UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

ESCUELA TÉCNICA
SUPERIOR INGENIEROS
INDUSTRIALES VALENCIA

Curso Académico:

AGRADECIMIENTOS

"A mi familia, en especial a mi madre, a mi padre y a mi hermana por ser siempre un apoyo fundamental

A Ángeles por creer en mí y ayudarme a que yo lo hiciera

A mi tutora por toda su ayuda, paciencia y dedicación

A mis compañeros y amigos por todos estos años de esfuerzo, risas y experiencias

A Jorge por estar a mi lado en todo momento"

RESUMEN

El trabajo final de grado que a continuación se expone pertenece al campo de la robótica, y en él se propone como objetivo el desarrollo de una aplicación basada en cámaras 3D para la generación de movimientos en un robot humanoide. Es decir, con ello se pretende crear un código de programa que junto con otras herramientas y sincronizado a la vez con la cámara 3D, permita al robot humanoide de trabajo copiar los movimientos que realiza un usuario.

Para lograr este objetivo propuesto es necesario realizar un proceso de aprendizaje que se desarrolla a lo largo de la siguiente memoria. Entre los aspectos más importantes destaca el uso y comprensión de un nuevo entorno de programación llamado ROS, cuyo campo de aplicación en el mundo de los robots es muy extenso. En este entorno, construido para desarrollarse en el sistema operativo Linux, se fundamenta la gran base de este proyecto y es por ello que a lo largo de este documento se explican las diferentes herramientas que han sido de utilidad. Entre ellas, destaca la interfaz gráfica RViz de MoveIt sobre la cual se han realizado todas las pruebas previas a la conexión con el robot humanoide NAO.

Por otra parte, también se ha trabajado con una cámara 3D, en este caso con el sensor Kinect, capaz de captar los movimientos que un usuario realiza frente a este dispositivo para que después éstos puedan ser replicados por el robot humanoide, consiguiendo así el objetivo propuesto.

Palabras clave: robots humanoides, control automático, sistemas empotrados, cámaras 3D, programación modular, ROS.

RESUM

El treball fi de grau que a continuació s'exposa pertany al camp de la robòtica, i en ell es proposa com objectiu el desenvolupament d'una aplicació basada en càmeres 3D per a la generació de moviments en un robot humanoide. És a dir, amb açò es preten crear un codi de programa que junt amb altres ferramentes i sincronitzat a la vegada amb la càmera 3D, permeta al robot humanoide de treball copiar els moviments que reaitza un usuari.

Per aconseguir aquest objectiu proposat és necessari realitzar un procés d'aprenentatge que es desenvolupa al llarg de la següent memòria. Entre els aspectes més importants destaca l'ús i comprensió d'un nou entorn de programació anomenat ROS, el camp d'aplicació del qual en el món dels robots és molt extens. En aquest entorn contruït per a desenvolupar-se en el sistema operatiu Linux, es fonamenta la gran base d'aquest projecte i és per això que al llarg d'aquest document s'expliquen les diferents ferramentes que han sigut d'utilitat. Entre elles destaca la interfaç gràfica RViz de MoveIt sobre la qual s'han realitzat totes les proves previes a la conexió amb el robot humanoide NAO.

Per un altra banda, també s'h atreballat amb una càmera 3D, en aquest cas amb el sensor Kinect, capaç de captar els moviments que un usuari realitza en front d'aquest dispositiu per a que després aquestos puguen ser replicats pel robot humanoide, conseguint així l'objectiu propost.

Paraules clau: robots humanoides, control automàtic, sistemes empotrats, càmeres 3D, programació modular, ROS.

ABSTRACT

The final work of degree that later is exposed belongs to the field of the robotics, and it has the objective the development of an application based on 3D cameras for generating motion in a humanoid robot. That is to say, with it one tries to create a program code that together with other tools and synchronized simultaneously with the 3D camera, allows the humanoid robot of work to copy the movements made by a user.

To achieve this objective it is necessary to realize a learning process that develops along the following memory. Between the most important aspects emphasizes the use and comprehension of a new environment of programming called ROS, whose scope in the world of robots is very extensive. In this environment, constructed to develop in the operating system Linux, there is based the great base of this project and that is why throughout this document explains the different tools that have been helpful. Between them, emphasizes the graphical interface RViz of MoveIt on which there have realized all the tests before the connection with the humanoid robot NAO.

On the other hand, it has also worked with a 3D camera, in this case with the sensor Kinect, capable of catching the movements that a user realizes in front of this device in order that later these could be answered by the humanoid robot, obtaining the goal proposed.

Keywords: humanoid robots, automatic control, embedded systems, 3D cameras, modular programming, ROS.

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFG

- Memoria
- Presupuesto

ÍNDICE DE LA MEMORIA

CAPÍTULO 1. INTRODUCCIÓN 12
1.1. OBJETIVOS DEL TRABAJO 12
1.2. ESTRUCTURA DEL DOCUMENTO 13
CAPÍTULO 2. DESARROLLO TEÓRICO 16
2.1. ROBOTS HUMANOIDES 16
2.1.1. ¿Qué es un Robot Humanoide? 16
2.1.2. Tipologías de robots 17
2.1.3. Evolución histórica de los robots humanoides 19
2.1.4. Robot del proyecto: NAO 20
2.2. ENTONOS DE PROGRAMACIÓN 21
2.2.1. Choregraphe 21
2.2.2. ROS 22
2.2.2.1. MoveIt! 25
2.3. CÁMARAS 3D 27
2.3.1. Kinect 27
CAPÍTULO 3. DESARROLLO PRÁCTICO 30
3.1. INTRODUCCIÓN Y PLANTEAMIENTO 30
3.2. MATERIAL Y EQUIPO 31
3.3. CONFIGURACIÓN INICIAL: ROS Y MOVEIT 31
3.3.1. Uso de la interfaz RViz 33
3.3.1.1. Introducción 33
3.3.1.1. RViz con NAO 33
3.4. CONFIGURACIÓN DE LA KINECT 34
3.5. DESARROLLO DEL NODO 40

3.5.1. Introducción 40
3.5.2. Control del movimiento mediante la API de C++ 40
3.5.3. Posiciones máximas y mínimas de las articulaciones del tronco superior 44
3.5.4. Transformación del espacio de la Kinect al espacio del robot humanoide 48
3.5.4.1. Introducción 48
3.5.4.2. Configuración 48
3.5.4.3. tf en ROS 50
3.5.4.4. Transformaciones tf en el nodo 50
3.5.5. Creación de un nuevo grupo 54
3.5.6. Planificación de movimientos simples. 57
3.6. LANZAMIENTO DEL NODO 60
CAPÍTULO 4. CONLUSIONES Y TRABAJOS FUTUROS 64
CAPÍTULO 5. BIBLIOGRAFÍA 66

ÍNDICE DEL PRESUPUESTO

PRESUPUESTO 70
1. NECESIDAD DEL PRESUPUESTO 70
2. ESTUDIO ECONÓMICO 70
2.1. COSTE DE PERSONAL 70
2.2. COSTE MATERIAL INVENTARIABLE 71
2.3. COSTE MATERIAL FUNGIBE 72
3. RESUMEN DEL PRESUPUESTO 73

MEMORIA DESCRIPTIVA

CAPÍTULO 1. INTRODUCCIÓN

A día de hoy, el mundo de la robótica avanza con rapidez y no es para menos ya que muchos de los progresos y avances que han visto la luz a lo largo de las últimas décadas nos están dejando cambios sustanciales en multitud de ámbitos.

Sin embargo, para trabajar en este campo de la ciencia es necesario avanzar con la misma rapidez y saber adaptarse a nuevos entornos de trabajo más dinámicos, que se encuentran en una continua evolución y crecimiento.

En el desarrollo del presente trabajo final de grado, el aprendizaje ha sido un constante diario, enriquecedor y a la vez necesario para lograr cumplir todos los objetivos que se han ido planteando en el transcurso de su desarrollo.

En este primer capítulo se van a recoger de forma resumida los principales objetivos a resolver y la estructura que sigue el presente documento.

1.1. OBJETIVOS DEL TRABAJO

El principal objetivo del presente trabajo final de grado es el desarrollo de una aplicación basada en cámaras 3D para la generación de movimientos en un robot humanoide.

Con este trabajo se pretende crear una aplicación capaz de captar los movimientos que realiza una persona frente a una cámara 3D, en este caso un sensor denominado Kinect, y que acto seguido un robot humanoide sea capaz de reproducir los movimientos que previamente ha realizado el usuario.

Para cumplir este objetivo final primero se han de cumplir otros que se presentan en el camino hasta llegar a la meta marcada.

En primer lugar, es importante destacar que el entorno en el que se desarrolla todo el proyecto no es el más usual o con el que hemos trabajado a lo largo de estos años cursados en el grado. Para poder desarrollar la aplicación es muy importante aprender a manejarse en un entorno propio de la robótica en el cual el sistema operativo sea Linux. En este software es importante tener un dominio básico del terminal, una ventana de comandos a partir de la cual se descargarán y ejecutarán todos los programas y drivers necesarios así como la propia aplicación creada.

Otro objetivo previo a la creación de la aplicación y cuyo papel es fundamental en el desarrollo de este trabajo, es la comprensión del modo de funcionamiento de ROS. Este, es un entorno de

programación que presenta una estructura basada en una filosofía de código abierto, en el que cualquier usuario puede colaborar y aportar sus conocimientos e investigaciones. Por ello a lo largo del desarrollo de este trabajo se accederá a mucha información a priori desconocida, pero que con el paso de los días acabará por ser más familiar.

Dentro de este entorno de programación cuyo nombre es ROS, se sostiene una red de "nodos" y "topics" a los cuales se les dedicará una explicación en el desarrollo teórico, pues es importante su comprensión ya que la aplicación a desarrollar constituirá un nodo al cual se suscribirán tanto el robot humanoide como la cámara 3D.

Para poder elaborar el nodo y cumplir el objetivo del trabajo, a lo largo de este proyecto se realizarán diversos tutoriales para conocer el entorno, la interfaz en la que se trabaja y aprender conceptos nuevos de C++, el lenguaje en el que se desarrolla el código.

Finalmente, otro paso a realizar será aprender a trabajar con cámaras 3D, en este caso, el sensor Kinect conocido por su uso en las consolas Xbox 360, pero en este caso aplicado a ROS. Con este dispositivo, se captarán los movimientos que realiza el usuario situado frente a él, y para ello antes tendrá que estar correctamente instalado y calibrado.

Una vez se hayan cumplido y solucionado todos los aspectos que vayan presentándose a los largo de este trabajo y que de forma resumida se acaban de expresar, se llegará al objetivo final, en el cual se consiga controlar el robot humanoide según los movimientos que realice una persona.

1.2. ESTRUCTURA DEL DOCUMENTO

La estructura del documento, es la propia de un trabajo final de grado, con la presentación de un tema y su posterior desarrollo tras introducir brevemente los conceptos teóricos que sean de utilidad para el desarrollo del proyecto.

Por sus características y temática cabe destacar que no contiene planos ya que no son necesarios para este tipo de trabajo y además por la extensión de tiempo limitada para la realización, no incluirá pliego de condiciones, así, finalmente el trabajo constará de los dos documentos básicos y principales, la memoria y el presupuesto.

De una forma más concreta, los puntos que se seguirán para la redacción son los que a continuación se muestran:

Memoria descriptiva:

1. Introducción: se presenta el tema, los objetivos del trabajo y la estructura que éste va a seguir.

2. Desarrollo teórico: se estudia el tema a tratar, en este caso la robótica y se particulariza para una tipología concreta, los robots humanoides. Por otra parte se desarrollan conceptos referentes al entorno de programación ROS y las distintas herramientas que se utilizarán o que en cualquier caso podría utilizarse para aplicaciones similares. Finalmente, se trata la temática de las cámaras 3D y su funcionamiento.

3. Desarrollo práctico: en este apartado se explican detalladamente los pasos a seguir para cumplir el objetivo marcado al inicio del proyecto. Tratando punto por punto los problemas que han ido apareciendo y cuál ha sido la solución óptima para resolverlos.

4. Conclusiones y trabajos futuros: tras la realización del trabajo en este punto se analizan los pasos seguidos y los objetivos que se han cumplido. También se plantean posibles aplicaciones de aquello que se ha realizado en vistas a una posible proyección en el futuro.

5. Bibliografía: en esta sección de la memoria se encuentran los puntos de información a los que se ha accedido para documentar el capítulo que hace referencia al desarrollo técnico y la páginas a las que se ha accedido para insertar ciertas imágenes.

Presupuesto:

1. Necesidad del presupuesto: todo proyecto ingenieril debe considerar un estudio económico, y es por ello que en este punto se expresa la necesidad de elaborar un presupuesto.

2. Estudio económico: en este apartado se hace un desglose de todo aquello que se ha utilizado para el desarrollo del presente proyecto, tanto si supone un coste económico como si no. Para ello se tiene en cuenta el coste del personal, el material al que se le puede hacer un inventario y el material fungible.

3. Resumen del presupuesto: En este punto se recopilan todos los datos económicos anteriores y se agrupan para realizar la suma del total.

CAPÍTULO 2. DESARROLLO TEÓRICO

Debido al amplio campo tecnológico en el que se va a trabajar y las diferentes ramas que se van a necesitar para el desarrollo del siguiente trabajo, resulta necesario realizar un desarrollo teórico de los distintos conceptos que de seguido van a ir apareciendo a lo largo de la memoria. Además es importante describir el espacio de trabajo (software, programas y herramientas) para entender por qué se hace uso de ellas y qué papel desempeñan en el desarrollo del proyecto.

En primer lugar, se va a identificar y a diferenciar del resto, el tipo de robot a utilizar, acto seguido se explicará el sistema operativo utilizado, su interfaz, y las distintas herramientas empleadas para el desarrollo del proyecto y en último lugar se abordará la temática de las cámaras 3D.

2.1. ROBOTS HUMANOIDES

Actualmente el mundo de la robótica se presenta en nuestras vidas en múltiples y variadas situaciones, desde los grandes robots en forma de brazo utilizados en las industrias hasta los novedosos y pequeños robots limpiadores, cada vez más comunes en los hogares. Existen diversas tipologías en lo que se refiere al campo de la robótica, pero de todas ellas se extrae un principio común: todos los robots están diseñados para sustituir o ayudar al ser humano en trabajos sistemáticos, repetitivos e incluso complejos que se puedan automatizar, para así poder aliviar a las personas de algunas tareas físicas. Debido al extenso campo que abarcan los diferentes tipos de robots, a continuación se va a definir qué es un robot para poder partir de un concepto general y así posteriormente entender conceptos más particulares como es el caso de un robot humanoide.

2.1.1. ¿Qué es un Robot Humanoide?

En primer lugar se define un robot como una máquina automática programable diseñada para moverse, manipular objetos y realizar tareas, en especial aquellas que son complicadas, repetitivas, pesadas o peligrosas para las personas al mismo tiempo que interacciona con su entorno [1]. En función de la situación en la que se encuentren dichas máquinas, responderán con una acción u otra, ya que están dotadas de diversos sensores que recopilan información del entorno en el que se desenvuelven. Debido a todas estas cualidades, puede parecer que los robots tengan la capacidad de pensar o razonar ante las distintas situaciones que se les plantean, sin embargo estas máquinas se limitan a seguir los algoritmos programados previamente en un ordenador.

La robótica une múltiples disciplinas como la informática, la electrónica, la mecánica, la inteligencia artificial, la ingeniería de control y muchas otras que se suman en función del objetivo a desempeñar por el robot.

Un robot humanoide es un robot diseñado para imitar tanto el aspecto físico como los movimientos de los seres humanos [2]. Estos robots pueden estar diseñados con diferentes fines, tales como la investigación en el campo de la medicina, su uso en aplicaciones de elevado riesgo químico o nuclear o como ayuda en diferentes labores cuotidianas entre otros usos. Estos robots pueden presentar diferentes configuraciones, pero por lo general constan de dos piernas, un torso, dos brazos y una cabeza. Mantienen esta apariencia física para poder desempeñar tareas en entornos en los que se desenvuelven los seres humanos y para los cuales se necesita la movilidad de los brazos y las piernas.

Este tipo de robots al igual que el resto está en constante evolución y crecimiento y ese es el motivo por el cual cada vez aparecen más robots en la sociedad actual desempeñando multitud de diversas y variadas funciones.

2.1.2. Tipologías de robots

Según el campo en el que desarrollan sus funciones, los robots se pueden clasificar en distintas tipologías, ya que su uso maraca su diseño así como las acciones para las que tienen que estar programados y automatizados.

Entre las distintas tipologías podemos destacar:

-Robots médicos

En este grupo de robots se encuentran fundamentalmente las prótesis de órganos y extremidades que se implantan a los disminuidos físicos para tratar de igualar los movimientos y funciones que éstos sustituyen.

-Robots manipuladores

Los robots manipuladores son, principalmente, brazos articulados. Para concretar más, un manipulador industrial convencional es una cadena cinemática abierta formada por eslabones conectados mediante articulaciones o pares cinemáticos.[3] En el último enlace se coloca una pinza o dispositivo especial para realizar las distintas operaciones, cabe destacar, que este tipo de robots se encuentran fijos en su base.

Imagen 1. Brazo robótico [4]

- Robots móviles

En esta categoría se agrupan todos aquellos robots que son capaces de desplazarse ya sea mediante ruedas, con una configuración de oruga o con patas. Este tipo de robots son muy utilizados en ambientes industriales para transportar mercancías, y también en ambientes de difícil acceso, como pueden ser las exploraciones espaciales o submarinas.

Los robots teleoperadores, pertenecen a este grupo, y estos se caracterizan por ser controlados de forma remota por un operador humano. Trabajan de esta forma para poder actuar en situaciones extremas como pueden ser la manipulación de residuos tóxicos o la desactivación de bombas, entre otras.

Imagen 2. Robot teleoperado para desactivar explosivos [5]

Los zoomórficos se caracterizan por imitar la locomoción de los animales. En base a este principio existen de dos tipos: caminadores y no caminadores.

