy=”0 0 1.57”.

5.2 Visualización con rviz

Para comprobar que está bien construido, hay que ejecutarlo con la siguiente instrucción:

roslaunch urdf_tutorial display.launch model:=manfred1.urdf gui:=True

Esto abrirá el archivo URDF con la aplicación rviz y joint state publisher de ROS, tal y como se muestra en la siguiente imagen:

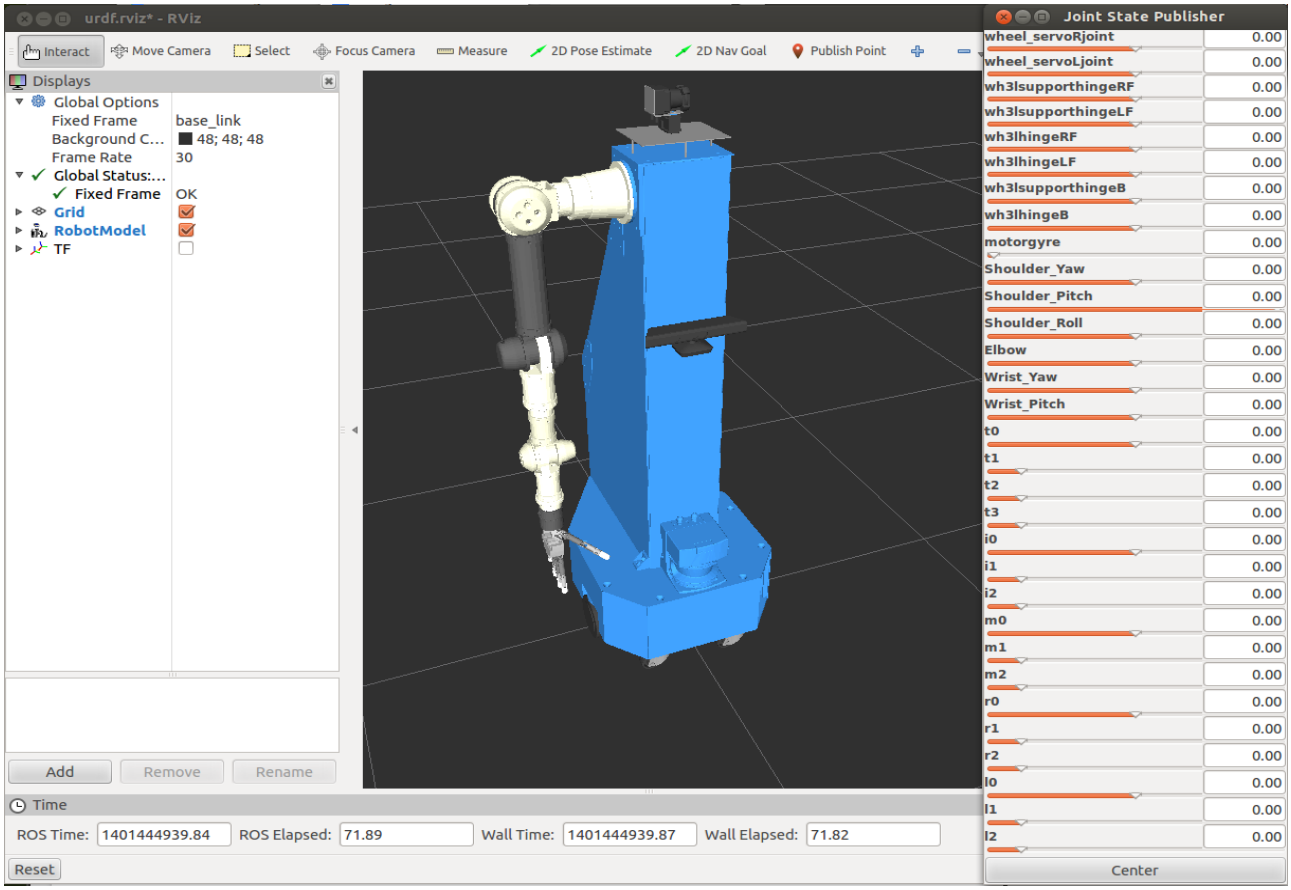

Figura 29: Imagen de Manfred1en rviz.

Displays:

En la parte izquierda de rviz, aparece la sección displays. Aquí se puede seleccionar qué opciones y figuras que se desea mostrar:

• RobotModel: Puede mostrar el robot con visual o colisión. Además, se puede seleccionar qué eslabones (links) muestre por pantalla.

• TF (Transform Frame). Es el marco de transformadas de cada link. Puede mostrar el nombre, los ejes del sistema, y la flecha que indica su referencia padre-hijo. Se puede seleccionar qué eslabones muestre su TF.

