Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA
Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Algoritmos de ‘graspit’ (agarre) en función de la herramienta para el brazo robot IRB 120.

ESCUELA POLITECNICA
SUPERIOR

Autor: Juan Martín Andrés

Tutor/es: Rafael Barea Navarro

2015

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Algoritmos de 'graspit' (agarre) en función de la herramienta para el brazo robot IRB 120"

Autor: Juan Martín Andrés
Director: Rafael Barea Navarro

TRIBUNAL:

Presidente: Elena López Guillén

Vocal 1º: Juan Jesús García Domínguez

Vocal 2º: Rafael Barea Navarro

FECHA: .....

AGRADECIMIENTOS

A mis profesores y compañeros de carrera, por compartir conmigo buenos problemas que resolver.

Por ser amigos.

A mis amigos, por su alegría y optimismo sin parangón matemático cuantificable.

Por ser familia.

A mi familia, por su apoyo durante todo este tiempo, y en especial, a mi madre, por sufrir en mis derrotas y enorgullecerse en mis victorias.

Por ser única.

¿Pero quién te dijo que la vida fuera justa?

David Fuentes

RESUMEN

En este trabajo se implementan técnicas de GraspIt! sobre el robot IRB 120, que mediante el entorno de ROS y junto a nodos de visión complementan y desarrollan las posibilidades de acción del robot.

Se presentan dos aplicaciones para demostrar la funcionabilidad del paquete de agarre desarrollado: una aplicación de clasificación industrial de objetos, y una aplicación que integra al brazo robot dentro de un laboratorio químico.

Palabras clave: GraspIt!, ABB IRB 120, ROS.

ABSTRACT

This project implements algorithms from GraspIt! for an IRB 120 robot, working with ROS and a vision module, to enhance de capabilities of the robot.

Two applications using the grasp package are developed: an application of industrial classification, and an application to integrate a robot arm in a chemical laboratory.

Keywords: GraspIt!, ABB IRB 120 , ROS.

RESUMEN EXTENDIDO

El departamento de electrónica de la Escuela Politécnica Superior ha trabajado de manera continua sobre el desarrollo del brazo robot IRB 120. Este brazo, mediante el entorno de ROS, puede desplazarse mediante el uso de planificadores, tiene definida aplicaciones de visión artificial mediante Kinect, además de control de la herramienta BarrettHand. Estos paquetes permiten efectuar acciones directas sobre la planta de robot: lo que se pretende en este trabajo es dotar al sistema de inteligencia lógica asociada a la forma en la que los objetos pueden ser agarrados en el entorno, de tal forma que el paquete permita de forma fluida y práctica ofrecer soluciones de agarre precisas para la herramienta de agarre acoplada al robot.

Mediante técnicas de desarrollo de objetos por primitivas y software planificador asociado a estas, como GraspIt!, se pretende simular agarres para objetos singulares, de tal forma que la información pueda ser luego almacenada, y que el robot pueda acceder a esta información, valorarla y obtener el mejor agarre que pueda ofrecer.

Esto permite al robot actuar con su entorno de forma dinámica, recreando situaciones de agarre tal como lo haría un humano, mediante el proceso de aprendizaje.

Este proyecto continua a partir del trabajo previo realizado por Alejandro Gómez Rubio, Alberto Lázaro Enguita y Javier Moriano Martín, desarrolladores del sistema de control de la BarrettHand [1], visión artificial [2] y control de trayectorias del robot [3], respectivamente.

Así pues, una vez recreadas las condiciones de funcionamiento requeridas, el proyecto aborda los siguientes puntos:

-Pruebas de las soluciones de agarre aportadas por GraspIt!: Se harán pruebas de la calidad de los agarres y las capacidades de las herramientas para realizarlos.

-Diseño de lógica de actuación: se implementará mediante código software algoritmos para dotar al sistema de inteligencia, de tal forma que pueda operar de forma autónoma, y que pueda responder con soluciones prácticas ante problemas de agarre o situaciones comprometidas.

-Aplicaciones: se realizarán una serie de aplicaciones para mostrar el potencial de la aplicación y sus posibles usos dentro del apartado industrial. La primera aplicación complementa el desarrollo de una aplicación de clasificación industrial al aportar soluciones de agarre directas desde el paquete de agarre, y la segunda aplicación propone la implementación de un brazo robot dentro de un laboratorio de investigación química, aplicando condiciones de calidad en el agarre.

ÍNDICE GENERAL

RESUMEN 7
ABSTRACT 9
RESUMEN EXTENDIDO 11
ÍNDICE DE FIGURAS 15
ÍNDICE DE TABLAS 17
GLOSARIO 19

CAPÍTULO 1 - INTRODUCCIÓN AL PROYECTO 21
1.1 DESARROLLO DE LA ROBÓTICA 22
1.2 LA ROBÓTICA EN LA INDUSTRIA 23
1.3 GRIPPERS 24
1.4 ANTECEDENTES AL PROYECTO 26
1.5 TEORÍA DE LA ROBÓTICA Y VISIÓN GENERAL 28

CAPÍTULO 2 - ARQUITECTURA GENERAL DEL SISTEMA 33
2.1 ROS 34
2.1.1 Introducción 34
2.1.2 Paquetes 35
2.1.3 Nodos 36
2.1.4 Mensajes 37
2.1.5 Servicios 40
2.2 GRIPPER TIPO BARRETTHAND 42
2.2.1 Introducción 42
2.2.2 Características 42
2.2.3 Estructura y parámetros mecánicos 43
2.2.3.1 Mecanismo TorqueSwitch 43
2.2.3.2 Mecanismo Joint-Torque Sensor 43
2.2.3.3 Cinemática de la BarrettHand 44
2.2.3.4 Nodo de control 45
2.3 VISIÓN ARTIFICIAL MEDIANTE KINECT 46
2.3.1 Introducción 46
2.3.2 Características 46
2.3.3 PCL y asociación a primitivas 47
2.3.4 Paquete de visión mediante Kinect 48
2.4 BRAZO ROBOT IRB 120 49
2.4.1 Introducción 49
2.4.2 Estructura y parámetros mecánicos 49
2.5 CONTROL POR MOVEIT! 50
2.6 SIMULACIÓN MEDIANTE ROBOTSTUDIO 51
2.7 Grasplt! 52
2.7.1 Introducción 52
2.7.2 Descripción del entorno de Grasplt! 52
2.7.3 Modelado en el entorno de Grasplt! 54
2.7.4 Edición de objetos 56
2.7.5 Planificadores de agarre usando la BarrettHand 57
2.7.5.1 -PLANNER- 57
2.7.5.2 -EIGENGRASP- 59
2.7.6 La base de datos del proyecto 61
2.7.7 La base de datos de la Universidad de Columbia 61

CAPÍTULO 3 - DESARROLLO DEL PAQUETE DE AGARRE "GRASP" 65
3.1 INTRODUCCIÓN 66
3.2 DESARROLLO TEÓRICO 66
3.2.1 Cambio de posición relativa respecto a la orientación del objeto 69
3.2.2 Cambio de orientación relativa respecto a la orientación del objeto 71
3.3 DESARROLLO LÓGICO 72
3.4 CONDICIONES RESTRICTORAS 73
3.4.1 Distancia efectiva del IRB 120 74
3.4.2 Limitación de borde y radio mínimo efectivo 75
3.4.3 Replanteamientos al desplazamiento y tiempo de planificación 76
3.4.4 Limitación angular de seguridad 77
3.5 ARQUITECTURA EN ROS 77
3.5.1 Suscripción 77
3.5.2 Publicación 77

CAPÍTULO 4 - APLICACIONES 81
4.1 INTRODUCCIÓN 82
4.2 APLICACIÓN PARA CLASIFICACIÓN DE OBJETOS A NIVEL INDUSTRIAL 82
4.3 APLICACIÓN DE LABORATORIO ROBOTIZADO 86
4.3.1 Investigación biológica 86
4.3.2 Industria química 87
4.3.2.a Rutina opcional de la aplicación ROBLAB–Modo manual 90
4.3.2b Continuación del proceso de rutina 91

CAPÍTULO 5 - RESULTADOS Y CONCLUSIONES FINALES 97
5.1 RESULTADOS 98
5.2 CONCLUSIONES 100
5.3 TRABAJOS FUTUROS 100
5.3.1 Respecto al paquete de agarre 100
5.3.2 Respecto al sistema de aplicaciones completo 100

CAPÍTULO 6 - PRESUPUESTO 103

CAPÍTULO 7 - MANUAL DE USUARIO 105
7.1 Instalación de ROS HYDRO MEDUSA 105
7.2 Instalación de ROS openni 105
7.3 Instalación de MoveIt! 106
7.4 Instalación de GraspIt! 106
7.5 Instalación de las dependencias relacionadas con el control de la BarrettHand 106
7.6 ESTACIÓN VIRTUAL EN ROBOTSTUDIO (Recomendado) 107
7.7 INSTALACIÓN DEL SOCKET 108
7.8 CARGA DE LOS PAQUETES 109
7.9 LANZAR LAS APLICACIONES 110
7.9.1 APLICACIÓN DE CLASIFICACIÓN 110
7.9.2 APLICACIÓN DE ROBOT EN LABORATORIO 111

BIBLIOGRAFÍA 113

ÍNDICE DE FIGURAS

Figura 1.1: Disposición natural del entorno de trabajo estático 22
Figura 1.2: Brazo robótico espacial y cadena de montaje 22
Figura 1.3: Modelos de robot 23
Figura 1.4: Planta industrial robotizada 24
Figura 1.5: Tipos de grippers 25
Figura 1.6: Aplicación de precisión por ABB 26
Figura 1.7: Aplicación de clasificación por colores 27
Figura 1.8: Proyecto de planificación de agarre por Shimojo Laboratory 27
Figura 1.9: Robot humanoide cocinando 27
Figura 1.10: Matriz de transformación homogénea 28
Figura 1.11: Estructura del sistema 29
Figura 2.1: Logo oficial de ROS HYDRO MEDUSA 34
Figura 2.2: Estructura del paquete 35
Figura 2.3: Estructura del workspace 36
Figura 2.4: Ejemplo de interconexiones entre nodos 37
Figura 2.5: Suscripción y publicación entre nodos 37
Figura 2.6: Estructura del mensaje "geometry_msgs/Pose" 39
Figura 2.7: Ejemplo de comportamiento entre nodos 40
Figura 2.8: Relación entre cliente y servidor 40
Figura 2.9: Herramienta BarrettHand 42
Figura 2.10: Mecanismo TorqueSwitch 43
Figura 2.11: Mecanismo Joint-Torque Sensor 43
Figura 2.12: Composición de los dedos de BarrettHand 44
Figura 2.13: Límites rotacionales de BarrettHand 45
Figura 2.14: Interfaz gráfica de ROS para BarrettHand 45
Figura 2.15: Kinect para Windows 46
Figura 2.16: Componentes de Kinect 46
Figura 2.17: Nube de puntos primitiva 47
Figura 2.18: Disposición de los ejes y los motores del ABB IRB 120 49
Figura 2.19: Dimensiones del ABB IRB 120 49
Figura 2.20: Entorno de RobotStudio 51
Figura 2.21: Logo oficial de GraspIt! 52
Figura 2.22: Entorno de GraspIt! 52
Figura 2.23: Fichero world.xml de GraspIt! 54
Figura 2.24: Formato de la etiqueta <fullTransform> 55
Figura 2.25: Formato de la etiqueta <dofValues> 55
Figura 2.26: Fichero model.xml de GraspIt! 56
Figura 2.27: Computación y resolución al agarre mediante PLANNER 57
Figura 2.28: Configuración del criterio de calidad en PLANNER 58
Figura 2.29: Configuración general de PLANNER 59
Figura 2.30: Ejemplo de EIGENGRASP 60
Figura 2.31: Estructura de la información en la base de datos 61
Figura 2.32: Ejemplos de la base de datos de la Universidad de Columbia 62
Figura 3.1: Matriz identidad de rotación 66
Figura 3.2: Relación entre referencias de GraspIt! y ROS MoveIt! 66
Figura 3.3: Transformación matricial entre GraspIt! y ROS MoveIt! 67
Figura 3.4: Transformación espacial entre GraspIt! y ROS MoveIt! 67
Figura 3.5: Distancia entre robot y palma del gripper 68
Figura 3.6: Desplazamiento del efector final entre robot y herramienta 68
Figura 3.6: Ejemplo de desplazamiento frente a una referencia fija 69
Figura 3.7: Representación del cuaternio como matriz homogénea 69
Figura 3.8: Relación entre cuaternios y componentes Tait-Bryan 70
Figura 3.9: Ejemplo de rotación secuencial de planos 70
Figura 3.10: Matrices de rotación 70
Figura 3.11: Ejemplo de rotación frente a una referencia fija 71

Figura 3.12: Expresión de producto entre cuaternios 71
Figura 3.13: Esquema de funcionamiento del paquete de agarre 72
Figura 3.14: Soluciones de agarre simuladas 73
Figura 3.15: Rango efectivo del IRB 120 74
Figura 3.16: Rango delimitado de la zona de trabajo60 75
Figura 3.17: Ejemplos de solución aportados por el paquete para agarrar una esfera 79
Figura 4.1: Robots de clasificación industrial 82
Figura 4.2: Diagrama de la aplicación de clasificación 83
Figura 4.3: Solución al agarre del brick de leche 84
Figura 4.4: Solución al agarre de la lata de refresco 84
Figura 4.5: Solución al agarre de la pelota de tenis 85
Figura 4.6: Cartel homologado de riesgo biológico 86
Figura 4.7: Barreras de protección convencionales para experimentación química 88
Figura 4.9: Cartel homologado de riesgo radiactivo 88
Figura 4.9: Diagrama de la aplicación para el laboratorio robotizado 89
Figura 4.10: Controles para modo manual 90
Figura 4.11: Situación de reposo primaria 91
Figura 4.12: Sistema en espera, segunda situación de reposo 92
Figura 4.13: Detección del recipiente, aproximación 92
Figura 4.14: Agarre de máxima calidad al recipiente 92
Figura 4.15: Desplazamiento cartesiano hasta el vaso 93
Figura 4.16: Vertido del recipiente en el vaso 93
Figura 4.17: Fluido vertido, enderezar el recipiente 93
Figura 4.18: Desplazamiento cartesiano del recipiente a su situación inicial 94
Figura 4.19: Fin de la rutina del laboratorio 94
Figura 5.1: Orientación de soluciones aportadas para la clasificación respecto el eje Z 98
Figura 5.2: Tiempo empleado (%) Vs. Calidad de agarre 99
Figura 7.1: Activación de modo manual en FlexPendant 108

ÍNDICE DE TABLAS

Tabla 2.1: Formatos de mensaje _____ *38*
Tabla 2.2: Estructuras de mensaje _____ *38*
Tabla 2.3: Parámetros Denavit-Hartenberg _____ *50*
Tabla 4.1: Riesgos biológicos _____ *86*
Tabla 4.2: Riesgos químicos _____ *87*
Tabla 7.1: Configuración de las tareas del socket _____ *108*
Tabla 7.2: Configuración de los módulos del socket _____ *109*

GLOSARIO

DOF	Degrees Of Freedom
IRC5	Industrial Robotic Controller (v5)
PCL	Point Cloud Library
ROS	Robot Operating System
TCP	Transmission Control Protocol
TFG	Trabajo de Final de Grado

CAPÍTULO 1
INTRODUCCIÓN AL PROYECTO

1.1 DESARROLLO DE LA ROBÓTICA

El término robótica fue acuñado por primera vez en 1921 por Karel Capek, en Praga al estrenar su obra Rossum´s Universal Robot (R.U.R), del término *"robota"*, que del eslavo significa "trabajo realizado de manera forzada".

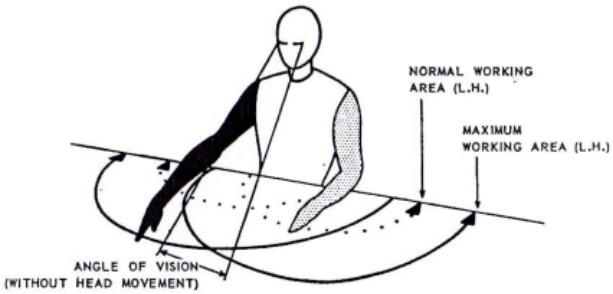


Figura 1.1: Disposición natural del entorno de trabajo estático

En los años 50 se propulsó de manera muy fuerte el término de operario mecánico: En 1948 *George Devol* patentó un manipulador programable, empresas como *R.C. Goertz* del *Argonne National Laboratory* trabajaron para desarrollar el primer operador tele operado, implementando más tarde sistemas de servocontrol (1954), *General Electric* reveló su dispositivo *Handy-Man* (1958), formado por dos brazos mecánicos teleoperados.

Dado a la diversidad de aplicaciones y a consideraciones éticas surtidas en el apogeo de la robótica, el escritor *Issac Asimov* determinó "las tres leyes de la robótica", que exponían:

1. *Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.*
2. *Un robot debe de obedecer las órdenes dadas por los seres humanos, salvo que estén en conflicto con la primera ley.*
3. *Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.*

En los años setenta la industria nuclear, submarina y el interés en la carrera espacial aumentaron el interés por este tipo de tecnología robot.

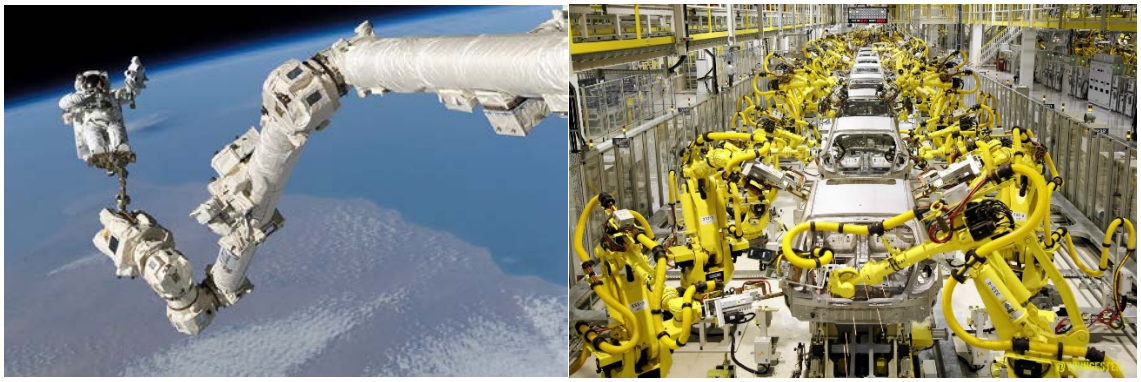


Figura 1.2: Brazo robótico espacial y cadena de montaje

En este punto, el robot manipulador se especializó en la industria, puesto a su elevado tamaño y coste, no podía ofrecerse como un producto comercial. Debido a las condiciones a las que tenía que ser cargado el robot, o de la situación por la cual tenía que operar, este adquirió diferentes configuraciones, pero siempre mantuvo su propósito inicial: un operador mecánico tele operado capaz de realizar trabajo con carga de manera continuada.

Otras aplicaciones de la robótica, fuera del apartado de potencia, han sido los usados de manera remota como herramienta de operación móvil. En este campo la diversidad es plena: robots para aplicaciones domóticas, destinados al apartado militar en reconocimiento del terreno, como herramienta de apoyo o simplemente para emular la esencia humana.

