Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería de Telecomunicación

MOVIMIENTO DE UN BRAZO ROBÓTICO EN EL ENTORNO EDUCATIVO UNIBOTICS

Memoria del Trabajo Fin de Grado

en Ingeniería en Sistemas Audiovisuales y Multimedia

Autor: Ignacio Malo Segura

Tutor: José María Cañas Plaza

Co-tutor: José Francisco Vélez Serrano

2019

Agradecimientos

Resumen

Capítulo 1

Introducción

1.1.

1.2.

1.2.1.

1.2.2.

Capítulo 2

Objetivos

2.1.

Capítulo 3

Infraestructura

En este capítulo se detalla el software necesario para el desarrollo de las aplicaciones construidas. Desde ROS Kinetic y JdeRobot como middlewares robóticos, pasando por la herramienta MoveIt! para facilitar el trabajo con robots(PR2) hasta el simulador Gazebo. Todo ello corriendo en el sistema operativo Ubuntu 16.04 y desarrollado en Python y C++. Las comunicaciones entre los distintos componentes utilizan el motor de comunicaciones ICE además de ROS topics.

3.1. PR2

El robot sobre el que se han desarrollado las aplicaciones es el PR2 de Willow Garage, empresa especializada en robótica responsable también del software ROS y otras herramientas open-source. Contiene dos brazos(derecho e izquierdo)capaces de generar un amplio abanico de oportunidades de movimiento que no serían posible con un único brazo, y que terminan en una pinza(gripper) que permite agarrar objetos. El nombre viene de 'Personal Robot 2', haciendo referencia a la idea de que fuera un asistente en casa y no un robot industrial como la mayoría de los comercializados hasta el momento. Su diseño modular permite integrarlo con otros grippers, brazos o sensores. Su muñeca tiene dos grados de movimiento, que se suman a la libertad que aportan las articulaciones del hombro y el codo para permitir prácticamente cualquier movimiento necesario para una labor doméstica.

Además de su estructura, una característica fundamental del robot PR2 son las cámaras y láseres que permiten conocer el entorno y actuar según lo que percibe: detectar objetos, saber a qué distancia se encuentran e identificar zonas objetivo permiten llevar a cabo acciones como abrir una puerta o llevar objetos de un lugar a otro.

3.2. Gazebo

Este simulador open source nació en el año 2002 en la Universidad del Sur de California(USC) como proyecto entre un profesor y uno de sus estudiantes. El objetivo era poder trabajar con robots bajo condiciones de alta fidelidad, pensando principalmente en entornos exteriores. Se comenzó a integrar con ROS en 2009 y dos años después la citada

Willow Garage comenzó a financiar el desarrollo para este software, del que se hizo cargo la Open Source Robotics Foundation(OSRF). Gazebo fue adaptado para participar en la prestigiosa DARPA Robotics Challenge (DRC) en el año 2013. Hay una comunidad activa detrás del proyecto que permite introducir mejoras, resolver bugs y consultar dudas técnicas.

Como simulador de robótica, lo que ofrece es crear aplicaciones para robots sin necesidad de depender de la máquina física. Así, las aplicaciones creadas para un modelo concreto podrán utilizarse posteriormente en el mundo real sin necesidad de realizar modificaciones, disminuyendo tanto los costes tanto económicos como derivados de la peligrosidad de las pruebas en fase de desarrollo. Ofrece visualización 3D y un potente motor de físicas para, en el caso de un brazo robótico, poder determinar parámetros como fricción, rango de movimiento, colisiones etc.

La simulación nace a partir de un fichero launch que arranca Gazebo. El entorno de la simulación está compuesto por un world que contiene diferentes models. Estos modelos están definidos en un fichero con formato SDF con los siguientes componentes principales:

- Links: contiene las propiedades físicas de un trozo del modelo, incluyendo algunas de visualización y colisión, sensores, inercias o iluminación.
- Joints: conexión entre dos links.
- Plugins: librería externa que controla un modelo.

La versión utilizada para este proyecto es Gazebo7.

