Universidad Carlos III de Madrid

Benchmarking en *MoveIt!*

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

Febrero 2014

Miguel Mora Cuesta

Tutor: Javier Victorio Gómez González

Contenido

Índice de figuras

1. Introducción

La planificación de trayectorias o *motion planning* consiste en encontrar un camino entre un punto de inicio y otro de final, en un sistema específico sujeto a ciertas restricciones. Existen muchas aplicaciones para las que se planifican trayectorias; una de ellas es la robótica.

Incluso dentro de la robótica aparecen diferentes tipos de problemas debido a la variedad de formas y funcionalidad de robots, como por ejemplo, la diferencia entre un robot bípedo frente a uno con ruedas, o entre uno con movimientos rápidos y uno más lento.

La planificación de trayectorias en robótica necesita trabajar con algoritmos que obtengan el movimiento del robot ajustado a unos criterios deseados, entre dos puntos, y que ajuste correctamente cada una de sus articulaciones para dar lugar a ese movimiento. Dadas todas las posibilidades de un robot y la variedad de algoritmos que existen es imprescindible una herramienta para procesar y comparar los resultados de todas las opciones, y así poder elegir la más apropiada según la aplicación: éste es el concepto de *benchmarking*.

La aplicación de *benchmarking* no es exclusiva en la robótica. El término se define como un proceso para comparar productos o servicios con el objetivo de seleccionar los mejores y más eficientes. En este proyecto *benchmarking* se refiere a la comparación de algoritmos de planificación de trayectorias para escoger, según las características del robot o el escenario, cuál es el más eficiente.

2. Motivación

En este proyecto convergen dos ideas para hacerlo posible: el desarrollo de la robótica y el uso de software libre.

La ***robótica*** empezó como una forma de sustituir puestos de trabajo peligrosos o difíciles para las personas por máquinas. En la actualidad se utilizan robots para una amplia variedad de aplicaciones, y en el futuro el crecimiento promete ser mayor: funciones como por ejemplo de ayuda en tareas domésticas, a personas con discapacidad, en grandes construcciones o en misiones espaciales.

El ***software libre*** es el que ofrece a los usuarios la posibilidad de compartirlo, estudiarlo y modificarlo. Se entiende como una elección ética y política a favor del derecho al conocimiento y a su difusión desde la informática. Este proyecto, que utiliza software libre en su totalidad, pretende servir de guía abierta y accesible para cualquier usuario de ROS del trabajo contenido en él.

ROS, el sistema operativo de robótica que es la base del proyecto, se crea para unificar el trabajo de distintos grupos de desarrolladores en robótica de forma colaborativa. Y en este proyecto se pretende estudiar y documentar la parte de ROS, específicamente *MoveIt!,* dedicada a la planificación de trayectorias y *benchmarking*.

3. Objetivos

La idea principal para este proyecto es trabajar sobre *MoveIt!* y sus funcionalidades en *benchmarking* para planificación de trayectorias en robots. Los objetivos se pueden agrupar como:

- **Comprobar las capacidades de *MoveIt!*** Esto implica un conocimiento a fondo de las posibilidades que ofrece *MoveIt!* siguiendo los tutoriales o guías de sus aplicaciones.
- **Explorar la herramienta de *benchmarking*.** Trabajar con diferentes robots y escenarios para asegurar si es posible usar *benchmarks* en planificación de trayectorias.
- **Analizar los resultados comparativos.** Extraer, comprobar y entender los resultados cuantitativos de *benchmarking*.
- **Adaptación a robots propios.** Tiene que ser universal, es decir, tiene que poder realizarse benchmarking en cualquier robot con una configuración compatible con ROS.
- **Documentar los problemas que surjan en el proceso.** Debido a que aún está en desarrollo, es muy útil identificar los errores, *bugs* u otros problemas de funcionamiento de las aplicaciones o de *MoveIt!*
- **Elaborar un manual de ayuda.** Redactar una guía o manual que asegure los progresos alcanzados en este proyecto para actualizaciones en el futuro.
- **Contribuir al desarrollo de *MoveIt!*** Ayudar a la comunidad *open source* de *MoveIt!* con la documentación y con las debilidades de *benchmarking* encontradas en el proyecto.

4. Tecnologías utilizadas

4.1. ROS (*Robot Operating System*)

ROS en un meta-sistema operativo para robótica en código abierto que proporciona librerías, herramientas y convenciones, que trata de simplificar la tarea de desarrollo en plataformas de robótica. ROS está en constante crecimiento con nuevas versiones y mejoras frecuentes.

En su página web se descarga el software necesario y es donde está además toda la documentación y tutoriales. La versión actual sobre la que se ha trabajado en el proyecto ha sido *Hydro*, la última hasta entonces.

El elemento más básico con el que funciona ROS es con paquetes. Cualquier archivo o aplicación está contenido en algún paquete de ROS. De modo que para trabajar con *MoveIt!* se tiene que hacer a través de estos paquetes, pues a su vez está basado en ROS.

Figura 1. Ecuación de ROS = paquetes + herramientas + capacidades + colaboración. Fuente: web de ROS

En la figura 1 se representa de forma conceptual la "ecuación ROS", que no sólo incluye el nivel de paquetes, herramientas y capacidades sino también un ecosistema de colaboración para favorecer su desarrollo y estabilidad, que es uno de los objetivos del proyecto.

4.2. MoveIt!

4.2.1. INTRODUCCIÓN A MOVEIT!

MoveIt! es una aplicación reciente creada para la planificación de trayectorias en ROS. Es una herramienta muy útil para este campo porque permite interactuar con las articulaciones y los elementos del robot a partir de una interfaz gráfica sencilla.

MoveIt! mejora la forma de interactuar con el robot para aplicaciones de planificación de trayectorias. En versiones más antiguas de ROS el equivalente era el simulador *Gazebo*, pero no permitía la acción directa del usuario sobre las articulaciones. El plugin de ROS llamado RViz sobre el que trabaja *MoveIt!* ayuda a comprobar visualmente conceptos como el espacio de movimiento o las posiciones prohibidas y colisiones.

El estado de desarrollo de *MoveIt!* es prematuro en algunos casos (tabla 1), por lo que muchas de las aplicaciones y herramientas no trabajaron como se esperaba a lo largo del proyecto. Como es *open source* se contribuirá a su mejora con este trabajo, aportando documentación y errores encontrados.

Tabla 1. Estado de desarrollo de las aplicaciones de MoveIt!

Sección de *MoveIt!*	Estado
Motion Planning	Alpha
RViz Plugin	Beta
Benchmarking	Alpha
Setup Assistant	Beta

Las funcionalidades de *MoveIt!* se dividen en: bajo nivel, núcleo y alto nivel (figura 2).

En bajo nivel está la base física que controla las cinemáticas que soportan la computación de nivel superior.

Entre las competencias de *MoveIt!* de núcleo está la que orienta todo el proyecto: *benchmarking*. Pero ésta a su vez es dependiente de otras funciones

como *motion planning* o *warehouse* para funcionar, como se verá más adelante. En el núcleo se encuentran las aplicaciones y herramientas principales de *MoveIt!*

Por último, las competencias en alto nivel interactúan con el usuario y extraen la información sobre planificación necesaria para emplear los recursos del núcleo.

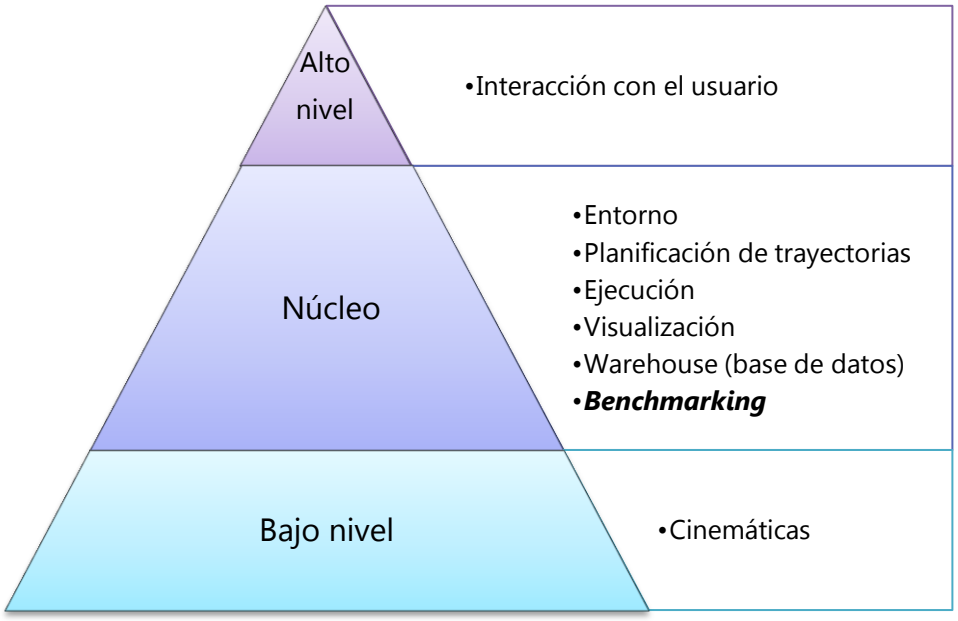

Figura 2. Pirámide de Competencias de MoveIt!

4.2.2. ELEMENTOS DE MOVEIT!

Para poder comparar el comportamiento de diferentes algoritmos sobre el robot se necesitan todos y cada uno de estos elementos:

- *ROBOT*

La planificación de trayectorias puede ser del robot desplazándose por el suelo o por el espacio, el movimiento de un brazo o extensión de su cuerpo, o una combinación de ambas.

Figura 3. Robot PR2 sobre el suelo. Fuente: web de Willow Garage

Para este proyecto los robots se mueven sólo en el plano del suelo junto con sus brazos (figura 3). Al final del brazo hay una pinza acoplada en su muñeca que es la que se desplaza de un punto a otro siguiendo la trayectoria planificada. Este movimiento es posible por la acción de cada articulación del brazo. Y es en ese movimiento descompuesto en las articulaciones donde el algoritmo de planificación de trayectorias interviene, ajustando qué trabajo realiza cada articulación.

→ *SRDF (Semantic Robot Description Format) y URDF (Universal Robot Description Format).*

En MoveIt! los robots se definen en un archivo de tipo *srdf* que establece la forma y texturas del robot, la movilidad de cada elemento que lo compone e información adicional sobre colisiones. A veces, el robot puede estar dividido en varios archivos, en cada uno de los cuales se define una parte (por ejemplo: un archivo de tipo *srdf* para la base, otro para la cabeza y otro para el brazo). Se utiliza un asistente para la creación del archivo descriptivo que va incluido en el paquete del robot, como se detallará más adelante.

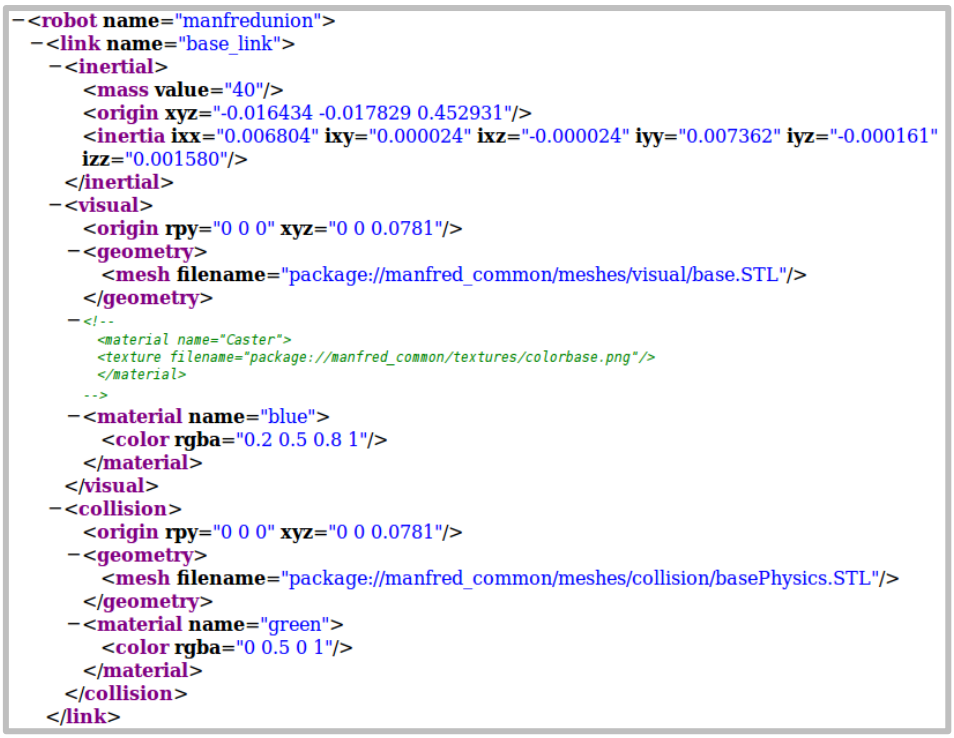

Figura 4. Ejemplo de un archivo descriptivo de tipo URDF

El tipo de archivos *srdf* complementa a los *urdf*, que es el formato original para robots en ROS. Un archivo descriptivo de formato *urdf* sólo contiene la información de las articulaciones físicas del robot. Como se ve en la figura 4, contiene descripciones del origen de la geometría, masa, inercia o color del conjunto físico del robot, así como llamadas a formas geométricas en paquetes externos. El tipo *srdf* puede incluir posiciones, datos de la matriz de colisión y grupos de planificación del robot, representado en la figura 5.

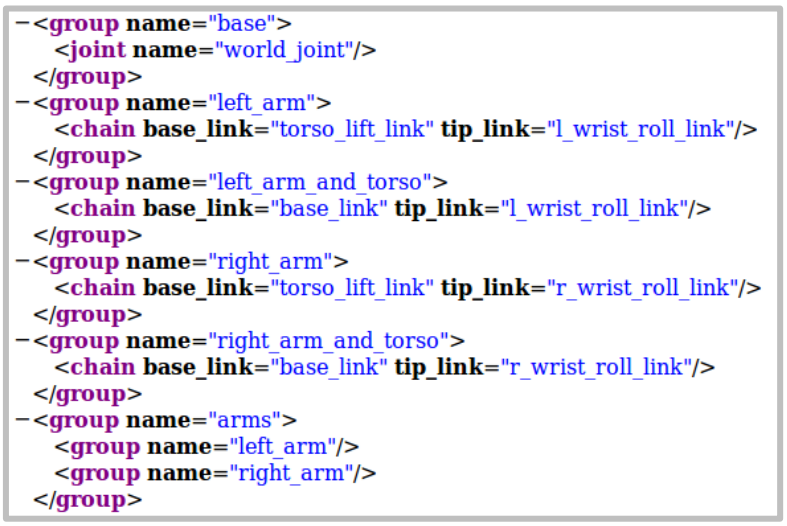

Figura 5. Ejemplo de archivo descriptivo de tipo SRDF

- *ELEMENTO MÓVIL DEL ROBOT*

Si se trabaja con un robot estático, el elemento para planificar será un brazo (figura 6) u otra pieza móvil (pinzas, cabeza, cámara…). Si es un robot móvil también se incluye el su cuerpo entero. Cualquier parte del robot está incluida en la descripción del mismo, como se detalla en el apartado anterior. La planificación trabaja con los elementos móviles que el usuario haya configurado, pues son los que trazarán la trayectoria obtenida.

Figura 6. Brazo de PR2. Fuente: web de Willow Garage

- *SCENE: ENTORNO*

Es el espacio virtual sobre el que el robot se moverá. En el escenario puede haber obstáculos que los planificadores de trayectorias deberán evitar.

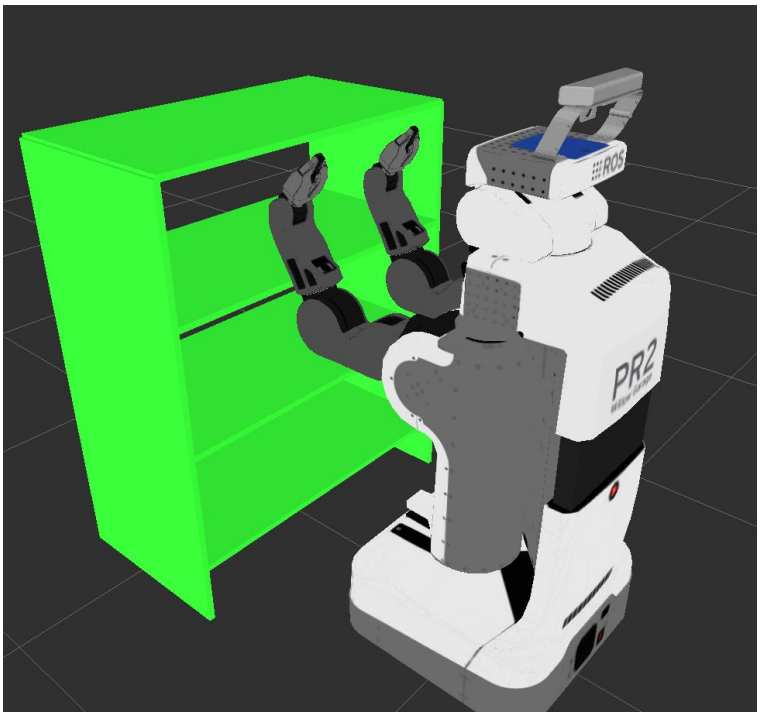

Figura 7. Robot PR2 en un escenario con una estantería

Los entornos o escenarios se definen como un archivo de tipo *scene*, que se almacena en la base de datos *warehouse*. Los escenarios se pueden descargar o crear a partir de un archivo de texto, y es en ese mismo formato en el que se introducen en la base de datos. Una vez ejecutado en el plugin RViz se puede ver gráficamente y planificar una trayectoria para el robot moviendo los objetos que contiene (figura 7).

El escenario puede estar vacío y no contener obstáculos, pero es necesario que exista para la planificación de trayectorias. Esto ocurre porque *MoveIt!* asocia los *queries* (que se verán en el siguiente punto) a un escenario.

- *QUERIES: PUNTOS DE INICIO Y FIN*

La trayectoria debe estar contenida por dos puntos: el de comienzo y el de fin. Esto puede significar el punto de reposo del robot y el punto que se desea alcanzar para la tarea que se le quiere asignar (coger un objeto o herramienta, trabajar sobre una zona,…). Cada algoritmo de planificación de trayectorias trazará un recorrido diferente entre esos dos puntos, y por tanto es necesario sean declarados.

Se define con un archivo de tipo *queries*. No es sólo la información sobre la ubicación de dos puntos en el espacio, sino también la orientación de la parte móvil a planificar. *MoveIt!*, a través de RViz, guarda estos puntos en la base de datos asociándolos al escenario (figura 20). Así la planificación de trayectorias accede al escenario y de él extrae toda la información necesaria de los puntos de inicio y fin especificados por el usuario.

- *WAREHOUSE*

La base de datos en *MoveIt!* se denomina *warehouse* (almacén). Actúa en forma de servidor con un puerto asociado, de modo que se ejecuta y ocupa un puerto en un terminal (figura 8). Para añadir escenarios y *queries* se importa por texto al puerto en el que está ejecutándose.

Figura 8. Captura de la ejecución del servidor de Warehouse

Tanto para el proceso de *benchmarking* como para poder cargar la información de escenarios y *queries* en RViz se necesita la base de datos activa. Ambas aplicaciones se conectan con el servidor *warehouse* para acceder a la información requerida.

- *VISUALIZACIÓN: RVIZ*

Como ya se ha mencionado, RViz es un plugin de ROS que *MoveIt!* utiliza para la visualización, es decir, una interfaz gráfica. A través de RViz el usuario puede posicionar el robot, mover objetos o ajustar las pinzas en el espacio interactivo.

No sólo sirve como interfaz gráfica del robot, escenario, *queries* y trayectorias, sino que también facilita la incorporación y unificación de estos elementos para el posterior proceso de *benchmarking*.

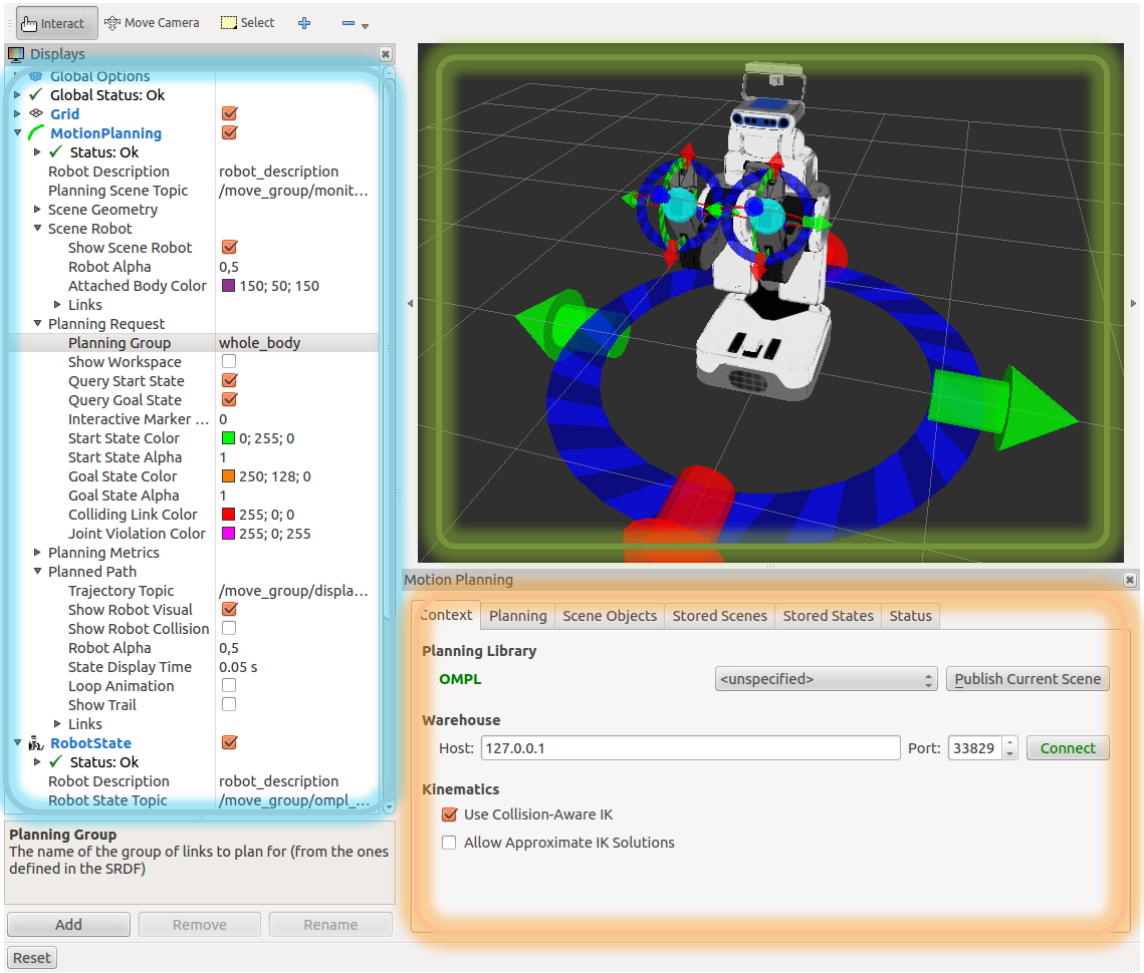

Figura 9. Ventana de RViz

Si se observa la figura 9, a la izquierda en la selección azul está el menú *Displays* para seleccionar qué ver o qué ocultar en la ventana de visualización, recuadrada en color verde. Debajo, de color naranja, la pestaña desde la que se ejecuta la librería OMPL. También sirve para conectarse a la base de datos y poder trabajar con los escenarios o *queries* guardados, o para establecer unos nuevos que poder guardar.