Barredora Roomba **Humanoide ASIMO** **Robot militar BigDog**

Figura 1.3: Modelos de robot

1.2 LA ROBÓTICA EN LA INDUSTRIA

El mayor avance de la robótica fue el poder implementar un ordenador para que controlara los movimientos del manipulador. Los laboratorios de investigación se dedicaron en la década de los 60 a construir equipos específicamente destinados al control de la herramienta robot. Japón le tomó la delantera a Estados Unidos mediante la empresa *Nissan*, que formó la primera asociación robótica del mundo.

El robot industrial se caracteriza por tener un modelo de brazo mecánico, complicándose por cada grado de libertad añadido a las articulaciones, y por la capacidad de poder adherir una herramienta o un aferrador a su punto terminal. También la capacidad de poder implementar varios programas en su base de funciones, o de poder trabajar en distintos entornos informáticos. Esto hace al robot más robusto y flexible respecto a los cambios y mejoras posteriores.

Las cinco fases más relevantes en el desarrollo industrial [4] se enuncian a continuación:

1. El laboratorio *ARGONNE* diseña manipuladores amo-esclavo para manejar material radioactivo (1950).
2. *Unimation* expande el negocio de fabricación de automóviles con robots de campo (1961-1968).

3. Los laboratorios de la *Universidad de Stanford* y del *MIT* trabajan sobre la idea del robot controlado por ordenador de manera autónoma (1970).
4. Se implementa el microcontrolador como alternativa al terminal para el robot industrial (1975).
5. A partir de 1980 el robot industrial es usado en investigación, como un conjunto de aplicaciones de sensores y efector, y se desarrolla la idea de robot inteligente, capaz de trabajar en tiempo real y capaz de reaccionar con su entorno. En 1980, gracias a la revolución de la microelectrónica, el parque de robots produjo un aumento acumulativo del 25%.

Desde estas últimas fechas clave, el robot industrial ha ido paulatinamente formando parte de la vida de la empresa, tanto en grandes como en pequeñas organizaciones: optimizando procesos recurrentes, otorgando seguridad al trabajador ante tareas hostiles y aumentando la producción de la empresa.

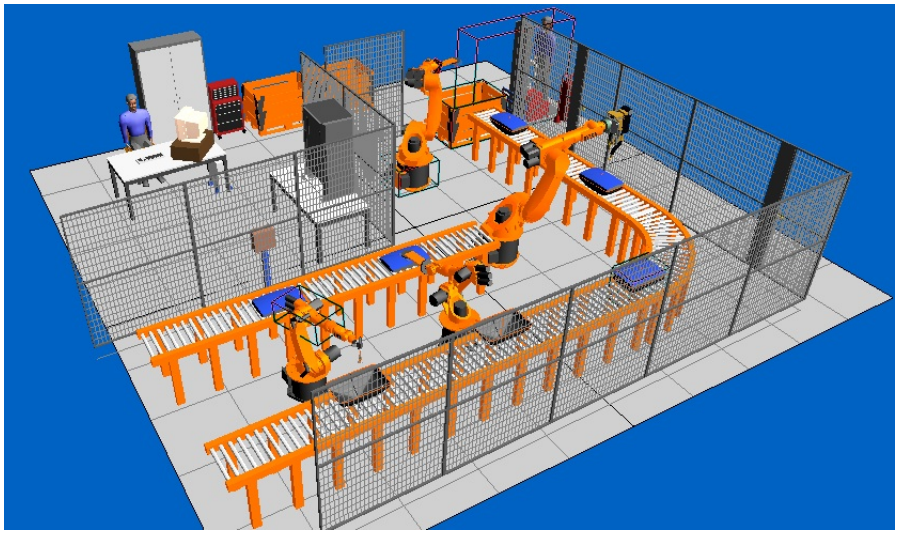


Figura 1.4: Planta industrial robotizada

1.3 GRIPPERS

El robot industrial puede estar destinado a múltiples tareas, aunque las tareas más comunes se subdividen en dos grupos:

-Usar herramientas con precisión: como pinzas de soldadura por puntos, sierras, taladros, pulidoras… Permiten realizar una tarea con una alta precisión debido al control informático.

-Servir como pinza: existen tareas por las cuales es necesario elevar un peso lo suficientemente alto como para que sea necesario utilizar maquinaria para alzarlo, o es necesaria una velocidad de respuesta para agarrar tan alta que no puede realizarse mediante operarios humanos. La implementación de una pinza un brazo robot permite agarrar y trasladar elementos con alta precisión de manera segura y automatizada, o en otras circunstancias, aferrar y distribuir elementos a alta velocidad con efectividad.

Ampliando este último concepto, existe un importante campo de desarrollo dedicado al elemento terminal de la herramienta que permite adherir los elementos de campo al robot. Estas herramientas se denominan "*grippers*" o pinzas: hay grippers específicos para las diferentes aplicaciones de agarre, y su efectividad es evaluada siempre respecto a las condiciones de calidad con las que se quiere agarrar el elemento en cuestión.

Existe una amplia gama de grippers según su tipo de accionamiento, ya sea neumático, eléctrico, magnético, etc; y su forma morfológica propia: grados de libertad del gripper, número de terminaciones prénsiles, máximo peso a agarrar, volumen medio por agarre, etc.

En las siguientes imágenes se muestran varios tipos de grippers industriales.

BarrettHand eléctrica

Ventosa neumática

Pinza simple neumática

Pinza "coffee filled" neumática

Pinza magnética

Figura 1.5: Tipos de grippers

Existe una serie de características que acuñan casi en su totalidad la mayoría de grippers:

-<u>Independientes al sistema al que se acoplan:</u> Todos los elementos que lo forman e intervienen para su funcionamiento son independientes al robot principal, por lo que son altamente compatibles e intercambiables entre aplicaciones y sistemas. Usualmente la base del gripper está montada sobre una platina metálica a 1 o 4 puntos de tornillería, según su peso y dimensiones.

-<u>Su estructura terminal puede modificarse</u>: usualmente el gripper comercial sale al mercado con sus características morfológicas más básicas. Un proceso habitual es

añadir modificaciones terminales a los grippers, de tal forma que mejoren su rendimiento: se caracteriza un producto global para una aplicación definida.

-Amplia gama de tamaños y precios: puesto que los grippers se utilizan como herramienta terminal tanto en aplicaciones industriales como en la robótica a nivel de usuario, el catálogo para encontrar el gripper adecuado es muy extenso. Esto permite ajustar en primera instancia el gasto económico respecto a la situación de trabajo, y en segundo lugar reducen la cantidad de modificaciones a posteriori sobre el gripper para ajustarse a las características de la aplicación.

1.4 ANTECEDENTES AL PROYECTO

Este proyecto ha sido concebido para establecerse como una extensión a las rutinas de planificación para planificadores robóticos. En la gran mayoría de los planificadores no se toma en cuenta la necesidad de plantear soluciones para robots mientras portan una herramienta. Esto es debido, en la mayoría de las veces, a que el brazo robot en sí es una máquina usualmente estática, asociada a realizar un trabajo concreto, en un tiempo determinado. Esta serie de factores restrictores acotan las variables del entorno que afectan al desempeño de la tarea del robot, por lo que sencillamente, basta con programar en su rutina una única solución de trabajo, dado que si está bien definida, bastará para que el robot pueda desempeñar su tarea durante toda su vida útil sin cambios. Esto es propio de cadenas industriales modernizadas mediante robótica.

Un ejemplo a este comentario se puede observar en uno de los videos promocionales de ABB [5], donde un operario mueve de forma manual un robot con herramienta entre una serie de latas, de manera estática, para luego repetir el proceso, de forma dinámica.

Figura 1.6: Aplicación de precisión por ABB

Pero, en aplicaciones donde los robots deben interactuar con el entorno, usualmente desconocido y con variables aleatorias, es necesario de implementar una lógica o inteligencia artificial que pueda valorar estos parámetros y poder responder de forma hábil. Para que el robot pueda procesar todos estos datos, es necesario una gran cantidad de rutinas de acción y dependencias a otros sistemas, tales como visión artificial, sensores táctiles, de temperatura o proximidad.

Dado que la tarea más sencilla que puede realizar un robot respecto a su entorno es desplazar un objeto de este, la gran cantidad de aplicaciones de inteligencia artificial para planificadores de agarre utilizan como herramienta terminal una mano robótica.

Estas manos robóticas, que constan al menos de 3 dedos articulados y posibilidad de giro respecto la palma, son adecuadas para la mayoría de aplicaciones de agarre en un entorno cotidiano, debido a su semejanza con la mano antropomórfica. Así pues, el desarrollo de este apartado ha sido mejorado mediante aplicaciones como la calificación de objetos, brazos robot que interactúan con humanos en un mismo espacio de trabajo, o androides [6].

Figura 1.7: Aplicación de clasificación por colores

Figura 1.8: Proyecto de planificación de agarre por Shimojo Laboratory

Figura 1.9: Robot humanoide cocinando

Es manifiesto que para cada aplicación, la lógica asociada al procedimiento de los agarrares completamente distinta. El entorno GraspIt! ha sido un referente para la planificación de agarres desde el año 2002, pero debido a la antigüedad del proyecto y

a que a partir de ROS HYDRO MEDUSA los paquetes del entorno ROS migraron a formato CATKIN, GraspIt! ha dejado de ser compatible dentro del entorno.

Abstrayéndose a la finalidad de la aplicación, es necesario agrupar un planificador de agarres y una lógica que los pueda establecer, cuyo objetivo justifica la finalidad de este proyecto y que a continuación se aborda con todo detalle.

1.5 TEORÍA DE LA ROBÓTICA Y VISIÓN GENERAL

Hablar de robótica es relacionar el carácter propio de la matemática y el álgebra con el control de señales y la capacidad de abordar una aplicación real mediante un sistema mecánico.

El principal objetivo, desde el punto de vista matemático, es la necesidad de poder relacionar la situación espacial del efector de un robot (su punto terminal útil) con la situación espacial de su base (su punto de anclaje). De esta forma se consiguen dos puntos clave [7]:

-Caracterizar el sistema, haciendo especial hincapié en sus grados de libertad y tipos de efectores.

-Permitir la aplicación de técnicas de control cinemático y control dinámico.

De manera habitual, se usan matrices de transformación homogénea, esto es, matrices de tamaño 4x4 que representan la transformación de coordenadas que realiza un sistema al cambiar su referencia respecto a otro sistema, a modo de planta de sistema.

Estas matrices agrupan dos apartados: rotación entre el punto de referencia y el punto transformado; y translación entre el punto de referencia y el punto transformado.

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{p}_{3\times1} \\ \mathbf{f}_{1\times3} & \mathbf{w}_{1\times1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix}$$

Figura 1.10: Matriz de transformación homogénea

En robótica, la submatriz f, que representa una transformación de perspectiva, es nula, y la submatriz w, que representa un escalado global, es la unidad. Esta transformación matricial consigue relacionar la orientación y la posición de un sistema OUVW rotado y trasladado con respecto al sistema de referencia OXYZ.

Con esta misma lógica, conociendo los parámetros matriciales del gripper se puede referenciar el punto final del sistema (conocido como TCP) una vez acoplado y programarse mediante computación. De esta forma, independientemente a la herramienta que se coloque en el punto final del brazo robot, podemos obtener la posición y la orientación respecto al origen de coordenadas de la base.

La misión de los algoritmos de agarre o “graspit” es relacionar la situación espacial del efector de la herramienta con la situación del objeto a aferrar, de tal forma que el programa otorgue soluciones válidas para desplazar el efector del robot hasta la situación de agarre deseada. Esta situación dependerá de los requerimientos lógicos del agarre, como la condición de una mayor calidad en el agarre, o una mayor velocidad para resolverlo.

Dado que cada pinza tiene su morfología propia, la forma en la que se situará espacialmente para trabajar será distinta en cada caso. Es importante destacar que dado que la información sobre estos parámetros puede ser almacenada y referenciada siempre al extremo final del brazo robot, el proceso por el cual se cambia de “gripper” entre aplicaciones puede ser computado y actualizado de forma trivial por el sistema de inteligencia del robot, lo que confiere flexibilidad y robustez al hecho de poder usar varias herramientas de agarre para una misma aplicación, según se requiera.

En este proyecto se hará especial énfasis sobre el gripper BarrettHand, debido a su estructura parecida a la mano antropomórfica y a la orientación de las aplicaciones propuestas por el proyecto.

Es importante matizar que aunque el software ha sido especialmente diseñado para esta herramienta de agarre, el programa permite ofrecer soluciones de agarre para cualquier herramienta de agarre que se le acople al robot. Más adelante se explican los conceptos asociados a la base de datos de agarres ofrecidos por GraspIt! y la transformación matricial correspondiente a la herramienta BarrettHand.

Este concepto ha sido la idea principal desde la que se ha partido para desarrollar la idea de proyecto. Este concepto se explica con detalle en el aparatado referente a la información del paquete de agarre.

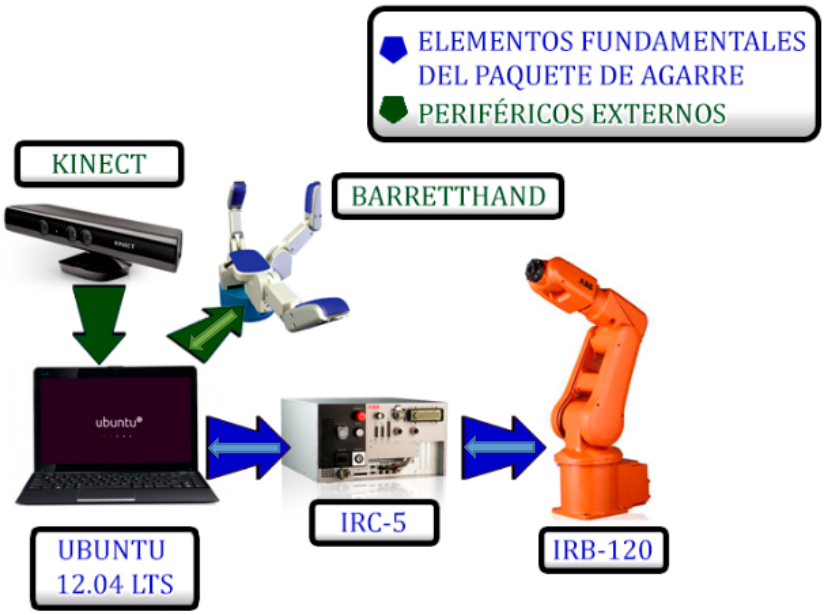


Figura 1.11: Estructura del sistema

Se pretende que el sistema esté compuesto por:

-Un **módulo de visión artificial**, que permita reconocer figuras dentro de la zona de trabajo para poder ser analizadas de manera individual. Esta información permitirá al robot reconocer el espacio real y poder evaluar cómo debe interaccionar con él.

- Se utiliza la cámara **Kinect**.

-Un **robot** que pueda servir a modo de efector, capaz de desplazarse con precisión y con la capacidad de tener una variedad de poses articulares para poder generar la situación de agarre adecuada.

- Se utiliza el robot **ABB IRB 120**.

-Un **controlador del robot**, que pueda interpretar las órdenes de desplazamiento desde un socket de información, capaz de trabajar con el protocolo de ROS.

- Se utiliza el controlador **IRC5**.

-Un **paquete de agarre**, que permita otorgar soluciones mediante los algoritmos implementados en GraspIt!, y que otorgue la mejor solución ante la información recibida por el módulo de visión artificial.

- Se utilizan los algoritmos de **GraspIt!**

-Una **herramienta** capaz de realizar el agarre, y a su vez, que pueda ofrecer soluciones variadas y que tenga la capacidad de ser compatible con GraspIt!.

- Se utiliza el gripper comercial **BarrettHand.**

-Un **sistema compatible a ROS** para poder procesar la información entrante del sistema de visión, del paquete de agarre y que en última instancia genere órdenes para efectuar un desplazamiento sobre el robot y un control sobre la herramienta de agarre.

- Se utiliza **Ubuntu 12.04 LTS**.

A continuación se analizarán por separado los componentes del sistema con mayor profundidad.

CAPÍTULO 2

ARQUITECTURA GENERAL DEL SISTEMA

2.1 ROS

2.1.1 Introducción

El entorno software con el que se trabaja en este proyecto es ROS HYDRO MEDUSA. ROS es una plataforma open-source, que implementa servicios propios de un sistema operativo, tales como abstracción de hardware, bajo nivel de programación, envío de mensajes entre procesos o sistema de paquetes. Diseñado para trabajar en el entorno de Linux, permite una alta compatibilidad para trabajar entre equipos informáticos. El lenguaje con el que se puede programar en ROS es C++ y Python.

Figura 2.1: Logo oficial de ROS HYDRO MEDUSA

ROS como plataforma de programación robótica tiene una larga serie de ventajas:

-Permite la multitarea entre un único o varios equipos informáticos controlando un robot, mediante el uso de conceptos simplificados como “nodos” y “mensajes”.

-El sistema de paquetes unificado hace que la compilación y la compatibilidad de los programas sea mayor, y permite al usuario centrarse en la programación de las directrices del robot en sí y no en los parámetros intermedios de configuración software.

-Dado a que el software es abierto, permite la incorporación de software adicional, tal como el caso de simuladores, que agilizan el proceso en la prueba y error al desarrollar el código.

-La comunidad robótica abala a ROS como un software robusto para trabajar, y dado que es ampliamente usado, la comunidad de usuarios que la forma y mejora la plataforma es enorme, lo que se traduce directamente en mayor cantidad de soluciones y apoyo al usuario.

ROS como cualquier otro software necesita de aprendizaje inicial [8]: no es una mera interfaz. Personalmente, resultó de gran ayuda en el desarrollo de este proyecto el trabajo ***“A Gentle Introduction to ROS”***, de *Jason M. O´Kane* [9], que aborda en su trabajo todo lo necesario para aprender ROS desde cero hasta conceptos avanzados. En este trabajo se aborda con gran extensión los conceptos principales y únicos de ROS.

A continuación se hace una explicación a los más importantes:

2.1.2 Paquetes

La distribución del software en ROS se realiza en modo de paquetes. Es importante matizar que al tratarse de la versión ROS HYDRO se utiliza el formato de paquetes CATKIN, actualmente la versión actual. Estos paquetes no tienen compatibilidad con ROS FUERTE o anteriores versiones debido a que se cambió el estándar de paquetes.

Un paquete CATKIN es una carpeta de archivos que contiene:

-Un programa en **C++** *(.cpp)* o **Python** *(.py)*, que serán las instrucciones a implementar por el robot. A veces este programa se incorpora dentro de una carpeta interna *"source" (src)* a modo de organizar los estratos del paquete (opcional).

-Un descriptor de construcción **CMakeLists.txt**, que incorpora todas las dependencias y direcciones de los recursos necesarios para crear el paquete.

-Un descriptor de paquete **package.xml**, que funciona como identificador de este.

-Carpetas de inclusión de archivos adicionales: se admiten ficheros de texto, imágenes o mapas de bits.

