Universidad Carlos III de Madrid
www.uc3m.es

Departamento de Ingeniería de Sistemas y Automática

Grado en Ingeniería en Tecnologías Industriales

Trabajo Fin de Grado

ARQUITECTURA PARA EL AGARRE DE OBJETOS EN MOVEIT!

Autor: Nadia Ailén Ferreyra Parrilla

Directores: David Álvarez Sánchez

Javier Victorio Gómez González

Leganés, Madrid, septiembre 2016

Resumen

Hoy en día, las investigaciones en el ámbito de la robótica están en auge, ya que se pretende aumentar la calidad de vida del ser humano. Una de las aplicaciones que cobra importancia en ese aspecto es el brazo robótico, cuya finalidad es manipular los objetos que lo rodean.

Manfred es un robot manipulador móvil, desarrollado por el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid. Consta de un brazo robótico situado sobre una base móvil, diseñado para agarrar objetos y poder moverse por ambientes interiores.

Este proyecto se basa en la necesidad de mejorar esa interacción entre Manfred y sus alrededores, por lo que se requiere mejorar su capacidad de percibir el entorno. Además, para ser capaz de manipular los elementos de su alrededor, se necesita estudiar la arquitectura para el agarre de los mismos. Por tanto, este proyecto está orientado a desarrollar un software que permita al robot ver y entender el entorno, así como planificar el movimiento del brazo de Manfred para que se aproxime al objeto que se quiere agarrar.

Palabras clave: MoveIt!, ROS, PCL, Octomap, filtrado, segmentación, planificación de trayectoria, Manfred.

Abstract

Nowadays, research in robotics is booming, because the objective of that is to increase the quality of human life. The robotic arm is one application that becomes important in this way, since it allows to manipulate the objects around it.

Manfred is a mobile manipulator robot developed by the Systems Engineering and Automation department of the Carlos III University of Madrid. It consists of a robotic arm located on a mobile base, and it is designed to grip objects and to move around indoors.

This project is based on the need to improve the interaction between Manfred and its surroundings, which requires improving its ability to perceive the environment. In addition, it is necessary to study the architecture for gripping to be able to manipulate the elements around. Therefore, this project is oriented to develop a software that allows the robot to see and understand the environment and plan the approaching movement of Manfred arm to grab an object.

Key words: MoveIt!, ROS, PCL, Octomap, filtering, segmentation, planning trajectory, Manfred.

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores por su paciencia y por todo lo que me han enseñado en general y de ROS en particular.

En segundo lugar, dar las gracias a mi familia, mi pareja, mis amigos, mis compañeros de universidad y del trabajo y a todos aquellos que me han estado apoyando a lo largo de este tiempo. Y pedir disculpas si durante esta época he estado un poco ausente.

Finalmente, agradecer en especial a mi madre, porque sin ella no hubiera podido estudiar en la universidad y no sería quién soy hoy en día.

ÍNDICE

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Descripción general

En la actualidad, los avances de la robótica están muy presentes. Nuevas tecnologías y desarrollos hacen que la robótica evolucione cada vez más rápido. No es casualidad que esto suceda, ya que se pretende evitar tanto trabajos que requieran un sobreesfuerzo como aquellos que suponen un riesgo para el ser humano o incluso que resulten tediosos. Por tanto, el objetivo principal de estos avances es mejorar la calidad de vida del ser humano, así como mejorar la productividad de las empresas aumentando la velocidad de producción o abaratando costes a medio-largo plazo. Por todo ello, gran parte de las investigaciones se centran en este ámbito.

Una clase de robot frecuente en los desarrollos e investigaciones actuales es el brazo o mano robótica. Existen numerosas utilidades que se le pueden otorgar a un brazo robótico, lo que supondrá una variación en la tecnología utilizada en cada caso, tanto en el hardware como en el software. De esta forma, no se diseñará del mismo modo un brazo preparado para trabajar en una cadena de producción como uno capacitado para sujetar cualquier objeto que se presente de un tamaño razonable. En el primer caso, probablemente sea suficiente con dotarlo con un sensor para detectar la pieza y unos motores para realizar los movimientos pertinentes. El segundo caso sería bastante más complejo, ya que no sería suficiente con detectar el objeto, si no que se requería de algún dispositivo capaz de obtener información 3D del entorno para poder conseguir la mayor información del objeto en cuestión, ya que a priori sería desconocido. Este último caso es muy estudiado aplicado a los robots autónomos en general, ya que es necesario este estudio del entorno para la toma de decisiones por parte del propio robot.

1.2. Motivación

El robot en el que se ha basado en este proyecto es Manfred (*MAN FRiEnDly mobile manipulator*). Manfed es un robot desarrollado por el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid [1].

Se trata de un robot manipulador móvil diseñado para tratar con objetos, agarrar cosas y

poder moverse por entornos interiores. Su principal finalidad es la de servir como plataforma experimental para el área de robots móviles. Consta de una base móvil, el torso y un brazo articulado de fibra de carbono que se extiende hasta la mano robótica (véase Figura 1). Además, dispone de una cámara RGB-D para captar la información del entorno.

Figura 1. Partes del robot Manfred.

Desde sus comienzos hasta la fecha, se han desarrollado numerosos proyectos con el fin de implementarlos en Manfred para incrementar sus capacidades de reacción con el entorno. Estos proyectos del ámbito de la investigación, abarcan desde la parte electromecánica del conjunto robótico brazo-mano hasta la parte del software orientado a la percepción del exterior para lograr una interacción viable con el entorno.

En lo relativo al nivel de control, supone un problema conseguir una fiabilidad alta en cuanto a percepción del entorno derivado de la incertidumbre que supone tratar con entornos reales [2]. Por ello, es importante mejorar los algoritmos de percepción, navegación y planificación del robot, aumentando la robustez del software implementado para cada tarea.

Este proyecto está, por tanto, orientado a mejorar las capacidades de Manfred en cuanto a percepción y planificación de trayectorias para lograr mayor adaptabilidad con respecto al entorno.

1.3. Objetivos

El presente proyecto consiste en la implementación de un software capaz de identificar el número de objetos dispuestos sobre una mesa y sus posiciones, con el fin de calcular el movimiento que tiene que realizar el brazo del robot para que la mano alcance uno de esos objetos. No se pretende determinar cómo agarrar el objeto, sino determinar el movimiento de aproximación que supone realizar dicha acción.

Para llevar a cabo esa funcionalidad, se han establecido tres etapas claramente diferenciadas:

- En la primera fase se realiza una captación de los elementos del entorno a través de un sensor RGB-D, captando la información en forma de nube de puntos.

- En la segunda fase se trata de distinguir cuántos objetos hay en esa grabación y dónde están situados, con el fin de poder designar uno de ellos como objetivo de agarre por parte del robot. La nube de puntos, que representa los objetos existentes, proporciona la información suficiente para tratar dichos objetos por separado aplicando filtros a través de la librería PCL (Point Cloud Library) y visualizándolo en el plugin Rviz de ROS (Robot Operating System).

- En la tercera fase, se pretende configurar el entorno en MoveIt! para que Manfred pueda, a partir de la información obtenida del previo procesamiento de la nube de puntos, planificar una trayectoria adecuada para el agarre de los objetos. Para que el robot pueda interactuar con este entorno, es necesario aplicar las librerías de Octomap con el fin de modularlo como una rejilla 3D, lo que facilita la planificación de las trayectorias.

Para lograr esos objetivos, se requiere disponer de ciertos conocimientos para aplicar la librería PCL así como conocer los fundamentos principales de la comunicación entre los nodos configurados en ROS, a través de los cuales podremos visualizar finalmente el conjunto en MoveIt! Además, para el desarrollo del programa necesario se ha recurrido al lenguaje de programación C++ por ser uno de los soportados por las herramientas

utilizadas y por tratarse de un lenguaje muy extendido.

1.4. Estructura del documento

La organización de este documento se ha llevado a cabo de manera escalonada, es decir, partiendo de los conocimientos más básicos relacionados con el proyecto hasta la resolución obtenida tras la realización del mismo.

En primera instancia, se establecerá una visión global sobre el problema al que se enfrenta el robot, proporcionando unas bases sobre los conceptos más importantes aplicados en este proyecto, así como diferentes alternativas en cuanto a las herramientas a utilizar y el motivo de la elección final. En segundo lugar, se explicarán los principales conceptos sobre las tecnologías utilizadas para entender mejor su implementación en este proyecto. Y, por último, se detallarán las fases desarrolladas para cumplir con los objetivos establecidos, aplicando las herramientas anteriormente especificadas.

2. PLANTEAMIENTO DEL PROBLEMA

El problema que se plantea en este proyecto consiste en conseguir unas bases para que el robot sea capaz de interactuar con los elementos de su alrededor. Para este caso en particular, el robot se enfrenta a una mesa con una serie de objetos encima, como puede ser la representada en la Figura 2, con el propósito de obtener una trayectoria de aproximación para que el brazo agarre uno de los objetos. Por tanto, tiene que ser capaz de ver y entender el entorno, para posteriormente poder planificar dicha trayectoria.

Figura 2. Escenario al que se enfrentaría el robot: mesa con objetos encima.

Para enfrentarse al problema, es necesario disponer de unos conocimientos básicos sobre percepción 3D, procesamiento de los datos del entorno y planificación de trayectorias. En adición, se propondrán diferentes herramientas para la aplicación en cada uno de los ámbitos y el motivo de la elección final para su uso en este proyecto.

2.1. Percepción de objetos

La percepción es la organización e interpretación de los estímulos sensoriales con el objetivo de representar y entender el entorno. Algo que puede parecer tan intuitivo, ha sido motivo de estudios durante varias décadas.

Tradicionalmente, los campos de investigación relacionados con la percepción están

organizados según los sentidos clásicos: visión, audición, tacto, olfato y gusto. A lo largo de los años, se ha ido incrementando la lista completa de sentidos relacionados con la percepción, englobando más posibilidades como la percepción del dolor o la percepción del tiempo.

La capacidad de un robot para realizar tareas complejas, adaptarse a posibles variaciones de su entorno y afrontar situaciones imprevistas, viene condicionada en gran parte por el sistema de percepción del que esté dotado. Sin embargo, a la hora de implementar sistemas de percepción en robótica, puede resultar muy complejo el desarrollo de cualquiera de ellos, por lo que es importante definir previamente la finalidad del mismo. Este es un motivo por el cual la percepción artificial hoy en día se considera uno de los aspectos más complicados en la robótica. Estas complicaciones vienen determinadas por algunas limitaciones como el tiempo que lleva procesar toda la información, costes para llevarlo a cabo o conseguir alcanzar una solución simple y a la vez fiable.

En el ámbito de la electrónica, estos sistemas de percepción corresponden con los sensores del robot. Entre los diversos sistemas de percepción posibles, los de rango de aplicación más amplio son los de percepción visual y los de percepción táctil, es decir, sensores de distancia y sensores de contacto respectivamente. A grandes rasgos, el primer grupo proporcionaría la información suficiente para guiar al robot hacia un elemento en concreto, mientras que el sensor de contacto facilitaría el agarre de la pieza. El sistema de percepción en el que está basado este proyecto es la visual, lo que se conoce como visión.

Los grandes avances de la robótica en la década de los 70 trajeron consigo las cámaras de vídeo y los microprocesadores, lo que hizo que, junto con la evolución de la computación sea factible la visión artificial [3]. La visión artificial pretende capturar la información del entorno físico para extraer características relevantes visuales, utilizando procedimientos automáticos.

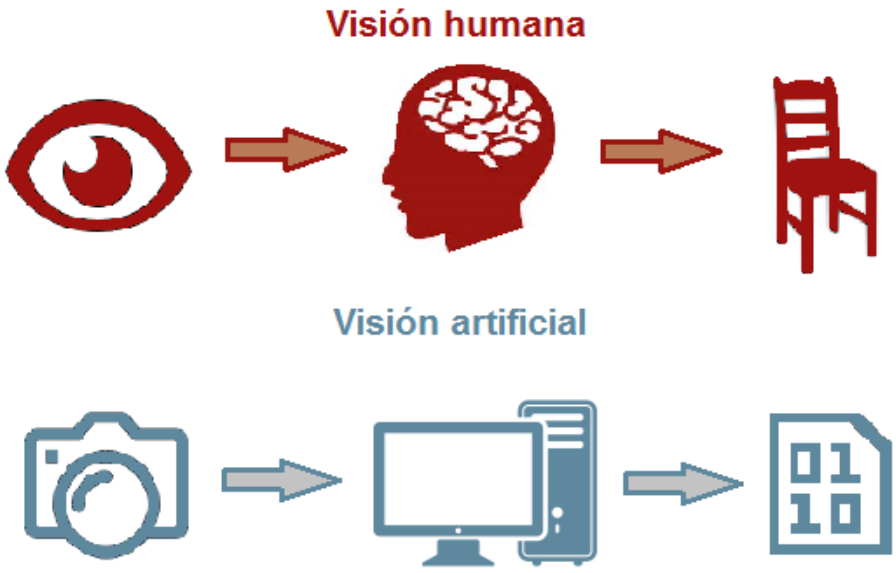

Figura 3. Visión humana vs Visión artificial.

Tanto en seres humanos como en otras especies de seres vivos, el sentido de la vista es el más desarrollado y el que más información nos proporciona del entorno. Sin embargo, se trata de una actividad inconsciente, por lo que resulta complicado descifrar exactamente cómo se produce, motivo por el cual es un ámbito muy estudiado en robótica. Para poder tener unas bases sólidas en el sector de la visión computacional sería idóneo disponer de más conocimientos sobre el sentido de la vista de los humanos. Aun así, actualmente existen varias tecnologías que facilitan el desarrollo de este tipo de percepción, lo que ayuda a que surjan cada vez más avances y aplicaciones en ese sector.

Para comprender mejor cómo surgen esos desarrollos, es necesario distinguir los dos pilares de sistema de visión artificial: el sistema de formación de las imágenes y el sistema de procesamiento de éstas. El primer apartado estaría constituido por la captación de las imágenes del entorno. Una vez introducida dicha información en el ordenador, ésta es procesada mediante algoritmos para transformarla en datos de alto nivel, los cuales pueden ser utilizados para su representación visual o para actuar en el planificador de un robot, entre otros.

2.1.1. Captación de información del entorno

La captación de la información sensorial es fundamental para el reconocimiento de formas u objetos. Hoy en día, existen algunas alternativas en cuanto a su desarrollo. Dependiendo de la finalidad que se le va a dar al robot, será conveniente utilizar unas técnicas u otras. A grandes rasgos, se pueden clasificar estas técnicas en dos grandes grupos: visión 2D y visión 3D.

La visión 2D consiste en el estudio en dos dimensiones (X, Y) de objetos a partir de imágenes captada con cámaras digitales. En función de características como la calidad de imagen, luminosidad y la facilidad para ser procesada, puede proporcionar información fiable sobre los objetos a estudiar a partir de una identificación sin contacto directo. Este tratamiento de la imagen se realizaría a través de algoritmos que determinen el contorno de los elementos de la misma o a través de la variación del contraste. Una aplicación con visión 2D muy extendida es la mostrada en la Figura 4, que se corresponde con la inspección de piezas. De este modo, autómatas especializados en la inspección están dotados de sistemas de visión 2D para reconocer defectos que hayan podido surgir en una pieza durante una cadena de producción. Esto resulta muy útil para garantizar la fiabilidad del producto final. Otras aplicaciones posibles pueden ser la lectura de códigos de barra y reconocimiento de personas en una habitación, entre otras.

Figura 4. Inspección 2D de un neumático.

En cuanto a la visión 3D se puede decir que es la técnica que proporciona más información, pero, a su vez, es la que necesita mayor tiempo y complejidad de procesamiento. De un modo similar a la visión en 2D, ésta consiste en el estudio de los elementos del entorno a

partir de imágenes en tres dimensiones (X, Y, Z). Como cabe esperar, su procesamiento puede resultar bastante complejo dependiendo del objetivo que se quiere alcanzar. Por tanto, para que un robot pueda detectar cambios en su entorno físico y adaptarse en consecuencia, se aplica la visión por ordenador en 3D. Ésta permite aumentar la flexibilidad y la utilidad que se le otorga a los robots, por lo que se considera una herramienta que da apoyo a nuevas aplicaciones hoy en día, tales como la robótica de visión guiada o reconocimientos de objetos. Debido a que este proyecto se basa en la planificación de un robot hacia un objeto, se ha utilizado técnicas de visión 3D.

La obtención de esta información 3D se suele realizar con la ayuda de nubes de puntos, como se representa en la Figura 5. Las nubes de puntos son modelos digitalizados de la forma y ubicación de los elementos representado con vértices en un sistema de coordenadas tridimensional. Esto significa que se requerirá de algún dispositivo que proporcione este conjunto de puntos representativos de la escena a tratar.

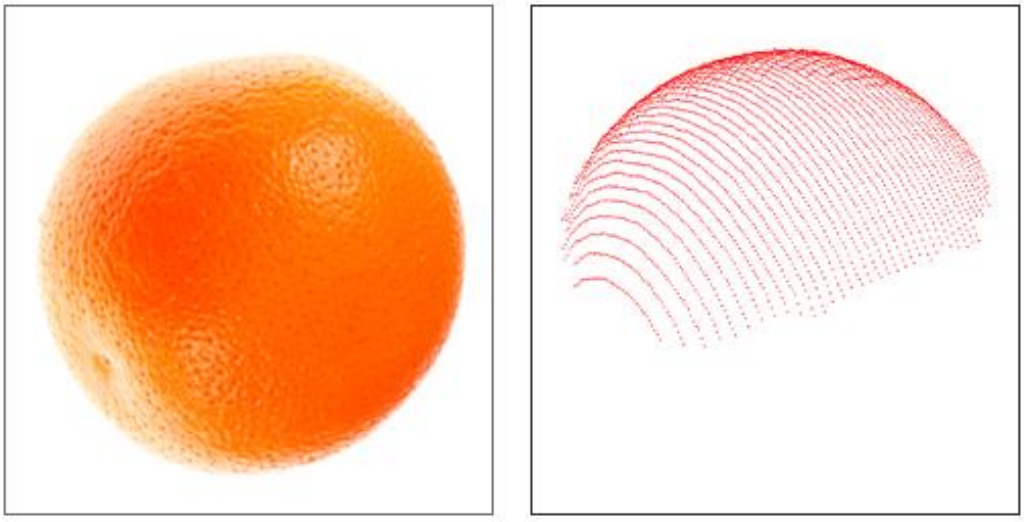

Figura 5. Nube de puntos de una naranja.

Actualmente existen diversas tecnologías capaces de proporcionar información del entorno en 3D, que se clasifican en función de si hay contacto con los objetos o no [4] [5]. En caso de necesitar contacto, se trata de escáneres que examinan el objeto apoyando el elemento de medida (palpador) sobre la superficie del mismo y, a través de unos sensores internos se obtiene la posición espacial del palpador. Suelen emplearse mayormente en controles dimensionales de piezas en procesos de fabricación, ya que son muy precisos. Como

inconveniente, destacar que el contacto con las piezas puede dañarlas si son frágiles y, a su vez, hace que resulte un sistema poco flexible y adaptable a cualquier entorno 3D.

Por el contrario, se encuentran los escáneres sin contacto que abarcan un mayor rango de posibles aplicaciones. Aun así, cabe distinguir dos grandes grupos:

- Activos, que se basan en emitir algún tipo de señal con el fin de analizar su retorno para capturar las geometrías del entorno.
- Pasivos, que se basan en detectar la radiación reflejada del ambiente sin emitir ninguna señal. Éstos suelen ser menos precisos con los activos.

Dentro del escaneo activo existen diversos métodos que sirven para obtener la información del entorno 3D. Entre ellos se encuentran: tiempo de vuelo, triangulación láser y luz estructurada.

