UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Software de control de un robot manipulador móvil de intervención

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: David Redó Nieto

Tutor: Vicente Javier Julián Inglada
Román Navarro García

Curso 2017-2018

Resum

Aquest treball està enmarcat dins d'unes pràctiques d'empressa, l'objectiu principal es el desenvolupament d'un sistema que permeta controlar un robot de manera simple i intuïtiva per mitjà de l'ús d'un joystick. El robot pertany al projecte Rising, el qual consistix en el desenvolupament d'un robot manipulador mòbil que permeta la intervenció, evaluació i reconeiximent del entorn per a cossos de seguretat.

Per a aconseguir amb éxit aquest objectiu s'ha empleat ROS, un middleware robótic de codi obert que permet el desenvolupament de sistemes robótics complexos, mitjançant el qual s'ha implementat la descripició del robot i el control manual del mateix. Addicionalment, s'han plantejat algunes possibles solucions davant del problema que emergeix de la integració del braç de Kinova de quatre graus de llibertat amb el paquet Moveit.

Paraules clau: ROS, robot, joystick, braç robòtic

Resumen

Este trabajo está enmarcado dentro de unas prácticas de empresa, cuyo objetivo principal es el desarrollo de un sistema que permita controlar un robot de forma simple e intuitiva mediante el uso de un joystick. El robot pertenece al proyecto Rising, el cual consiste en el desarrollo de un robot manipulador móvil que permita la intervención, evaluación y reconocimiento del entorno para cuerpos de seguridad.

Para alcanzar con éxito dicho objetivo se ha utilizado ROS, un middleware robótico de código abierto que permite el desarrollo de sistemas robóticos complejos, mediante el cual se ha implementado la descripción del robot y el control manual del mismo. Adicionalmente, se han planteado algunas posibles soluciones ante el problema que emerge de la integración del brazo de Kinova de cuatro grados de libertad con el paquete Moveit.

Palabras clave: ROS, robot, joystick, brazo robótico

Abstract

This is a project I created as part of my internship, the main goal is to develop a system that allows the user to control a robot in a simple and intuitive way by using a joystick. Furthermore, the robot belongs to the Rising project, whose purpose is to develop a machine ready to evaluate and recognize the environment for the security forces.

The software used is called ROS, which is an open-source robotic middleware for the development of complex robotic systems, in order to achive the aim of the project successfully. Controlling the robot manually and implementing the robot's description is made by this middleware. Besides, some posible solutions have been suggested to face the Kinova arm integration with Moveit problem.

Key words: ROS, robot, joystick, robotic arm

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

La palabra robótica proviene del checo, en concreto de dos términos: la primera es robota, cuya definición es "trabajo forzado" y la segunda rabota, que se define como "servidumbre" [1]. Se puede apreciar como después de cien años el significado apenas ha cambiado, ya que actualmente los robots se encargan, por lo general, de realizar tareas repetitivas, actividades de gran precisión y trabajos que podrían poner en peligro a un ser humano. La creación y mejora de los robots ha provocado que se hayan involucrado disciplinas como la informática, la mecánica, la electrónica y la ingeniería para materializar ideas que hoy en día se pueden encontrar en múltiples lugares como hospitales, centros de investigación e incluso hogares.

Una de las áreas que cabe destacar dentro de la robótica es la de los manipuladores móviles. Dicho término hace referencia a los sistemas robóticos que se componen de un brazo manipulador robótico montado sobre una plataforma móvil. Éste, es un campo presente en entornos de desarrollo e investigación, ya que puede ser utilizados en diversas áreas como: atención domiciliaria, misiones de intervención en entornos peligrosos, vigilancia, etc. A nivel mundial, las fuerzas de defensa, seguridad y emergencias están adquiriendo este tipo de robots ya que son de gran ayuda en misiones de gran peligro.

1.1 Motivación

Mediante los convenios que mantiene la Universidad Politécnica de Valencia con las empresas, pude realizar este proyecto en la empresa **Robotnik Automation SLL**, una compañía valenciana que ofrece servicios de ingeniería e I+D de alta calidad en el ámbito nacional e internacional en el área de la robótica. Este TFG está enmarcado dentro del proyecto Rising[1], mediante el cual se pretende el desarrollo de una nueva plataforma robótica con manipuladores, sistemas de comunicaciones y capacidad de procesamiento avanzadas. El robot será un nuevo producto que combinará soluciones existentes con nuevas tecnologías. El proyecto Rising consiste en el desarrollo de un robot manipulador móvil que permita la intervención, evaluación y reconocimiento del entorno para cuerpos de seguri-

[1] El proyecto Rising https://www.robotnik.es/rising/

dad. Se pretende que la plataforma trabaje en ambientes hostiles, de difícil acceso y arriesgados. Rising persigue los siguientes objetivos:

- Plataforma robótica compuesta de múltiples sensores así como un brazo robótico con diferentes opciones en su efector final (pinzas o herramientas)
- Consola de control de operaciones avanzada, con capacidades de visualización e interfaz de usuario mejoradas así como controles que permitan una experiencia de control inmersiva
- Algoritmos de localización indoor y modelado en 3 dimensiones (3D) así como la generación de un mapa tridimensional del entorno

1.2 Objetivos

Una vez se conocen los objetivos principales de Rising se procede a explicar el marco en el que se desarrolla este TFG. Éste tiene como objetivo principal el desarrollo de un sistema de control del Rising mediante el uso de un *joystick*. La importancia de este desarrollo reside en la necesidad de controlar el robot de una forma sencilla para poder desempeñar diversas tareas, como por ejemplo la teleoperación de la plataforma a distancia, poder mapear el entorno para su posterior uso en aplicaciones de navegación autónoma, etc.

A través de este dispositivo, el individuo que lo utilice será capaz de mover tanto la base móvil como el brazo robótico. Para poder desarrollar estas funcionalidades, el robot ha atravesado distintos estados de desarrollo hasta el momento. Cabe distinguir dos caminos que sigue todo proceso de creación de un robot: el diseño del hardware y desarrollo del software. En lo concerniente al hardware, el robot se encuentra con lo necesario para comenzar con el desarrollo del software.

Se parte de componentes software que son comunes para todos los robots, como por ejemplo: las librerías que se comunican con los motores a través del bus CAN (*Controller Area Network*) y los paquetes oficiales del repositorio de ROS.

Ante la necesidad de crear un sistema de control para el robot se utilizará las herramientas que proporciona ROS. De esta manera, para alcanzar dicha meta los objetivos más específicos a cumplir para este proyecto son:

- Desarrollo del modelo URDF(*Unified Robot Description Format*) del robot, mediante el cual se integrará tanto el modelo visual del robot como la cinemática y dinámica del mismo.
- Integración del brazo robot Kinova mediante Moveit para poder planificar trayectorias válidas.
- Controlar de forma sencilla e intuitiva tanto la base móvil como del brazo mediante un *joystick*.
- Pruebas y validaciones en diversas sitauciones para verificar el funcionamiento del robot.

1.3 Estructura

La memoria de este trabajo está compuesta por nueve capítulos. En el **primer capítulo**, donde se encuentra el lector, se describe la motivación del trabajo así como los objetivos del mismo.

El **segundo capítulo** se compone de un análisis de los distintos robots que satisfacen objetivos comunes con el Rising, además de una descripción de algunos de los middleware para robots de código abierto que existen.

En el **tercer capítulo**, se ha realizado un análisis del problema, mediante el cual se ha podido describir cómo se ha diseñado la aplicación.

Por otro lado, el **cuarto capítulo** está compuesto por la descripción de la solución, la cual abarca desde la arquitectura hardware y software del robot hasta la tecnología empleada para el desarrollo del mismo.

En el **quinto capítulo** se encuentra todo el desarrollo que se ha realizado para alcanzar con éxito los objetivos propuestos.

El **sexto capítulo** detalla tanto el proceso de implantación de la aplicación en el robot así como su configuración para su uso posterior.

A continuación, el **capítulo séptimo** describe las pruebas que se han diseñado y realizado para poner a prueba la eficacia del sistema desarrollado.

El **séptimo capítulo** se compone de las conclusiones obtenidas tras la realización de este proyecto.

Por último, en el **noveno capítulo** se ofrece un resumen de las líneas de desarrollo que se abren tras la realización de este trabajo.

CAPÍTULO 2
Estado del arte

En este capítulo se va a analizar el estado del arte desde dos perspectivas: la de los robots que existen actualmente en el mismo nicho de mercado y la de los *middleware* de código abierto que facilitan el desarrollo de los sistemas robóticos. Además, se realizará una crítica al estado del arte junto con una propuesta para el problema a resolver.

2.1 Robots

El mercado de la robótica se encuentra actualmente en auge, con una estimación de crecimiento del 8 % hasta 2020. Una de las razones de este aumento en la demanda de robots terrestres se debe a la utilización de los mismos por parte de las fuerzas militares y los cuerpos de seguridad en su lucha contra el terrorismo. Además, se busca que estos robots se compongan de sistemas altamente innovadores con unos presupuestos ajustados.

Aunque el objetivo general de todos ellos es muy similar, existen diferencias notables tanto en las características como en el coste de cada uno. A continuación, se presentará distintos robots que se encuentran actualmente en el mercado.

2.1.1. Packbot

El PackBot[1] es un robot móvil táctico desarrollado y fabricado por iRobot. Ha sido creado para que sea usado por militares en el campo de batalla para la realización de misiones peligrosas.

Destaca por su diseño modular, como se puede observar en la Figura 2.1, lo que permite que se pueda configurar para cada misión. Es un robot ligero que puede ser desplegado en dos minutos independientemente de las condiciones ambientales.

[1]PackBot de iRobot https://www.darley.com/documents/guides/robotics/spec_sheets/PackBot_Specs.pdf

Figura 2.1: Robot Packbot de iRobot

El PackBot puede realizar distintas tareas, entre las que destacan:

- Vigilancia y reconocimiento
- Detección de químicos, agentes biológicos y nucleares
- Desactivación de explosivos o EOD (*Explosive Ordnance Disposal*)

El PackBot se controla de manera remota mediante dos joysticks conectados a un ordenador. Éste, además, es usado para almacenar imágenes y vídeos en tiempo real que se utilizan en el análisis de cada misión. También puede guardar la posición en la que ha estado el brazo durante la misión así como la localización del robot y la batería. El coste de adquisición del PackBot es de 120.000$.