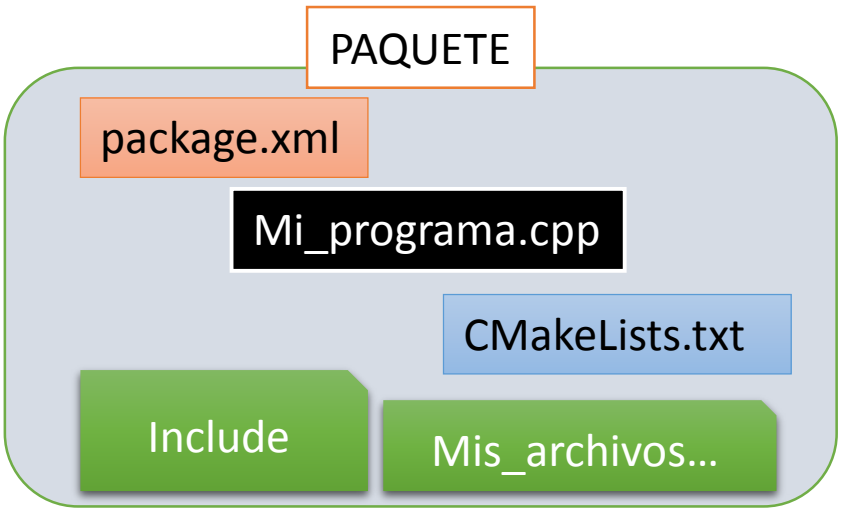


Figura 2.2: Estructura del paquete

Con estos componentes ROS puede convertir el paquete en un ejecutable, que puede ser ejecutado en la terminal.

Para realizar un paquete primero es necesario disponer en Ubuntu de una carpeta de trabajo (*Workspace*). En el manual de usuario se adjuntan las acciones realizadas para configurarlo en el equipo. En esencia, la carpeta de trabajo está formada de una serie de carpetas estructuradas por ROS en las cuales se ordenan y procesan los paquetes para su compilación y ejecución. El nombre del *Workspace* de este proyecto se llama "*ros_catkin_ws*". En este directorio se crearan (de forma manual o automática) las siguientes cuatro carpetas.

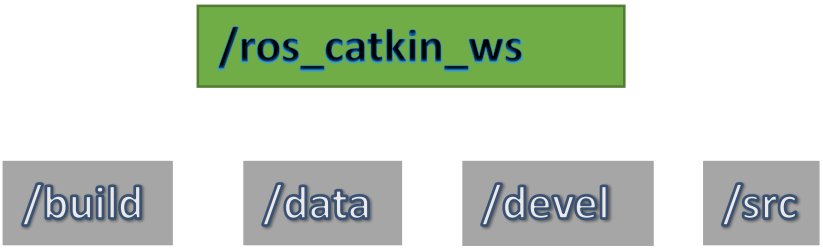


Figura 2.3: Estructura del workspace

Una vez creado el directorio raíz, se puede proceder a crear un paquete. Para ello existen funciones propias en ROS para crearlo de manera automática, o se puede directamente copiar un paquete ya creado y editar sus componentes de forma manual. Este paquete debe ser alojado dentro del subdirectorio *"/[ros_catkin_ws]/src",* indicando que es una "fuente*"(source)* por la cual ROS tiene que comprobar y compilar si es necesario.

Una vez alojado en el directorio, se puede proceder a construir los ejecutables de los paquetes, situando el terminal de Ubuntu en la dirección de la *[ros_catkin_ws]* y ejecutando el comando de construcción **"catkin_make".**

Si todas las dependencias son correctas, si no hay fallos por omisión de archivos adjuntos y si no hay errores de escritura en el código fuente del programa .cpp /.py, el paquete realiza *"link"* y se crea el ejecutable. Este ejecutable puede luego lanzarse en una terminal, previa referencia al workspace de origen mediante **"source devel/setup.bash".**

De esta forma se habría conseguido ejecutar un paquete en el entorno de ROS.

Todas las sentencias de terminal anteriormente citadas con orientativas, en el manual de usuario se exponen paso por paso.

Importante: *hay que asegurar que todas las dependencias necesarias mediante la compilación están instaladas previamente en el equipo. En ciertos equipos usados durante la elaboración de este proyecto, se necesitaron accesos de superusuario ("sudo su") para poder realizar la compilación de forma satisfactoria. Actué con precaución respecto a esta última instancia*.

2.1.3 Nodos

Una vez que se ha construido el paquete se puede ejecutar. La forma que tiene ROS para gestionar los paquetes que se ejecutan es mediante la propia terminal de Ubuntu. Existen unos encabezados preestablecidos que hay que respetar al lanzar cada aplicación para que ROS reconozca que es un paquete suyo y que es su deber gestionarlo. Para ello siempre que se quiera ejecutar un paquete en ROS es necesario llamar previamente a la instrucción **"roscore**". Lanzar en un terminal esta instrucción ejecutará el nodo maestro de ROS. Este nodo permite la interconexión de los demás nodos que se lancen a continuación, y reserva en el sistema una cantidad de memoria limitada para su ejecución.

Un nodo es una instancia en ejecución de un paquete en ROS. Cuando se ejecuta un paquete, crea un nodo. Es importante recalcar que, si un paquete se ejecuta más de una vez mientras hay alguno en ejecución, la repercusión es que aparecen varios nodos. Cada nodo tiene su identificador, por lo que no se confunden unos ante otros.

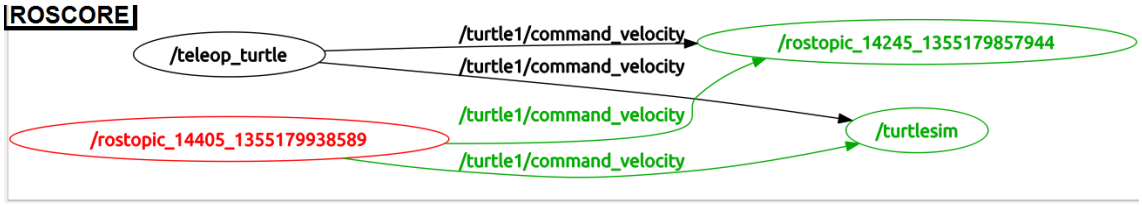


Figura 2.4: Ejemplo de interconexiones entre nodos

En este punto, un nodo es equivalente a un hilo de programa ejecutándose en paralelo, y en esencia funciona igual: Un nodo puede estar en ejecución, a la espera de otro nodo, o a la espera de iniciarse.

Se pueden tener tantos nodos como se quiera, teniendo en cuenta las limitaciones del equipo informático.

2.1.4 Mensajes

Teniendo varias aplicaciones funcionando en paralelo, es necesario que se comuniquen entre ellas. Para ello, se utilizan mensajes. Un mensaje es una información que puede transitar entre nodos. A estos mensajes se puede acceder a ellos mediante “topics” (*asuntos*), a los que los nodos pueden “suscribirse” o que pueden “publicar”.

Ejemplo básico entre un nodo publicador (teleop_turtle) y un nodo suscriptor (turtlesim).

***Rosout* es un nodo genérico de ROS que sirve como reporte al sistema de los nodos en ejecución.**

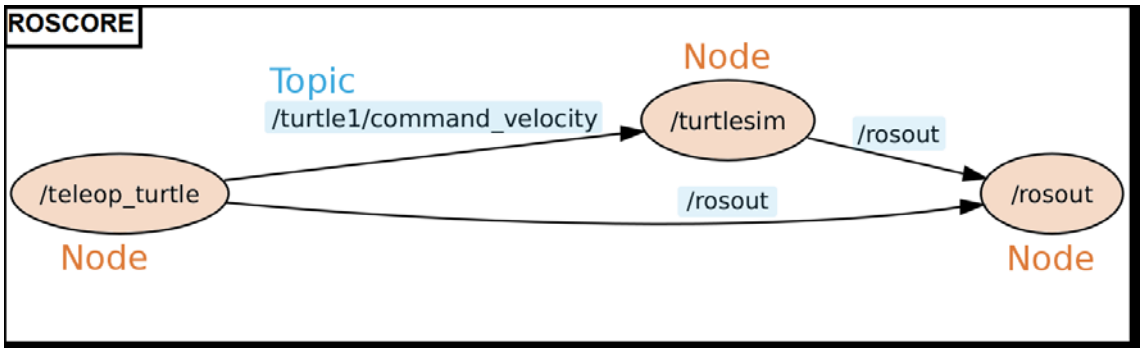


Figura 2.5: Suscripción y publicación entre nodos

Publicando: un nodo cualquiera puede publicar mensajes en el entorno de ROS. Estos mensajes deben tener una estructura conocida, es decir, deben conocerse en el entorno de ROS. Un ejemplo, la estructura del tipo de mensaje "geometry_msgs/Pose.msg". Así pues, un "topic" puede tener esta estructura.

Los formatos de mensaje más comunes acuñados por defecto en ROS son:

common_msgs	
actionlib_msgs	GoalID GoalStatus GoalStatusArray
diagnostic_msgs	DiagnosticArray DiagnosticStatus KeyValue
geometry_msgs	Accel AccelStamped AccelWithCovariance AccelWithCovarianceStamped Inertia InertiaStamped Point Point32 PointStamped Polygon PolygonStamped Pose Pose2D PoseArray PoseStamped PoseWithCovariance PoseWithCovarianceStamped Quaternion QuaternionStamped Transform TransformStamped Twist TwistStamped TwistWithCovariance TwistWithCovarianceStamped Vector3 Vector3Stamped Wrench WrenchStamped
nav_msgs	GridCells MapMetaData OccupancyGrid Odometry Path
sensor_msgs	CameraInfo ChannelFloat32 CompressedImage FluidPressure Illuminance Imag e Imu JointState Joy JoyFeedback JoyFeedbackArray LaserEcho LaserScan MagneticField MultiDOFJointState MultiEchoLaserScan NavSatFix NavSatStatus PointCloud PointCloud2 PointField Range RegionOfInterest RelativeHumidity Temperature TimeReference

Tabla 2.1: Formatos de mensaje

Las estructuras de dato que pueden tener estos formatos son:

Built-in types	
bool	Entero a 8 bits sin signo, booleano
int8	Entero a 8 bits con signo
uint8	Entero a 8 bits sin signo, orientado a Python
int16	Entero a 16 bits con signo
uint16	Entero a 16 bits sin signo
int64	Entero a 64 bits con signo
uint64	Entero a 64 bits sin signo
float32	Decimal a 32 bits, formato standard IEEE
float64	Decimal a 64 bits, formato standard IEEE
string	Cadena ASCII
time	Doble instancia a entero a 32 bits para segundos/ nanosegundos
duration	Doble instancia a entero de 32 bits para segundos /nanosegundos

Tabla 2.2: Estructuras de mensaje

Haciendo hincapié en el tipo de mensaje *Pose* de la estructura *geometry_msgs*:

Esta estructura de mensaje, muy usada en el proyecto para representar la posición y la orientación de un objeto, consta de los siguientes campos:

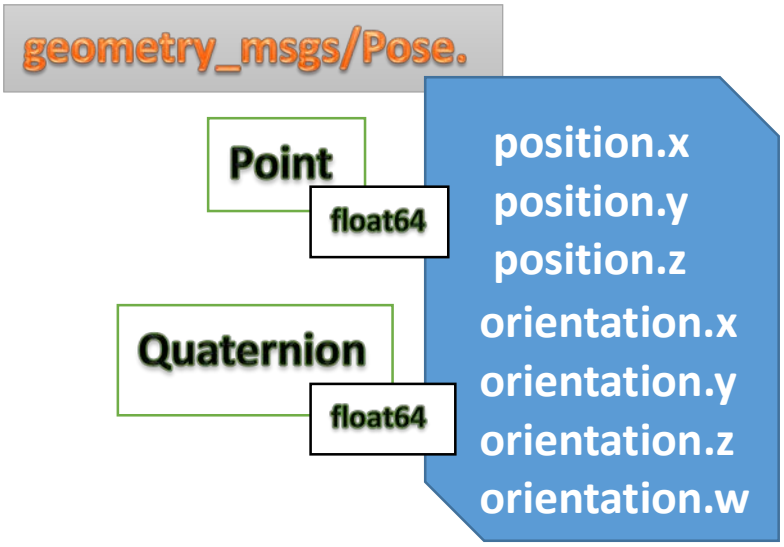


Figura 2.6: Estructura del mensaje "geometry_msgs/Pose"

El campo descriptor "Point" tiene 3 campos relativos a posición llamados "*position.[]",* con formato numérico float64 *(double float).*

El campo descriptor "Quaternion" tiene 4 campos relativos a la orientación llamados "*orientation.[]",* también a float64.

Cuando un nodo publica un mensaje, este aparece como un "topic" en el entorno de ROS. Este "topic" tiene un nombre conocido, establecido por el nodo que publica.

Ejemplo: *Un nodo publica la situación de un robot mediante un tipo de mensaje geometry_msgs/Pose.msg con un topic llamado "Robot". De esta forma, se podría publicar en ROS un mensaje con la posición y la orientación del robot.*

Una forma de ver los resultados de lo que publica el nodo seria mediante la instrucción "echo". En este caso particular:

rostopic list

rostopic echo /Robot

Remitiría continuamente los valores de posición y orientación publicados por el nodo.

Se pueden crear estructuras de mensaje propias. Estas deben de ser instanciadas dentro del paquete para que el compilador pueda evaluarlas.

Suscribiendo: Publicar ha permitido extraer información de un nodo hacia el entorno, si es necesario extraer información desde el entorno a un nodo, este ha de suscribirse. Un nodo puede estar interesado en recibir información de otro nodo. Para ello, el nodo se suscribe a un "topic" concreto, de tal forma que, cada vez que se publique algo nuevo en ese "topic" concreto, el nodo suscriptor realice una operación. Esto permite extraer información del entorno hacia el nodo.

Ejemplo: *Un controlador está interesado en la posición y la orientación del Robot del ejemplo anterior, por lo que se suscribe al topic "Robot". Cada vez que el robot publica su situación tridimensional, dentro del nodo del controlador se ejecuta un algoritmo que calcula si va a golpearse con su entorno.*

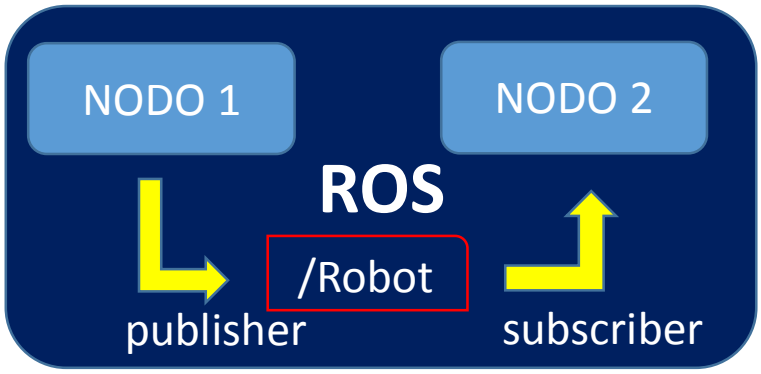


Figura 2.7: Ejemplo de comportamiento entre nodos

Un "topic" puede ser suscrito por varios nodos, y a su vez un nodo puede publicar y suscribirse a la vez.

De esta forma se es capaz de conectar múltiples paquetes entre sí. Realizando una similitud directa con la programación en C, el nodo ROSCORE sería la base MAIN del programa, mientras que los NODOS ejecutados serían las funciones del programa, que pueden relacionarse entre sí mediante mensajes con sus correspondientes argumentos.

Esto permite una flexibilidad de programación muy alta, que además permite a varios equipos informáticos suscribirse a la información publicada o poder publicar de manera global.

2.1.5 Servicios

Una instrucción especial de mensaje entre nodos es el de tipo servicio. Servicio se define como un protocolo de pregunta y respuesta entre nodos. A diferencia de los mensajes, los servicios tienen carácter bidireccional, y se tramitan como una comunicación "uno a uno": un nodo toma el rol de cliente y envía una petición a un nodo servidor. Mientras el nodo servidor contesta, el cliente se queda a la espera, hasta que el servidor responde y se cierra la llamada del servicio.

Figura 2.8: Relación entre cliente y servidor

Los servicios son comúnmente usados como interrupciones o peticiones puntuales de comunicación. Una cualidad de los servicios es que pueden ser llamados desde el propio programa o por el usuario en la terminal, lo que permite la interacción directa con el programa y hace al entorno más accesible a nivel de usuario.

El lenguaje usado para el desarrollo de este proyecto ha sido C++.

Para el desarrollo del paquete de agarre, se ha diseñado un paquete capaz de publicar y suscribirse a varios mensajes, capaz de abrir bases de datos en ficheros .txt previamente almacenados dentro del paquete.

Todos los mensajes usados por los nodos de este proyecto pertenecen a las librerías básicas de ROS.

2.2 GRIPPER TIPO BARRETTHAND

2.2.1 Introducción

La herramienta BarrettHand es un gripper comercial [10] que emula a una mano artificial humana a 3 dedos. Esta mano tiene 4 grados de libertad: 3 para cada dedo y uno para el movimiento simétrico angular de estos respecto a la palma de la mano.

Figura 2.9: Herramienta BarrettHand

Es usada en investigación dado a que, a diferencia de grippers de pinza, permite un mejor agarre de objetos esféricos y cilíndricos. Funciona de manera totalmente eléctrica e incorpora la tecnología TorqueSwitch, que permite plegar las falanges superiores de los dedos cuando las falanges inferiores son bloqueadas.

2.2.2 Características

Peso: 1,2 Kg

Dimensiones: 200x200x300 mm

Número de dedos: 3

Motores por dedo: 1

Articulaciones por dedo: 2

Carga: 6 Kg (2Kg por dedo)

Transformador: 230 AC to 24 DC

2.2.3 Estructura y parámetros mecánicos

2.2.3.1 Mecanismo TorqueSwitch

El mecanismo funciona como un tornillo sin fin que evalúa el par de giro que tiene que aportar el robot para cerrar los dados de manera individual. Cuando una falange queda bloqueada, el bloque de engranajes engrana con la siguiente falange, de forma secuencial, hasta que todas las falanges quedan bloqueadas dado a que su contacto con el objeto es máximo. El sistema mantiene el par de agarre durante toda la acción, y lo ofrece de forma indefinida hasta que se mande a la mano abrir el dedo.

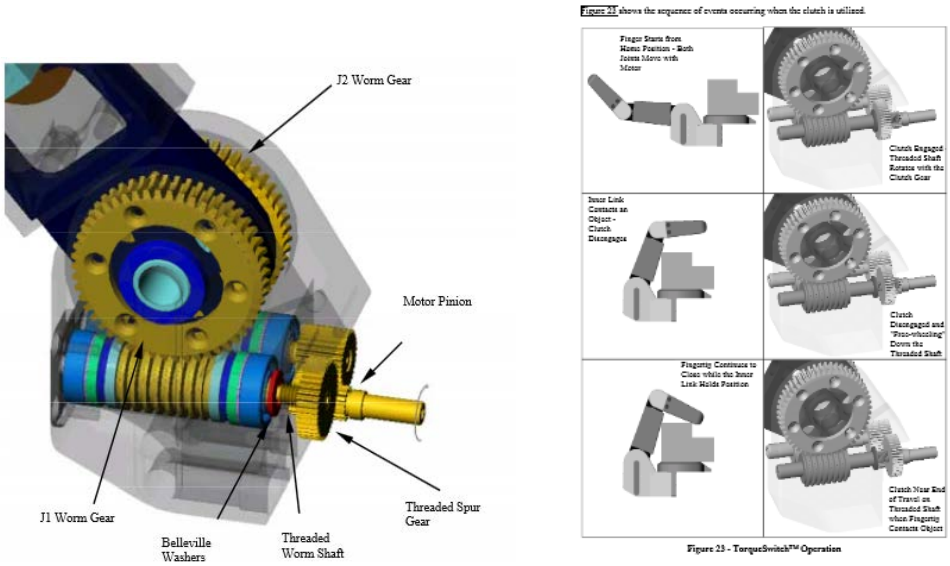


Figura 2.10: Mecanismo TorqueSwitch

2.2.3.2 Mecanismo Joint-Torque Sensor

Evalua la energía puesta en juego en cada dedo al generar par para realizar un agarre. Utiliza un puente de Wheatstone para evaluar la diferencia real y la impuesta, y esta diferencia es amplificada y convertida digitalmente para proveerla al usuario.