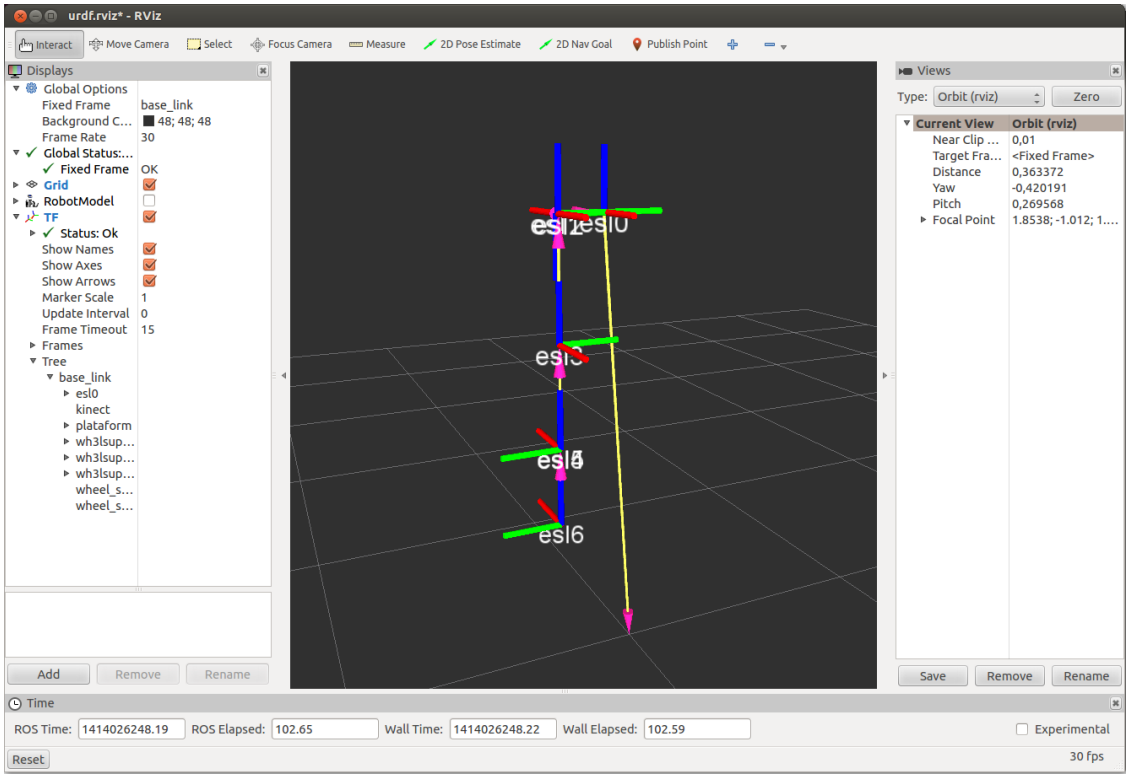

Figura 30: Transform Frame de rviz.

Herramientas:

- Select: Selecciona los eslabones que hay en el recuadro seleccionado.
- Focuscamera: Muy útil para centrar y ajustar qué zona mostrar por pantalla.
- Measure: Mide la distancia en metros desde un punto a otro. Especialmente útil para saber la separación entre dos eslabones.

Joint State Publisher:

Muestra todas las articulaciones del robot, y permite girarlas. Aparecen los límites superiores e inferiores de las articulaciones, y en el caso de las articulaciones continuas (como las ruedas), desde $-\pi$ a π. Para asegurarse de que está bien definidas las articulaciones, hay que comprobar cada una de ellas moviendolas.

5.3 Configuración de Moveit!

El Asistente de Configuración MoveIt! es una interfaz gráfica que se utiliza para configurar cualquier robot. Su función principal es la generación de un archivo SRDF (Semantic Robot Description Format) para el robot. El SRDF complementan la URDF y especifica los grupos de articulaciones, las configuraciones por defecto de robots, información de comprobación de colisión adicional y transformaciones adicionales que puedan ser necesarios para especificar completamente la posición inicial del robot.

Iniciar el asistente de configuración:

Para iniciar moveit!, hay que hacer la siguiente instrucción:
roslaunch moveit_setup_assistant setup_assistant.launch

Arrancará la pantalla inicial con dos opciones: Create New MoveIt! Configuration Package o Edit Existing MoveIt! Configuration Package. Hay que seleccionar la primera opción para crear el paquete de configuración para manfred3. Para ello, hay que buscar la ruta en la cual se encuentre el archivo manfred3.urdf. Después de cargar todos los datos, aparecerá la pantalla principal con la figura del robot:

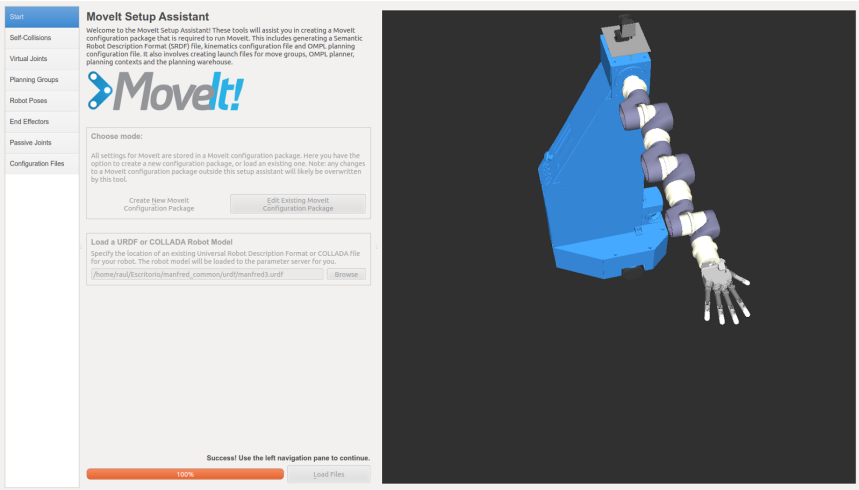

Figura 31: Asistente Moveit! con Manfred3.

Generar la matriz de Auto-Colisión

El generador de matriz Auto-Colisión busca pares de eslabones del robot que con seguridad se pueden desactivar en el chequeo de colisiones, disminuyendo el tiempo de procesamiento de planificación de movimiento. La densidad de muestreo especifica cuántas posiciones del robot al azar para comprobar si hay auto colisión. Las densidades más altas requieren más tiempo de cálculo y las densidades más bajas tienen una mayor posibilidad de deshabilitar pares que no deben ser desactivados. El valor predeterminado es de 10.000 controles de colisión.

Hay que seleccionar la opción *Self_collisions* en el panel de la izquierda y después en el botón *Generate Collsion Matrix*. El asistente buscará los posibles pares de eslabones y los representará en una tabla:

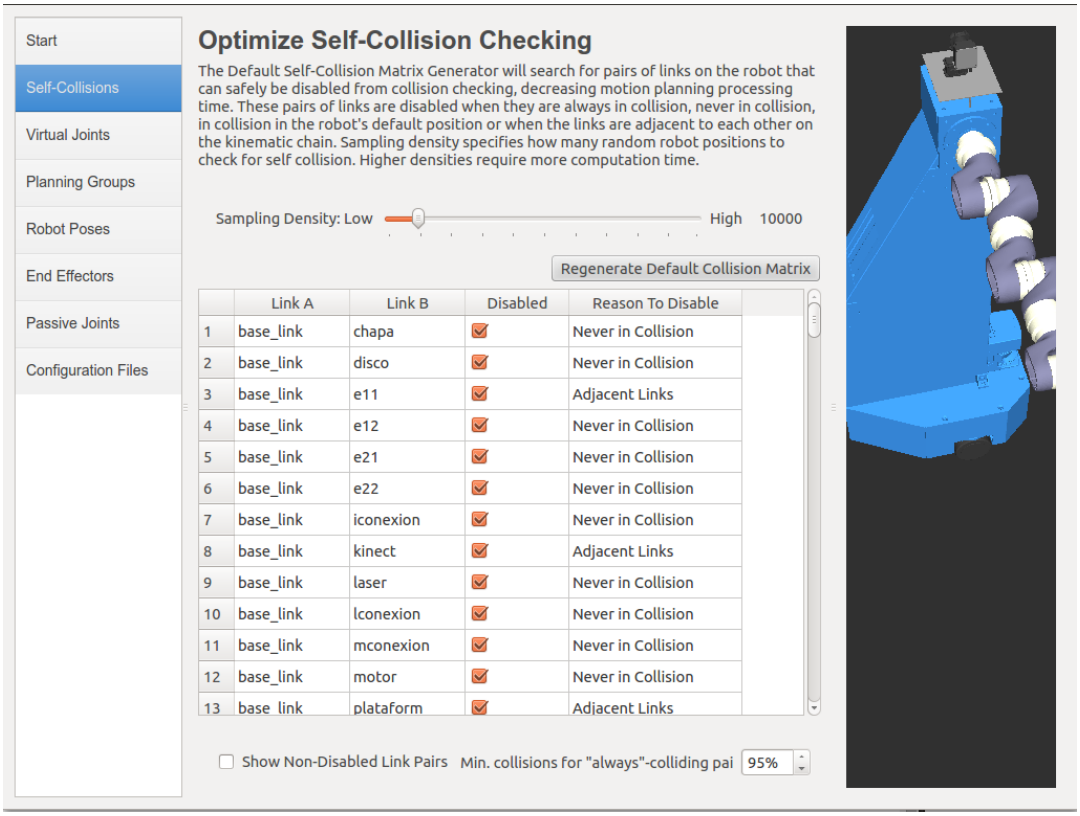

Figura 32: Captura de Auto-colisión.

Bajando hasta el final de la tabla, se aprecia que hay un total de 726 pares.

Añadir articulaciones virtuales:

Las articulaciones virtuales se utilizan principalmente para fijar el robot con respecto el mundo (o suelo). Hay que seleccionar *Virtual joints* de el panel izquierdo, y luego en *Add Virtual Joint.* Aquí hay que nombrar la articulación virtual, seleccionar el eslabón hijo y el marco del padre (que será odom_combined), y en *Joint Type* dar el valor de *planar*; y después *Save*. Como manfred tiene cinco puntos de apoyo con el suelo (dos ruedas laterales y tres de apoyo) hay que crear cinco articulaciones virtuales; como se muestra en la siguiente imagen:

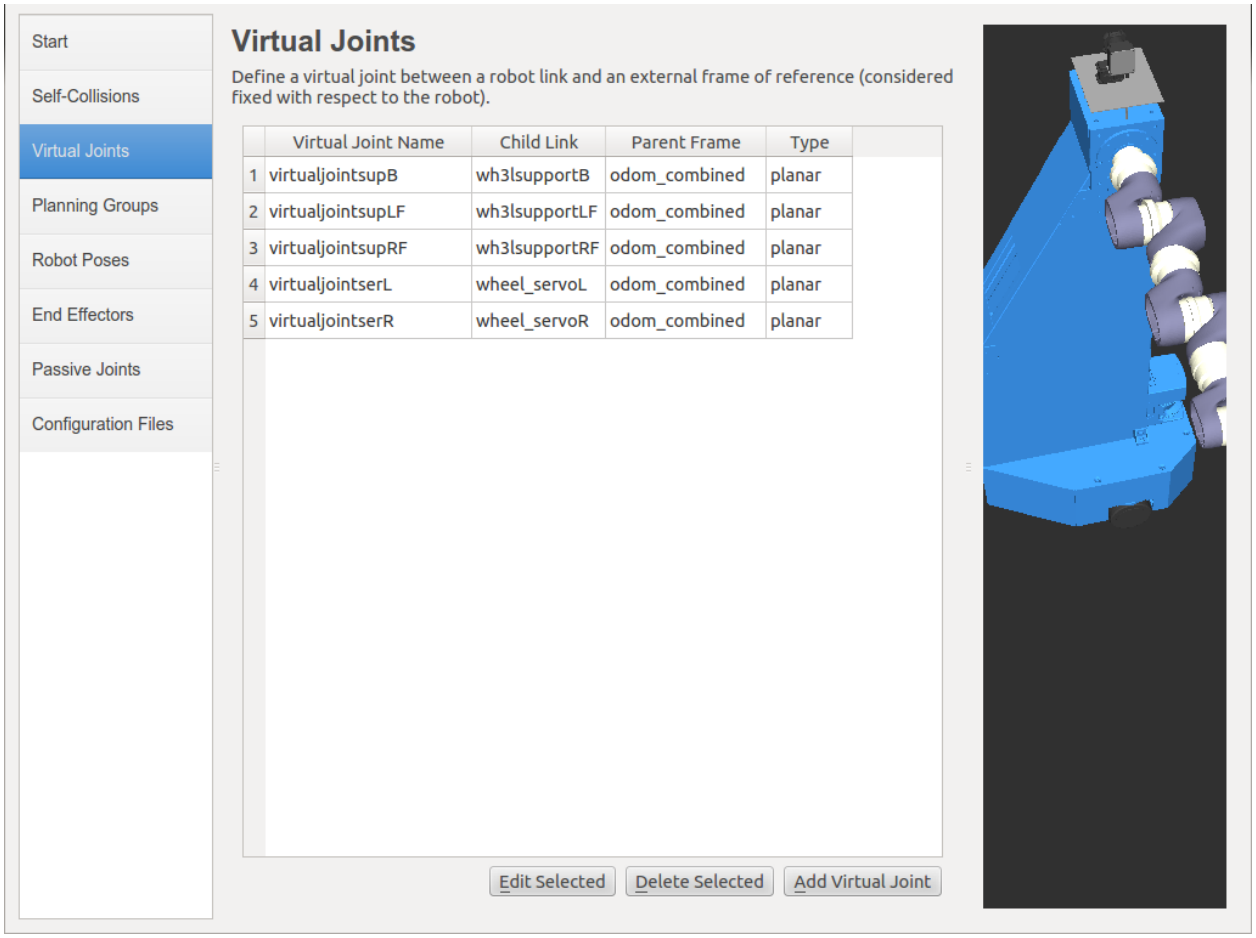

Figura 33: Captura de articulaciones virtuales.