2.1.2. Dragon Runner 20

El Dragon Runner 20[2] es un sistema sin tripulación que provee conciencia situacional. La conciencia situacional consiste en la percepción de uno mismo y del entorno, con la capacidad de pronosticar qué puede ocurrir, es decir, tener una percepción exacta de la situación. Originalmente fue diseñado para la Marina de los Estados Unidos y puede realizar tareas de seguridad, de reconocimiento y ofrece ayuda táctica en las operaciones de desactivación de explosivos. Sus componentes principales son:

- Brazo manipulador
- Cámara PTZ (pan-tilt-zoom)
- Detectores de movimiento
- Micrófono

[2]Dragon Runner 20 https://www.qinetiq-na.com/products/unmanned-systems/dragon-runner/

Figura 2.2: Robot Dragon Runner 20 durante una misión

El Dragon Runner está diseñado para poder operar en distintos modos:

- Modo conducción: el robot es controlado remotamente mientras transmite imágenes al operador
- Modo centinela: el robot permanece quieto y, usando los distintos sensores que posee, monitoriza la situación. Alertará al operario si detecta algo
- Modo visión: permanece quieto mientra proporciona imágenes al usuario que lo controle

El conjunto de componentes que forman este robot, además de sus distintas funcionalidades hacen que su coste sea de 40000$.

2.1.3. Cobra MK2

Cobra MK2[3] está equipado con cámaras diurnas y nocturnas para las operaciones de inteligencia y reconocimiento. Su principal característica es su arquitectura modular, lo que le permite tener distintas configuraciones dependiendo de la misión y del entorno.

Las principales misiones del Cobra MK2 son:

- Inspección tanto de interiores como de exteriores
- Detección y neutralización de explosivos
- Misiones de reconocimiento del terreno

Sus componentes son láseres, cámaras diurnas y de visión nocturna, GPS y un micrófono. Al ser un robot modular, estos sensores pueden variar dependiendo de la tarea a realizar. Por ello, tiene un coste de 28000$.

[3]Cobra MK2 https://www.ecagroup.com/en/solutions/cobra-mk2-i-ugv-unmanned-ground-vehicle

Figura 2.3: Robot Cobra MK2

2.2 Middleware para robots

Como se muestra en la figura 2.4 el *middleware* robótico es una capa de abstracción que reside entre el sistema operativo y las aplicaciones software. Éste mejora la calidad del software, simplifica el proceso de desarrollo y reduce sus costes. Por ello, un desarrollador se podrá enfocar en construir la lógica de la aplicación. Como se menciona en [2], las ventajas de estos middleware para robots son: la modularidad de los componentes software, la abstracción hardware para ocultar los detalles de más bajo nivel con el objetivo de proporcionar a los desarrolladores una API hardware estandarizada, y la portabilidad, es decir, que sea capaz de funcionar en cualquier plataforma sin cambiar la configuración del sistema. Además, cualquier *middleware* debe ser fácil de usar y de mantener, robusto, eficiente y seguro.

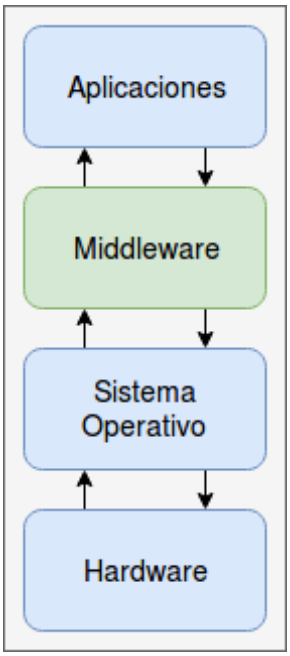

Figura 2.4: Middleware

Existen proyectos *middleware* que comparten conceptos básicos y objetivos, sin embargo, muchos de ellos son solo propuestas o ya no están en desarrollo. Entre todos ellos cabe destacar OROCOS, Player/Stage y ROS. A continuación, se detallarán las principales características de los middleware más destacados actualmente.

2.2.1. OROCOS

Este proyecto[4] de software libre surgió en Diciembre del año 2000 debido a numerosas experiencias no satisfactorias con el software comercial de aquella época. Además, no existía software robótico, de propósito general, de código abierto, por lo que hizo de OROCOS una solución innovadora para mucha gente. El sistema OROCOS está compuesto por las siguientes [3] cuatro librerías escritas en C++:

- Librería de **tiempo real**: provee tanto la infraestructura como las funciones necesarias para construir aplicaciones robóticas en tiempo real. Esto permite que los diseñadores creen aplicaciones altamente configurables, como modificar los parámetros de un algoritmo en tiempo de ejecución.
- **Componentes de OROCOS**: ofrece algunos componentes ya preparados para ser usados. Algunos de estos son usados como interfaz de algún dispositivo hardware, otros como planificador de tareas, etc.
- **Cinemática y Dinámica**: es una librería que permite calcular una cadena cinemática en tiempo real como pueden ser robots, modelos biomecánicos humanos, figuras animadas, etc.
- Librería de **filtros bayesianos**: como su nombre indica, se compone de funciones que permiten el cálculo de estimaciones algorítmicas basadas en la regla de Bayes, como por ejemplo los Filtros de Kalman o los filtros de partículas.

2.2.2. Player/Stage

El proyecto Player[5] es gratis y de código abierto el cual permite la investigación con los robots y los sistemas sensorizados. Está lanzado bajo una licencia GNU (*General Public License*) y ha sido desarrollado por un equipo internacional de investigadores en robótica y usado en los laboratorios de los centros de investigación. El software se puede dividir en:

- **Player**: ofrece una interfaz de red a una variedad de sensores y componentes hardware de robots. El modelo de cliente/servidor que implementa permite que los programas puedan ser escritos en cualquier lenguaje de programación y ejecutarlo en cualquier máquina que mantenga una conexión con el robot. Player soporta múltiples conexiones de clientes de forma concurrente creando nuevas posibilidades de colaborar de una manera distribuida.

[4] Página oficial de OROCOS http://www.orocos.org/
[5] Página oficial de *The Player Project* http://playerstage.sourceforge.net/

- **Stage**: simula una población de robots móviles en un terreno bidimensional. Distintos sensores están disponibles, incluyendo sonares, láseres, cámaras PTZ, etc. Presenta una interfaz estándar de Player, por lo que los cambios que se han de realizar para trasladar la simulación al hardware real son mínimos. De hecho, muchos controladores que han sido diseñados en Stage han sido empleados en robots reales.

2.2.3. ROS

En este trabajo se ha empleado ROS[6] como *middleware* robótico cuyas características principales se explican en la sección 4.3.1. No obstante, se va a realizar una breve explicación de los elementos principales.

ROS se define como un meta sistema operativo de código abierto que trata de ser algo más que un *middleware*. Aunque no es un sistema operativo en sí, proporciona servicios esperados en uno, incluyendo la capa de abstracción hardware, control de los dispositivos a bajo nivel, funcionalidades comunes y gestión de paquetes. Está compuesto de tres capas: la primera el **sistema de archivos** donde se almacenan los recursos de ROS. La segunda se trata del **grafo computacional**, una red peer-to-peer de procesos ROS en los que se computan las tareas. Por último, la **comunidad** donde se puede encontrar recursos desarrollados por otras personas. El núcleo de todo el sistema es el grafo compuesto por una red distribuida de procesos, mejor conocidos como nodos. Los elementos básicos de esta red son:

- **Nodos**: son procesos que realizan tareas computacionales
- **Servicio máster**: ofrece el registro de nombres que está disponible para el resto del grafo
- **Servidor de parámetros**: almacena estos datos en una localización central
- **Mensajes**: son usados por los nodos para comunicarse entre ellos
- **Topics**: se usan para identificar el contenido de un mensaje. Los mensajes se transportan mediante un sistema publicador/suscriptor. Un nodo envía el mensaje, publicándose, en un topic mientras que otro puede suscribirse a dicho topic
- **Servicios**: cuando el modelo publicador/suscriptor no es apropiado se utilizan los servicios, los cuales implementan una interacción petición/respuesta

2.3 Crítica al estado del arte

Debido a que el desarrollo de este trabajo está enmarcado dentro de una empresa, no es fácil encontrar proyectos realizados por miembros de la Universidad

[6] Página de ROS http://www.ros.org/

Politécnica de Valencia cuyos objetivos sean similares. No obstante, existen algunos como *Simulación en entornos con robots manipuladores móviles - Carlos Martínez Navarro*[7] y *Programació i simulació del robot mòbil Guardian mitjançant ROS i Gazebo - Marc Benetó Juan*[8] que comparten características.

Figura 2.5: Logitech Freedom (izda.) y Wireless Controller PS4 (dcha.)

Aunque se trate de un entorno simulado, en el primer trabajo se hace uso de manipuladores. En él, utiliza V-REP para la simulación junto con un módulo propio de ese entorno llamado *Motion Planning* para la planificación del movimiento. Sin embargo, en este proyecto se ha empleado Moveit, ya que es el software que lidera la manipulación móvil y además está disponible para trabajar con ROS.

Por otro lado, en el segundo trabajo se realiza una simulación de un robot real, en la cual la teleoperación de la base móvil se puede realizar a través de dos dispositivos: el teclado y un *joystick*. El manejo de un robot mediante teclado puede llegar a ser poco intuitivo y complejo para entornos donde se requiera especial atención. Con respecto al joystick, la diferencia reside en el modelo que se ha empleado, en dicho proyecto se utilizó un Logitech Freedom mientras que en este se emplea un Wireless Controller PS4. Como se aprecia en la figura 2.5 existen diferencias significativas entre ambos.

2.4 Propuesta

El propósito de este trabajo, como se ha mencionado en el capítulo anterior, es proporcionar una aplicación que permita el manejo de un robot y sus componentes a través de un joystick. Como se puede observar, los distintos robots presentados en el apartado 2.1 también pueden ser manejados remotamente por un usuario. Por ello, aunque existan aplicaciones para la teleoperación de un robot, la tecnología empleada en el desarrollo de este proyecto, junto con la arquitectura de software que emplea la empresa donde estoy actualmente realizando este trabajo, no solo ofrece un producto más robusto, fácil de usar y seguro, sino que permite centrarse en el desarrollo de aplicaciones de más alto nivel.