- *PLANIFICACIÓN DE TRAYECTORIAS: LIBRERÍA OMPL*

OMPL (*Open Motion Planning Library*) es una librería que contiene algoritmos para la planificación de trayectorias. Está incluida en ROS (en fase beta de desarrollo) para aplicaciones de robótica, que es el uso que recibirá esta librería en el proyecto. La figura 10 es un ejemplo ilustrativo de una trayectoria entre dos puntos de un entorno montañoso calculada con OMPL.

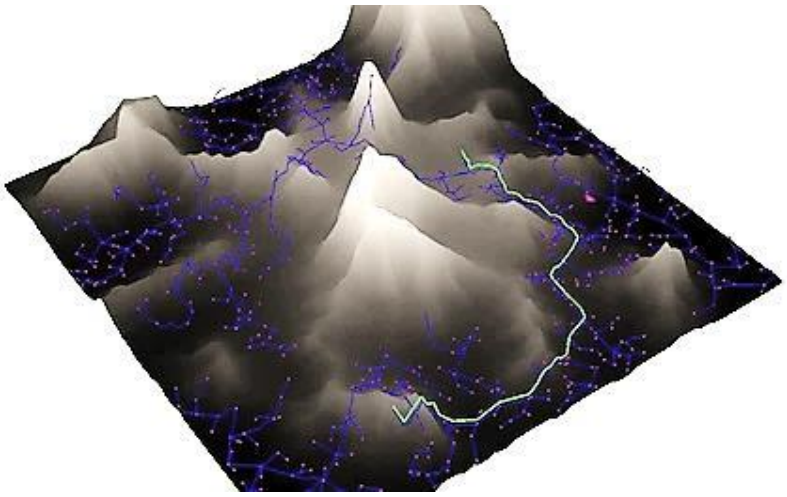

Figura 10. Ejemplo de la planificación de trayectorias con OMPL. Fuente: web Kavraki Lab, Rice University

4.2.3. BENCHMARKING: EVALUACIÓN POR COMPARACIÓN DE ALGORITMOS DE PLANIFICACIÓN

Mediante varios algoritmos preinstalados (en este proyecto de la librería OMPL) *MoveIt!* puede planificar trayectorias. La aplicación de *benchmarking* compara los resultados de la planificación según varios algoritmos. Así es más fácil ver qué algoritmo es el adecuado según la aplicación.

Para planificación de trayectorias en *MoveIt!* se parte de los archivos del escenario y *queries* y con RViz y *Warehouse* se importan al paquete del robot. La ejecución detallada en la figura 11 sigue las condiciones de un archivo de configuración escrito por el usuario y da lugar a un archivo de tipo *log* para crear los resultados finales gráficamente en PDF.

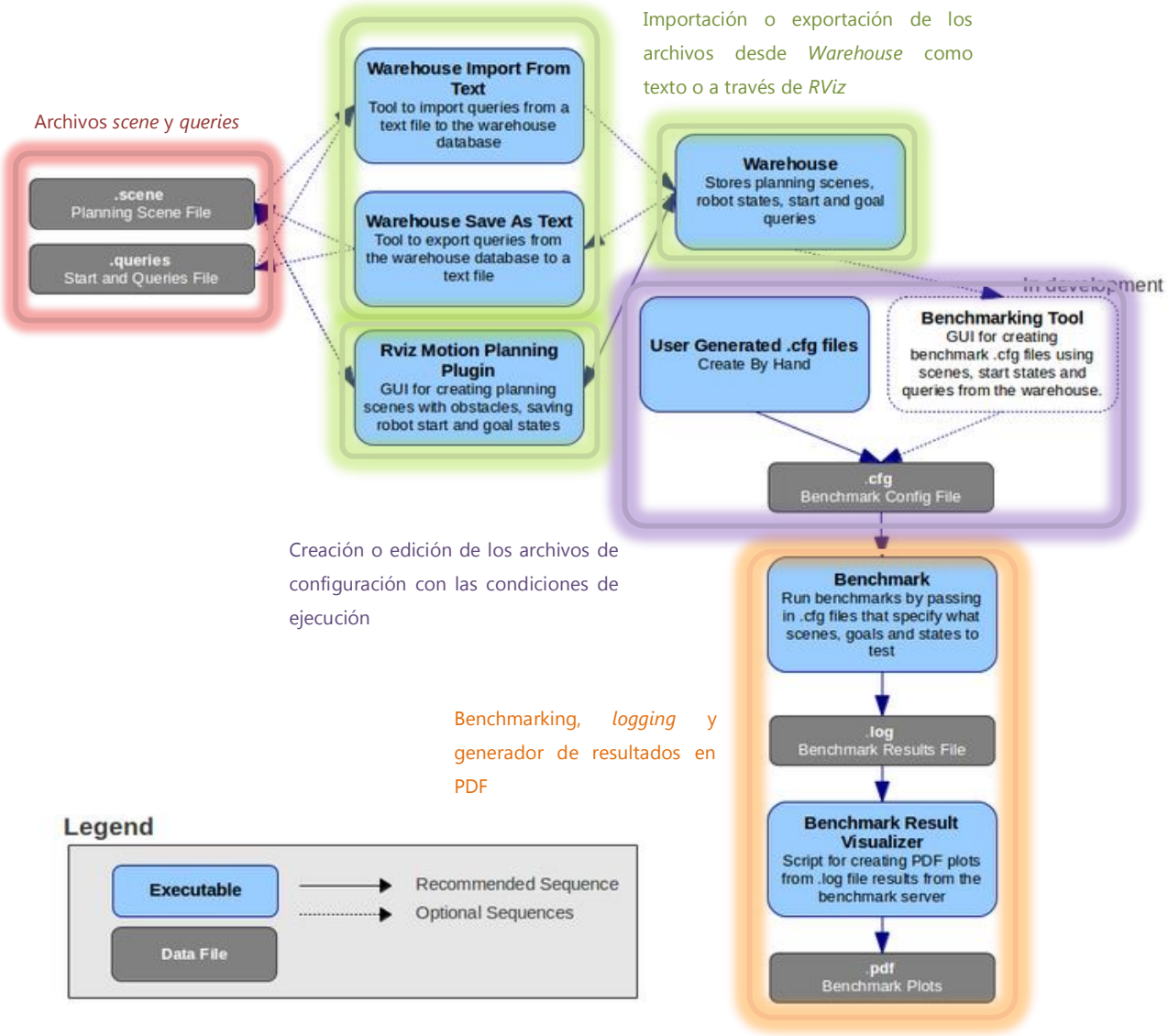

Figura 11. Diagrama del proceso completo de benchmarking

Todo este proceso es automático y el usuario sólo interviene en la introducción del escenario y estados del robot, y en las condiciones de la planificación que se detallan en los archivos de configuración.

4.3. Robot PR2

PR2 (figura 12) es un diseño creado por *Willow Garage* pensado para desarrolladores de aplicaciones robóticas y tecnológicas. Está construido para ser operado con ROS. Tiene sensores distribuidos por todo la estructura (láser y cámaras) y software de visión artificial en código abierto para desarrolladores.

Además, ROS y *MoveIt!* permiten el uso de sus funciones en simuladores virtuales.

Figura 12. Robot PR2 (vista completa, cabeza y pinza). Fuente: web de Willow Garage.

4.4. Robot Manfred

Manfred (figura 13) es un diseño de robot del Laboratorio de Robótica de la Universidad Carlos III de Madrid. Está pensado como robot "mayordomo", es decir, que realice tareas o desplazamientos en entornos domésticos. Al igual que PR2, este robot también es móvil gracias al desplazamiento de la base. Está formado por la base móvil, el torso y un brazo articulado.

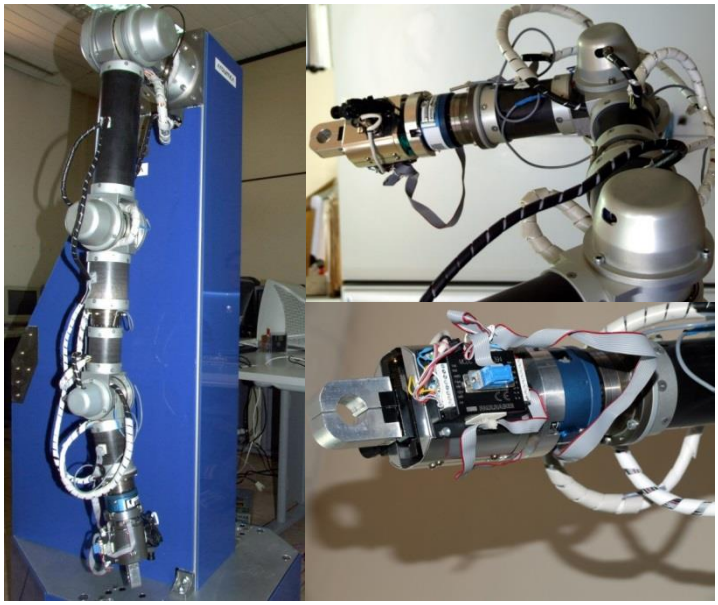

Figura 13. Robot Manfred (vista completa y detalle del brazo y muñeca). Fuente: Roboticlab UC3M

5. Experimentación con *MoveIt!* y Benchmarking

5.1. Conceptos básicos

Una vez definido el objetivo, era necesario conocer bien la herramienta. La web de los desarrolladores de ROS y *MoveIt!* ofrece tutoriales para conocer y practicar con las funcionalidades básicas. Además, como *MoveIt!* está basado en ROS, hay que conocer previamente muchos de los comandos y métodos para poder usarlo.

5.1.1. COMANDOS DE ROS

ROS y *MoveIt!* se instalan en Ubuntu y desde el terminal se dan las órdenes. La organización por paquetes de ROS se controla desde el explorador de carpetas de Ubuntu pero se ejecuta desde el terminal. Las instrucciones llaman a archivos ejecutables contenidos en los paquetes.

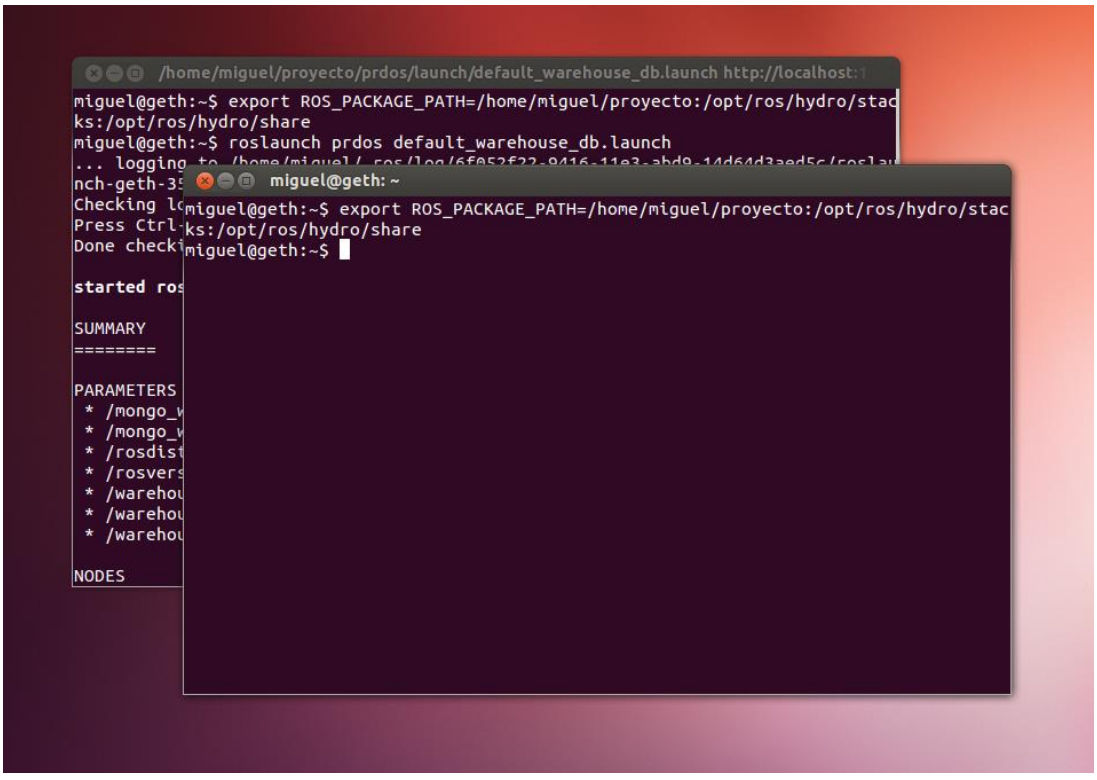

Figura 14. Escritorio con dos ventanas del terminal de Ubuntu

Como algunas de las herramientas de *MoveIt!* y ROS funcionan como servidores (por ejemplo, la base de datos *warehouse*), el trabajo con varias terminales como en el escritorio de la figura 14 es habitual: el terminal en segundo plano tiene en ejecución un nodo de ROS que se mantiene activo para realizar las tareas necesarias en primer plano.

5.1.2. CONFIGURACIÓN DEL ROBOT

MoveIt! ofrece al usuario un asistente para la creación de paquetes de ROS con un robot: *MoveIt Setup Assistant* (figura 15). Es muy útil porque crea automáticamente los archivos necesarios para el paquete, que de otro modo habría que hacerlo manualmente. Se ha utilizado para crear los paquetes de los robots con los que se trabaja en este proyecto: PR2 y Manfred.

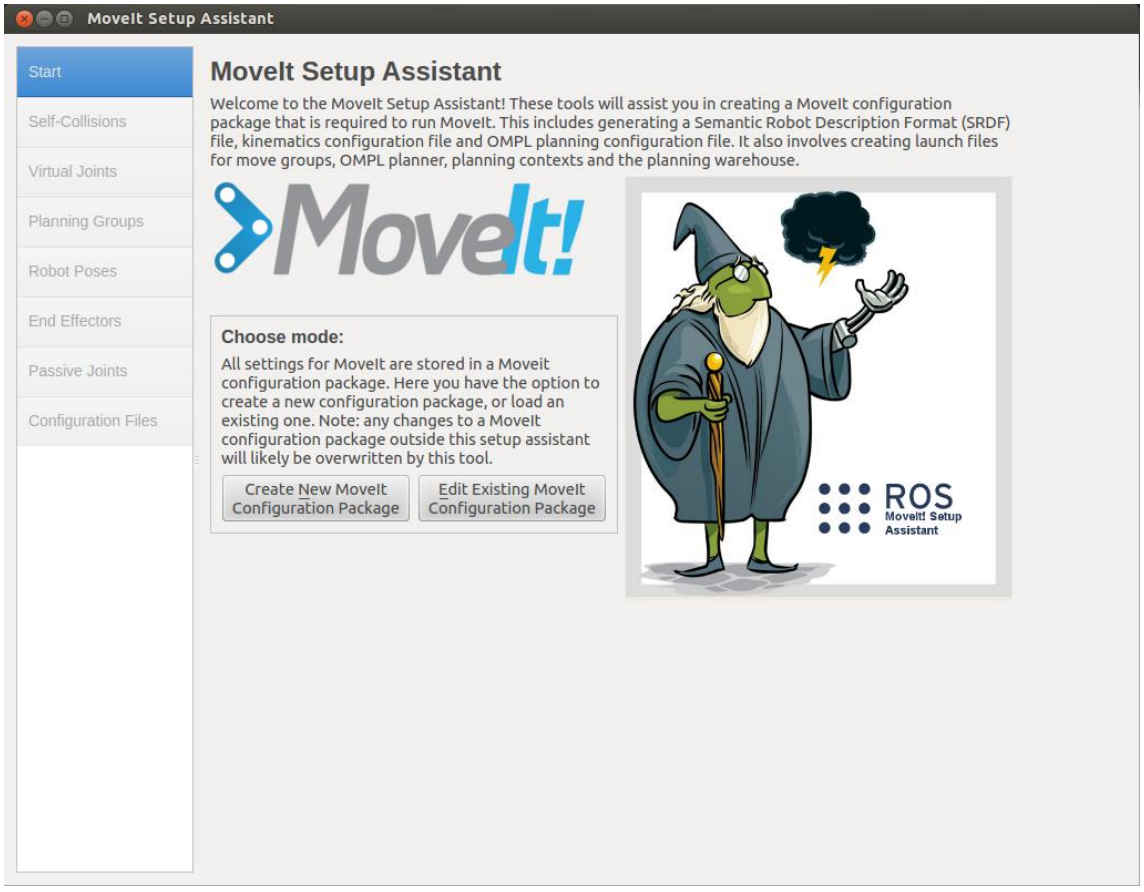

Figura 15. Ventana del asistente de configuración de los paquetes

Para poder operar en *MoveIt!* y poder planificar trayectorias, entre otras cosas, hay que transformar el archivo descriptivo base del robot (*urdf*) en un paquete de ROS. El asistente lo configura paso a paso, siguiendo estas secciones:

- **Matriz de auto-colisión.** Sirve para delimitar un margen de seguridad sobre el propio robot y las colisiones que puede tener con un miembro suyo. Es decir, crea un espacio virtual que genera colisión si cualquier parte de robot entra en él, con la tolerancia ajustada por el usuario.

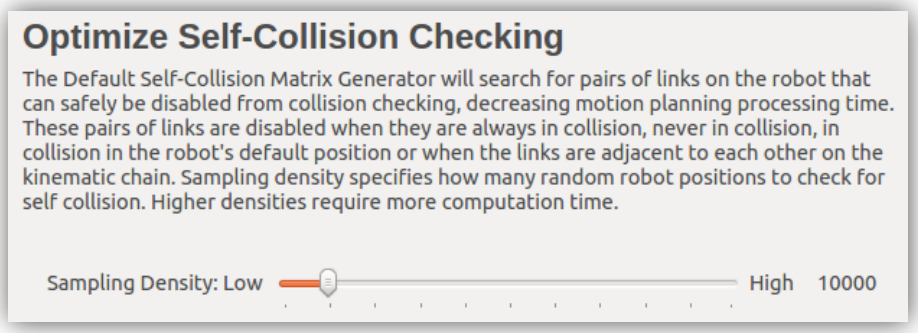

Figura 16. Ajuste de la densidad de la matriz de auto-colisión.

En la figura 17 se muestra una comparación de dos capturas en RViz del robot PR2 con la matriz desactivada y con la matriz activada, y otra del volumen equivalente a la matriz sin la textura del robot. El campo virtual o densidad de muestreo de la malla de colisión en este caso es bajo, el seleccionado en la figura 16, y la superficie en las articulaciones coincide prácticamente.

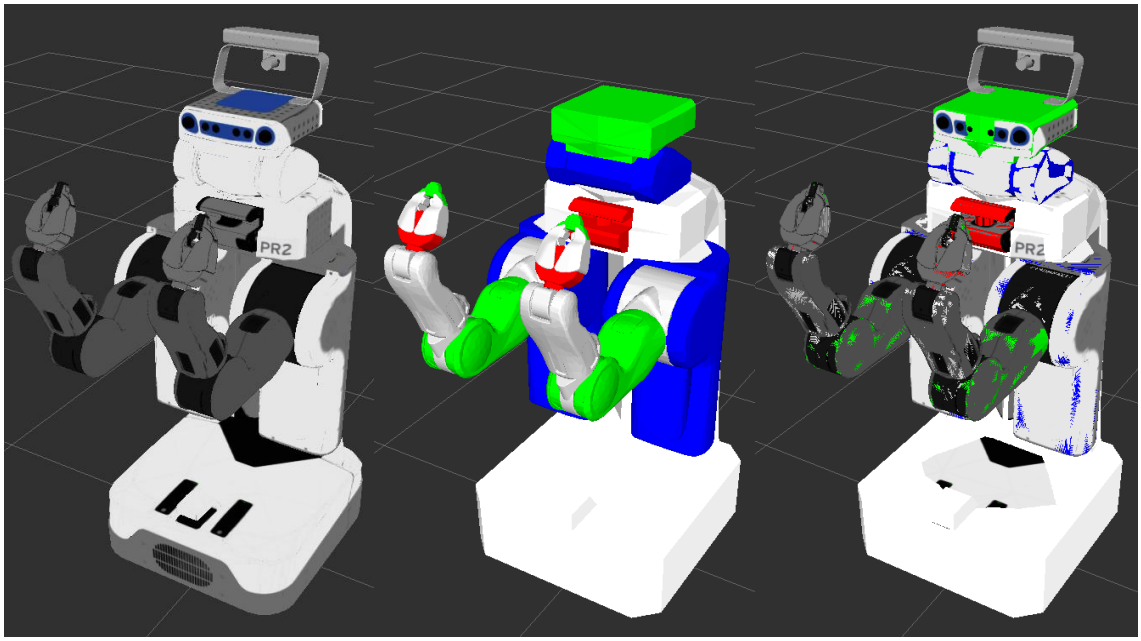

Figura 17. Robot PR2 y su matriz o malla de colisión de baja densidad.

- **Articulaciones virtuales.** Así se definen las articulaciones que no se representan gráficamente (no forman parte del cuerpo físico del robot) y cuya función es necesaria. Por ejemplo, la articulación virtual con la que trabajan los robots en este proyecto es la del robot-suelo, representada en la figura 18. Entre el robot y el suelo hay un movimiento y grados de libertad por lo que es, teóricamente, una articulación.

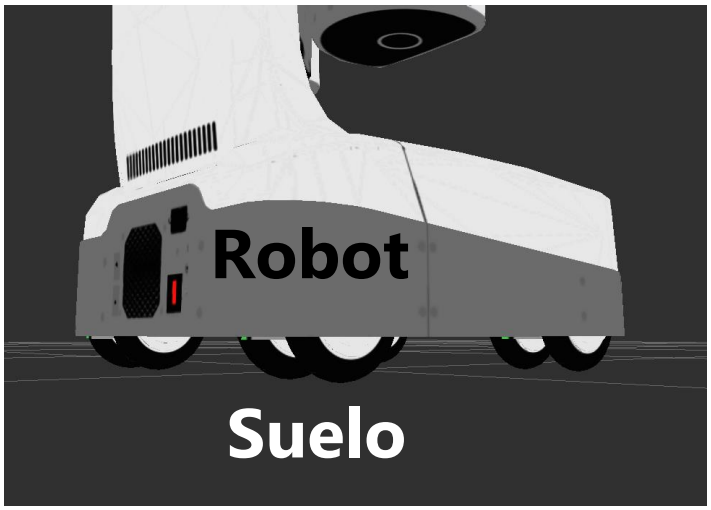

Figura 18. Articulación virtual robot-suelo del robot PR2

- **Grupos de planificación.** Es la parte más importante. Cada grupo es un conjunto de enlaces y articulaciones con las que se debe trabajar más adelante en RViz o *benchmarking*. Aceptan múltiples configuraciones y puntos de referencia, según la función del robot.

En la figura 19 es fácil comprobar en la ventana derecha el grupo seleccionado en color rojo. El asistente permite ver al usuario qué piezas del robot forman parte del grupo de planificación en el instante de la selección. Poder ver de forma directa qué articulación o parte del robot es la que está seleccionado el usuario es una ventaja de interfaces gráficas como el Asistente de Configuración o RViz.