Imagen 3. Robot zoomórfico [6]

Los humanoides son robots que tratan de imitar el aspecto físico y las acciones que realizan los seres humanos, para poder integrase en los ambientes en los que ellos se desenvuelven. Posiblemente existen otras configuraciones más rápidas, como las que están compuestas por ruedas y otras más estables, como las que están formadas por cuatro apoyos. Sin embargo la configuración bípeda es el sistema más flexible que se adapta a multitud de tareas como andar, subir escaleras, agacharse, gatear y otras muchas que necesitan de dos piernas y dos brazos. Además si lo que se pretende es integrar a los robots humanoides en la vida cuotidiana de cada persona, es razonable pensar que la aceptación será mayor si estos tiene un aspecto más alejado del de una máquina y más cercano al de una persona.

Imagen 4. Robot humanoide NAO [7]

2.1.3. Evolución histórica de los robots humanoides

La aparición de este tipo de robots es relativamente reciente ya que el primero de ellos data del siglo pasado. Entre todos los prototipos y modelos que han ido surgiendo a lo largo de los años y de muchas investigaciones, podemos destacar los siguientes:

- Wabot 1: 1973, fue creado por un grupo de ingenieros de la Universidad de Waseda de Tokio. Es el primer robot antropomórfico a escala real. Podía comunicarse con una persona, calcular distancias y direcciones, caminar lentamente y coger y transportar pequeños objetos. [8]

- Wabot 2: 1984, segunda versión del anterior, era capaz de recitar y entender algunas palabras en japones y leer y tocar una partitura de mediana dificultad.

-EO(Experimental 0): 1986, primer prototipo creado por la empresa Honda Motor, este robot bípedo necesitaba 5 segundos para completar un paso.

-E6:1991, este robot tenía un control de balance autónomo para sortear obstáculos y para subir y bajar escaleras.

-P1:1993, primer prototipo de la empresa Honda con aspecto similar al de un humano, al contar con extremidades superiores e inferiores.

-P2:1996, primer robot humanoide presentado al público, con capacidad para caminar, mover los brazos y con visión periférica.

-ASIMO:2000,(Advanced Step in Innovative Mobility), robot de la empresa Honda, caracterizado por su avanzada tecnología para caminar, su peso ligero, y la gran libertad de movimientos que posee, así como su aspecto.

-NAO:2004, robot humanoide desarrollado por la compañía francesa Aldebarán Robotics, este robot es interactivo y programable. Escucha, ve, habla y se relaciona con el medio que le rodea, es un robot capaz de interactuar de forma natural.

-Wabian 2:2006, robot japonés que mejor imita el caminar de un ser humano ya que tiene caderas flexibles que le permiten girar las piernas para suavizar la forma con la que anda.

2.1.4. Robot del proyecto: NAO

El robot humanoide que se va a utilizar para el desarrollo del proyecto es NAO. Éste es un robot humanoide de dimensiones reducidas (58cm), totalmente programable y capaz de interactuar de forma natural con todo tipo de personas. Pertenece a la empresa francesa Aldebarán Robotics, la cual ya ha realizado 5 versiones hasta llegar al modelo actual.

NAO es capaz de ver, hablar escuchar y relacionarse con el entorno en el que se encuentra, sus movimientos y acciones pueden ser complejos ya que su pequeño tamaño así como todos los componentes que lo forman le confieren todas estas cualidades. Entre los componentes que forman este robot humanoide cabe destacar los siguientes sensores: dos cámaras, nueve sensores táctiles, dos sensores de ultrasonidos, ocho de presión, cuatro micrófonos, un giróscopo y un acelerómetro. También incluye multitud de LEDs, un sintetizador de voz y dos altavoces.

NAO incluye un software gráfico de programación, Choreographe, compatible con Windows, Mac y Linux, esta interfaz gráfica permite la programación del robot sin la necesidad de adquirir

conocimientos de programación, sin embargo las posibilidades de nuevas acciones y actividades se reducen. Para usuarios con mayores conocimientos incluye un conjunto completo para desarrollo de software, en el cual se pueden usar distintos lenguajes como JAVA, C++ y Python entre otros. [9]

Entre sus especificaciones técnicas se puede destacar:

- Peso: 4,3 Kg

- Altura: 58 cm

- Autonomía: 60 minutos en uso activo y 90 minutos en uso normal

- Grados de libertad: desde 21 hasta 25

- Conectividad: Ethernet, Wi-Fi

- Procesador: Intel Atom 1,6 GHz

-Construido en el sistema operativo NAOqi 2.0, basado en Linux

-Sistemas operativos compatibles: Windows, Linux y Mac OS

-Lenguaje de programación: C++, Python, MATLAB, Java y Urbi entre otros

-Visión: dos cámaras 1289 × 960 HD

2.2. ENTONOS DE PROGRAMACIÓN

Como se acaba de exponer en el apartado anterior NAO, es capaz de trabajar en distintos entornos. Este robot humanoide está construido en el sistema operativo NAOqi, un entorno multiplataforma que puede desarrollarse en ordenadores que contengan Windows, Linux o MAC. Además los lenguajes de programación que acepta son C++, Python, Urbi y Java.

La metodología para trabajar con NAO depende del entorno que se emplee ya que existen algunos más básicos con paquetes o bloques ya definidos y otros sin embargo que necesitan unos conocimientos más amplios y avanzados para su uso, pero que por otra parte permiten una mayor libertad de investigación en el alcance y las posibilidades a explotar con el robot.

2.2.1. Choregraphe

Choregraphe es el software de programación creado de forma específica por la empresa Aldebarán Robotics para sus robots NAO. Este software permite a los usuarios crear y editar movimientos y comportamientos de una forma sencilla, sin necesidad de emplear grandes conocimientos de programación, para aquellos que se inician en el mundo de la robótica. Pero por otra parte, este

sistema operativo es capaz de satisfacer las necesidades de aquellos usuarios más experimentados que buscan explorar nuevos caminos.

Choregraphe cuenta con una intuitiva interfaz gráfica, una biblioteca de comportamientos y funciones de programación avanzadas. Los comportamientos de NAO se pueden crear utilizando la biblioteca de cajas de comportamiento, tan solo con arrastrar o copiar la caja correspondiente o codificando comportamientos nuevos y guardándolos en la biblioteca personal.[10]

Las cajas de comportamiento ya modeladas son fáciles de configurar y con el editor de curvas o mediante el lenguaje Python, el usuario puede editar los movimientos.[10]

La programación del robot puede realizarse por bloques o por código y esta puede estar basada en eventos, de forma secuencial o en paralelo, ya que el sistema cuenta con una línea de tiempo.

Sin embargo, pese a ser un sistema muy intuitivo, un inconveniente que presenta este software es la dificultad al añadir otras funciones mediante código externo. Por este motivo no se utiliza, ya que dificultaría la conexión con el sensor Kinect.

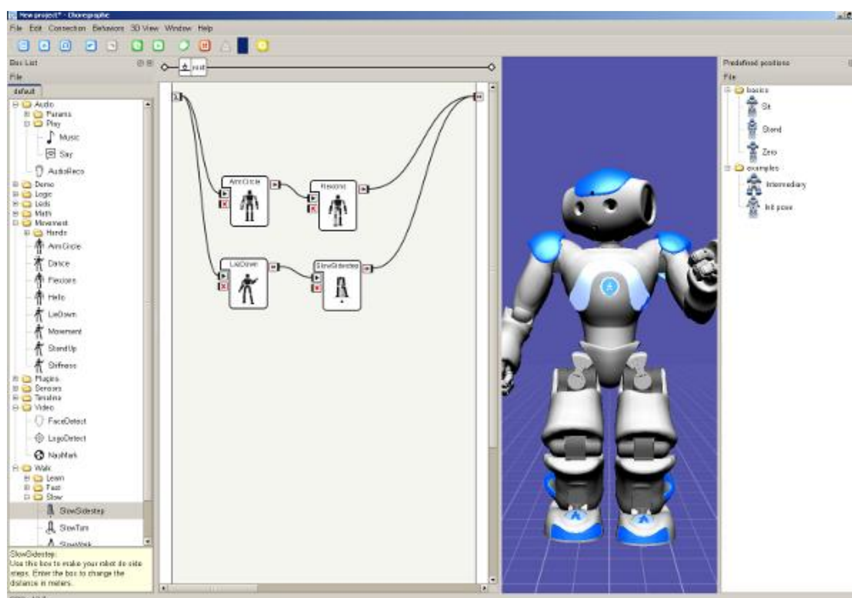


Imagen 5. Interfaz gráfica Choregraphe [11]

2.2.2. ROS

Las siglas ROS responden al nombre Robot Operating System y hacen referencia a un framework, es decir a una estructura tecnológica y conceptual, para el desarrollo de un software libre y flexible que provee la funcionalidad de un sistema operativo para el manejo, estudio e investigación de los robots.

ROS surgió en el año 2007 bajo el nombre de "switchyard", este sistema operativo fue creado por el Laboratorio de Inteligencia Artificial de Stanford para ayudar al proyecto STAIR. Sin embargo,

actualmente es una plataforma de código abierto que está en manos del instituto de investigación robótico Willow Garage, bajo la licencia open source.

Según la propia página oficial [12], ROS es un sistema operativo de código abierto que proporciona los servicios propios de los sistemas operativos y además incluye abstracción de hardware, control de dispositivos de bajo nivel, implementación de la funcionalidad de uso común, paso de mensajes entre procesos y gestión de paquetes. Además proporciona herramientas y bibliotecas para la obtención, la construcción, la escritura y la ejecución de código en diversos equipos. [12]

Esta plataforma de software de robótica se diferencia del resto no por ser la que presenta mejores capacidades, ya que el objetivo principal tras su creación fue el de fomentar la cooperación y apoyar la reutilización de código ya investigado y desarrollado en el mundo de la robótica. El fin con el que se siguió esta trayectoria es debido a que actualmente desarrollar programas completos para aplicaciones robóticas puede llegar a convertirse en una tarea muy costosa, ya que los problemas a resolver no siempre son fáciles. Por todo esto, se ha decidido plantear una plataforma con código abierto en el que personas especializadas de diversos sectores y empresas del mundo puedan colaborar para que cada investigación se centre en aquello que quiera resolver y no ocupe sus esfuerzos en resolver problemas a los que otros equipos ya han encontrado respuesta.

Por lo que respecta a los sistemas operativos a día de hoy, ROS solo se ejecuta en plataformas basadas en UNIX. El software para ROS se prueba principalmente en sistemas Ubuntu y Mac OS X, aunque también se ha desarrollado para otras plataformas de Linux como Fedora, Arch Linux y Gentoo entre otras. Pese a existir una posibilidad para Microsoft Windows, ésta no se ha explorado todavía completamente.

El lenguaje de programación que se puede utilizar para implementar código es amplio, ya que puede usarse cualquier lenguaje de programación moderno como pueda ser Python y C++.

Para tratar de entender mejor el modo de funcionamiento de ROS, a continuación se van a definir los siguientes conceptos, dividiéndolos en tres bloques, según hagan referencia al nivel del sistema de archivos (Filesystem Level), al nivel de computación gráfica (Computation Graph level) o al nivel comunitario (Communitary level): [13]

Sistema de archivos

Los conceptos que se definen en el nivel del sistema de archivos se refieren a los recursos que se encuentran en el disco duro.

-Paquetes (packages): Los paquetes son la unidad principal en la que se basa ROS. Cada paquete puede contener aplicaciones(nodos), librerías, bases de datos, archivos de configuración, o cualquier información que se haya organizado de manera útil. Los paquetes son el elemento básico bajo el que se construye la filosofía de ROS ya que el hecho de agrupar las aplicaciones o programas en paquetes hace que cada vez que se requiera una función, tan solo con descargar el paquete que se haya elaborado para tal fin se obtenga el resultado que se desee. Esta forma de operar hace que sea factible la creación de una comunidad para desarrollar cada vez más esta plataforma.

- Metapaquetes (metapackages): Los metapaquetes son paquetes especializados que solo sirven para representar a un grupo de otros paquetes relacionados entre sí.

- Repositorios (repositories): Los repositorios son una colección de paquetes que comparten un sistema VCS común. Los paquetes que forman un repositorio tiene la misma versión y pueden ser liberados juntos.

Nivel de Computación Gráfica

La computación gráfica es la red de procesos de ROS en la que se están procesando datos en conjunto.

-Nodos: Los nodos son procesos que el sistema operativo ejecuta y que realizan el cálculo para llevar a cabo distintas actividades. El modo de funcionamiento de ROS es modular y es por ello que un sistema de control de un robot comprende normalmente muchos nodos, por ejemplo, un nodo controla los motores de las ruedas, otro nodo realiza la planificación de la trayectoria, otro nodo realiza la localización y así sucesivamente.

-Maestro: El maestro es un nodo que proporciona el registro de nombres y de consulta para que el resto de los nodos sean capaces de intercambiar mensajes y utilizar servicios.

- Servidor de parámetros: permite que los datos sean almacenado mediante una clave en una ubicación central, actualmente forma parte del maestro.

-Mensajes: Los mensajes son la forma que tienen los nodos para comunicarse entre sí. Un mensaje es una estructura de datos que pueden ser de diversos tipos, ya sea booleanos, enteros o de punto flotante entre otros.

-Temas (topics): Los nodos se hacen servir de los "topics" para enviar mensajes, ya que estos son un sistema de transporte con publicación y suscripción. Es decir, un nodo envía un mensaje determinado mediante su publicación en el correspondiente "topic". Éste es el nombre que se utiliza para identificar el contenido del mensaje. Gracias a esta peculiar forma de trabajar de ROS, un nodo que está interesando en un determinado tipo de datos se suscribirá al "topic" correspondiente; de esta forma pueden existir multitud de editores y suscriptores para un solo tema, y a la vez, un solo nodo puede publicar y suscribirse a varios temas. Esta idea de desacoplar la producción de información de su consumo es un hecho que diferencia a ROS del resto de sistemas operativos, facilitando así las posibilidades de compartir información.

- Servicios: Pese a que el modelo mencionado anteriormente en la definición de los "topics" es muy flexible y a la vez potente, hay otro sistema de petición/respuesta que se realiza a través de los servicios. En este modelo hay dos estructuras, una para la solicitud y otra para la respuesta; mientras un nodo ofrece un servicio, un nodo cliente envía la solicitud y espera la respuesta para poder hacer uso del servicio.

- Bolsas: Las bolsas son el formato que tiene ROS para almacenar y reproducir los datos de mensaje.

Nivel comunitario

En este nivel se encuentran los recursos que permiten la comunicación en ROS.

- Distribuciones: Las distribuciones son las distintas versiones disponibles de ROS que se pueden instalar.

- ROS Wiki: es una comunidad en la que cualquier persona puede registrarse para tener una cuenta y contribuir en la ampliación de la documentación o también se pueden visitar los tutoriales y documentos explicativos acerca del uso y funcionamiento de ROS.

2.2.2.1. MoveIt!

Después de todos los conceptos definidos cabe destacar que ROS es capaz de integrar otras librerías como pueden ser Gazebo o MoveIt y es por ello que en el presente trabajo el entorno de programación que se va a utilizar será ROS junto con MoveIt, dado las altas prestaciones que presentan y el amplio campo en la investigación que ofrecen.

MoveIt es un software de código abierto para ROS que está cobrando fuerza ya que a día de hoy más de 65 robots incluyendo los que desarrolla Robonik utilizan este software. La filosofía que persigue es la misma que ROS reutilizando el código. [14]

Iniciarse en MoveIt es sencillo pues solo es necesario que el usuario proporcione la URDF (Unifield Robot Description Format) y defina algunos campos como la cadena cinemática deseada para el control del robot; el resto de la información se define automáticamente gracias al asistente que proporciona el paquete.

Cuando el asistente se ha competado, es fácil parametrizar el nodo "move_group", éste es el núcleo que proporciona la funcionalidad del software y permite llevar a cabo tareas de percepción 3D, cálculos cinemáticos, control y navegación de forma sencilla, control de la colisión y planificación de trayectorias complejas.

Para acceder a las acciones y servicios que proporciona el move_group hay tres opciones:

-En C++ utilizando el paquete move_group que proporciona la interfaz.

-En Python utilizando el paquete moveit_commander.

-A través de la interfaz gráfica de usuario, utilizando el plugin de Planificación de movimiento a Rviz.

Interfaz del robot

La comunicación entre el robot y el move_group se realiza a través de temas y acciones de ROS. La comunicación se lleva a cabo para obtener información sobre el estado actual del robot, para obtener datos de los sensores y para comunicarse con los controladores del robot.

-Información de las articulaciones (Joint State Information): el move_group recibe la información que le proporciona el "topic" joint_states acerca del estado actual del robot, es decir de cuál es la posición de cada articulación. Además el move_group es capaz de recibir la información de varios editores sobre este "topic".

- Controlador de la interfaz: El move_group se comunica con los controladores del robot utilizando la interfaz.

-Planificación de la escena (Planning Scene): El move_group utiliza esta función para representar el entorno y el estado actual del robot. También se pueden incluir objetos que son llevados por el robot y que se considera que están firmemente sujetos a él.

-Capacidades extensibles: El move_group está estructurado para que su extensión sea fácil; las capacidades individuales, la cinemática o la planificación de movimientos están implementados como "plugins" por separado con una base común. Todos ellos son configurables mediante ROS a través de la biblioteca de ROS o mediante un conjunto de parámetros. Sin embargo, la mayoría ya vienen configurados automáticamente en los archivos que se generan al iniciar el asistente de configuración de MoveIt.

Planificación del movimiento (Motion Planning)

-El "plugin" de la planificación del movimiento (Motion Planning Plugin) trabaja a través de una interfaz de complementos con los planificadores del movimiento Gracias a esto, MoveIt puede utilizar y comunicarse con los diferentes planificadores del movimiento a partir de las bibliotecas.