Añadir grupos de planificación:

Los grupos de planificación son usados para describir semánticamente diferentes partes del robot, como por ejemplo definir lo que es un brazo. Para ello, hay que seleccionar *Planning Groups* en el panel izquierdo, y al pulsar *Add Group*, aparecerá la siguiente pantalla:

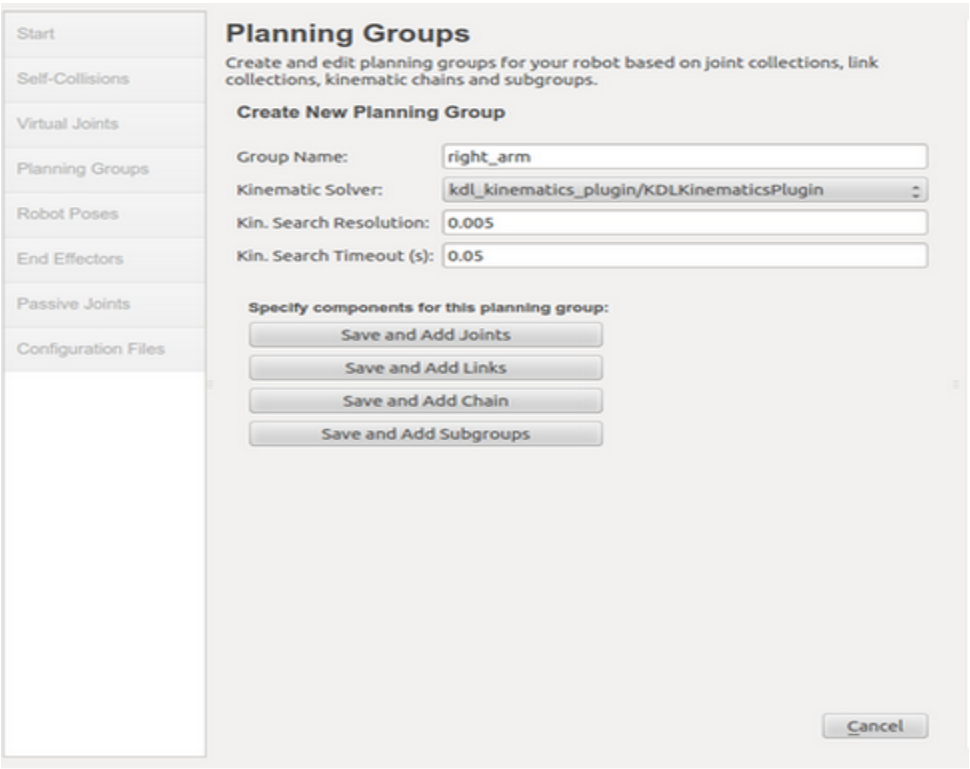

Figura 34: Captura de grupos de planificación.

Hay que poner el nombre del grupo, que es right_arm, y seleccionar KDLKinematicsPlugin en Kinematic Solver y pulsar *Save and Add Joints*. Aparecerán dos columnas (la izquierda con todas las articulaciones del robot y la de la derecha vacía), y hay que seleccionar las articulaciones que se correspondan al brazo. Estas articulaciones son los shoulder, elbow y wrist. Para finalizar hay que darle a *Save*.

De éste mismo modo, hay que hacer el grupo mano y base con todas sus articulaciones correspondientes.

Añadir la posición inicial:

Es posible definir una posición definida en la configuración, como por ejemplo una de reposo. Para ello hay que seleccionar *Robot Poses* en el panel izquierdo. Dentro de *Add Pose* hay que elegir un nombre. El robot mantendrá su posición de defecto en la que todas las articulaciones están con valor de 0. Hay que mover cada articulación para conseguir que el robot obtenga la posición que el usuario desee.

Añadir articulaciones pasivas:

La tabla de articulaciones pasivas permite la especificación de cualquier articulación pasiva que puede existir en un robot. Como en el caso de manfred no tiene, no hay que incluir ninguna.

Generar los archivos de configuración:

Una vez que esté el robot configurado, hay que guardar los archivos para poder ejecutarlos con moveit! después. Para ello hay que seleccionar *Configuration Files* en el panel izquierdo. Hay que elegir un lugar y el nombre del paquete ROS en el cual contendrá todo el conjunto de archivos. Para finalizar, hay que pulsar *Generate Package button*. Esto generará todos los archivos, tanto iniciadores como de configuración, dentro de la carpeta seleccionada.

6. Resultados

6.1 Las partes de Manfred en rviz.

Mediante capturas de pantalla, se mostrará cada parte del robot y al lado un ejemplo de las articulaciones movidas.

Ruedas Manfred:

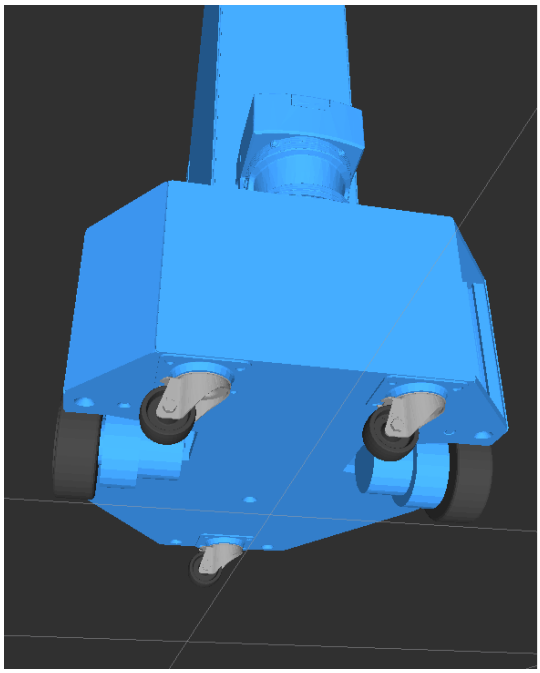

Figura 35: Ruedas rviz.