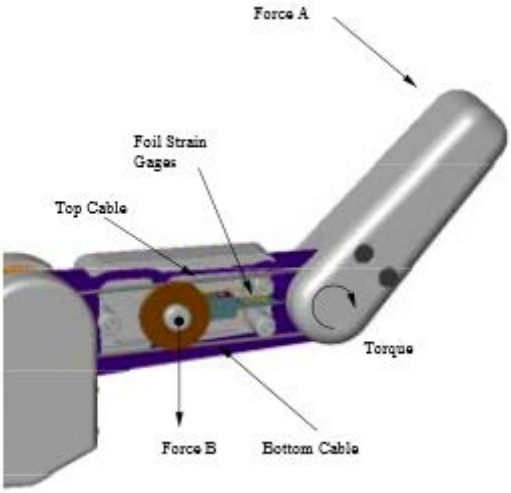
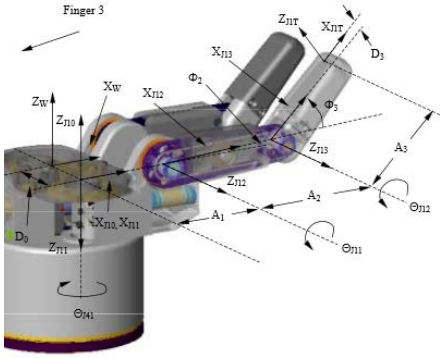


Figura 2.11: Mecanismo Joint-Torque Sensor

2.2.3.3 Cinemática de la BarrettHand

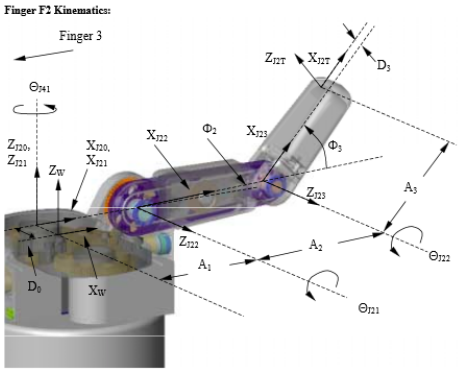
Este punto puede ser contemplado con mayor profundidad en el manual de usuario oficial de BarrettHand. Se expone aquí un resumen con las principales fórmulas y casos particulares de los dedos.

Articulación F1



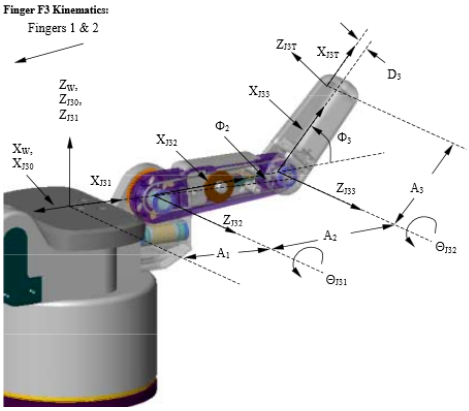
Joint	a_{k-1}	α_{k-1}	d_k	θ_k
1	0	π	0	Θ_{J41}
2	A_1	$-\pi/2$	0	$\Theta_{J11}+\Phi_2$
3	A_2	0	0	$\Theta_{J12}+\Phi_3$
T	A_3	$-\pi/2$	D_3	0

Articulación F2



Joint	a_{k-1}	α_{k-1}	d_k	θ_k
1	0	0	0	Θ_{J41}
2	A_1	$\pi/2$	0	$\Theta_{J21}+\Phi_2$
3	A_2	0	0	$\Theta_{J22}+\Phi_3$
T	A_3	$-\pi/2$	D_3	0

Articulación F3



Joint	a_{k-1}	α_{k-1}	d_k	θ_k
1	0	0	0	π
2	A_1	$\pi/2$	0	$\Theta_{J31}+\Phi_2$
3	A_2	0	0	$\Theta_{J32}+\Phi_3$
T	A_3	$-\pi/2$	D_3	0

Figura 2.12: Composición de los dedos de BarrettHand

Limitaciones

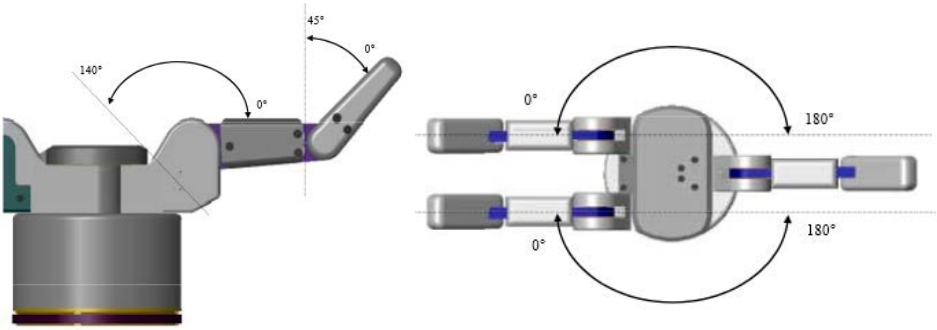


Figura 2.13: Límites rotacionales de BarrettHand

2.2.3.4 Nodo de control

Para controlar el gripper se utiliza el entorno de ROS, que permite el uso de nodos y mensajes, muy útiles en la cuestión a este tipo de actuadores. De esta forma los programas escritos como paquetes de ROS pueden usar la BarrettHand sin distinción. ROS cuenta con su propio paquete de control de BarrettHand, muy recomendable para realizar ensayos y probar el robot.

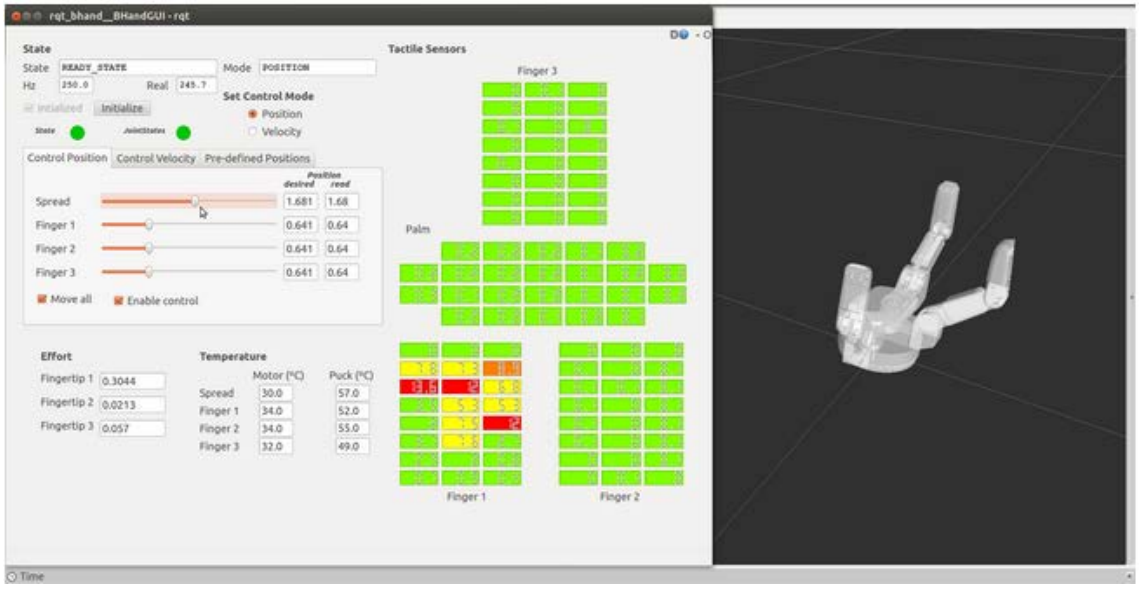


Figura 2.14: Interfaz gráfica de ROS para BarrettHand

Este proyecto se usa el paquete "**bhand**" de control para BarrettHand diseñado por Alejandro Gómez Rubio en su trabajo "Control de la mano robótica BarrettHand BH8-262 en entorno ROS" [1].

De esta forma, mediante el paquete de agarre, se es capaz de enviar las soluciones de agarre para las articulaciones de los dedos y la palma de la herramienta.

,

2.3 VISIÓN ARTIFICIAL MEDIANTE KINECT

2.3.1 Introducción

Para el desarrollo del apartado de visión artificial del proyecto, se utiliza la cámara Kinect de Microsoft. Esta cámara sirve de puente entre el entorno virtual y el entorno real. Para procesar los datos que esta aporta, se utilizan drivers diseñados específicamente para Ubuntu, como Freenect o OpenNi. Mediante aplicaciones de visión ejecutadas en ROS se obtendrán los datos necesarios para procesar objetos del espacio real.

Figura 2.15: Kinect para Windows

2.3.2 Características

- Cámara RGB, lentes de color y sensor de profundidad infrarrojo.

-Campo de visión horizontal 57 grados, vertical 43 grados.

-Rango de inclinación física ± 27 grados

-Rango de profundidad efectiva: 1,2 - 3,5 metros

-Data streams: 320x240 a 16 bits, 640x480 a 32 bits.

-Refresco de imagen: 30 FPS.

-Micrófono integrado (no se utiliza en el proyecto).

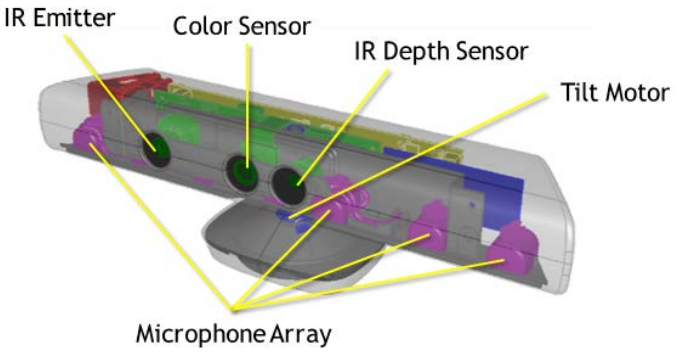


Figura 2.16: Componentes de Kinect

2.3.3 PCL y asociación a primitivas

La visión artificial se basa en obtener información del entorno real, para procesarlo y codificarlo, de tal forma que pueda recrearse la información a nivel de código. El aprendizaje a la visión es un criterio ampliamente usado: se da información al sistema sobre a qué se debe parecer el resultado obtenido, y mediante comparaciones con los valores obtenidos y aproximaciones sucesivas se discierne entre si el objeto se parece al requerido o no.

Mediante la adquisición de la nube de puntos relacionada con el valor de la forma primitiva de los objetos se puede obtener información relativa a la forma básica del objeto, de tal forma que puede ser almacenada en una estructura de datos. Las geometría otorgada por las primitivas asociadas al objeto en cuestión otorga una configuración original a la matriz de valores de la nube de puntos, y de esa forma se pueden obtener similitudes una vez almacenadas. Usando algoritmos de Point Cloud los objetos se pueden filtrar por primitivas, obtener su ubicación y su orientación.

La extensión a la información obtenida por la nube de puntos abarca desde la situación espacial, como al color de la forma, o incluso material de fricción.

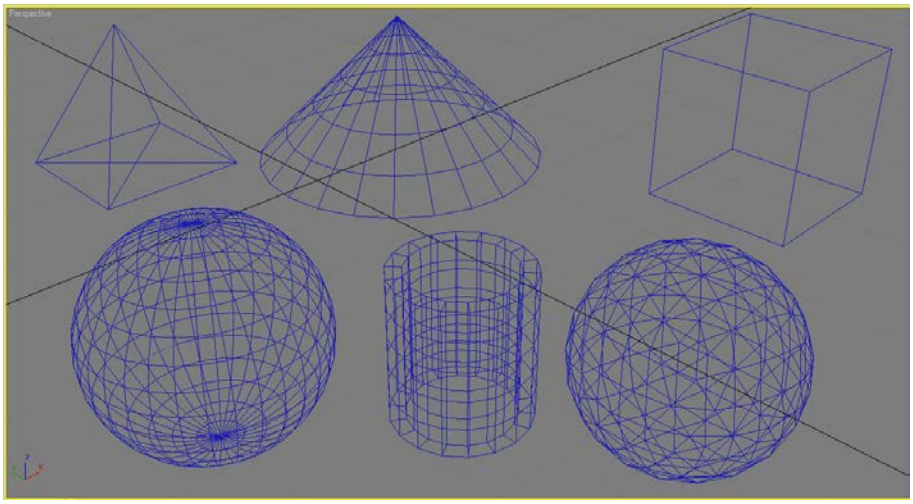


Figura 2.17: Nube de puntos primitiva

La librería PCL está basado en C++ y se dedican a procesar nubes de puntos en 3D.

Es una librería de software libre, que se ha ido enriqueciendo por los desarrolladores aumentando la cantidad de formas primitivas asociadas a objetos guardadas en su base de datos. Dado que la cantidad de información almacenada es enorme, el proceso de aprendizaje es muy completo, por lo que, mediante el uso de cámaras como la Asus XTionPro, PrimeSensor 3D o Microsoft Kinect, una vez obtenido la nube de puntos del objeto a reconocer, es trivial la asociación a una forma conocida ya aprendida en la biblioteca.

En este proyecto el apartado de visión se centrará en el reconocimiento de formas básicas, tales como la esfera, el cilindro y el prisma.

2.3.4 Paquete de visión mediante Kinect

Este proyecto utiliza el paquete "**threeDOR**" desarrollado por *Alberto Lázaro Enguita* en su trabajo: 3D Object Recognition and Pose Estimation using VFH Descriptors [2]. En él se hace un análisis más profundo de la nube de puntos y de la adquisición de primitivas. También se utiliza un paquete de acondicionamiento para la cámara Kinect en cuestión.

El paquete "threeDOR" realiza la tarea de visión sobre objetos de carácter geométrico de tipo esférico, cilíndrico o prismático. Este paquete se encarga de obtener la posición XYZ del objeto, su orientación respecto al plano, y la identidad de la geometría del objeto.

Este nodo publica en el entorno de ROS un mensaje llamado "Objects", del tipo moveit_msgs::CollisionObject del cual se extrae toda la información relevante.

La información del mensaje está referida a la librería de PCL, sobre su forma primitiva, y no muestra su situación real respecto al origen de coordenadas del robot.

Dado que la cámara y el robot tienen dos sistemas de referencia distintos, la posición y orientación deben ser procesadas para referirlas al sistema de referencia del robot. Además, la cámara Kinect tiene sus propio cifrado al recibir la señal de video, por lo que también hay que acondicionar los valores que obtiene, respecto al sistema real.

Es por eso que se utiliza el paquete "**object_atta**" para acondicionar completamente la señal aportada por la cámara. Este nodo publica en sus mensajes las posiciones y orientación de los objetos referidos al origen de coordenadas del robot. Estos mensajes serán los que se tramiten para componer las nuevas situaciones de posición y orientación de la herramienta de agarre.

Un problema al trabajar con el paquete threeDOR ha sido el reconocimiento de la orientación espacial de los objetos: solo otorga la orientación respecto al eje Z del sistema (eje vertical). Esto es útil únicamente al posicionar el objeto prismático, estando este perpendicular a la mesa.

El paquete de agarre diseñado en este proyecto proporciona soluciones a agarres oblicuos al plano mediante cálculo matricial. En las aplicaciones que se desarrollan más adelante, el paquete threeDOR se utilizará solo con la intención de obtener la posición y el tipo de objeto sobre el entorno de trabajo. Condiciones particulares de orientación y su representación en el entorno de MoveIt! serán tratadas más adelante, no obstante, en un futuro se podría disponer de otro paquete de visión que cubriera el inconveniente inicial.

2.4 BRAZO ROBOT IRB 120

2.4.1 Introducción

El robot IRB 120 es producido por la compañía ABB en su colección de robots de cuarta generación de robots manipuladores.

Este robot es idóneo respecto a la investigación en laboratorio, dado que su estructura y cinética es muy parecida a la tirada de robots industriales de ABB para cargas elevadas. Esto hace que los proyectos realizados a pequeña escala puedan ser extrapolados fácilmente a gran escala.

2.4.2 Estructura y parámetros mecánicos

Tiene 6 grados de libertad, muñeca esférica, tamaño reducido, capacidad de levantar cargas de 3Kgs y software de simulación propio (RobotStudio). La precisión del robot (sin carga) oscila entre +-0.01mm. Los motores son todos eléctricos [11].

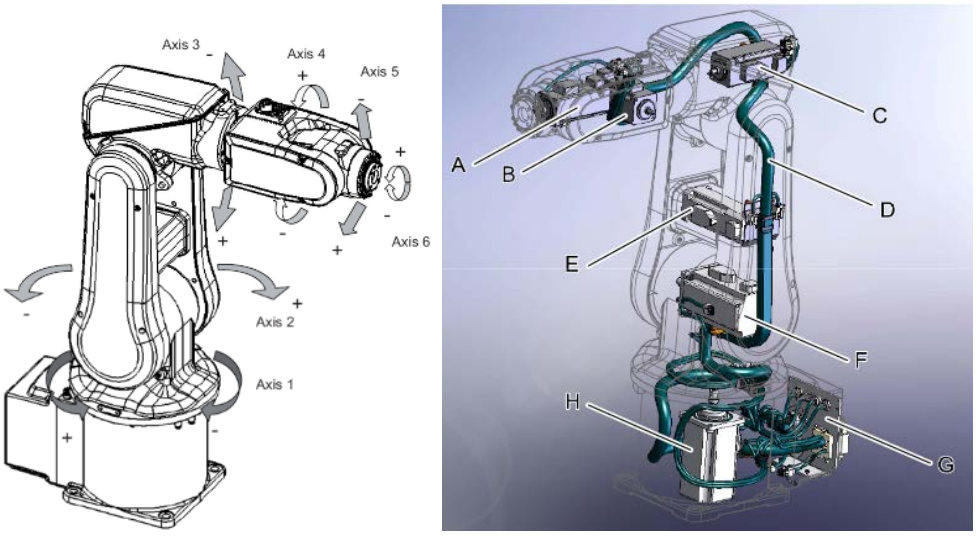


Figura 2.18: Disposición de los ejes y los motores del ABB IRB 120

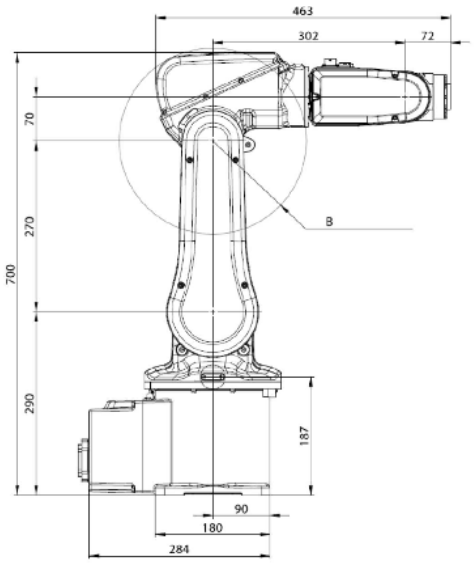


Figura 2.19: Dimensiones del ABB IRB 120

Dado a su diseño de brazo puede ser anclado mediante su base a paredes o techos de la instalación de trabajo, teniendo un rango efectivo de 580mm.