- La solicitud del Plan de Movimiento(Motion Plan Request): En ésta acción se especifica con precisión lo que deseamos que el planificador del movimiento realice. Normalmente se utiliza para especificar el movimiento que ha de seguir algún miembro (brazos y piernas entre otros) o un efector final del robot (manos, pinzas u otra configuración que se tenga en los extremos). La colisiones se pueden controlar incluyendo la comprobación de auto-colisiones. Además los objetos que se adjuntan al robot se incluyen en la planificación del movimiento del robot. Finalmente cabe destacar que se pueden incluir restricciones cinemáticas como por ejemplo: de posición, de orientación y de visibilidad.

-El resultado del movimiento (Motion Plan Result): El nodo move_group generará la trayectoria deseada como respuesta a la solicitud del plan de movimiento enviado. Con esta acción se moverá el grupo de articulaciones seleccionado a la posición final definida. Las aceleraciones y velocidades empleadas en la generación de la trayectoria se pueden especificar, atendiendo siempre a los máximos establecidos.

-OMPL (Open Motion Planning Library): Es una biblioteca de planificación del movimiento de código abierto que principalmente planifica el movimiento al azar.

Planificación de la escena

Esta acción se utiliza para representar el entorno que rodea al robot y también el propio estado del mismo. Por ello, la información que recibe proviene del estado (joint_states topic), del sensor y de la geometría que introduce el usuario (planning_scene topic).

-Monitor de geometría (Worls Geometry World): Éste construye la geometría del entorno con la ayuda de la información que recibe de los sensores del robot y de la entrada del usuario.

-Percepción 3D: La percepción 3D se dirige a través del monitor de mapa de ocupación, éste utiliza una arquitectura basada en "plugins" para manejar las diversas entradas del sensor. En particular MoveIt es capaz de manejar dos tipos de entradas: nubes de puntos e imágenes con profundidad.

2.3. CÁMARAS 3D

Las cámaras 3D, también llamadas cámaras estereoscópicas tratan de recrear la visión en 3D humana. Para ello constan de dos objetivos o dos cámaras separadas una cierta distancia que captan la imagen en el mismo instante, con el fin de imitar la visión binocular humana. Ésta funciona de la siguiente manera: cada ojo produce una imagen y después ambas son mezcladas en el cerebro para crear la imagen en tres dimensiones.

2.3.1. Kinect

El sensor de detección de movimientos que va a utilizarse es el Kinect, un controlador de juegos creado por Alex Kipman y desarrollado por la compañía Microsft. Este dispositivo fue creado para la videoconsola Xbox 360 con el fin de competir con el resto de consolas con sensor de movimiento del mercado, tales como Nintendo WII y Playstation 3.

Kinect permite a los usuarios de los videojuegos interactuar y controlar la consola sin necesitar un contacto físico con un controlador tradicional. Esto es posible gracias a una interfaz natural de usuario que reconoce los gestos del jugador , su voz, sus movimientos y los objetos que se muestran alrededor y que pertenecen al campo visual del sensor. [15]

En un principio se diseñó para renovar el mundo de las videoconsolas pero con el tiempo amplió su campo de utilización, pues la información que se podía llegar a obtener con el sensor Kinect podía resultar muy útil. Para ello, se recurrió a métodos de ingeniería inversa, desarrollando un controlador para Linux que permite utilizar el sensor en ordenadores para aplicaciones como el reconocimiento de objetos.

Especificaciones técnicas más relevantes

-Campo de visión

·Campo de visión vertical: 43 grados

·Campo de visión horizontal: 57 grados

·Rango de profundidad del sensor: 1,2 - 3,5 metros

·Rango de inclinación física: ± 27 grados

-Sistema de seguimiento

· Rastrea hasta 6 personas

·Rastrea 20 articulaciones por persona

·Capacidad para mapear

-Flujo de datos

·Color: 640×480 32 bit de color

·Sensor de profundidad: 320×240 a 16 bits de profundad

·Audio de 16 bit a 16 kHz

<u>Componentes principales</u>

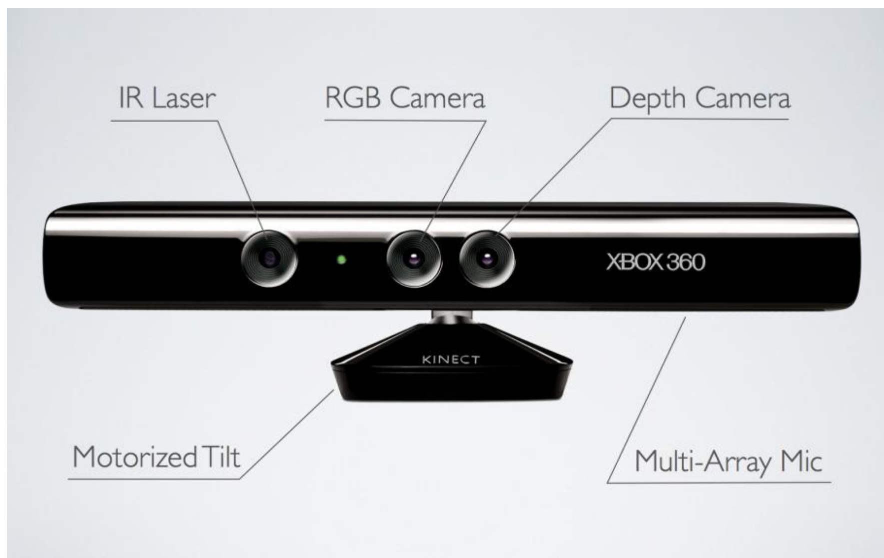


Imagen 6. Componentes Kinect [16]

Los componentes principales que forman el controlador son los que a continuación se enumeran y explican:

-Cámara RGB

Las siglas RGB hacen referencia a las palabras Red, Green y Blue; los colores primarios de la luz.

La cámara RGB se utiliza para obtener imágenes de color y su funcionamiento es el mismo que presentan la mayoría de las cámaras digitales. Un haz de luz atraviesa una lente que dirige la luz a un filtro diseñado para separarla en los colores anteriormente nombrados, para después proyectarlos sobre un sensor fotosensible. El sensor genera una señal eléctrica en función de la intensidad de la luz recibida, ésta señal pasa por un convertidor analógico-digital para finalmente poder analizarla y almacenarla.

-Sensor de profundidad

El sensor de profundidad que se encuentra en el Kinect permite ver el entorno en 3D en cualquier condición de luz ambiental y también permite ajustar el rango de detección de la profundidad del sensor.

Su funcionamiento se basa en dos elementos: un proyector de luz infrarroja combinado con un sensor CMOS monocromo. Con el proyector se obtiene una proyección de una matriz de rayos de luz infrarroja sobre el entorno en el que se encuentra el Kinect; estos rayos rebotan en los objetos y son capturados por el sensor infrarrojo.

La resolución del sensor de profundidad es de 640×480 píxeles. Para codificar los píxeles se utilizan once bits, es decir contiene 2048 niveles de profundidad posibles.

-Base monitorizada

El sensor de Kinect es una barra horizontal conectada a una pequeña base circular monitorizada con un eje de articulación de rótula. Esta base permite inclinar el sensor mediante un impulso mecánico y de manera automática hacia arriba o hacia abajo según la posición y movimientos de la persona que este visualizando.

-Micrófonos

El controlador contiene en el borde frontal inferior un conjunto de micrófonos que permiten reconocer la voz del usuario y desacoplarla del ruido ambiente, permitiendo así participar en el chat de la videoconsola sin necesitar auriculares.

CAPÍTULO 3. DESARROLLO PRÁCTICO

3.1. INTRODUCCIÓN Y PLANTEAMIENTO

Una vez se ha desarrollado la parte teórica del trabajo en el capítulo anterior, ya se puede iniciar la explicación del desarrollo práctico del trabajo.

En el presente capítulo se van a resolver paso a paso los objetivos planteados al inicio del proyecto para lograr desarrollar una aplicación basada en cámaras 3D para generar movimientos en un robot humanoide.

Antes de dar inicio al desarrollo de la aplicación es necesario decidir el entorno en el cual se trabajará de entre los propuestos en el capítulo uno referente a la teoría. Finalmente, se decide trabajar con el entorno de programación ROS por las amplias posibilidades que ofrece en cuanto a la libertad de programación y uso de herramientas . De entre ellas, destaca la implementación de una de sus librerías, MoveIt, en las que se encuentra el modelado de distintos robots, pr2 y NAO, que van a ser de utilidad para aprender y realizar pruebas en la interfaz gráfica RViz.

Por otra parte, en este capítulo se estudiará el comportamiento del sensor Kinect para lograr realizar una conexión correcta entre la información que capta y aquella que debe recibir el robot.

De forma esquemática se van a establecer los pasos que se van a seguir en el siguiente desarrollo para conseguir el objetivo final.

- En primer lugar es necesario configurar el entono de trabajo adecuadamente, descargando todos los programas necesarios y comprobando que las versiones tomadas son las correctas para el funcionamiento en conjunto.
- En segundo lugar se debe instalar con éxito el sensor de trabajo, descargando para ello los drivers adecuados que recojan la información necesaria del movimiento de las articulaciones del usuario.
- Una vez realizados estos dos pasos, queda investigar el funcionamiento de este entorno de trabajo. Saber que nodos y topics son los que están activos cuando la Kinect esté en funcionamiento. También tratar de buscar una transformación del espacio de coordenadas del sensor al espacio de coordenadas del robot para que la información intercambiada sea correcta. Encontrar el modo correcto de enviar la información de las posiciones a las articulaciones del robot.
- Y, finalmente, recoger todo lo aprendido en un nodo, en el cual se desarrolle el trabajo expresado y además presente la funcionalidad de recoger movimientos básicos para después poder automatizarlos.

3.2. MATERIAL Y EQUIPO

Una parte fundamental y necesaria en el desarrollo del presente proyecto ha sido un ordenador capaz de soportar los entornos de programación ROS y MoveIt que se necesitan como canal de interconexión entre el robot humanoide y las órdenes que realiza el usuario y son captadas mediante el sensor Kinect.

En un primer lugar se partió con un ordenador portátil cuyo sistema operativo de base es Windows 7, por ello, para poder trabajar con ROS se instaló una máquina virtual VMware con el software Ubuntu 14.04 de 64 bits.

En las sucesivas instalaciones para configurar el entorno de trabajo no apareció ningún problema que no tuviera solución, sin embargo, una vez instalados todos los componentes necesarios para trabajar con la cámara 3D, ésta no llegó a iniciarse correctamente pues si que era detectada por el ordenador pero el hecho de trabajar con una máquina virtual le impedía trabajar correctamente.

Por todo lo comentado, se optó finalmente por un ordenador portátil con sistema operativo de base Linux, cuya versión es Ubuntu 14.04 y con la instalación de ROS Indigo.

Para la captación de movimientos, se ha utilizado un controlador de juegos llamado Kinect, creado para la videoconsola Xbox 360. La elección de este dispositivo ha sido debido a las características que presenta, ya que se adapta a la perfección al tipo de trabajo que se quiere desarrollar. Además es compatible con el entorno de trabajo ROS, y existen los drivers necesarios para su correcta instalación y desarrollo. Las principales características de la Kinect han sido enumeradas y desarrolladas en el capítulo 2.

Por último, toda la información captada con el sensor y procesada en el ordenador debe enviarse correctamente al robot humanoide para que éste reproduzca los movimientos del usuario. Es por ello que el tercer elemento fundamental de este proyecto es el robot humanoide NAO, cuyas características más relevantes también se encuentran desarrolladas en el capítulo 2.

3.3. CONFIGURACIÓN INICIAL: ROS Y MOVEIT

El primer paso a realizar es la instalación de ROS, ésta se puede llevar a cabo fácilmente siguiendo las pautas que se marcan en la página web http://wiki.ros.org/es.

La versión que se ha instalado es ROS Indigo, ya que instalar la versión más actual, Jade, podría implicar problemas en la búsqueda de paquetes y drivers para la Kinect y el robot que aun no se encuentran disponibles. Por otra parte lo mismo sucede si se descarga una versión más antigua como es la Hydro, para la cual, algunos paquetes ya están obsoletos.

Una vez seguidas las instrucciones marcadas para la correcta instalación de ROS, es necesario configurar el entorno de trabajo creando un área de trabajo de ROS (ROS workspace). Este paso se hace necesario debido a la metodología empleada en ROS, ya que los comandos se ejecutan en un terminal de Linux, y si el entorno no está configurado correctamente, al escribir comandos en el terminar, éstos no se reconocerán ni se podrán ejecutar los programas compilados en el workspace.

ROS ofrece dos métodos para organizar y construir nuestro propio código, estos dos son: rosbuild y catkin. El primero se caracteriza por ser fácil de usar y simple, el segundo es más sofisticado pero proporciona mayor flexibilidad, especialmente para aquellos que pretenden integrar bases de códigos externos o quieren lanzar su software.

El primer paso a realizar es escribir en el terminal:

$ source /opt/ros/indigo/setup.bash

Este código se escribe para tener acceso a los comandos de ROS, por lo tanto se tendrá que ejecutar cada vez que se abra un terminal nuevo o si se prefiere se puede añadir esta línea al fichero .baschrc.

En este proyecto se ha optado por el entorno de trabajo catkin, ya que es la opción más recomendada, la portabilidad es mejorada con respecto a rosbuild y catkin está soportado por más paquetes. Su instalación se realiza de la siguiente forma:

$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/src

$ catkin_init_workspace

En este momento, aunque el espacio de trabajo está vacío, pues no contiene paquetes en la carpeta "src", hay un fichero CMakeLists.txt que compila el contenido del espacio de trabajo. Si ejecutamos el comando catkin_make, pese a no contener ningún paquete para compilar, se crearán dos directorios: "build" y "devel".

$ cd ~/catkin_ws/

$ catkin_make

Finalmente para comprobar que el directorio ha sido creado y configurado correctamente, escribimos:

$ source devel/setip.bash

$ echo $ROS_PACKAGE_PATH

$ /home/nuestrousuario/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks

Después de descargar y configurar correctamente el entorno de programación ROS así como el workspace, el siguiente paso es la descarga de MoveIt para tener acceso a todos los robots

prototipos que ofrece (entre ellos NAO y pr2, dos robots que se utilizarán en las interfaces gráficas más adelante) y también para ejecutar y visualizar movimientos de forma virtual. La descarga es muy sencilla, pues basta con seguir los pasos que marca la página oficial de MoveIt.

3.3.1. Uso de la interfaz RViz

3.3.1.1. Introducción

Junto con la descarga realizada anteriormente de la herramienta MoveIt, se puede ejecutar un complemento muy útil que de ahora en adelante se utilizará en múltiples ocasiones. Este complemento se llama RViz y es un visualizador de ROS que permite cargar y visualizar escenas con un robot, generar planes, visualizaros e interactuar directamente con el robot que aparece por pantalla. Además se pueden añadir objetos e interactuar con ellos. Por todo ello, es en esta interfaz gráfica el lugar en el que se probarán las trayectorias y planes que se deseen que ejecute el robot.

Además también es de gran utilidad, como se ve en el siguiente punto en el que se configura el sensor Kinect, para realizar pruebas con los complementos de dicho sensor que se relacionan con el entorno de programación ROS. Entre ellos destaca el nodo openni_tracker a partir del cual se toma la información de las articulaciones del usuario y se muestran a modo de esqueleto en la interfaz gráfica RViz. Este concepto se desarrollará más adelante.

3.3.1.1. RViz con NAO

Para lanzar la interfaz basta con realizar una sola llamada por el terminal con el nombre del robot que se desea que se configure; es decir si se quiere que aparezca el robot de trabajo NAO, la llamada debe ser la siguiente:

$ roslaunch nao_moveit_config demo.launch

Tras la ejecución de esta llamada, se carga el modelo del robot humanoide en la interfaz y en ella se pueden hacer pruebas para ver los movimientos que éste puede realizar. Seleccionando en el menú desplegable que aparece a la izquierda el grupo que queremos mover ya sea la mano, el brazo, la cabeza u otra parte del cuerpo del NAO, se acciona un controlador en el grupo seleccionado que muestra diversas flechas señalando los sentidos en los que se puede mover. El marcador de color naranja se utiliza para señalar la meta a la cual se quiere llegar ("Goal State") y el marcador de color verde hace referencia al estado inicial ("Start State").

Por otra parte en el menú de la parte inferior se puede activar la planificación del movimiento para visualizar cual sería la supuesta trayectoria que realizaría el robot, y si esta es adecuada se puede ejecutar para desplazar el grupo seleccionado a la meta fijada.

Otra opción my interesante que presenta el manejo de los robots en MoveIt es la opción "Use Collision Aware IK" que también se encuentra en el menú inferior Motion Planing, en la pestaña Context. Al activar esta opción conseguimos que el propio sistema planifique las trayectorias del

robot evitando colisionar con los objetos que se le presenten o con los miembros de su propio cuerpo.

Sin embargo, pese a todas las ventajas y opciones que aparecen en esta interfaz, su uso para controlar el movimiento del robot con el cursor del ratón es muy pobre ya que no se puede especificar con la suficiente precisión y claridad el punto exacto en el que se desea que se sitúe el grupo seleccionado. Además, no siempre se selecciona correctamente el marcador y a veces da problemas para realizar algún movimiento. Por lo tanto esta interfaz no se utilizará para realizar trayectorias mediante el uso del cursor, pero si para comprobar movimientos que hayan sido planificados a través de un código de programación en un nodo elaborado.

3.4. CONFIGURACIÓN DE LA KINECT

Para poder empezar a trabajar con el controlador Kinect es necesario realizar una configuración previa que permitirá la captación de información a través de los sensores del controlador. Con este dispositivo se pretende obtener la posición, gestos y movimientos del usuario; para posteriormente enviar dichos datos al correspondiente topic al que se conectará el robot de trabajo NAO. Con esta conexión se conseguirá la reproducción de los gestos por el propio robot virtual a tiempo real y más adelante con el robot humanoide con desfases muy cortos del orden de milisegundos.