[7] Simulación en entornos con robots manipuladores móviles https://riunet.upv.es/handle/10251/55467

[8] Programació i simulació del robot mòbil Guardian mitjançant ROS i Gazebo https://riunet.upv.es/handle/10251/11716

CAPÍTULO 3
Análisis del problema

En este capítulo se va a analizar el problema a resolver y qué solución es la más adecuada para ello. El robot sobre el que se va a trabajar es un prototipo que sigue actualemente en desarrollo y el problema a resolver, como se ha mencionado, consiste en el control de dicho robot mediante un *joystick*. Para ello, se han establecido una serie de requisitos que ha de cumplir:

- **Fácil de usar**: el manejo del robot mediante el *joystick* ha de ser intuitivo ya que será usado en situaciones que requieren precisión, por tanto éste será un aspecto que estará presente en todo el trabajo.
- **Seguro**: se trata de un robot colaborativo, es decir, que realiza tareas para las personas con las que ha de compartir, de forma segura, su espacio.
- **Robusto**: un sistema de este tipo tiene que ser consistente y estable para que todos sus elementos tanto hardware como software funcionen correctamente en todo momento.

3.1 Identificación y análisis de soluciones posibles

Tras haber analizado e identificado el problema a resolver, es posible presentar distintas posibles soluciones que permitan lograr con éxito el control del robot.

Una posible solución sería la utilización de algún *framework* de código abierto, como los mencionados anteriormente, ya que permite el uso de un sistema que facilita las tareas a los desarrolladores debido a que algunos de ellos ofrecen abstracciones del hardware así como también herramientas para que la puesta en marcha de un robot sea fácil y rápida, pudiendo centrarse, el programador, en tareas de más alto nivel.

Otra posible solución se centraría en tener un software privativo de la empresa, mediante el cual se pueda resolver los problemas propuestos. Esta solución no solo implicaría crear un sistema desde cero sino que sea compatible entre distintos robots de la empresa, ya que de no ser así, habría que adaptarlo según las necesidades de cada uno.

3.2 Solución propuesta

La solución propuesta consiste en la utilización de una herramienta de código abierto que facilite la puesta en marcha de un robot. Debido a que este trabajo está enmarcado dentro de unas prácticas en empresa, el *framework* utilizado ha sido ROS, ya que es el empleado en la misma para el desarrollo del resto de sistemas robóticos. No obstante, ROS presenta unas características que lo convierten en una opción realmente viable frente al resto de opciones:

- **Interplataforma**: permite el trabajo desde distintas máquinas y subsistemas, los cuales pueden estar escritos con distintos lenguajes de programación.
- **Modularidad**: debido al diseño distribuido que tiene ROS, si un componente falla, éste no provoca la caída de todo el sistema. Aunque sería más robusto si no ocurre, en la improbable situación en la que un sensor fallase, el sistema permitiría que el robot continuase.
- **Comunidad**: equipos de investigación, compañías y otras organizaciones han elegido ROS como su método de trabajo. Esto implica que sus aportaciones sean accesibles por cualquier persona, pudiendo aportar nuevas herramientas como mejorar las existentes. Estas aportaciones no solo son en forma de software sino que existe una plataforma donde hay resueltas numerosas dudas sobre cualquier tema relacionado con ROS.

De esta forma, todas estas caracterísitcas hacen que ROS sea una elección adecuada para el desarrollo de sistemas robóticos. Además, las aportaciones de la comunidad a los repositorios oficiales pertimen que todos los usuarios se beneficien de dichas mejoras, pudiendo integrarlas en sus aplicaciones.

CAPÍTULO 4
Diseño de la solución

En este capítulo se va a especificar la arquitectura del sistema, donde se comentarán los distintos elementos que componen al robot. Además, se detallará la estructura de paquetes software que se ha empleado para este proyecto. Por último, se realizará una descripción del *framework* utilizado para el desarrollo de este trabajo.

4.1 Arquitectura del sistema

Una vez se han identificado los requisitos del sistema que se va a desarrollar, se procede al diseño de la solución del problema. Sin embargo, antes de indicar los grandes bloques en los que se dividirá la solución, se procede a explicar qué componentes se encuentran en el robot y cómo interactúan entre sí.

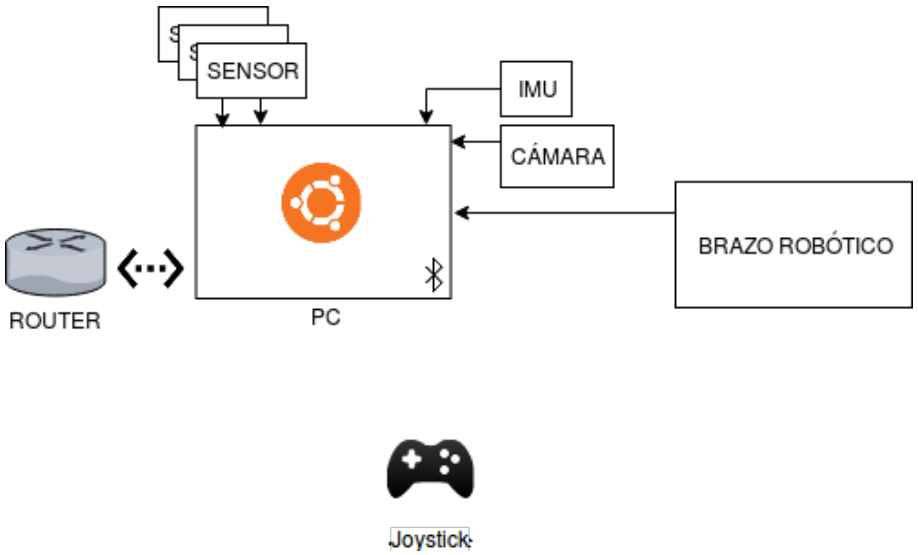

Figura 4.1: Componentes del Rising

Como se puede apreciar en la figura 4.1, el Rising está compuesto por los siguientes componentes:

- **PC**: el PC es el elemento principal del robot, ya que todos los dispositivos están conectados a él. Además, es el que se encarga de ejecutar el software necesario para que todo funcione correctamente.

- **Router**: el router está conectado al PC por cable, y se encarga de proporcionar una red local para el robot. Un usuario con autorización puede estar dentro de la misma red que el robot y, de esta manera, tener acceso a los distintos servicios que proporcione el robot.

- **Cámara PTZ**: ésta se conecta al PC a través de *ethernet*. La cámara PTZ puede rotar alrededor de dos ejes, el vertical y el horizontal. Además, puede usar el zoom para acercarse o alejarse a su objetivo.

- **Cámara rgbd**: se conecta al PC mediante un cable USB. Esta cámara está situada en la parte frontal del robot y proporciona tanto imagen en rgb como imagen en profundidad.

- **IMU**: al igual que la cámara rgbd, la IMU se conecta por USB. Este dispositivo se encuentra en el interior del robot y proporciona información acerca de la velocidad, orientación y fuerzas gravitacionales. Para ello, utiliza una serie de acelerómetros y giróscopos.

- ***Joystick***: este dispositivo se conecta mediante *bluetooth* al PC. Así, el operario podrá manejar tanto la base móvil como el brazo robótico. El *joystick* enviará direcciones y posiciones al PC, el cuál se encargará de procesar y enviar comandos a los dispositivos correspondientes.

- **Brazo robótico**: tanto la primera versión del brazo como el de Kinova se conectan por serie al ordenador.

Una vez comentados qué componentes existen y cómo se relacionan con el PC, se procede a comentar los bloques en los que se dividirá la solución. Cada uno de ellos se corresponde con un objetivo específico.

El **primer** bloque se corresponde con el primer objetivo detallado en la sección 1.2, que consiste en desarrollar el modelo URDF (*Unified Robot Description Format*) del robot. Este formato es una especificación XML donde se describe al robot. Como más adelante se explica, es un elemento esencial tanto para la simulación de un robot como para el uso de uno real.

En el **segundo** consta de la integración del brazo robótico mediante el paquete ROS "MoveIt". En él, se detallará algunas de las posibles soluciones que se han planteado para resolver el problema de la cinemática inversa con brazos de cuatro DOF (que existe en la versión actual del paquete utilizado).

El **tercer** bloque abarca la implementación del control a través de un joystick. Al finalizar, este dispositivo deberá ser capaz de comandar al robot, así como mover el brazo a la posición deseada.

El **cuarto** se centrará en las distintas pruebas realizadas para validar el correcto funcionamiento del robot, así como la planificación de trayectorias con el brazo robótico.

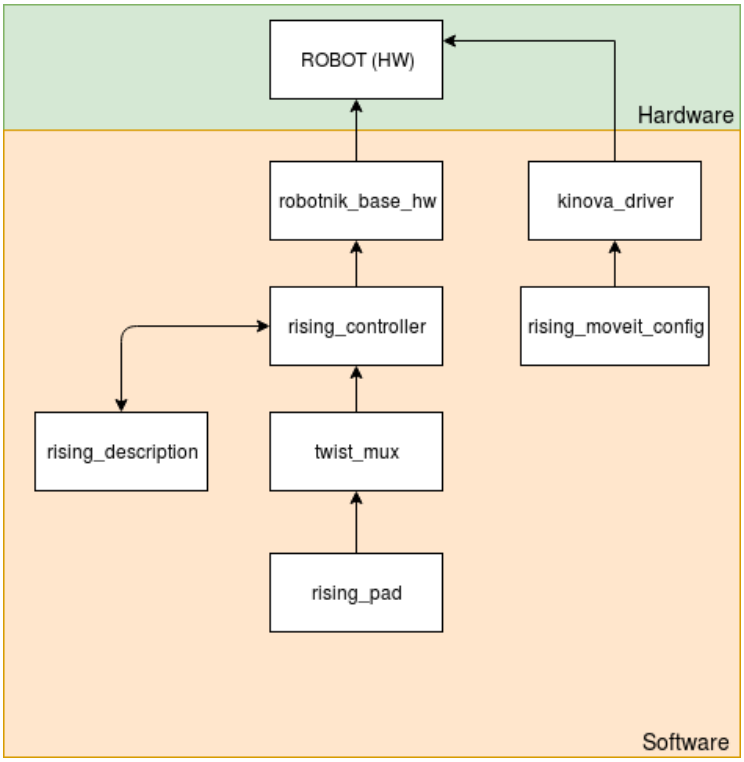

Figura 4.2: Estructura de paquetes software

4.2 Diseño detallado

El software está desarrollado con ROS, un sistema *middleware* para robots, como se comenta en la sección 2.1. En este apartado, se detalla la estructura de paquetes software que componen al robot y cómo interactúan entre ellos.

La figura 4.2 representa los paquetes software de ROS y cómo están organizados. En primer lugar, se destaca la librería **robotnik_base_hw**, la cual está desarrollada por Robotnik y se encarga de la comunicación con los *drivers* de los motores. Dicho componente puede ser configurado según las características de cada robot. Por ello, como el Rising está compuesto de tres motores en velocidad, se deberá especificar esta información en la configuración. De esta manera, la librería puede ser usada por distintos robots dentro de la compañía de una manera estándar y cómoda.