El robot se comunica y controla mediante IRC5, que está dotado del control de las articulaciones del robot y de un apartado de INPUT/OUTPUT.

ESLABÓN	MATRIZ	θ.*(rads)*	d.*(mm)*	r.*(mm)*	α.*(rads)*
1	A01	q1	290	0	-pi/2
2	A12	q2-pi/2	0	270	0
3	A23	q3	0	70	-pi/2
4	A34	q4	302	0	pi/2
5	A45	q5	0	0	-pi/2
6	A56	q6+pi	72	0	0

Tabla 2.3: Parámetros Denavit-Hartenberg

2.5 CONTROL POR MOVEIT!

MoveIt! es un software controlador que permite planificar rutas de movimiento para un robot. Es muy importante su implementación, puesto que resuelve el desafío de definir rutas punto por punto de manera manual. Dentro del entorno, MoveIt! permite integrar los componentes que forman el sistema robot en un espacio de simulación 3D, por el cual se puede definir la morfología del robot, objetos y obstáculos. Para visualizar dicho entorno se utiliza la aplicación RVIZ.

Mediante la integración en escena de todos los componentes que están en juego en el entorno real, se pretende recrear virtualmente para que MoveIt! pueda usar planeadores de movimiento. De esta forma, basta con establecer la situación inicial del desplazamiento (posición y orientación inicial) y una situación final de desplazamiento, y MoveIt! se encargará de planear una serie de rutas hábiles, teniendo en cuenta la cantidad de tiempo invertido para planificarlo, y si hay necesidad de esquivar los obstáculos del mapa.

Es muy importante resaltar dos condiciones a considerar ante la posibilidad de error:

-Existen diferentes planificadores para ser usados, el usuario debe saber cuál se ajusta mejor a los movimientos a trazar del sistema. En caso de no usarse el planificador adecuado, MoveIt! puede retornar error de planificación.

-MoveIt! planificará de acuerdo a las condiciones lógicas impuestas por el usuario: situaciones por las cuales existan colisiones forzadas o giros articulares descontrolados pueden reportarse como errores de planificación.

Por defecto, tanto los planificadores como las situaciones de agarre han sido probados para resultar funcionales.

De esta forma MoveIt! realizará las tareas de control articular sobre el brazo robot, atendiendo a la orientación y posición definidas por las aplicaciones.

2.6 SIMULACIÓN MEDIANTE ROBOTSTUDIO

Durante el desarrollo del proyecto ha sido necesario poder realizar pruebas sobre un brazo robot simulado, por dos razones:

-Seguridad: En el entorno de simulación no se pone en riesgo la integridad del robot, el entorno o el operario. Los sistemas de simulación actuales son lo suficientemente fidedignos para representar la gran totalidad de restricciones del robot real. Además, simular el sistema tiene un gasto energético menor y no produce desgaste articular en el robot.

-Acceso a pruebas ilimitado: el brazo robot real es único, y tiene un acceso restringido. En el entorno de simulación, siempre y cuando se disponga de un equipo para simular, el acceso puede ser ininterrumpido y se pueden realizar tantas pruebas como se desee. Mantiene las ventajas de simulación en entorno real, e incorpora características adicionales propias de la aplicación informática, tales como los valores de las articulaciones o los mensajes intermedios entre brazo y controlador.

Para realizar las simulaciones del brazo robot se ha utilizado el software de ABB RobotStudio 5.15. En el apartado de manual de usuario se adjunta la guía paso a paso para configurar el entorno, junto a la configuración del socket.

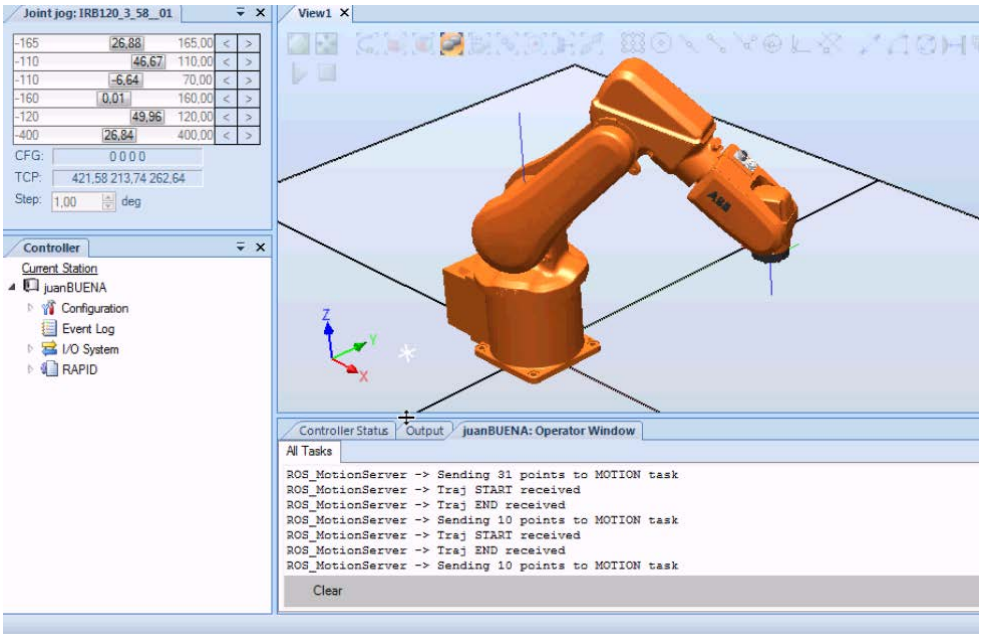


Figura 2.20: Entorno de RobotStudio

2.7 GraspIt!

2.7.1 Introducción

La plataforma de simulación GraspIt! [12] fue diseñada como una ayuda visual para la práctica de agarres robóticos con pinzas tanto industriales como antropomórficas. Diseñado por el departamento de computación de la Universidad de Columbia, GraspIt! permite simular agarres y reacciones de fuerza sobre objetos con diferentes pesos, fricciones y tamaños, realizar análisis de la calidad del agarre y planificarlos respecto a la descomposición de componentes primitivas del objeto a agarrar.

Figura 2.21: Logo oficial de GraspIt!

Los planificadores de GraspIt! están reservados para la planificación de agarres mediante BarrettHand, debido a que tiene una cantidad de grados de libertad moderada con el que el sistema puede computar de forma estable: 4 grados de libertad en articulaciones, orientación y posición.

Más adelante se entrará en detalle en los procesos de planificación.

2.7.2 Descripción del entorno de GraspIt!

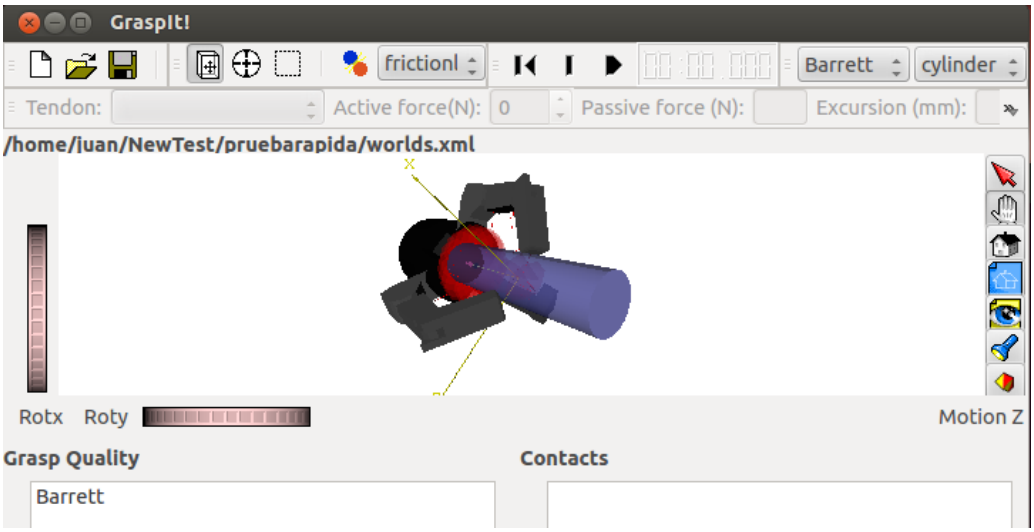


Figura 2.22: Entorno de GraspIt!

La interfaz de GraspIt! se presenta como un entorno 3D donde se pueden cargar modelos de objetos, robots u obstáculos. Aquellos elementos que se hagan aparecer en escena serán reflejados dentro de un archivo "*world.xml*" que almacena los datos del proyecto. De este archivo se pueden obtener los valores numéricos para los que el software simula.

En la pantalla principal las opciones de navegación que da GraspIt! permiten desplazar los elementos de la escena dentro del espacio 3D.

Permite desplazar mediante el espacio cartesiano el elemento seleccionado.

Permite desplazar la cámara dentro del escenario de la simulación.

Devuelve la posición de la cámara a la posición de referencia de cámara, por defecto el origen de coordenadas.

Permite cambiar la posición de referencia de la cámara.

Permite alejar la cámara a una situación en la que se realiza una perspectiva general del escenario.

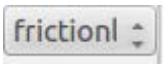
Selecciona un objeto y lo convierte en la nueva referencia.

Permite alternar entre una representación caballera o isométrica.

Otras opciones a la vista comúnmente utilizadas son:

Activar /Desactivar colisiones:

Permite realizarlas o ignorarlas. Es importante matizar el hecho de que para planificar agarres es necesario activar dicha casilla, si no el planificador nunca encontrará puntos de presión como referencia a la primitiva.



Carácter de fricción:

En esta casilla es posible especificar la fricción del material a agarrar. La fricción, junto al centro de masas, es un carácter importante a tener en cuenta respecto a planificar un agarre, y es necesario tenerlo en cuenta en aquellas aplicaciones que requieran levantar objetos con cierto peso.

Respecto a la carga de modelos, mediante la pestaña *File* se pueden importar modelos de robots, obstáculos y objetos. Por defecto estas librerías se encuentran en la carpeta *"models"* del directorio raíz.

A continuación se procede a explicar cómo modelar objetos para añadirlos al entorno de trabajo y su representación adjunta en primitivas.

2.7.3 Modelado en el entorno de GraspIt!

GraspIt! cuenta de un fuerte apartado visual, pero es necesario introducirse en el manejo a nivel de código de ficheros para poder modelar su entorno con precisión.

Existen tres tipos de componentes que se pueden añadir a la escena:

-Obstáculos: no están permitidos su agarre. Su único cometido es generar una zona en el espacio que no se pueda atravesar.

-Objetos: son formas geométricas capaces de ser agarradas. Todas pueden ser probadas para realizar un agarre de forma manual, pero solo algunas tienen asociadas un modelo de forma primitiva. Estas últimas son las únicas a las que se puede planificar un agarre automático.

-Robots: son los efectores finales que realizan el agarre de los objetos. Todos pueden simularse para un agarre manual, pero solo la BarrettHand puede usarse para planificar agarres de manera automática.

Siempre que se guarda un proyecto en GraspIt! se guarda con la extensión *.xml,* como entorno "*world*". Este fichero muestra toda la información de la escena del proyecto, los componentes que la forman, su orientación, posición e incluso situación de la cámara.

```
worlds1.xml
<?xml version="1.0" ?>
<world>
        <graspableBody>
                <filename>///////home/juan/NewTest/graspit/models/objects/caja.xml</filename>
                <transform>
                        <fullTransform>(+1 +0 +0 +0)[+0 +0 +0]</fullTransform>
                </transform>
        </graspableBody>
        <obstacle>
                <filename>///////home/juan/NewTest/graspit/models/obstacles/table.xml</filename>
                <transform>
                        <fullTransform>(+1 +0 +0 +0)[-16.4077 -615.237 +415.939]</fullTransform>
                </transform>
        </obstacle>
        <robot>
                <filename>///////home/juan/NewTest/graspit/models/robots/Barrett/Barrett.xml</filename>
                <dofValues>+0 +1.2793 +0 +0 +1.2793 +0 +0 +1.2793 +0 +0 </dofValues>
                <transform>
                        <fullTransform>(+0.5 +0.5 +0.5 -0.5)[+0 +38.8094 +0]</fullTransform>
                </transform>
        </robot>
        <camera>
                <position>+1209.38 +1066.4 +421.009</position>
                <orientation>+0.244361 +0.559328 +0.717924 +0.334702</orientation>
                <focalDistance>+1667.47</focalDistance>
        </camera>
</world>
```

Figura 2.23: Fichero world.xml de GraspIt!

Tal y como se muestra, los parámetros son análogos para los elementos en la escena:

- Las etiquetas de título muestran el carácter del componente: <world>, <graspableBody>, <obstacle>, <robot>

- <filename> muestra la dirección donde se haya guardado el componente

-<transform> <fullTransform> representa la orientación y la posición del componente

La forma en la que se expresa este término es mediante un vector de 7 valores: orientación expresada en cuaternios y posición expresada en milímetros.

Figura 2.24: Formato de la etiqueta <fullTransform>

-<dofValues> este vector está asociado únicamente para los casos de componente tipo robot, y muestran los valores que toman las articulaciones de este. Esto hace que, dado que existen diferentes efectores finales con diferente número de articulaciones, el tamaño de este vector no sea igual para todos los robots: aquellos robots que tengan más articulaciones tendrán un vector de grados de libertad mayor. En este vector también existe un apartado relacionado con el Torque Switch, para manos robóticas como la BarrettHand.

La forma en la que se expresan los valores en el vector está ordenada de forma personalizada según el tipo de robot. En el caso de la BarrettHand, el vector de dofValues se expresa de esta forma:

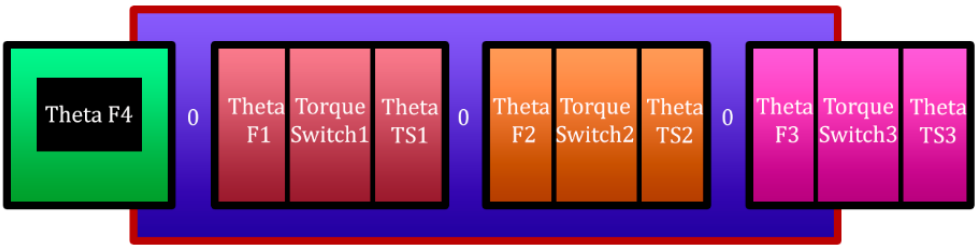


Figura 2.25: Formato de la etiqueta <dofValues>

Donde los ángulos Theta están expresados en radianes, el parámetro Torque Switch es un *flag* que indica si es necesario su uso, y Theta TS los grados que deben girarse con torque. Si el flag de Torque Switch es 0, el campo Theta TS se suprime del vector y se salta directamente al separador, “0”.

Mediante este archivo de world.xml se puede obtener los datos de la escena en todo momento y utilizarlos para agarres futuros. Todas las simulaciones de agarres con GraspIt! fueron guardadas y leídas sobre el archivo world.xml y luego transcritas a la base de datos de agarres.

2.7.4 Edición de objetos

Los modelos de los objetos a cargar en GraspIt! se encuentran en el subdirectorio de la carpeta principal de instalación: *Graspit/models*. Dentro de esa carpeta se subdividen los modelos para los objetos, robots y obstáculos. El formato de los modelos está en extensión *.iv*, usada por Open Inventor. Esto permite que los diseños puedan ser exportados con facilidad al entorno de GraspIt! . Cada modelo también tiene asociado un fichero *.xml* a modo de paquete descriptor.

box.iv box.xml

```
<?xml version="1.0" ?>
<root>
        <material>plastic</material>
        <mass>440</mass>
        <cog>0 0 0</cog>
        <geometryFile type="Inventor">box.iv</geometryFile>
</root>
```

Figura 2.26: Fichero model.xml de GraspIt!

-<root> es la cabecera para especificar al programa desde donde tiene que abrir el modelo

-<mass> especifica la masa del objeto a simular, en gramos.

-<cog> determina el centro de masas del objeto respecto a su centro de coordenadas, en milímetros.

-<geomeryFile> carga el modelo de Open Inventor, que debe estar en la misma carpeta que el fichero .xml

El fichero en .iv se compone a su vez de varios campos que indican las características del modelo guardado. La extensión de este archivo depende de la cantidad de primitivas u objetos simples que formen el modelo.

Dentro de la carpeta de modelos, existe una carpeta llamada “primitives”. En esta carpeta se almacena la información referente a las primitivas de los modelos. Es importante que exista un archivo, análogamente nombrado al modelo referente, para que se puedan planear agarres de objetos.

Para sintetizar un objeto singular basta con componer o modificar los archivos .iv y .xml respectivos a su modelo y a su primitiva, o exportar del entorno Open Inventor el modelo deseado.

Para la simulación de los agarres en este proyecto se han creado tres modelos de objetos para ser simulados en agarre, llamados “pelota”, “lata” y “caja”, editados directamente de los modelos por defecto de GraspIt!.

Las dimensiones de estos objetos se muestran a continuación:

Pelota: Esfera de 32 mm de radio.

Lata: Cilindro de 32 mm de radio, 105 mm de largo.

Caja: Prisma de 75x70x200 mm.

Estos modelos tienen representación en primitivas, cualidad necesaria para que los planificadores puedan actuar.

2.7.5 Planificadores de agarre usando la BarrettHand

Una vez que un objeto y un robot están en escena, GraspIt! puede empezar a planificar agarres. Es importante destacar que GraspIt! limita sus interacciones con un solo objeto, por lo que para un correcto funcionamiento, en el entorno de simulación solo debe haber un robot y un objeto a agarrar. La opción de activar colisiones debe estar activa para corroborar los agarres.

En la opción "*Grasp*" de la barra de herramientas se puede acceder a todos los modos que ofrece GraspIt! para simular. Existen las opciones *Auto Grasp* y *Auto Open*, que permiten probar el correcto funcionamiento de las físicas y el robot.

También existen otras opciones relacionadas con ajustar los parámetros de la calidad del agarre o su fuerza, aunque la función más potente es la planificación de agarres.

GraspIt! cuenta con dos planificadores principales: ***Planner*** y ***EigenGrasp***.

2.7.5.1 -PLANNER-

Planner es el planificador propio de GraspIt! basado en la captación de primitivas, solo funciona con la BarrettHand y necesita de archivos de primitivas en "*models/objects/primitives*". Realiza dos fases: en un principio genera agarres tomando solo las primitivas como referencia, usando tantos pre-agarres como potencia computacional se le indique.

La segunda fase realiza acercamientos al objeto según las directrices de los pre-agarres, y computa la situación de la calidad del agarre hasta que todos los casos son resueltos.