El primer paso en la configuración de la Kinect consiste en descargar el driver OpeNI que permite el uso del controlador. En el terminal de Ubuntu escribimos el siguiente código:

```
$ sudo apt-get install lobopenni0 libopenni-dev

$ cd ~/catkin_ws/src

$ git clone https://github.com/ros-drivers/openni_launch

$ git clone https://github.com/ros-dribers/openni_camera

$ cd ..

$ catkin_make

$ catkin_make install
```

Tras la descarga al ejecutar el siguiente comando, el contorlador OpeNI debería funcionar en ROS:

```
$ roslaunch openni_laucnh openni.launch
```

Sin embargo esto no sucede así, se produce un error bastante conocido que puede solucionarse si se instalan los paquetes avin2-sensorkinect y NITE 1.5.2.23.

Además en el trabajo a desarrollar se necesita la información proporcionada por el nodo openni_tracker, el cual se instala junto con el resto de paquetes que empiezan por ros-indigo-openni- y que se han descargado al inicio.

Para poder comprobar el correcto funcionamiento del sensor Kinect escribimos en un terminal de Ubuntu:

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch openni_launch openni.launch

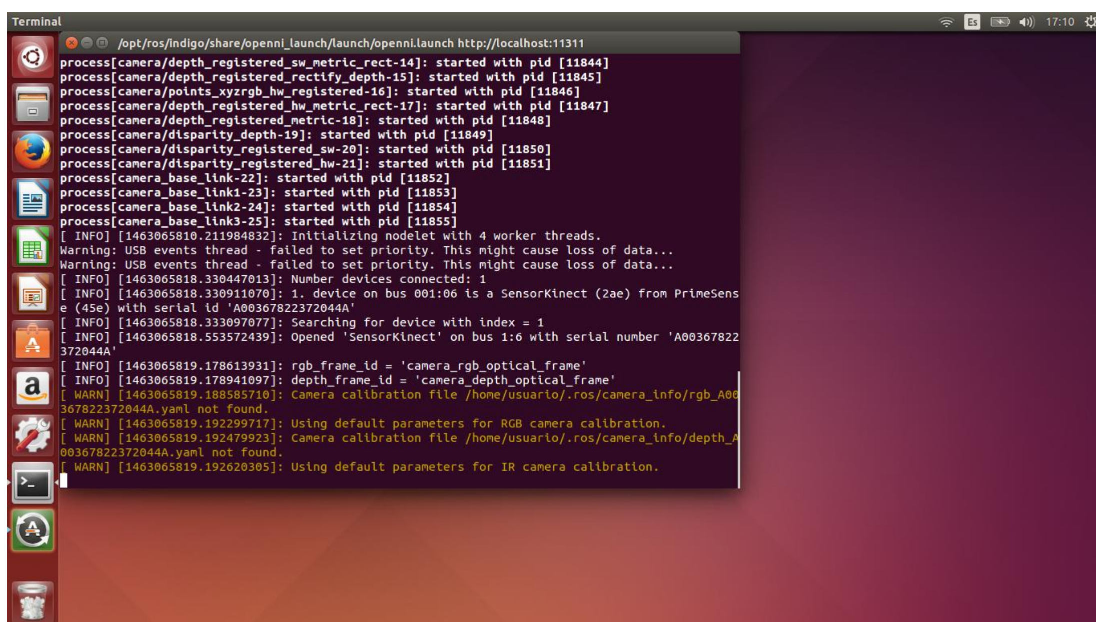


Imagen 7. Lanzamiento openni_launch

En esta imagen se puede observar el resultado que se obtiene tras la escritura de los anteriores comandos. Todo funciona correctamente y en las últimas líneas se muestra una señal de aviso para que se realice la calibración de la Kinect. Este paso se realiza a continuación al lanzar en otro terminal, mientras este continúa abierto, la siguiente instrucción:

$ cd ~/catkin_ws

$ source devel/setup.bash

$ rosrun openni_tracker openni_tracker

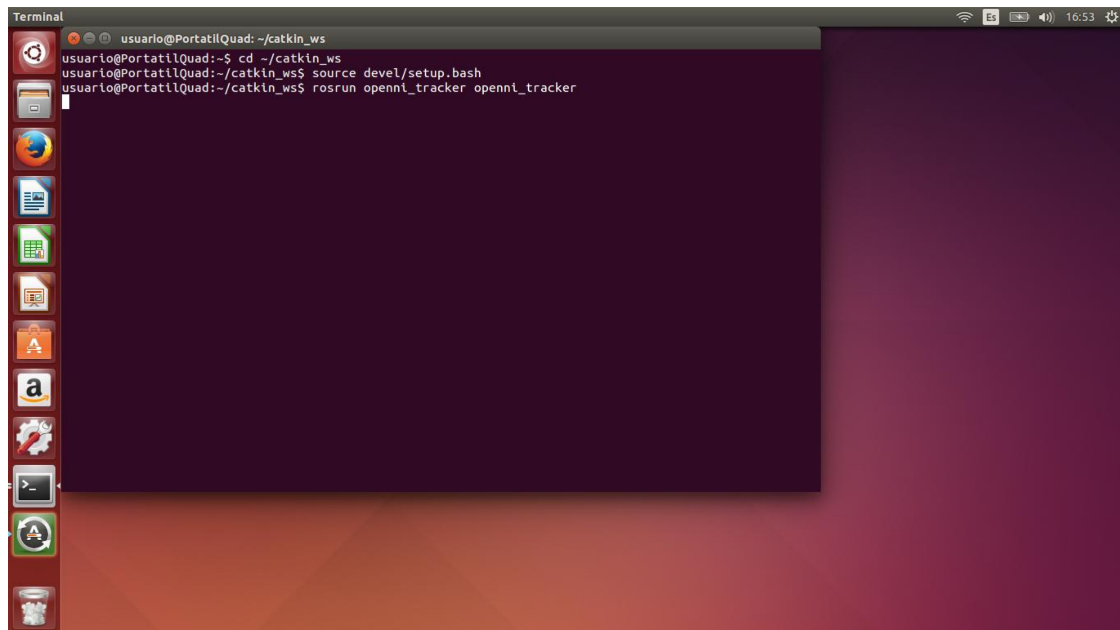


Imagen 8. Lanzamiento openni_tracker

Tras la ejecución del anterior código, el terminal se queda esperando una señal para poder detectar al usuario y calibrar el sensor.

Esto se realiza situándose delante de la Kinect a una distancia considerable, a partir de la cual el sensor pueda captar el cuerpo del usuario. Una vez colocado en este punto, la posición que se debe realizar es la que a continuación se muestra:

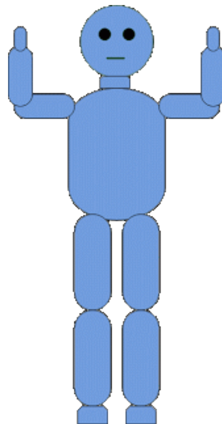


Imagen 9. Calibración de la Kinect [17]

Si se siguen los pasos especificados, en pocos segundos aparecerá en la terminal el siguiente mensaje que indica la detección del usuario y la calibración del controlador.

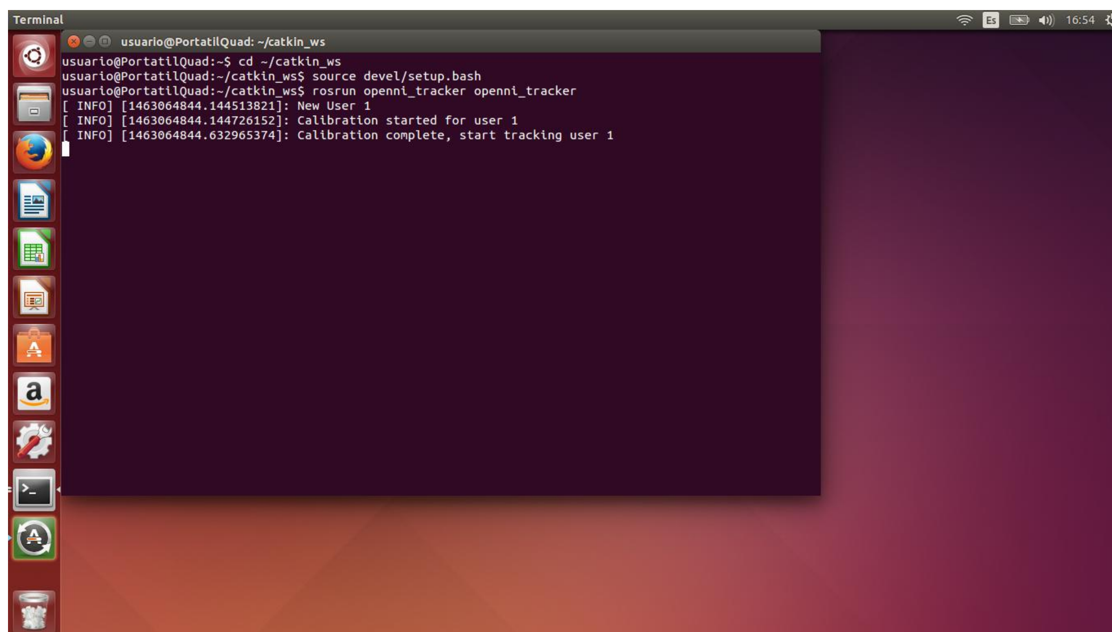


Imagen 10. Calibración correcta

Una vez tenemos abiertos estos dos terminales, el siguiente paso para visualizar en la interfaz gráfica Rviz el correspondiente esqueleto que imita los movimientos del usuario que está situado frente a la Kinect es abrir un tercer terminal y escribir:

$ rosrun rviz rviz

Con este procedimiento, la información que proporciona el "tracker" de Kinect puede visualizarse ya que el nodo openni_tracker publica la información acerca de la posición de los puntos del esqueleto del usuario que se sitúa delante del controlador.

Para obtener la información del nodo openni_tracker se necesita acceder al conjunto de trasformaciones "tf" que está publicando. En la interfaz gráfica Rviz se puede configurar el entono, en primer lugar se elige la opción openni_depth_frame del conjunto de opciones que contiene Fixed Frame del menú Global Options. Después, tras pulsar en Add se añaden los "display" Image y TF, que nos permiten ver la imagen que está captando el sensor y a la vez se muestra en la interfaz gráfica un esqueleto formado por rectas y puntos, simulando las articulaciones. Este esqueleto recrea el movimiento que el usuario está realizando frente a la Kinect, mostrando por pantalla tanto la imagen real como el esqueleto virtual. Al añadir Image, para que nos muestre la imagen que está captando la Kinect, en el desplegable que aparece a la izquierda, se debe seleccionar en "Image Topic" del submenú "Status" la opción /camera/rgb/image_color para que la configuración esté completa. Si no se realiza este último paso, no se verá la imagen real de la postura que está realizando el usuario frente al sensor.

A continuación se muestran distintas posiciones:

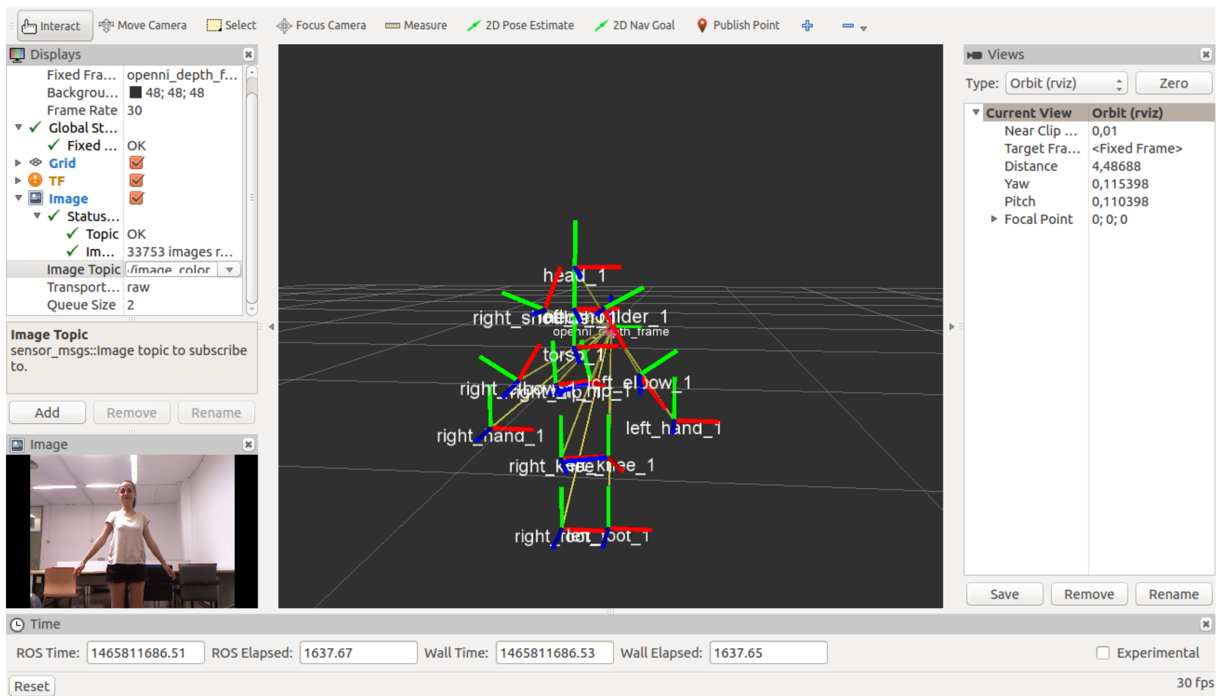


Imagen 11. Pose 1 Kinect

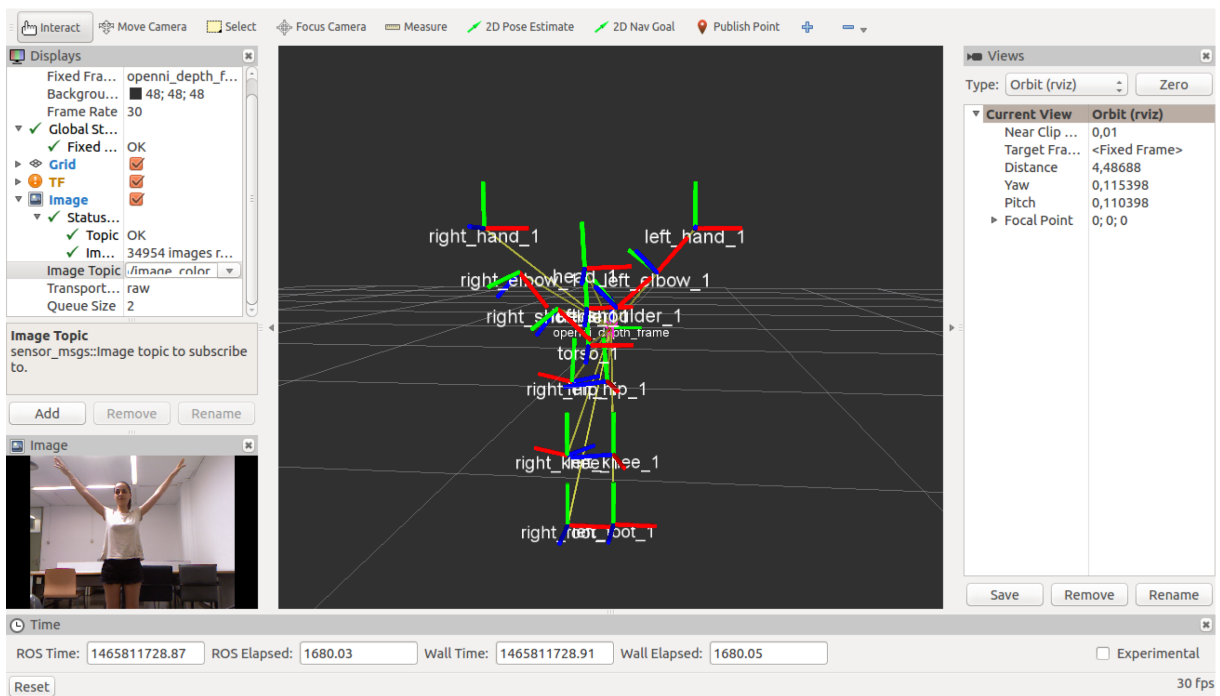


Imagen 12. Pose 2 Kinect

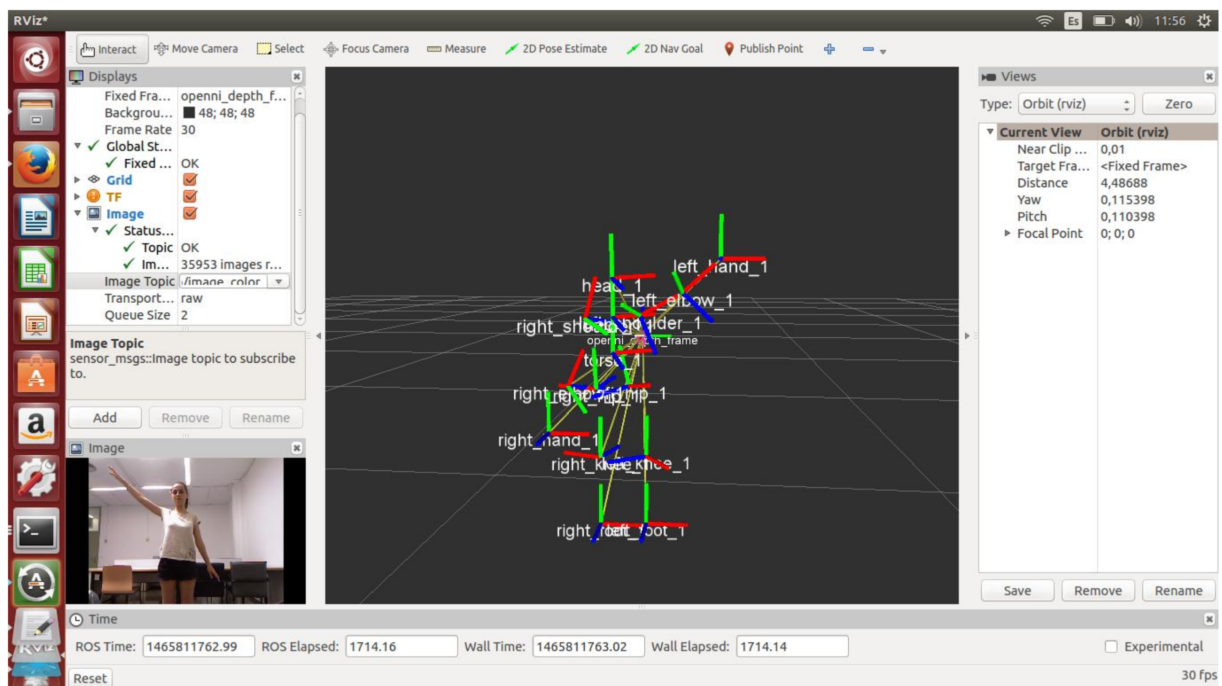


Imagen 13. Pose 3 Kinect

El nodo openni_tracker a partir del cual se están publicando los datos de los puntos del esqueleto que el sensor detecta al usuario que tiene situado en frente, funciona junto con la función "tf". Este es el sistema que utiliza ROS para realizar las transformaciones del espacio en 3D. La transformaciones que se realizan se aplican a este conjunto de articulaciones:

- Head
- Neck
- Torso
- Left_Shoulder
- Right_Shoulder
- Left_Elbow
- Right_Elbow
- Left_Hand
- Right_Hand
- Left_Hip
- Right_Hip
- Left_Knee
- Right_Knee
- Left_Foot
- Right_Foot

3.5. DESARROLLO DEL NODO

3.5.1. Introducción

La información que captamos con el sensor Kinect debe enviarse de una forma correcta al robot humanoide NAO. Es por ello que resulta necesaria la creación de un nodo, al cual se suscriban la Kinect y el robot. Es decir, el sensor de captación del movimiento actuará como un publicador de información y por otra parte el robot se suscribirá al nodo donde se encuentre esa información disponible.