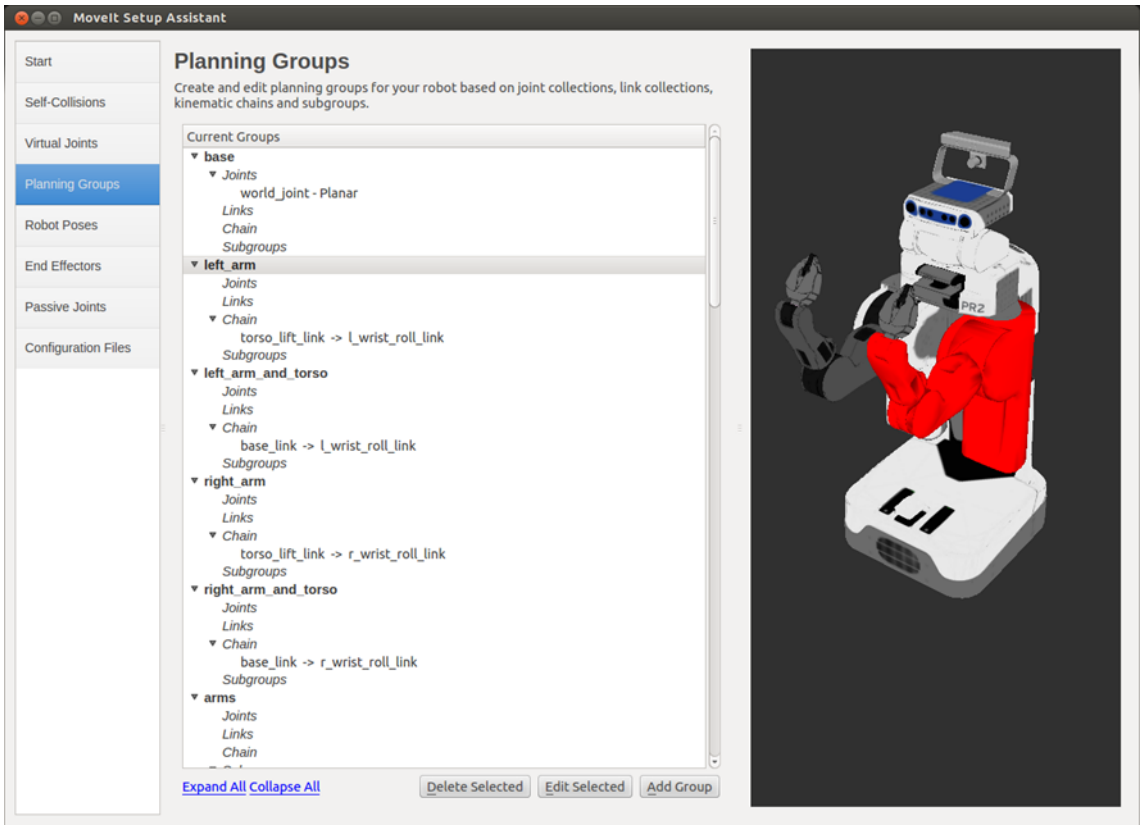

Figura 19. Ventana de configuración de los grupos de planificación del robot PR2

Conviene diferenciar los cuatro tipos en los que se dividen los grupos de planificación. Puede ser un grupo de uniones (*links*), articulaciones (*joints*), cadena cinemática (*kinematic chain*) o subgrupos. Así se distinguen en el archivo descriptivo del robot (*srdf*), que contiene la configuración establecida de los grupos.

- **Poses del robot.** En este apartado el usuario establece posiciones para el robot. Aunque no es obligatorio que un robot tenga poses definidas, sí es imprescindible para *benchmarking* pues es un proceso que requiere estados. Por tanto es en este paso donde se configuran los puntos de reposo, inicio o final, de cada grupo de planificación.

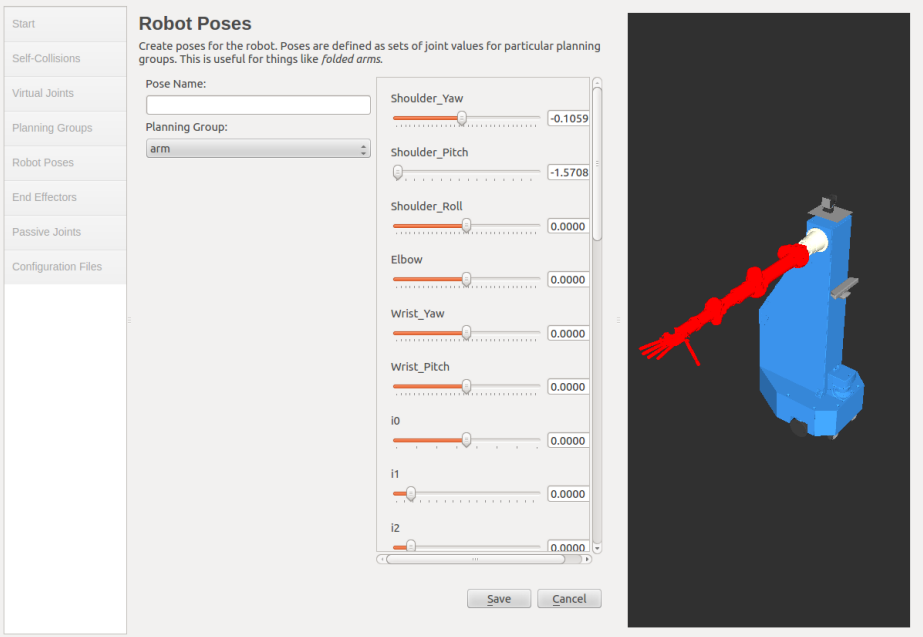

Figura 20. Ventana de configuración de la pose del robot

Las posiciones del robot que después se guardarán en *warehouse* como *queries* se pueden definir en el asistente de configuración. Desde RViz se pueden asignar también posiciones del robot (figura 20), pero hay diferencias (tabla 2). Al final, tanto las definidas gráficamente como ajustando el valor de las barras en la ventana de *Robot Poses* del asistente de configuración, se guardan como *queries* desde RViz en *warehouse* del mismo modo.

Tabla 2. Comparativa de tipos de configuración de posiciones

Definición de *robot poses*	**Manualmente en el espacio de RViz**	**Ajustando los valores en *Setup Assistant***
Precisión	Baja: el usuario no conoce las coordenadas exactas de los puntos del espacio	Muy alta: el usuario ajusta las articulaciones con precisión
Interacción con el entorno	Normal: el usuario ubica el robot o su elemento a planificar en el mismo espacio en el que están representados los obstáculos	Nula: en la ventana de *Setup Assistant* no se cargan los escenarios
Articulaciones	Automático: en RViz se calcula una posición de las articulaciones al mismo tiempo que se ubica el marcador	Manual: es el usuario el que define exactamente cuánto se mueve cada articulación

Con estas diferencias el usuario puede decidir qué se adapta mejor a su aplicación. En este proyecto se han usado ambas y se ha justificado respectivamente en cada experimento.

- ***End effectors*.** Una vez están definidos los grupos de planificación, hay que hacer lo mismo con los *end effectors*. Los *end effectors* son las partes del robot que, como dice su nombre, están al final del grupo de planificación y las que efectúan la trayectoria. Son partes importantes acopladas al extremo de un brazo, como una pinza, que en la aplicación real sería el elemento clave que justifica el desplazamiento.

Se declara como *end effector* un grupo de planificación creado en los pasos anteriores, y vinculado a un elemento que lo sostiene. En el caso de PR2, es el grupo que contiene los enlaces en las pinzas y que está vinculado a la muñeca.

- **Articulaciones pasivas.** Como indica el nombre, son las articulaciones que aunque en el archivo descriptivo del robot figuren como articulación no deben efectuar ningún movimiento. Se definen con la intención de que queden inmóviles en la planificación.

El último paso genera todas las configuraciones hechas en el robot en un paquete ROS, en el que está incluido el archivo *srdf*, con el que se podrá trabajar en *MoveIt!* Una vez creado el paquete se puede acceder a los archivos y modificarlos. No sólo es útil para pequeñas modificaciones y arreglos que haya que hacer sin necesidad de abrir toda la aplicación y cargar el robot, sino que a lo largo de los experimentos ha servido para corregir errores o malas configuraciones del propio asistente.

5.1.3. PLANIFICACIÓN DE TRAYECTORIAS EN MOVEIT!

La planificación de trayectorias, explicada antes, es un proceso relativamente sencillo en *MoveIt!* Consiste en un plugin que se puede ejecutar desde RViz o desde *benchmarking*.

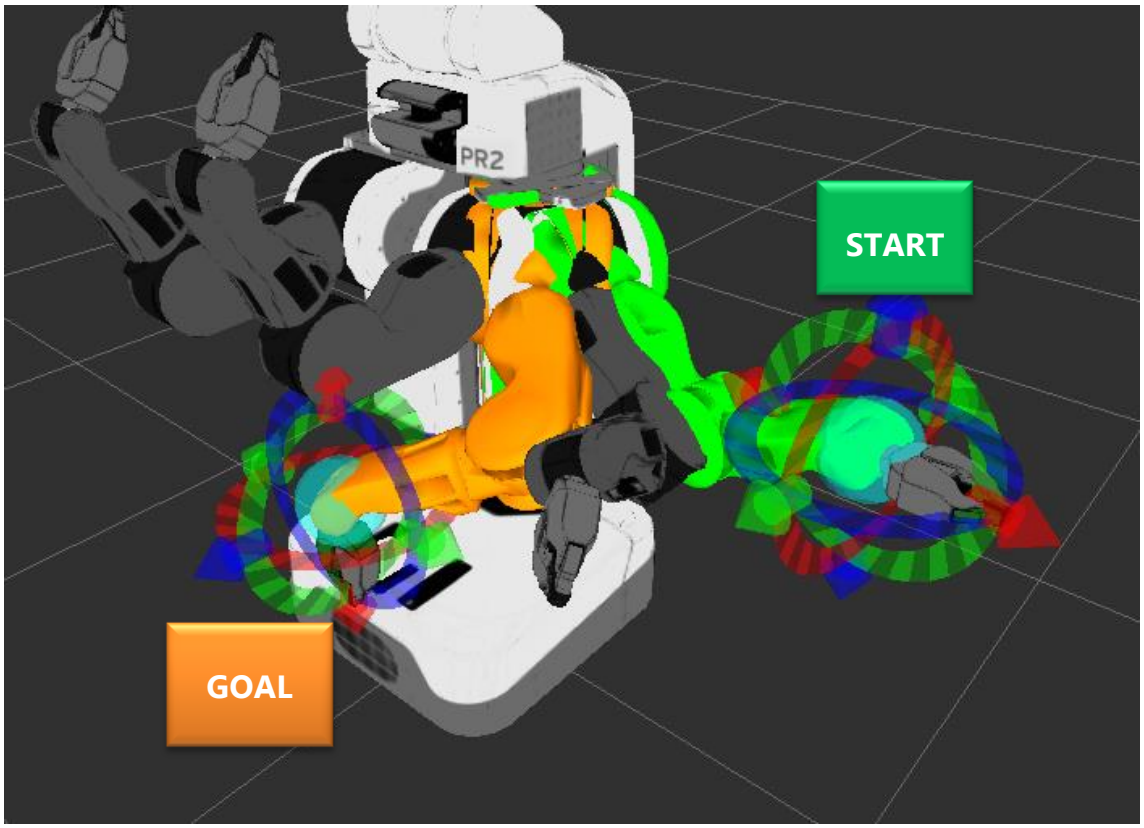

Figura 21. Planificación de trayectorias de un brazo del robot PR2 en RViz

En la Figura 21, el brazo del robot de color verde representa la posición inicial (*start state*) y el naranja la posición de final (*goal state*). Las esferas, las flechas y las circunferencias son marcadores interactivos de RViz que puede manipular el usuario en el espacio.

El plugin de planificación necesita el entorno y los requerimientos. Dentro de la librería OMPL se elige un algoritmo de planificación concreto. En el proceso de obtención de la trayectoria el plugin realiza además reducciones y suavizaciones (figura 22) para simplificar el recorrido del robot.

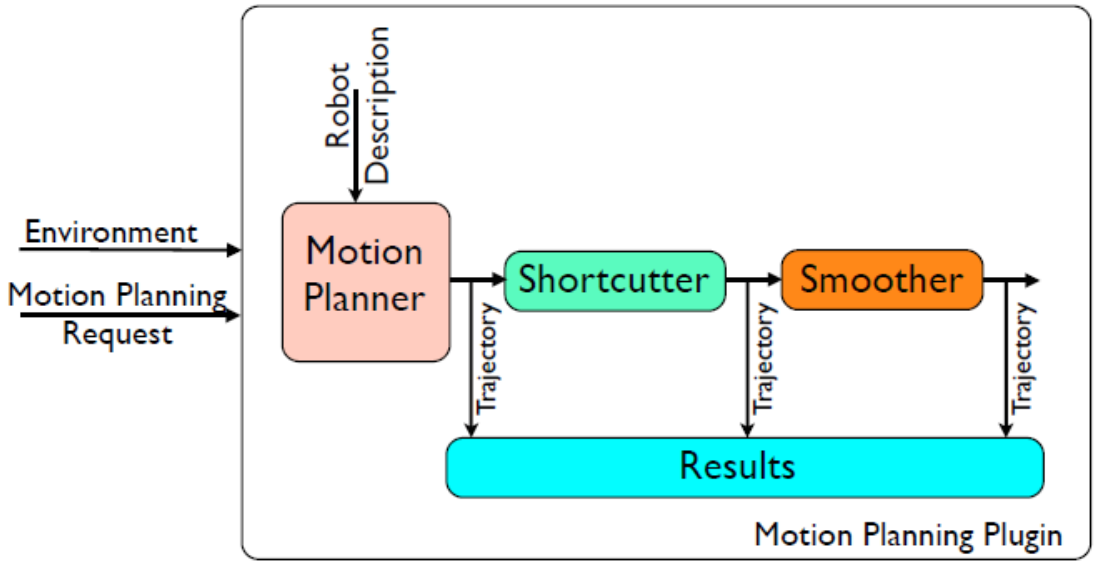

Figura 22. Esquema del plugin de planificación de trayectorias

5.1.4. BENCHMARKING

La idea que justifica el proceso de *benchmarking* es hacer una ejecución de todos los planificadores y comprobar el más eficiente con el propósito. Por tanto para realizar los *benchmarks* se necesita a su vez el plugin de planificación de trayectorias explicado en el apartado anterior, y eso da lugar al nombre de servidor referido a *benchmarking* (representado en la figura 23).

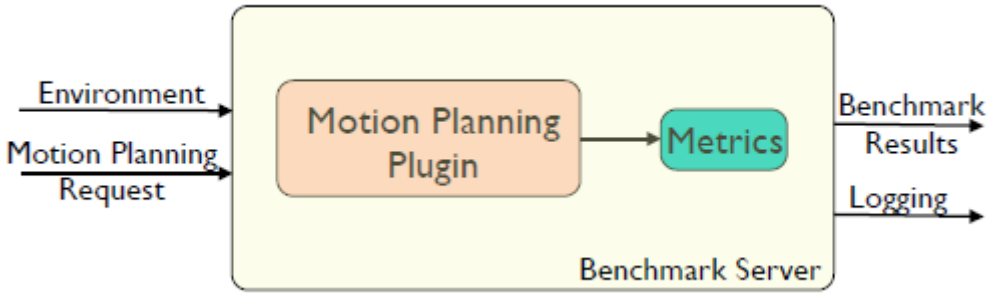

Figura 23. Ejecución del plugin de planificación de trayectorias en el servidor benchmark

La ejecución del plugin de planificación exige los mismos recursos que si se hiciera por separado, y da lugar a los resultados o *metrics* de esa planificación, que en *benchmarking* no es más que la comparación de una serie de ejecuciones.

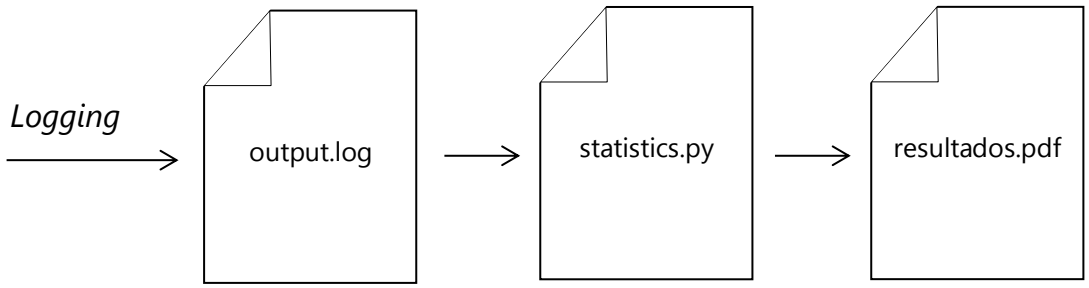

Figura 24. Proceso de logging detallado

Para obtener los resultados de *benchmarking* de forma gráfica y ordenada se ejecuta un archivo en lenguaje *phyton* proporcionado por *MoveIt!* que transforma las notas de *logging* de todos los procesos de planificación en formato PDF (figura 24).

- **Configuración de los requerimientos para la planificación**

Los requerimientos de la planificación son condiciones que se debe exigir al planificador seleccionado. En *MoveIt!* se hace a través de un archivo de configuración escrito por el usuario en el que se detalla el nombre del entorno y los planificadores deseados para comparar y el número de ejecuciones que se desea realizar de cada uno. Según la figura 23 el archivo de configuración es el *input* denominado "*Motion Planning Request*".

Para obtener un resultado más fiable se debe aumentar el número de *runs* o ejecuciones de cada planificador, aunque esto requiere de mucho más tiempo de procesado. Esto se debe a que los resultados entre una prueba y otra pueden ser completamente diferentes.

Este tiempo también podrá variar en función del tiempo máximo que establezca el usuario para procesar la trayectoria en el plugin de planificación. Según las aplicaciones y la complejidad del entorno o de las articulaciones del robot, habrá que especificar un límite máximo de tiempo de planificación mayor o

menor. En las pruebas realizadas en este proyecto, la mayoría de valores típicos obtenidos en benchmarking tenían éxito planificando en un intervalo de 5 segundos, como en la captura de la figura 25.

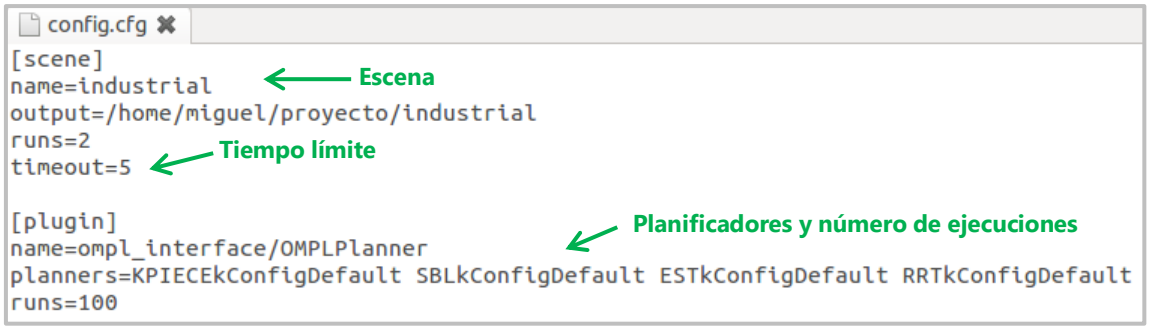

Figura 25. Ejemplo de archivo de configuración

- **Metrics**

A lo que se refiere el esquema del servidor de *benchmark* con *metrics* es a las variables que miden el rendimiento, la fiabilidad y la calidad de las trayectorias planificadas por los algoritmos. *MoveIt!* calcula y genera los *metrics* gráficamente para que el usuario pueda comprobar qué algoritmo de los que se han probado se adapta mejor a la planificación requerida.

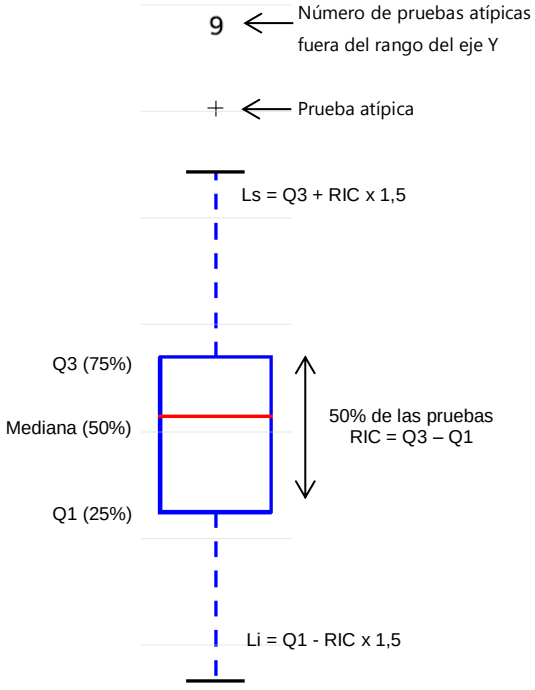

Figura 26. Diagrama de cajas y bigotes.

La representación gráfica se hace en distribución por cajas y bigotes (figura 26). Este tipo de gráficos suele utilizarse en distribuciones de un número alto de pruebas con resultados numéricos. Se ve fácilmente la ubicación de marcadores útiles como la mediana y valores atípicos, o la posición axial de la mitad de las pruebas realizadas.

En *MoveIt!* están implementadas las siguientes variables o *metrics*:

→ *Tiempo de computación (Plan computation times):* incluye el tiempo de procesado en la planificación inicial y en las simplificaciones de la trayectoria. Está excluido el tiempo de post-procesado. Lo deseable son tiempos muy bajos, que no sólo hacen que *benchmarking* sea rápido sino que significan que probablemente la ejecución se esté realizando bien porque no está consumiendo el tiempo máximo. Se mide en segundos.

→ *Longitud del camino (Path lenght):* es la distancia total en el espacio de la trayectoria planificada. Esta es una de las variables con mayor dispersión, y en las que se puede detectar fácilmente qué algoritmos son ineficientes para la aplicación. Se mide en radianes, porque el movimiento de las articulaciones es angular (figura 27).

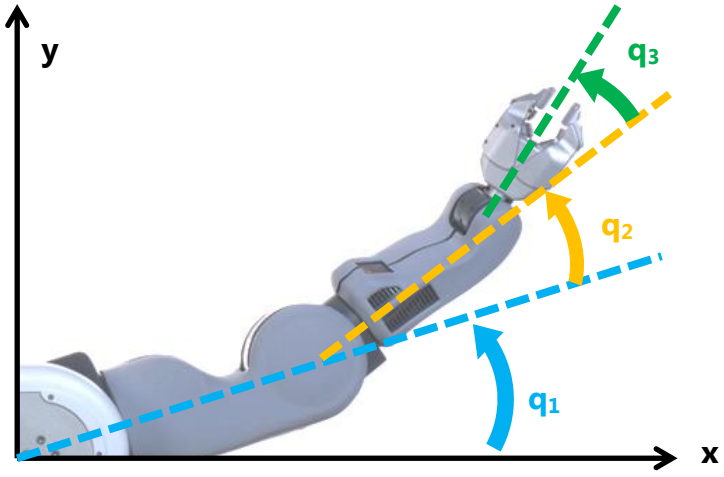

Figura 27. Movimiento angular de las articulaciones del brazo

→ *Suavidad de la planificación (Smoothness of plans):* variable cuantitativa de la suavidad de las trayectorias generadas. Se calcula con la ecuación:

$$k' = \frac{1}{n} \sum_{i=2}^{n} \alpha_i^2$$

Donde k' es la variable de la suavidad, α_i el ángulo entre dos tramos consecutivos de una planificación con n tramos.

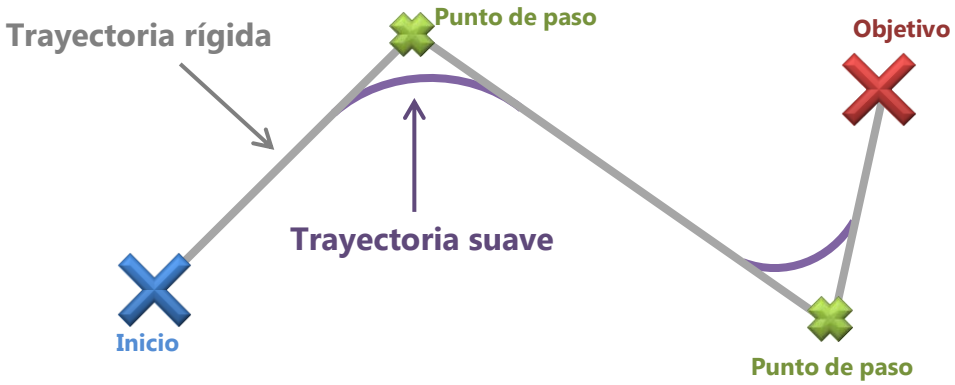

Figura 28. Comparación gráfica de una trayectoria rígida frente a una suave

En la figura 28 se representa el concepto de la suavidad en las trayectorias y muestra que se pueden conseguir los objetivos de una planificación con diferentes grados de suavidad. El inconveniente de una planificación más suave frente a una más rígida, es que el tiempo de procesamiento y la longitud de la trayectoria serán mayores.