Todos los paquetes que comienzan por 'rising' han sido creados específicamente para el robot. El resto son paquetes de terceros que están a disposición de todos los usuarios de ROS.

En segundo lugar, el paquete **rising_controller** se compone de funciones de control que permiten obtener estimaciones de odometría a partir de los sensores del robot. Además, se encarga de procesar qué velocidad debe llevar cada motor y enviarla a la capa inferior (robotnik_base_hw) la cual, como se ha explicado anteriormente, la trasladará a los motores.

Como se aprecia en la Figura 4.2, **twist_mux**[1] está relacionado con el controlador. Este paquete ofrece un multiplexor para mensajes de tipo Twist (comandos

[1]Paquete twist_mux http://wiki.ros.org/twist_mux

de velocidad en ROS), de esta manera, se puede tener varios nodos publicando una velocidad deseada, sin embargo, el multiplexor se encargará de enviarle al controlador solo una, aquella del topic que se haya seleccionado como más prioritario. Por ejemplo, se dispone de dos nodo que publican qué velocidad debe llevar el robot, el primero, y más prioritario, un joystick. El segundo, es un programa que calcula la trayectoria que el robot deberá seguir de forma autónoma. En esta situación, mientras el joystick no proporcione ningún comando de velocidad, el twist_mux enviará la velocidad que el programa de navegación autónoma le proporcione. En el momento que alguna persona utilice el joystick, este responderá a dichos comandos y se asumirá un control manual.

El desarrollo del paquete **rising_description** se corresponde con el primer bloque mencionado en el apartado anterior. En él, se incorporarán todos los elementos necesarios para alcanzar el objetivo con éxito como por ejemplo, la especificación que describe al robot, así como los archivos modelados que permiten su representación gráfica.

Por otro lado, en el segundo bloque del trabajo se tienen dos paquetes relacionados con el manipulador de Kinova. El primero se trata de **kinova_driver**[2], el cual contiene los *drivers* para ROS que ha desarrollado la misma compañía que fabrica dicho brazo. El segundo, llamado **rising_moveit_config** y que se ha desarrollado mediante el asistente de Moveit, se compone de los ficheros necesarios para que el manipulador siga una trayectoria como se explica detalladamente en el capítulo siguiente.

Por último, el tercer bloque se corresponde con el desarrollo del paquete **rising_pad**. Se compone del software necesario para realizar la teleoperación tanto de la base móvil como del manipulador.

4.3 Tecnología usada

4.3.1. ROS (Robot Operating System)

La tarea de diseñar y desarrollar software para plataformas robóticas puede ser un proceso largo y complejo. Requiere de habilidades de distintas disciplinas, desde los *drivers* de nivel más bajo hasta las abstracciones software de las capas más altas. Ante tal necesidad surgieron sistemas middleware para robots tales como Player, OROCOS, YARP, ROS, etc [4]. Actualmente, ROS es uno de los más utilizados, además cuenta con una amplia comunidad que contribuye en el desarrollo software del mismo. Una de las principales razones por las que ROS es ampliamente utilizado actualmente se debe a que provee una capa de abstracción del hardware, la cual traduce cualquier dato físico, proveniente de sensores o actuadores, en estructuras de datos estándares para la comunidad de ROS. Esto permite que los desarrolladores se centren en aplicaciones de más alto nivel sin preocuparse de los problemas hardware.

ROS, es un *framework* para desarrollar software para robots. Está compuesto por una colección de herramientas, librerías y convenciones cuyo objetivo es sim-

[2]Repositorio oficial de Kinova-ROS https://github.com/Kinovarobotics/kinova-ros

plificar la tarea de crear un sistema robótico robusto y complejo [5]. La necesidad de un *framework* colaborativo abierto estaba presente en numerosas instituciones, así como en la mente de muchos investigadores y desarrolladores. Fue en Willow Garage donde un grupo de visionarios de la robótica decidieron proveer de importantes recursos para extender estos conceptos y crear implementaciones adecuadamente probadas. Fue entonces, cuando innumerables investigadores dedicaron su tiempo y su conocimiento para crear las ideas del núcleo de ROS, además de los principales paquetes software [6]. El software ha sido desarrollado bajo una licencia BSD[3], una característica que incita a las personas que no desean o no pueden adquirir una licencia, a utilizar este sistema, creando así una comunidad global compuesta de miles de personas.

4.3.2. Arquitectura

El grafo de computación de ROS es una red *peer-to-peer* (P2P) de procesos procesando los datos al mismo tiempo. Los conceptos básicos que componen esta arquitectura son: nodos, mensajes, servicios, topics y el ROS máster.

Los **nodos** son las unidades de computación en ROS. Cada nodo realiza un proceso computacional distinto (control de las ruedas, localización, lectura de los sensores, etc.), por lo que un sistema robótico requerirá el compromiso de distintos nodos, confiriendo a ROS ser un sistema modular.

Los **mensajes** son estructuras de datos simples. Éstos están formados por tipos de datos primitivos (*integer, boolean, floating point, etc*) así como *arrays* de estos mismos tipos. Los nodos utilizan estos mensajes para comunicarse entre sí.

Los mensajes son enviados por los nodos mediante un sistema publicador/suscriptor. Un nodo envía un mensaje publicandolo en un **topic**. De la misma manera, si un nodo está interesado en cierta información enviada por un nodo en particular, se deberá suscribir a dicho topic. Pueden existir tanto múltiples publicadores para un mismo topic como múltiples suscriptores para dicho topic. Por otro lados, los nodos pueden tanto publicar como suscribirse a todos los topics necesarios.

Aunque el modelo publicador/suscriptor es muy flexible, no es apropiado para las interacciones petición/respuesta requeridas en un sistema distribuido. Esto se soluciona mediante los **servicios**, compuestos por dos mensajes, uno para la petición y otro para la respuesta. Los nodos ofrecen los servicios bajo un nombre y el nodo cliente utiliza este nombre para enviarle el mensaje de petición y quedarse a la espera de la respuesta.

El **ROS Máster** proporciona un servicio de nombres y de registros para los nodos dentro del sistema de ROS. Mantiene un seguimiento tanto de los publicadores como de los suscriptores, así como también de los servicios. El papel que ejerce el Máster es el de permitir que los nodos individuales del sistema puedan localizarse entre ellos. Una vez los nodos se han localizado entre sí, la comunicación que se establece entre ellos es *peer-to-peer*.

A continuación, se mostrará un ejemplo sobre cómo opera el Máster [7]. Para ello, supondremos que tenemos dos nodos, uno llamado "camera" y otro llamado

[3]Licencia BSD https://opensource.org/licenses/BSD-3-Clause

"image_viewer". La secuencia típica empezaría por el nodo "camera" notificando al Máster que desea publicar imágenes en el topic "images".

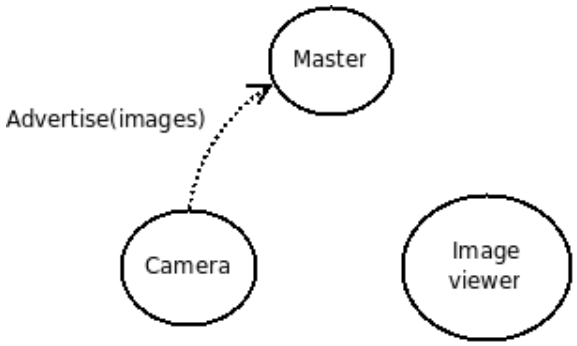

Figura 4.3: Nodo "camera" notifica a Máster que va a publicar en "images"

Por el momento el nodo "camera" está publicando a través del topic "images", pero nadie está suscrito a dicho topic, por lo que no se está enviando datos. En este instante, "image_viewer" quiere suscribirse a "images" para ver si existe alguna imagen en el topic.

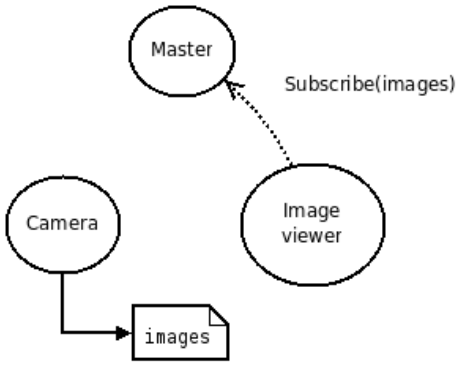

Figura 4.4: Nodo "image_viewer" notifica que quiere suscribirse a "images"

Por último, ahora que "images" tiene un publicador y un suscriptor, el Máster notifica a los nodos correspondientes de la existencia de cada uno para que pueda comenzar la transferencia de imágenes entre ellos.

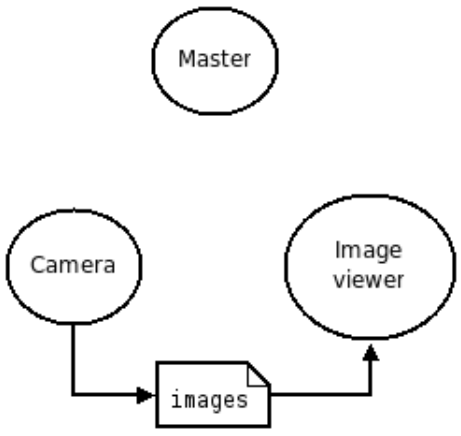

Figura 4.5: El topic "images" ya tiene publicador y suscriptor

4.3.3. Sistema de ficheros

ROS no es exactamente un sistema operativo, pero tiene un sistema de archivos que está organizado de la siguiente manera:

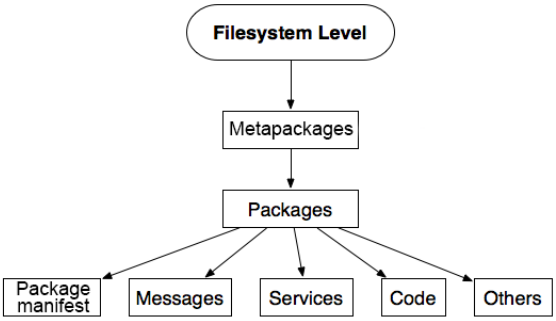

Figura 4.6: Sistema de archivos de ROS

- **Paquetes**: son la unidad principal de ROS para organizar el software. Éste contiene procesos ROS (nodos), librerías que dependen de ROS, archivos de configuración y cualquier elemento necesario para que dicho paquete funcione correctamente. Los paquetes son la unidad más atómica que se puede construir. El objetivo de estos paquetes es proveer funcionalidades útiles de una manera estructurada e intuitiva, facilitando la reutilización del software.