Cabe destacar que la información que capta la Kinect no se puede enviar de forma directa al robot, ya que el espacio en el que trabaja cada dispositivo es distinto. El sensor tiene su propio sistema de referencia a partir del cual proporciona las coordenadas cartesianas de cada articulación que se le detecta al usuario. Por otra parte el robot tiene su propio sistema de referencia con un origen diferente y ese es el motivo por el cual en el nodo se realizará una transformación de espacios de coordenadas.

3.5.2. Control del movimiento mediante la API de C++

En primer lugar, es importante conocer el método utilizado para comunicarse con el robot virtual mediante la API de C++. Este procedimiento es necesario para entender cuál es el formato utilizado, y así posteriormente conectar la información recibida a través de la Kinect y transformada de forma adecuada al código del nodo que le envía ordenes a las articulaciones del robot.

En este caso, se ha tomado un tutorial disponible en la página oficial de MoveIt cuyo título es: "The move_group_interface (C++)". En este documento se explican distintas metodologías para planificar movimientos en un robot modelo denominado pr2. Tomando como base este robot de ejemplo, el tutorial muestra varias opciones para planificar movimientos, tanto simples como compuestos. Sin embargo, el trabajo que se aborda en este proyecto, gira en torno al robot humanoide NAO.

Para cargar en la interfaz gráfica el modelo del robot NAO y lanzar la planificación de los movimientos, basta con cambiar la llamada del robot. Los pasos a seguir son los siguientes:

Primero se accede a la carpeta catkin_ws que se encuentra en el espacio de carpetas personales. Dentro del workspace, entramos en este orden a las carpetas src, moveit_pr2, pr2_moveit_tutorials y planning; hasta llegar a visualizar las siguientes carpetas:

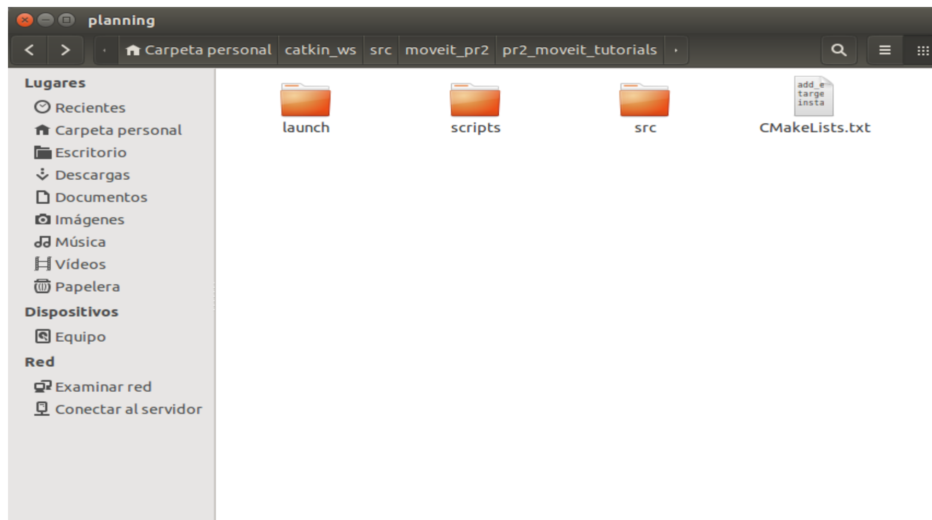


Imagen 14. Carpeta planning

El siguiente paso es acceder a la carpeta launch, en la que encontramos la llamada al archivo que queremos modificar: move_group_interface_tutorial.launch.

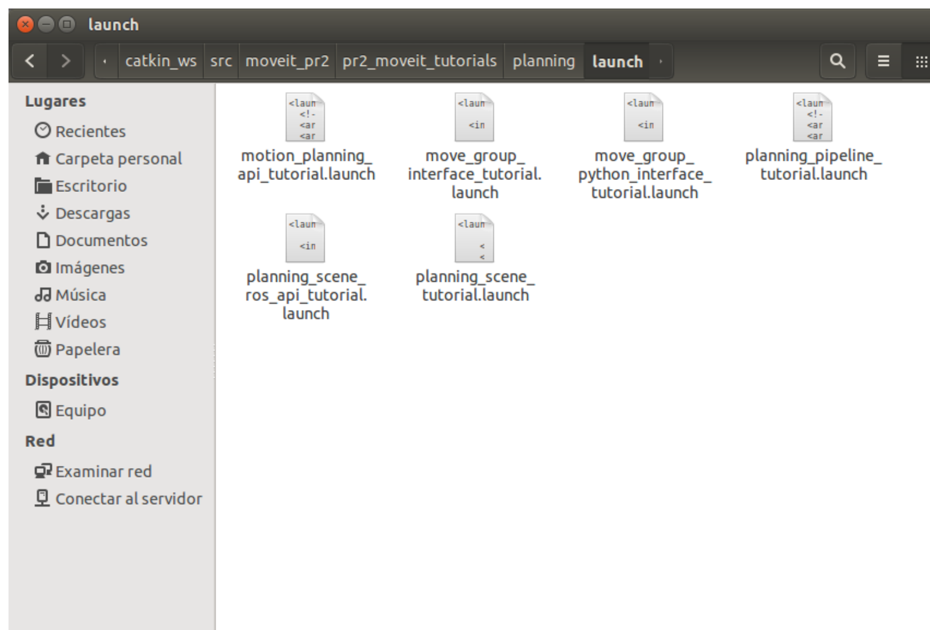
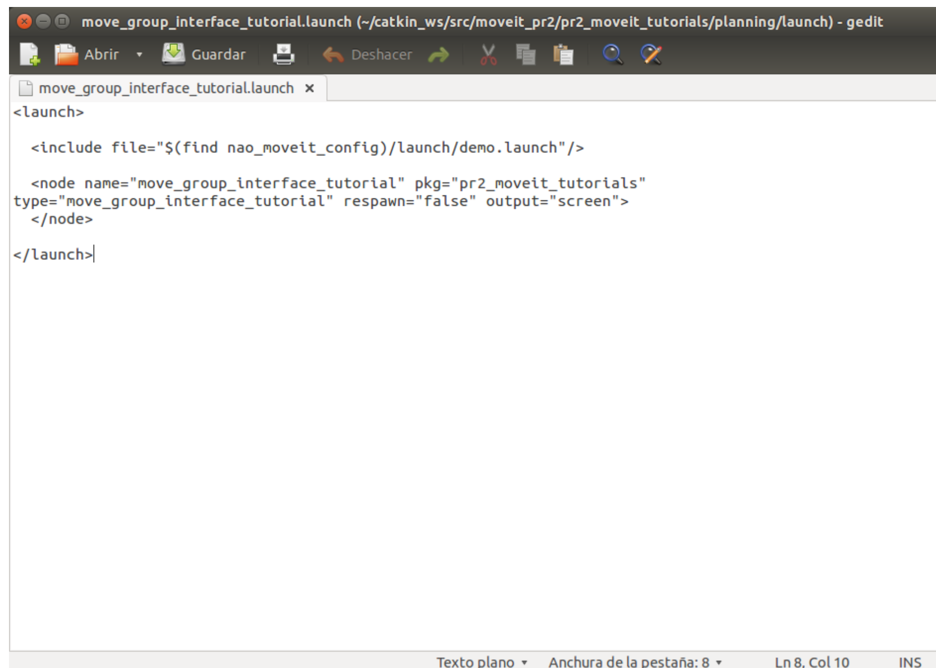


Imagen 15. Archivo move_group_interface_tutorial.launch

Finalmente, abrimos con el editor de texto, el archivo mencionado anteriormente y sustituimos en la segunda línea de código pr2_moveit_config por nao_moveit_config.


```
<launch>

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

  <node name="move_group_interface_tutorial" pkg="pr2_moveit_tutorials"
type="move_group_interface_tutorial" respawn="false" output="screen">
  </node>

</launch>
```

Imagen 16. Llamada robot NAO

Ahora, si ejecutamos en el terminal las siguiente instrucciones, podemos cargar el modelo del robot NAO con los movimientos planificados.

$ cd ~/catkin_ws

$ catkin_make

$ source devel/setup.bash

$roslaunch pr2_moveit_tutorials move_group_interface_tutorial.launch

Sin embargo, todos los movimientos planificados en el nodo no se llevan a cabo, esto se debe a que las configuraciones del robot pr2 y del NAO son distintas y a la hora de cargar las instrucciones creadas para un modelo en otro que no es el de partida se producen errores en los nombres asignados a cada miembro del robot. Por ejemplo, en la mayor parte del código se hace referencia a la posición inicial y final del "end_effector", considerando este como el extremo fina del miembro seleccionado. Este nombre, provoca un error en el modelo del robot humanoide, pues no está definido el nombre anterior como parte de su cuerpo.

Del código de este nodo podemos utilizar las siguientes instrucciones compatibles con el robot humanoide y que a continuación se explicarán detalladamente para conocer la metodología de envío de la información.

Primero se indica qué grupo queremos controlar y planificar, en este caso, se toma por ejemplo el brazo derecho.

$ moveit::planning_interface::MoveGrop group("right_arm");

```
$ moveit::planning_interface::PlanningSceneInterface planning_scene_interface;
```

En este paso, creamos un publicador para poder visualizar los planes en la interfaz gráfica Rviz.

```
$   ros::Publisher display_publisher =
node_handle.advertise<moveit_msgs::DisplayTrajectory>("/move_group/display_planned_path"
,1,true);
```

```
$ moveit_msgs::DisplayTrajectory display_trajectory;
```

Ahora se declara la variable my_plan que más adelante se utilizará en la visualización del plan realizado.

```
$ moveit::planning_interface::MoveGroup::Plan my_plan;
```

```
$ bool success = group.plan(my_plan);
```

Finalmente, de todas las posibles opciones para realizar la planificación que se proponen en el tutorial, tan solo una es válida para el robot humanoide, ya que no utiliza la definición del efector final del miembro que está moviendo.

En este tipo de planificación, primero se obtiene el estado actual del grupo definido anteriormente, al cual se le va a modificar su posición. Esto se realiza mediante las siguientes líneas de comandos:

```
$ std::vector<double> group_variable_values;
```

```
$  group.getCurrentState()  ->copyJointPositions(group.getCurrentState()  ->getRobotModel()  -
>getJointModelGroup(group.getName()), group_variable_values);
```

Una vez se tiene el estado actual del grupo que se va a planificar, el último paso consiste en modificar el movimiento de una de las articulaciones de dicho grupo, planificar el nuevo objetivo y visualizarlo. Para ello se escribe:

```
$ group_variable_values[0]=-1.0;
```

```
$ group.setJointValuueTarget(group_variable_values);
```

```
$ succes = group.plan(my_plan);
```

Si además queremos que el robot ejecute el plan, hay que añadir a la última línea escrita lo siguiente:

```
$ succes = group.execute(my_plan);
```

En este proyecto se pretende generar movimientos en el tronco superior del robot, por ello, es de gran interés conocer los movimientos que pueden realizarse en cada extremidad superior, y cuál es la forma de comunicarlo mediante código en C++. Para ello, el estudio se centrará en la línea de comando escrita anteriormente:

```
$ group_variable_values[0]=-1.0;
```

En la primera parte se hace referencia al grupo llamado anteriormente, en este caso, el brazo derecho. Con el valor entre corchetes se indica la articulación y el tipo de movimiento que esta va a realizar.

Existen cinco números disponibles que hacen referencia a cinco movimientos distintos. El cero y el uno se refieren al hombro, el dos y el tres al codo y el cuatro a la muñeca.

[0]: Rotación del hombro en sentido ascendente y descendente.

[1]: Rotación del hombro de izquierda a derecha y viceversa.

[2]: Rotación del codo en sentido ascendente y descendente.

[3]: Rotación del codo de izquierda a derecha y viceversa.

[4]: Rotación de la muñeca.

Con el último parámetro que se escribe en esta instrucción se indica el sentido del movimiento con el signo del número (positivo o negativo) y la amplitud del movimiento, es decir, el ángulo de rotación que realiza la articulación seleccionada del grupo definido.

Para poder escoger con criterio el valor del movimiento que tiene que realizar el robot, es necesario conocer las posiciones máximas y mínimas de los movimientos de las articulaciones del tronco superior, que en el punto siguiente se van a desarrollar.

3.5.3. Posiciones máximas y mínimas de las articulaciones del tronco superior

Para tener un correcto funcionamiento del robot humanoide es necesario conocer y tener en cuenta los rangos máximo y mínimo entre los cuales las articulaciones pueden realizar los movimientos.

Para acceder a esta información existen dos vías disponibles:

La primera consiste en acceder a la carpeta equipo, y en el siguiente orden a las carpetas: opt/ros/indigo/share/nao_description/urdf/ hasta llegar a la situación que se muestra a continuación en la imagen.

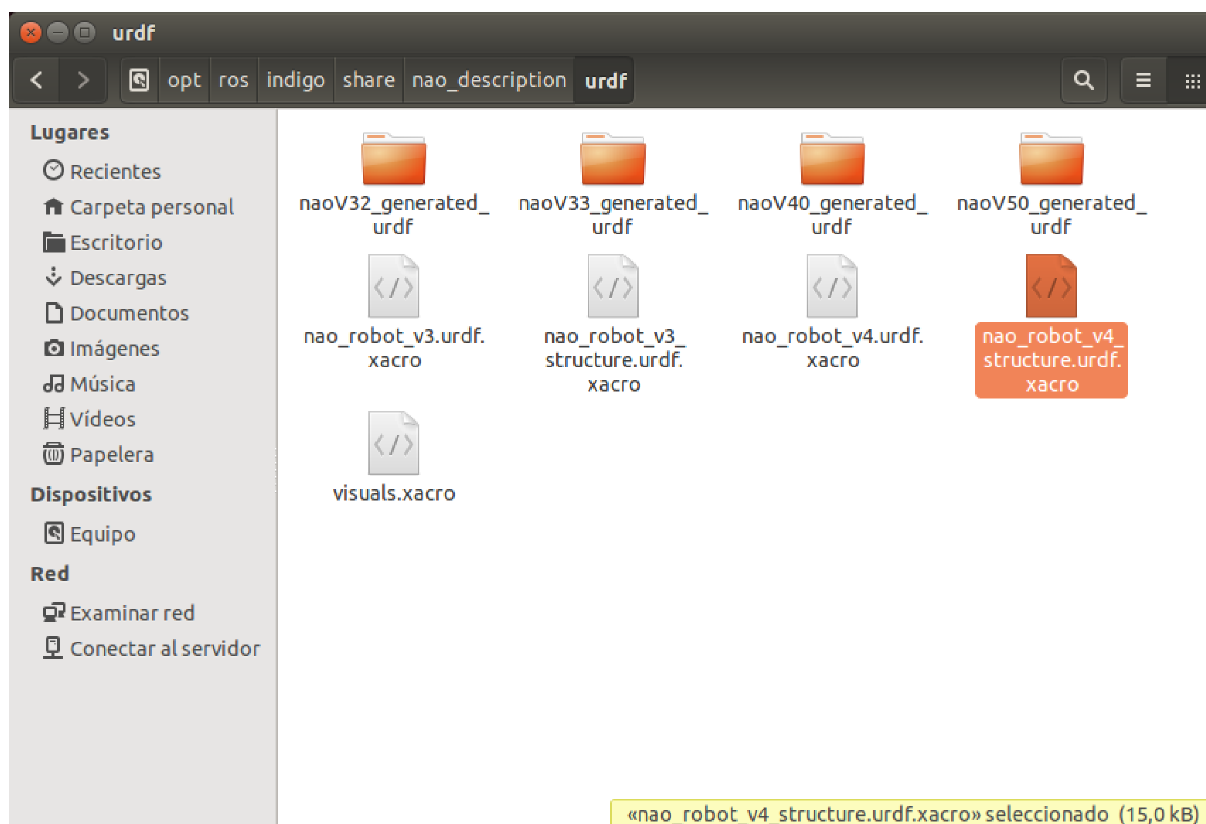


Imagen 17. Archivo nao_robot_v4_structure.urdf.xacro

El siguiente paso consiste en abrir el archivo nao_robot_v4_structure.urdf.xacro, en el cual se especifican los valores máximos y mínimos de los distintos movimientos de las articulaciones.