Los escáneres de tiempo de vuelo (*time of flight*) utilizan un pulso láser que es emitido a través de un espejo rotatorio y a partir del cual se determina el tiempo que tarda en recorrer el trayecto de ida y de retorno al escáner. Por tanto, la certeza de este tipo de dispositivos depende de la precisión con la que se puede medir el tiempo, que en este caso es del orden de picosegundos, lo que significa que son equipos de alta precisión (submilimétrica).

Algunos ejemplos de dispositivos basados en esta tecnología son Callidus CP3200 o la correspondiente con la Figura 6, Leica ScanStation 2 y su aplicación suele estar destinada para trabajos de alta precisión en monumentos o elementos constructivos con el fin de analizar deformaciones en el terreno. Debido a su alto coste y al estar orientado a distancias largas, no es la tecnología más adecuada para la adquisición de datos de entornos interiores.

Figura 6. ScanStation2 de la empresa Leica.

Las técnicas de triangulación láser para escáneres 3D se basan en determinar la forma y posición de un objeto emitiendo una luz láser. La luz láser incide en el objeto y se usa una cámara para determinar la ubicación del punto. Esta técnica se llama triangulación porque el punto donde llega el láser, la cámara y el emisor del láser forman un triángulo. Dependiendo de la distancia a la que se encuentre el objeto, el láser llegará a posiciones diferentes de la cámara, como se puede observar en la Figura 7.

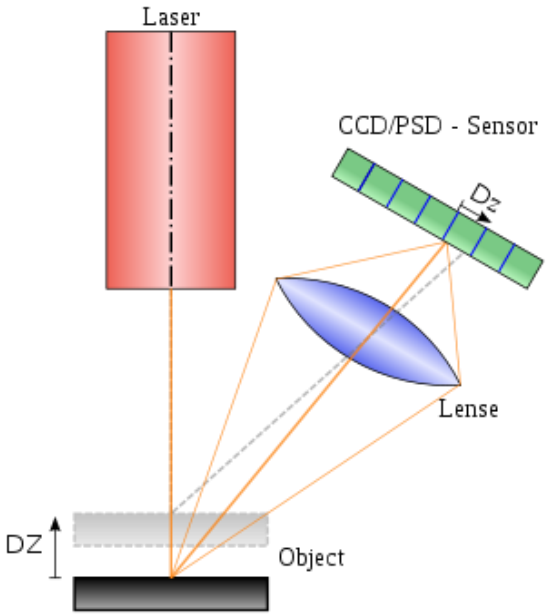

Figura 7. Funcionamiento de la técnica de triangulación para escáneres 3D.

La precisión de estos sistemas puede llegar a ser muy elevada, incluso más que siguiendo el método tiempo de vuelo, ya que la precisión puede alcanzar el orden de micrómetros. Sin embargo, el alcance máximo de estos escáneres se limita a unos 30 cm, lo que supone poco apropiado para largas distancias. Para aumentar el alcance, sería necesario incrementar mucho el tamaño de los equipos de medidas, lo que hace que suelan ser más voluminosos que otros dispositivos.

Ejemplos de escáneres 3D por triangulación son VIVID 910 de la empresa Konica Minolta (véase Figura 8) o Digitizer de Makerbot. A pesar de que los precios y tamaños varían, su rango de visión suele estar bastante limitado si se quiere mantener la precisión, lo que hace que no resulte la opción más adecuada para este proyecto. Por tanto, este tipo de dispositivos están orientados a obtener un modelo 3D muy preciso de un único objeto más que a un conjunto de ellos a modo de escena.

Figura 8. VIVID 910 de la empresa Konica Minolta.

En cuanto al escáner de luz estructurada, consiste en un dispositivo capaz de obtener la geometría de la escena mediante la proyección de un patrón de luz y su registro en un sistema de adquisición. Aunque es un método diferente al anterior, también está basado en

técnicas de triangulación [6]. De esta forma, una cámara lee la desviación de la proyección sobre la escena y, a través de técnicas de triangulación, se calcula la distancia de cada punto. Una de las mayores ventajas es la rapidez, ya que es capaz de escanear inmediatamente múltiples puntos o incluso la escena entera. Esto reduce el problema de deformación que pueden crear otros dispositivos cuya velocidad o rango de adquisición de datos sea menor.

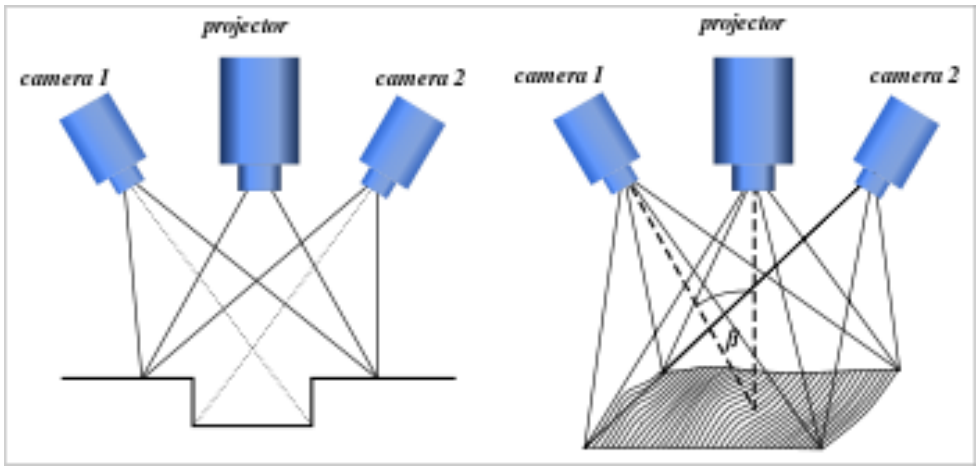

Figura 9. Proyección de patrones de luz sobre la superficie de la escena a través de un escáner de luz estructurada.

Como se observa en la Figura 9, la captación de los múltiples puntos que conforman la escena, se basa en la proyección de patrones de luz sobre las superficies. Mediante un análisis de las deformaciones de esos patrones de luz a través de dos cámaras, se obtienen las coordenadas de los puntos.

Un ejemplo de dispositivo basado en la técnica de luz estructurada es la cámara Kinect de Microsoft [7] (véase Figura 10). Este ha sido el periférico elegido para el desarrollo de este proyecto, ya que, además de tratarse de una cámara muy utilizada en proyectos sobre robótica en el ámbito de la investigación, es buena en relación calidad-precio. Su uso extendido en la robótica se debe, en mayor grado, al desarrollo de software orientado a facilitar la adquisición de nubes de puntos con este dispositivo.

Figura 10. Dispositivo Kinect para la Xbox 360.

2.1.2. Procesamiento de entornos 3D

Tras captar la información de una escena por medio de técnicas 3D, se obtiene una nube de puntos que contiene gran cantidad de información, por lo que es necesario ser procesada para determinar los detalles que interesen del entorno. Para ello, se deben aplicar una serie de algoritmos desarrollados para mejorar la confiabilidad de los datos y obtener un correcto modelamiento tridimensional de la escena. Estos algoritmos pueden ser muy diversos y se agrupan según su finalidad [8]:

- Filtrado (véase Figura 11). En esta librería está orientada a la eliminación de datos atípicos y ruido que pueda existir en la nube de puntos de partida.
- Características. Los algoritmos de este grupo permiten estimar parámetros de funciones que determinan si un conjunto de datos representa una función geométrica en el espacio.
- Puntos de interés. Esta librería permite hallar los puntos clave de una nube de puntos para trabajar con un menor número de puntos.
- Registro. La biblioteca de registro identifica la correspondencia entre cada nube de puntos captada sobre un objeto, realizando una transformación para minimizar el error de alineación.
- Kd-Tree. Consiste en una estructura de tipo árbol que almacena un conjunto de puntos k-dimensional con el fin de realizar búsquedas eficientes del vecino más cercano.
- Octree. Proporciona métodos eficaces para la creación de una estructura de datos

tipo árbol jerárquico, subdividiendo la nube de puntos en conjuntos para realizar el procesamiento de forma eficiente.

- Segmentación. Se trata de algoritmos encargados de la separación de puntos que pertenecen al mismo elemento para distinguirlos unos de otros, a partir de las características homogéneas que comparten dichos puntos.
- Modelado por consenso. Contiene herramientas para la estimación de modelos específicos en una nube de puntos, tales como líneas, planos, cilindros o esferas.
- Reconstrucción de superficies. A través de esta librería, se puede realizar una reconstrucción de la superficie original captada mediante un escáner 3D.
- Reconocimiento. Este módulo contiene algoritmos utilizados para aplicaciones con fines orientados al reconocimiento de objetos.
- IO. Esta biblioteca permite operaciones de lectura y escritura de archivos *PCD* (*Point Cloud Data*).
- Visualización. El objetivo de esta librería es permitir una rápida visualización de los resultados de los algoritmos aplicados sobre la nube de puntos.

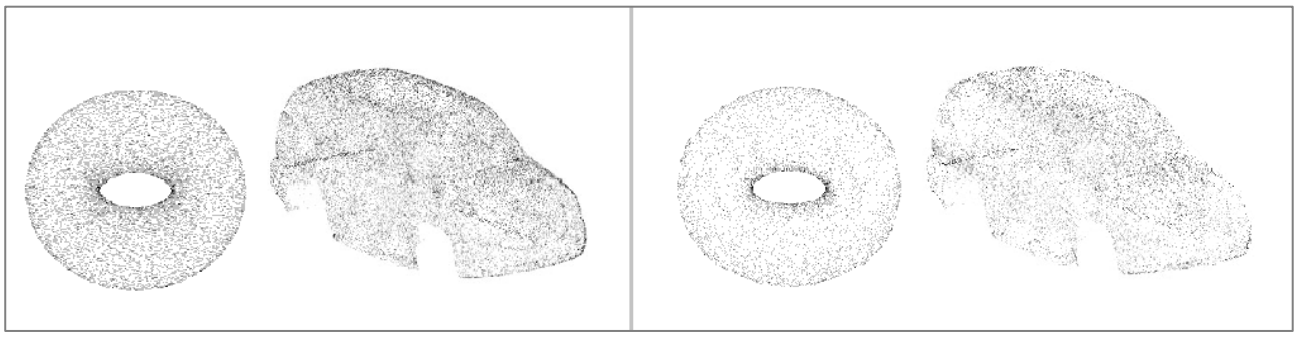

Figura 11. Aplicación del filtro de reducción del número de vértices. A la izquierda, la nube de puntos original de un toroide y de un coche; a la derecha, las nubes de puntos filtradas.

Entre todos los módulos descritos, en el presente proyecto se tiene interés por: filtrado, para reducir el número de puntos contenidos en la nube original; modelado por consenso, para detectar el plano correspondiente a la mesa; segmentación y kd-tree, para agrupar los puntos de cada objeto que hay encima de la mesa.

Para llevar a cabo cada una de las etapas, existen varias alternativas en cuanto a herramientas de desarrollo que pueden ser de utilidad hoy en día. Matlab, LabVIEW o PCL

son tres de las herramientas más utilizadas en aspectos de tratamiento de nubes de puntos.

En cuanto a Matlab, se trata de un conjunto de librerías para analizar y desarrollar sistemas y productos orientados a resolver problemas de ingeniería y científicos [9]. Su lenguaje de alto nivel basado en matrices hace que sea fácil su integración en trabajos de matemáticas computacionales. El amplio abanico de aplicaciones que abarca supone una ventaja con respecto a otras herramientas, lo que promueve su uso extendido tanto en el ámbito de la investigación como empresarial. Además, está preparado para tratar grandes conjuntos de datos como pueden ser las nubes de puntos. Como aspecto negativo, no se trata de software libre, por lo que su uso conlleva un coste.

Similar a Matlab, se encuentra LabVIEW, que consiste en un conjunto de aplicaciones orientado también para el ámbito de la ingeniería y la ciencia [10]. Su sintaxis de programación se basa en lo gráfico para facilitar visualizar, crear y codificar diferentes sistemas. Dentro de las posibles aplicaciones realizadas a través de LabVIEW, se encuentra la del procesamiento de nubes de puntos. Al igual que Matlab, el software tiene un coste alto.

Por otro lado, se encuentra la librería *PCL* (Point Cloud Library), orientado exclusivamente para el desarrollo de software para robots. Eso significa que incluye numerosas herramientas que facilitan el poder llevar a cabo cualquier proyecto de robótica, lo que conlleva el tratamiento con nubes de puntos. Por este motivo y por tratarse de software libre, es el más indicado y el elegido para este proyecto.

2.2. Planificación de trayectorias

Existen diversos objetivos que se le quiera proporcionar al robot, pero no cabe duda de que muchos de ellos necesitan de la planificación de trayectorias, ya sea para una repetición automática de los movimientos o para ser más flexible de cara a reaccionar ante las variaciones del entorno.

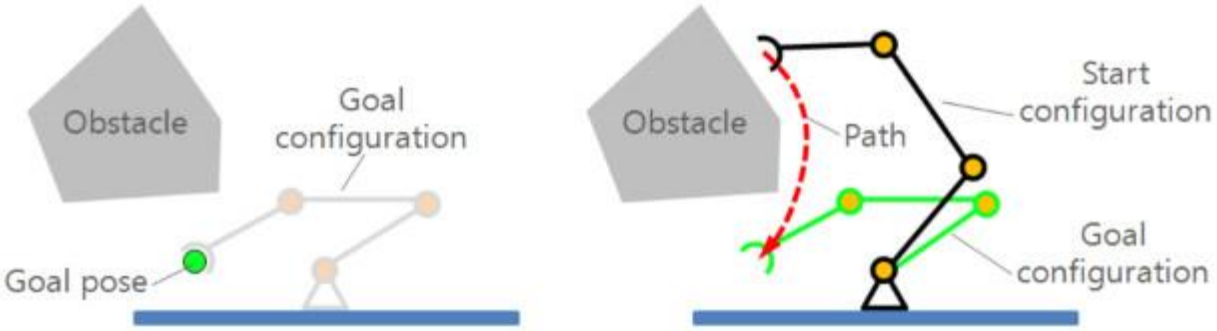

Figura 12. Representación gráfica de una planificación de trayectoria. A la izquierda, la posición final que debe alcanzar el robot; a la derecha, su estado inicial y el plan de movimiento generado.

El problema que se plantea es determinar el camino que debe seguir el robot para alcanzar un punto final determinado a partir de su posición inicial (véase Figura 12). Existen dos formas de solucionar ese problema: a través del espacio cartesiano o a través del espacio de las articulaciones [11]. En el espacio cartesiano, la especificación de caminos se realiza en términos de las posiciones y orientaciones del efector final con respecto a un sistema de referencia. En líneas generales, se trata de estudiar la evolución de las variables cartesianas en el tiempo. Para el caso del espacio de las articulaciones, se trata de determinar una función suave de interpolación para cada articulación, sin tener en cuenta un control de posiciones/orientaciones cartesianas entre dos puntos de paso consecutivos. Mientras que en el caso del espacio cartesiano conlleva una mayor complejidad de cálculo por tener que resolver en tiempo real el modelo cinemático inverso, en el caso de las articulaciones lo calcula únicamente una vez. Por otra parte, este último es más difícil de interpretar, precisamente porque se deja de trabajar con variables cartesianas para pasar a tratar con variables más complejas.

Para simplificar el proceso para alcanzar dichos objetivos, existen algunas herramientas capacitadas para solventar la planificación de trayectorias de forma resolutiva, de tal forma que facilite al usuario el poder obtener una solución acorde a lo que se necesita. Entre las herramientas disponibles, destacan OMPL, OpenRave y MoveIt!, principalmente por tratarse todos ellos de software libre.

En cuanto a OMPL (Open Motion Planning Library), es un paquete de software para el cálculo de las trayectorias de movimiento usando algoritmos basados en el muestreo del entorno [12]. El contenido de la biblioteca se limita a los algoritmos de planificación de

movimiento, lo que significa que no hay especificación del entorno, es decir, no existe detección de colisiones. Con esto se pretende orientar esta librería para integrarse fácilmente en los sistemas que ya proporcionan los componentes adicionales requeridos.

Por otro lado, existe OpenRAVE que proporciona un entorno para pruebas, desarrollo e implementación de algoritmos de planificación de movimiento en aplicaciones de robótica del mundo real [13]. Su función se centra en la simulación y el análisis de la información cinemática y geométrica relacionada con la planificación de movimientos. Incluye librerías adaptadas para detectar las colisiones, por lo que resulta un candidato para implementarlo en este proyecto. Sin embargo, se descartó el uso de OpenRave porque su desarrollo se encuentra parado.

La última opción propuesta es MoveIt!, que es capaz de resolver cinemáticas, representar entornos 3D y planificar trayectorias teniendo en cuenta los obstáculos de la escena [14] [15].Es Similar a OpenRave en cuanto a funcionalidades, aunque, a diferencia de éste, utiliza el paquete OMPL como planificador de trayectorias. Finalmente, se optó por este software, ya que facilita la integración de la visualización del entorno 3D y la planificación de trayectorias, incorporando los últimos avances en estas tecnologías. Además, existe una comunidad detrás proporcionando todo lo necesario para su correcto funcionamiento e implementación.

Tras la elección de las herramientas a utilizar en el presente proyecto, tanto de hardware (Kinect) como de software (PCL y MoveIt!), se ha decidido unificarlas a través del framework ROS, ya que permite la implementación en un mismo entorno de todas ellas [16].

3. Herramientas usadas en el desarrollo

En el presente apartado se explica las principales herramientas necesarias para la elaboración del trabajo, tanto software como hardware. En el caso del hardware, se ha utilizado la cámara Kinect como medio de captación de la nube de puntos que representa el entorno. Para tratar esa nube de puntos, se ha requerido de la librería PCL y el framework ROS.

3.1. Kinect

La Kinect consiste en un dispositivo de procesamiento digital de imágenes, orientado desde sus inicios para su uso exclusivo con la consola Microsoft Xbox 360. Sin embargo, en 2011 se lanzó el SDK (Software Development Kit) que permitía usar las funciones de profundidad de la Kinect de manera libre para obtener información del entorno. Esto, sumado a su bajo coste y al desarrollo de nuevas aplicaciones orientadas a la lectura de esos datos, hizo que hoy en día sea muy utilizado en el ámbito de la investigación [7].

El hardware de Kinect está compuesto por una barra horizontal unido a una base que a través de un eje de articulación motorizado permite inclinarse ± 27º. Los sensores que dotan a la Kinect de sus funcionalidades son:

- Cámara RGB que detecta la información del color de cada pixel a través de los tres colores primarios (rojo, verde y azul) y un número indicativo del porcentaje de cada uno de ellos.

- Sensor de profundidad, que se compone de un proyector infrarrojo y un sensor CMOS (Semiconductor de Óxido de Metal Complementario) monocromático. Su campo de visión es de unos 57º horizontalmente y 43º verticalmente. Además, puede detectar un rango de 1,2 - 3,5 metros.

- Conjunto de micrófonos para la obtención del audio.

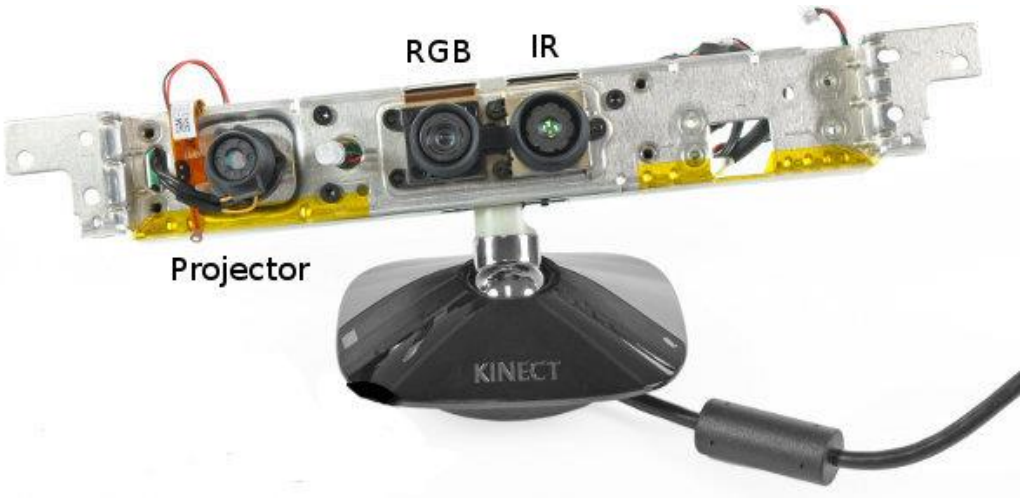

Figura 13. Interior de la Kinect.