- **Meta-paquetes**: son paquetes cuya tarea es agrupar a un conjunto de paquetes que tienen un propósito común.

- **Manifiesto**: el manifiesto (package.xml) tiene como fin proporcionar metadatos del paquete. Estos datos son: nombre del paquete, versión, descripción del mismo, información sobre la licencia, dependencias, etc.

- **Descripción del mensaje**: ROS utiliza un lenguaje de mensajes simplificado que permite describir el valor de los datos que serán usados por los nodos. Dicha descripción es muy útil, ya que permite generar automáticamente código fuente para el tipo del mensaje, pudiendo ser utilizado por diversos lenguajes de programación. La descripción de los mensajes está almacenada en ficheros .msg pertenecientes al subdirectorio msg/ del paquete de ROS. Por ejemplo, el fichero Point.msg se corresponde con:

  int32 x

  int32 y

- **Descripción del servicio**: De la misma forma que ROS emplea un lenguaje simplificado para los mensajes, también utiliza éste para describir los tipos de los servicios. Estos archivos .srv que se encuentran en el subdirectorio srv/ de los paquetes de ROS se componen de dos partes: la primera parte, la petición, y la segunda se corresponde con la respuesta. Como la petición y la respuesta se diseñan en el mismo archivo, se utilizarán tres guiones para separarlas.

string req

- - -

string res

Este es un ejemplo donde tanto la petición como la respuesta son de tipo string. No obstante, se debe estar familiarizado con la descripción de los mensajes ya que estos pueden ser utilizados como tipo de alguna de las dos partes de los servicios. Por ejemplo, podemos usar el mensaje que hemos creado previamente (Point.msg) como petición del servicio:

nombre_del_paquete/Point punto

- - -

bool res

Esta descripción del servicio toma como petición un punto y devuelve como respuesta un *bool*. Por ejemplo, este tipo podría ser usado en un servicio que calcule posiciones válidas del robot. Recibe como entrada un punto destino en el mapa y tras comprobar si es posible enviar el robot a esa posición, el servicio devolverá *True* o *False* en función de si puede alcanzarla o no.

4.3.4. Comandos de ROS

A continuación, se va a presentar alguna de las herramientas (usada a través de la línea de comandos) que proporciona ROS para facilitar el desarrollo del software:

- **roscore**: pone en funcionamiento el ROS Máster. Lanza internamente el servidor de parámetros además del nodo de monitorización del sistema.
- **rosrun**: lanza un nodo en concreto indicando tanto el nombre del paquete que lo contiene como el nodo a ejecutar.
- **roslaunch**: ejecuta archivos bajo la extensión launch. Dichos archivos tienen un formato XML que permite lanzar nodos, configurar los parámetros que se utilizan dentro de cada nodo, inclusión de otros ficheros, etc.
- **rosnode**: permite conocer la información relacionada con cada nodo para su depuración, desde los topics que publica, a los que está suscrito, conexiones, etc.
- **rostopic**: mediante este comando se puede imprimir los mensajes que se están publicando en el topic seleccionado. Además, proporciona información sobre el tipo de mensaje y la frecuencia de actualización.
- **rosservice**: permite extraer información sobre los servicios que existen además de realizar peticiones a cualquiera de ellos.
- **rosparam**: a través de este comando es posible extraer el valor del parámetro alojado en el servidor de parámetros, además de poder cambiar dicho valor.

4.3.5. Herramientas gráficas

En esta sección se va a presentar herramientas con interfaz gráfica que ayudan en el proceso de desarrollo de las aplicaciones:

- **Rviz**: este paquete contiene un visualizador 3D cuya importancia reside en la capacidad que tiene para mostrar los topics de manera gráfica, como por ejemplo los láseres, las nubes de puntos, el modelo del robot, el mapa que se emplea para la navegación, etc. La figura 4.7 muestra un ejemplo de su potencia.

- **Gazebo**: es el simulador más utilizado dentro de la comunidad de ROS. Permite probar de una manera fácil y cómoda la aplicación que se esté implementando. El motor de físicas y gráficos de alta calidad que lo compone ofrece una experiencia bastante cercana al escenario real. Es de gran utilidad ya que no es necesario tener el robot físico para comenzar el desarrollo.

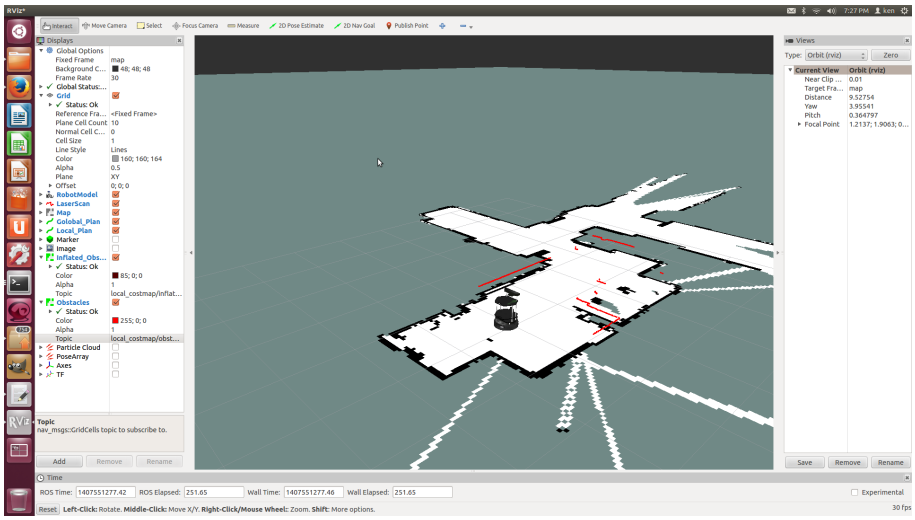

Figura 4.7: Rviz visualizando un mapa, el láser y el modelo del robot

CAPÍTULO 5
Desarrollo de la solución propuesta

En este apartado se va a describir el desarrollo seguido para alcanzar los objetivos inicialmente planteados. Como se ha mencionado en el capítulo anterior, el trabajo se puede dividir en tres bloques, el primero es el desarrollo del modelo URDF del robot, el segundo se corresponde con la integración del brazo robótico y el último con el manejo del robot mediante un *joystick*.

5.1 El modelo URDF

Para la modelización del robot se ha empleado el paquete urdf[1] que podemos encontrar en ROS. Este paquete contiene un *parser* escrito en C++ para archivos URDF, el cual es un documento en formato XML que representa el modelo del robot. En este modelo se incorpora la cinemática y la dinámica del robot, así como la representación visual del robot junto con el modelo de colisiones del mismo. Este paquete contiene especificaciones XML para los elementos que puede incorporar un robot.

El elemento ***joint*** es el encargado de describir la cinemática y dinámica de la articulación. Además, especifica los límites de seguridad en los que esa articulación puede trabajar sin poner en riesgo la integridad del robot.

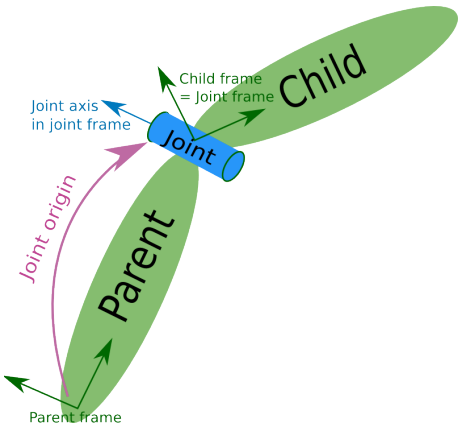

Figura 5.1: Representación de un *joint*

[1]Paquete urdf http://wiki.ros.org/urdf

Los *joints* pueden ser de los siguientes tipos:

- ***revolute***: una articulación que rota sobre un eje y que tiene un rango máximo de actuación definido por los límites tanto superior como inferior.
- ***continuous***: al igual que el tipo anterior, rota sobre un eje, sin embargo, no tiene límites de rotación.
- ***prismatic***: una articulación que se desplaza por el eje y que posee un límite máximo de desplazamiento.
- ***fixed***: no es exactamente una articulación ya que no se puede mover, ya que tiene bloqueados todos los grados de libertad.
- ***floating***: permite la movilidad en los seis grados de libertad.
- ***planar***: permite a la articulación moverse en un plano perpendicular al eje.

El elemento ***link*** describe a un cuerpo rígido con una inercia y propiedades visuales. Dichas propiedades especifican la forma del objeto, el cual puede ser una figura geométrica (prisma, cilindro o esfera) o un archivo mesh. El formato recomendado para la representación gráfica de colores y texturas es *Collada* con extensión .dae, aunque los archivos .stl también son soportados. Además, se puede definir la colisión de este *link*. No es obligatorio que la representación gráfica y la colisión coincidan, de hecho, para reducir tiempo de cómputo se utilizan modelos de colisión más simples. Estas características son opcionales, no obstante, si no dispone de un robot físico, éste puede ser simulado, en cuyo caso sí se requiere que estas propiedades estén disponibles.

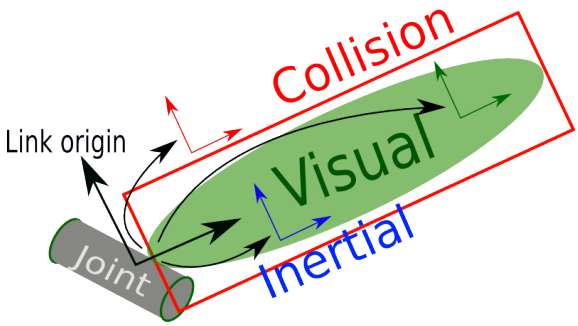

Figura 5.2: Representación de un *link*

La descripción básica de un robot consiste en una serie de ***links*** y ***joints*** conectados entre sí. La figura 5.3 representa dicha descripción completa de un robot cualquiera. Además de estos dos elementos necesarios, existen otros que son empleados en la simulación, ya que permiten el desarrollo de las aplicaciones sin necesidad de tener un robot físicamente. Los elementos más importantes son:

- **transmisión**: describe la relación entre un actuador y una articulación, lo que permite modelar conceptos como la reducción del motor.
- **gazebo**: permite describir las propiedades de la simulación. El nombre de este elemento coincide con el programa de simulación que se emplea en la comunidad de ROS.

- **sensor**: permite especificar qué tipo de sensor utiliza el robot, como por ejemplo una cámara o un láser. Esto permite el uso de estos sensores desde un entorno virtual, obteniendo los mismos datos que se podrían recibir con una cámara o láser real.