La segunda opción, es posiblemente más visual y sencilla que la primera, ya que consiste en activar un desplegable de forma automática cada vez que se lanza la interfaz gráfica Rviz con la llamada del nodo que estamos construyendo. Para configurar este entorno debemos acceder a la carpeta equipo, y a las siguientes carpetas con el siguiente orden: opt/ros/indigo/share/nao_moveit_config/launch, hasta llegar a la siguiente configuración:

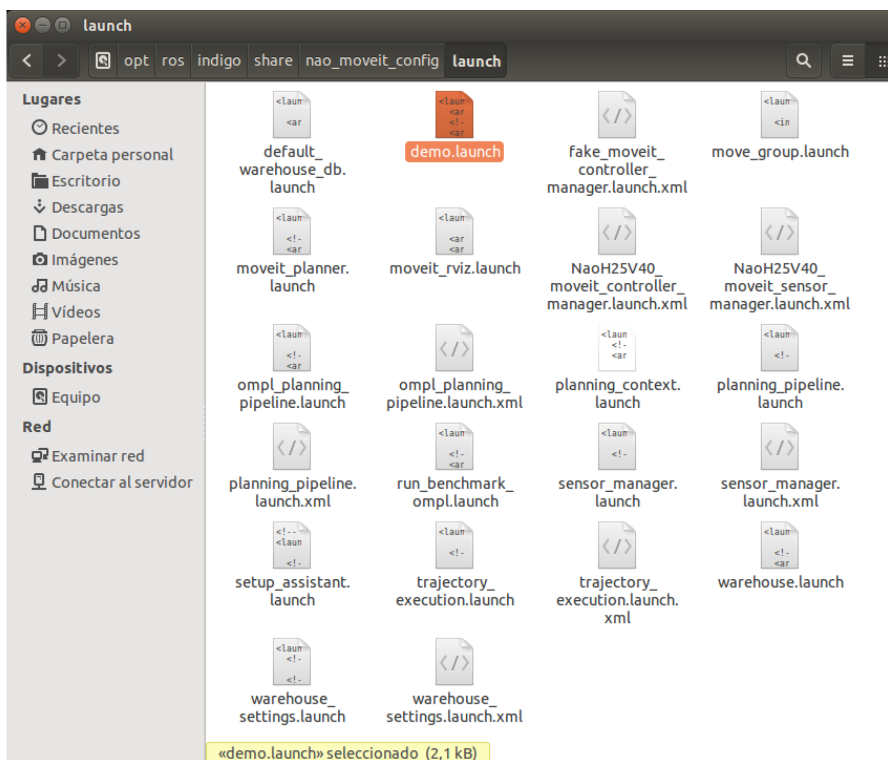


Imagen 18. Archivo demo.launch

Ahora se debe abrir el archivo demo.launch y modificar una de las líneas de comando, pero para que esta acción pueda realizarse, desde el terminal debemos acceder a este documento y autorizar su modificación.

Una vez tengamos el código abierto mediante la terminal, hay que buscar la línea que dice:

<parm name="/use_gui" value="false"/>

Y, modificarla por:

<parm name="/use_gui" value="true"/>

Con esto se consigue activar la aparición de un listado en otra ventana que se abre al lanzar la interfaz gráfica Rviz. En esta nueva ventana aparecen todos los posibles movimientos de cada articulación del robot humanoide, estos valores se muestran en relación al instante en el que se encuentra el robot. Sin embargo, si desplazamos la barras naranjas de cada movimiento a los extremos derecho e izquierdo obtenemos los valores máximo y mínimo.

De todos los rangos que aparecen en esta nueva ventana y que se pueden visualizar en la siguiente imagen, solo son de interés aquellos que pertenecen al tronco superior, en concreto los que hacen referencia al hombro y al codo, tanto derecho como izquierdo.

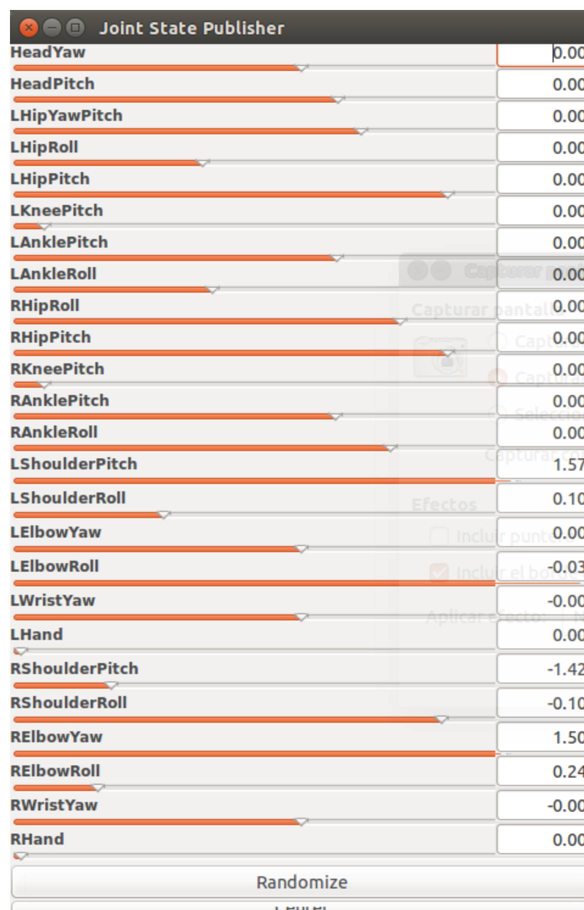


Imagen 19. Listado de rangos de movimientos

Los rangos son los siguientes:

-Right elbow roll (giro del codo derecho de izquierda a derecha):[0.03 , 1.54]

-Left elbow roll (giro del codo izquierdo de izquierda a derecha): [-1.54 , -0.03]

-Right elbow yaw (giro del codo derecho de abajo a arriba):[-2.09 , 2.09]

-Left elbow yaw (giro del codo izquierdo de abajo a arriba): [-2.09 , 2.09]

-Right shoulder roll (giro del hombro derecho de izquierda a derecha):[-1.33 , 0.31]

-Left shoulder roll (giro del hombro izquierdo de izquierda a derecha): [-0.31 , 1.33]

-Right shoulder pitch (giro del hombro derecho de arriba a abajo):[-2.09 , 2.09]

-Left shoulder pitch (giro del hombro izquierdo de arriba a abajo): [-2.09 , 2.09]

3.5.4. Transformación del espacio de la Kinect al espacio del robot humanoide

3.5.4.1. Introducción

Para poder manejar los brazos del robot de forma remota por el usuario que esté situado frente al sensor de captación del movimiento, es necesario tener en cuenta diversos aspectos, ya que los datos que son captados mediante la Kinect no se pueden enviar directamente al robot. Además como se ha visto en el apartado anterior, los movimientos de las articulaciones del robot tienen unos máximos y mínimos establecidos que como es comprensible, deben coincidir con aquellos que presenten los movimientos que sean enviados al robot.

Por otra parte, el cuerpo del robot se asemeja al de un ser humano pero tiene variantes, por ejemplo, los brazos de una persona tienen siete grados de libertad: tres en la muñeca, tres más en el hombro y uno en el codo. Sin embargo, los brazos del robot NAO tienen cinco grados de libertad: uno en la muñeca, dos en el hombro ("pitch and roll") y dos en el codo ("roll and yaw"). Por lo tanto la configuración de los movimientos que realiza una persona y un humanoide son distintas.

Una vez expuesta la situación, el objetivo que ahora se presenta es la transformación de la información captada con la Kinect, al espacio de coordenadas del robot.

3.5.4.2. Configuración

Para configurar esta conversión de espacios de coordenadas, se ha recurrido a la información contenida en un paquete publicado en internet. Este paquete se encuentra en un repositorio de github con el nombre de rsait_public_packages.

El primero paso que se ha de realizar es la descarga del paquete para poder acceder a la información de las transformaciones y así poco a poco ir configurando el nodo que se está creando.

Para que el nodo tenga acceso al grupo de transformaciones (/tf) tenemos que añadirle el correspondiente componente en la ejecución del archivo CmakeLists.txt de nuestro paquete. Para ello se deben seguir los siguientes pasos:

Hay que acceder a la carpeta catkin_ws y, en el siguiente orden a las carpetas: src/moveit_pr2/pr2_moveit_tutorials hasta llegar a esta situación que aparece en la imagen 20. Después se ha de acceder al archivo CmakeLists.txt y como se muestra en la imagen 21 agregar el componente tf para que cuando se ejecute el nodo, este encuentre la información correspondiente a las transformaciones.

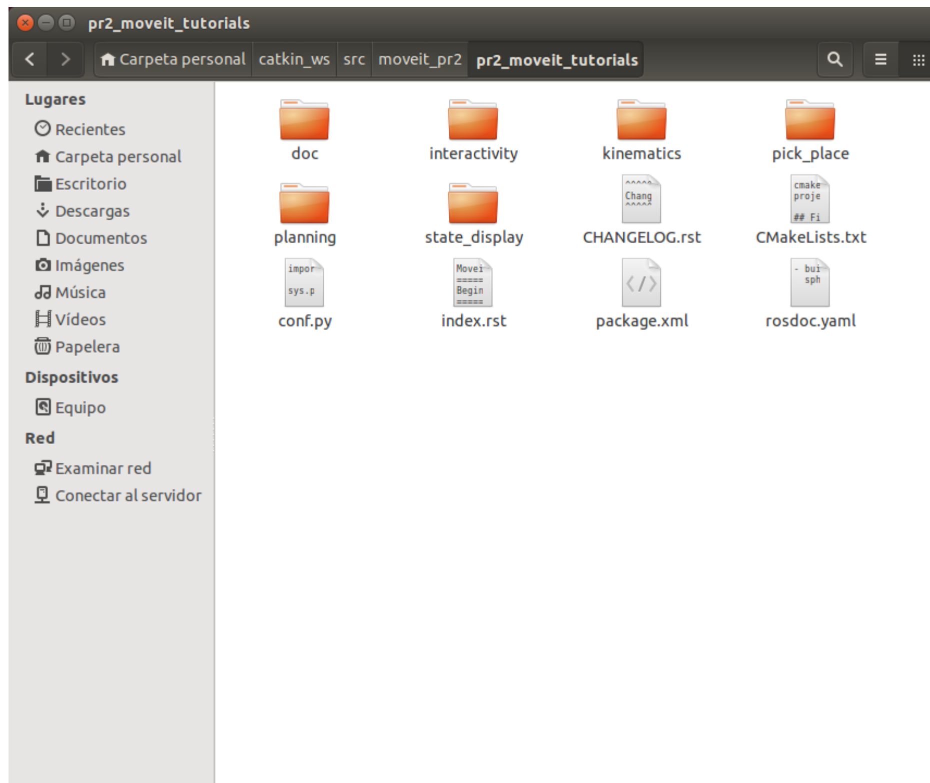
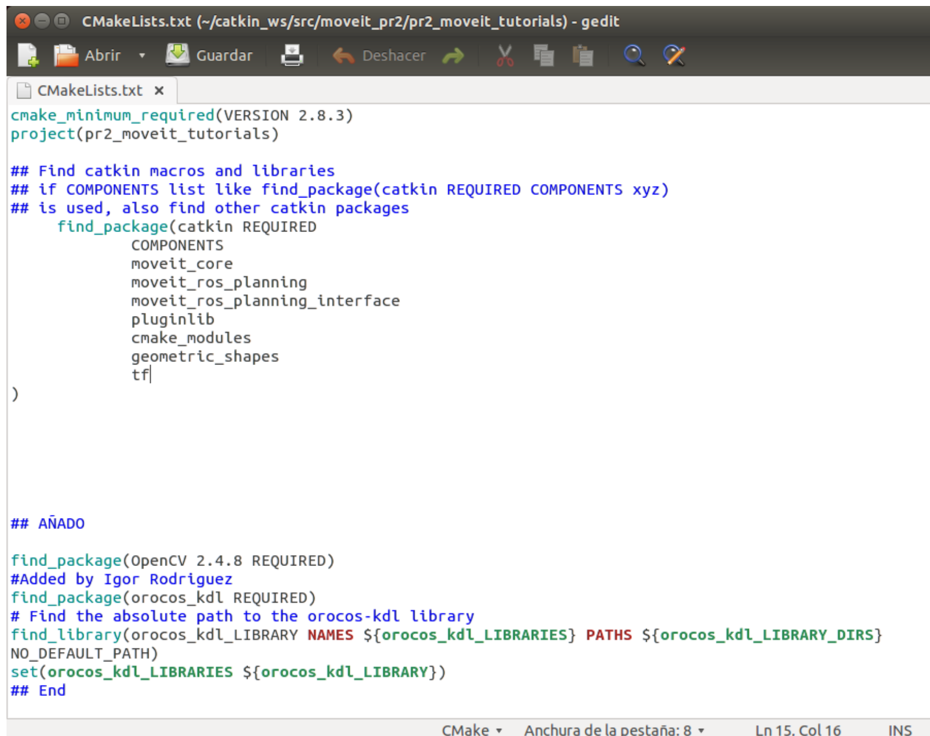


Imagen 20. Carpeta pr2_moveit_tutorials



```
cmake_minimum_required(VERSION 2.8.3)
project(pr2_moveit_tutorials)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
      find_package(catkin REQUIRED
                COMPONENTS
                moveit_core
                moveit_ros_planning
                moveit_ros_planning_interface
                pluginlib
                cmake_modules
                geometric_shapes
                tf
)






## AÑADO

find_package(OpenCV 2.4.8 REQUIRED)
#Added by Igor Rodriguez
find_package(orocos_kdl REQUIRED)
# Find the absolute path to the orocos-kdl library
find_library(orocos_kdl_LIBRARY NAMES ${orocos_kdl_LIBRARIES} PATHS ${orocos_kdl_LIBRARY_DIRS}
NO_DEFAULT_PATH)
set(orocos_kdl_LIBRARIES ${orocos_kdl_LIBRARY})
## End
```

Imagen 21. Archivo CmakeLists.txt

3.5.4.3. tf en ROS

Como se ha nombrado anteriormente, uno de los principales problemas a resolver es la transformación de espacios de coordenadas. Para su solución el paquete tf de ROS juega un importante papel, por ello a continuación se va a explicar su funcionamiento y las herramientas de las líneas de comando que se necesitan.

Según la propia página web de ROS [18], tf es un paquete que permite al usuario realizar un seguimiento de múltiples marcos de coordenadas a lo largo del tiempo. Además, tf mantiene la relación entre los marcos de coordenadas en una estructura tipo árbol organizada en el tiempo; permitiendo al usuario transformar puntos y vectores entre dos marcos de coordenadas en cualquier punto deseado.

Un sistema robótico normalmente tiene multitud de marcos 3D que cambian en función del tiempo, entre ellos se encuentran el marco global (world frame), el marco de base (base frame) y el marco de agarre (gripper frame). Con el paquete tf, existe la opción de realizar un seguimiento a todos estos marcos a través del tiempo y responder a preguntas como: cuál es la postura del objeto que sostiene mi pinza en relación a mi base, cuál es la pose actual de la base en el marco global y otras muchas que pueden surgir y que se pueden responder, ya que el paquete tf puede operar en un sistema distribuido; lo que significa que toda la información sobre los marcos de las coordenadas del robot están disponibles para todos los componentes de ROS.

3.5.4.4. Transformaciones tf en el nodo

En el desarrollo del nodo, las transformaciones tienen un papel fundamental. Para realizar correctamente esta función se deben definir ciertas variables y aplicar las siguientes ecuaciones.

En primer lugar se deben definir vectores de tipo KDL, los cuales son de gran utilidad para cadenas y árboles cinemáticas; así como para espacios en 3D y transformaciones. Por ello se definen los siguientes vectores para los distintos grupos del robot.

```
KDL::Vector left_hand;

KDL::Vector left_elbow;

KDL::Vector left_shoulder;

KDL::Vector right_hand;

KDL::Vector right_elbow;

KDL::Vector right_shoulder;
```

También se define la variable tl como un puntero y del siguiente tipo:

```
tf::TransformListener* tl;
```

Además se definen antes de la función principal (main) las siguientes funciones que crean el vector de transformadas y recogen las poses del sensor Kinect:

```
void set_vector( tf::StampedTransform t, KDL::Vector& v ){
  v.x(t.getOrigin().x());
  v.y(t.getOrigin().y());
  v.z(t.getOrigin().z());
}
bool get_poses(){
  try{
  tf::StampedTransform stf;
  tl->lookupTransform( global_frame, "right_shoulder" +suffix, ros::Time(0), stf );
  set_vector( stef, left_shoulder );
  tl->lookupTransform( global_frame, "right_elbow" +suffix, ros::Time(0), stf );
  set_vector( stef, left_elbow );
  tl->lookupTransform( global_frame, "right_hand" +suffix, ros::Time(0), stf );
  set_vector( stef, left_hand );
  tl->lookupTransform( global_frame, "left_shoulder" +suffix, ros::Time(0), stf );
  set_vector( stef, right_shoulder );
  tl->lookupTransform( global_frame, "left_elbow" +suffix, ros::Time(0), stf );
  set_vector( stef, right_elbow );
  tl->lookupTransform( global_frame, "left_hand" +suffix, ros::Time(0), stf );
  set_vector( stef, right_hand );
  time_ = stf.stamp_;
}
catch(  tf::TransformException &ex ){
```

```
if ( count==0){
  ROS_WARN( "Waiting for Calibration Pose");
  count++;
  }
return false;
}
return true;
}
```

Una vez definidas estas dos funciones, en la función principal se crea un bucle del tipo:

```
while ( ros::ok() )
```

Dentro de este bucle se llamarán a las funciones que se acaban de definir y se realizarán los cálculos pertinentes para enviarlos al robot de forma continua hasta que por terminal se finalice el bucle escribiendo Ctrl+C.