→ *Holgura (Clearance):* distancia media límite entre los puntos de paso de la trayectoria planificada y los obstáculos del entorno. Se mide en metros.

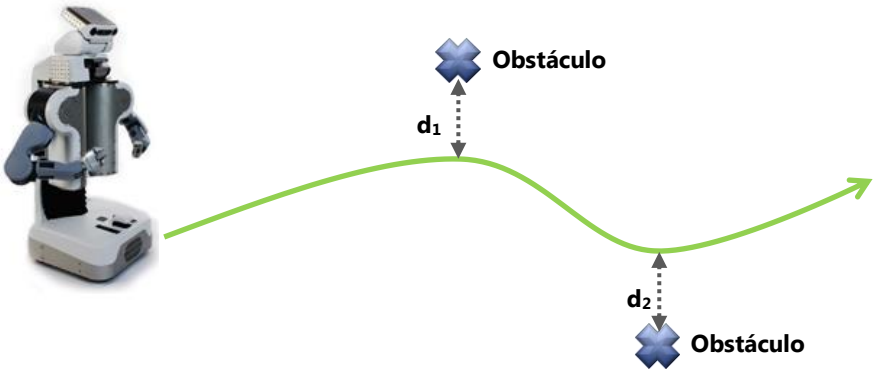

Figura 29. Trayectoria con obstáculos

La figura 29 es una simplificación de una trayectoria con dos obstáculos. En este caso, la holgura sería la media entre las dos distancias desde los obstáculos a la trayectoria planificada.

→ *Ratio de éxito (Success rate):* porcentaje de intentos de los cuales han conseguido resolver la trayectoria en el tiempo especificado, respecto de los intentos totales. Esto significa que no todas las ejecuciones del *plugin* de *motion planning* alcanzan una solución en el tiempo máximo especificado, motivo por el que para *benchmarking* se realizan muchas pruebas. Es un valor porcentual.

→ *Tiempo total (Total time):* es la suma entre el tiempo de procesado y el de post-procesado (suavización, interpolación,...). Se mide en segundos.

5.2. Robots

Para este proyecto se han utilizado dos robots: PR2 y Manfred. A continuación se detalla la configuración utilizada para las pruebas del trabajo.

5.2.1. PR2

Los principales grupos de planificación de PR2 (figura 30) son los brazos (desde el la articulación con el cuerpo hasta el final, pinza incluida). Además se define una articulación virtual entre el robot y el suelo sobre la que se efectúa el desplazamiento planar del robot.

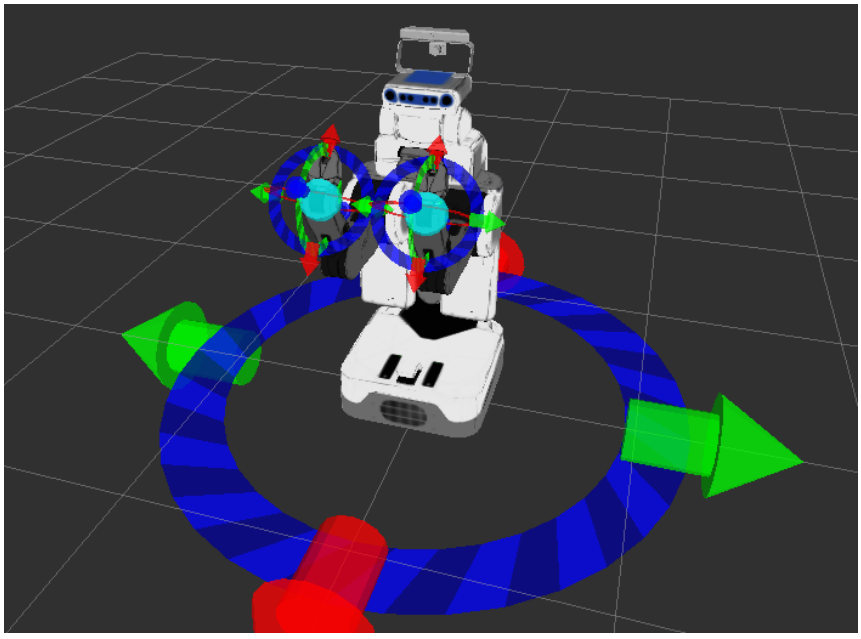

Figura 30. PR2

Se puede ver que sus grupos de planificación configurados son los brazos izquierdo y derecho (que mueven las pinzas correspondientes como *end effectors*) y el robot entero sobre el suelo (sólo se puede mover en dos dimensiones y no en tres como los brazos porque ha sido configurado como movimiento planar).

5.2.2. MANFRED

Para Manfred, los grupos de planificación son el brazo y la mano. La mano además está configurada como *end effector* porque es el elemento principal de la planificación. En la figura 31 se representa el robot con su diseño en RViz, la matriz de colisión y la superposición de ambos volúmenes.

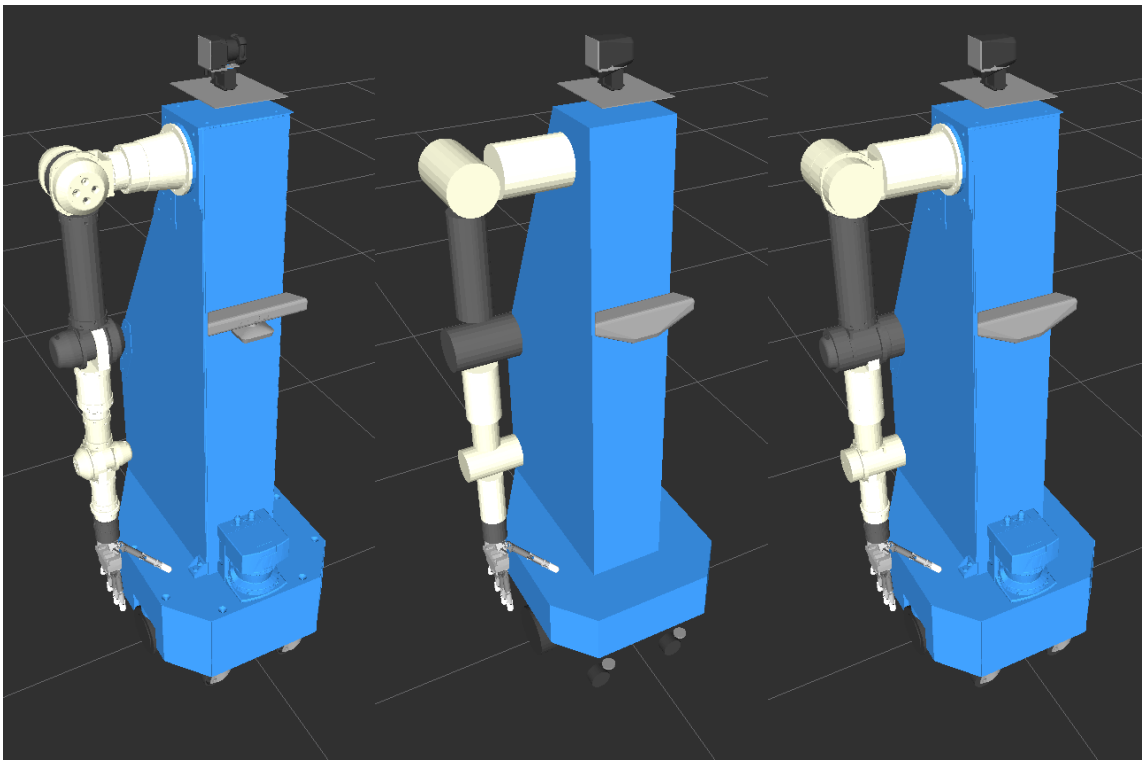

Figura 31 Manfred y su matriz de colisión

5.3. Pruebas de benchmarking

Con los robots configurados, el siguiente paso es probar en varios entornos y con varias trayectorias y planificadores, la verdadera utilidad práctica de *benchmarking* en *MoveIt!* Para conseguirlo se han realizado pruebas concretas en ambos robots que comprueban la eficacia de cada algoritmo según el robot o los obstáculos del escenario.

La librería OMPL en *MoveIt!* proporciona cuatro algoritmos básicos (figura 32), que a su vez tienen variantes o mejoras. Por ello, el procedimiento realizado ha sido ver a modo de comparación qué algoritmo es más adecuado para la aplicación estudiada respecto a los demás o respecto a las versiones de ese mismo.

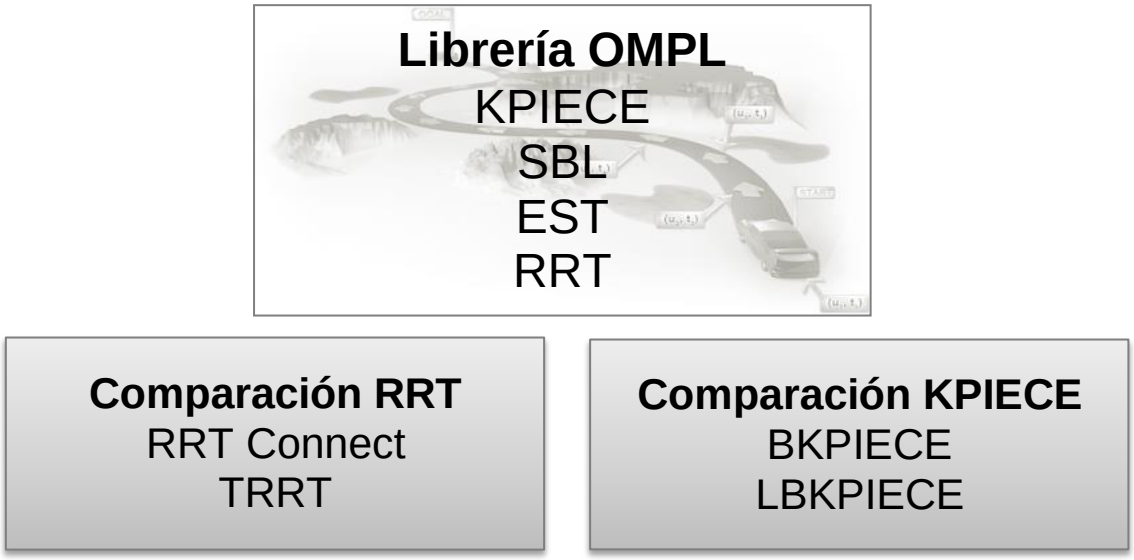

Figura 32. OMPL y sus variantes

En las pruebas se trata de ver qué variables son críticas en diferentes escenarios y con diferentes trayectorias con cada robot, así como qué algoritmo es más adecuado para cada una de ellas. A continuación se describen algunas pruebas realizadas con los robots.

5.3.1. MOVIMIENTO SIMPLE DE BRAZO

El objetivo de esta prueba es el de establecer la opción más sencilla como la base sobre la que se irá complicando la planificación de trayectorias. Es decir,

tener una referencia de cómo se comportan los planificadores en las condiciones más simples (figura 33).

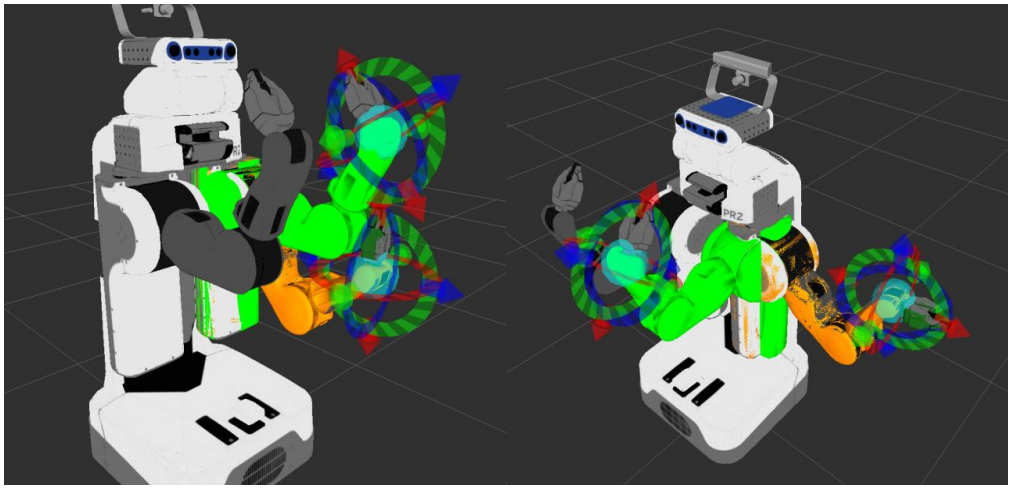

Figura 33. Dos planificaciones sencillas con un solo brazo

5.3.2. MOVIMIENTO COMPUESTO DE BRAZO Y BASE

Aquí interesa la articulación virtual robot-suelo (figura 34) y su comportamiento en *benchmarks*. Si para el movimiento simple del brazo algunos algoritmos de planificación no conseguían tener éxito en el tiempo máximo establecido, para este caso se necesita aumentar sustancialmente. En *MoveIt!* los algoritmos de OMPL funcionan mucho más rápido con el movimiento de articulaciones que con desplazamientos del robot entero.

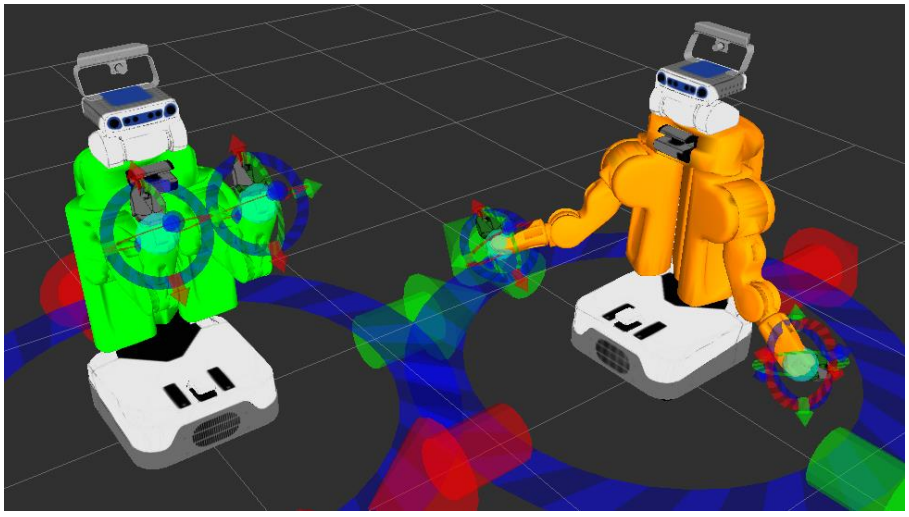

Figura 34. Posiciones del robot con desplazamiento de la base

5.3.3. MOVIMIENTO COMPUESTO DE BRAZO Y PINZA/MANO

Es otro tipo de movimiento compuesto, pero esta vez con la base del robot fija en el suelo. La planificación de la pinza parece sencilla por, a pesar de tener varios links, ser bi-estable (abierto y cerrado), pero la mano del robot Manfred puede complicarla mucho. Por este motivo va a ser este último robot el elegido para este caso.

La configuración de las posiciones deseadas se ha hecho con el asistente de configuración de los robots porque permite más precisión que con RViz. En el asistente el usuario puede ajustar con mucha precisión la posición de cada articulación. Las barras y los cuadros de texto sirven para variar el valor de cada articulación contenida en un grupo de planificación determinado (el brazo entero en este caso, en la figura 35). Es el método elegido para generar las posiciones en Manfred, porque sería mucho más difícil manipulando cada articulación en el espacio de RViz.

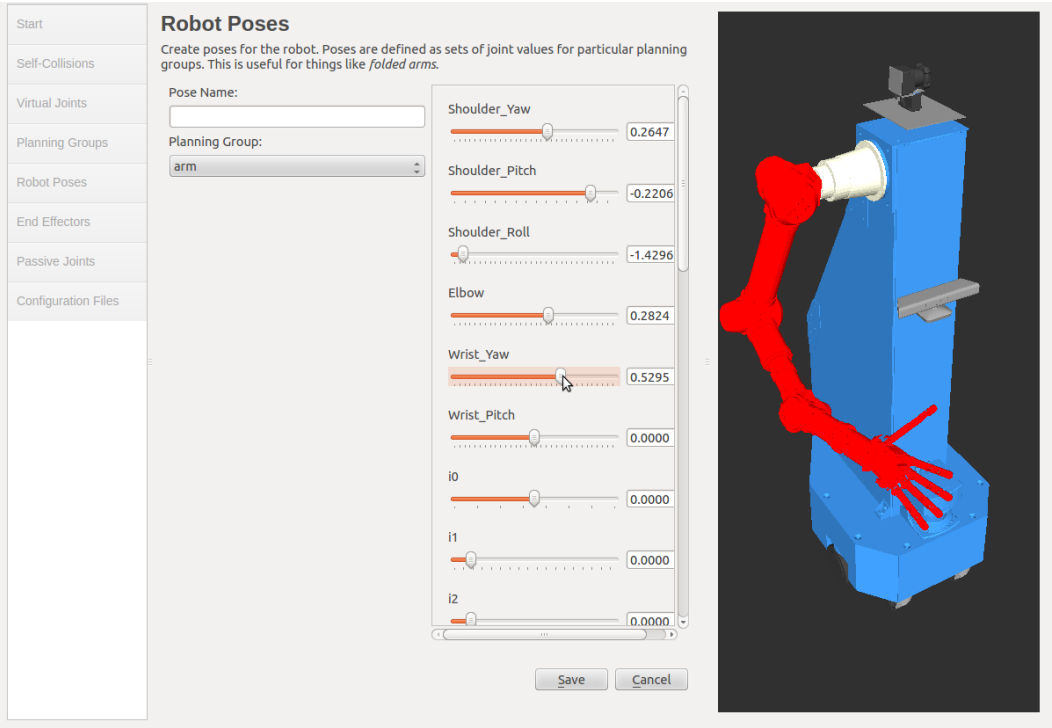

Figura 35. Configuración de las poses del robot en el asistente

Después de configurar ciertas posiciones del robot se procede a establecer esas posiciones como de inicio o meta, y finalmente se procede con los *benchmarks*. Con esta prueba se podrá determinar qué diferencias significativas hay entre

ambos robots y los algoritmos, así como ver de qué manera aumenta la dificultad de la planificación según haya más o menos articulaciones.

Para este caso se han configurado posiciones que pueden ser interesantes para la planificación.

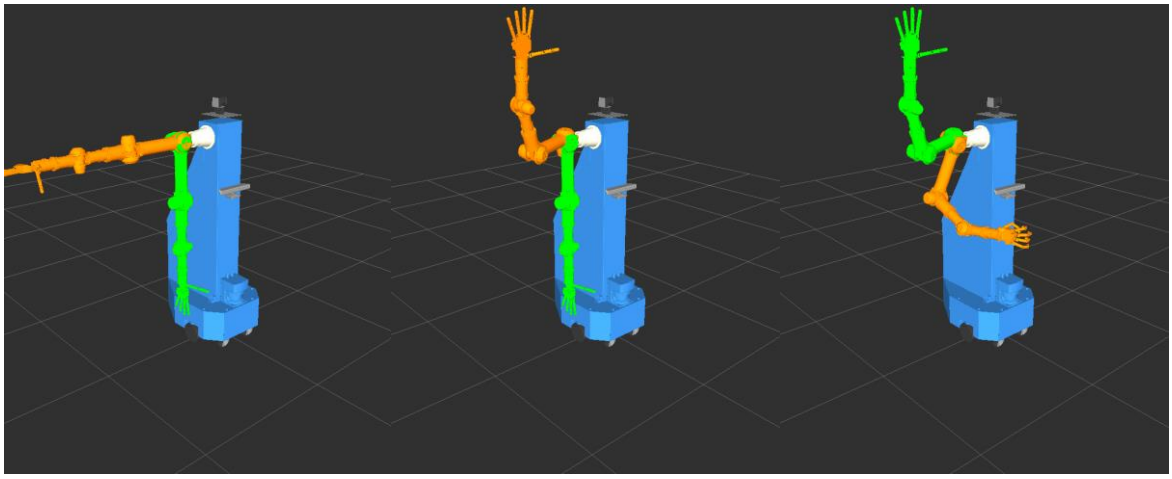

Figura 36. Pruebas realizadas en Manfred

Entre las posiciones elegidas está la del brazo en reposo, el brazo estirado, y otras en las que se ha intentado mover todas las articulaciones para complicar al máximo la tarea del planificador. La figura 36 representa las tres pruebas realizadas con Manfred y la planificación brazo-mano.

5.3.4. PRUEBA INDUSTRIAL

Esta prueba es en un escenario concreto: una mesa de operaciones con huecos y cajones para objetos, donde el robot puede montar o transportar piezas. Se van a probar algunas posiciones y se van a analizar los resultados.

- **Movimiento de un solo brazo**

En este caso, se han configurado dos posiciones diferentes en el brazo del robot, una con el izquierdo y otra con el derecho (figura 37). La finalidad comprobar que ambos brazos pueden ser planificados por separado. Interesan posiciones para las que sea necesario una planificación compleja, es decir, exigir a los algoritmos esquivar obstáculos.

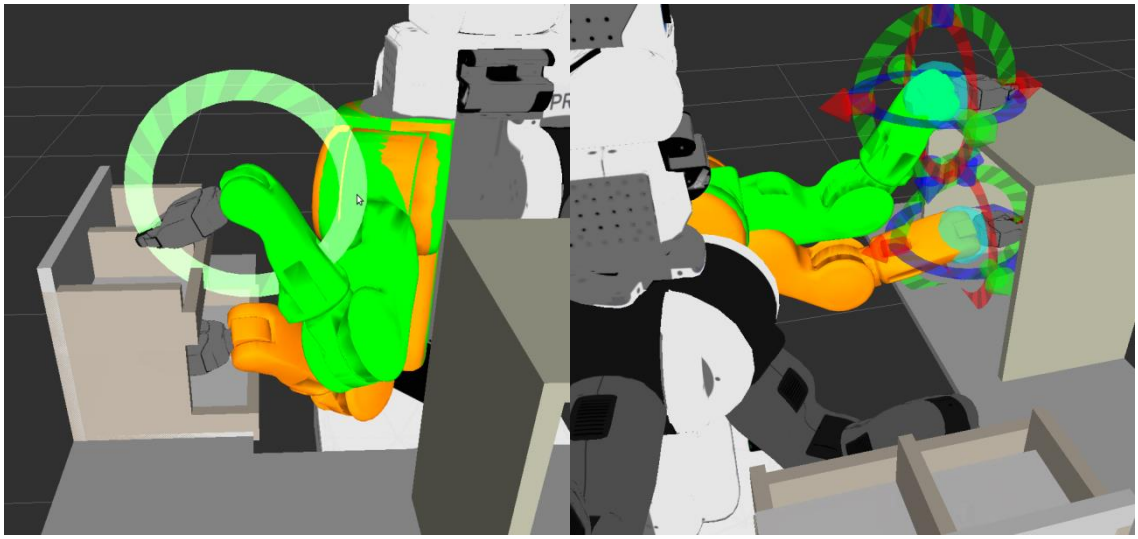

Figura 37. Posiciones de los brazos para la prueba Industrial

- **Movimiento de ambos brazos**

Para esta prueba se aumenta la dificultad debido a que los planificadores trabajan con un grupo que incluye dos subgrupos, y ambos realizan una trayectoria paralelamente.