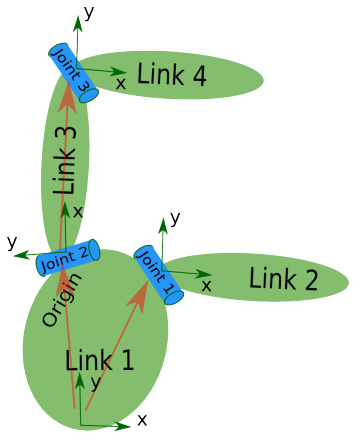

Figura 5.3: Posible descripción de un robot

5.1.1. Descripción del Rising

Para la descripción del Rising se ha empleado Xacro (además del URDF), un macro lenguaje XML que permite escribir archivos más cortos y entendibles mediante el uso de macros. Esto permite que las descripciones sean modulares, por lo que, no solo evita duplicar código, sino que favorece su reutilización. Por ello, todos los sensores que pueden ser empleados por algún robot se encuentran en un paquete llamado **robotnik_sensors**[2]. Éste tiene una macro para cada sensor que en algún momento ha sido incluido en algún robot, por lo que aquellos sensores nuevos que incluye el Rising deben ser creados en este paquete. Tanto la cámara PTZ como la rgbd son modelos nuevos que no habían sido utilizados en otros robots de la compañía. Por ello, han de ser incluidos en dicho paquete.

Para poder modelar cualquier objeto, ya sea un brazo, una cámara o un láser se debe conocer dónde están situados los elementos del dispositivo a modelar. Por ejemplo, en el modelado de la cámara PTZ se tienen en cuenta tres elementos, la base y los dos ejes de giro. Esto facilita la creación de estructuras más complejas, ya que importando dicha macro solo será necesario especificar la posición de cada sensor con respecto a la base del robot.

El brazo robótico tiene su propia descripción, desarrollada por la compañía que lo fabrica, por lo que se ha incluido la macro que ofrece el paquete kinova_description, el cual identifica a dicho brazo.

Por otro lado, para la cámara rgbd se ha necesitado conocer la posición de cada uno de los focos que tiene ya que cada uno ofrece una información distinta. El resto de elementos del robot, la base, las ruedas y las aletas se implementan

[2]Respositorio de robotnik_sensors https://github.com/RobotnikAutomation/robotnik_sensors

siguiendo las mismas pautas que el resto de componentes, cada uno se define dentro de una macro.

Una vez se han modelado cada componente por separado, se procede a crear el robot según la especificación de ROS[3]. Lo primero que establecemos es la base del robot, de la cual depende el resto de componentes del sistema. A continuación, se colocan las ruedas, estableciendo la posición de cada una de ellas. De esta manera se continúa con el resto de elementos creando el robot de la figura 5.4.

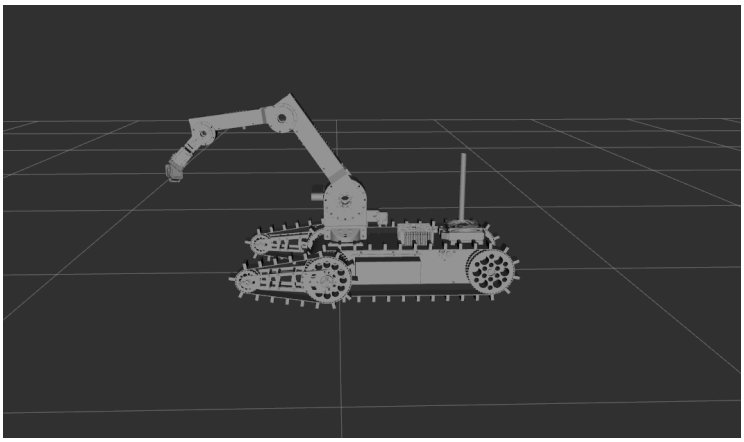

Figura 5.4: Representación visual de la descripción del Rising

La descripción del robot es una de las primeras tareas que hay que realizar ya que es necesaria para poder comenzar a desarrollar el resto de funcionalidades del robot. Esta descripción es utilizada por paquetes básicos (**robot_state_publisher**[4] y **ros_control**[5]) y necesarios para el funcionamiento de un robot.

Ros_control está compuesto por un conjunto de paquetes que toman como datos de entrada el estado de cada *joint* proveniente de los encoders del robot. Éste, utiliza un bucle de control genérico, típicamente un controlador PID, que controla la salida enviada a los actuadores. Estos *joints* deben ser los que se han definido previamente en la descripción.

Por otro lado, el robot_state_publisher utiliza esta descripción junto con el estado de cada *joint* para calcular la cinemática del robot y publicar los resultados, por lo que se puede conocer la posición de cualquier elemento del robot con respecto a otro en tiempo real.

Una posible aplicación en la que se podría utilizar esta información sería la de alcanzar objetos con el brazo. Para ello, con la utilización de la cámara rgbd no solo se podría detectar el objeto sino que se conocería la posición de éste con respecto a la cámara. Una vez se conoce esta información y con la utilización de la API que ofrece este paquete se dispondría de la información necesaria para comandar el brazo a dicha posición.

[3]REP 105, *Coordinate Frames for Mobile Platforms* http://www.ros.org/reps/rep-0105.html
[4]Documentación de robot_state_publisher http://wiki.ros.org/robot_state_publisher
[5]Documentación de ros_contorl http://wiki.ros.org/ros_control

5.2 Integración del brazo robótico

En este apartado se va a explicar los pasos necesarios para integrar cualquier brazo robótico con Moveit. Además, se va a mencionar los problemas que han surgido y cómo se han solucionado.

En primer lugar, para poder comunicarse con el brazo se han de instalar los drivers que Kinova ha desarrollado junto con el soporte para ROS, lo que permite el envío de comandos a través de los topics y servicios. Mediante estos, se puede manejar el brazo eje a eje, además de conocer en tiempo real el estado de cada *joint*.

Tras confirmar que funcionan todas las articulaciones, se procede a configurar el brazo mediante el paquete **Moveit**, el cual contiene el software más moderno para la manipulación móvil, incorporando los últimos avances en la planificación del movimiento, percepción en tres dimensiones, control y navegación [8]. Es el software de código abierto más usado para la manipulación. Los tres pilares de Moveit son:

- **Librería de funciones**: proporciona una biblioteca de habilidades robóticas para la manipulación, planificación de movimiento y control.
- **Comunidad**: tiene una comunidad de usuarios y desarrolladores que ayudan tanto en el mantenimiento del software como en la creación de nuevas aplicaciones.
- **Herramientas**: tiene una serie de herramientas que permiten a los nuevos usuarios integrar sus robots con dicho paquete, y a los más avanzados desplegar nuevas aplicaciones.

Para poder mover el brazo a través de Rviz, es necesario utilizar un plugin que ofrece, mediante el cual es posible comandar el brazo más fácilmente y, para ello, se ha empleado una de las herramientas que proporciona este paquete. Este asistente se compone de una interfaz gráfica para configurar cualquier robot con Moveit. Su principal característica consiste en generar un archivo SRDF, el cual complementa el archivo URDF ya que especifica grupos de articulaciones, información adicional sobre las colisiones y establecer ciertas posiciones prefijadas. Para obtener dicho archivo se ha seguido los distintos pasos que proporciona el asistente:

- **Matriz de colisiones**: por defecto, el generador de la matriz de colisión busca pares de links en el robot que no pueden colisionar, lo que reduce notablemente el tiempo de procesamiento a la hora de planear la trayectoria.
- **Grupos de planificación**: son usados para describir semánticamente las diferentes partes del robot, como el brazo o la mano. Además, se ha seleccionado KDL como solución cinemática.
- **Posiciones del robot**: el asistente permite añadir ciertas posiciones fijas en la configuración, como por ejemplo la posición de *home*.

Figura 5.5: Asistente de Moveit

- **Archivos de configuración**: es el último paso antes de poder utilizar el plugin en Rviz. Con ello, se generan todos los archivos necesarios para comenzar a usar Moveit.

Una vez acabada la configuración del brazo, éste estaría preparado para manejarlo a través del espacio. No obstante, el planificador no permitía la ejecución de la trayectoria hacia el punto seleccionado. Esto se debe a un problema con el cálculo de la cinemática inversa para manipuladores de cuatro DOF que la comunidad de ROS ya ha notificado.

Para solucionar dicho problema, se plantearon varias soluciones. La primera propuesta consistía en configurar el plugin IKFast para Moveit. Para ello se ha seguido la documentación[6] que existe para ello. IKFast es un potente solucionador de cinemática inversa que, a diferencia de otros solucionadores, puede resolver complejas cadenas cinemáticas y proporcionar el código C++ correspondiente para su posterior uso. Aunque, según la guía, solo ha sido probado con manipuladores de seis y siete grados de libertad, era una opción válida para solucionar dicho problema. Sin embargo no se pudo planificar ninguna de las trayectorias planteadas.

En la segunda se optó por modificar la descripción del brazo. La propuesta consiste en añadir dos *joints*, falsos, al modelo del brazo. Estos dos nuevos *joints* se han posicionado en el centro de la palma de la mano, sin entrar en contacto con ninguna parte como se puede observar en la figura 5.6. El primero que se ha añadido se encuentra perpendicularmente con respecto del último *joint* real que tiene el brazo. De la misma forma se ha colocado el segundo, también de forma perpendicular, pero en este caso con respecto de este último añadido. Además de estos se han creado sus respectivos *links*, sin embargo, estos son meramente ficticios, no se incluye ni elementos visuales ni modelo de colisiones. El resto de elementos que conforman el brazo se han mantenido con la misma configuración.

En este momento se ha procedido a utilizar de nuevo el asistente, pero esta vez con el nuevo modelo. De esta manera, se consigue que el planificador trate

[6]Guía para configurar el plugin IKFast http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html

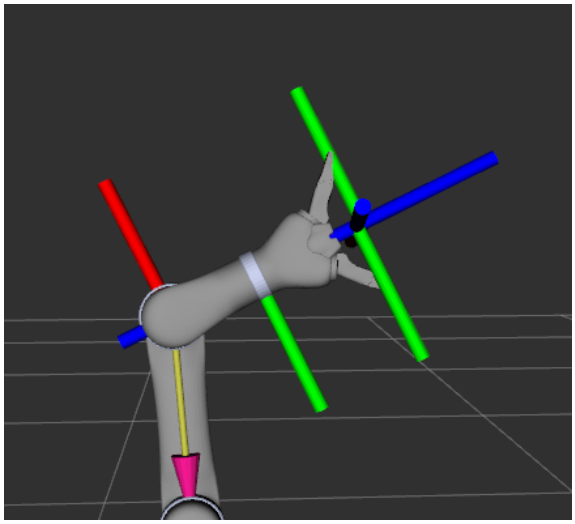

Figura 5.6: Los dos *joints* nuevos

de crear una trayectoria en la cual están implicadas las seis articulaciones del brazo. No obstante, sólo tendrá en cuenta cuatro de ellas, las reales, ya que independientemente de la posición final deseada, estas dos últimas que se han añadido no tendrán ningún peso dentro del planificador.