De forma resumida, sin incluir todos los cálculos realizados, se van a exponer los componentes principales que aparecen en el bucle while de la función main y que se refieren a las transformaciones. Esta parte del código se ha tomado como se ha comentado anteriormente de uno de los nodos del paquete rsait_public_packages, publicado en el repositorio github.

```
if( !get_poses() ){
  ROS_INFO("Espera una pose");
  loop_rate.sleep();
  ros::spinOnce();
  continue;
}
tf::StampedTransform stf;
ros::Time now = ros::Time::now();
tl -> waitForTranform( " /openni_depth_frame " , " /right_shoulder " +suffix , ros::Time(0) , ros::Duration(20.0) );
tl -> lookupTransform( " /openni_depth_frame " , " /right_shoulder " +suffix , ros::Time(0), stf);
```

```
tl -> lookupTransform(  global_frame  , " /right_shoulder " +suffix , ros::Time(0), stf);
```

A continuación se detallan los cálculos realizados para los dos movimientos disponibles para el hombro izquierdo y los dos movimientos del codo derecho. Para el brazo izquierdo los cálculos son análogos.

Cálculo del ángulo "roll" del codo izquierdo:

```
KDL::Vector left_elbow_hand( left_hand - left_elbow);

KDL::Vector left_elbow_shoulder( left_shoulder - left:elbow);

left_elbow_hand.Normalize();

left_elbow_shoulder.Normalize();

static double left_elbow_angle_roll = 0;

{

left_elbow_angle_roll = acos(KDL::dor(left_elbow_shoulder,eft_elbow_hand));

left_elbow_angle_roll  = left_elbow_angle_roll - PI;

}
```

Cálculo del ángulo "yaw" del codo izquierdo:

```
static double left_elbow_angle_yaw = 0;

left_elbow_angle_yaw = (left_elbow_hand[2] / sin(legt_elbow_angle_roll)) * HALFPI;

if (left_elbow_angle_yaw > 1.5) left_elbow_angle_yaw = 1.5;

else if(left_elbow_angle_yaw < -1.5) left_elbow_angle_yaw  = -1.5;
```

Cálculo del ángulo "roll" del hombro izquierdo:

```
KDL::Vector left_shoulder_elbow(left_elbow - left_shoulder);

KDL::Vector left_shoulder_neck(right_shoulder - left_shoulder);

left_shoulder_elbow.Normalize();

left_shoulder_neck.Normalize();

static double left_shoulder_angle_roll = 0;

left_shoulder_angle_roll = acos(KDL::dor(left_shoulder_elbow, left_shoulder_neck));
```

```
left_shoulder_angle_roll = left_shoulder_angle_roll - HALFPI;

}
```

Cálculo del ángulo "pitch" del hombro izquierdo:

```
static double left_sholder_angle_pitch = 0;

{

left_shoulder_angle_pitch = asin(left_shoulder_elbow.z());

left_shoulder_angle_pitch = -( left_shoulder_angle_pitch);

}
```

3.5.5. Creación de un nuevo grupo

Según los grupos definidos para el robot NAO, entre los que se encuentran el brazo derecho e izquierdo, la mano derecha e izquierda, la cabeza, etz; una vez realizadas las transformaciones del espació de la información recibida a través del sensor a la información útil para el robot humanoide, el procedimiento para enviar esa información sería el siguiente:

Para el brazo izquierdo:

```
$ std::vector<double> group_variable_values;

$ group.getCurrentState() ->copyJointPositions(group.getCurrentState() ->getRobotModel() ->getJointModelGroup(group.getName()), group_variable_values);

$ group_variable_values[0] = left_shoulder_ange_pitch;

$ group_variable_values[1] = left_shoulder_ange_roll;

$ group_variable_values[2] = left_elbow_ange_yaw;

$ group_variable_values[3] = left_elbow_ange_roll;

$ group.setJointValuueTarget(group_variable_values);

$ succes = group.plan(my_plan);

$ succes = group.execute(my_plan);
```

Para el brazo derecho:

```
$ std::vector<double> group2_variable_values;
```

$ group2.getCurrentState() ->copyJointPositions(group2.getCurrentState() ->getRobotModel() ->getJointModelGroup(group2.getName()), group2_variable_values);

$ group2_variable_values[0] = left_shoulder_ange_pitch;

$ group2_variable_values[1] = left_shoulder_ange_roll;

$ group2_variable_values[2] = left_elbow_ange_yaw;

$ group2_variable_values[3] = left_elbow_ange_roll;

$ group.2setJointValuueTarget(group2_variable_values);

$ succes = group2.plan(my_plan);

$ succes = group2.execute(my_plan);

Así, al ejecutar el nodo creado conseguimos el control de los dos brazos. Sin embargo el movimiento que se logra replicar no es continuo, ya que como se puede observar en esta parte del código de programación, primero se planifica y ejecuta el movimiento del brazo derecho y después el del brazo derecho.

Para lograr la planificación y ejecución de un movimiento compuesto más elaborado, debería existir un grupo definido que englobara el movimiento de los dos brazos. Como no es el caso, el paso a realizar es la modificación de los grupos definidos para el robot NAO para poder crear uno nuevo.

El primer paso es entrar mediante el terminal a las siguientes carpetas: equipo/opt/ros/indigo/share/nao_moveit_config/config hasta llegar a la siguiente ventana:

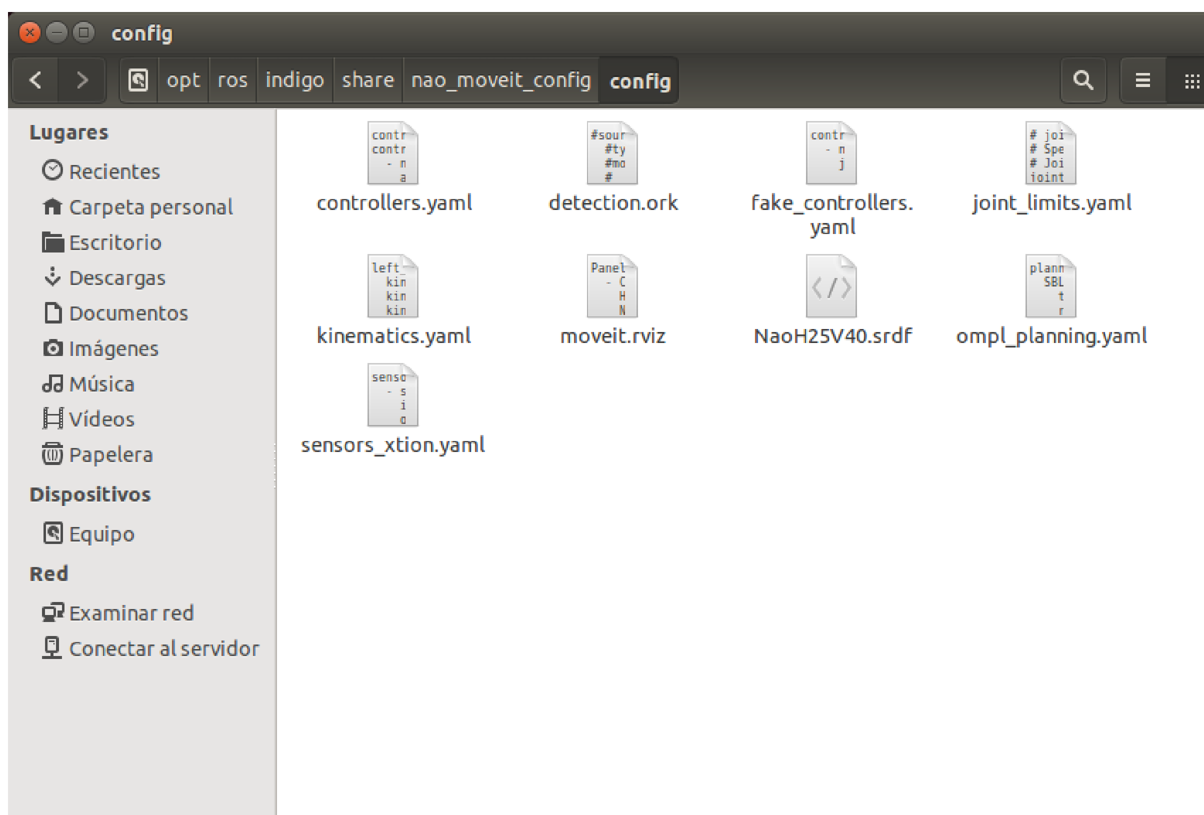


Imagen 22. Carpeta config

De los archivos que aparecen se ha de seleccionar el que tiene por nombre NaoH25V40.srdf, para ello se escribe por el terminal:

$ sudo gedit NaoH25V40.srdf

Con esta acción, se abrirá el archivo con el editor de textos gedit para poder modificarlo. Para incluir un nuevo grupo que incluya los dos brazos simultáneamente se escribe en el archivo abierto lo siguiente:

</group>

<group name"arms">

<chain base_link="torso" tip_link"l_wrist" />

<chain base_link="torso" tip_link"r_wrist" />

Con este nuevo trozo de código se configura un nuevo grupo llamado "arms", el cual contiene la cadena cinemática que tiene como referencia el torso del robot y va hasta la muñeca izquierda y la cadena cinemática que también cuenta con el torso como referencia y finaliza en la muñeca derecha.

Por lo tanto, para realizar un movimiento mas continuo, en el nodo se puede llamar al grupo arms en lugar de llamar a dos grupos por separado, uno para cada brazo. La escritura de la información quedaría de la siguiente forma:

```
$ std::vector<double> group_variable_values;

$ group.getCurrentState() ->copyJointPositions(group.getCurrentState() ->getRobotModel() ->getJointModelGroup(group.getName()), group_variable_values);

$ group_variable_values[0] = left_shoulder_ange_pitch;

$ group_variable_values[1] = left_shoulder_ange_roll;

$ group_variable_values[2] = left_elbow_ange_yaw;

$ group_variable_values[3] = left_elbow_ange_roll;


$ group_variable_values[5] = left_shoulder_ange_pitch;

$ group_variable_values[6] = left_shoulder_ange_roll;

$ group_variable_values[7] = left_elbow_ange_yaw;

$ group_variable_values[8] = left_elbow_ange_roll;


$ group.setJointValuueTarget(group_variable_values);

$ succes = group.plan(my_plan);

$ succes = group.execute(my_plan);
```

Así, se tiene un vector con más capacidad que almacena el valor de las articulaciones de cada brazo, permitiendo la reproducción de un movimiento compuesto.

Es importante destacar el salto de posición en el vector al pasar del brazo izquierdo al brazo derecho, dejando la posición número 4 sin ningún valor asignado. Esto se debe realizar de esta forma, porque como se vio en el punto 3.5.2. Control del movimiento mediante la API de C++, en la posición número 4 del vector group_variable_values se encuentra el movimiento que se le asigna a la rotación de la muñeca, que en este caso no es de interés.

3.5.6. Planificación de movimientos simples.

Con el fin de lograr desarrollar una programa con vistas a una futura aplicación, se ha decidido incluir en el código del nodo la posibilidad de crear movimientos simples. Para ello, se pretende realizar la planificación de un movimiento, visualizar la trayectoria y poder escoger si se desea guardarlo o no para una posterior aplicación.

En este procedimiento, se permite al usuario interactuar con la recogida de planes, decidiendo si se quiere guardar el movimiento realizado para una posterior ejecución. Gracias a esta aplicación se

permite obtener un vector que contiene movimientos simples que en un futuro podrían ser guardados en ficheros para realizar el ensamblaje de los mismos y constituir así cadenas de tareas automatizadas que previamente han sido programadas y almacenadas. Todo esto sin la necesidad de tener frente al sensor de captación un usuario realizando los mismos movimientos.

Para ello dentro de la función principal y del bucle en el que se capta la información de la Kinect, se transforma y se envía al robot, se realiza otro bucle while de la siguiente manera:

```
while( pl == 1 ){

sleep(5.0);

Resto del código

}
```

Siendo "pl" una variable definida como entera (int) y con valor inicial 1.

Una vez realizados todos los cálculos correspondientes y ejecutada la planificación de una pose, se pregunta por el terminal si el movimiento realizado es correcto y se desea guardarlo en un vector previamente definido.

Definición del vector:

```
moveit::planning_interface::MoveGroup::Plan my_plan1[30];

bool success1;
```

Planificación del movimiento (dentro del bucle while(pl == 1)):

```
success1 = group.plan(my_plan1[i]);

printf( "¿La trayectoria es la deseada? 1=si 0=no" \n);

scanf( "%d", &num);

if( num == 1){

i++;

}

if( num == 0 ){

i=i;

}
```

Además también se pregunta si se quiere continuar planificando nuevos planes.

```
printf( "¿Desea realizar nuevos planes? 1=si 0=no"\n);

scanf( "%d", &pl);
```

Si se escoge la opción de volver a realizar un nuevo movimiento básico, se repite el mismo procedimiento pero incrementando en una posición el valor de la i para que en caso de guardar otro movimiento se mantenga en el vector el movimiento anterior.

En caso de escoger la opción negativa, se sale del bucle while (pl == 1) ya que pl pasaría a tener un valor de 0.

Una vez fuera de este bucle se propone la opción de ejecutar los planes guardados en el vector my_plan.

```
printf( "¿Desea ejecutar los planes guardados? 1=si 0=no" \n);

scanf( "%d"; num);

i=0;

if( num ==1 ){

  for( i=0; i<30;i++){

  succes1 = group.execute(my_plan1[i]);

  }

}
```

El bucle for que recorre el vector de movimientos guardados realiza el mismo número de iteraciones que posiciones tenga el vector en su definición. En este caso se ha tomado por ejemplo treinta, pero podría tomarse otro valor distinto siempre y cuando el tamaño del vector definido coincida con el número de iteraciones realizadas por el bucle for, para que no hayan problemas de desbordamiento.

3.6. LANZAMIENTO DEL NODO

Para realizar el lanzamiento del nodo y una correcta conexión con el sensor y la interfaz RViz, se deben realizar los siguientes pasos:

En primer lugar, al igual que en el punto 3.4 en el que se detalla la configuración de la Kinect, ahora se deben de realizar los mismos pasos, es decir, se escribe en una terminal:

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch openni_launch openni.launch

Después en otro terminal se escriben las siguientes líneas de comandos:

$ cd ~/catkin_ws

$ source devel/setup.bash

$ rosrun openni_tracker openni_tracker

Una vez se tienen estas dos terminal abiertas, se dejan en espera y en un tercer terminal se escribe el lanzamiento del nodo:

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch pr2_moveit_tutorials move_group_interface_tutorial.launch

Llegados a este punto, es muy importante resaltar un aspecto que ha llevado días de pruebas y búsquedas para resolver un error que aparecía en el terminal al lanzar el nodo, y es que si se lanzan estos tres terminales en pocos segundos aparece en el terminal del lanzamiento del nodo un error que advierte no encontrar la información de la opción openi_depth_frame que proporciona el nodo openni_tracker y con el cual están relacionadas las transformadas del sensor. Pues bien, este error se da si al lanzar el tercer terminal no se sitúa un usuario con suficiente rapidez frente a la Kinect y la calibra con la posición de los brazos en L. Si este paso se efectúa tomando un mayor tiempo, pese a realizar la calibración de la Kinect correctamente, el nodo no llega a sincronizarse con la información que le envía el sensor y por ello se muestra el error.

A continuación se adjuntan tres imágenes para poder visualizar la planificación gráfica que se realiza en la interfaz RViz al ejecutar el nodo:

-En la imagen 23 se muestra el estado inicial de la interfaz tras añadirle el visualizador Image para que aparezca la imagen de la cámara por pantalla. En este momento, el nodo ya ha

realizado la planificación de un movimiento, pero éste no se ha reproducido por el robot porque aun no se había cargado el modelo. Por lo tanto se desecha el plan inicial y se realiza uno nuevo.

-En las imágenes 24 y 25 aparecen dos planes que está realizando el robot, tras la planificación, en el terminal se puede observar cómo se pregunta si la trayectoria que ha realizado el robot es la deseada para guardar o no el plan realizado.