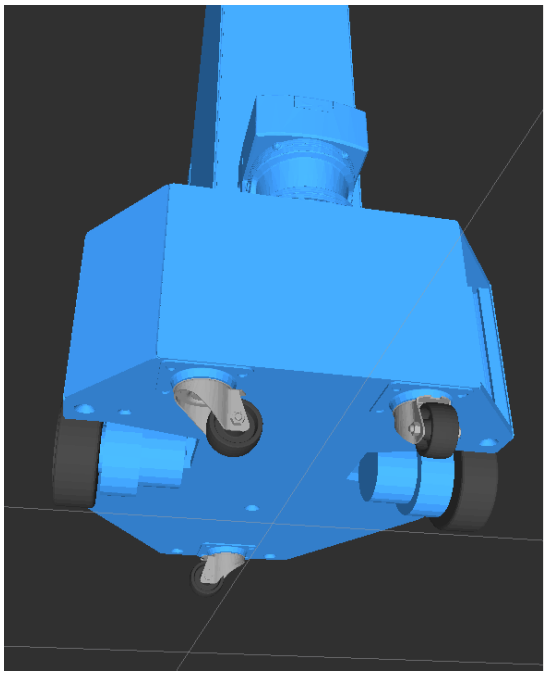

Figura 36: Ruedas giradas rviz.

En la figura 36, la articulación wh3lsupporthingeRF (la de la izquierda) es girada hasta el valor -180º, es decir, ha dado media vuelta. En cambio, el soporte de la rueda derecha ha sido girada a 90º. Las ruedas también giran, pero no se aprecia en la imagen.

Plataforma motor-láser:

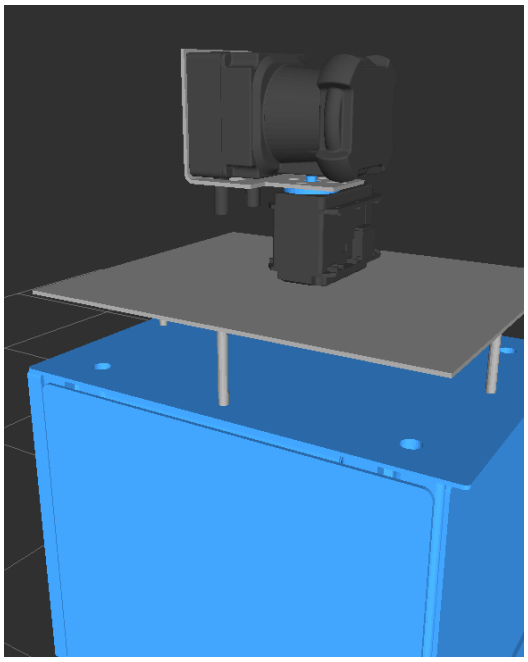

Figura 37: Motor-láser rviz.

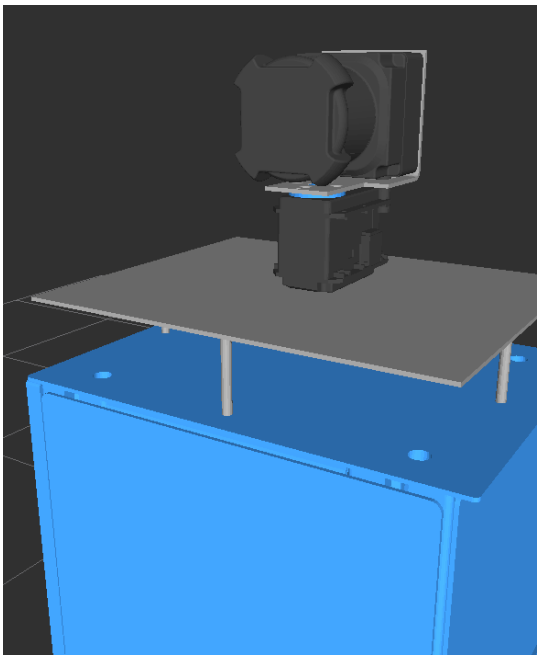

Figura 38: Motor-láser girado rviz.

Aquí lo único que se mueve es el disco del motor, que está coloreado de azul. Está limitado a 270º, que es el necesario para que el láser sea capaz de operar en todo el entorno de la sala.

Brazo de Manfred1:

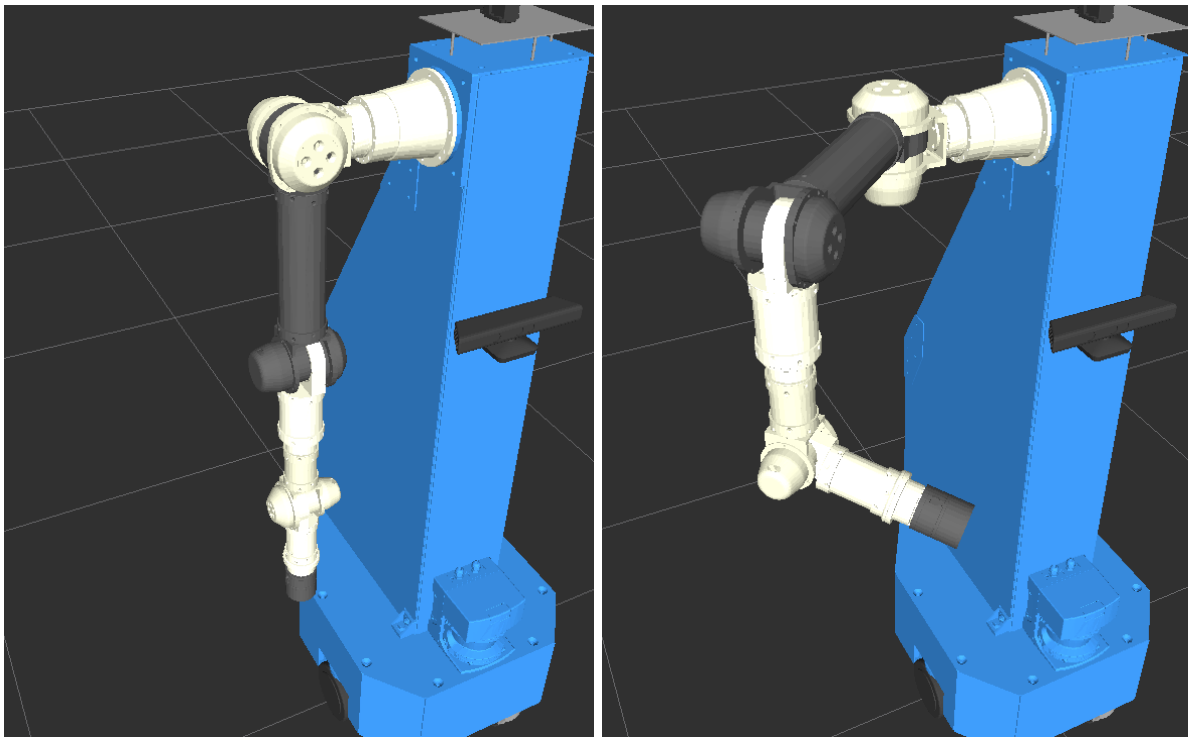

Figura 39: Brazo Manfred1 rviz. Figura 40: Brazo Manfred1 articulado rviz.