Esta ha sido la solución que se ha alcanzado para poder trabajar, por el momento, con los manipuladores de cuatro DOF. Una vez realizado todo ello, ya es posible mover el brazo mediante el plugin para Rviz.

5.3 Teleoperación del robot mediante un *joystick*

Por último, como se ha mencionado anteriormente el Rising está equipado con un joystick, un Wireless Controller PS4 más concretamente. A continuación se indica cómo está organizado el paquete rising_pad el cual incorpora todo el código implementado en este apartado.

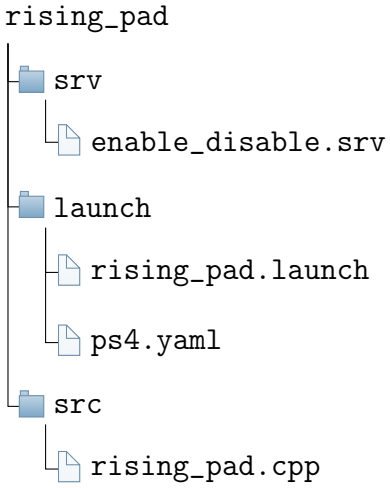

Figura 5.7: Estructura del paquete rising_pad

En el directorio **launch** se encuentran los archivos que se utilizan para lanzar los nodos necesarios para el correcto funcionamiento del *joystick*. El archivo ps4.yaml contiene la configuración del mando en la que se identifica de forma única a cada botón y eje. El otro fichero contiene la especificación XML para lanzar los nodos. En éste se lanzan dos nodos, el primero es joy_node del paquete de ROS joy[7]. Es realmente útil ya que es el driver de ROS que transforma los datos de entrada del *joystick* en mensajes estándar de ROS. El segundo se trata del que se ha desarrollado para este apartado y es el rising_pad.cpp, el cual utiliza los parámetros que se han mencionado anteriormente.

En segundo lugar, el directorio **srv** incorpora la especificación de un servicio específico para este paquete. Se trata de enable_disable.srv cuya definición incorpora un bool tanto en la petición como en la respuesta. Este será el tipo que utilice el servicio del nodo que active y desactive el uso del mando.

La tercera carpeta nombrada **src** incluye el código fuente que permite el control tanto de la base móvil como del brazo. El archivo rising_pad.cpp contiene el código para mover tanto la base móvil como el brazo. En primer lugar se ha establecido la definición de todos los parámetros que utiliza el nodo, que se obtienen del archivo 'yaml' que se ha comentado previamente.

Por otro lado, se crean un publicador, un subscriptor y un servicio. El servicio se llama enable_disable_pad y se encarga de habilitar o deshabilitar la publicación de mensajes del joystick. El publicador se encarga de publicar la velocidad en el topic 'rising_pad/cmd_vel' el cuyos datos son leídos desde el nodo twist_mux, ya mencionado anteriormente. El suscriptor se subscribe al topic 'joy' que está publicando el nodo joy_node (cuyo tipo de mensaje se muestra a continuación), mediante el cual se publican los valores de caja eje, así como qué botones están presionados. Dicho suscriptor tiene asociado una función callback que se ejecutará cada vez que un mensaje nuevo aparece en el topic al que se ha suscrito.

```
- - -
header:
  seq: 9415
  stamp:
    secs: 1325530130
    nsecs: 146351623
  frame_id: ''
axes: [-0.003875850699, -0.003845332190, 0.0, -0.99996948, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- - -
```

Como medida de seguridad se ha establecido un botón de hombre muerto, de esta manera, si no está pulsado dicho botón el robot no podrá ser controlado manualmente. Por defecto, la velocidad que el *joystick* publica es un 10 % de la velocidad máxima del robot. Para modificar esta velocidad se han habilitado dos botones que permiten aumentar o disminuir la velocidad a la que avanza. Por úl-

[7] Documentación de joy http://wiki.ros.org/joy

timo, para acabar de controlar todos los elementos de la base, además de disponer de otros dos botones que permiten mover las aletas del robot hacia la dirección deseada, se ha habilitado cuatro más (las flechas de dirección) que permiten el control de la cámara PTZ.

Por otro lado, siguiendo el mismo planteamiento que se ha mencionado anteriormente, disponemos de un botón de hombre muerto, en este caso para el brazo, de esta manera no se podrá comandar el brazo sin mantenerlo pulsado. No solo es una medida de seguridad, sino que permite usar los mismos botones que se han utilizado para el control de la base con funcionalidades totalmente distintas.

Cuando el control del brazo es de forma manual, éste se maneja eje a eje, por ello se dispone de un botón que cambia la articulación a mover. Una vez seleccionada la que que se desea controlar, solo hace falta mover el *joystick* en la dirección correcta para mover dicha articulación. También se dispone de un botón que permite mover la pinza a sus dos posiciones, abierta o cerrada.

Por último, cabe destacar que si el brazo se integra con Moveit sería posible seleccionar una de las posiciones predeterminadas que se han establecido mediante el asistente. Así pues, si primeramente se establecen algunas, como el brazo recogido, luego podrán ser seleccionadas mediante un botón para su posterior ejecución. De esta manera, se evita que posiciones habituales del brazo sean repetidas continuamente disminuyendo el tiempo que se emplea para cada movimiento.

CAPÍTULO 6
Implantación

En este capítulo se va a presentar la etapa de implantación de la solución. Para ello, no solo hay que instalar el software en el ordenador del robot, sino que hay que configurar todos los archivos que sean necesarios para su correcto funcionamiento.

En primer lugar, se ha instalado Ubuntu 16.04 ya que el software se ha desarrollado en ROS kinetic. Tras la instalación del sistema operativo y de kinetic se procede a la creación del espacio de trabajo (*workspace*) en ROS, en el cual se va a incluir todos los paquetes mencionados en la sección 4.2. Una vez se han incluido todos los paquetes necesarios, es posible compilar el *workspace* para proseguir con la implantación del sistema. Si todo ha ido bien, el proceso de compilación terminará satisfactoriamente, tras lo cual será posible lanzar los nodos que permitirán el control del robot. No obstante, en vez de lanzar cada nodo de forma individual, se ha creado un archivo 'launch' que incluya todos los nodos necesarios para que funcione todo el sistema. De esta manera con un solo archivo es posible lanzar el sistema completo. Para ello se deberá ejecutar el siguiente comando:

```
> roslaunch rising_bringup rising_complete.launch
```

El primer argumento del comando es el nombre del paquete, que se ha creado específicamente para contener tanto los archivos de configuración específicos para este robot como el archivo launch que se ha comentado previamente, el cual es el segundo argumento.

Por otro lado, se ha establecido que dicho comando se ejecute siempre al arranque del robot. Así, es posible controlarlo sin necesidad de conectarse al robot para iniciarlo. Para ello, se ha empleado las **tty**, que son las terminales que nos ofrece linux por defecto.

Editando el archivo .bashrc del sistema y configurando las tty1, tty2 y tty3 para que se inicien automáticamente, solo hace falta incluir, en dichas terminales, el driver del *joystick*, el roscore y el archivo rising_complete.launch, para que se lance todo lo necesario siempre que se inicia el robot. Llegados a este punto, ya se puede dirigir el robot hacia el objetivo deseado, además de manejar el brazo. A continuación se muestra cómo configurar dichas terminales:

```
> sudo systemctl edit getty@ttyX
```

En el editor de la terminal, se deben copiar las siguientes líneas:

```
[Service]
ExecStart=
ExecStart=-/sbin/agetty --autologin rising --noclear %I 38400 linux
```

Una vez editado, se procede a habilitarlas:

```
> systemctl enable getty@ttyX.service
```

Donde X es el número de terminal que se desea habilitar. En este caso se han activado la uno, la dos y la tres.

A continuación se muestra el contenido que se ha añadido al final del archivo .bashrc:

```
# RISING
# AUTOBOOT
    Terminal=‘tty‘
    case $Terminal in
        "/dev/tty1") ds4drv;;
        "/dev/tty2") roscore;;
        "/dev/tty3") sleep 15;
        roslaunch rising_bringup rising_complete.launch;;
    esac
```

Figura 6.1: Script en el archivo .bashrc

Como se ha mencionado, así se consigue ejecutar el driver del *joystick*, el roscore y el rising_complete.launch en paralelo, desde distintas terminales al arranque del ordenador.

Por último, al ser un sistema en el que el usuario puede modificar o añadir funcionalidades nuevas, es posible tener acceso a todo el entorno de ROS sin necesidad de realizar ninguna conexión ssh o conectar un monitor al robot. Para ello, solo hay que configurar dos cosas en el ordenador del usuario. La primera, el archivo hosts que se encuentra en el directorio /etc/, en el cual se debe incluir la IP y hostname del robot. La segunda consiste en exportar la variable de entorno de ROS llamada ROS_MASTER_URI, cuyo valor será: http://rising:11311/.

En este momento, ya sería posible comenzar a trabajar desde un ordenador distinto al del robot, lo cual no solo permite desarrollar nuevas cosas sino que facilita la detección de errores o comportamientos extraños.

CAPÍTULO 7
Pruebas

En este apartado se va a comentar las pruebas que se han realizado para verificar el funcionamiento del robot. Para ello, se han planteado distintas situaciones en las que el robot debe ser capaz de completar el objetivo.

En primer lugar, se procedió a realizar pruebas con el brazo, ya que son las más fáciles de ejecutar. Se efectuaron movimientos en cada uno de sus ejes, así como la apertura y cierre de la pinza con un objeto en su extremo.

Figura 7.1: La primera versión del brazo del Rising

Aunque los resultados son satisfactorios, surgió una situación en la que la actuación del brazo se encontraría limitada. Si un objeto se encuentra a un lado del robot, para alcanzarlo el robot debería girar su base y posteriormente mover el brazo. Esto se debe a que los ejes del brazo realizan su moviemiento en la misma dirección como se puede observar en la figura 7.1. Por tanto, es posible que en zonas estrechas la utilización del manipulador se viese afectada.