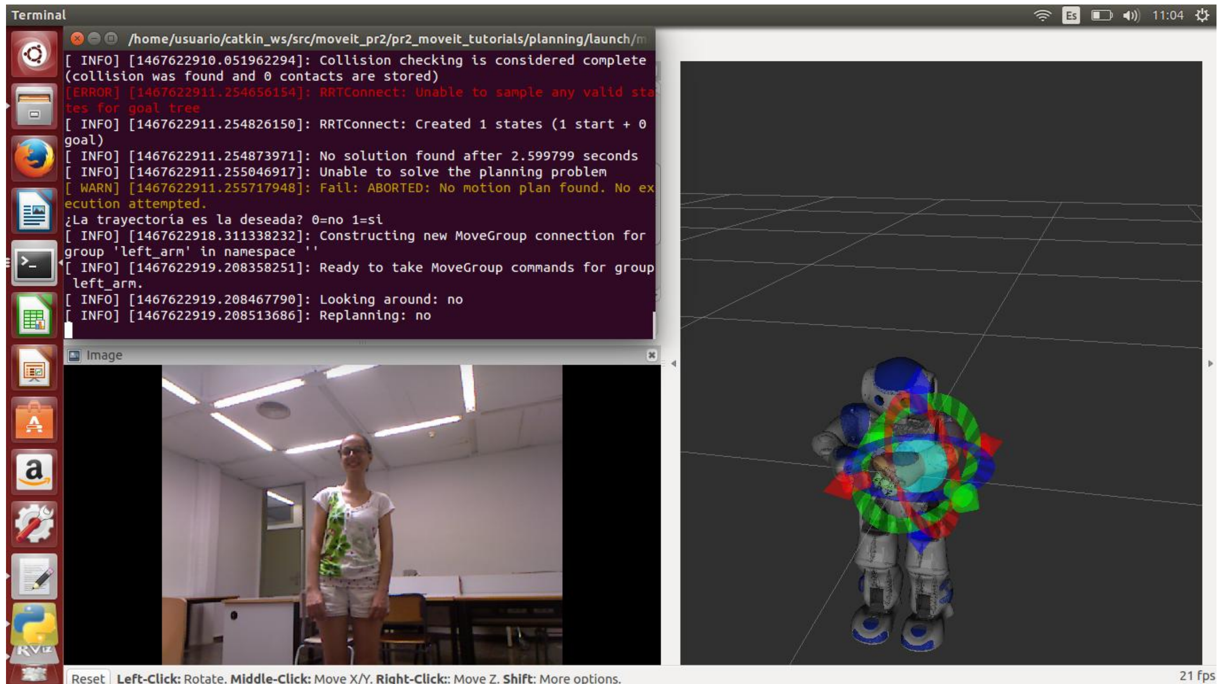


Imagen 23. Estado inicial

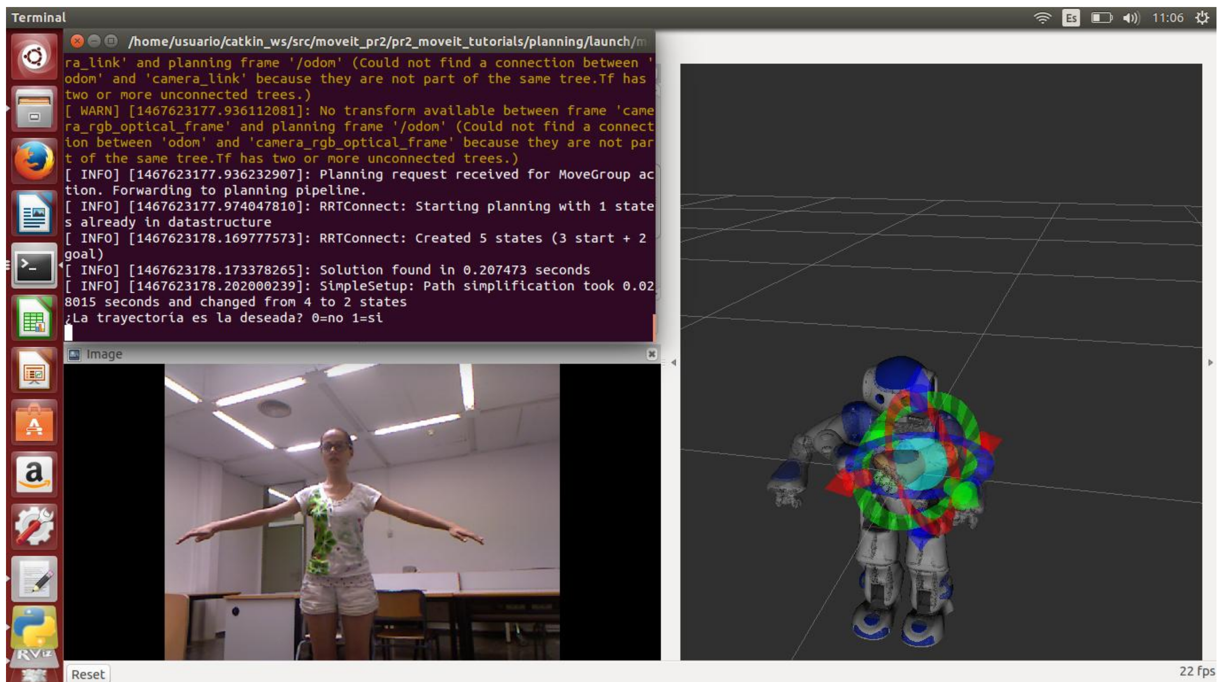


Imagen 24. Primer plan

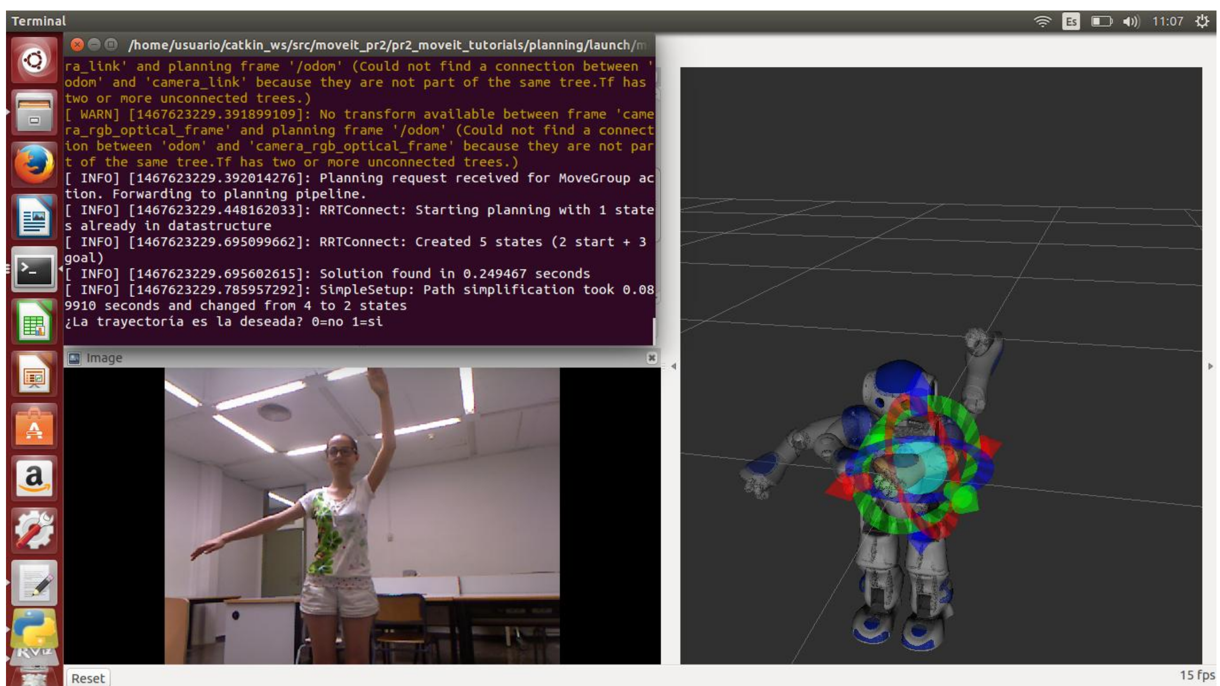


Imagen 25. Segundo plan

CAPÍTULO 4. CONLUSIONES Y TRABAJOS FUTUROS

Llegados a este punto en el que se ha finalizado la parte práctica del proyecto, se puede realizar un balance de todo aquello que se ha logrado.

En primer lugar cabe destacar que todos los objetivos propuestos al inicio del trabajo se han cumplido con éxito ya que finalmente se ha logrado desarrollar un código de programa que constituye una aplicación basada en cámaras 3D para generar movimientos en un robot humanoide NAO.

El objetivo final se ha desarrollado bajo el entorno de programación ROS junto con una de las librerías compatibles que presenta esta plataforma, MoveIt. Con estas dos herramientas que permiten al usuario una gran libertad para consultar, desarrollar y visualizar nuevos proyectos, se ha sustentado el trabajo realizado. Con ello se ha demostrado una de entre las múltiples posibilidades que ofrece ROS, un sistema basado en el código abierto en el que cualquier usuario puede aportar sus conocimientos.

Si se hace un repaso de los objetivos definidos al inicio del trabajo para lograr la meta propuesta se puede concluir que:

-Se han instalado con éxito todas las herramientas necesarias para la elaboración de la aplicación: ROS Indigo y MoveIt así como la configuración previa del entorno de trabajo (catkin workspace).

-Se han instalado correctamente los drivers del sensor de trabajo Kinect, realizando su calibración y junto con la interfaz gráfica RViz se ha probado su funcionamiento, obteniendo el esqueleto virtual gracias al nodo openni_tracker.

-Se ha aprendido a manejar la interfaz gráfica RViz para poder ejecutar a modo de prueba el código creado.

-Se ha desarrollado una aplicación para generar movimientos en un robot humanoide NAO a través de los movimientos que realiza un usuario situado frente al sensor Kinect.

Por lo tanto, tras realizar un balance resumido de todo aquello que se ha ido logrando se puede concluir que se han cumplido con éxito todos los objetivos propuestos al inicio de este trabajo final de grado.

En cuanto a un posible trabajo futuro para mejorar la aplicación creada se puede comentar un sistema más elaborado para recoger los movimientos guardados. Es decir, cada pose final que ha sido planificada con éxito en el simulador RViz podría guardarse en un fichero en lugar de en una posición de un vector. Al guardar los movimientos en ficheros independientes después podría

realizarse la llamada a aquellos que fueran de utilidad con tal de elaborar movimientos complejos con los miembros del tronco superior.

Por otra parte, estos movimientos guardados en ficheros podrían ser automatizables sin la necesidad de tener un usuario frente al sensor realizándolos cada vez que se desee ejecutar uno de ellos. Así, el robot podría realizar una serie de movimientos repetitivos para llevar a cabo tareas automatizadas y que previamente han sido elaboradas correctamente por una persona.

Este sistema de trabajo es similar al que utiliza el programa propio de los robots humanoides NAO. Como bien se ha explicado en el capítulo dos dedicado al desarrollo teórico, el sistema operativo Choreographe dispone de cajas de comportamiento que pueden ser ensambladas para crear cadenas de movimientos. Sin embargo con la aplicación desarrollada se introduce una gran mejora, ya que los movimientos se pueden elaborar por el propio usuario y por lo tanto son más precisos y específicos para las tareas que se pretende que realice el robot. En el sistema operativo Choreographe los movimientos básicos o bien se toman de entre los predefinidos o se definen otros. Sin embargo, para crear movimientos nuevos hay que indicar las posiciones de las articulaciones que formen la pose deseada o soltar los frenos de los motores de un grupo en concreto, moverlo directamente en el cuerpo del robot y una vez esté en la posición correcta volver a frenar los motores. Con este sistema se obtienen movimientos poco naturales al contrario que con la aplicación creada.

Por otra parte, otro posible trabajo a realizar en el futuro sería incorporar el tronco inferior del robot para controlar el desplazamiento de este.

Finalmente, cabe destacar que por motivos de compatibilidad de versiones y tiempo la conexión con un robot real NAO no se ha llegado a realizar.

Debido al periodo establecido que presentan este tipo de trabajos, el desarrollo de la aplicación se ha limitado a cumplir los objetivos iniciales, pero se propone para futuras investigaciones el resto de propuestas que se han planteado.

CAPÍTULO 5. BIBLIOGRAFÍA

[1] "Conceptodefinicion.de" Robot [en línea]

http://conceptodefinicion.de/robot/

[2] Colaboradores de Wikipedia. Robot Humanoide [en línea]. Wikipedia, La enciclopedia libre

https://es.wikipedia.org/wiki/Robot_humanoide

[3] Libro Robótica: manipuladores y robots móviles

[4] Imagen Brazo Robótico

http://reset.etsii.upm.es/es/projects/robotic-arm/

[5] Imagen Robot teleoperado para desactivar explosivos

http://www.fierasdelaingenieria.com/robots-de-desactivacion-de-explosivos-de-la-evolucion-a-la-revolucion/

[6] Imagen Robot zoomórfico

https://ti3-emergente.wikispaces.com/Nueva+Generaci%C3%B3n+de+Robots+-+(Mar%C3%ADa+Fernanda+Ramirez)

[7] Imagen Robot humanoide NAO

http://www.grupo-mediatec.com/robotica/h25.html

[8] http://www.abadiadigital.com/wabot-la-familia-de-robots-humanoides-de-la-universidad-de-waseda/

[9] http://aliverobots.com/

[10] http://www.grupo-mediatec.com/robotica/choregraphe.html

[11] Imagen Choreographe

https://dkor.wordpress.com/2010/04/16/adobe-flash-platform-autodesk-digital-entertainment-creation-and-more/

[12] http://wiki.ros.org/ROS/Introduction

[13] http://wiki.ros.org/ROS/Concepts

[14] http://www.robotnik.es/moveit-un-estandar-para-manipulacion-movil/

[15] https://es.wikipedia.org/wiki/Kinect

[16] Imagen componentes Kinect

http://kinectgva.blogspot.com.es/2014/04/kinect-sdk.html

[17] Pose to calibrate Kinect

http://rock-vacirca.blogspot.com.es/

[18] Buscar ROS tf

http://wiki.ros.org/tf#What_does_tf_do.3F_Why_should_I_use_tf.3F

PRESUPUESTO

PRESUPUESTO

1. NECESIDAD DEL PRESUPUESTO

En el presente documento que a continuación se desarrolla, se pretende realizar una contabilización económica tanto de los esfuerzos realizados como de los materiales utilizados para la realización de este trabajo final de grado. Aunque el ámbito en el que se mueve este proyecto es académico, no hay que olvidar que todo proyecto que quiera ser tratado como tal, y más aquellos que son de carácter ingenieril, deben incluir junto con la memoria el correspondiente presupuesto en el que se contabilicen los gastos que supondría la realización del mismo trabajo en un futuro.

Dado que el ámbito de este proyecto se decanta más por la rama de la investigación, a la hora de elaborar el presupuesto se van a seguir las pautas que se recomiendan en el Centro de Apoyo a la Innovación, la Investigación y la Transferencia de Tecnología (CTT) de la Universidad Politécnica de Valencia. Estas recomendaciones están expuestas en el documento: "Recomendaciones en la elaboración de presupuestos en actividades de I+D+I", en su última versión publicada del año 2015. Esta estructura se considera más apropiada para realizar una valoración económica en este tipo de trabajos que se centran más en el ámbito de la investigación.

2. ESTUDIO ECONÓMICO

2.1. COSTE DE PERSONAL

Para evaluar el coste del personal se va a seguir la siguiente fórmula:

$$\text{Coste } (€) = Ch \times Dh$$

Siendo:

Ch = Coste horario en euros $\left(\frac{€}{h}\right)$

Dh = Dedicación en horas

En el coste de la mano de obra deben incluirse los honorarios, la seguridad social, las indemnizaciones y los costes indirectos.

Si se consulta la tabla del Anexo 2 del documento anteriormente citado, en ella se especifican los diferentes intervalos de coste mínimo y máximo para el caso que se correspondería con este proyecto, es decir, una contratación de personal temporal de un titulado medio.

En las tarifas recomendadas para personal eventual considerando 1760 horas de trabajo anuales (40 horas/semana), se incluyen en el coste horario: el coste de la Seguridad Social (32,1%), las indemnizaciones (3,04%) y el coste indirecto (16,5€/h).

Para un titulado medio el coste horario recomendado se encuentra entre el rango de valores que comprenden el mínimo de 26,8€ y el máximo de 36,2€. Para el cálculo del presupuesto, se considerará un valor medio de 31,5€.

A continuación, en la siguiente tabla se resumen las principales tareas realizadas así como el número de horas dedicada a cada una de ellas.

Trabajo	**Tiempo (h)**	**Coste (€/hora)**	**Coste Parcial (€)**
Instalación de ROS y MoveIt	30	31,5	945
Instalación drivers , configuración y calibración sensor Kinect	20	31,5	630
Desarrollo de la aplicación	200	31,5	6300
Redacción documentos	50	31,5	1575
TOTAL	**300**		**9450**

Tabla 1. Costes de personal

2.2. COSTE MATERIAL INVENTARIABLE

Los equipos utilizados en la elaboración del proyecto se presupuestarán según la amortización y ésta se calculará según la expresión que facilita el CTT en el documento mencionado anteriormente.

$$Coste\ (€) = \left(\frac{Meses\ de\ uso\ del\ equipo\ \times \frac{1\ año}{12\ meses}}{Periodo\ de\ amortización\ en\ años} \right) \times Coste\ del\ equipo\ (€)$$

El periodo de amortización de los equipos utilizados se expresa en años y puede tomar tres valores distintos según la siguiente clasificación:

-La adquisición de equipos para procesos de información y la adquisición de aplicaciones informáticas se evalúa con un periodo de amortización de 6 años.

-La adquisición de maquinaria, instalaciones técnicas, útiles y herramientas u otro inmovilizado material se evalúa con un periodo de amortización de 12 años.

-La adquisición de quipos didácticos y de investigación se evalúa con 10 años de amortización.

Por lo tanto, según lo especificado, el coste de material es el que se detalla en la siguiente tabla:

Concepto	**Precio (€)**	**Meses de uso**	**Periodo de amortización (años)**	**Unidades**	**Coste (€)**
Ordenador portátil	700	4	6	1	38,89
Sistema operativo Ubuntu 12.04	0	4	6	1	0,00
Entorno de programación ROS	0	4	6	1	0,00
Plataforma MoveIt	0	4	6	1	0,00
Simulador RViz	0	4	6	1	0,00
Sensor Kinect	94	4	10	1	3,13
Robot humanoide NAO	7247	4	10	1	241,57
TOTAL					**283,59**

Tabla 2. Coste material inventariable

2.3. COSTE MATERIAL FUNGIBE

En este punto se recoge el coste del material fungible y de aquellos aparatos de vida útil baja que se utilicen en la realización del proyecto:

Concepto	**Coste (€)**
Impresión y encuadernación	15
Folios (200 ud)	2
TOTAL	**17**

Tabla 3. Coste material fungible

3. RESUMEN DEL PRESUPUESTO

En este último punto se resumen los costes calculados en los apartados anteriores y se realiza la suma de todos ellos aplicándoles el correspondiente impuesto para obtener el coste total de la realización del proyecto.

Concepto	**Coste (€)**
Mano de obra (300h graduado en Ing. en Tec. Ind.)	9450
Material Inventariable	283,59
Material Fungible	17
Subtotal	9750,59
IVA (21%)	2047,6239
TOTAL	**11798,2139**

Tabla 4. Coste total

El coste total del proyecto asciende a la cifra total de: ONCE MIL SETECIENTOS NOVEINTA Y OCHO EUROS Y VEINTIDOS CÉNTIMOS.