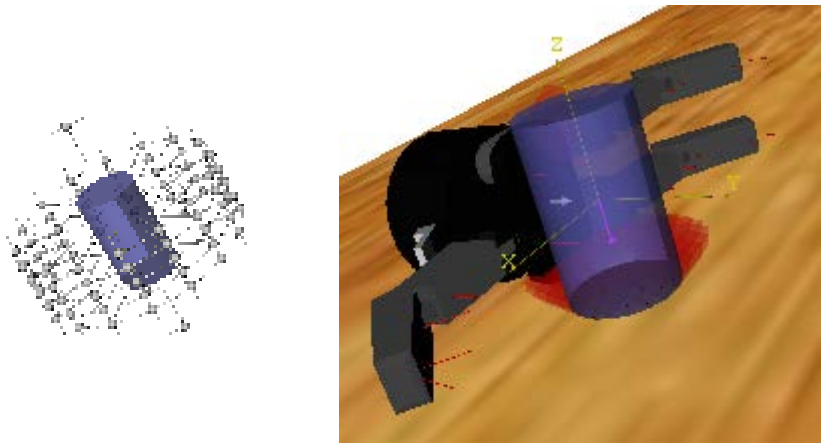


Figura 2.27: Computación y resolución al agarre mediante PLANNER

Para realizar este tipo de simulación de agarres, se siguen los siguientes pasos:

Grasp>>Planner>>>>New>>Add/Edit

En esta pestaña se puede seleccionar la condición más óptima para calcular la calidad de los agarres. Para este proyecto se ha usado la configuración:

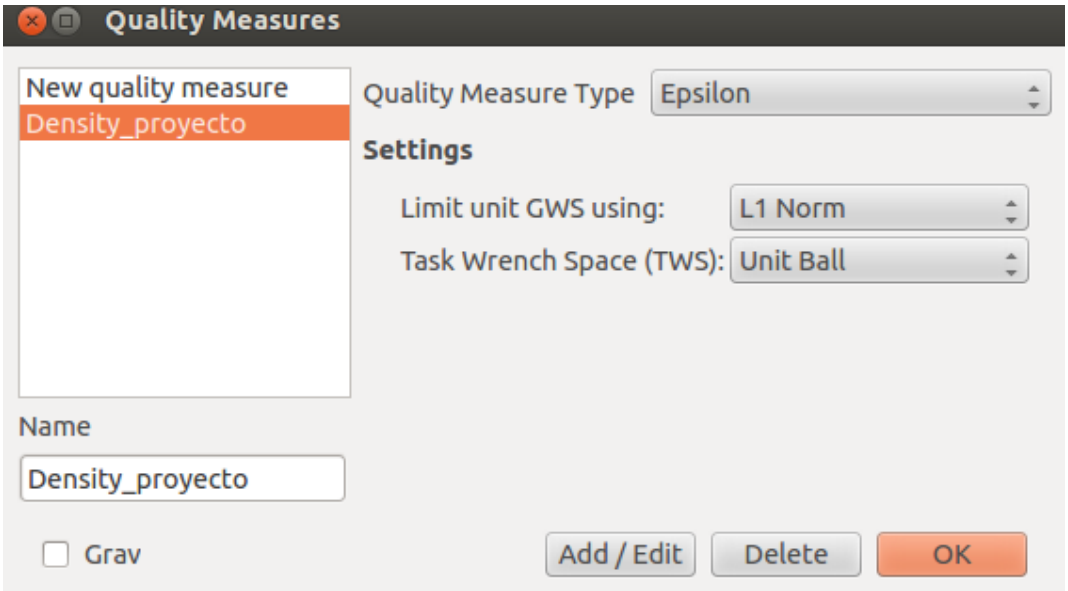


Figura 2.28: Configuración del criterio de calidad en PLANNER

Una vez configurado este campo se puede validar mediante la opción OK.

De vuelta a la ventana principal, se pueden especificar la cantidad de pre-agarres que se van a realizar, cuanto se van a retraer una vez pre calculados, o zonas críticas por las cuales tiene que surgir una solución de agarre.

La cantidad de soluciones de agarre y la varianza entre ellos, depende críticamente de este apartado. Es por ello que es necesario evaluar de forma independiente cada aplicación, objeto y coste de tiempo asumible para obtener una base de datos propia.

Los factores que condicionan el tiempo de cálculo y la cantidad de iteraciones son:

-Sampling density factor: Factor que relaciona la cantidad de puntos en la nube de primitivas que se tienen en cuenta al calcular las soluciones. A mayor factor de densidad, mayor cantidad de descomposiciones primitivas del objeto en cuestión.

-Backstep size (mm): Desplazamiento mínimo para alejarse de la superficie del objeto y calcular una solución.

-Max # of backsteps: Cantidad de alejamientos entre la superficie del objeto y la BarrettHand. Se rige por el criterio de desplazamiento mínimo, y permite considerar la distancia a la que podría aportarse una solución de agarre ante alejamientos sucesivos de la palma del robot con la superficie del objeto.

Para los modelos de simulación realizados se ha usado la siguiente configuración:

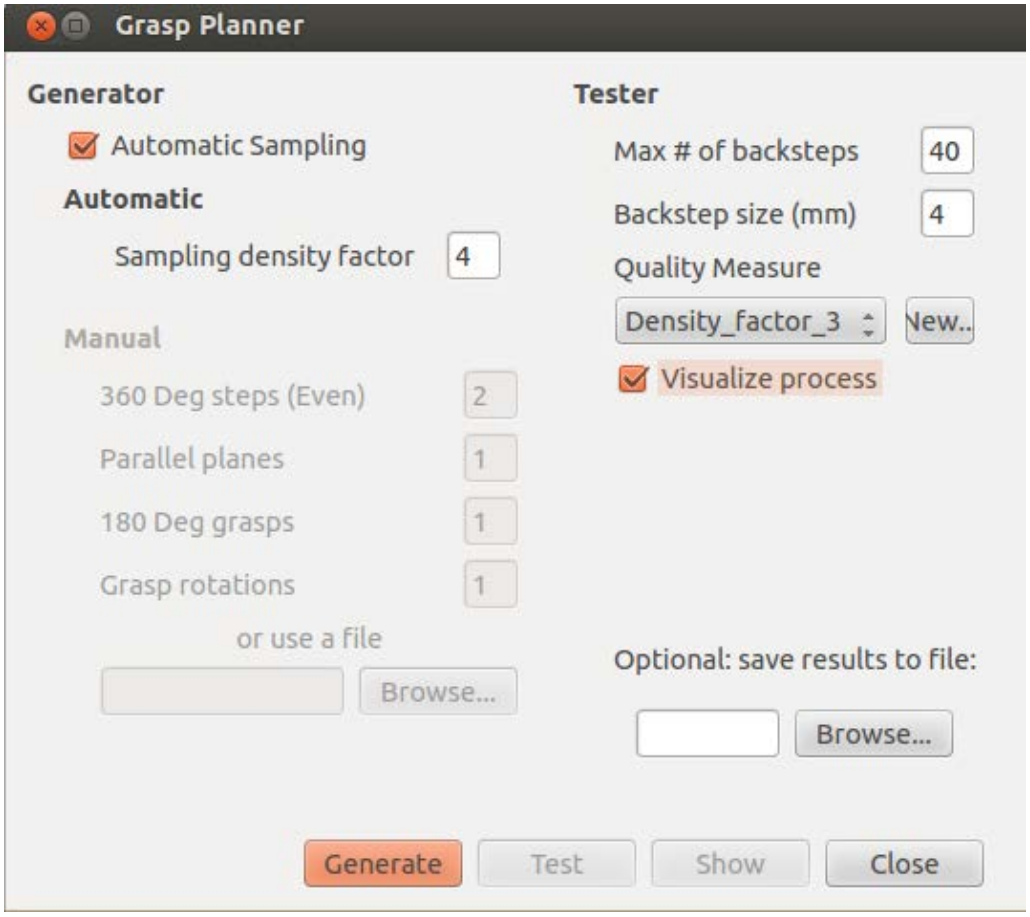


Figura 2.29: Configuración general de PLANNER

Actuando secuencialmente sobre los botones: *Generate*, *Test* y *Show* se generan y pre-visualizan los agarres calculados, siempre y cuando la casilla "*Visualice process*" haya sido activada.

La cantidad de agarres calculados se corresponde con la potencia del equipo que lo simula. GraspIt! tiene problemas de estabilidad ante situaciones de proceso de cálculo elevado, por lo que se recomienda que se trabaje siempre a un nivel inferior de las máximas prestaciones admisibles.

2.7.5.2 -EIGENGRASP-

Los agarres mediante computación se representan mediante un vector relacionado con los grados de libertad del efector. Existen una serie de agarres con sinergia entre ellos para realizar agarres, debido usualmente al tamaño y a la forma de los objetos: A forma de ejemplo, es factible agarrar una bombilla con la misma configuración de agarre que se utilizaría para agarrar un cilindro.

El término EigenGrasp se apoya en este concepto: provee posiciones de agarre en concepto a situaciones ya conocidas o parecidas a las aportadas por la experiencia, necesita menos tiempo de procesado y puede ser modelado como un vector a nivel computacional.

Es muy usado para herramientas de agarre como manos robóticas, dado que la filosofía de agarre es parecía a la de los seres humanos, como prueba, error, y extrapolación.

Tiene una serie de limitaciones, dado su "nivel alto" de ejecución: Para grandes DOF tiene problemas de optimización, no es posible calcular de forma precisa la energía o calidad del agarre, y puede obviar agarres puntuales, debido a que por su experiencia anterior no hubiera podido planteárselo.

Este planificador otorga ventajas en el hecho de flexibilidad de soluciones de agarre ante objetos semejantes. Es importante remarcar que es un planificador a un nivel mayor de comprensión que el planificador de primitivas PLANNER, por lo que, aunque resulte más cómodo para pruebas puntuales, no se utilizará en este proyecto para obtener agarres mediante GraspIt! . Se espera que en el futuro el planificador sea más robusto y más rico en soluciones múltiples de agarre.

Ejemplo de agarre para mano:

-En un nuevo proyecto, cargue el robot "HumandHand20DOF" y el objeto "flask.iv".

-Seleccione en el menú: Grasp-> EigenGrasp Planner

-Con las especificaciones por defecto, click en Init, click en ">", y luego click en "Best"

La pantalla muestra el mejor agarre que ha simulado el EigenGrasp para aferrar el frasco.

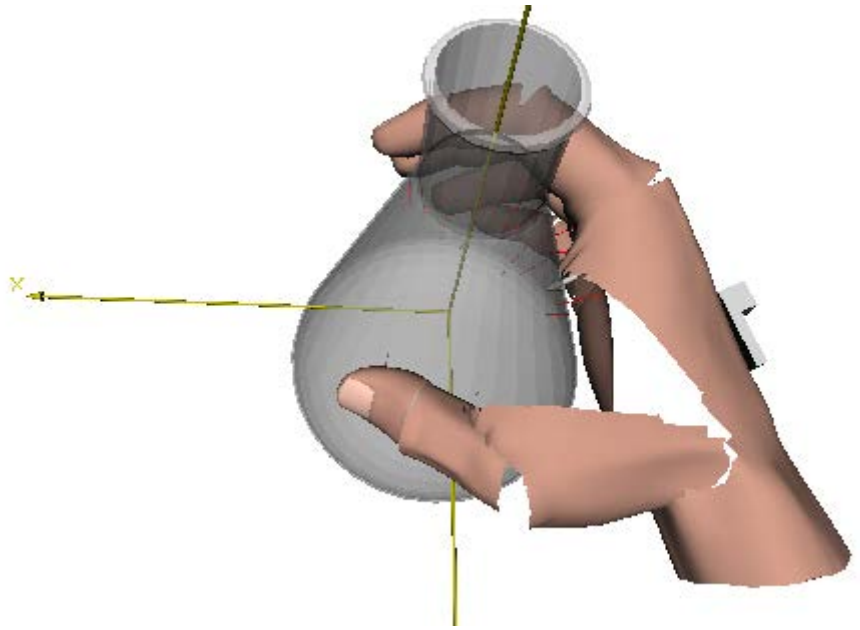


Figura 2.30: Ejemplo de EIGENGRASP

2.7.6 La base de datos del proyecto

Tal y como se ha explicado en los anteriores pasos, “PLANNER” ha sido el planificador de agarre elegido para simular los agarres usando la herramienta BarrettHand. En la carpeta “*datos_graspit*”, dentro del directorio del paquete de agarre, se pueden encontrar los archivos “*.txt*” de las soluciones de agarre para los objetos singulares definidos con anterioridad.

Dado que la aplicación carga la información relativa a los agarres desde estos ficheros, esta estructura permite poder enriquecer la base de datos de la aplicación, siguiendo los pasos tal como se ha explicado en este tutorial de GraspIt!.Para almacenar la información mediante archivos de texto es necesario que se respete la estructura de carga, tal y como muestra el siguiente diagrama:

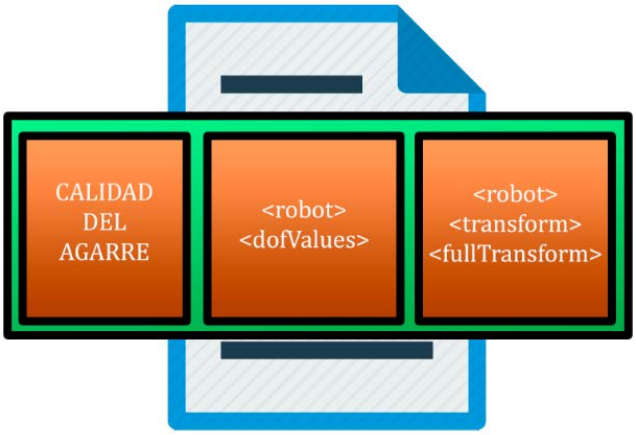


Figura 2.31: Estructura de la información en la base de datos

Estos valores han de almacenarse únicamente mediante sus cifras, sin símbolos ni texto, y dejando espacio entre cada parámetro a modo de separación. Es recomendable, una vez que los archivos puedan tomarse como definitivos, cambiar sus atributos a “Solo lectura” para evitar problemas de acceso o corrupción en los archivos.

2.7.7 La base de datos de la Universidad de Columbia

El hecho de recabar información para realizar agarres y computarla para el aprendizaje es un proceso costoso y que requiere tiempo. Una iniciativa desarrollada como solución para agilizar el proceso ha sido realizar bases de datos de estos agarres, de tal forma que se pueda requerir a las soluciones planeadas con anterioridad sin tener que simular de nuevo. La base de datos de la Universidad de Columbia (CGDB) está estructurada como una base de datos de PostgreSQL, que contiene información sobre más de 250000 agarres a 8000 modelos 3D diferentes.

Para acceder a la base de datos es necesario presentar una petición de acceso Esta información ha sido simulada para tres tipos de robot:

-BarrettHand con bajo coeficiente de fricción (dedos de plástico).

-BarettHand con alto coeficiente de fricción (dedos de goma).

-Mano antropomórfica.

Si se dispone de la herramienta y de un sistema de visión artificial adecuado, sería posible mediante inteligencia virtual recrear una enorme cantidad de agarres a objetos comunes.

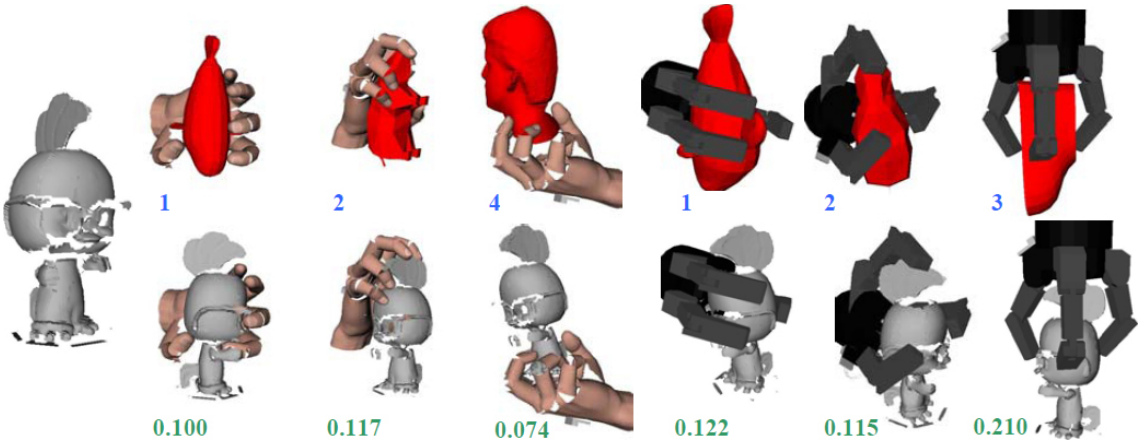


Figura 2.32: Ejemplos de la base de datos de la Universidad de Columbia

Este proceso tiene limitaciones respecto a la similitud entre el objeto detectado y el objeto supuesto a agarrar: Existen multitud de soluciones de agarre para objetos cotidianos simples, como tazas, botellas, cubiertos, o juguetes. Si el objeto detectado es singular, no se puede asegurar un correcto agarre del objeto, dado que no se ha almacenado previamente en la base de datos. En este caso, es necesario obtener previamente el modelo tridimensional del objeto en simulación y planificar los agarres. Debido al robot, la base de datos también está restringida respecto al tamaño, peso o fricción de los objetos a los que puede dar solución.

La base de datos de la Universidad de Columbia es una alternativa eficiente para realizar simulaciones virtuales de agarres y para probar múltiples disposiciones de la mano robótica, pero no es recomendable para pruebas reales donde la calidad del agarre y la precisión es primordial.

En este proyecto todos los modelos y sus soluciones de agarre han sido obtenidos mediante la propia simulación en el entorno de GraspIt! para asegurar una correcta ejecución de agarre.

CAPÍTULO 3

DESARROLLO DEL PAQUETE DE AGARRE “GRASP”

3.1 INTRODUCCIÓN

En este apartado se expondrá el contenido, los criterios y los resultados obtenidos al desarrollar el paquete de inteligencia lógica de agarres. Este paquete ofrece solución a un agarre puntual para un objeto concreto, necesitando saber para ello:

-La forma del objeto y su situación espacial, tanto en posición como orientación.

-La situación inicial desde la que se pretende agarrar el objeto.

3.2 DESARROLLO TEÓRICO

En explicaciones anteriores se introdujo las ideas básicas sobre los sistemas vectoriales relacionados con matrices de transformación para caracterizar el comportamiento de un robot. Tal y como se ha introducido con anterioridad, la orientación y la posición del efector final son los dos factores fundamentales que se deben caracterizar.

El proceso seguido para caracterizar los agarres y su situación espacial ha sido, tal y como ha sido explicado, mediante la herramienta GraspIt!, trabajando las simulaciones al agarre en primera instancia, y luego exportando los resultados obtenidos a un documento de texto.

El criterio seguido para la realización de esta base de datos se fundamenta en que, dado que el robot debe realizar el mismo movimiento que el otorgado por la simulación, su orientación y su posición relativa frente al objeto a agarrar debe permanecer constante. De esta forma, se consigue recrear con fidelidad la solución al agarre simulado.

Dentro del entorno de GraspIt! se aseguró que el entorno vectorial del espacio de trabajo se regía con la misma disposición que el entorno vectorial de ROS MoveIt!, de tal forma que las dos referencias de coordenadas OXYZ son iguales e equivalentes entre ellas, por lo que su desfase inicial de giro es 0, factor que simplifica cualitativamente la idea de exportación de datos.

$$^{Rotacional}GraspIt! \quad = {}^{Rotacional}MoveIt! \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 3.1: Matriz identidad de rotación

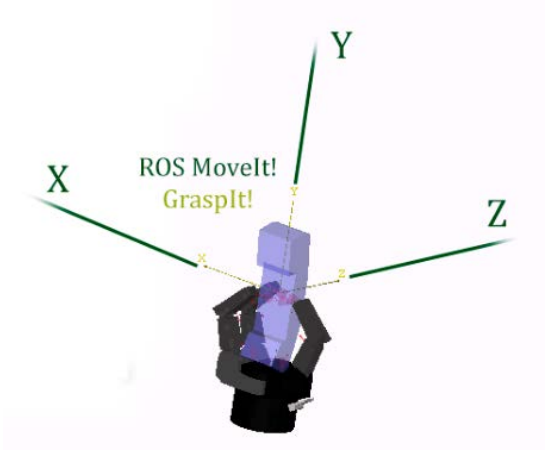


Figura 3.2: Relación entre referencias de GraspIt! y ROS MoveIt!