La Kinect es capaz de capturar 30 *frames* por segundo, con una resolución de 640x480 píxels. La alimentación de la cámara se basa en una conexión USB y una alimentación de 12 V y 1,08 A en corriente continua.

3.2. Point Cloud Library

La librería de nube de puntos (*Point Cloud Library* o *PCL*) es un proyecto a gran escala de software libre, para imágenes 2D/3D y procesamiento de nubes de puntos [8]. Contiene numerosos algoritmos incluyendo filtrado, reconstrucción de la superficie, registro, modelado y segmentación. Estos algoritmos pueden ser usados, por ejemplo, para filtrar valores extremos de los datos, unir nubes de puntos 3D distintas, dividir partes relevantes de una escena, extraer puntos claves y calcular descriptores para reconocer objetos reales por su apariencia geométrica, entre otros.

Figura 14. Logo de la librería PCL.

PCL es de código abierto y funciona bajo los términos de licencia BSD. Es gratuito tanto

para uso comercial como de investigación. Además, es multiplataforma, y ha sido lanzado satisfactoriamente en Linux, MacOS, Windows y Android/IOS. Su éxito se debe a un gran número de ingenieros y científicos de diferentes organizaciones que contribuyen en su desarrollo.

Para simplificar su aplicación en proyectos, PCL se divide en una serie de librerías que pueden ser compiladas por separado. Esta modularidad es importante para distribuir PCL en plataformas con una capacidad computacional reducida o almacenamiento reducido.

3.2.1. ¿Qué es una nube de puntos?

Para entender mejor las posibles aplicaciones de la PCL, es necesario tener unas nociones de lo que representa una nube de puntos.

Se trata de una estructura de datos utilizada para representar una colección de puntos tridimensionales. En una nube de puntos 3D, los puntos son representados normalmente por los ejes de coordenadas geométricas X, Y, Z de una superficie muestreada. También puede estar incluida la información correspondiente con el color y su visualización es similar a la representada en la Figura 15, aunque dependerá de la resolución con la que se muestre.

Figura 15. Nube de puntos de diferentes objetos.

Las nubes de puntos se pueden crear a partir de sensores, tales como cámaras, escáneres 3D, o generados a partir de un programa de ordenador de forma sintética. Tal y como se ha

señalado anteriormente, en ese proyecto se ha utilizado la cámara Kinect de Microsoft para la captación de la nube de puntos.

3.3. ROS

El proyecto se ha desarrollado en el framework ROS (del inglés *Robot Operating System*), orientado para escribir software aplicado a robots. Consiste en una colección de herramientas, librerías y convenios que tienen como objetivo simplificar la tarea de crear una funcionalidad compleja y robusta para un robot a través de una amplia variedad de plataformas [17] [18]. Se trata de una herramienta de código abierto y la versión de ROS utilizada en este proyecto es *Indigo Igloo.*

Figura 16. Logo de ROS Indigo Igloo.

ROS funciona con el sistema operativo Linux[1], principalmente por tratarse ambos de código abierto. Además, ROS se organiza por paquetes que se controlan explorando las carpetas y ejecutando los comandos necesarios desde la terminal (véase Figura 17), lo que hace que Ubuntu sea una buena opción como sistema operativo por su uso extendido de las terminales. Sin embargo, se están desarrollando otras versiones de ROS experimentales, por lo que no se descarta que se adapten a otros sistemas operativos como Mac OS X o Microsoft Windows.

[1] La versión utilizada de Linux es Ubuntu 14.04.

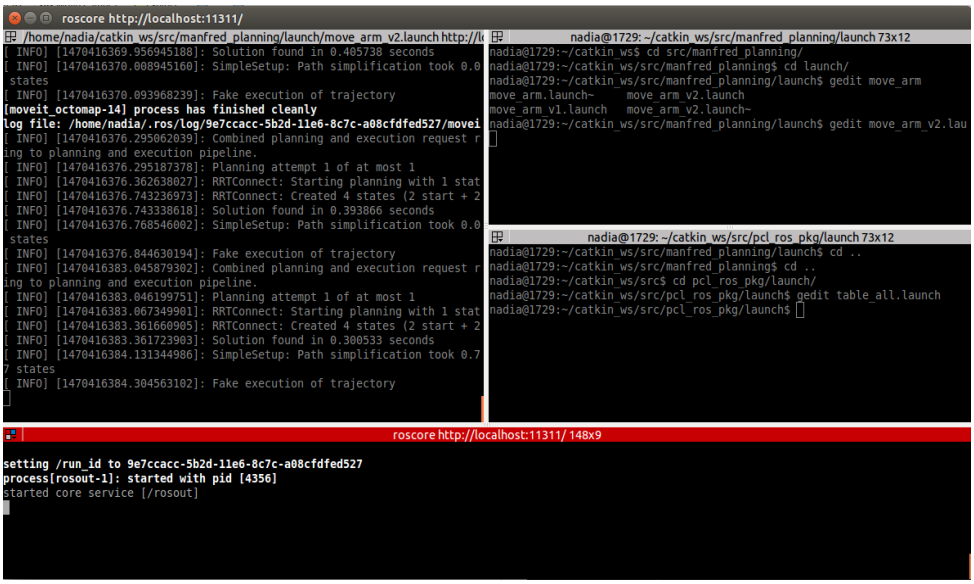

Figura 17. Uso de ROS a través de las terminales en Ubuntu.

ROS facilita mucho el desarrollo de aplicaciones para robots, ya que obtener un software de propósito general puede resultar ser una tarea difícil. Desde la perspectiva de un robot, tareas que pueden resultar triviales para un ser humano, puede variar enormemente su desarrollo en función de la tarea a alcanzar y el entorno en el que se encuentra, lo que haría cambiar su programación. Hacer frente a estas variaciones no resulta nada fácil para nadie, por lo que poder participar en una comunidad con los mismos objetivos puede ser muy provechoso.

Es por ello que ROS se creó para fomentar una participación masiva entre los contribuyentes que se quisiesen unir a la comunidad para compartir los avances y conseguir así desarrollos de software colaborativos. De esta forma ROS consigue que se colabore y construya sobre trabajos de otros para lograr avanzar de manera más efectiva.

Figura 18. Ecuación de ROS: Paquetes + Herramientas + Capacidades + Ecosistema. (Fuente: http://www.ros.org/)

En cuanto a su funcionamiento, ROS fue diseñado para ser todo lo modular posible, con el objetivo de que los usuarios puedan usar ROS para cualquier nivel de aplicación que deseen, de tal forma que permita seleccionar al usuario qué partes son útiles para aplicar en sus proyectos y qué partes prefiere implementarlo por su cuenta. Esto sucede gracias a ROS se compone de un sistema de nodos independientes, en el que cada nodo se comunica con el resto utilizando el modelo publicador/suscriptor.

Tras realizar un resumen de los conceptos principales de ROS tratados en este proyecto, se procede a explicar las herramientas utilizadas.

3.3.1. Conceptos básicos

El sistema de organización de ROS se basa en carpetas estructuradas siguiendo un orden específico que ayuda a que la compilación se realice correctamente.

En primer lugar, hay que configurar un espacio de trabajo (*workspace*) que contendrá toda la información necesaria para el desarrollo del proyecto. Esta información se agrupa, a su vez, en paquetes (*package*) que pueden contener librerías, ficheros de configuración, ficheros ejecutables y en general todo lo necesario para la implementación de las funcionalidades que se le quieran otorgar al robot.

A nivel de organización computacional, pueden existir varios procesos que se encuentren procesando información al mismo tiempo. Para entender ese funcionamiento, se requiere disponer de unas nociones básicas de los siguientes conceptos:

- *Master*: es lo primero que se lanza y es el núcleo de la estructura para el funcionamiento de ROS. Su principal función es permitir al resto de procesos encontrarse y comunicarse entre ellos.

- Nodos (*nodes*): son ejecutables encargados de realizar diferentes procesos. Esto hace que ROS pueda ser modular, ya que cada nodo es independiente del resto y se comunican utilizando un modelo de mensajes publicador/suscriptor. En general, los nodos se encargan de la publicación o suscripción de flujos de datos tales como

imágenes, láser, de control, actuadores, etc.

- *Topics*: son las etiquetas usadas para identificar el contenido de los mensajes enviados por los nodos. De esta forma, un nodo que necesite ciertos tipos de datos podrá suscribirse al *topic* correspondiente.

- Mensajes (*Messages*): es simplemente una estructura de datos que comprende diferentes tipos de datos. Sirven principalmente para que los nodos se puedan comunicar entre ellos enviando estos mensajes.

- Servicios (*Services*): A pesar de que el sistema de publicación y suscripción constituye un conjunto de comunicación flexible, no resulta del todo apropiado cuando se busca una interacción de tipo petición/respuesta, normalmente requeridos en sistemas distribuidos. Esta clase de peticiones se realizan a través de servicios. Su funcionamiento se basa en que un determinado nodo ofrece un servicio bajo un nombre y un cliente usa ese servicio para enviar un mensaje de petición y esperar una respuesta.

- *Bags*: son un formato de archivo para guardar y reproducir información de los mensajes de ROS. Suele ser útil para almacenar los datos provenientes de los sensores, como puede ser la nube de puntos que proporciona un escáner 3D.

El flujo que representa las funcionalidades de un proyecto desarrollado en ROS, empieza por el *Master*, que almacena la información registrada de los *topics* y los servicios para proporcionársela a los nodos. Los nodos pueden reportar al *Master* sus datos de registro, así como recibir información de registros de otros nodos realizando las comunicaciones apropiadas. Una vez establecidos los registros, los nodos son capaces de comunicarse entre ellos directamente siguiendo el sistema de publicación/suscripción de *topics*. De esta forma, un nodo que se suscribe a un *topic*, solicitará las conexiones a los nodos que publican en ese *topic* (véase Figura 19).

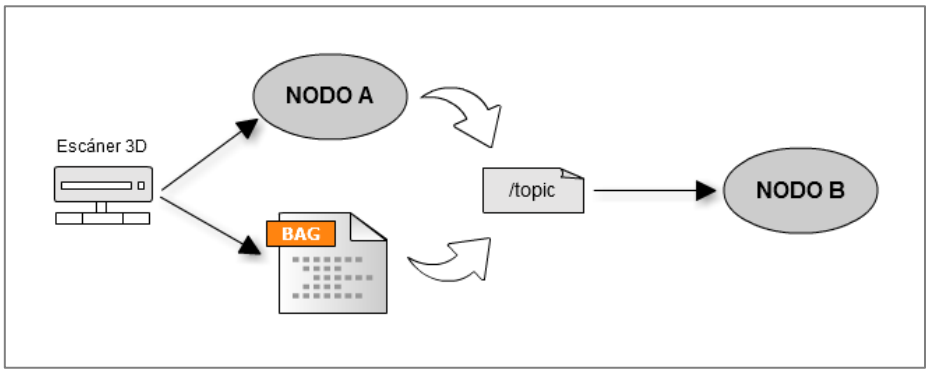

Figura 19. Representación gráfica del sistema de publicación/suscripción de ROS.

3.3.2. Librería TF

La librería tf fue diseñada para proporcionar una forma estándar de realizar un seguimiento de los *frames* o sistemas de referencia y transformar los datos dentro de un mismo marco de tal forma que los usuarios que trabajen con diferentes componentes individuales pueden estar seguros de que los datos están en el sistema de coordenadas que quieren sin necesidad de preocuparse de todos los existentes [19]. La mayoría de los sistemas robóticos usan muchos sensores o actuadores diferentes, con distintos marcos de coordenadas (véase Figura 20) lo que hace que los sistemas robóticos se vuelvan complicados. Ser capaz de concentrarse precisamente en el marco de tareas y sólo en los sistemas de referencias relevantes se vuelve crítico.

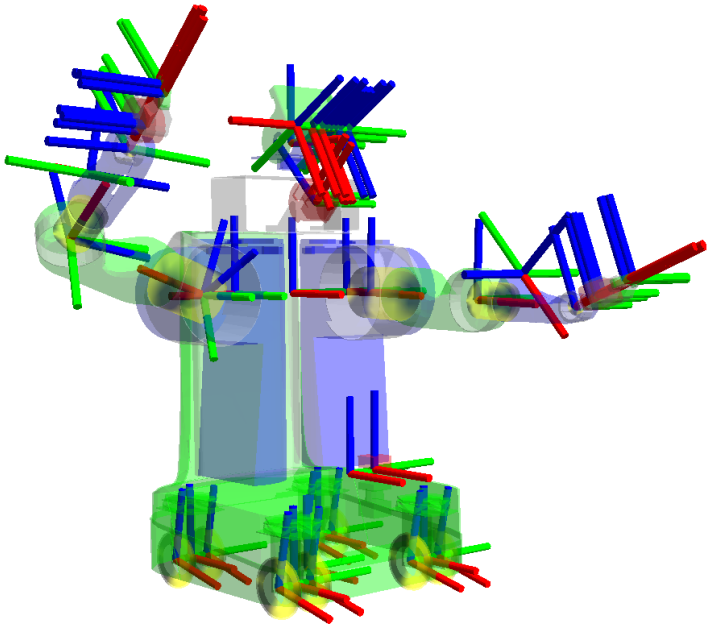

Figura 20. Vista de todos los frames estándar del robot PR2.

En líneas generales, tf es un paquete que permite al usuario realizar un seguimiento de múltiples sistemas de coordenadas a lo largo del tiempo. Esto quiere decir que mantiene esa relación basándose en una estructura de árbol almacenada en el tiempo, con el objetivo de que el usuario pueda transformar puntos, vectores, etc., entre dos sistemas de referencia en cualquier momento deseado en el tiempo. Dicha relación se base en vincular un sistema de referencia a otro sistema padre, proporcionando la información necesaria para especificar la correspondencia que existe entre ambas. Esta información puede ser la posición y la orientación de uno con respecto al otro.

3.3.3. Rviz

Rviz se trata quizás de la herramienta más extendida de ROS, que proporciona un entorno de propósito general para la visualización tridimensional de muchos tipos de datos provenientes de sensores e incluso cualquier robot con su modelo ya configurado.

Rviz puede visualizar muchos de los tipos de mensajes comunes previstos en ROS, tales como escáneres láser, nubes de puntos tridimensionales e imágenes desde una cámara. También se utiliza información de la biblioteca tf para mostrar todos los datos de los sensores junto con una representación tridimensional del robot en un sistema de coordenadas común. La visualización de todos los datos en la misma aplicación permite determinar rápidamente lo que ve el robot e identificar problemas tales como errores de alineación de los sensores o inexactitudes del modelo del robot.

Además, se obtiene una representación de cómo sería la interacción entre robot y entorno, determinando de una manera rápida e intuitiva cómo afrontarlo para lograr alcanzar los objetivos.

Por tanto, Rviz está preparado para representar de manera virtual diferentes tipos de datos, así como nubes de puntos, octomap, sistemas de coordenadas a través del paquete tf, modelo del robot, etc.

Según se indica en Figura 21, los parámetros a tener en cuenta en Rviz para una correcta

visualización de los elementos a mostrar (*displays*) incluyen los relacionados con el sistema de referencia y las características parametrizables del tipo de dato en cuestión. Como sistema de referencia existe el sistema fijo (*Fixed Frame*) que proporciona una base de referencia para la visualización, es decir, todos los elementos incluidos en Rviz se mostrarán en ese sistema de referencia. Además, se dispone de una lista de *displays* que incluye todos los elementos posibles que pueden ser representados en el mundo 3D de Rviz. Una vez seleccionado un *display* se mostrará una lista de parámetro configurables desde Rviz que variará en función del tipo de *display.*

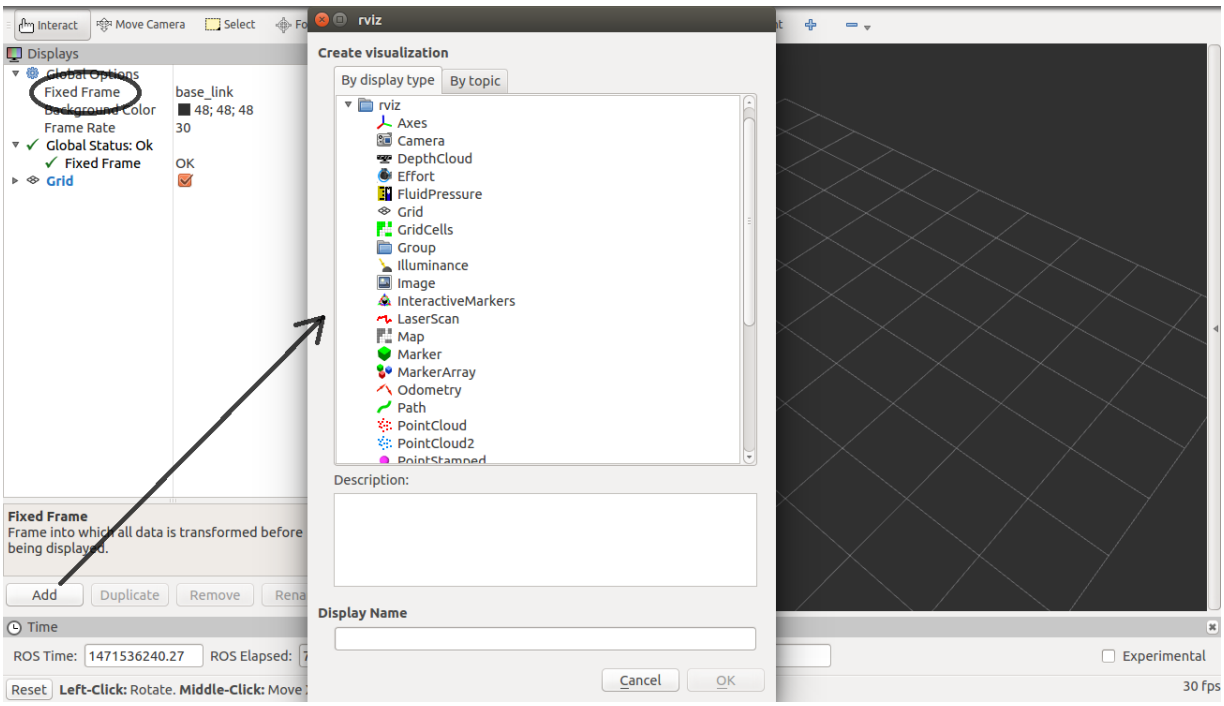

Figura 21. Interfaz de rviz, incluyendo configuración del Fixed Frame y listado de Displays disponibles.

Rviz permite cargar y guardar diferentes configuraciones con el fin de no tener que repetir el proceso de definir los parámetros cada vez que se abre la interfaz. El archivo que almacena las configuraciones contiene:

- Los diferentes *display* establecidos junto con sus propiedades.
- Las propiedades de la herramienta.
- El tipo de cámara seleccionada para cargar un punto de vista determinado.

3.3.4. Octomap

Octomap consiste en un eficiente mapa en 3D basado en Octrees [20]. Como se representa en la Figura 22, un Octree es una estructura en árbol de datos en la cual cada nodo interno se subdivide en 8 octantes (de ahí su nombre, *oct* de octante y *tree* de árbol).