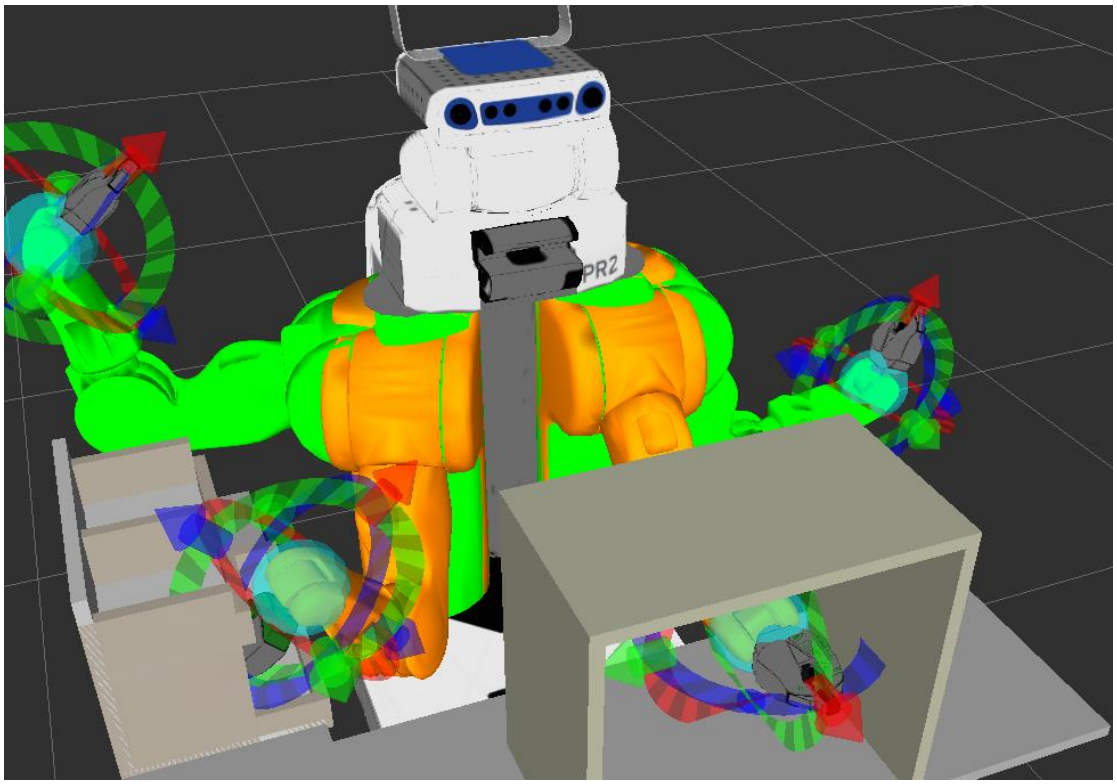

Figura 38. Posiciones de inicio y final en ambos brazos

Como se ve en la figura 38, ambos brazos parten del reposo hacia posiciones dentro de los huecos del banco del escenario con la intención de probar los planificadores en trayectorias con obstáculos.

5.4. Obtención de resultados

El análisis de los resultados se realiza comparando el rendimiento de los planificadores obtenido del proceso de benchmarking. Pero también pueden interpretarse ciertos comportamientos durante la ejecución.

Desde que el usuario da la orden hasta que se generan los resultados puede llegar a pasar mucho tiempo. De una ejecución lenta se puede extraer la conclusión de que los planificadores están empleando más tiempo de lo esperado, lo que significa una trayectoria difícil o imposible con las condiciones establecidas.

Los mensajes de información indican datos relevantes para la planificación como el tiempo que se ha tardado en encontrar la solución, el tiempo invertido en la simplificación o la imposibilidad de encontrar una solución.

En la figura 39 se puede ver un ejemplo de *benchmarking* lento. Se trata de la prueba *Industrial* con los dos brazos. Aunque el planificador RRT en este caso sí encuentra una solución en el intervalo de tiempo especificado, la mayoría de los intentos fracasan.

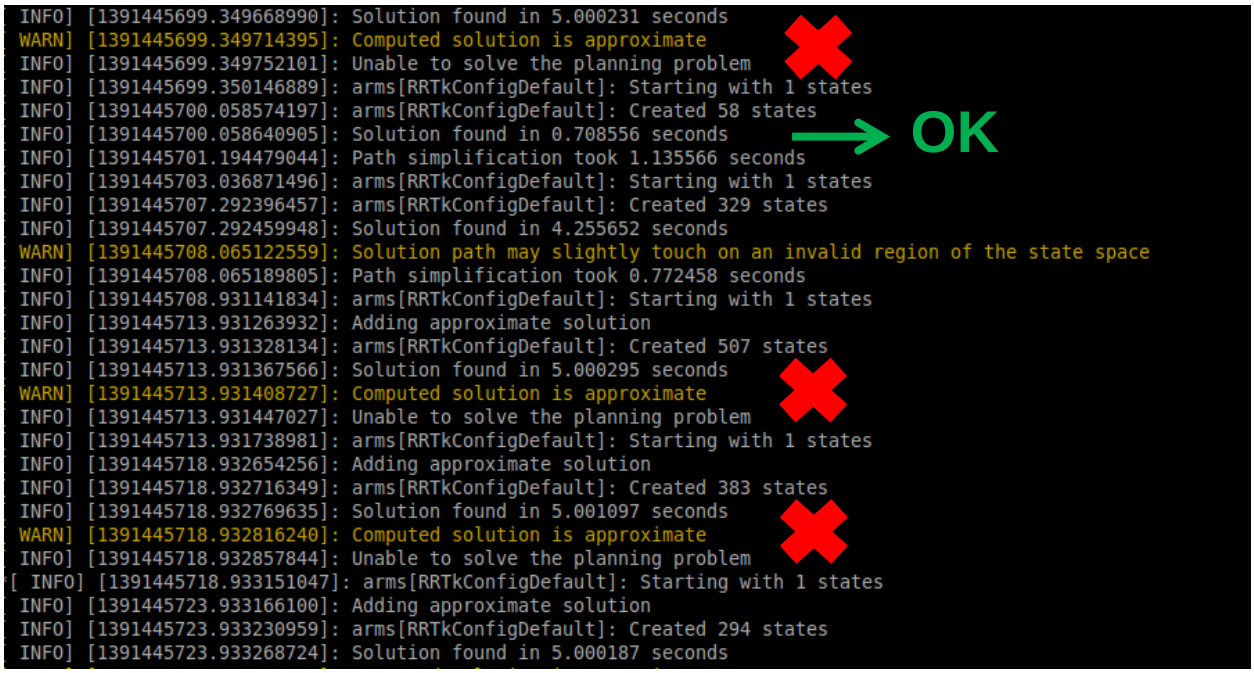

Figura 39. Intentos fallidos en la planificación.

Un gran número de intentos fallidos supone mucho tiempo perdido en la planificación. Por tanto, para planificaciones complejas se disminuye el número de ejecuciones a la vez que se aumenta el tiempo límite para aumentar la tolerancia y que más pruebas sean satisfactorias. En la tabla 3 se representan los casos más habituales y la configuración alternativa en el caso de cuatro planificadores simultáneos.

Tabla 3. Cálculo de tiempos de planificación máximos.

Nº de planificadores	Tiempo límite (s)	Ejecuciones	Tiempo máx. consumido
2	5	100	16,6 min
3	5	100	25 min
4	5	100	**33,3 min**
4	10	20	13,3 min

- **Estudio de *benchmarking* con trayectorias simples**

En este estudio se trata de observar cómo funcionan los parámetros de *benchmarking* en situaciones sencillas. No han surgido problemas ni retrasos en la planificación, como se esperaba. Para ilustrar las trayectorias planificadas la figura 40 compara el robot PR2 en RViz con los *queries* indicados con el trazado proporcionado. En este caso la planificación mueve las articulaciones del brazo y la orientación de la pinza.

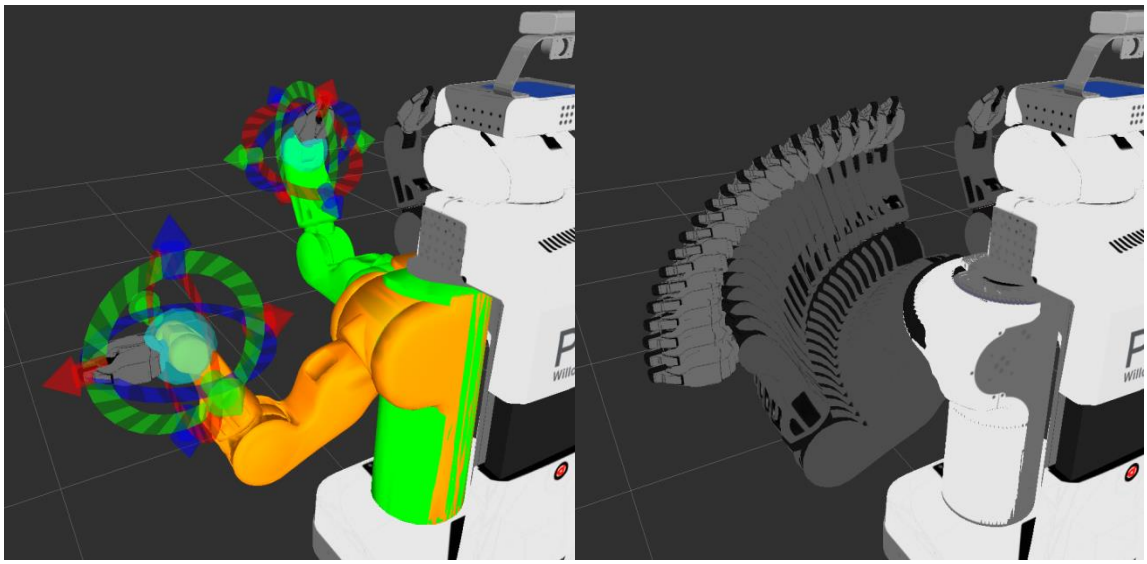

Figura 40. Queries y planificación del brazo izquierdo de PR2

En cuanto a los resultados cuantitativos, las gráficas muestran tiempos de planificación muy bajos en todos los planificadores (figura 41) debido a que es una trayectoria muy sencilla, porcentaje de éxito del 100% y holgura nula, porque no hay obstáculos alrededor. En cuanto a la longitud de las trayectorias

planificadas, hay una diferencia importante entre KPIECE, que traza trayectorias muy largas, y el resto de planificadores. Esta diferencia es proporcional a la suavidad, es decir, cuanto más largas son las trayectorias de un planificador determinado, más suaves las planifica (figura 42).

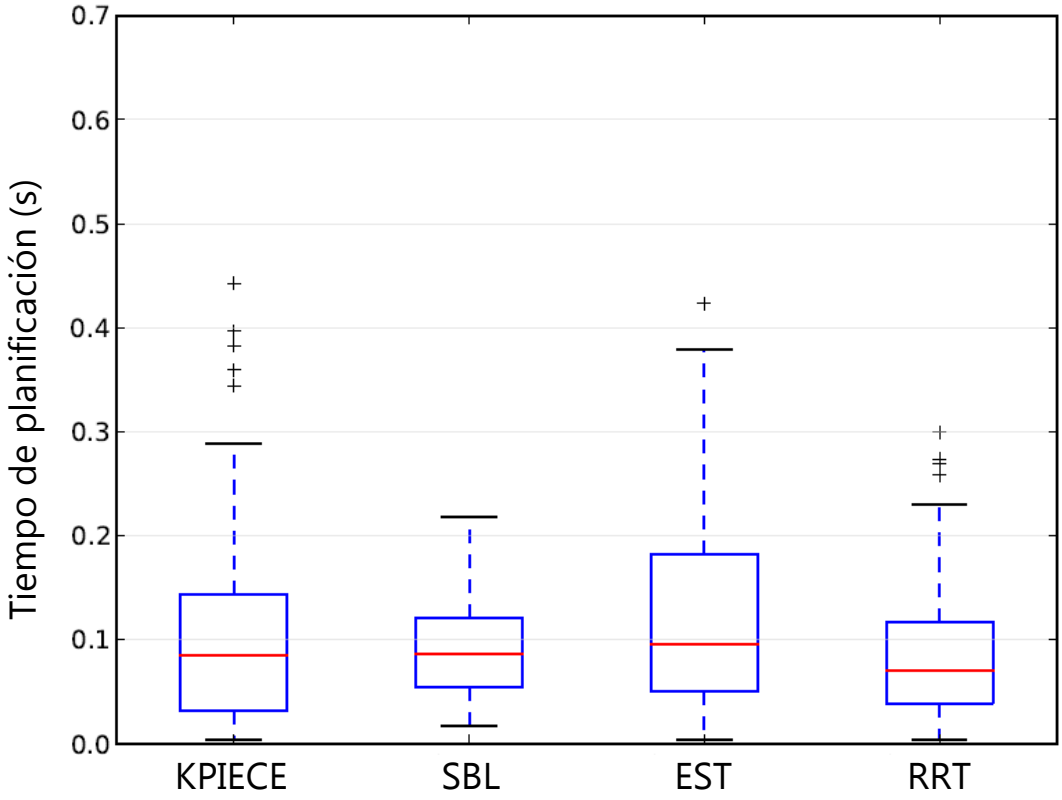

Figura 41. Tiempo de planificación en la prueba simple

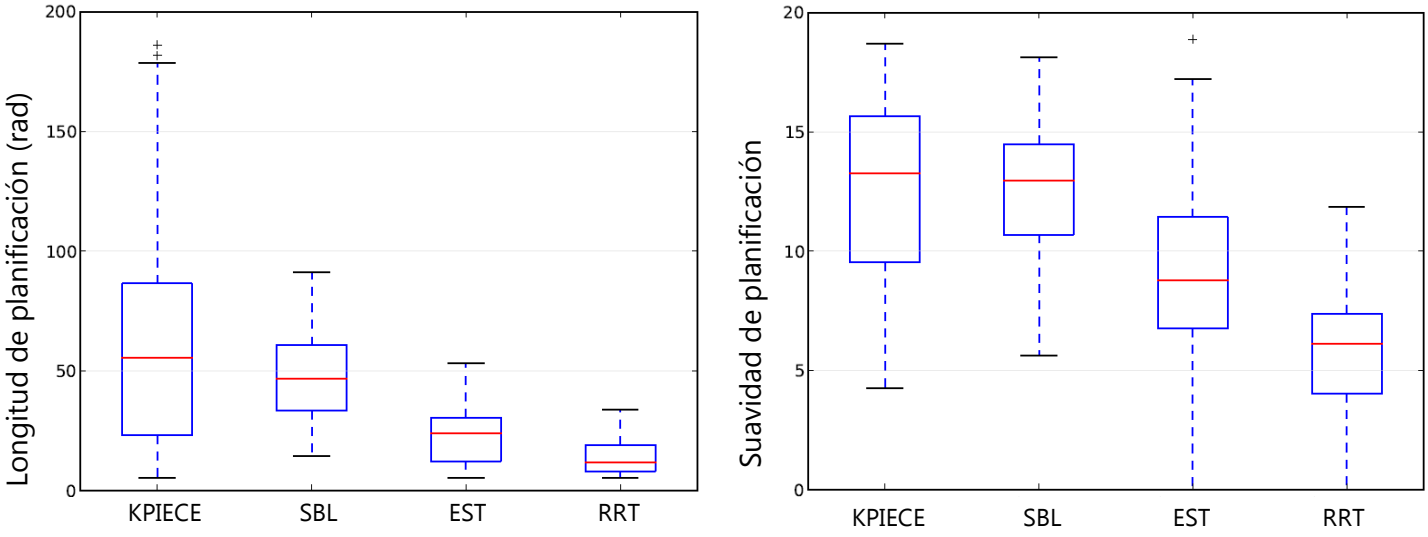

Figura 42. Comparación entre longitud y suavidad de planificación en la prueba simple

También se han realizado pruebas de otros movimientos simples que no se han incluido en este trabajo por dar resultados idénticos. Esto puede interpretarse como que ante un escenario sin obstáculos los planificadores siguen un patrón de comportamiento similar.

- **Estudio de planificación de trayectorias con brazos y base**

Los planificadores operan correctamente trayectorias con una combinación de grupos de planificación, como se verá en las pruebas siguientes. Pero las articulaciones virtuales (el movimiento de la base del robot respecto al suelo) no están implementadas correctamente para la ejecución de *benchmarks* y no se han obtenido resultados numéricos de esta prueba.

A pesar de este problema, el plugin de planificadores en RViz sí es capaz de planificar trayectorias de este tipo y dar una solución gráfica del robot en movimiento, como se ve la captura en la figura 43.

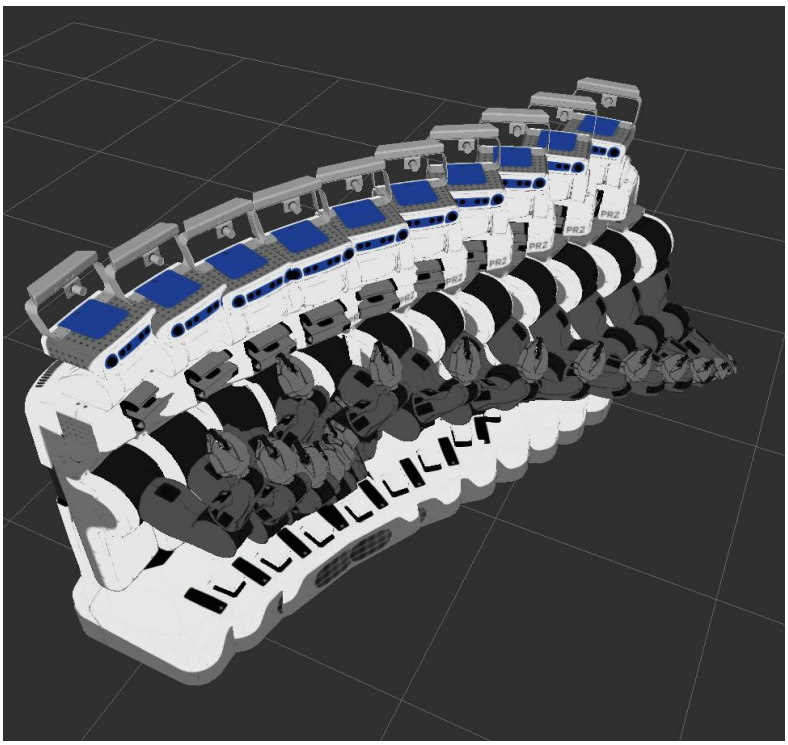

Figura 43. Planificación de una trayectoria de base y brazos

- **Análisis de los resultados de Manfred**

Como se ha explicado antes, Manfred es un robot sobre el que se va a probar la funcionalidad de los algoritmos sobre la mano y el brazo sobre-articulados.

De las tres pruebas realizadas, dos partían de la posición de reposo del brazo. La primera simplemente hacía girar una articulación al máximo: el hombro. Frente a lo que cabía esperar, ha sido mucho más costosa para los planificadores que las demás pruebas. El planificador SBL ha sido incapaz de realizar las pruebas, así que se ha eliminado de los resultados.

Una pre-visualización en RViz (figura 44) proporciona información como el tiempo aproximado de planificación o si es capaz de realizarse en el tiempo establecido, así como qué algoritmos no son aptos.

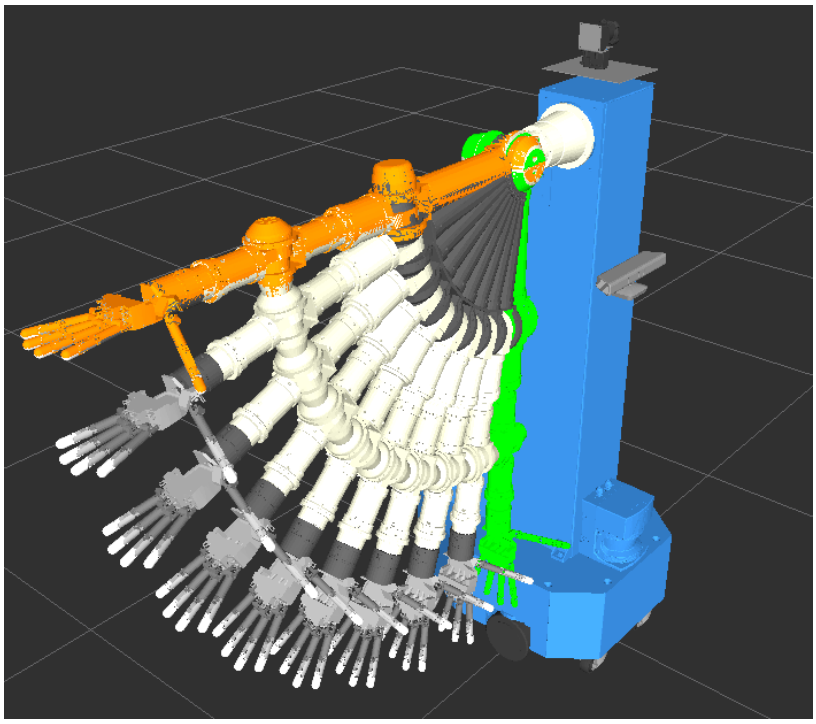

Figura 44. Planificación en RViz (prueba 1)

En los resultados gráficos se observa que la prueba ha sido completada a tiempo por todos los planificadores y en todos los casos, pero no todos con el mismo rendimiento.

Según el tiempo de planificación el algoritmo EST ha sido mucho más lento que KPIECE y RRT (figura 45), y su trayectoria mucho más compleja (figura 46). Esta prueba concluye con el algoritmo RRT como el más eficiente.