Se han girado las siguientes articulaciones:

Shoulder_Yaw (primera articulación) hasta -90º.
Shoulder_Pitch (segunda articulación) hasta -45º.
Shoulder_Roll (tercera articulación) hasta 90º.
Elbow (cuarta articulación) hasta -90º.
Wrist_Yaw (quinta articulación) hasta -75º.
Shoulder_Pitch (sexta articulación) hasta 0º (no se aprecia en la figura).

Mano Gifuhand III:

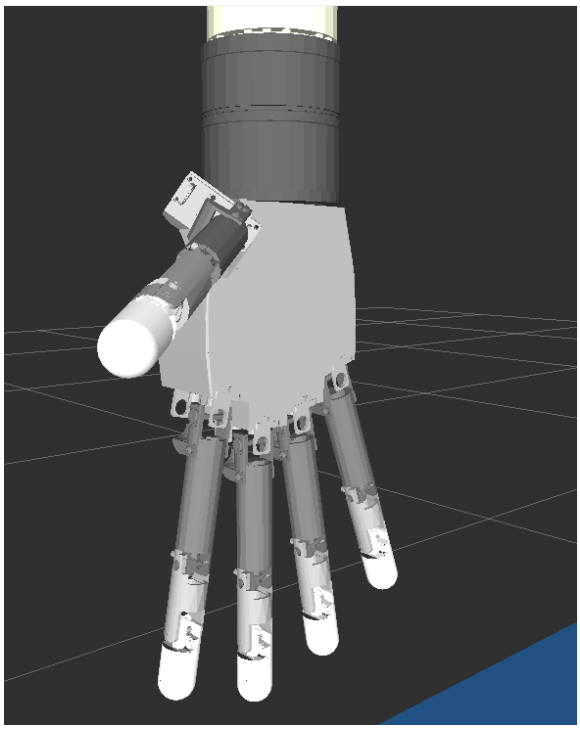

Figura 41: Mano extendida.

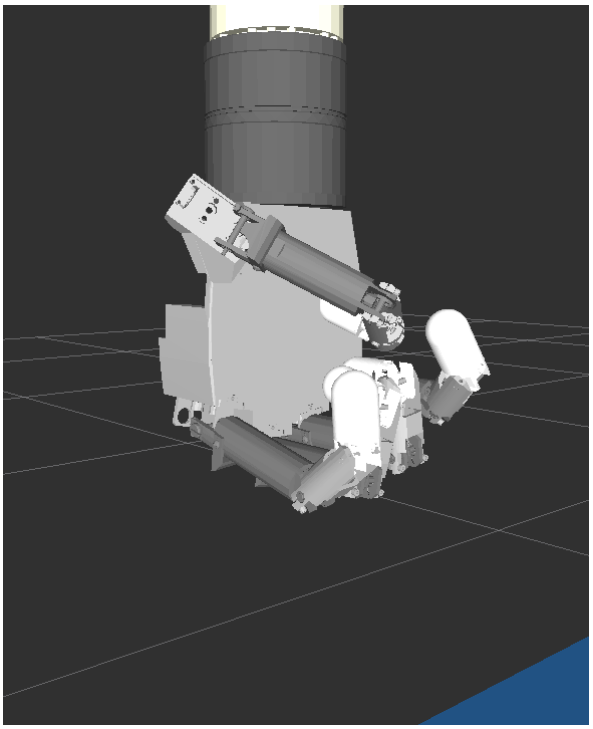

Figura 42: Mano puño cerrado.

La figura 41 representa la mano Gifuhand III en estado de reposo (todas las articulaciones a 0º), es decir, extendida. Para cerrar el puño (figura 42), la segunda articulación de cada dedo (t1, i1, m1, r1 y l1) tienen que estar a casi 90º para colocar perpendicularmente la primera falange respecto a la palma. La tercera articulación aproxima los dedos hacia la palma cuanto mayor sea su valor, hasta 90º. La primera articulación separa y cierra en abanico los dedos (figura 24). La cuarta articulación es imitada por la tercera, excepto en el pulgar, que se puede mover como se desee.

Brazo de Manfred2:

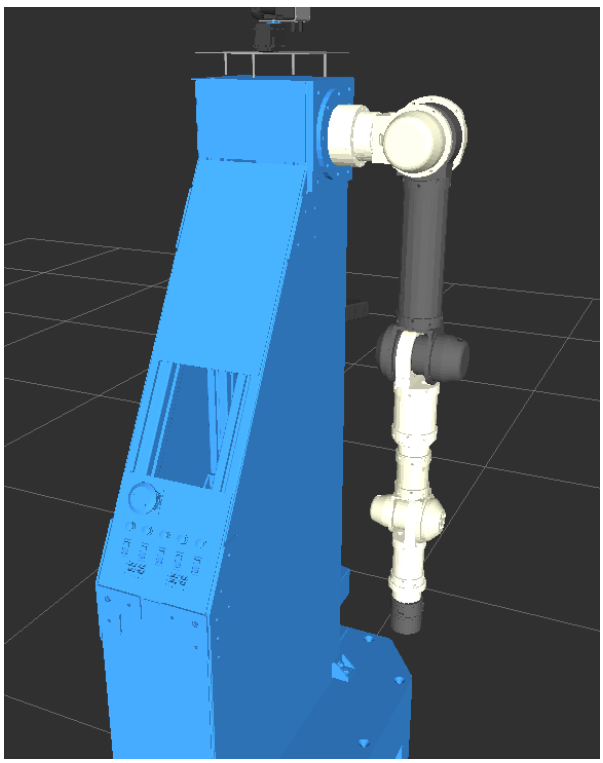

Figura 43: Brazo Manfred2 rviz.