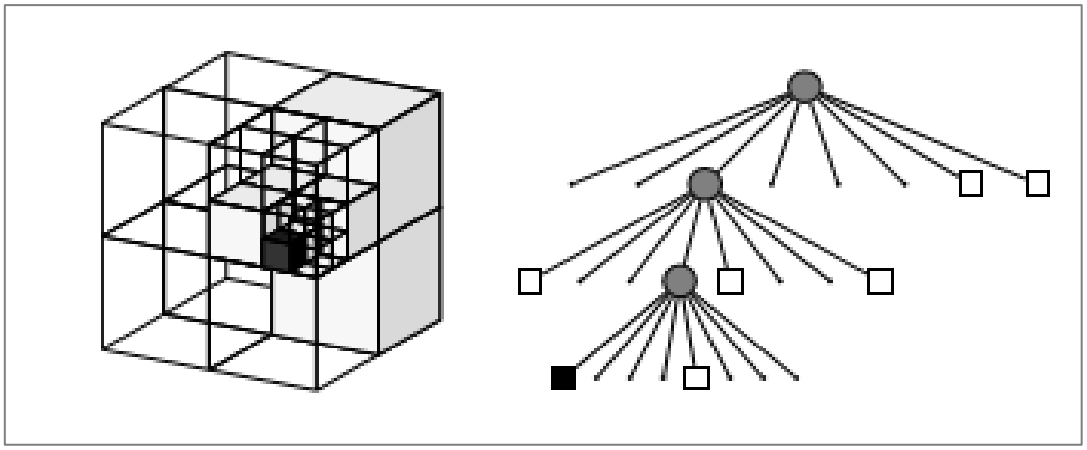

Figura 22. Representación gráfica de una estructura octree.

La librería OctoMap implementa un método de asociación y de ocupación rejilla 3D, proporcionando estructuras de datos y algoritmos de mapeado en C++ especialmente adecuado para la robótica. Entre sus principales ventajas se encuentran:

- Modelo completo 3D. El mapa es capaz de modelar entornos arbitrarios, sin necesidad de proporcionarle más información que la propia nube de puntos. Estos modelos de representación engloban tanto áreas ocupadas como el espacio libre.

- Actualizable. Es posible añadir nueva información de los sensores en cualquier momento. Por otra parte, varios robots son capaces de contribuir al mismo mapa y un mapa previamente grabado es extensible cuando se exploran nuevas áreas.

- Flexible. La extensión del mapa no tiene porqué ser conocida de antemano. En lugar de ello, el mapa se expande dinámicamente según sea necesario. Además, el mapa es multi-resolución de manera que, por ejemplo, un planificador de alto nivel es capaz de utilizar un mapa de alta resolución, mientras que un planificador local puede operar utilizando una baja resolución. Esto permite visualizar mapas eficientemente,

los cuales escalan en función de la cantidad de detalles que se le quiera proporcionar.

- Compacto. El mapa se almacena de manera eficiente, tanto en memoria como en disco. Es posible generar archivos comprimidos para ser utilizados posteriormente o ser intercambiados entre robots incluso bajo restricciones en el ancho de banda.

La librería OctoMap está disponible como distribución para Linux, Mac OS y Windows. La versión de Linux es la más recomendada y la utilizada en este proyecto.

Figura 23. Octomap con diferentes resoluciones de un árbol.

3.3.5. MoveIt!

MoveIt! es software diseñado para la manipulación móvil, que incorpora los últimos avances en planificación del movimiento, manipulación, percepción 3D, cinemática, control y navegación [14]. Proporciona una plataforma fácil de usar para el desarrollo de aplicaciones de robótica avanzada, evaluación de nuevos diseños de robots y construcción de robot integrados para el sector industrial, comercial, I+D y otros dominios.

La arquitectura del sistema se basa en un nodo primario proporcionado por MoveIt! llamado *move_group,* como se puede observar Figura 24. Este nodo actúa como un integrador: lanzando todos los componentes individuales juntos para proporcionar a los usuarios un conjunto de acciones y servicios de ROS. Como el proyecto se ha implementado en C++, el paquete necesario para poder configurar la interfaz de *move_group* es *move_group_interface.*

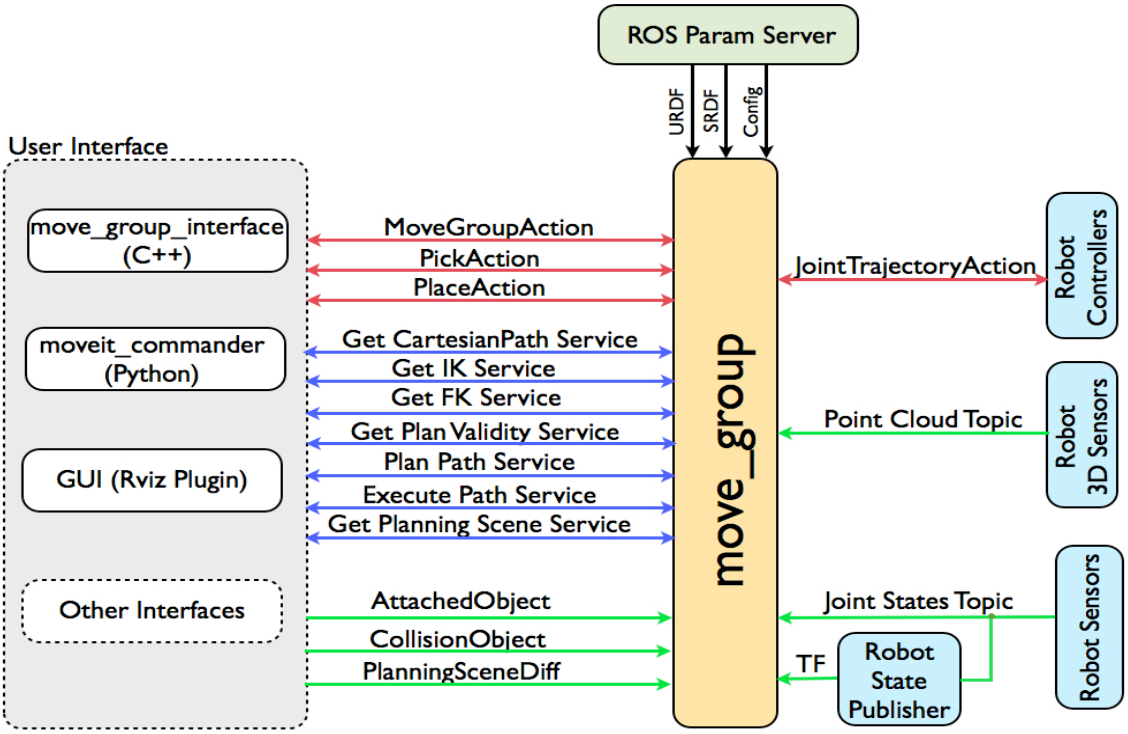

Figura 24. Arquitectura de MoveIt! basado en el nodo primario move_group. (Fuente: http://moveit.ros.org/)

En cuanto a la interfaz del robot, move_group permite comunicarse con él a través de *topics* y acciones de ROS para obtener información sobre el estado actual (posiciones de las articulaciones, etc.), para obtener nubes de puntos y otros datos de los sensores del robot, así como comunicarse con los controladores del robot. Para ello, es necesario establecer una serie de configuraciones:

- Transformación de la información: *move_group* monitoriza la transformación de la información usando la librería de ROS TF. Esto permite al nodo obtener la información global sobre la posición del robot.

- *move_group* utiliza el monitor de planificación de la escena (*Planning Scene Monitor*) para llevar a cabo una planificación, la cual incluye una representación del mundo y el estado actual del robot y las configuraciones para permitir las interacciones entre ambos.

MoveIt! trabaja con los planificadores de movimiento (*motion planner*) a través de una interfaz de complementos. Esto permite a MoveIt! comunicar y utilizar los diferentes planificadores de movimiento a partir de múltiples librerías, haciendo MoveIt! fácilmente extensible.

La petición de *motion planner* especifica claramente lo que le se quiere que el robot haga. Normalmente, se le pedirá al *motion planner* mover el brazo a una ubicación diferente (en el espacio de la articulación) o en su defecto a una nueva pose. Las colisiones se comprueban de forma predeterminada (incluyendo auto-colisiones). Además, permite adjuntar un objeto al efector final (o a cualquier parte del robot) para, por ejemplo, adjuntar un objeto al robot. Esto hace que el *motion planner* tenga en cuenta el movimiento del objeto mientras planifica la ruta.

De este modo, partiendo del Octomap, se puede obtener un mapa de ocupación de los alrededores del robot a través del *Occupancy map monitor*, encargado de construir una representación 3D del entorno, sumado a la información que proporciona el *planning scene* sobre los objetos.

4. Diseño y desarrollos

Para afrontar el desarrollo del proyecto cuyo objetivo consiste en planificar al robot Manfred para el agarre de un objeto situado sobre una mesa, se pueden agrupar las etapas en cuatro: configuración del sistema, adquisición de datos del entorno, tratamiento de la nube de puntos y la configuración necesaria para su representación e implementación en MoveIt!

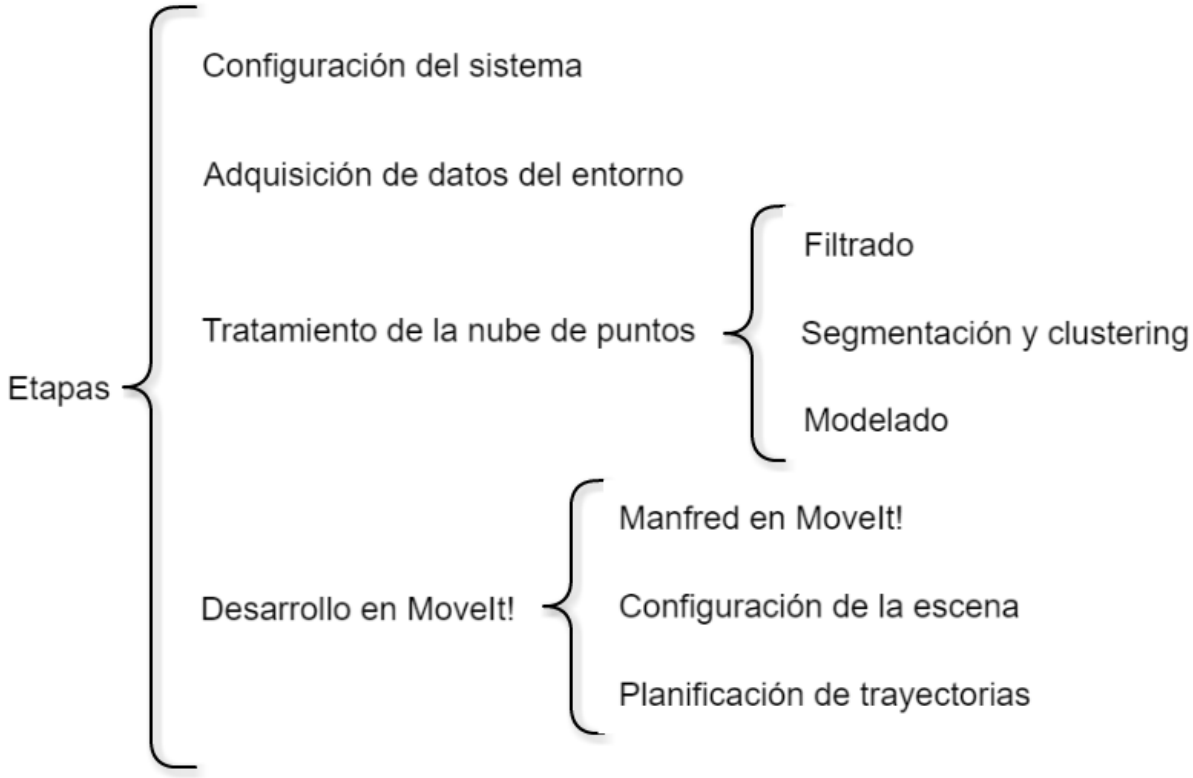

Figura 25. Esquema de las fases llevadas a cabo en el proyecto.

Como se observa en el esquema de la Figura 25, el primer bloque consiste en la configuración del sistema para poder aplicar las librerías necesarias, así como establecer la relación entre los sistemas de referencia utilizados.

La siguiente fase consiste en la obtención de la nube de puntos que representa el entorno físico para disponer de los objetos posibles para su agarre por parte del robot Manfred. Se ha optado por realizar una grabación de dicha nube de puntos.

Dentro del tercer bloque, se han establecido tres fases con el objetivo de obtener una nube

de puntos preparada para su posterior adaptación en el entorno *MoveIt!:*

- Aplicación de filtros para un procesamiento más rápido y efectivo.
- Segmentar y extraer la información necesaria de los objetos para distinguirlos entre ellos.
- Modelado de la nube de puntos a través de Octomap.

Finalmente, en el tercer bloque se procede a configurar el robot Manfred y las colisiones de dicho entorno para la consiguiente planificación con el objetivo de simular el agarre de un objeto.

Cabe destacar que todos los resultados obtenidos se reflejarán individualmente en cada apartado.

4.1. Configuración del sistema

Para poder desarrollar el proyecto, es necesario crear un paquete en ROS indicando las dependencias, es decir, las librerías o paquetes necesarios para el correcto funcionamiento de los nodos que se desarrollen. En este caso, el paquete incluirá, principalmente, los relacionados con la librería PCL para su uso en el tratamiento de la nube de puntos, la librería *roscpp* que facilita la programación en C++ de los nodos, y el paquete *sensor_msgs* para uso de los diferentes tipos de mensajes que permite ROS.

Además, para una ejecución conjunta de los nodos en ROS se ha hecho uso de los archivos *launch*. Estos archivos de formato *.launch* proporcionan una recomendable forma de lanzar múltiples nodos así como establecer las configuraciones de parámetros requeridos. En cuanto a los lenguajes de programación empleados, se ha hecho uso de un formato específico de XML para la implementación de los archivos *launch*.

Otra configuración a realizar inicialmente está relacionada con los sistemas de coordenadas que se utilizarán. En este caso, existe la referencia de la cámara, que se le llamará *camera_link*, y la del mundo o base del robot, *base_link* (véase Figura 26). Será necesario, por tanto, establecer una relación entre ambas para poder planificar las trayectorias con una escena colocada de manera coherente con respecto al robot.

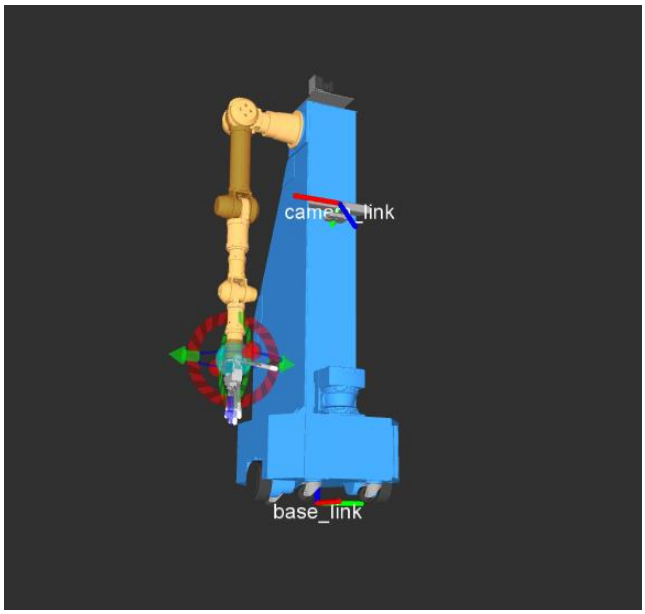

Figura 26. Sistema de referencia del mundo (base del robot) y de la cámara Kinect sobre Manfred.

Para ello, se ha utilizado el paquete tf, ya que permite al usuario transformar los puntos de la nube basándose en esa relación establecida entre los sistemas de referencia. Dentro del paquete tf, existen diversas herramientas para configurar todo lo necesario entre sistemas de coordenadas. En este caso, debido a que se trata de dos sistemas cuya relación permanece estática a lo largo del tiempo (no se trata de partes móviles) se hizo uso de la herramienta *static transform publisher* (publicador de una transformada estática) configurado en el archivo *launch* que lanzará todos los nodos necesarios para la visualización deseada de la nube de puntos.

```
<node pkg="tf" type="static_transform_publisher" name="tranform" args="0.08 -0.5
1.3 -1.57 0 -2.356 base_link camera_link 100" />
```

Los argumentos requeridos para la transformación son:

- *x/y/z offset:* los tres primeros números corresponden con el desplazamiento en metros de cada eje x, y ó z.
- *yaw/pitch/roll*: giros que son necesarios aplicar sobre el sistema de referencia de la cámara para obtener la misma orientación que el sistema de referencia global. *Yaw* es la rotación correspondiente al eje Z, *pitch* sobre el eje Y, y *roll* sobre X. Los tres se expresan en radianes.
- *Frame id*: sistema de referencia global, en este caso el mundo *base_link*.

- *Frame id hijo*: sistema de referencia sobre el que se aplica la transformación (*camera_link*).
- *Periodo*: tiempo en milisegundos que indica la frecuencia con la que se envía la transformación. 100ms es el valor recomendado.

De esta manera, el robot pasa de tener la información de la escena con respecto a la cámara (Figura 27, izquierda) a tenerla respecto a su *frame* de referencia principal (Figura 27, derecha).

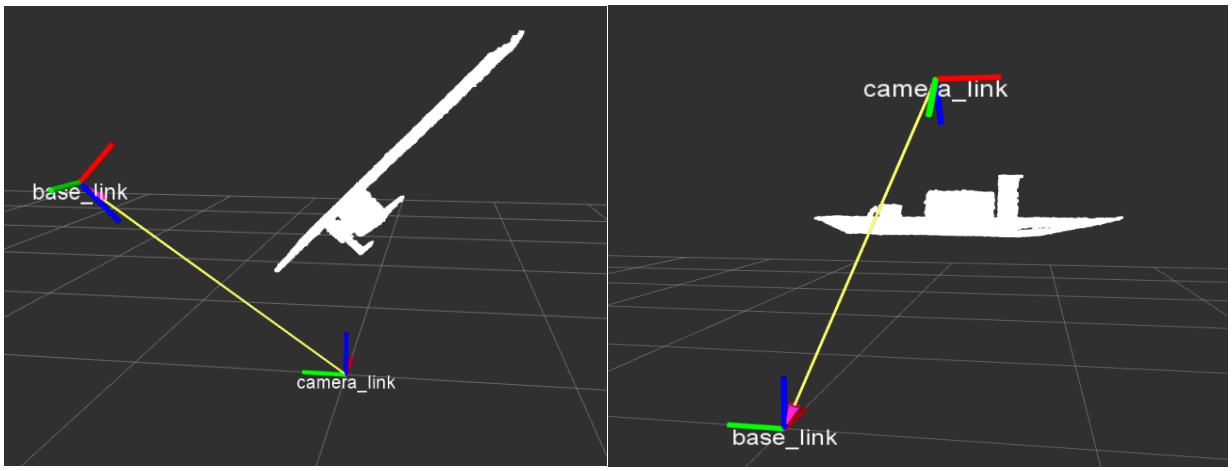

Figura 27. Nube de puntos referenciada con respecto a los frames camera_link y base_link.

4.2. Adquisición de datos del entorno

La adquisición de los datos del entorno se ha realizado mediante una grabación del entorno. Esta idea se basa, principalmente, en la comodidad que esto supone con respecto a realizarlo en tiempo real. Esto quiere decir que, una vez se ha almacenado la información de la escena en un archivo, es más fácil trabajar con ella por dos motivos principalmente:

- No se requiere del periférico encargado de la captación del entorno, en este caso la Kinect, en cada momento que se vaya a trabajar con la nube de puntos. Asimismo, se evita tener que estar preparando la mesa con los objetos continuamente.

- Se trabaja siempre con el mismo entorno, lo que resulta útil para llevar a cabo la depuración de errores.