Así se consigue que no haya que realizar ninguna transformación matricial aparente entre sistemas, por lo que la orientación otorgada en la simulación es la misma que la realizada en el entorno de trabajo.

Respecto a la posición relativa del objeto frente al efector, independientemente de la situación espacial del objeto en el entorno de trabajo, la posición entre objeto y efector debe mantenerse fiel a la otorgada por la solución al agarre. Para poder extrapolar esta condición a cualquier situación de trabajo, dentro de la simulación de GraspIt! todos los objetos han sido colocados mediante su centro geométrico en el punto (0,0,0) de coordenadas. Así pues, toda solución de agarre está referenciada al centro geométrico propio de cada objeto. Sabiendo en segunda instancia la situación espacial del objeto en la zona de trabajo, la solución a la posición de agarre final se obtiene de sumar la posición de agarre relativa del robot y el objeto, con la posición entre objeto y zona de trabajo.

Teóricamente, la transformación espacial se corrobora mediante la siguiente expresión:

$$^{Robot}T_{XYZ} = {}^{Robot}T_{objeto} * {}^{objeto}T_{XYZ}$$

$$^{Robot}T_{XYZ} = \begin{bmatrix} 1 & 0 & 0 & x1 \\ 0 & 1 & 0 & y1 \\ 0 & 0 & 1 & z1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & x2 \\ 0 & 1 & 0 & y2 \\ 0 & 0 & 1 & z2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x1+x2 \\ 0 & 1 & 0 & y1+y2 \\ 0 & 0 & 1 & z1+z2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 3.3: Transformación matricial entre GraspIt! y ROS MoveIt!

Resolución gráfica del desplazamiento final

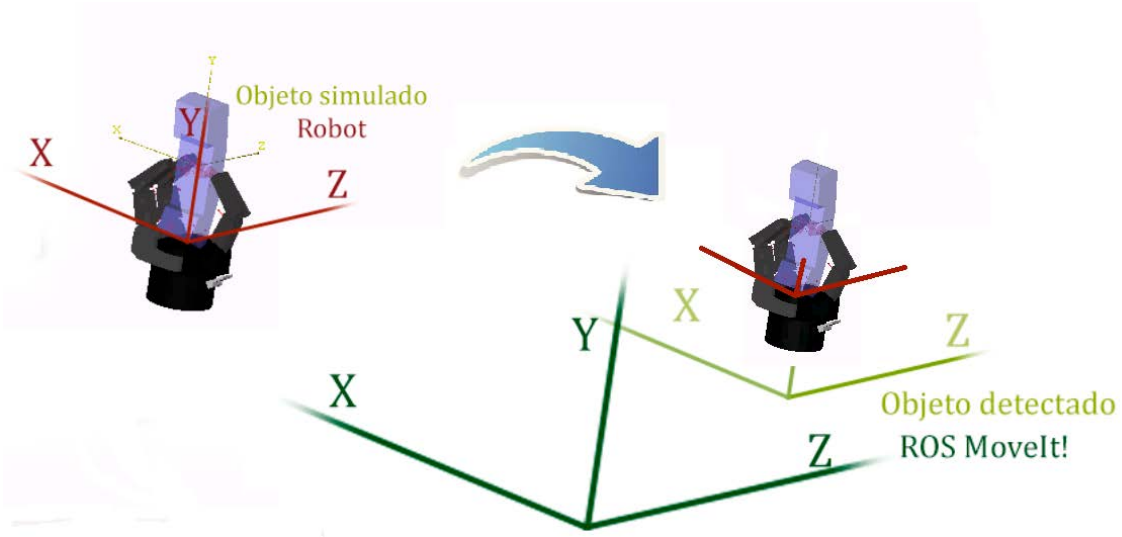


Figura 3.4: Transformación espacial entre GraspIt! y ROS MoveIt!

La posición relativa del objeto simulado frente al robot, en este caso, la herramienta BarrettHand, se toma desde la palma de la herramienta. Debido a que cada gripper tiene su punto de referencia propio y es necesario indicar al brazo robot anexo cuanto se tiene que desplazar para colocar correctamente el efector, se ha optado por dejar

una variable dentro del programa que permite ajustar la distancia relativa entre referencia del brazo robot y la referencia del efector.

De este modo, simulando un offset proporcional como un tensor, se soporta la capacidad del sistema de ser configurable para diversos grippers de acción perpendicular, de manera trivial.

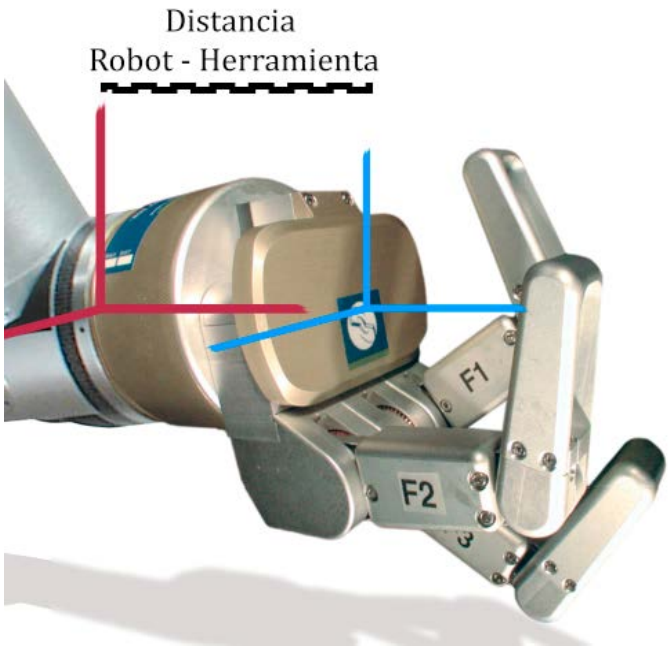


Figura 3.5: Distancia entre robot y palma del gripper

Conociendo la distancia entre robot y herramienta, la distancia entre el objeto y la herramienta aportados por GraspIt!, se puede obtener el resultado a la transformación entre sistemas de referencia:

$$Distancia\ entre\ objeto\ y\ herramienta = Módulo1 = \sqrt{(X_{GraspIt!})^2 + (Y_{GraspIt!})^2 + (Z_{GraspIt!})^2}$$

$$Distancia\ entre\ objeto\ y\ robot = Módulo1 + Distancia\ Robot_Herramienta = \sqrt{(X_{GraspIt!} * tensor)^2 + (Y_{GraspIt!} * tensor)^2 + (Z_{GraspIt!} * tensor)^2}$$

Figura 3.6: Desplazamiento del efector final entre robot y herramienta

El resultado de la obtención de este tensor otorgará la situación espacial del punto por el cual hay que desplazar el brazo robot.

3.2.1 Cambio de posición relativa respecto a la orientación del objeto

Cuando un objeto tiene una orientación determinada, y esta es capaz de ser analizada por el sistema de visión artificial, la posición del efector final debe "rotar" alrededor del objeto, encarando la orientación adecuada.

Este criterio solo ha podido ser utilizado en el caso determinado del agarre al prisma, puesto que el sistema de visión artificial sí reconoce las situaciones de sus caras frontales. Esto es necesario al agarrar el prisma de forma robusta, dado que se caracterizas las caras más anchas de este como principales zonas de agarre. Para los casos del cilindro y la esfera, el sistema de visión artificial no reportó información, debido principalmente a que es necesario caracterizar una orientación determinada para estos objetos, como por ejemplo en el caso de la esfera, con infinitas orientaciones. Para el cilindro, se supone que este está siempre apoyado sobre una de sus bases.

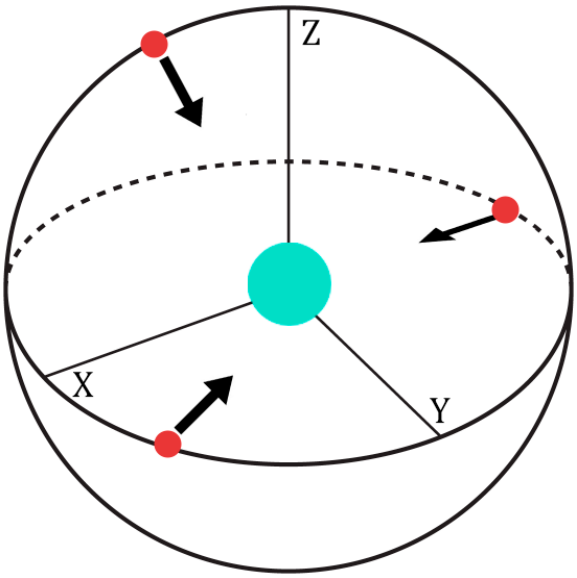


Figura 3.6: Ejemplo de desplazamiento frente a una referencia fija

Por comodidad de operación, dado que se trabaja con ángulos de rotación y cuarternios, se ha optado por enfocar este problema mediante ángulos de Euler [13].

La caracterización de un cuaternio $Q = q_0 + iq_1 + jq_2 + kq_3$ respecto a su matriz homogénea de rotación responde a la estructura:

$$\begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Figura 3.7: Representación del cuaternio como matriz homogénea

Transformando la matriz de rotación en componentes rotacionales sencillas, siguiendo la nomenclatura establecida por Tait–Bryan, las componentes de rotación pueden obtenerse de la siguiente manera:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0q_2 - q_3q_1)) \\ \operatorname{atan2}(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

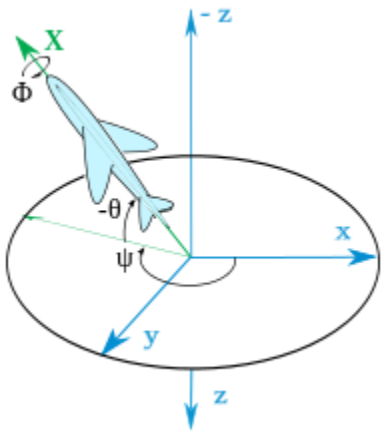


Figura 3.8: Relación entre cuaternios y componentes Tait-Bryan

Sabiendo la rotación respecto a los planos del entorno, se pueden rotar estos secuencialmente hasta rotar por completo los puntos requeridos.

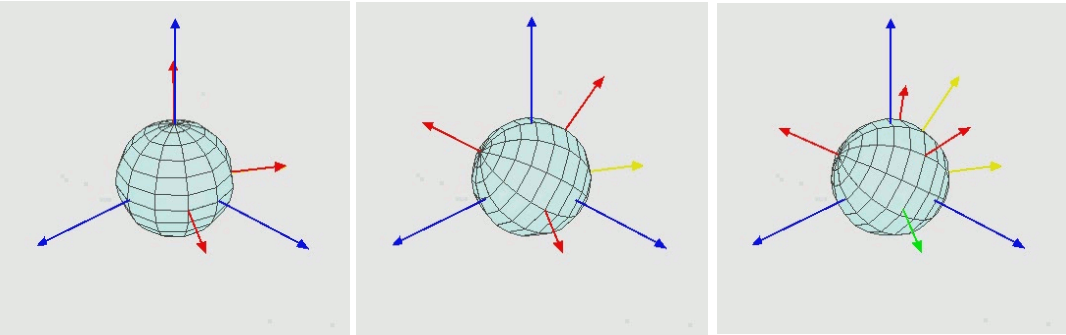
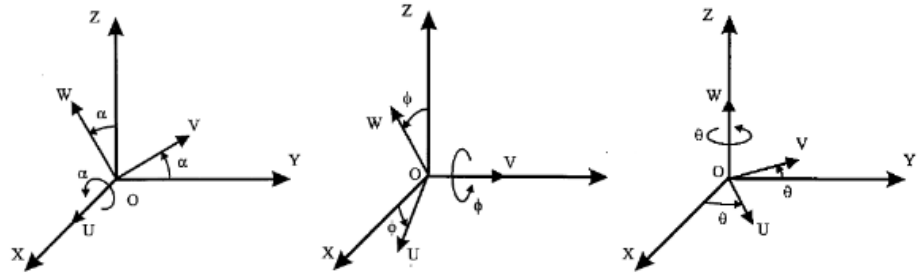


Figura 3.9: Ejemplo de rotación secuencial de planos



$$\mathbf{Rotx}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha \\ 0 & \operatorname{sen}\alpha & \cos\alpha \end{bmatrix} \quad \mathbf{Roty}(\phi) = \begin{bmatrix} \cos\phi & 0 & \operatorname{sen}\phi \\ 0 & 1 & 0 \\ -\operatorname{sen}\phi & 0 & \cos\phi \end{bmatrix} \quad \mathbf{Rotz}(\theta) = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 3.10: Matrices de rotación

3.2.2 Cambio de orientación relativa respecto a la orientación del objeto

La orientación del efector también debe evaluarse respecto a la orientación definida del objeto a agarrar. Al igual que en el anterior punto, en este proyecto solo el prisma depende de este aspecto, pero al igual que las expresiones matemáticas anteriores, las fórmulas se han expresado para ser funcionales en la totalidad de giros en el espacio a tres dimensiones.

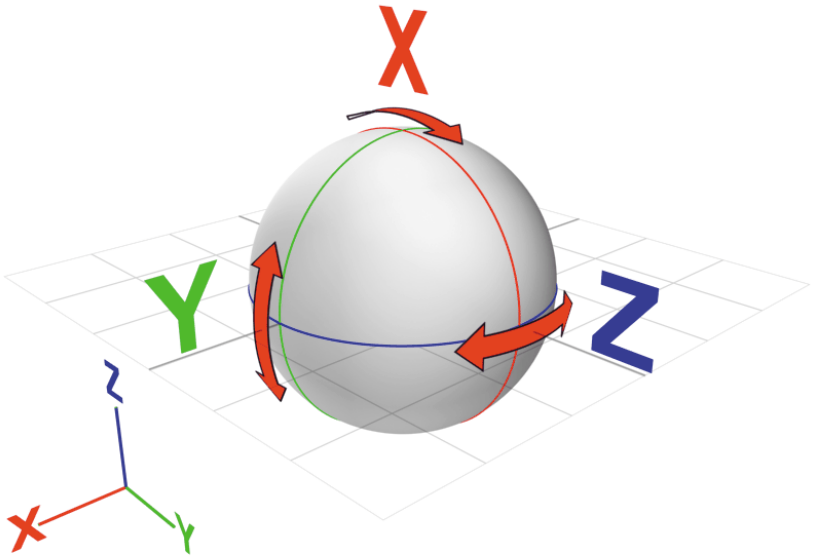


Figura 3.11: Ejemplo de rotación frente a una referencia fija

Para obtener las soluciones entre rotaciones relativas del sistema, se ha optado por la multiplicación de las componentes rotaciones de los cuaternios asociados a la situación otorgada por el módulo de visión artificial y la solución de GraspIt!. De esta forma se consigue la transformación final entre rotaciones de sistemas [14].

La fórmula aplicada para realizar el producto entre cuaternios ha sido:

$$ab = (a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4) + (a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3)i$$
$$+ (a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2)j + (a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1)k$$

Figura 3.12: Expresión de producto entre cuaternios57

Debido a que el producto entre cuaternios es asociativo y no conmutativo, es necesario especificar cuál es el sistema de referencia y el sistema que rota respecto a él. En este caso, el sistema "a" es la referencia, es decir, la orientación aportada por el sistema de visión artificial, mientras que el sistema "b" es la orientación relativa a la solución de agarre aportada por GraspIt!. El resultado de esta operación es la solución del paquete de agarre para orientar el efector final del robot para agarrar el objeto deseado.

De esta forma, se ha compuesto tanto en posición como orientación la solución del paquete de agarre para el efector final del brazo robot.

3.3 DESARROLLO LÓGICO

Una vez evaluada una solución de agarre para un caso concreto dentro de la zona de trabajo, es necesario compararla con más soluciones de agarre transformadas, con la misión de obtener la mejor solución entre todas las ofrecidas por la base de datos de agarres. De esta forma la aplicación se hace más eficiente, y se pueden introducir condiciones de agarre. En este proyecto se han desarrollado dos condiciones de agarre: máxima calidad y menor distancia para agarre.

Para realizar esta tarea, el paquete de agarre recibe un indicador de condición, el programa calcula soluciones de agarre a proponer y, de esas soluciones, realiza una selección entre ellas para obtener la solución más óptima, que es la que ofrece al final el paquete.

Dentro de estas soluciones de agarre también se evalúan las condiciones de seguridad del sistema, tales como evitar soluciones de agarre fuera del rango de acción del robot, o fuera del perímetro acotado por las barreras de la zona de trabajo.

El algoritmo de funcionamiento del paquete se expone a continuación:

Figura 3.13: Esquema de funcionamiento del paquete de agarre

La condición de "distancia" se basa en calcular el agarre con menos trayecto entre el punto inicial y el objeto a agarrar. Esta condición es útil cuando se trata de agarrar de manera rápida un objeto, tal como podría darse el caso en aplicaciones de clasificación o atrapar objetos en una cinta. La condición de "calidad" antepone la mayor calidad de agarre, y de entre las de mayor o igual calidad de agarre, elige la que tiene menos recorrido de agarre. La calidad de agarre se basa en la cantidad de superficie en contacto con la que entra el objeto y el gripper, por lo que estos tipos de agarres son recomendados para objetos pesados, agarres de precisión u objetos con bajo nivel de fricción.

El paquete publica en el entorno de ROS los resultados para que el brazo robot pueda realizar el agarre. Estos resultados son publicados con un refresco de 0.6 segundos

para evitar un excesivo consumo del procesador, por lo que se recomienda que las otras aplicaciones que dependan de este paquete respeten este tiempo.

3.4 CONDICIONES RESTRICTORAS

Colaborar con otras aplicaciones en el entorno de ROS como MoveIt! o paquetes de visión artificial implica que el sistema tiene una serie de restricciones que tienen que ser tomadas en cuenta. El entorno para el que se modela un espacio hábil de agarre, y la propia disposición del robot, implican una alta cantidad de variables a tener en cuenta en el desarrollo lógico del mejor agarre. En las primeras fases de desarrollo de este proyecto, estas incongruencias sacaban a la luz la necesidad de una serie de cotas o inteligencia lógica para evitar situaciones imposibles o irrealizables.