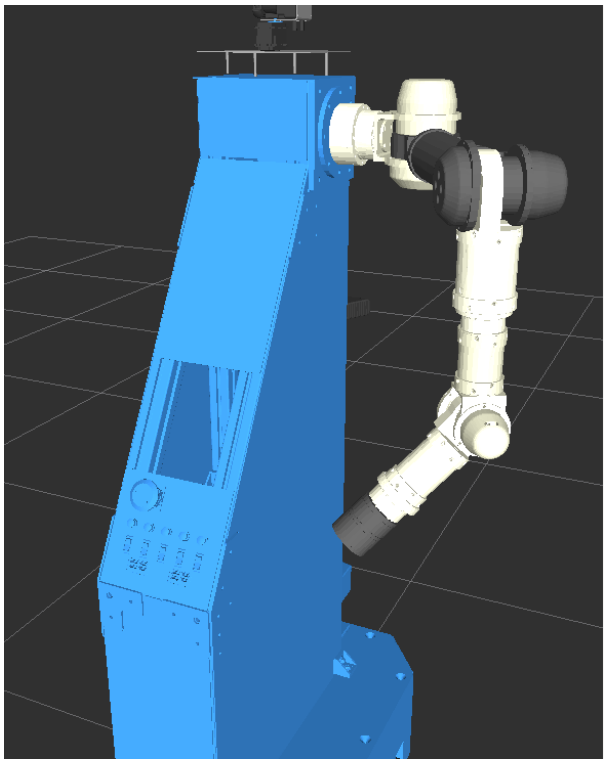

Figura 44: Brazo Manfred2 articulado rviz.

Se han girado las siguientes articulaciones:

Shoulder_Yaw (primera articulación) hasta 90º.
Shoulder_Pitch (segunda articulación) hasta -30º.
Shoulder_Roll (tercera articulación) hasta 90º.
Elbow (cuarta articulación) hasta 90º.
Wrist_Yaw (quinta articulación) hasta 60º.
Shoulder_Pitch (sexta articulación) hasta 0º (no se aprecia en la figura).

Brazo de Manfred3:

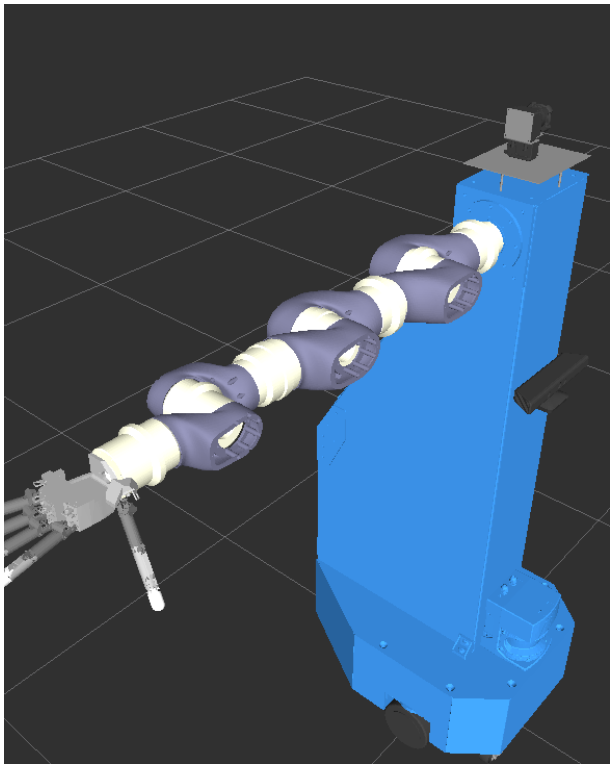

Figura 45: Brazo Manfred3 rviz.

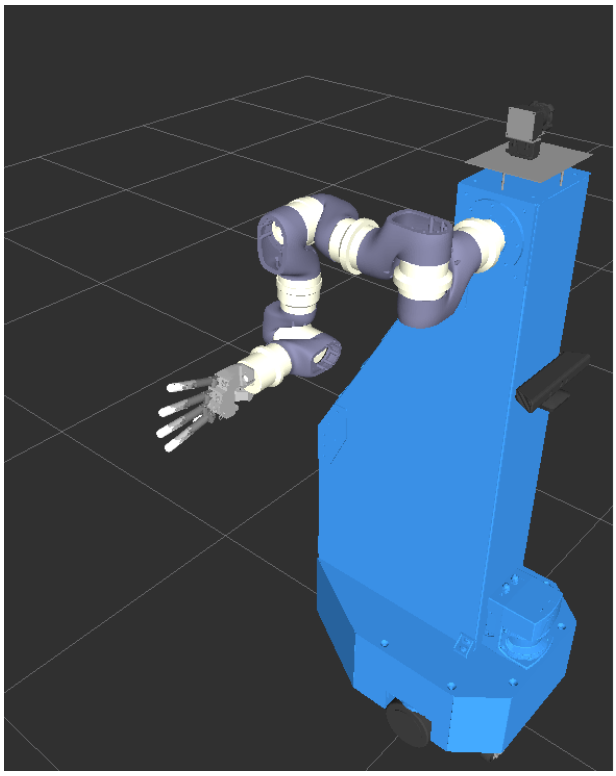

Figura 46: Brazo Manfred3 articulado rviz.

Se han girado las siguientes articulaciones:

Shoulder1 (primera articulación) hasta 90º.
Shoulder2 (segunda articulación) hasta -90º.
Shoulder3 (tercera articulación) hasta -90º.
Elbow1 (cuarta articulación) hasta 90º.
Elbow2 (quinta articulación) hasta -90º.
Wrist1 (sexta articulación) hasta -90º.
Wrist2 (septima articulación) hasta -90º (como está unido a la palma de la mano, esta también se moverá).