Sin embargo, esto no quiere decir que el software a desarrollar no sea genérico para cualquier entorno similar. Todo lo contrario, se ha comprobado que funciona con diferentes nubes de puntos de similares características. En concreto, como el objetivo del proyecto es conseguir que Manfred se aproxime a un objeto para simular su agarre, se parte de la grabación de un plano con ciertos objetos encima: una taza, una caja y un bote de patatas fritas, tal y como se puede observar en la Figura 28.

Figura 28. Visualización de la nube de puntos grabada.

La opción que se utiliza para almacenar la información correspondiente al entorno es a través de la librería PCL. El archivo que almacena dicha información tiene una extensión *pcd* y su contenido incluye las coordenadas de los puntos a visualizar para conformar la nube de puntos en cuestión, como se muestra en la Figura 29.

```
$ cat test_pcd.pcd
# .PCD v.5 - Point Cloud Data file format
FIELDS x y z
SIZE 4 4 4
TYPE F F F
WIDTH 5
HEIGHT 1
POINTS 5
DATA ascii
0.35222 -0.15188 -0.1064
-0.39741 -0.47311 0.2926
-0.7319 0.6671 0.4413
-0.73477 0.85458 -0.036173
-0.4607 -0.27747 -0.91676
```

Figura 29. Contenido de un archivo PCD.

Para poder tratar la nube de puntos contenida en el archivo PCD a través del framework ROS, es necesario ejecutar un comando para convertir esa información en un tipo de mensaje de ROS (llamado *PointCloud2*), pasando como primer parámetro el archivo PCD (*test_pcd.pcd*) y como segundo el intervalo de tiempo en segundos que permanece dormido entre cada ejecución (en este caso, 1 segundo). Además, se indica el sistema de referencia en el que se ha grabado (*camera_link*) y como resultado, publica la nube de puntos con el topic *cloud_pcd*:

```
<node pkg="pcl_ros" type="pcd_to_pointcloud" name="pcd_to_ros" output="screen"
      args ="$(find pcl_ros_pkg)/pcd/test_pcd.pcd 1">
      <param name="frame_id" value="camera_link" />
      <remap from="cloud_in" to="cloud_pcd" />
</node>
```

Una vez obtenemos el tipo de mensaje *PointCloud2* de ROS, podemos proceder a visualizarlo a través de rviz. Para ello, es necesario establecer las siguientes propiedades:

- *Fixed Frame*: *base_link.*
- Añadir un display de tipo *PointCloud2* indicando el *topic* como */cloud_pcd.*

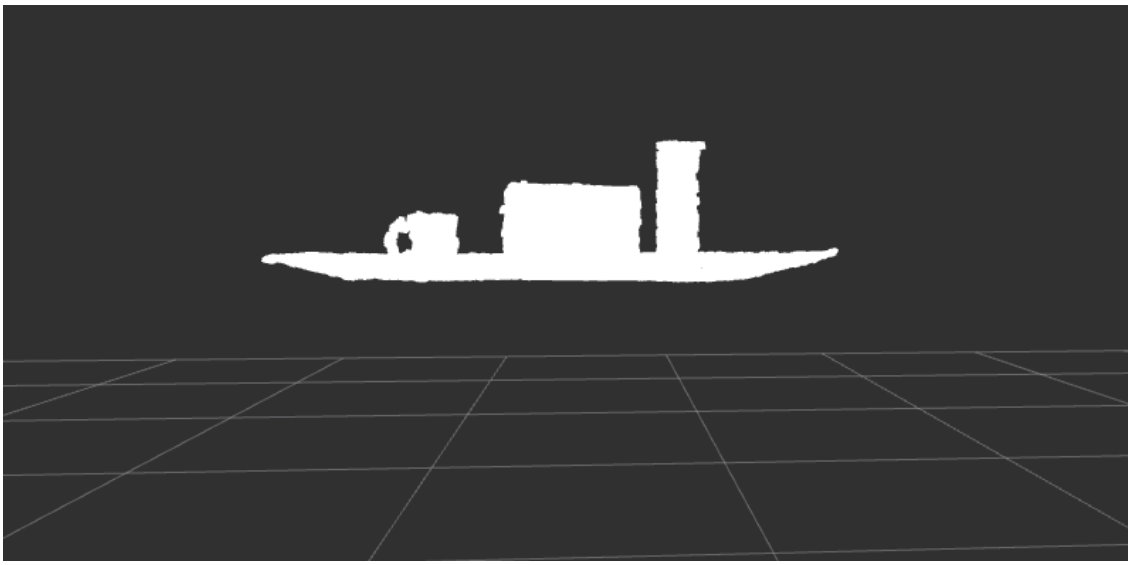

Figura 30. Nube de puntos original vista en rviz.

4.3. Filtrado

Usando la librería PCL se pueden aplicar numerosos filtros que ayuden a realizar un procesamiento de la nube de puntos de forma más efectiva, eliminando puntos que no aporten información en el procesamiento o incluso que se consideren atípicos.

El filtro utilizado con esos fines en este proyecto es el filtro *VoxelGrid*, encargado de reducir la resolución o el número de puntos para una mayor rapidez de procesamiento. Se basa principalmente en configurar un tamaño de *vóxel* o cubo en el que todos los puntos contenidos en el mismo se aproximan al centroide, tal y como se representa en la Figura 26. Este enfoque es más lento que aproximarla al centro del vóxel, pero representa la nube de puntos de manera más precisa.

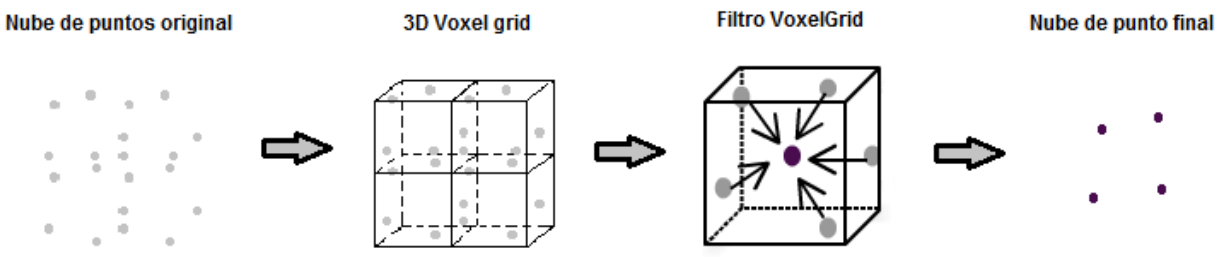

Figura 31. Representación gráfica del funcionamiento teórico del filtro VoxelGrid.

Estos cubos se sitúan a modo de rejilla a lo largo de toda la nube de puntos para conseguir una resolución homogénea y menor a la de partida. Por tanto, resulta importante tener especial cuidado en seleccionar un tamaño del vóxel adecuado para no perder la esencia de la nube de puntos, pero a su vez reduciendo suficiente su resolución para facilitar su procesamiento.

Para hacer uso de ese filtro, hay que crear un objeto de la clase VoxelGrid. A continuación, se le pasa como dato de entrada la nube de puntos anteriormente grabada y se establece el tamaño del vóxel que determinará el número de puntos de la nube final, suponiendo que se tratará de un cubo, es decir, los módulos de las aristas x, y, z del vóxel son iguales y se expresan en metros. Para determinar el tamaño adecuado del vóxel, se realiza una comparación con diferentes valores de arista que determinan la calidad final del filtro.

Partiendo de un filtro de 0.005 (véase Figura 32) no se consigue reducir suficientemente la resolución como para determinar que el tiempo de procesamiento haya disminuido, por lo que se ha descartado esta configuración. Aumentando el tamaño de la arista del vóxel a 0.015 (véase Figura 33), la reducción del número de puntos era tan grande que se perdía información de la escena, puesto que, tras el procesamiento, uno de los objetos dejaba de aparecer. Esto determina que no se trata del valor idóneo para el filtrado. Por tanto, se deduce que el tamaño del vóxel tiene que situarse entre los dos valores anteriormente citados, lo que conllevó a aplicar un filtro de 0.01 (véase Figura 34). Con este filtro se consigue disminuir el número de puntos considerablemente, con la consiguiente reducción de la carga computacional que supone realizar el posterior procesamiento. Además, no se pierde información tras el procesamiento, lo que determina que se puede considerar un filtro adecuado.

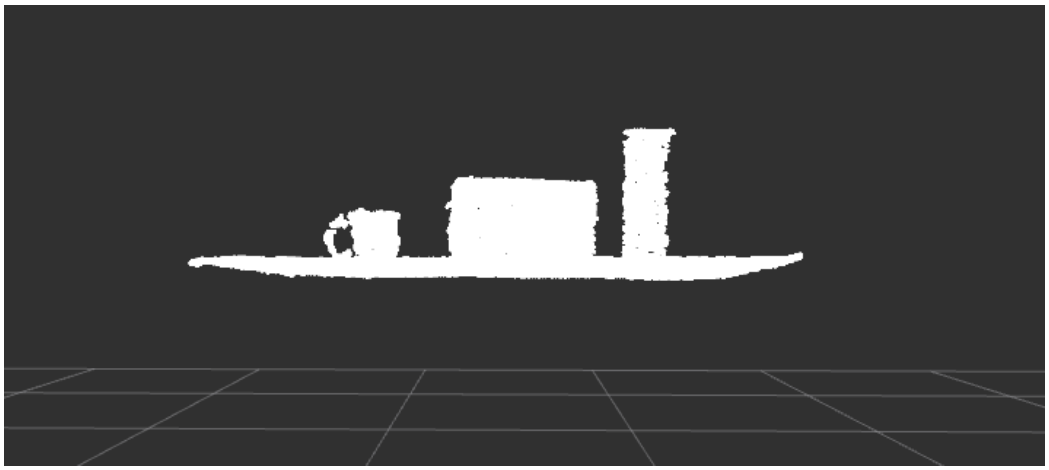

Figura 32. Tamaño del vóxel de 0.005 metros.

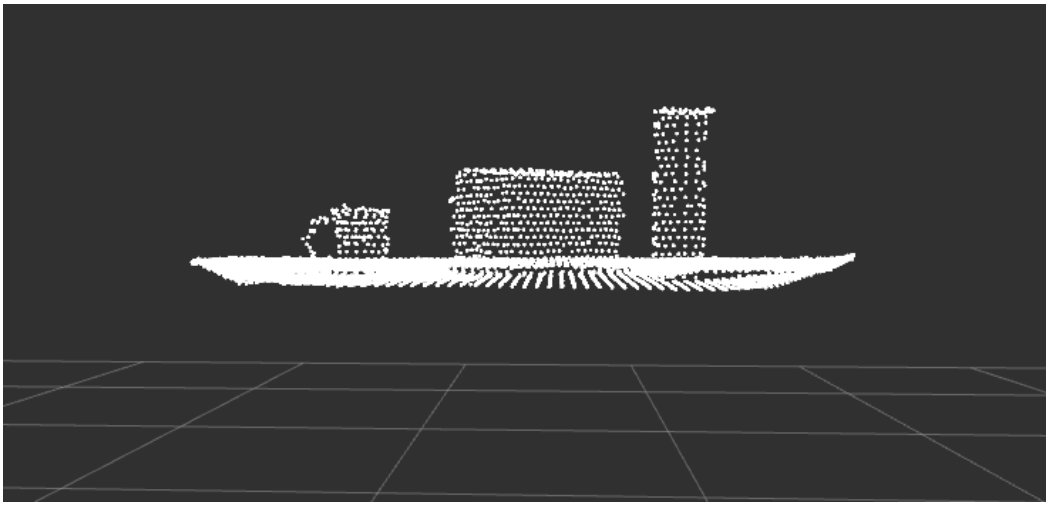

Figura 33. Tamaño del vóxel de 0.015 metros.

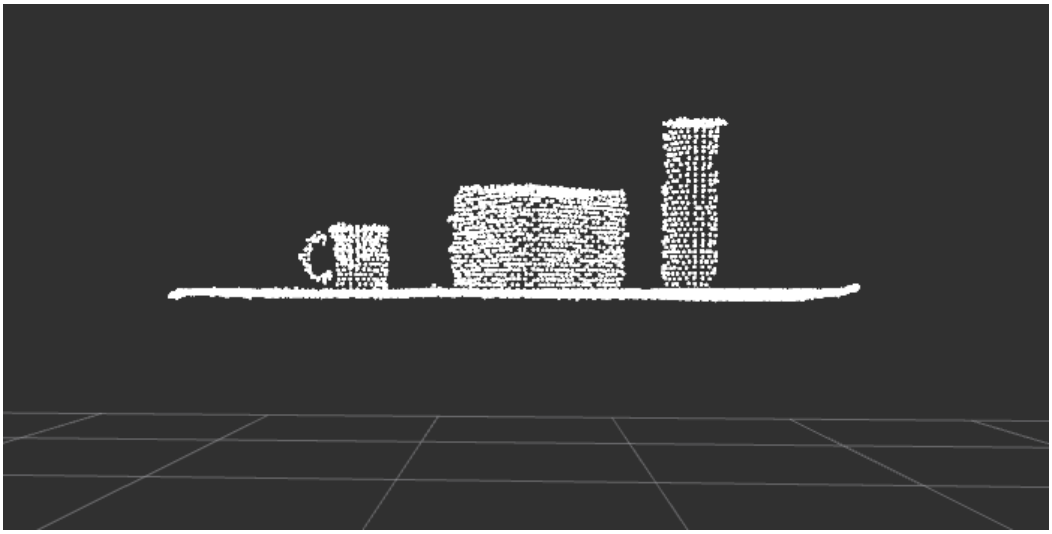

Figura 34. Tamaño del vóxel de 0.01 metros.

Por tanto, la elección del tamaño del vóxel ha sido de 0.01 metros. Para esta configuración, partiendo de una nube de puntos original que contiene 307200 puntos, se consigue reducir el número de puntos en 16232, lo que supone una reducción de más del 90%.

4.4. Segmentación y clustering

La segmentación consiste en encontrar todos los puntos pertenecientes a un mismo objeto. Posteriormente se agrupan y se extraen para diferenciarlos unos de otros, proceso que adopta el nombre de *clustering*.

El funcionamiento genérico de la segmentación se basa en algoritmos de agrupación de píxeles significativos dentro de las regiones que se ajusten a los límites de los objetos. Sin embargo, para poder proceder con la segmentación de la nube ya filtrada, se diferenciará entre la superficie plana de la mesa y los objetos individuales situados encima. Con el objetivo de obtener una rápida visualización de la nube de puntos con los objetos diferenciados, se ha decidido colorear cada elemento con un color diferente.

4.4.1. Superficies planas

Para llevar a cabo la segmentación del plano de la mesa, se ha utilizado el método RANSAC para extraer los puntos pertenecientes a la mesa [21]. Se trata de un método iterativo para estimar parámetros desde un conjunto de datos siguiendo un modelo matemático. Este modelo matemático determina si los datos cuya distribución puede ser explicada bajo el mismo modelo de parámetros, en cuyo caso se le llaman *inliers*. En caso contrario, se le consideran datos atípicos u *outliers*.

Este método devuelve un resultado razonable con una cierta probabilidad que dependerá del número de iteraciones realizadas. Por tanto, requiere de ciertos parámetros de entrada para determinar cuánto de fiable puede ser el resultado, sin excederse para no aumentar la carga computacional. Estos parámetros son:

- Datos: en el caso de este proyecto los datos corresponden con la nube de puntos.
- Modelo: esto es el modelo o geometría que tienen que cumplir los *inliers*. En este caso, una superficie plana.

- n: el número mínimo de datos que se requieren para ajustar el modelo.
- k: el máximo de iteraciones permitidas por el algoritmo que se ha establecido en 100.
- t: referencia umbral que establece la distancia para la que un punto se ajusta al modelo de datos y le ha sido asignado el valor de 0,02 metros.
- d: el número de datos requeridos para afirmar que se ajusta al modelo deseado. Se ha establecido este valor en torno a un 30% de los puntos totales.

Se han determinado esos valores de los parámetros tras realizar varias pruebas, con lo que se concluye que son los más adecuados para este caso.

Como se observa en el diagrama de la Figura 35, el algoritmo aplicado para segmentar la superficie plana correspondiente a la mesa se basa en tratar los puntos de la nube original para determinar aquellos que cumplen con el modelo de un plano siguiendo el método RANSAC. Para ello, se comprueba que cumpla con dicho modelo para posteriormente extraerlo de la nube de puntos y así continuar comprobando los restantes. Una vez hallados todos los *inliers*, se colorean de gris para distinguirlos de los puntos pertenecientes a otros objetos no planos.

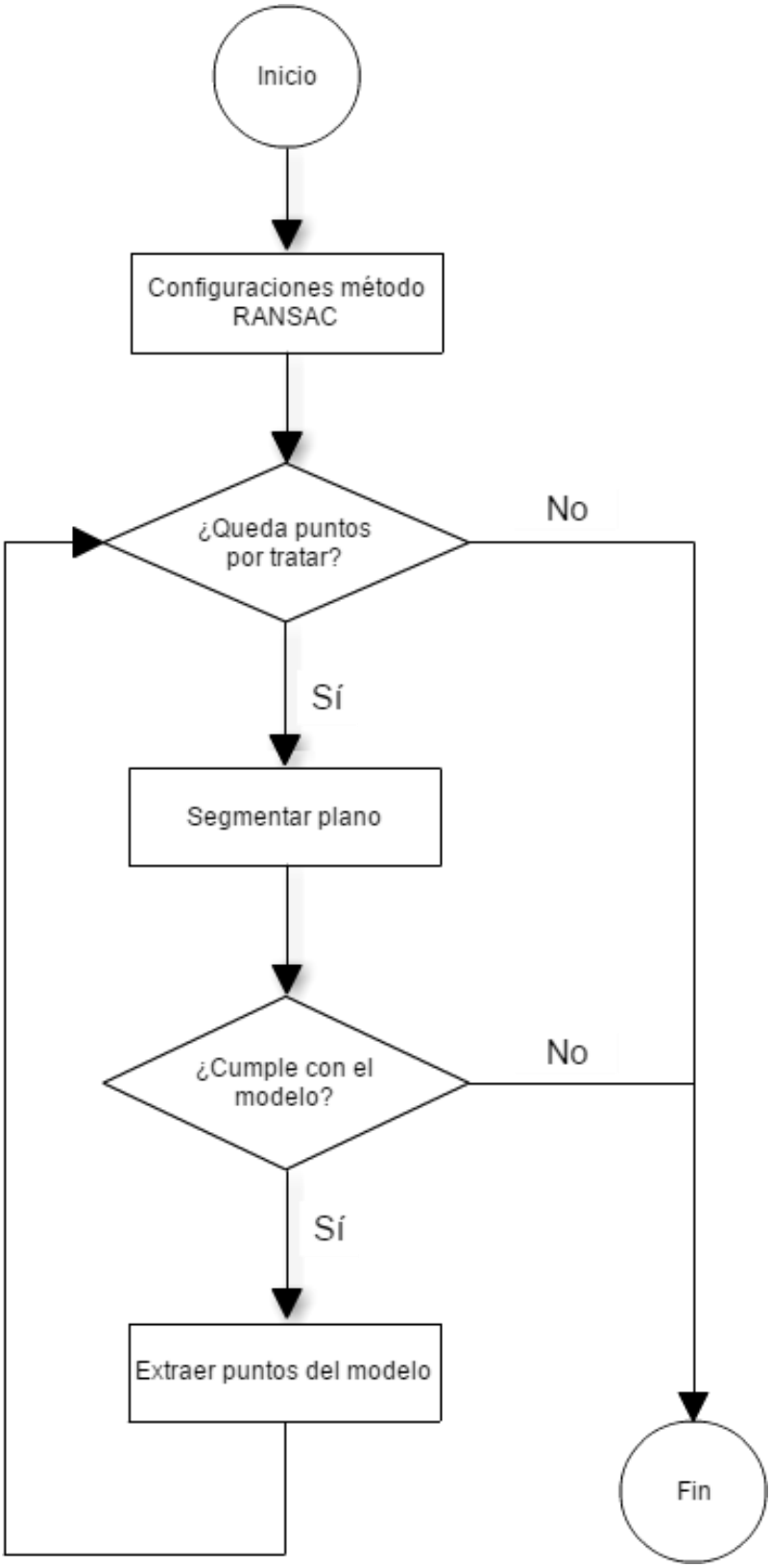

Figura 35. Diagrama de flujo para la segmentación de superficies planas a través del método RANSAC.