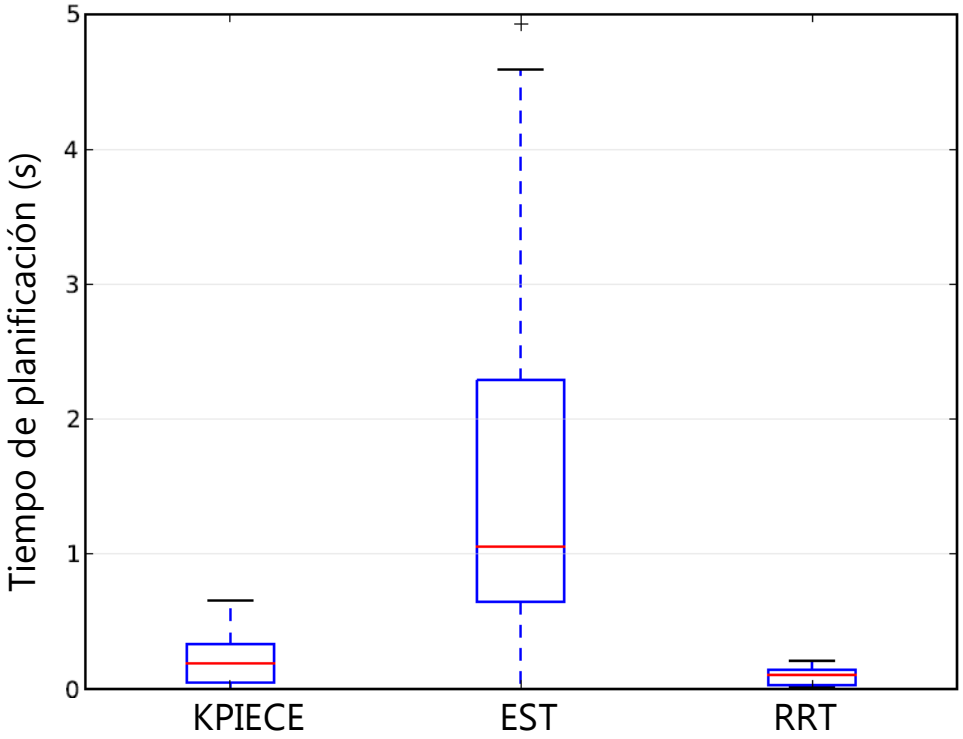

Figura 45. Tiempo de planificación en la prueba 1 (Manfred)

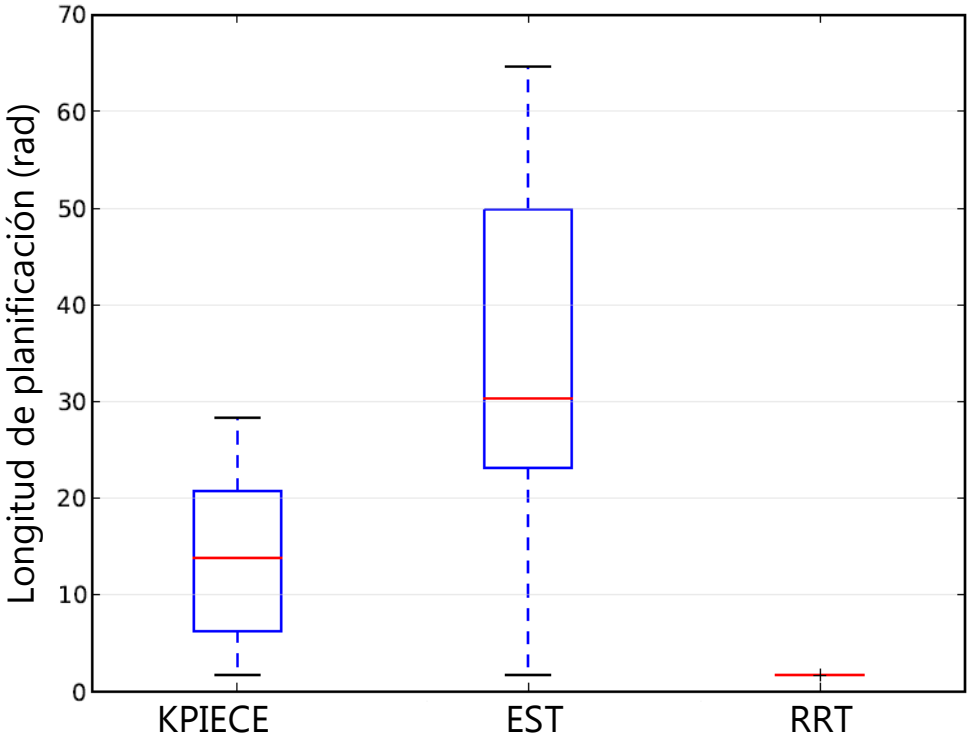

Figura 46. Longitud de planificación en la prueba 1 (Manfred)

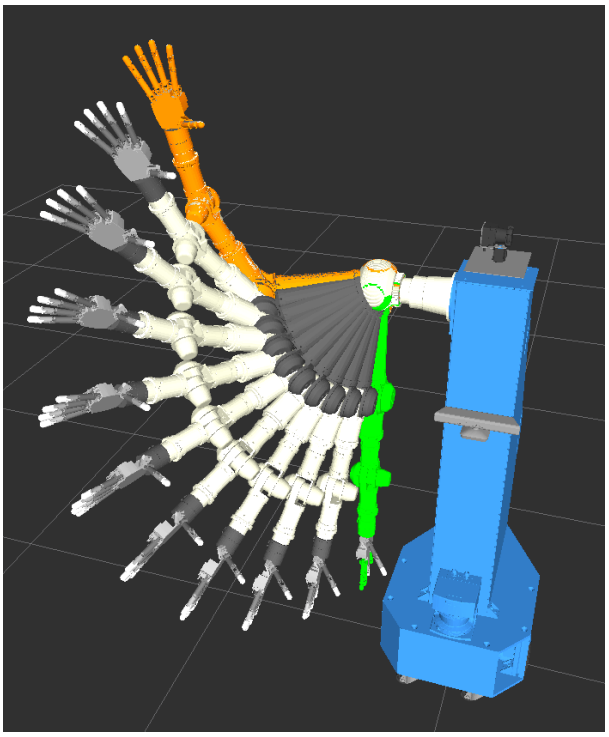

Figura 47. Planificación en RViz (prueba 2)

En la siguiente prueba el brazo parte también del reposo pero esta vez se mueven todas las articulaciones del brazo para llegar a la posición del final (figura 47). El patrón es muy parecido a la prueba anterior: RRT es el planificador que menos espacio de trayectoria necesita (figura 48).

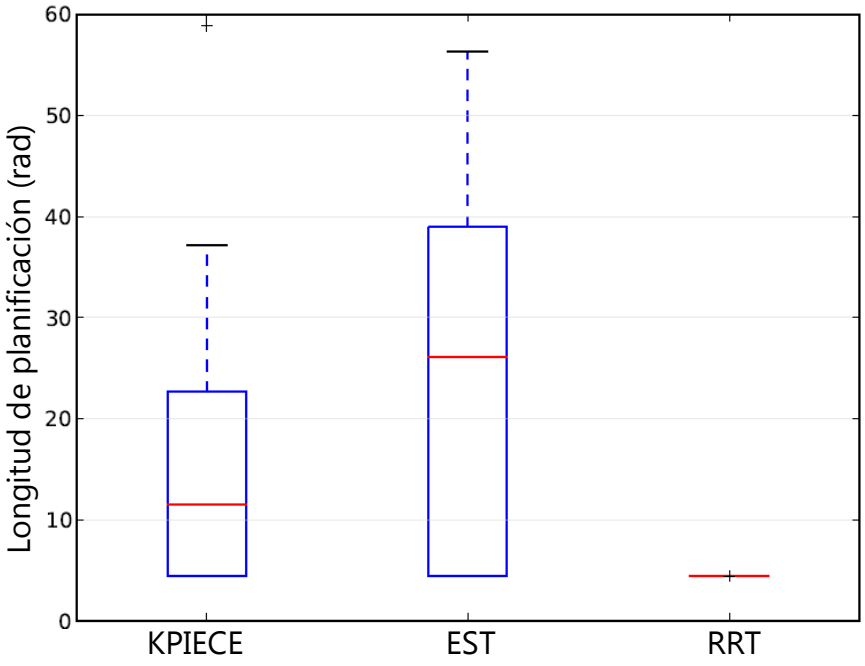

Figura 48. Longitud de planificación en la prueba 2 (Manfred)

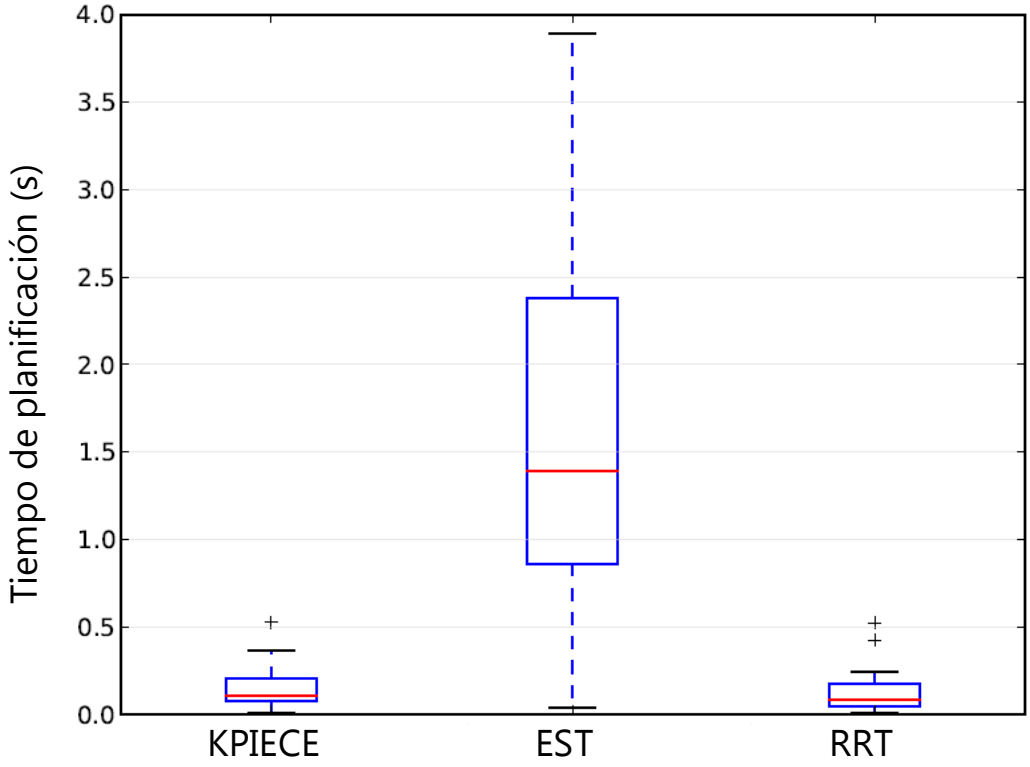

Figura 49. Tiempo de planificación en la prueba 2 (Manfred)

Lo mismo sucede con el tiempo de planificación (figura 49). La conclusión que se puede extraer de estos resultados es que los planificadores puestos a prueba no distinguen entre un movimiento de una sola articulación o de todo el brazo, e incluso son más eficientes frente a trayectorias más complejas.

El rendimiento del planificador RRT es el mejor, seguido por KPIECE. Para la última prueba se ha decidido generar *benchmarks* de comparación entre las dos variaciones del planificador RRT, porque es el que más eficiencia ha demostrado en el robot Manfred.

Ahora será necesaria la planificación de las articulaciones de toda la mano del robot, lo que va a añadir más complejidad a los algoritmos. En la figura 50 se representa el trazado de esta trayectoria y es fácil comprobar que los dedos del robot se flexionan progresivamente al mismo tiempo que el brazo cambia de posición.

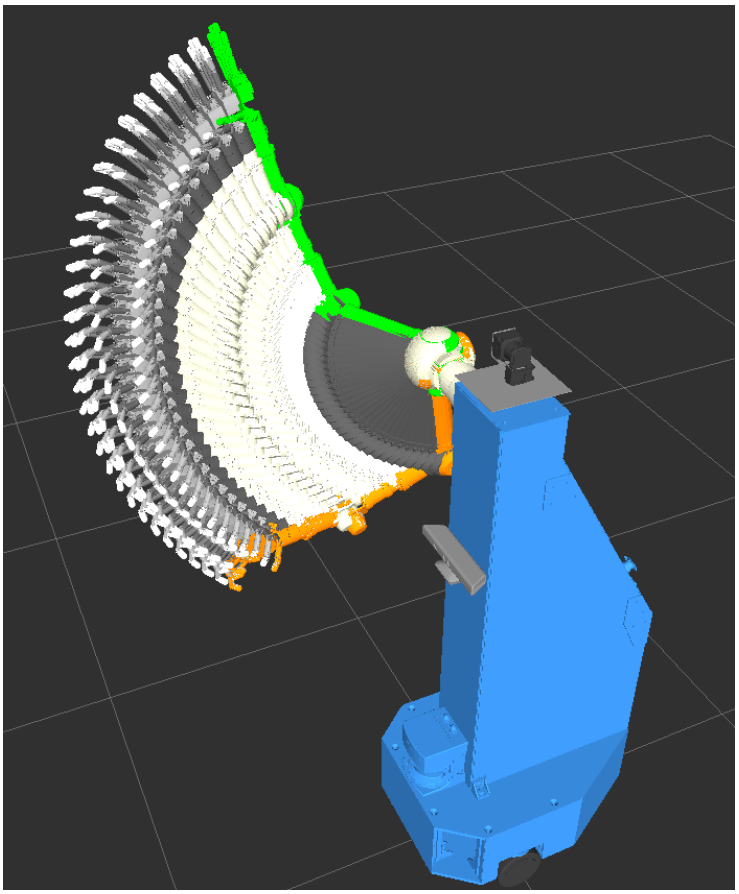

Figura 50. Planificación en RViz (prueba 3)

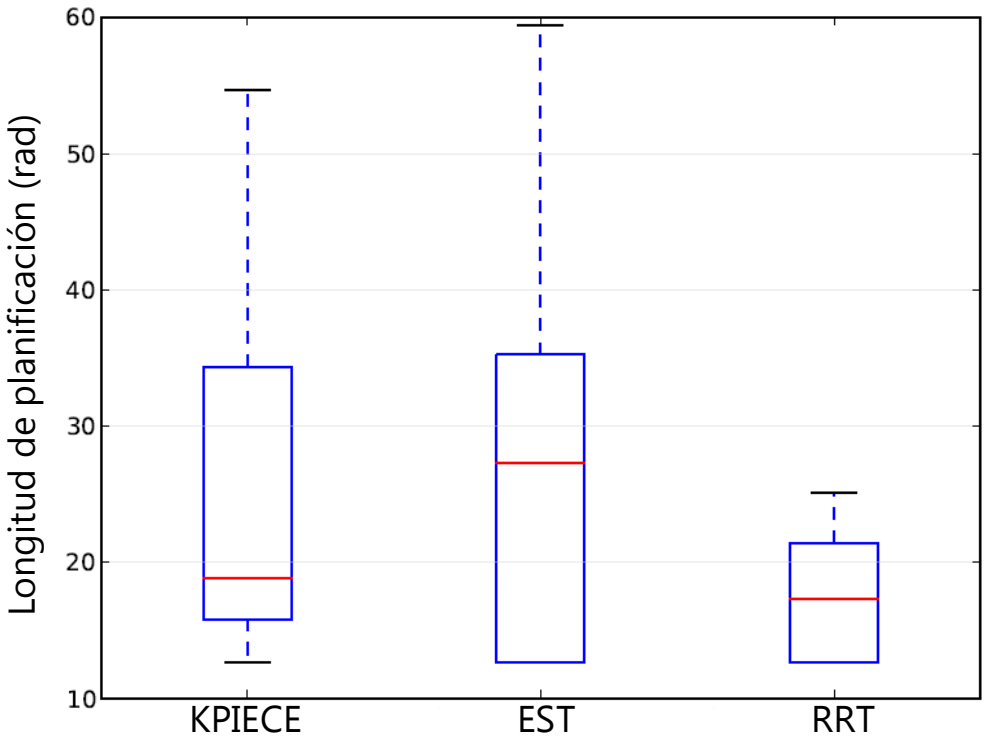

Figura 51. Longitud de planificación en la prueba 3 (Manfred)

Como se esperaba, la longitud de planificación de las articulaciones del brazo y de la mano (figura 51) aumenta respecto a la que sólo incluye el brazo. RRT sigue siendo el planificador que menos tiempo (figura 52) y menos longitud necesita. Como aspecto negativo, es el planificador que planifica trayectorias menos suavizadas (figura 53). Es decir, el tiempo invertido por los otros planificadores se dedica a la suavización.

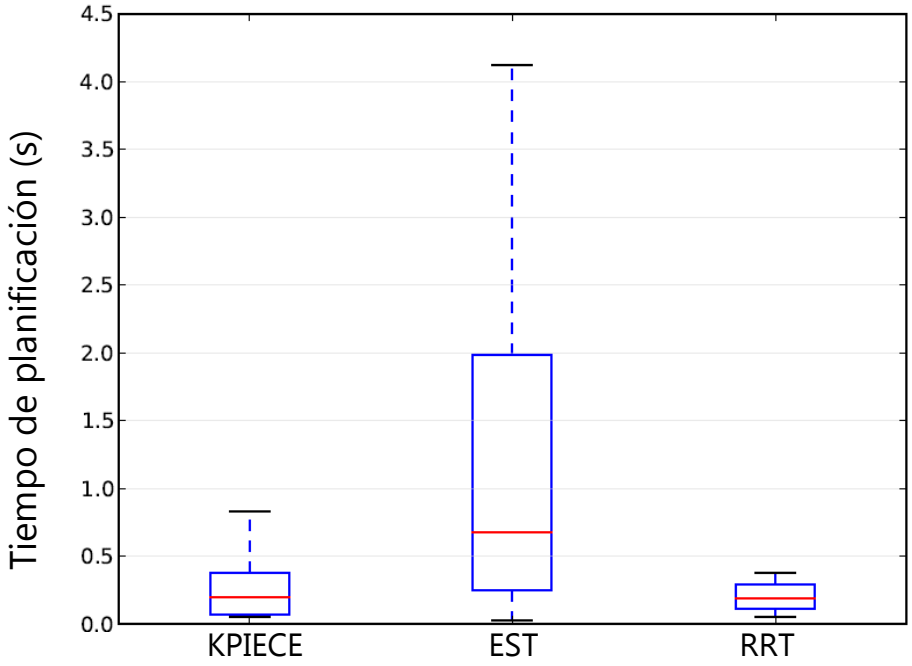

Figura 52. Tiempo de planificación en la prueba 3 (Manfred)

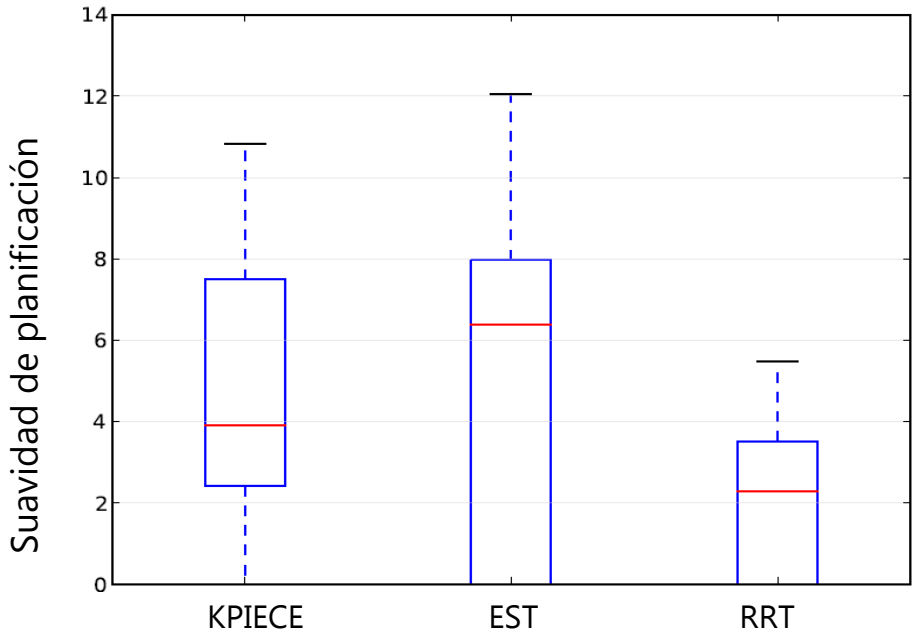

Figura 53. Suavidad de la planificación en la prueba 3 (Manfred)

En la comparativa entre planificadores de RRT se observa que el planificador RRT *Connect* es más eficiente aún que su forma origen: sus pruebas se completan en menos tiempo y con menos longitud de trayectoria (figuras 54 y 55). De nuevo se demuestra que los algoritmos más rápidos son los que generan trayectorias más rígidas (figura 56).

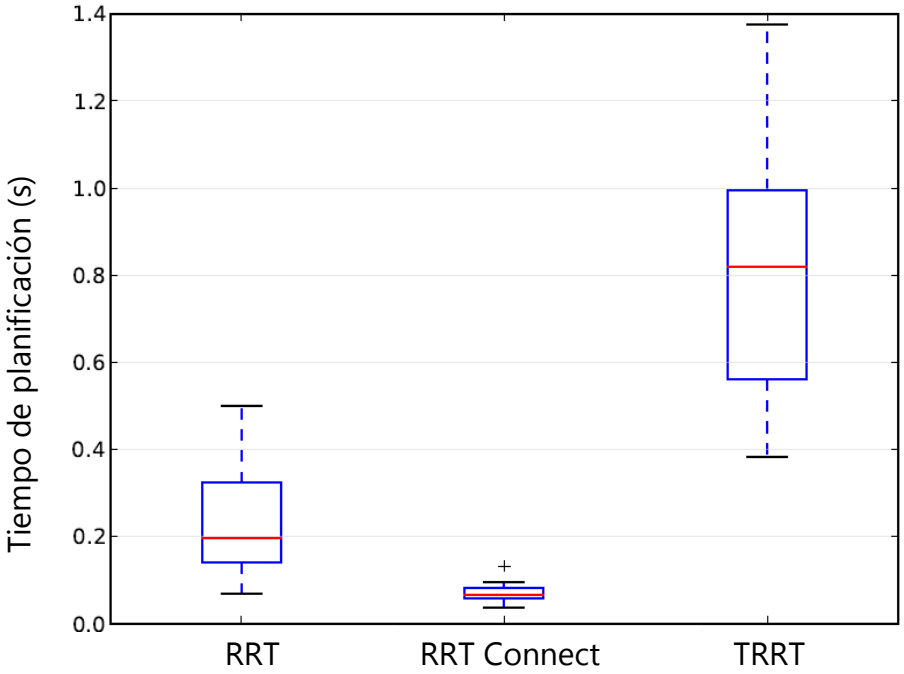

Figura 54. Tiempo de planificación en la comparativa RRT

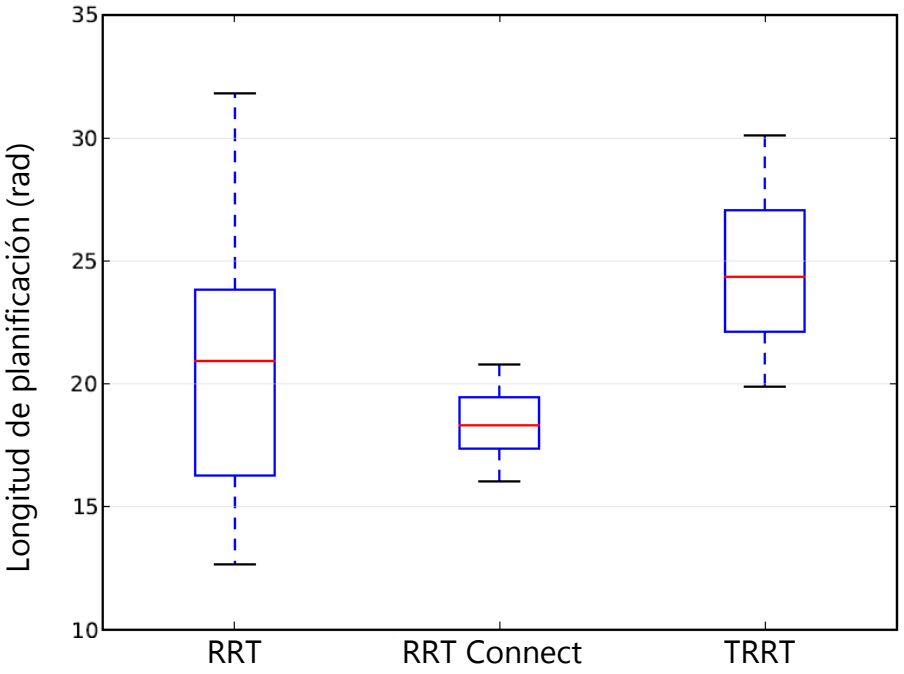

Figura 55. Longitud de planificación en la comparativa RRT

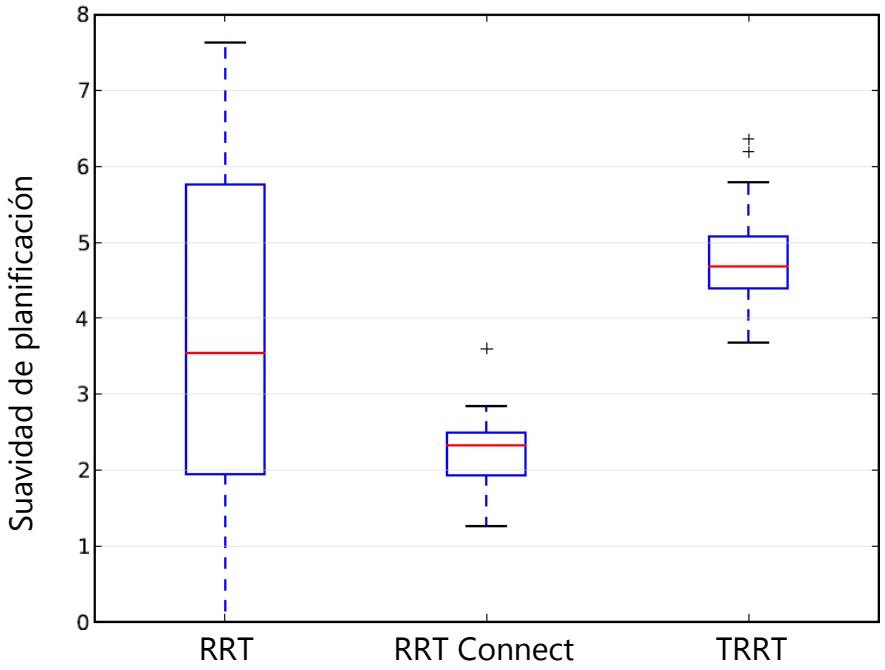

Figura 56. Suavidad de planificación en la comparativa RRT

La conclusión final es que el planificador más eficiente para este robot es RRT, y RRT *Connect* mejora aún más sus resultados. Si en lugar de rápidas y cortas se prefirieran trayectorias más suavizadas, el planificador elegido sería KPIECE.