Tras las primeras pruebas con el brazo, se ha proseguido con las de la base móvil.

La primera prueba con la base móvil consiste en que el robot sea capaz de moverse por una zona plana, realizando tanto movimientos rectilíneos como curvilíneos. Este simple movimiento es básico, pero permite a los distintos departamentos involucrados en su creación poner a prueba los distintos componentes del

robot. De esta manera se comprueban dos cosas, el hardware y el software. Tras los primeros movimientos se ha podido confirmar que el robot está preparado para la siguiente situación.

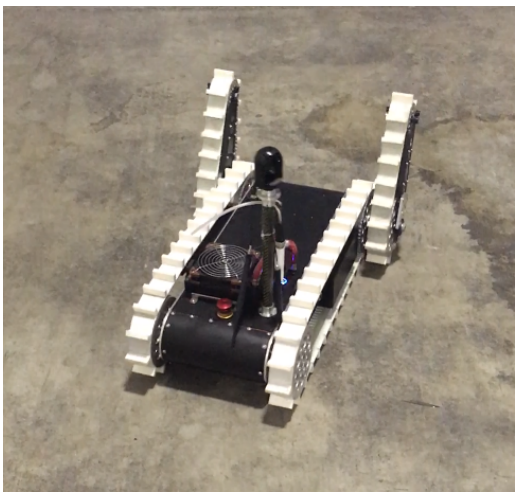

Figura 7.2: El Rising sobre un terreno plano

Aunque la primera prueba es clave para determinar posibles fallos de software (y de hardware), la monitorización de todo el sistema en el resto de situaciones, a cada cual más compleja, estará presente para poder tomar decisiones.

La segunda prueba es una continuación de la primera pero en un terreno distinto. En esta ocasión se dispone de una moqueta en la zona de pruebas, donde el robot va a describir distintos movimientos para comprobar si las ruedas se enganchan sobre esa superficie. No solo se verifica su estabilidad sino también permite la verificación del cálculo odométrico, ya que un mal agarre (o un derrape) afectaría negativamente a dicho cálculo.

En la tercera situación se optó por incrementar el nivel de dificultad a las pruebas, para ello se planteó subir y bajar unas escaleras. Para trepar dichas escaleras, el usuario debe orientar las aletas a la altura adecuada para que las cadenas hagan buen contacto. Como se muestra a continuación el robot logró el objetivo planteado.

Figura 7.3: El Rising subiendo unas escaleras

Por otro lado, cabe destacar que estas son las pruebas finales que el robot ha sido capaz de superar, aunque no han sido las únicas ya que durante todo el proceso de desarrollo se ha ido comprobando el funcionamiento de todo el sistema para detectar posibles errores.

Por último, solo queda por comprobar si finalmente el brazo de Kinova de cuatro DOF es capaz de planificar y realizar movimientos. Para ello, esta prueba se realizará mediante el plugin creado para Rviz.

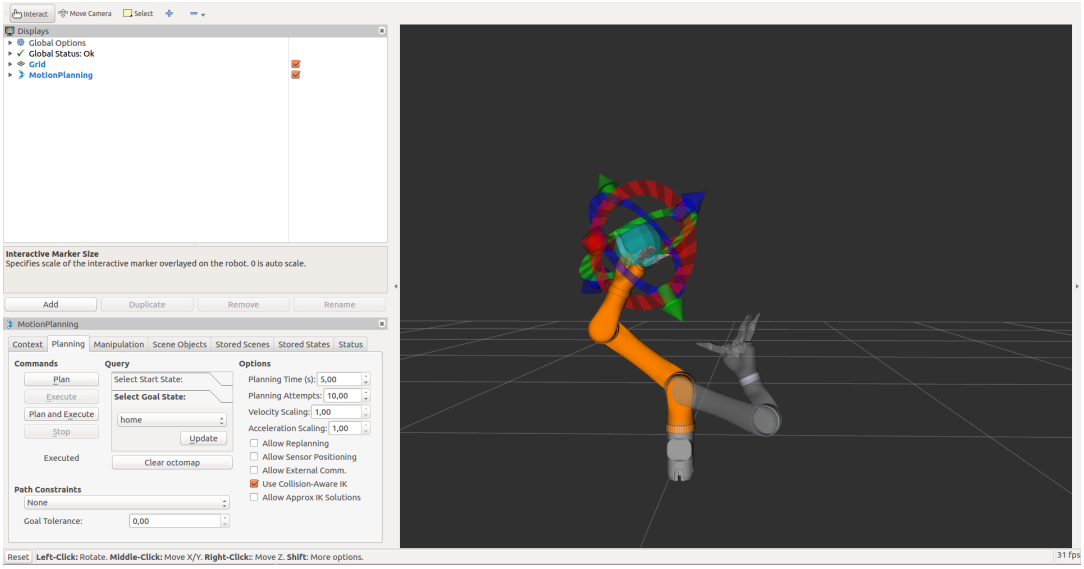

Figura 7.4: El brazo de Kinova planificando una trayectoria

Como se puede apreciar en la figura 7.4 hay una esfera, la cual se utilizará para mover el brazo por la escena. Una vez se alzance la posición deseada se podrá planificar y ejecutar. Finalmente, la solución planteada al problema del cálculo de la cinemática inversa ha resultado ser satisfactoria.

CAPÍTULO 8
Conclusión

El desarrollo de este trabajo ha implicado un continuo aprendizaje en el ámbito de la robótica, por ejemplo, cómo funciona todo el hardware implicado en el robot, desde saber qué protocolos se usan hasta conocer los distintos tipos de conexiones que ofrecen los sensores. Además, ha sido fundamental el aprendizaje de nuevas herramientas software como ROS, ya que todos estos conocimientos adquiridos han sido imprescindibles para alcanzar con éxito los objetivos inicialmente planteados en este trabajo.

Se ha desarrollado el modelo URDF del robot, el cual se encarga de modelar el robot. En él se incluye la cinemática y dinámica, así como la representación visual del mismo, entre otras características.

Por otro lado, debido a un problema con la versión empleada de ROS junto con Moveit en el cálculo de la cinemática en brazos con cuatro DOF, resultaba imposible planificar cualquier trayectoria, por tanto solo era posible trabajar eje a eje. Tras estudiar distintas posibles soluciones, se optó por una cuyos resultados satisfacen con éxito el objetivo inicialmente planteado, pudiendo así realizar trayectorias complejas.

Por último, el control tanto de la base como el del brazo cumple los requisitos de funcionalidad que se esperan en un robot de ese tipo controlado mediante un joystick, ya que durante el proceso de desarrollo, distintas personas lo manejaron con la intención de comprobar si el sistema era intuitivo.

8.1 Relación con los estudios cursados

La implementación de este trabajo ha sido posible a los conocimientos adquiridos en asignaturas como Mecatrónica (MEC). Aunque la tecnología empleada no ha sido vista en clase, los conceptos obtenidos en dicha materia han sido de gran ayuda tanto para comprender el hardware involucrado como las distintas partes software que componen un robot.

Por otro lado, asignaturas como Ingeniería del Software (ISW) o Gestión de Proyectos (GPR) aportan conceptos sobre el desarrollo del software, tanto en la implementación como en la planificación del mismo, que han sido utilizados de forma intrínseca.

Cabe destacar que la participación en varias ocasiones en concursos ofrecidos por la UPV como 'Hack For Good' o algún 'Challenge IDEAS-UPV', han permitido el desarrollo de competencias transversales como por ejemplo el análisis y resolución de problemas, creatividad, emprendimiento, trabajo en equipo, y otras que han sido llevadas a la práctica en la elaboración de este TFG.

CAPÍTULO 9
Trabajos futuros

En este capítulo se va a presentar alguna de las líneas de desarrollo que se abren tras la realización de este trabajo.

Actualmente, solo se ha integrado el brazo con Moveit, habiendo resuelto la problemática con la planificación de trayectorias. Por ello, tras las pruebas con la primera versión del brazo, la línea de desarrollo que surge a partir de esa solución, es la integración de la base y el brazo con dicho paquete. De esta manera, la base del robot será un elemento de colisión más que el planificador tendrá en cuenta cuando calcule las trayectorias para alcanzar el objetivo.

Siguiendo la línea anterior, una vez esté integrado será posible usar la API que ofrece dicho paquete para desarrollar aplicaciones en las que esté involucrado alcanzar objetos con el brazo. Es por ello que surgen nuevos vías de desarrollo, como la visión por computador, para crear programas más completos.

Por último, la posibilidad de navegación autónoma con dicho robot. Esto implica que el robot también sea capaz de localizarse dentro del mapa por el cual se desea navegar autonomamente.

Aunque pueden existir más líneas de desarrollo, estas son las que más encajan en el perfil del robot.

Bibliografía

[1] Definición de robótica. Disponible en https://definicion.de/robotica/. Consultado el *05 / 09 / 2018*.

[2] ELKADY, Ayssam; SOBH, Tarek. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, vol. 2012.

[3] CERIANI, Simone; MIGLIAVACCA, Martino. *Middleware in robotics*. Internal Report For „Advanced Methods of Information Technology for Autonomous Robotics", Politecnico di Milano, 2012.

[4] ANIS, Koubaa. ROS as a service: web services for robot operating system. *JOURNAL OF SOFTWARE ENGINEERING IN ROBOTICS*, 2015, vol. 6, no 1, p. 1-14.

[5] Sobre ROS. Disponible en www.ros.org/about-ros/ Consultado el *05 / 09 / 2018*.

[6] Historia de ROS. Disponible en http://www.ros.org/history/. Consultado el *05 / 09 / 2018*.

[7] Ejemplo de ROS Máster. Disponible en http://wiki.ros.org/Master. Consultado el *05 / 09 / 2018*.

[8] KOUBAA, Anis. Robot operating system (ROS). 2017.

APÉNDICE A
Glosario

En este apartado se explican algunos de los términos más específicos del trabajo:

- **Odometría**: es el estudio de la estimación de la posición relativa a la posición inicial del vehículo.
- **CAN bus** es un protocolo de comunicaciones utilizado para la transmisión de mensajes en entornos distribuidos.
- **P2P**: también conocido como *peer-to-peer*. Consiste en una serie de nodos que se comportan como iguales entre sí, es decir, todos actúan como clientes y servidores respecto a los demás nodos de la red.
- **PTZ**: sus siglas en inglés significan *pan*, *tilt* y *zoom*. Las cámaras PTZ rotan sobre dos ejes, el horizontal (*pan*) y el vertical (*tilt*), así como acercarse o alejarse (*zoom*).
- **DOF**: significa grados de libertad (degree of freedom).