En este caso, como se puede ver en la Figura 36, el único elemento que cumple con el modelo propuesto de superficies planas es la mesa.

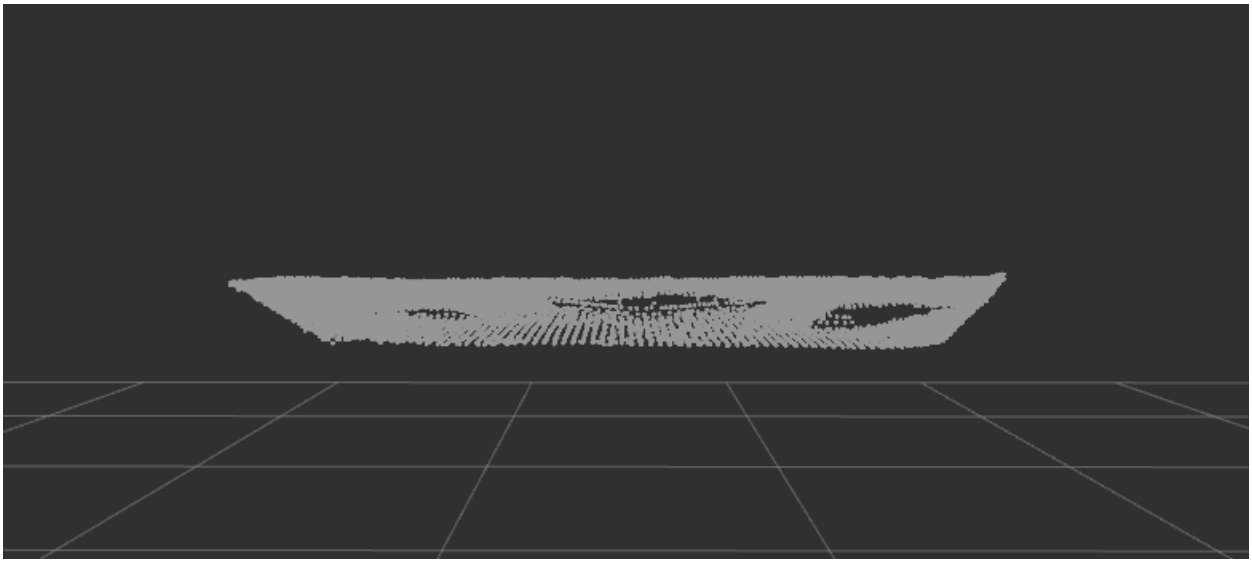

Figura 36. Visualización de la nube de puntos correspondiente a la superficie plana encontrada.

Hay que tener en cuenta que se ha utilizado este método por considerarse una estimación robusta pero lo suficientemente simplificada para que no complicar su adaptación en el proyecto.

4.4.2. Elementos sobre la mesa

A partir de la nube de puntos filtrada y con los puntos que corresponden al modelo de la superficie plana de la mesa extraídos, se continúa con la segmentación del resto de objetos situados sobre la mesa. Es decir, se dispone de una nube de puntos que contiene el resto de puntos por procesar, tal y como se observa en la Figura 37.

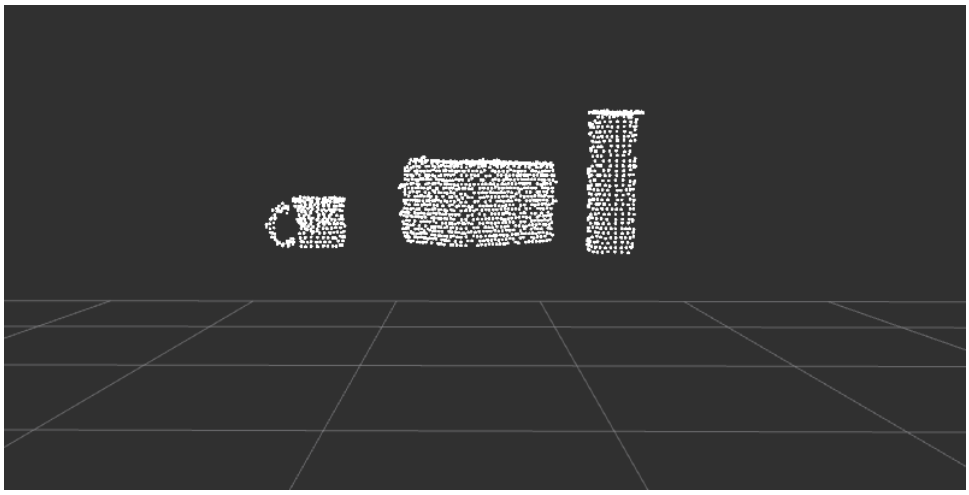

Figura 37. Nube de puntos de los elementos sobre la mesa que quedan por procesar.

En el presente proyecto se ha utilizado el método de extracción de clústers o grupos euclídeos (del inglés, *Extraction Euclidean Clusters*) que se basa en agrupar los datos para poder tratarlos posteriormente por separado [22].

Este agrupamiento puede implementarse a partir de una subdivisión del espacio en cuadrículas 3D usando cajas de tamaño fijo. En este caso, se ha basado en una estructura de árbol kd (estructura de datos de particionado del espacio que organiza los puntos en un espacio euclídeo de k dimensiones) para la búsqueda de los puntos más cercanos. De esta forma, partiendo de un punto de los que quedan por procesar, se delimita el conjunto de puntos vecinos pertenecientes al particionado realizado mediante la estructura de árbol kd. Una vez determinado dicho conjunto, se estudia si pertenecen al mismo elemento comprobando que la distancia entre ellos sea menor a un valor definido previamente, concretamente 0.02 metros. Finalmente, se almacenan los puntos pertenecientes a un mismo elemento en una lista de clústeres, donde se coloreará cada uno de ellos con un color diferente. En concreto, como se tienen tres objetos sobre la mesa, se han coloreado de verde, rojo y azul. Estos pasos algorítmicos realizados se reflejan en el diagrama de la Figura 38.

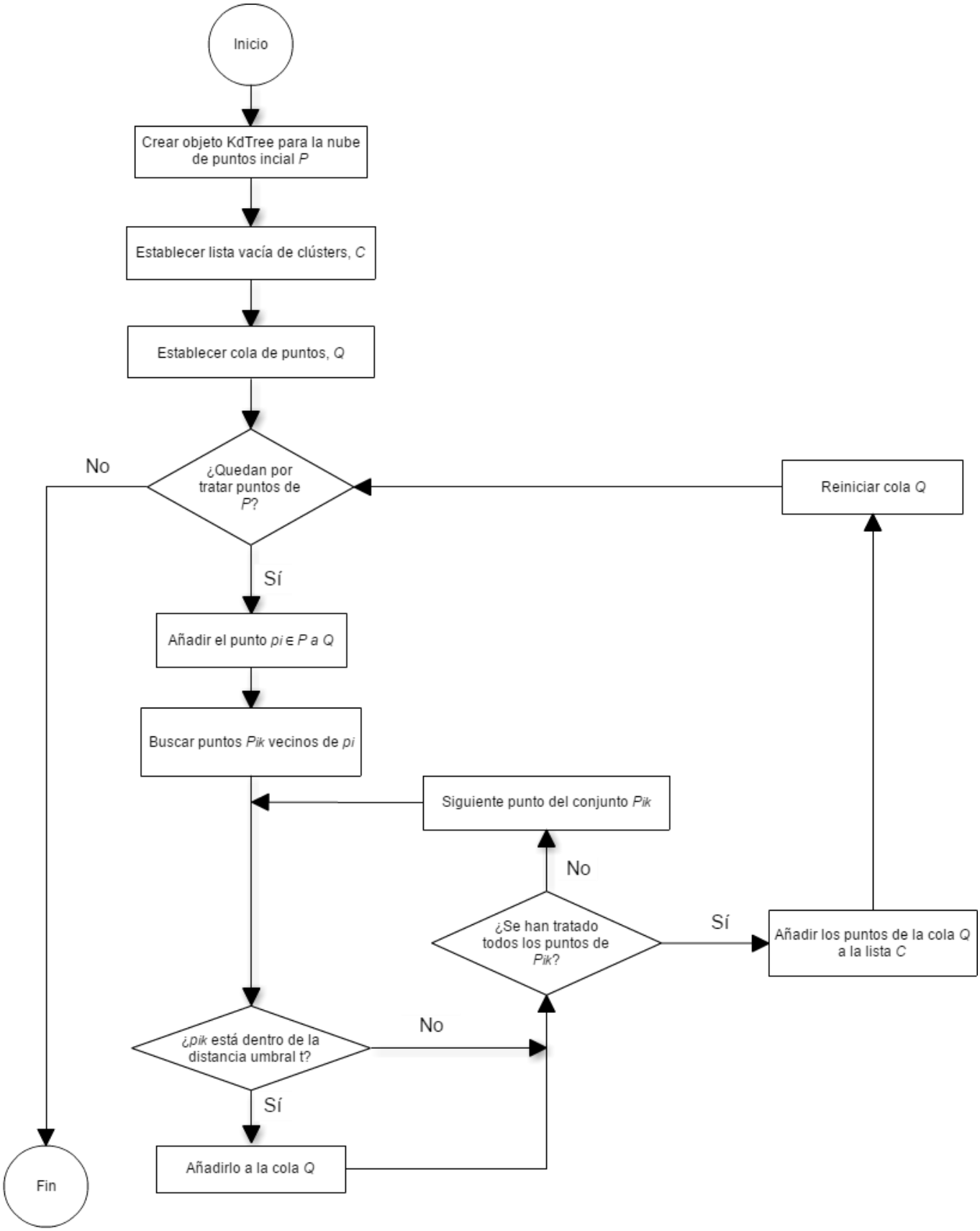

Figura 38. Diagrama de flujo correspondiente a la segmentación de los objetos situados sobre la mesa.

Tras encontrar los puntos de los diferentes objetos de la escena, se añaden a la nube que contenía la mesa. De esta forma, tras finalizar el algoritmo llevado a cabo, se dispone de una nube de puntos en la que cada elemento queda diferenciado del otro a través de los distintos colores.

Para proceder a visualizarlo en Rviz como se observa en la Figura 39, se establece la siguiente configuración:

- *Fixed Frame*: *base_link*
- Añadir un display de tipo *PointCloud2* indicando el *topic* como */output.*

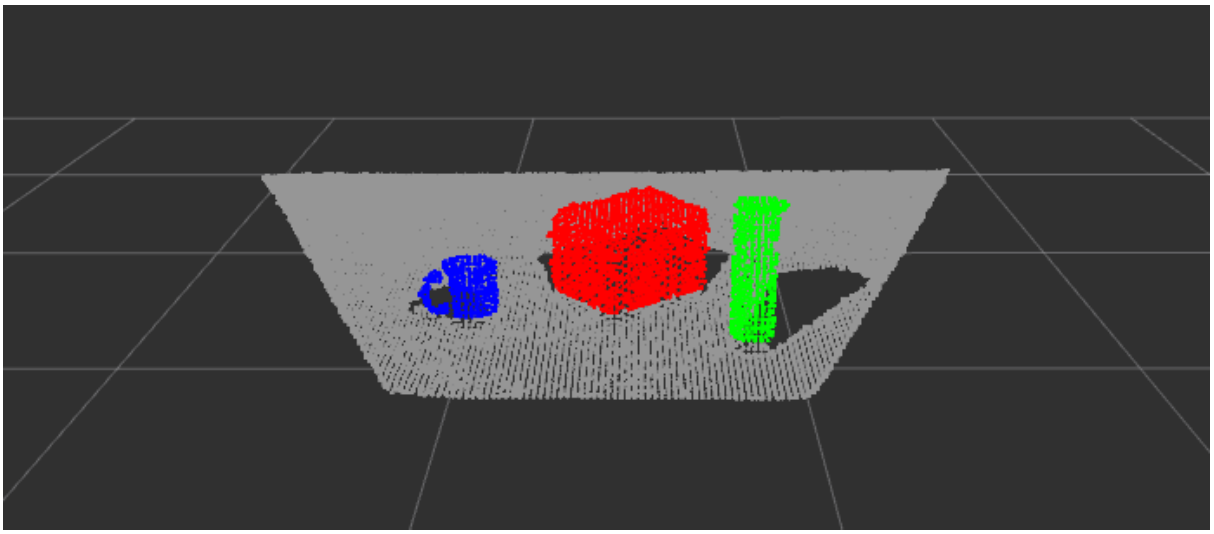

Figura 39. Nube de puntos segmentada y coloreada.

4.5. Modelado

Como se ha comentado en apartados anteriores, es necesario crear un mapa aproximado de la nube de puntos a modo de rejilla 3D basado en una estructura de tipo octree, lo que se adopta el nombre de modelado del entorno. Esto se ha construido gracias a la librería Octomap.

A través del paquete *octomap_server* se construye el mapa de ocupación volumétrica a partir de la nube de puntos como dato de entrada. Como salida proporciona ciertos publicadores con diferentes niveles de información: para visualizarlo en rviz, basta con

suscribirse al topic *occupied_cells_vis* de tipo *MarkerArray*. Sin embargo, para obtener toda la información del espacio libre y ocupado, útil para la planificación de trayectorias del robot, es necesario suscribirse al servicio *octomap_binary*.

Para hacer uso del o*ctomap_server*, se ha añadido un nodo al archivo *launch* con los siguientes parámetros de entrada:

- *resolution*: se corresponde con la resolución o tamaño de las cajas que conforman el *octomap*. El valor está en metros.
- *frame_id*: sistema de referencia global en el que será publicado el octomap, en este caso *base_link*.
- *cloud_in*: nube de entrada que será la que se corresponde con el último topic publicado tras realizar la segmentación, es decir, equivaldrá al topic *output*.

```
<node name="octomap_server" pkg="octomap_server" type="octomap_server_node">
   <param name="resolution" value="0.03" />
   <param name="frame_id" type="string" value="base_link" />
   <remap from="cloud_in" to="output"/>
</node>
```

Tras realizar una comparación con diferentes resoluciones, se descartó un tamaño de 0.05 metros (véase Figura 40), ya que se corresponde con un modelo rudimentario que dista de la escena original. Por otro lado, también se ha descartado un tamaño menor, 0.01 metros (véase Figura 41), por tener demasiada resolución, innecesaria para los fines de este proyecto y que pueden suponer una ralentización de su tratamiento. Por tanto, la configuración más acertada es la de 0.03 metros (véase Figura 42), puesto que se consigue un equilibrio entre la resolución y la posible pérdida de información, con la consiguiente aproximación al modelo de escena original.

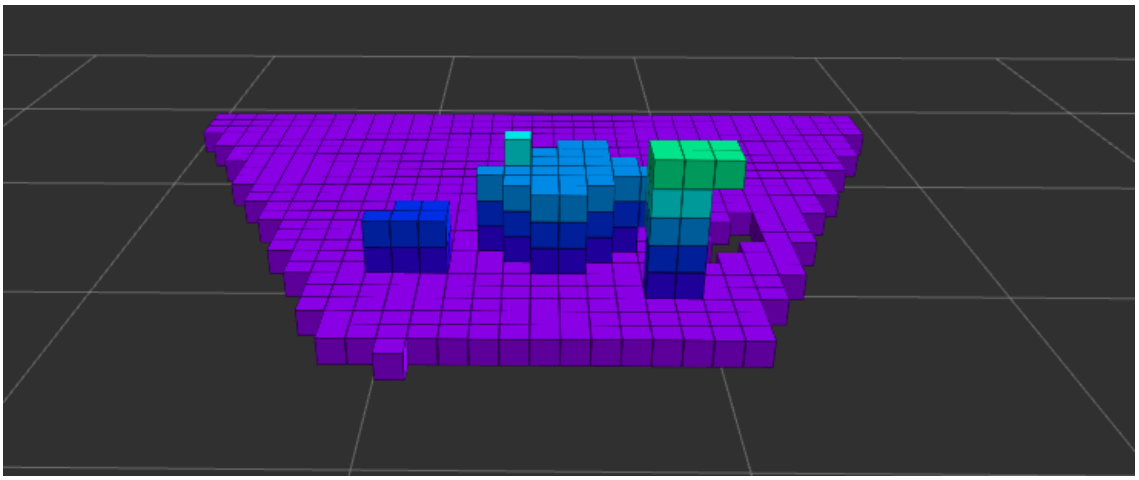

Figura 40. Octomap con resolución 0.05.

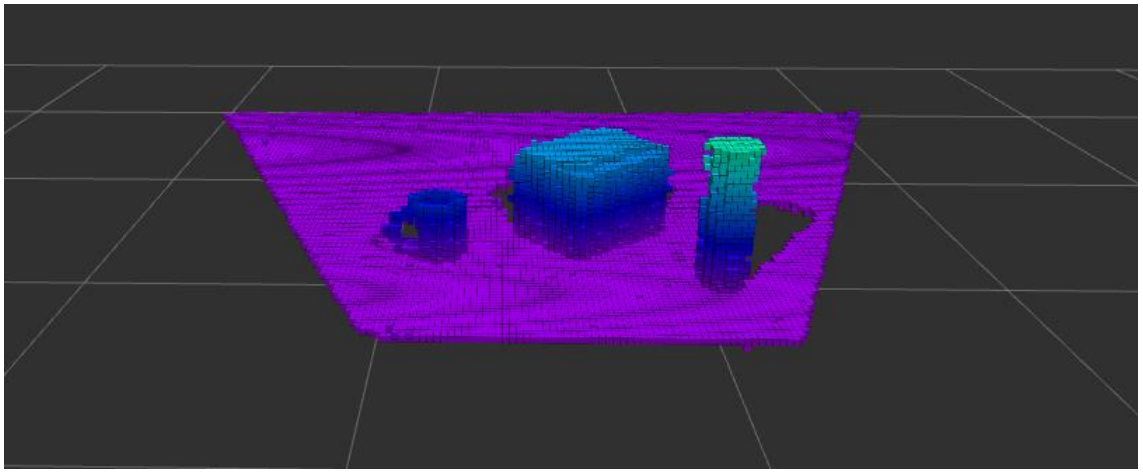

Figura 41. Octomap con resolución 0.01.

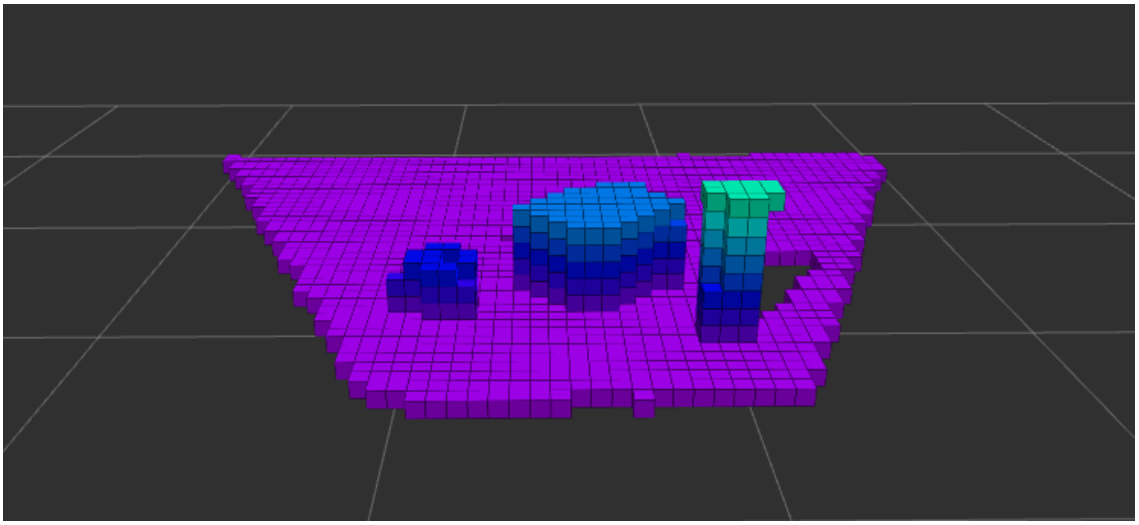

Figura 42. Octomap con resolución 0.03.