- **Análisis de la prueba Industrial**

El problema de las planificaciones fallidas puede apreciarse mejor en los resultados gráficos de *benchmarking*. En la figura 57 queda claro el elevado número de pruebas que han tomado más de 5 segundos, que es el tiempo máximo fijado. Si se compara con los resultados de la figura 58 es visible la correlación entre el porcentaje de éxito y el número de pruebas que han superado el tiempo límite. Los números sobre las cajas representan los valores atípicos superiores a 5 segundos, donde en el planificador SBL son 99 de 100 pruebas, es decir, 1% de éxito.

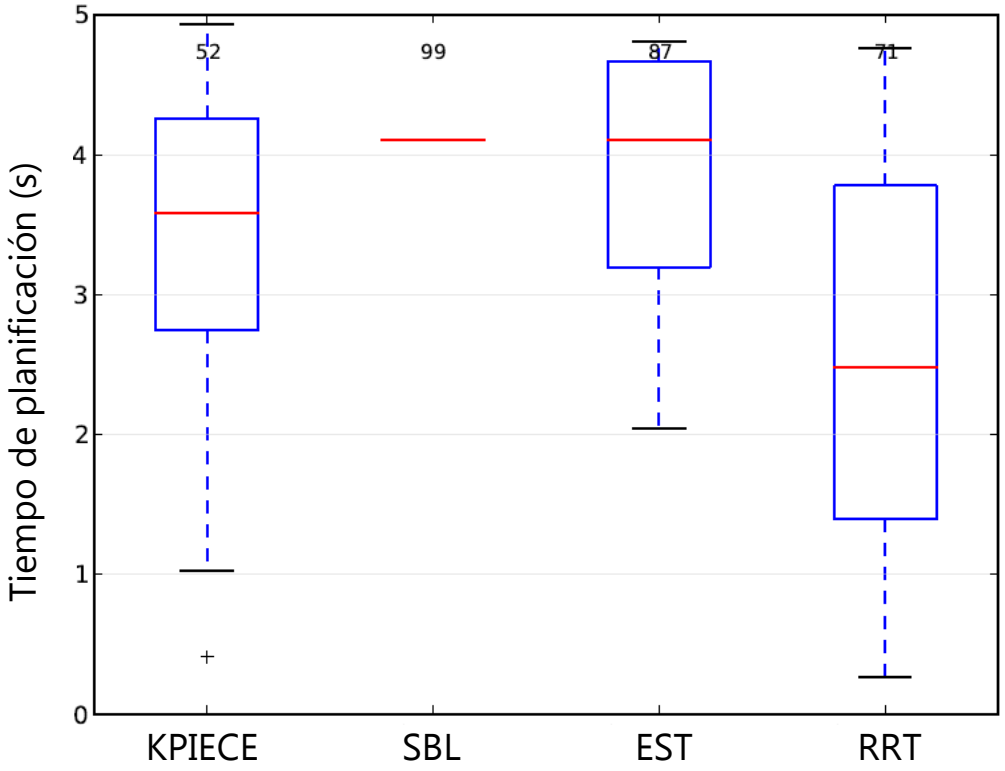

Figura 57. Tiempo de planificación en la prueba Industrial

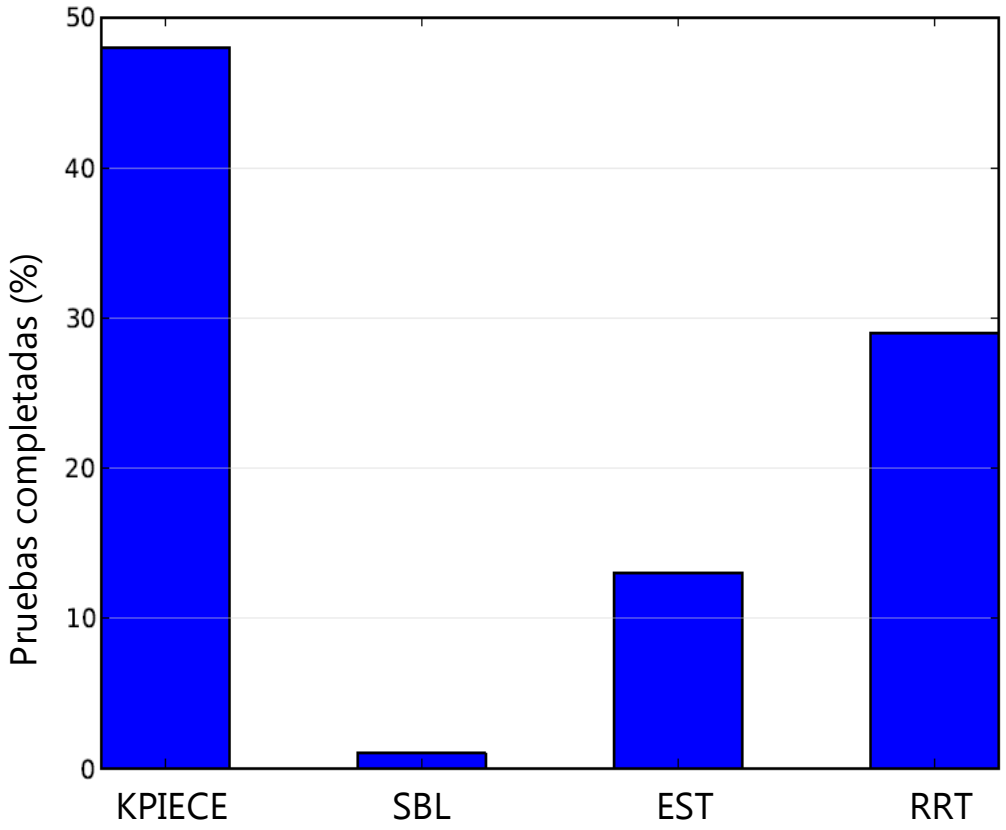

Figura 58. Porcentaje de éxito en la prueba Industrial

El planificador SBL, el más ineficaz para el entorno *Industrial*, no ha podido realizar más que una prueba dentro del margen de tiempo proporcionado. Los

otros algoritmos, a pesar de tener porcentajes muy bajos de éxito, sí han conseguido encontrar algunas soluciones a tiempo. Comparando otra vez las ilustraciones 57 y 58 se ve que el planificador KPIECE, con muchas pruebas en el intervalo aceptable, llega casi a un 50% de éxito, el mayor.

En este punto, analizar las otras variables en los planificadores SBL, EST o RRT no tendría mucho sentido, pues los resultados no han sido satisfactorios. En su lugar se ha recalculado la planificación para la misma prueba con las variaciones de los planificadores KPIECE (figura 59).

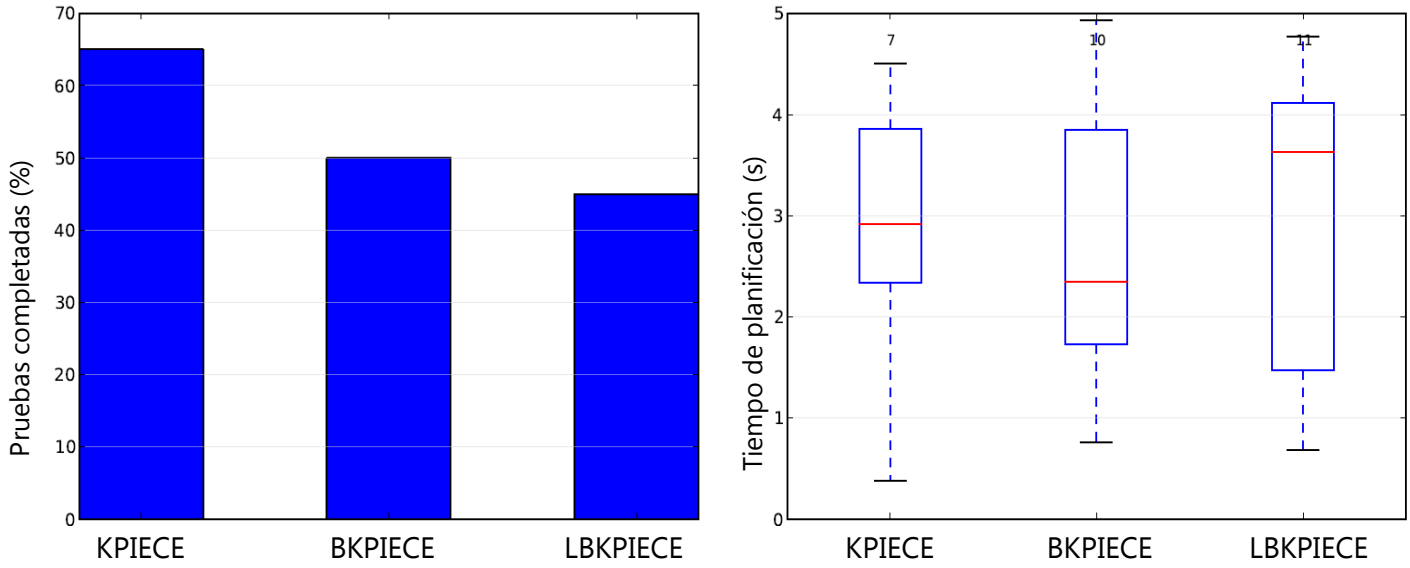

Figura 59. Pruebas completadas y tiempo de planificación en la comparativa KPIECE Industrial

Las variaciones de KPIECE no son mejores que el algoritmo original para este caso, y tienen más pruebas incompletas por superar el tiempo límite. Tampoco mejoran las variaciones en longitud de la trayectoria planificada (figura 60) o en holgura con objetos del entorno o del propio robot (figura 61).

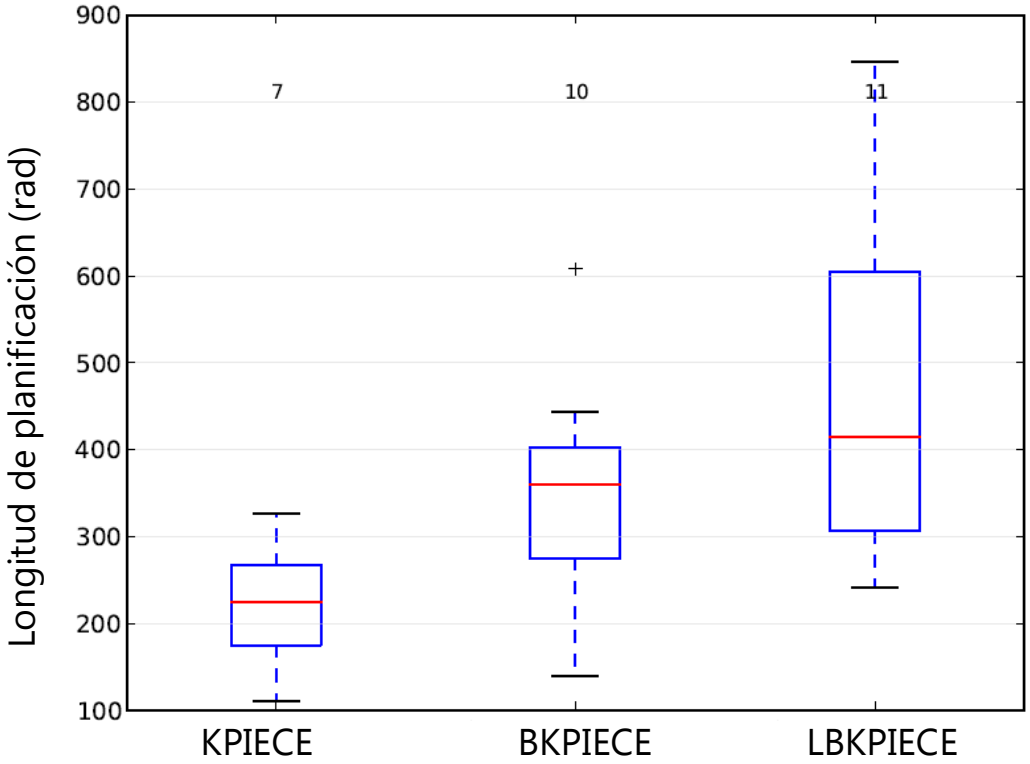

Figura 60. Longitud de la planificación en la comparativa KPIECE Industrial

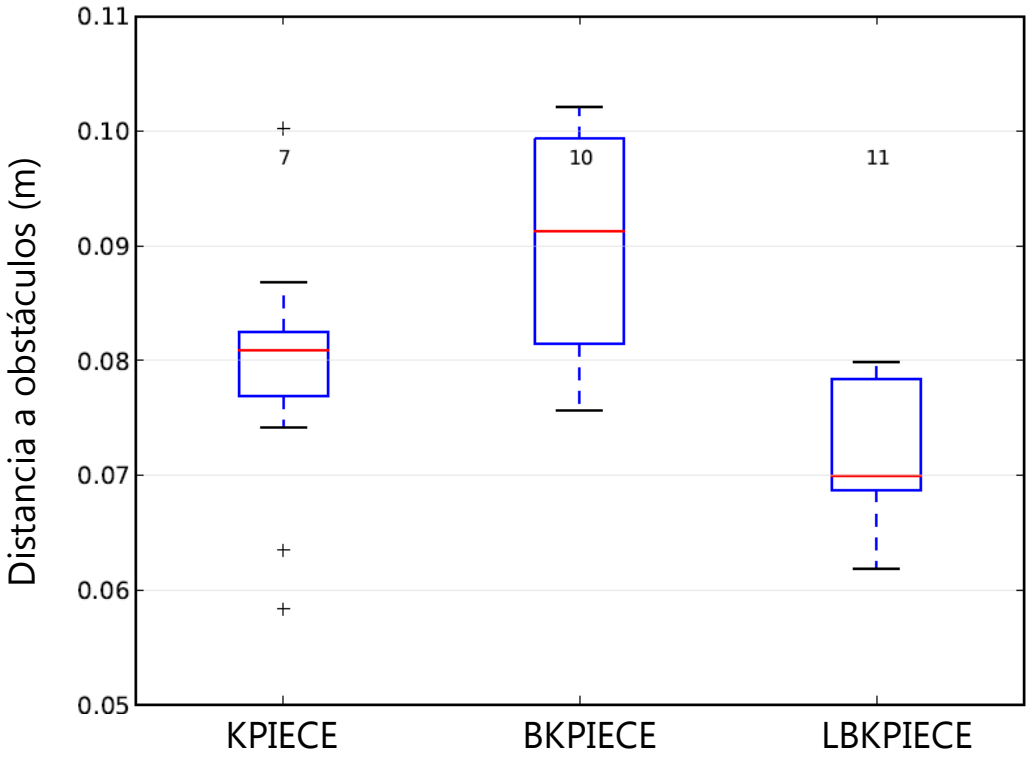

Figura 61. Distancia a obstáculos en la comparativa KPIECE Industrial

De esta comparación queda demostrado que el planificador original KPIECE es el más eficiente para la prueba *Industrial* diseñada.

- **Resultados globales**

Como se ha visto, es fácil observar las diferencias de cada planificador y relacionar sus comportamientos en los resultados de *benchmarking*. Incorporando a *MoveIt!* el escenario y los estados que se deseen para el robot, el usuario puede poner a prueba los planificadores y elegir el que más se adapte a sus necesidades.

En las pruebas de este trabajo se ha visto que hay planificadores más aptos que otros: por ejemplo, KPIECE ha funcionado bien en entornos con obstáculos y en PR2, mientras que RRT y RRT *Connect* lo han hecho en planificaciones con Manfred. La tabla 4 generaliza y resume los resultados para situaciones similares.

Tabla 4. Resumen del rendimiento de los planificadores

Rendimiento	KPIECE	SBL	EST	RRT
Movimiento sin obstáculos	↑↑ Longitud ↑↑ Suavidad ↓ Tiempo	↑ Longitud ↑↑ Suavidad ↓ Tiempo	↓ Longitud ↑ Suavidad ↓ Tiempo	↓ Longitud ↓ Suavidad ↓ Tiempo
Movimiento planar	*Benchmarking no realizable*			
Movimiento de brazo y mano	↑ Longitud ↑ Suavidad ↓↓ Tiempo	*No realizable*	↑↑ Longitud ↑ Suavidad ↑↑ Tiempo	↓↓ Longitud ↓ Suavidad ↓↓ Tiempo
Movimiento con obstáculos	↑ Éxito ↑ Tiempo	↓↓↓ Éxito ↑↑↑ Tiempo	↓ Éxito ↑ Tiempo	↓ Éxito ↑ Tiempo

Como inconveniente de la planificación que realiza *MoveIt!* están los casos ineficientes que puede plantear y que no se detectan en *benchmarking*. Por ejemplo, en la figura 62 se muestra una planificación correcta y calculada dentro de tiempo, pero que no aporta una trayectoria que pueda considerarse válida. Es un claro error de planificación, que en una gráfica con valores numéricos no puede apreciarse.

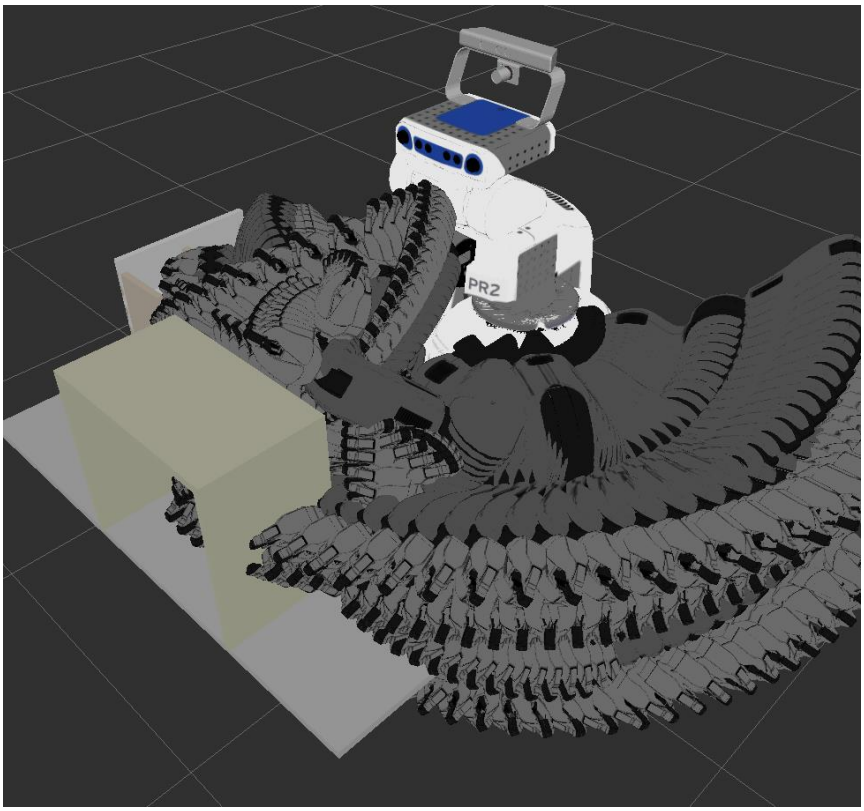

Figura 62. Ejemplo de planificación ineficiente

5.5. Problemas encontrados

En el desarrollo del trabajo han aparecido muchos problemas como errores o atascos en la ejecución de los procesos. Como parte del ecosistema de ROS se han recopilado los errores identificados para ayudar a otros usuarios mientras el desarrollo global avanza.

Los foros de preguntas de ROS o de usuarios de *MoveIt!* han sido las fuentes de consulta durante el proyecto. Uno de los inconvenientes del uso de Ubuntu es que hay varias versiones operativas y se actualizan semanalmente cambiando todos los paquetes con mejoras de los desarrolladores. Por tanto, no tiene la misma robustez, estabilidad y compatibilidad de aplicaciones de otros sistemas operativos cerrados. Se ha trabajado con la versión de Ubuntu: *Precise 12.04*.

5.5.1. NÚMEROS DECIMALES EN LA FORMA ANGLOSAJONA

Como ha ocurrido muchas veces a lo largo de la Historia, la falta de consenso para emplear ciertas unidades o métodos ha provocado problemas de todo tipo y situaciones complicadas. Sucede lo mismo en *MoveIt!* cuando en el *Setup Assistant* se configuran poses para el robot: se guardan los ajustes siguiendo el modelo de la región del equipo, europeo en este caso (decimales con coma), pero para sus aplicaciones se leen en modelo americano (decimales con punto).

Genera problemas cuando alguna aplicación trata de acceder a esta información, por ejemplo, cuando RViz importa el paquete del robot para su visualización. La figura 63 sería parte de un archivo de *srdf* mal redactado por el *Setup Assistant*, y la figura 5 de uno correcto.

```
−<group_state name="home_right_arm" group="right_arm">
    <joint name="r_elbow_flex_joint" value="-2,0881"/>
    <joint name="r_forearm_roll_joint" value="0"/>
    <joint name="r_shoulder_lift_joint" value="1,2963"/>
    <joint name="r_shoulder_pan_joint" value="0"/>
    <joint name="r_upper_arm_roll_joint" value="0"/>
    <joint name="r_wrist_flex_joint" value="0"/>
    <joint name="r_wrist_roll_joint" value="0"/>
  </group_state>
−<group_state name="home_left_arm" group="left_arm">
    <joint name="l_elbow_flex_joint" value="-1,8334"/>
    <joint name="l_forearm_roll_joint" value="0"/>
    <joint name="l_shoulder_lift_joint" value="1,1943"/>
    <joint name="l_shoulder_pan_joint" value="1,6044"/>
    <joint name="l_upper_arm_roll_joint" value="0"/>
    <joint name="l_wrist_flex_joint" value="0"/>
    <joint name="l_wrist_roll_joint" value="0"/>
  </group_state>
```

Figura 63. Archivo descriptivo SRDF con números decimales europeos

5.5.2. ERROR 48 DE LA BASE DE DATOS

Este error se ha podido identificar pero no solucionar. Aparece en la ejecución de *benchmarking*, pero no se sabe si afecta a algún parámetro. En cuanto a lo que concierne a este proyecto, el error 48 ha aparecido en todas las ejecuciones de *benchmarks* pero no ha alterado nada.

Figura 64. Error 48 en la base de datos

Se trata de un problema ubicado en la base de datos *warehouse,* que en las notificaciones de ROS se ejecuta como *Mongo process* (figura 64), aunque *benchmarking* acceda correctamente y se obtengan los escenarios y *queries*.