3.3. JdeRobot

Este software open source permite desarrollar aplicaciones en el campo de la Robótica a partir de código C++, Python o JavaScript. La compatibilidad con ROS, principalmente con la versión Kinetic es una de las grandes ventajas de esta herramienta. Los diferentes nodos o componentes se comunican a través del framework ICE(Internet Communications Engine) o de mensajes ROS. Ambas opciones permiten realizar dichas comunicaciones de forma agnóstica al lenguaje en el que están desarrollados los componentes. De esta estructura parte JdeRobot-Academy, una iniciativa para el aprendizaje en Robótica y Visión Computacional que incluye múltiples ejercicios para que los estudiantes puedan entender y añadir su propio código. JdeRobot contiene una amplia variedad de componentes y librerías que pueden ser reutilizados por la comunidad de desarrolladores. La última versión estable es la 5.6.4, lanzada en mayo de 2018. Uno de los últimos hitos es su participación en el Google Summer of Code 2018.

3.4. ROS Kinetic

Framework para el desarrollo de software para robots. Nació en 2007 en el Laboratorio de Inteligencia Artificial de Standford. La arquitectura de ROS(Robot Operating System) se basa en nodos que se comunican a través de mensajes, lo que nos permite obtener grafos

fácilmente. Es un sistema de código abierto que encarga de mantener la OSRF(Open Source Robotics Foundation). Aporta al usuario herramientas como por ejemplo planificación de movimiento(MoveIt) o reconocimiento del entorno y visualización (Rviz), además de soporte para un amplio abanico de Robots. La versión Kinetic fue lanzada en mayo de 2016, enfocada al uso desde Ubuntu 16.04. La versión Gazebo7 es la recomendada en ROS Kinetic para este simulador. -Estructura y comunicaciones Un paquete de ROS agrupa varios programas o nodos con funcionalidades similares. Estos nodos se comunican a través de mensajes, llamados ROS Topics, que les permiten interactuar entre sí. ROS Core es la herramienta encargada de arrancar el nodo máster y gestionar todas las comunicaciones, por lo que es la primera que debe ser arrancada. Es lanzado automáticamente con los ficheros .launch, que serán muy habituales en el desarrollo.

```
~$ roscore
```

Para ejecutar un nodo, necesitaremos el paquete y el nombre del nodo:

```
~$ rosrun turtlesim turtlesim_node
```

Una vez lanzado el nodo, podemos publicar topics de un tipo concreto que el nodo comprenda, consiguiendo así que realice una acción determinada. Estas comunicaciones son la base del desarrollo con robots en ROS.

```
~$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist --
   '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

También se pueden obtener por línea de comandos los mensajes publicados en un topic determinado.

```
~$ rostopic echo /turtle1/cmd\_vel
```

ROS dispone de herramientas como rqt que permiten observar grafos de las comunicaciones actuales. Si seguimos con el ejemplo de turtlesim:

```
$ rosrun rqt_graph rqt_graph
```

Podemos ver en cualquier momento los nodos o topics disponibles con los siguientes comandos:

```
~$ rosnode list
~$ rostopic list
```

3.5. ICE

Se trata de un framework RPC(Remote Procedure Call) compatible con lenguajes como C++, C, Java, JavaScript o Python. Nace a partir de un modelo cliente-servidor con el objetivo de permitir la comunicación entre distintos lenguajes y sistemas operativos. Esta abstracción ofrece al desarrollador la capacidad de despreocuparse por cómo se gestionan las comunicaciones entre los diferentes módulos para así enfocarse en el problema a resolver. Permite comunicaciones bidireccionales, tanto síncronas como asíncronas.

Fue diseñado para aplicaciones que requieren un performance y escalabilidad exigentes, minimizando el consumo de CPU y ancho de banda. IceSSL permite añadir seguridad a las comunicaciones, a través de autenticación y encriptación de los datos que viajan en las llamadas.

3.6. C++

Este lenguaje de programación fue creado por Bjarne Stroustrup en 1979, que lo bautizó en sus inicios como 'C con clases'. No en vano tiene una sintaxis similar a la de C, manteniendo el vínculo con el lenguaje del que proviene y añadiendo mecanismos de programación orientada a objetos a su predecesor. Su nombre además hace referencia a ese C incrementado, mejorado. Es un lenguaje fuertemente tipado, combinado y portátil. Se rige por un estándar de ISO(International Organization for Standardization) cuya última versión es C++17, lanzada ese mismo año.

El hecho de ser un lenguaje con un grado de complejidad elevado supone una gran versatilidad y potencia, pero también una curva de aprendizaje elevada para dominarlo. Algunos IDEs recomendados para trabajar con C++ son VisualStudio y Code:: Blocks. Como curiosidad, se trata del tercer lenguaje de programación más popular en 2018 según el conocido ranking de TIOBE, sólo por detrás de Java y C. Alcanzó su máximo de popularidad 2003 según los datos de esta empresa de software. (https://www.tiobe.com/tiobe-index/)

3.7. Python

Fue creado en los años 80 por Guido van Rossum. Es un lenguaje de programación interpretado, que utiliza tipado dinámico. Su sintaxis está enfocada a ser fácilmente legible, lo que hace que tenga una curva de aprendizaje suave en relación con otros lenguajes como Java o el propio C. Soporta programación orientada a objetos y es multiplataforma.

Una característica interesante es que puede extenderse con módulos de C o C++. Según el mencionado ranking TIOBE, Python es el cuarto lenguaje de programación en popularidad llegando al máximo nivel en 2011.

Se ha utilizado la versión 2.7.12 debido principalmente a su compatibilidad con ROS Kinetic. En este lenguaje se han desarrollado las aplicaciones de planificación de movimientos y pickplace.

3.8. MoveIt!: Motion Planning Framework

MoveIt! nació en octubre del año 2011 con la idea de agrupar todos los avances relacionados con planificación de movimientos, manipulación, percepción 3D, cinemática, control y navegación en una única herramienta. Actualmente es el software open-source más utilizado para manipulación con robots, con más de 65 autómatas soportados. Tiene un nodo principal llamado move_group que actúa como integrador, permitiendo a todos los componentes comunicarse entre sí realizando las acciones soportadas. Se puede acceder a move_group por tres vías: C++, Python y RVIZ. Como se puede ver en el esquema, move_group necesita varios parámetros de entrada:

- URDF(Universal Robot Description Format): Es el formato que utiliza ROS para la descripción del robot. Se ha utilizado el del paquete 'pr2_description', correspondiente al autómata utilizado.

- SRDF(Semantic Robot Description Format) complementa al URDF, añadiendo más información como grupos de joints, configuraciones por defecto para el robot o chequeo de colisiones. Para construir este fichero, se recomienda utilizar el MoveIt! Setup Assistant. Esta herramienta permite configurar cualquier robot para ser utilizado en MoveIt! A través de una interfaz gráfica que simplifica el trabajo.
- MoveIt! Configuration: Incluye otros ficheros de configuración específicos de MoveIt!, normalmente generados también a través del Setup Assistant. Contienen información necesaria para planificación de movimientos, cinemática o percepción del entorno.

Las principales interfaces de MoveIt! Permiten acceder a las diferentes funcionalidades a través de código C++. Sin embargo, existe un paquete llamado moveit_python que permite acceder a las principales interfaces de MoveIt! Utilizando python:

- MoveGroupInterface: Utilizada para acceder a move_group y mover el brazo.
- PlanningSceneInterface: Utilizada para añadir o eliminar objetos al entorno y cambiar su apariencia, ya sean conectados o de colisión.
- PickPlaceInterface: Utilizada para realizar acciones de pickplace.

Con el objetivo de conseguir la integración de MoveIt! con Gazebo, y poder así enviar los movimientos al robot real(o simulado), debemos configurar correctamente los controladores. Para ello necesitaremos dos ficheros clave: controllers.yaml y joint_names.yaml. En ellos se especifican el tipo de mensajes que el robot recibe y las joints que controla.

Capítulo 4

Aplicación con brazo robótico y planificación de trayectorias

Capítulo 5

Conclusiones

Bibliografía