Una vez configurado las propiedades para el Octomap, se ha llegado a cumplir con el tercer bloque del proyecto: todo lo relativo al procesamiento de la nube de puntos. Como se muestra en la Figura 43, se ha conseguido establecer la escena siguiendo los requisitos para la posterior planificación del robot, que consisten en separar los distintos objetos de la escena y modelar la misma para que pueda ser usada en MoveIt!

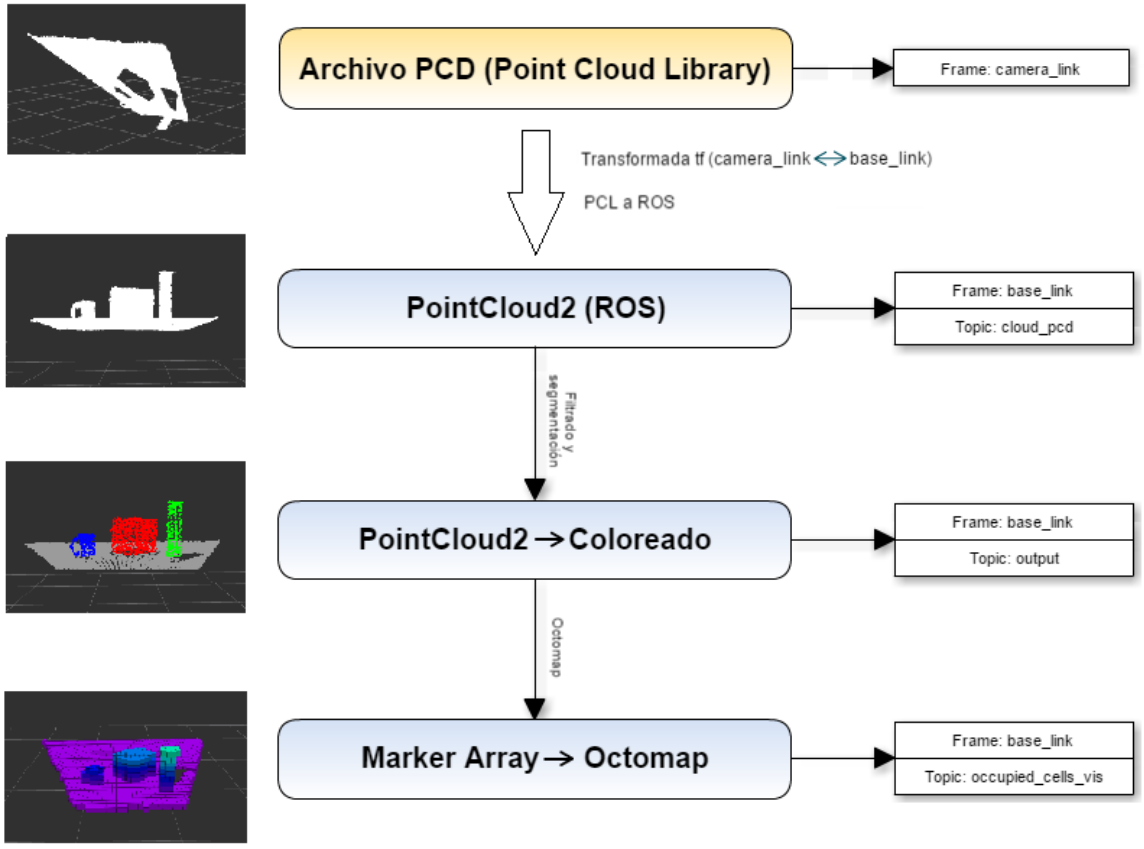

Figura 43. Diagrama de las fases realizadas para el tratamiento de la nube de puntos.

4.6. Manfred en MoveIt!

Para configurar un robot en MoveIt! es necesario describir el modelo URDF (*Unified Robot Description Format*), que consiste en un archivo XML cuyo contenido incluye toda la información del robot para poder ser representado.

El modelo URDF de Manfred ha sido desarrollado por el alumno de la Universidad Carlos III de Madrid, Raúl Merino [23]. Por tanto, ya se dispone de los modelos 3D de todas las

partes del brazo, así como su configuración en MoveIt! a través de un archivo SRDF (*Semantic Robot Description Format*) que especifica aspectos como la descripción de articulaciones o la información necesaria para la comprobación de colisiones.

Para poder visualizarlo, se parte de las carpetas con dichos archivos que contienen las especificaciones necesarias y a partir de ahí se continúa con el desarrollo del proyecto. A nivel práctico hace falta incluir el *launch* que contiene toda la configuración (*demo.lauch*) al *launch* correspondiente con este segundo bloque del proyecto (*move_arm.launch*):

```
<include file="$(find manfred_moveit_config)/launch/demo.launch"/>
```

Dicho launch lanza tanto rviz como las configuraciones de Manfred para ser visualizado, tal y como se puede observar en la Figura 44.

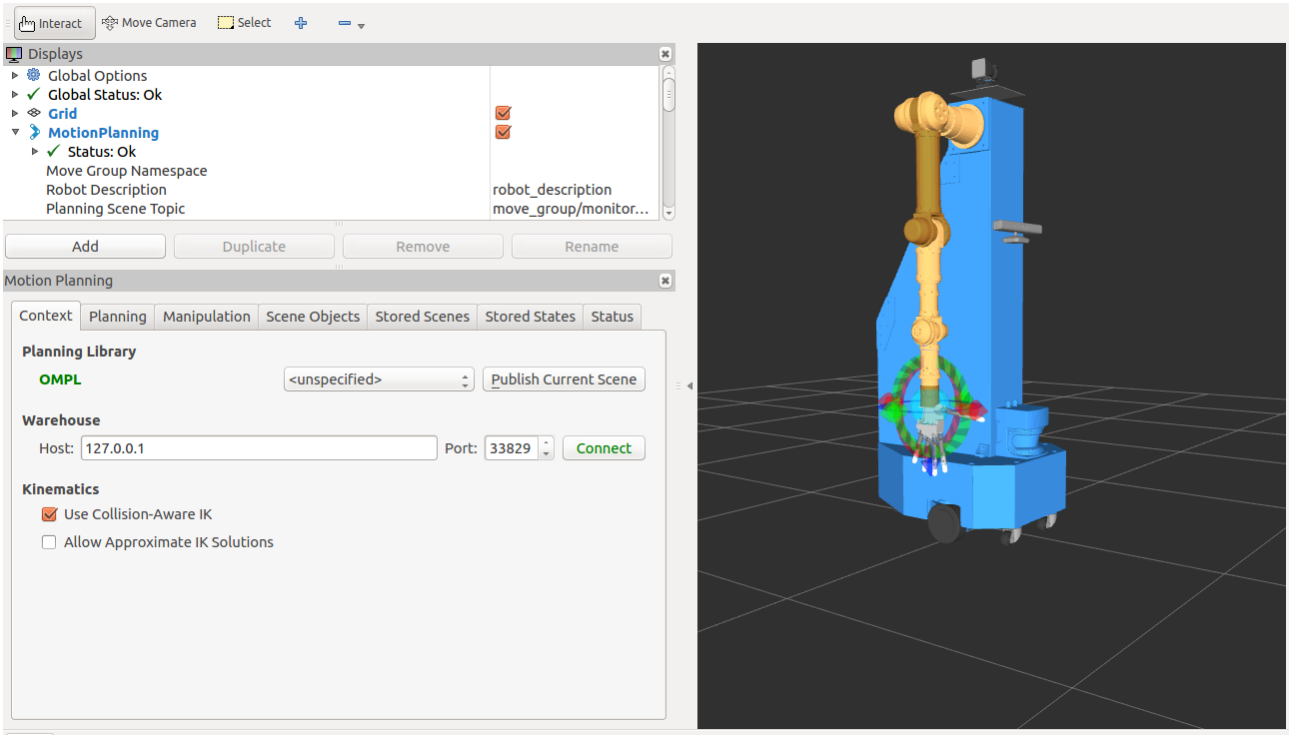

Figura 44. Visualización de Manfred en MoveIt!

4.7. Configuración de la escena

En este apartado se explicará los pasos a seguir para configurar el entorno 3D de tal manera que el robot pueda navegar por el mismo. Para ello, hay que desarrollar un nodo que parta del *octomap* obtenido anteriormente y a través del cual se genere un mapa de colisiones listo para ser interpretado por Manfred.

Dentro de las configuraciones a tener en cuenta, se necesita crear un cliente que solicite información al servidor *octomap_binary* para obtener el mapa de ocupación en binario del espacio, como se ha comentado en el apartado 4.5. Con el fin de que MoveIt! sea capaz de interpretar esa información, se modela a un tipo de dato *Planning Scene* que contiene la información de las celdas ocupadas, permitiendo adaptar la escena para que se puedan detectar las colisiones y así evitar los choques entre Manfred y los objetos de la misma. Afortunadamente, MoveIt! está diseñado para comprobar esas colisiones usando el objeto *CollisionWorld* del mensaje *Planning Scene* configurado.

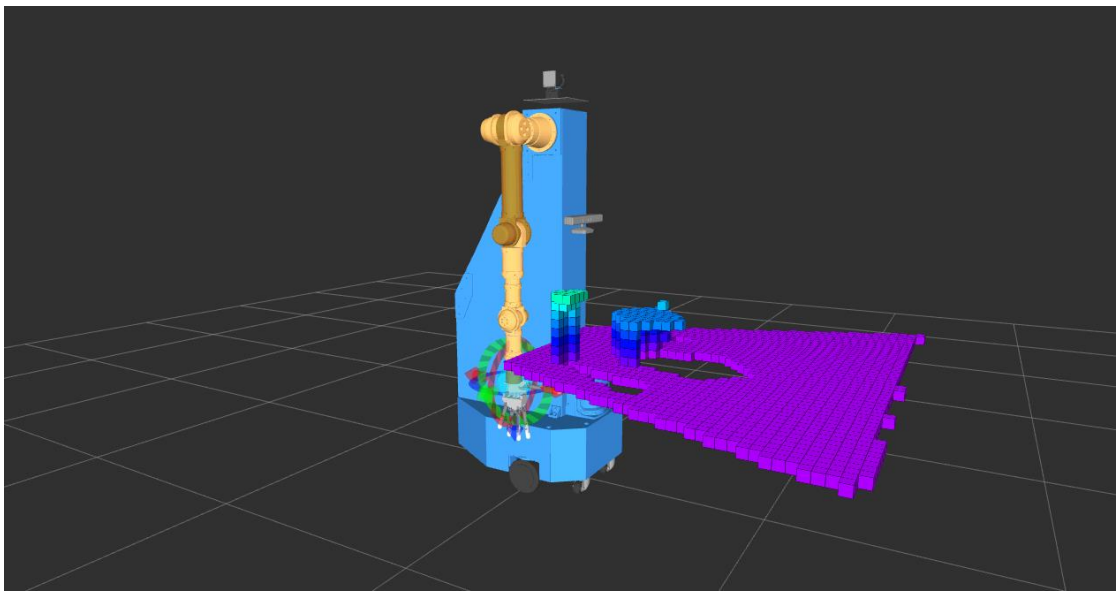

Figura 45. Visualización de la escena en MoveIt! preparada para detectar colisiones.

Por tanto, tras realizar las configuraciones descritas, se puede proceder a la visualización de la escena sobre MoveIt! (véase Figura 45) para, posteriormente, planificar a Manfred para cumplir con los objetivos del proyecto.

4.8. Planificación de trayectorias

Tras realizar el software necesario para que se establezcan las colisiones del robot con el entorno, se puede proceder a planificar el movimiento del brazo de Manfred, con el objetivo de lograr el agarre de los objetos.

La planificación de trayectorias en MoveIt! está disponible tanto a través de su interfaz como por código. En realidad, basta con indicarle un punto en concreto al efector final para que MoveIt! calcule una trayectoria que evite los obstáculos. De este modo, a través de la interfaz, se puede guiar al brazo de Manfred para que alcance los objetos, arrastrándolo hasta el punto deseado y posteriormente pulsando sobre el botón *Plan and Execute* de la pestaña *Planning,* como se representa en la Figura 46.

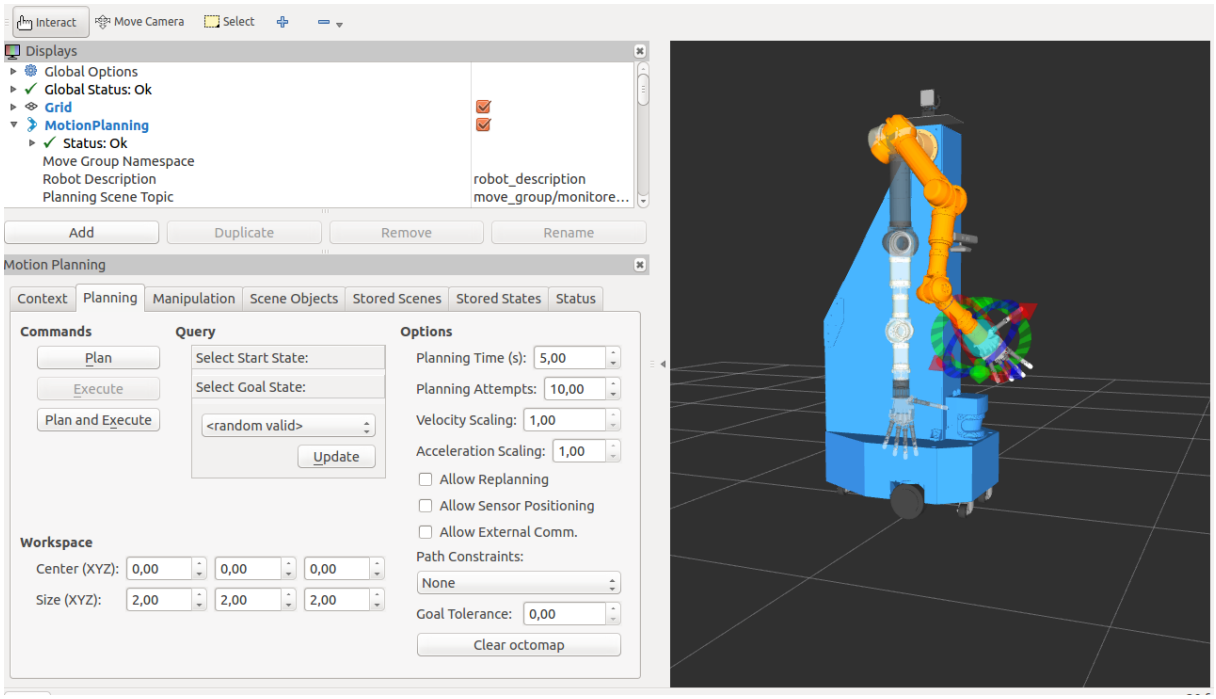

Figura 46. Pruebas de planificación de trayectorias del brazo de Manfred a través de la interfaz de MoveIt!

Esa determinación del objetivo a planificar se traduce en código como un punto definido con tres coordenadas (x, y, z) correspondientes con la posición, y un cuaternio (x, y, z, w) que representan la orientación y rotación del efector final, lo que se conoce como *goal state* en MoveIt!

Para hacer efectivo el movimiento a esos puntos de referencia definidos, MoveIt! es la encargada de establecer el camino a seguir para alcanzar el objetivo a través de los planificadores de movimientos internos. Es decir, MoveIt! incluye algoritmos de cálculo de trayectorias parametrizables teniendo en cuenta los límites máximos de velocidad y aceleración impuestas sobre las articulaciones individuales. Estos límites se leen desde un archivo especificado por cada robot que, en este caso, ya venían establecidos junto con todas las configuraciones de Manfred.

Finalmente, una vez se ha programado el punto a alcanzar por Manfred, se pueden establecer diversos objetivos definiendo todos los puntos que se requieran. En este caso, se han propuesto dos puntos correspondientes a dos de los objetos situados sobre la mesa (véase Figura 47 y Figura 48).

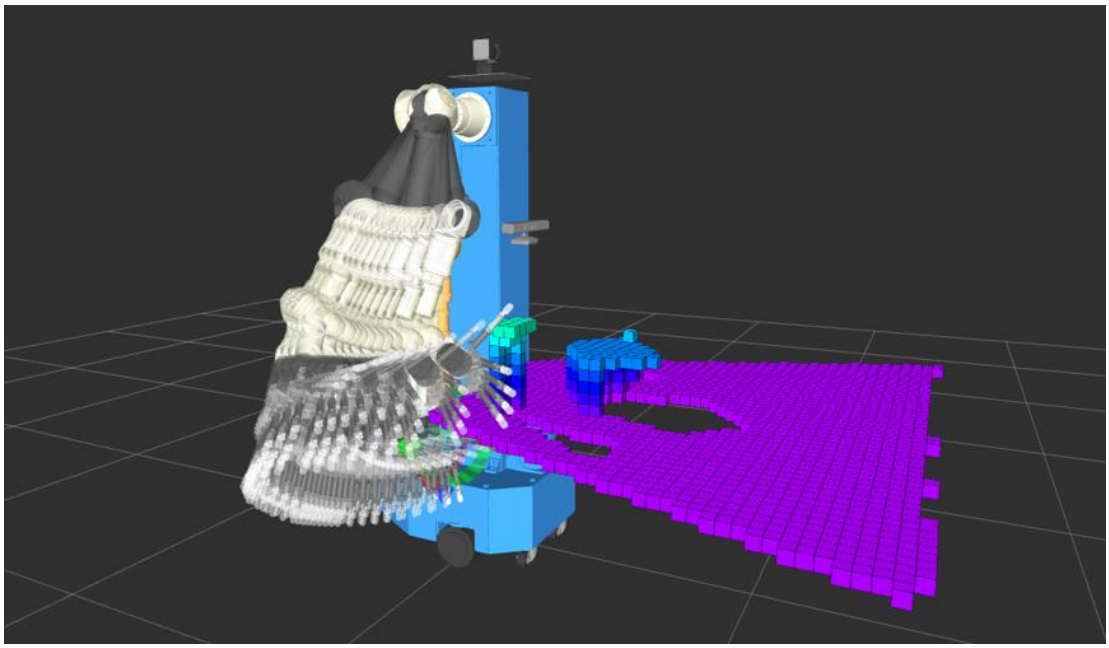

Figura 47. Planificación del brazo de Manfred para que alcance el primer objeto de su derecha.