5.5.3. MARCADORES INTERACTIVOS EN RVIZ

Las esferas de color azul que el usuario puede desplazar en el espacio de RViz para situar posiciones de elementos del robot son los marcadores interactivos. Estos marcadores no aparecen siempre y es debido a errores o a situaciones inexplicadas. La única que se ha podido comprobar en este trabajo y en foros de soporte es la relativa a los grupos de planificación.

Como se explica en la sección del *Setup Assistant* de *MoveIt!* hay varias formas de definir los grupos de planificación. El problema en este caso radica en uno de ellos: grupos de articulaciones (*joints group*). Por ejemplo, si un grupo de planificación se define con la selección de todas las articulaciones de un brazo del robot, el usuario seleccionaría una a una y sería un *joint group*. Pues con esta configuración, los marcadores no aparecerían.

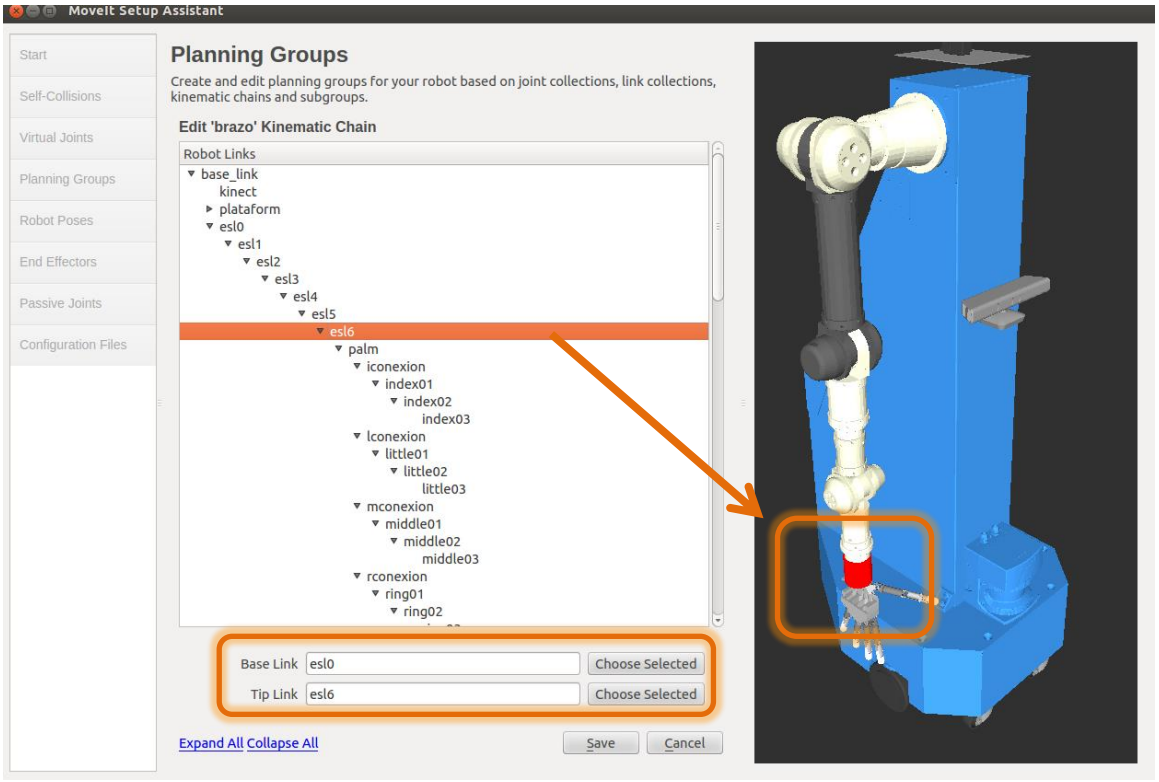

Figura 65. Árbol de articulaciones para configurar una cadena cinemática

Aunque conceptualmente todas las formas de agrupación son iguales, en la práctica hay diferencias. La elección correcta para evitar problemas de desaparición de marcadores sería la agrupación de tipo cadena cinemática (*kinematic chain*). La diferencia es que en una cadena se selecciona la articulación inicial y la final y agrupa automáticamente las articulaciones y enlaces entre ellas (figura 64). Otra diferencia es que así, y en muchos otros casos, se soluciona este inconveniente.

5.5.4. FALLO GRAVE QUE AFECTA A LA EJECUCIÓN DE BENCHMARKING

El fallo más llamativo es un error en una línea de código. Como *benchmarking* se encuentra en fase *alpha* de desarrollo, es normal encontrar errores graves de este tipo. En todas las instalaciones de paquetes *Debian* de *MoveIt!* aparece este problema, y hace que sea imposible la ejecución de la planificación y de los *benchmarks*.

Para solucionarlo se puede hacer una instalación fuente de los paquetes afectados. Se debe descargar el paquete afectado desde *GitHub*, el archivo en formato C++ correcto y construir con *catkin* el paquete. Después, hay que

sustituir los archivos de sistema o ejecutables generados, en cada fichero, por los del mismo nombre en la ubicación de los ficheros correspondientes del directorio de instalación de *MoveIt!* Es una operación poco correcta pero eficaz para poder ejecutar *benchmarks*.

5.5.5. AUSENCIA DEL EJECUTABLE DE ESTADÍSTICAS DE BENCHMARKING

Como en el problema anterior, la instalación normal de *MoveIt!* deja el archivo ejecutable que sirve para transformar el *output* de benchmarking en PDF sin instalar. En este caso, no es necesaria una instalación irregular sino que con descargar el archivo y ubicarlo en un paquete dentro de ROS es suficiente.

5.5.6. INCOMPATIBILIDAD DE BENCHMARKING CON MOVIMIENTOS PLANARES DE ARTICULACIONES VIRTUALES

En el apartado de resultados ya se ha explicado este problema, que ha incapacitado la posibilidad de utilizar *benchmarks* en planificaciones que incluían movimientos planares de articulaciones virtuales. No existen soluciones de momento.

6. Conclusiones y trabajos futuros

El trabajo realizado con ROS, *MoveIt!* y sus herramientas para planificación de trayectorias y benchmarking, ha demostrado la utilidad de estas aplicaciones. Sin embargo, necesita aún mucho desarrollo para poder convertirse en una opción a considerar. Durante el proyecto han aparecido demasiados obstáculos, fallos o lagunas de funcionalidad que hacen aún ineficiente *MoveIt!* para aplicaciones reales en la industria o domésticas, por ejemplo.

Muchos de los problemas vienen de errores de programación en el código de la fuente, y sustituirlos, modificarlos o entenderlos no es la función de un estudio de planificación de robótica. Debido al temprano estado de desarrollo también surgen fallos graves que impiden que el trabajo sea fluido. En las herramientas menos desarrolladas como *benchmarking* o *motion planning* los problemas son inexplicables la mayoría de las veces para el usuario.

Aunque en este trabajo hayan surgido muchos problemas, se han podido solucionar gracias a foros y temas on-line de dudas y ayuda. Y su propia finalidad es la de servir de pilar para futuros trabajos que empleen la planificación o *benchmarking*.

Las ventajas que tiene, en cambio, son mayores: consolidar un sistema de control y planificación de robótica libre y robusto, con ayuda y soporte colaborativos y documentación gratuita. Como se ha visto a lo largo de este proyecto, las funciones avanzadas de *MoveIt!* son una realidad y han conseguido aportar resultados satisfactorios en muchos casos. Por tanto, es más bien un proyecto enfocado al futuro que a un presente inmediato.

6.1. Interfaz gráfica durante el proceso

El avance de *MoveIt!* en ROS ha sido la posibilidad de que el usuario controle y visualice la planificación de trayectorias. Una mejora sería la incorporación de una interfaz gráfica que mostrara cada ejecución durante *benchmarking* al

mismo tiempo que ofrezca al usuario la posibilidad de detener, pausar o modificar la planificación, así como poder ver cada planificación al instante.

6.2. Mejora del rendimiento gráfico

La optimización del *plugin* de visualización en equipos de potencia gráfica media es insuficiente para las aplicaciones que *MoveIt!* Para la captura de imágenes de trazados de trayectorias con muchas texturas, como en las figuras 50 o 62, el *frame rate* baja demasiado y complica tanto las tareas en el plugin en el que se están procesando como en todo el equipo.

6.3. Aplicación para la edición de variables

Para realizar muchos procesos seguidos en diferentes escenarios o con diferentes condiciones se necesita cambiar en un editor de texto los valores de número de ejecuciones, planificadores,… Del mismo modo que se ha creado un asistente de configuración para los paquetes del robot, sería útil uno similar para modificar valores de *benchmarking* con aplicación automática del cambio.

6.4. Personalización de las gráficas

Las gráficas en PDF se transforman con el ejecutable tipo *Python*. Aunque es posible modificar el formato o los parámetros de las gráficas, es necesario conocer el lenguaje para hacerlo. Una ventana que ofrezca al usuario la opción de ajustar el rango de los ejes, la eliminación de datos atípicos o el color de las cajas, por ejemplo, ayudaría a obtener resultados personalizados.

7. Referencias

[1] Ioan A. Sucan and Sachin Chitta, *MoveIt!* [En línea] Disponible: http://moveit.ros.org

[2] *What is ROS?* [En línea] Disponible: http://www.ros.org/about-ros/

[3] Ioan A. Şucan, Mark Moll, Lydia E. Kavraki, *The Open Motion Planning Library*, *IEEE Robotics & Automation Magazine,* 19(4):72–82, December 2012. [En línea] Disponible: http://ompl.kavrakilab.org

[4] Benjamin Cohen, Ioan A. Sucan, Sachin Chitta, *A Generic Infrastructure for Benchmarking Motion Planners paper*

[5] Kavraki Lab, *How to benchmark OMPL* [En línea] Disponible: http://ompl.kavrakilab.org/benchmark.html

[6] ROS.org, *OMPL*, [En línea] Disponible: http://www.ros.org/news/2011/01/open-motion-planning-library-ompl-released.html

[7] Willow Garage, *Robot PR2 overview*, [En línea] Disponible: http://www.willowgarage.com/pages/pr2/overview

[8] Wikipedia, *Benchmarking*, [En línea] Disponible: http://es.wikipedia.org/wiki/Benchmarking

[9] *What is free software?*, The Free Software Foundation, [En línea] Disponible: http://www.fsf.org/about/what-is-free-software

[10] Raul Villajos Rayo, Javier V. Gómez, *Integración de un sensor laser 3d en el manipulador movil Manfred, 2013*.

[11] Roboticlab UC3M, *Manfred, el robot mayordomo 100% español*, [En línea] Disponible: http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/actualidad/reportajes/archivo_reportajes/Manfred_robot_mayordomo_espanol

8. Anexos

A. Guía para Benchmarking

Nota: *Esta guía está basada en la original de la web de MoveIt! con observaciones y correcciones. (Link: http://moveit.ros.org/wiki/Benchmarking)*

ORGANIZACIÓN DE PAQUETES DE ROS

Los paquetes de ROS que van a ser útiles para *benchmarking* son los creados por el usuario con el asistente de configuración.

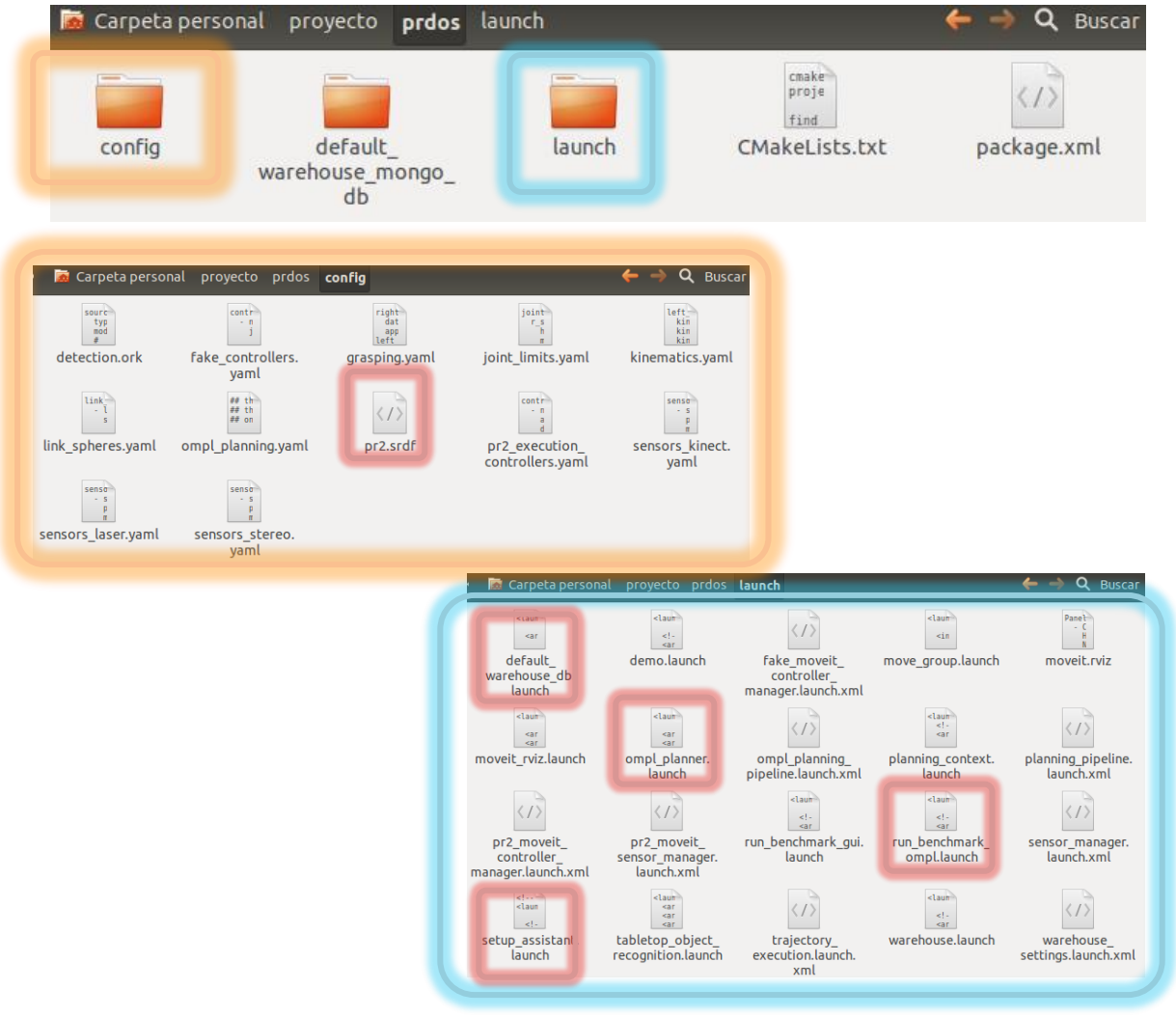

Figura 66. Ficheros en el paquete del robot

En la figura 66 se describe un ejemplo de un paquete de ROS organizado en el directorio de Ubuntu. Se puede acceder a los archivos y modificarlos. Están enmarcados algunos de los archivos más importantes y empleados en este trabajo. Por ejemplo, los cambios que el usuario realice en RViz, como el tamaño de las ventanas, el color de las texturas del robot,… se guardan en ese archivo.

Habrá que editar *run_benchmark_ompl.launch* para indicar la ubicación del archivo de configuración, pues este es el "lanzador" de los *benchmarks* con OMPL (figura 67).

run_benchmark_ompl.launch

```
<launch>

  <!-- This argument must specify the list of .cfg files to process for benchmarking -->
  <arg name="cfg" value="/home/miguel/proyecto/config.cfg"/>

  <!-- Load URDF -->
  <include file="$(find prdos)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <!-- Start the database -->
  <include file="$(find prdos)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path" value="moveit_ompl_benchmark_warehouse"/>
  </include>

  <!-- Start Benchmark Executable -->
  <node name="$(anon moveit_benchmark)" pkg="moveit_ros_benchmarks" type="moveit_run_benchmark"
args="$(arg cfg) --benchmark-planners" respawn="false" output="screen">
    <rosparam command="load" file="$(find prdos)/config/kinematics.yaml"/>
    <rosparam command="load" file="$(find prdos)/config/ompl_planning.yaml"/>
  </node>

</launch>
```

Ubicación del argumento

Figura 67. Archivo de benchmarks de OMPL

FUNCIONAMIENTO DE ROS

Para benchmarking las dos instrucciones de ROS más frecuentes que se utilizan son *rosrun* y *roslaunch*.

Rosrun: lanza archivos ejecutables (por ejemplo, archivos escritos en *phyton*).

```
rosrun nombre_paquete ejecutable.py
```

Roslaunch: ejecuta archivos de tipo *launch*.

```
roslaunch nombre_paquete archivo.launch
```

PASOS PARA BENCHMARKING

Los siguientes pasos sólo son posibles en el sistema operativo Ubuntu (el único que tiene soporte) con *ROS* y *MoveIt!* instalados. En las páginas webs correspondientes hay guías para la instalación completa.

1. *Configurar el robot*

El robot se configura desde el asistente de configuración. Para ejecutarlo se lanza la orden:

```
roslaunch moveit_setup_assistant setup_assistant.launch
```

Desde ahí se carga el archivo *urdf* del robot que se desee crear como paquete de ROS.

2. *Importar scene y queries en warehouse*

Los *queries* y escenas se almacenan en la base de datos, por tanto en primer lugar hay que ejecutar el servidor *warehouse*.

```
roslaunch ROBOT default_warehouse_db.launch
```

Nota: *Si el archivo ha sido editado antes y contiene datos en warehouse, esta orden no tendrá resultado. Deberá indicarse una raíz alternativa:*

```
roslaunch ROBOT_moveit_config warehouse.launch
moveit_warehouse_database_path:=~/moveit_db
```

Por último, se introduce con las acciones correspondientes los archivos *scene* o *queries* en la base de datos.

```
rosrun moveit_ros_warehouse moveit_warehouse_import_from_text --host
127.0.0.1 --port 33829 --scene NOMBRE.scene
```

```
rosrun moveit_ros_warehouse moveit_warehouse_import_from_text --host
127.0.0.1 --port 33829 --queries NOMBRE.queries
```

Nota: *se debe situar antes el terminal en la ubicación en la que se han descargado editado los archivos de texto en formato scene o queries. En el caso de que no se encuentre el archivo especificado, no se produce ningún mensaje de error y sólo se comprueba desde RViz.*

3. Asociar los *queries* a un escenario

Una vez importados a la base de datos, es necesario asociar cada *query* a un escenario. Para ello, se utiliza RViz. Aunque lo ejecuta automáticamente, es recomendable cargar la descripción del robot antes:

```
roslaunch ROBOT planning_context.launch load_robot_description:=true
```

Y finalmente RViz:

```
roslaunch ROBOT demo.launch
```

Desde RViz hay que conectarse a *warehouse* para poder acceder tanto a *queries* como escenas en las correspondientes pestañas (figura 9). Desde ahí pueden leerse los *queries* importados, o crear unos nuevos, para asociarlos a cada archivo *scene* guardado o creado.

Nota: *En el caso de que el usuario no defina un estado inicial, se asocia la posición por defecto indicada en la descripción del robot.*

4. Ejecutar los *benchmarks*

Con estados y un escenario, el último paso es la ejecución de los *benchmarks*.

```
roslaunch ROBOT run_benchmark_ompl.launch
```

Como se ha indicado antes, este archivo ejecuta los *benchmarks,* pero requiere un archivo de configuración en el formato de la figura 25. Se indica, según la figura 67, en el archivo en el paquete del robot la ubicación del archivo de configuración.

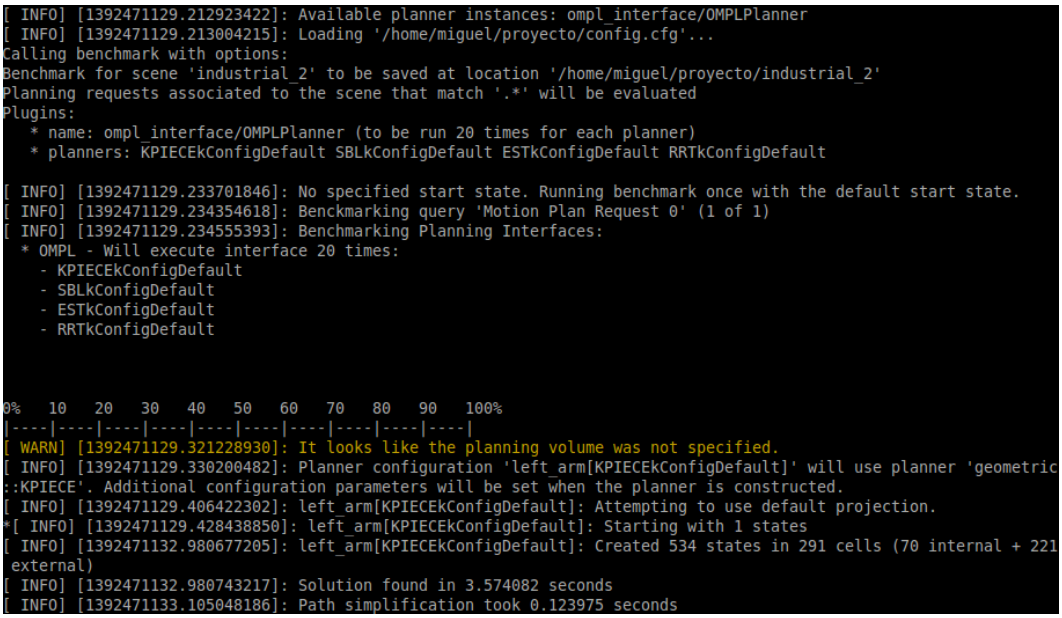

Figura 68. Proceso de benchmarking en ejecución

La figura 68 es una captura de la cabecera de notificación del proceso de *benchmarking*, y muestra la información incluida en el archivo de configuración. En caso de no producirse la ejecución, es muy probable que sea debido a algún error de los mencionados en el apartado de "Problemas encontrados".

5. *Logging* y transformación a PDF

Una vez ejecutado, se devuelve un archivo de notas tipo *log*, como se muestra en la figura 69.

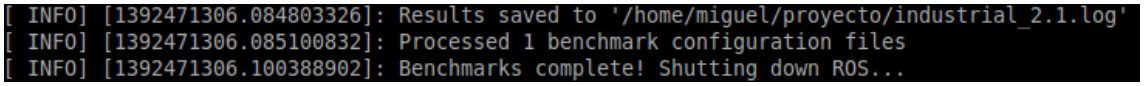

Figura 69. Logging posterior al proceso de benchmarking

El último paso, por el que se obtendrían los datos de *logging* en formato PDF se ejecuta con la orden:

```
rosrun moveit_ros_benchmarks moveit_benchmark_statistics.py NOMBRE.1.log -p
NOMBRE_RESULTADOS.pdf
```

Nota: *como para importar archivos, es necesario ubicar la terminal donde se ha generado el archivo log.*

B. Código desde *GitHub*

- Archivo inexistente en la instalación: *moveit_benchmark_statistics.py*. Link: https://github.com/ros-planning/moveit_ros/blob/hydro-devel/benchmarks/benchmarks/scripts/moveit_benchmark_statistics.py
- Paquete para construcción correcta en catkin: *ompl*. Link: https://github.com/ros-planning/moveit_planners/tree/hydro-devel/ompl
- Archivo incorrecto para sustituir en el paquete *ompl* y poder construirlo: *model_based_planning_context.cpp*. Link: https://09460855191131317480.googlegroups.com/attach/6a4ed45abd48afa1/model_based_planning_context.cpp?part=4&view=1&vt=ANaJVrFLzFTY_KSIBNaNMS8aOHmFG00g5aUdVcpXuRPKGNh0ORE0cgXQJFtiWlFvDyqZV7VntVyvL42M88xbW_VAYT-kaOc4sTCCpiXEzHGkvFHSTDrmyDQ