6.2 Visualización de Manfred en Moveit!.

A continuación se muestran los tres manipuladores Manfred en Moveit!:

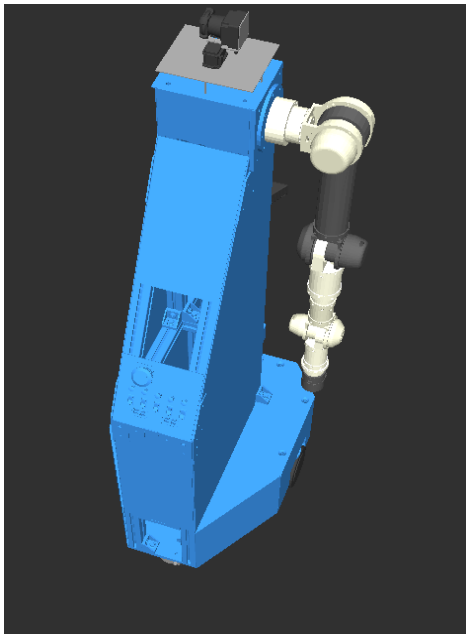

Figura 47: Manfred1 en Moveit!.

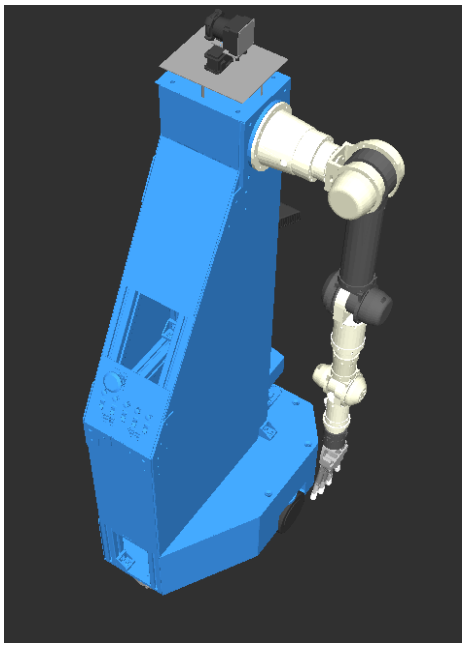

Figura 48: Manfred2 en Moveit!.

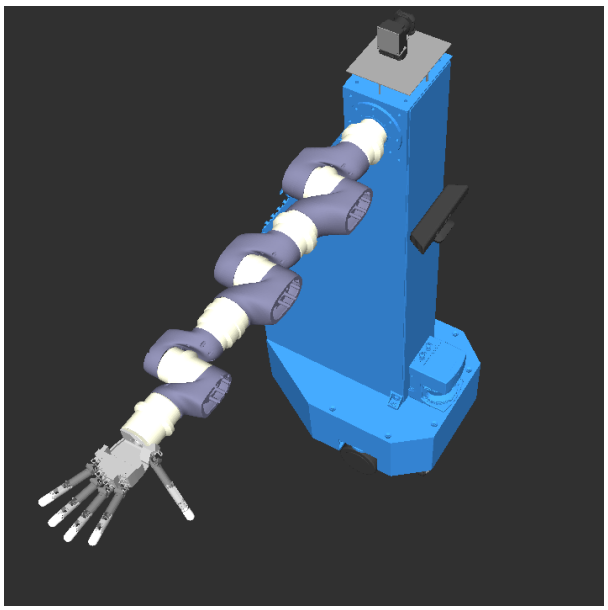

Figura 49: Manfred3 en Moveit!.

7. Conclusiones

Este proyecto ha sido completado de forma satisfactoria, cumpliendo los objetivos planteados y conseguir unos resultados satisfactorios. Todos los problemas y errores que han ido surgiendo durante la realización del proyecto, han sido superados con éxito. Al haber sido editado el manipulador en tres partes distintas (base, brazos y mano) es fácil poder cambiar la configuración o datos de una forma rápida y ordenada. Se ha comprobado que todas las articulaciones simuladas corresponden a los movimientos reales, respetando su velocidad máxima y sus límites de giro. Los tres manipuladores Manfred construidos constan de las siguientes partes:

Manfred1: Base, brazo LWR-UC3M-1 y Gifuhand III.
Manfred2: Base y brazo LWR-UC3M-1 con el primer eslabón cortado.
Manfred3: Base, brazo Manfred3 y Gifuhand III.

Al haber sido configurado con el asistente de moveit!, se podrá emular en el entorno de moveit!. Esto es interesante puesto que incorpora los últimos avances en la planificación de movimiento, manipulación, percepción 3D, cinemática, el control y la navegación.

Como trabajo futuro, sería interesante definir en moveit! unas tareas planificadas y después cargarlas en el robot Manfred para que las ejecute.

8. Referencias

[1] Tesis doctoral de Salah Hassan Kadhim de la universidad Carlos III, "*Metodología para el desarrollo de un manipulador móvil autónomo con características antropomórficas*", 2011.

[2] Ramiro Mena, A. Flores, Dorin Copaci y D. Álvarez, "*Documento técnico del Roboticslab, departamento de Ingeniería de sistemas y automática de la universidad Carlos III*", 2014.

[3] Documentación y tutoriales de ROS, disponible en :
https://www.wiki.ros.org
Última fecha de consulta: 10/2014.

[4] Información completa del lenguaje de programación XML, disponible en:
http://es.wikipedia.org/wiki/Extensible_Markup_Language
Última fecha de consulta: 10/2014.

[5] Descripción y tutoriales de URDF, disponible en:
https://www.wiki.ros.org/urdf
Última fecha de consulta: 10/2014.

[6] Programa Meshlab: documentación y descarga disponible en:
http://meshlab.sourceforge.net/
Última fecha de consulta: 10/2014

[7] Programa meshconv: documentación y descarga disponible en:
http://www.cs.princeton.edu/~min/meshconv/
Última fecha de consulta: 10/2014

[8] Aplicación web para conocer los valores rgba de los colores, disponible en:
http://www.css3maker.com/css-3-rgba.html
Última fecha de consulta: 10/2014

[9] Documentación y tutoriales de Moveit!, 2014. Disponible en:
http://moveit.ros.org/
Última fecha de consulta: 10/2014

[10] Tetsuya Nouri, Haruhisa Kawasaki, Keisuke Yoshikawa, Jun Takai, and Satoshi Ito, "*Anthropomorphic Robot Hand: Gifu Hand III*", 2002.