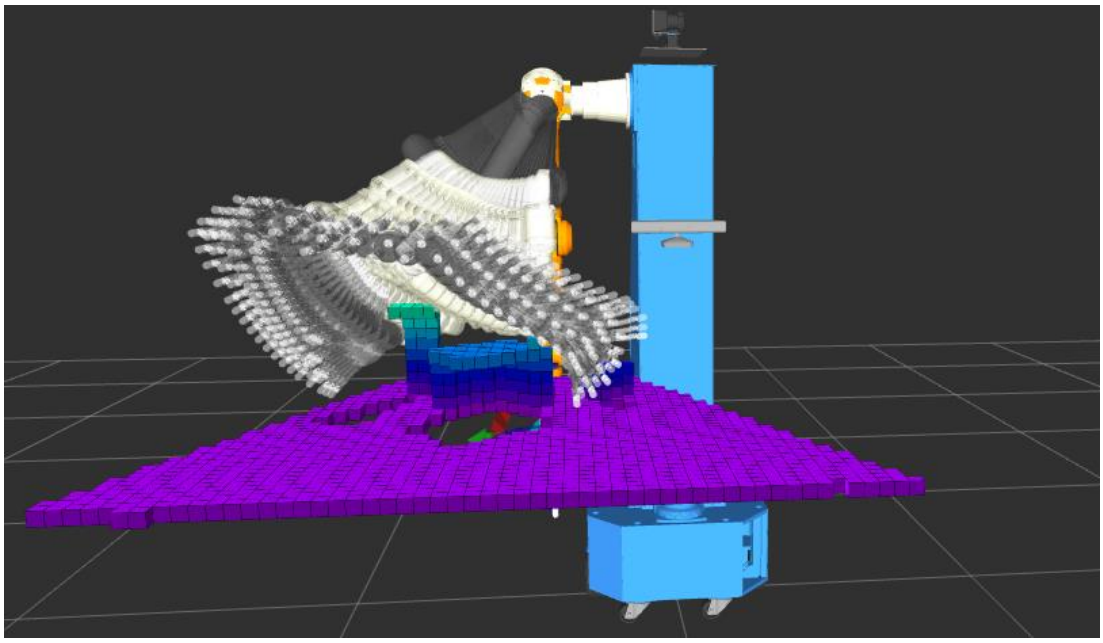

Figura 48. Planificación del brazo de Manfred para que alcance el primer objeto de su izquierda.

Los puntos orientativos de aproximación de los objetos, se han establecido determinando una posición próxima al objeto en cuestión. Esta determinación se realiza a través de la interfaz de MoveIt!, de tal forma que primero se guía al robot hasta *goal state* deseado y a continuación se obtienen los valores de posición y orientación del punto alcanzado. Tras la obtención de dichos datos, se procede a incluirlos en la programación para que realice el movimiento de forma automática.

Con esto, se finaliza la última fase del proyecto que incluye todo lo relativo al entorno de MoveIt! (véase Figura 49).

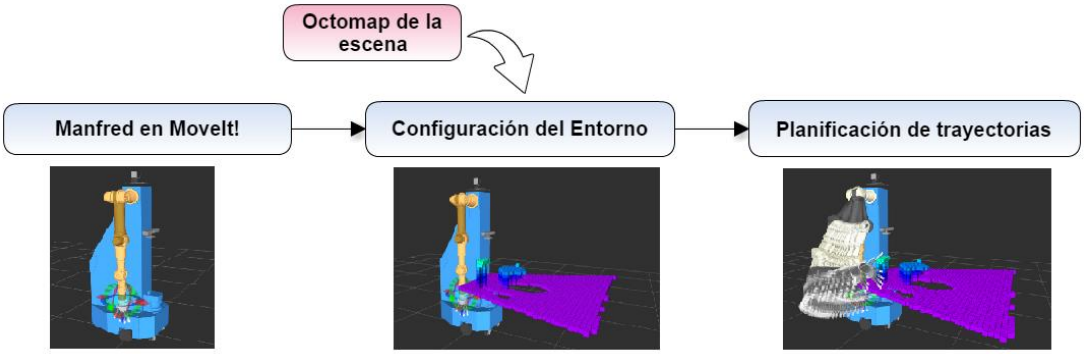

Figura 49. Diagrama de las fases realizadas para desarrollo en MoveIt!

4.9. Pruebas en robot real

Con el objetivo de comprobar el correcto funcionamiento del software desarrollado en este proyecto, se ha decidido probar las trayectorias que se obtienen a través de MoveIt!, en el robot real Manfred.

Para poder comparar la parte de simulación con la real, se prepara una escena similar a la tratada en este proyecto, en la que se tiene el bote a la derecha del robot situado sobre una mesa (Figura 50, izquierda). Para eliminar el resto de información del entorno que no interesa, se ha filtrado la nube de puntos a 85 cm sobre el eje z de la cámara del robot (Figura 50, derecha).

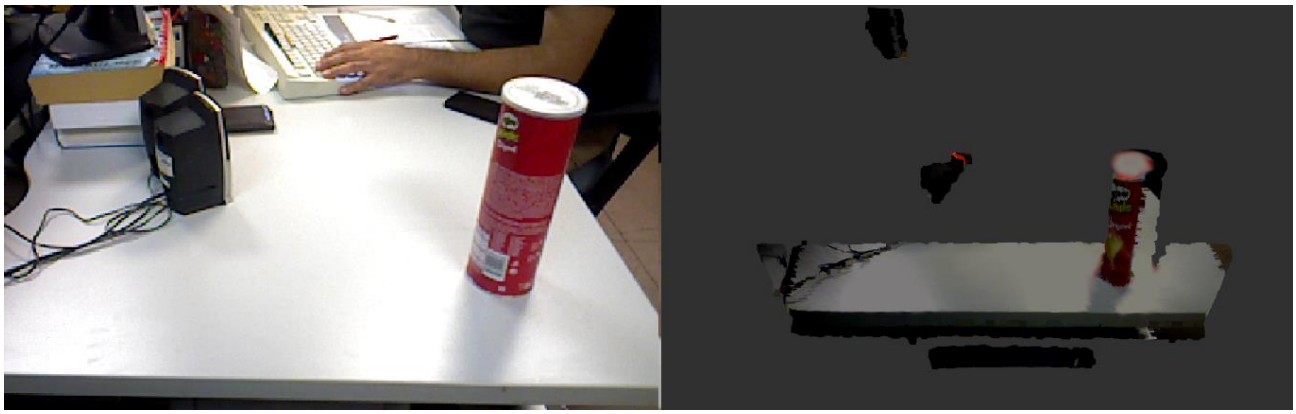

Figura 50. Nube de puntos de la escena para las pruebas reales sobre Manfred.

A partir de esa nube de puntos y del software diseñado para configurar una escena preparada para ser interpretada por MoveIt!, se procede a obtener las trayectorias que se generan para que la mano se aproxime al objeto en cuestión (véase Figura 51).

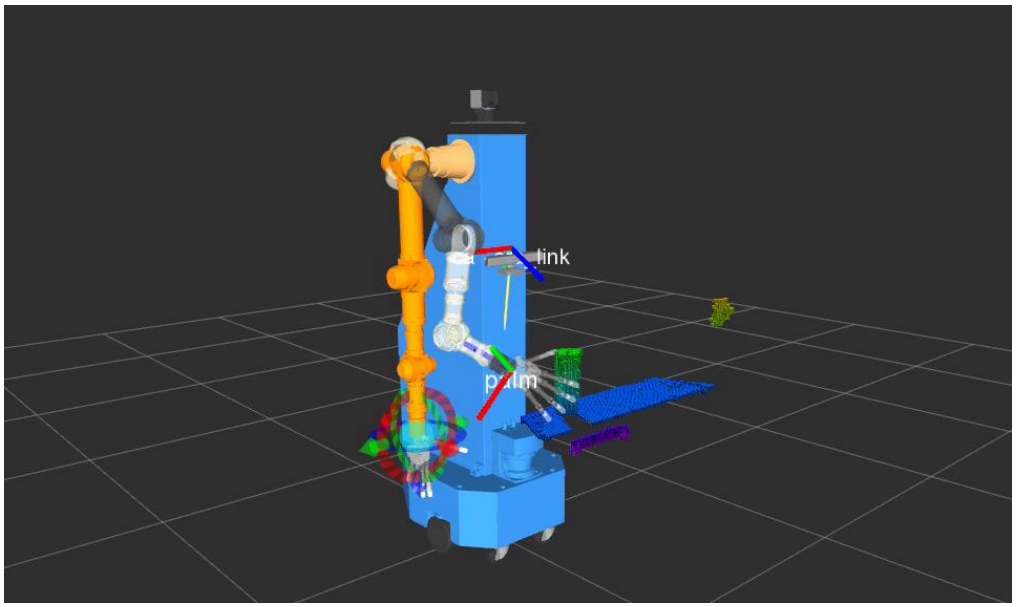

Figura 51. Goal State para calcular la trayectoria de aproximación al objeto.

Finalmente, una vez se ha planificado la trayectoria de aproximación que llevaría el brazo del robot, se obtienen los puntos de paso generados por MoveIt! para ejecutarlos en el robot Manfred, como se observa en la Figura 52.

Figura 52. Ejecución de la trayectoria en el robot Manfred.

5. Gestión del trabajo

En este apartado, se estudiarán los aspectos relacionados con la gestión del trabajo, es decir, la planificación para llevar a cabo el proyecto, así como un presupuesto estimado para su elaboración.

5.1. Planificación

Para establecer una planificación de las tareas a desarrollar para alcanzar los objetivos expuestos en este proyecto, se ha realizado un diagrama de Gantt (véase Figura 53). Se han agrupado en función de las fases establecidas en el proyecto:

- Planteamiento del problema
- Preparación del proyecto
- Tratamiento de la nube de puntos
- Desarrollo en MoveIt!
- Memoria

Todas las tareas de realización de la parte técnica del proyecto se consideran críticas, ya que se han tenido que realizar secuencialmente. En cuanto a la elaboración de la memoria, se ha establecido como fecha de inicio el fin de las tareas relacionadas con el tratamiento de la nube de puntos, ya que se considera que el proyecto está suficientemente avanzado como para empezar con dicha tarea.

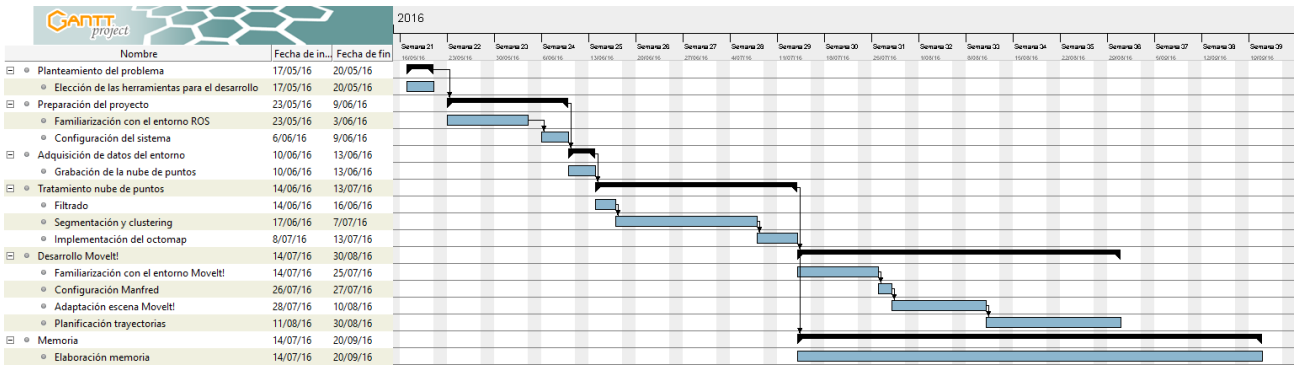

Figura 53. Planificación del proyecto a través de un diagrama de Gantt.

Para la elaboración de este diagrama, se ha excluido los fines de semana (sábados y domingos) pero no los festivos. Además, se han tenido en cuenta las horas de supervisión

por parte del tutor del proyecto. En total, las horas dedicadas suman 364 horas, de las cuales 40 horas han sido del tutor.

5.2. Presupuesto

El coste del proyecto se ha clasificado en dos partes: costes laborales y costes materiales. En cuanto al primero de ellos (véase Tabla 1), se ha tenido en cuenta las horas empleadas por todos los participantes en este proyecto.

NOMBRE	CATEGORÍA	SALARIO (€/h)	HORAS	COSTE TOTAL (€)
NADIA FERREYRA	Ingeniero junior	16	324	5.184
DIRECTORES DEL PROYECTO	Ingeniero senior	35	40	1.400
			TOTAL	6.584

Tabla 1. Resumen de los costes laborales.

En lo relativo al coste material, cabe destacar que todo el software utilizado es de código abierto[2], lo que implica que no ha generado costes. Por otro lado, la Kinect y otros gastos indirectos, sí tienen un coste, tal y como viene reflejado en la Tabla 2. Además, se ha establecido un porcentaje de uso para cada uno de ellos, teniendo en cuenta que, tanto el sistema operativo Linux como otros gastos indirectos, también se han utilizado para otros fines, así como la Kinect se considera que su ciclo de vida da para diez usos.

DESCRIPCIÓN	CATEGORÍA	COSTE (€)	% DE USO	COSTE ATRIBUIBLE (€)
POINT CLOUD LIBRARY	Software	0	100	0
ROBOT OPERATING SYSTEM	Software	0	100	0
SISTEMA OPERATIVO (LINUX)	Software	0	50	0
KINECT	Hardware	100	10	10
OTROS (LUZ, INTERNET, ORDENADOR)	Otros	700	20	140
			TOTAL	150

Tabla 2. Resumen de los costes materiales

Por tanto, el coste total que conlleva la realización del presente proyecto asciende a 6.734 euros.

[2] El software de código abierto, o software libre, permiten utilizar, escribir, modificar o redistribuir el código de forma gratuita.

6. Conclusiones y trabajos futuros

Una vez estudiado el desarrollo de este proyecto, se puede concluir que se ha cumplido con el objetivo inicialmente propuesto, que consistía en obtener una trayectoria de aproximación del brazo del robot para alcanzar uno de los objetos dispuestos sobre una mesa. Además de probar desde la interfaz de MoveIt! el software desarrollado, se comprobó que las trayectorias generadas se pueden aplicar al robot Manfred, por lo que las etapas establecidas para llevarlo a cabo se han conseguido realizar satisfactoriamente:

- Configuración del sistema. Se preparó todo lo necesario para disponer de un entorno de desarrollo listo para proceder con el resto de fases, tanto a nivel de paquetes como de la relación entre los sistemas de referencia para la correcta visualización de la nube de puntos.
- Adquisición de datos del entorno. Para ello, se realizó una grabación de la nube de puntos de la escena a tratar.
- Tratamiento de la nube de puntos. Se consiguió filtrar y segmentar satisfactoriamente, de tal forma que se obtuvo una nube de puntos en la que cada objeto se representa de un color diferente. Además, se modeló para su posterior uso en MoveIt!
- Desarrollo en MoveIt! En esta fase, también se cumplió con las fases establecidas, puesto que se logró configurar un entorno adaptado para que el robot Manfred pueda planificar trayectorias evitando los obstáculos. Aun así, no se pudo conseguir que la obtención de los puntos de aproximación de cada objeto se hiciese de forma automática, puesto que el hecho de que MoveIt! necesite, además de las posiciones (x, y, z), las orientaciones (x, y, z, w) de cada punto para generar una trayectoria libre de errores, hizo que se complicase la obtención de los mismos. Por ello, para determinar las orientaciones con la que llegaría el efector final del robot al punto establecido, se realizó mediante la interfaz de MoveIt!

A pesar de que los objetivos se han cumplido, existen una serie de posibles mejoras que se podrían implementar para obtener un software más sofisticado que sea capaz de realizar tareas más complejas. En primer lugar, cabe destacar que para este proyecto no se ha tenido en cuenta el movimiento de las falanges de las manos, por lo que en todas las planificaciones se observa la mano abierta. Esto no se ha implementado por su complejidad (al tener que incluir el control de más articulaciones) y por no estar estrictamente dentro de los objetivos. Por tanto, una mejora sería implementar el movimiento de la mano para que la arquitectura del agarre de un objeto sea más aproximada.

En segundo lugar, y relacionado con la anterior propuesta, se podría implementar una mejora que permita a Manfred poder agarrar un objeto para que pueda trasladarlo de lugar, lo que se conoce como *pick and place* [24]. El realizar esa tarea conllevaría desarrollar un software que permita encontrar los posibles puntos de agarre del objeto en cuestión y posteriormente ser capaz de determinar la superficie de contacto necesaria entre el robot y el objeto para que se considere que está sujeto.

8. Referencias

[1] L. Moreno. “MANFRED-2”, *Roboticslab*, 19-12-2014. [En línea]. Disponible en: http://roboticslab.uc3m.es/roboticslab/robot/manfred-2. [Consulta: 7 julio 2016].

[2] D. Blanco et al., “Manfred: Robot Antropomórfico de servicio fiable y seguro para operar en entornos humanos”, *Revista Iberoamericana de Ingeniería Mecánica,* vol. 9, nº 3, pp. 33-48, 2005.

[3] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2011.

[4] B. Curless, “From range scans to 3D models”, *ACM SIGGRAPH Computer Graphics,* vol. 33, nº 4, pp. 38-41, 2000.

[5] W. Boehler, G. Heinz y A. Marbs, “The potencial of non-contact close range laser scanners for cultural heritage recording”, *CIPA.* [En línea]. Disponible en: http://cipa.icomos.org/fileadmin/template/doc/potsdam/2001-11-wb01.pdf. [Consulta: 27 julio 2016].

[6] J. C. Torres, “Digitalización 3D”, *LSI*. [En línea]. Disponible en: http://lsi.ugr.es/~jctorres/MasterDesarrolloSoftware/D3D_1.pdf. [Consulta: 30 julio 2016].

[7] L. Cruz, “Kinect and RGBD Images: Challenges and Applications”, *Academia*. [En línea]. Disponible en: http://www.academia.edu/15384535/Kinect_and_RGBD_Images_Challenges_and_Applications. [Consulta: 1 agosto 2016].

[8] A. F. Calvo Salcedo, A. B. Martínez y E. A. Q. Salazar, “Procesamiento de nubes de puntos por medio de la librería PCL”, *Scientia et Technica,* vol. 2, nº 52, pp. 136-142, 2012.

[9] “El lenguaje del cálculo técnico”, *MathWorks*, [En línea]. Disponible en: http://es.mathworks.com/products/matlab. [Consulta: 15 agosto 2016].

[10] “Software de Desarrollo de Sistemas NI LabVIEW”, *National Instruments*, [En línea]. Disponible en: http://www.ni.com/labview. [Consulta: 15 agosto 2016].

[11] A. O. Baturone, *ROBÓTICA. Manipuladores y robots móviles*, Marcombo, 2001.

[12] I. A. Șucan, M. Moll y L. E. Kavraki, “The Open Motion Planning Library”, *IEEE Robotics & Automation Magazine,* vol. 19, nº 4, pp. 72-82, 2012.

[13] R. Diankov, “Automated Construction of Robotic Manipulation Programs”, Tesis doctoral,

Robotics Institute, Carnegie Mellon University, Pensilvania, Estados Unidos, 2010. [En línea]. Disponible en: http://www.programmingvision.com/rosen_diankov_thesis.pdf. [Consulta: 20 agosto 2016].

[14] I. A. Sucan y S. Chitta, *MoveIt!*, [En línea]. Disponible en: http://moveit.ros.org. [Consulta: 25 septiembre 2016].

[15] Y. Wu et al, "Improving human-robot interactivity for tele-operated industrial and service robot applications", *Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics Automation and Mechatronics (RAM) 2015 IEEE 7th International Conference on*, pp. 153-158, 2015.

[16] "Integration with Other Libraries", *ROS*, [En línea]. Disponible en: http://www.ros.org/integration/. [Consulta: 20 agosto 2016].

[17] M. Quigley et al, "ROS: an open-source Robot Operating System", *ICRA Workshop on Open Source Software*, 2009.

[18] *ROS*, [En línea]. Disponible en: http://www.ros.org/. [Consulta: 10 septiembre 2016].

[19] T. Foote, "tf: The transform library", *Technologies for Practical Robot Applications (TePRA) 2013 IEEE International Conference on*, pp. 1-6, 2013.

[20] A. Hornung, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees", *Autonomous Robots,* vol. 34, nº 3, pp. 189-206, 2013.

[21] "How to use Random Sample Consensus model", *pcl*, [En línea]. Disponible: http://pointclouds.org/documentation/tutorials/random_sample_consensus.php. [Consulta: 5 septiembre 2016].

[22] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments", Tesis doctoral, Computer Science department, Technische Universitaet Muenchen, Alemania, 2009.

[23] R. Merino, "Creación de modelo URDF del robot Manfred", Trabajo fin de grado, departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, Leganés, España, 2014.

[24] K. Harada et al, "Object placement planner for robotic pick and place tasks", *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 980-985, 2012.