Soluciones al agarre:

La primera resulta desaconsejable, siendo la segunda mejor

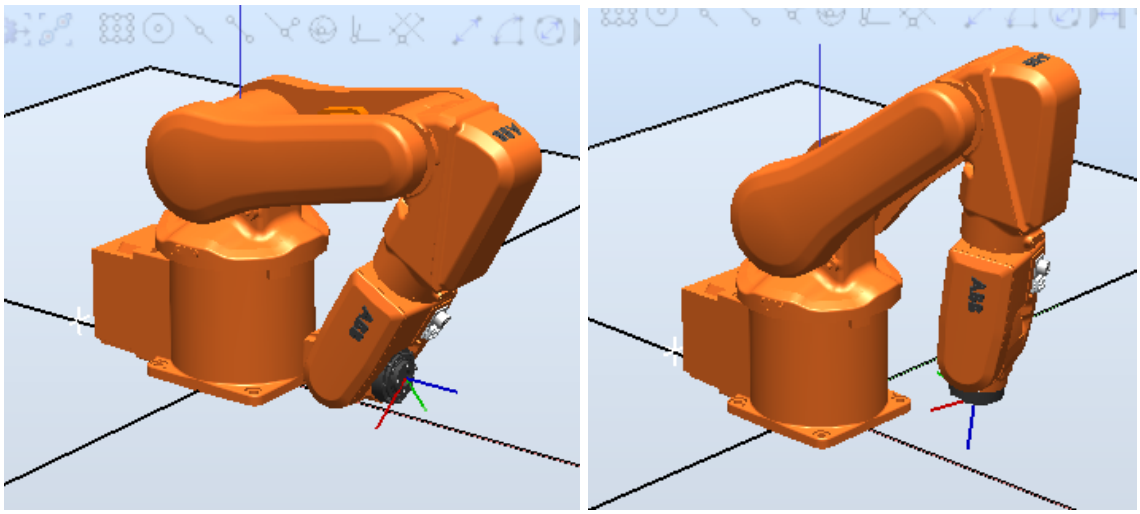


Figura 3.14: Soluciones de agarre simuladas

Como puede observarse, la primera solución, aunque teóricamente tuviera la mayor calidad de agarre, es impracticable. Las articulaciones del robot sufren demasiado y su desplazamiento es muy complejo.

Además, factores como la propia colisión del robot contra si mismo o elementos del tablero deben ser restringidos, puesto que MoveIt! debe asegurarse poder planificar en el entorno.

Es entonces necesario establecer al sistema de una serie de pautas lógicas que puedan discernir entre una solución de agarre realizable y una irrealizable. Además, dentro de las situaciones realizables, tiene que proceder a realizar la situación de agarre más óptima de la que se pueda disponer.

Así pues se han establecido en el paquete de agarre ciertos criterios necesarios para su correcto funcionamiento, partiendo de las siguientes premisas:

-La distancia efectiva de agarre está limitada por el robot (IRB120).

-La herramienta de agarre por defecto es la BarrettHand y su física debe ser considerada.

-El robot está anclado por su pie a una mesa (superficie planar limitada).

-Las objetos en la zona de trabajo con los que pueda haber colisión deben ser respetados y nunca ser colisionados, incluyendo el propio robot.

-Sí y solo sí todas las anteriores condiciones son respetadas, el paquete de agarre debe dar la mejor solución de agarre, respecto a uno de los siguientes criterios:

-Menor distancia para agarrar el objeto.

-Mejor calidad de agarre.

Mencionado esto, se procede a continuación a explicar cómo se ha ido acotando todas las posibles soluciones y sus restricciones.

3.4.1 Distancia efectiva del IRB 120

El robot tiene un límite de rango en el cual puede actuar. Este rango según el fabricante es 580mm, pero es importante recalcar que este alcance es para una orientación del efector concreta. En muchas soluciones de agarre, el efector final necesita adelantarse espacialmente al objeto a aferrar, debido a que la orientación de agarre así lo requiere. Además, debido a que el robot está anclado por su pie en la mesa, su máxima extensión respecto a su frontal no es la misma respecto a la superficie.

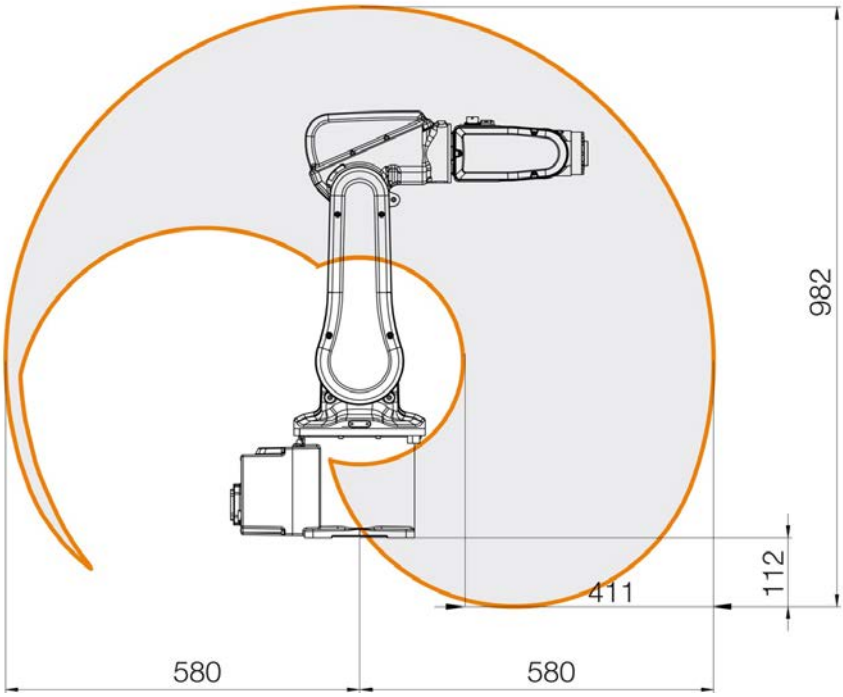


Figura 3.15: Rango efectivo del IRB 120

Este gráfico muestra con claridad la situación de alcance del robot. Dado que la mayor cantidad de objetos a los que se dispone agarrar permanecen apoyados en el plano de la mesa, la primera medida que se ha tomado ha sido reducir el rango efectivo de agarre del robot a 500mm. Esto se debe en parte dado que al incorporar la herramienta BarrettHand el TCP del sistema se desplaza aproximadamente 8 cm ,hay que garantizar la mayor cantidad de soluciones de agarre, y por eso es necesario acotar parcialmente las soluciones que impliquen un desplazamiento del robot más allá de 500mm, donde la situación de la herramienta podría poner al sistema en unas situación de agarre imposible.

Para esta causa, se ha creado un perímetro circular alrededor de la base del robot. Toda solución de agarre que quede fuera de este perímetro, será descartada.

3.4.2 Limitación de borde y radio mínimo efectivo

Existe otra limitación añadida respecto a la zona efectiva de agarre: se crea un borde de frontera de 2cm en paralelo al eje Y del robot, y un radio mínimo efectivo de agarre de 50mm.

La primera restricción evita que haya golpes contra la mampara protectora, de esta forma, ante la situación de agarre de un objeto cercano, el robot situará al gripper dentro del área de la mesa. El criterio de radio mínimo tiene la misma intención que este anterior comentario, pero velando por la propia integridad del robot. Con esta medida se pretende evitar soluciones de agarre desafortunadas, como la mostrada con anterioridad en la introducción a este tema, donde puede existir colisión propia del robot consigo mismo, de la misma forma que pueda haber problemas relacionados con torsión forzada. De forma análoga que para la anterior restricción, el planificador podrá obtener soluciones para objetos próximos a la base, pero solo tomará como soluciones válidas aquellas que sitúen el gripper dentro de la zona de trabajo limitada.

Con estas tres pautas, se ha conseguido crear una zona de trabajo segura y sin conflictos entre módulo planificador de agarre y módulo planificador de movimiento.

La zona de acción queda acotada tal y como se muestra a continuación.

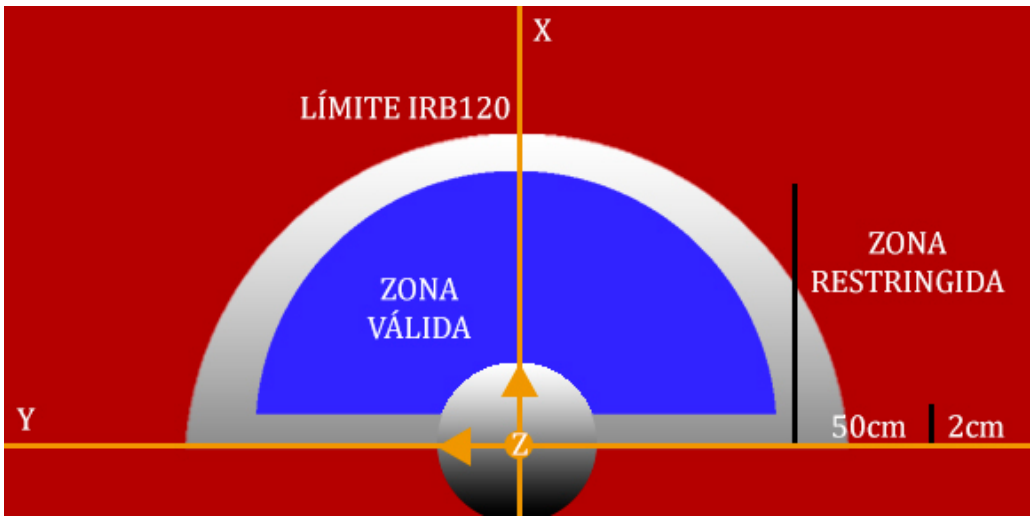


Figura 3.16: Rango delimitado de la zona de trabajo

3.4.3 Replanteamientos al desplazamiento y tiempo de planificación

Este aspecto del sistema depende en su totalidad de MoveIt!. El paquete de agarres diseñado en este proyecto da soluciones a una alta velocidad, por lo que en el aspecto de tiempos, no genera retardos aparentes en la ejecución del sistema. Respecto al planificador de desplazamientos MoveIt!, se han tenido que tomar una serie de pautas para asegurar que el sistema realiza la mejor ruta de desplazamiento del efector para la solución de agarre determinada.

-Habilitar las soluciones de desplazamiento respecto a espacios próximos de agarre:

allowLooking(true);

De esta forma se confiere al planificador una mayor versatilidad respecto a las solucione que pueda otorgar para aproximarse a la zona de agarre determinada. El criterio más importante que toma este comando es la capacidad de poder esquivar con más facilidad objetos de colisión apostados en la zona de trabajo. Esta solución puede hacer que el robot se desplace de manera impredecible a lo largo del entorno de trabajo hasta llegar a la situación de destino, por lo que es muy importante que todo elemento que se establezca dentro de la zona de trabajo aparezca reflejado en el entorno virtual del planificador como elemento de colisión.

-Permitir re-planificar los desplazamientos, al menos, una vez más:

allowReplanning(true);

setNumPlanningAttempts(2);

Un inconveniente que se ha reiterado a lo largo del desarrollo del proyecto ha sido la negativa del planificador a realizar un desplazamiento con orientación compleja, debido a que necesitaba de varios movimientos para poder colocar el efector final sin entrar en colisión con el propio objeto a agarrar u el propio brazo robot. Dado que la solución de agarre debe ser respetada, es necesario darle un margen de libertad al planificador de desplazamiento para que al final, coloque el efector en la situación deseada. Se ha comprobado que con la configuración anteriormente establecida, de manera directa o después de una serie corta de intentos fallidos, MoveIt! consigue encontrar una solución de desplazamiento válido para el robot.

-Tiempo de planificación del desplazamiento:

setPlanningTime(5);

A un mayor tiempo de planificación se consigue una mayor calidad en la ruta de desplazamiento del efector. Debido a la necesidad de asegurar una planificación válida, eficaz y funcional, se ha determinado que el tiempo máximo para poder establecer una solución de agarre sea 5 segundos. De esta forma, el planificador puede asegurarse de poder realizar agarres complejos que, por defecto (1 segundo de máxima planificación), podrían resultar en intentos fallidos.

3.4.4 Limitación angular de seguridad

Debido a que las soluciones de agarre pueden llegar a necesitar varios desplazamientos angulares para lograrse, y que una mala planificación de estos giros puede forzar mecánicamente los motores del robot, se han establecido una serie de limitaciones angulares, otorgando cierto margen de grados de giro respecto al máximo del robot. Este aspecto es relevante cuando se usa un robot real, dado que los esfuerzos angulares y torsiones no son evaluados.

3.5 ARQUITECTURA EN ROS

3.5.1 Suscripción

Respecto a los topics usados por el paquete el nodo de agarre está suscrito a:

-La posición inicial del robot

(geometry_msgs::Pose) , del cual solo se utiliza la información de la posición.

-La condición de agarre

(std_msgs::Bool) , TRUE para "calidad", FALSE para "distancia"

-La pose del objeto detectado por visión artificial, y su indicador geométrico.

(geometry_msgs::Pose) : situación espacial del objeto a agarrar

(std_msgs::Int16) : PRISMA="1", ESFERA="2", CILINDRO="3"

3.5.2 Publicación

El nodo de agarre publica:

-El ángulo de los dedos del gripper: pick_fingers

(geometry_msgs::Pose) : guardados en las variables de orientación, para la BarrettHand:

-pick_fingers.orientation.x = Ángulo del dedo F1

-pick_fingers.orientation.y= Ángulo del dedo F2

-pick_fingers.orientation.z = Ángulo del dedo F3

-pick_fingers.orientation.w = Ángulo del giro de la palma F4

Para grippers con más efectores móviles, pueden usarse las variables de posición de la Pose publicada para almacenar más valores.

-La solución de pose del robot: pick_pick

(geometry_msgs::Pose) : este resultado concatena la unión del resultado transformado mediante la visión artificial y la solución de GraspIt!, por lo que puede tomarse como la situación final a la que tiene que desplazarse el brazo robot para agarrar el objeto final.

Los campos relacionados con esta pose son los típicos para desplazar el robot:

-pick_pick.position.x = Desplazamiento respecto al eje X

-pick_pick.position.y = Desplazamiento respecto al eje Y

-pick_pick.position.z = Desplazamiento respecto al eje Z

-pick_pick.orientation.w = Componente natural del cuaternio de orientación.

-pick_pick.orientation.x = Componente "i" del cuaternio de orientación.

-pick_pick.orientation.y = Componente "j" del cuaternio de orientación.

-pick_pick.orientation.z = Componente "k" del cuaternio de orientación.

-La solución de pose del efector transformada, sin añadir el desplazamiento relativo debido a la visión artificial: pick_offset

Esta pose se publica a modo de carácter adicional de la aplicación. En aplicaciones que necesitan realizar más desplazamientos que el propio del agarrar un objeto, es necesario especificar la transformación que se ha realizado entre sistemas. De esta forma, mediante la suma de las posiciones relativas y la solución otorgada a modo de "offset", se puede desplazar el robot a la posición indicada, sin ser necesaria la aplicación de una visión artificial.

Los campos relacionados con la pose son equivalentes a los comentados para el topic pick_pick.

De esta forma, si todos los topics suscritos son recibidos correctamente, el paquete de agarre publica de manera automática soluciones de agarre para el objeto capturado por la visión artificial, teniendo en cuenta la situación inicial del robot para otorgar la solución más adecuada.

El paquete debe ser iniciado en un terminal de Ubuntu bajo la jurisdicción del nodo ROSCORE, y una vez que detecte datos de la visión artificial, este empezará a calcular soluciones, sin interrupción, hasta que el usuario decida acabar con la ejecución del nodo.

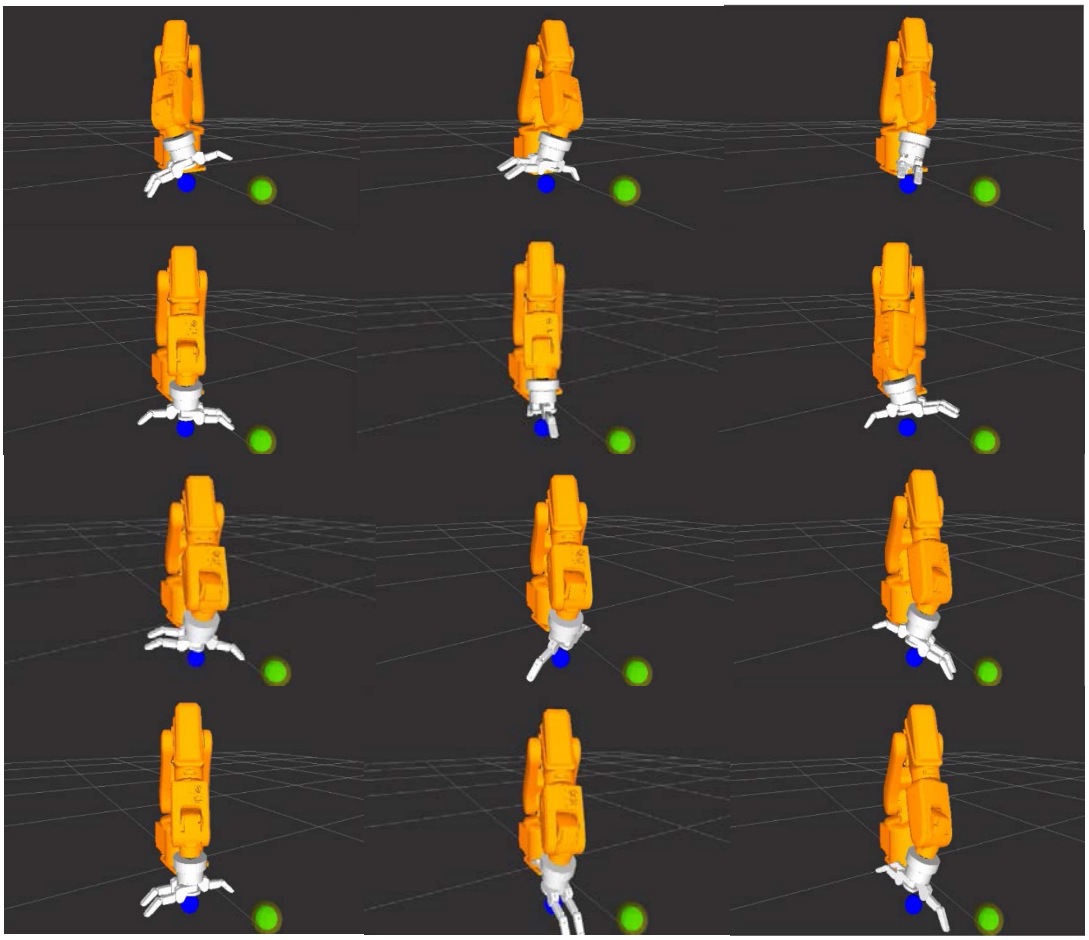


Figura 3.17: Ejemplos de solución aportados por el paquete para agarrar una esfera

CAPÍTULO 4

APLICACIONES

4.1 INTRODUCCIÓN

A continuación se exponen las aplicaciones realizadas para operar de manera conjunta con el paquete de agarre. Es necesario destacar que todas las aplicaciones realizadas resultaron funcionales dentro del entorno simulado. La herramienta BarrettHand con la que se pretendía trabajar a lo largo del proyecto resultó estar averiada, tanto en control informático como estructuralmente, puesto que el dedo F3 estaba inutilizado por completo. Dado que la BarrettHand es el modelo de gripper más polivalente, con mayor número de efectores móviles que la pinza convencional, y ser el único modelo de robot por el cual GraspIt! permitía poder planificar sobre agarres de manera rápida y personalizada, se optó por mantener el desarrollo del proyecto sobre el esquema inicial, con la salvedad de que todo se realizó de forma simulada dentro del entorno de ROS MoveIt!.

Es importante destacar que otros grippers pueden ser compatibles para ser computados dentro del paquete de agarre, pero necesitan de una base de datos de agarres personalizada para poder ser usados.

Dejando claro este punto, se procede a mostrar las aplicaciones relacionadas con el paquete de agarre:

4.2 APLICACIÓN PARA CLASIFICACIÓN DE OBJETOS A NIVEL INDUSTRIAL

Los robots de clasificación son usados ampliamente en el panorama industrial debido a que permiten operar a velocidades para las cuales un humano no podría responder de manera prolongada, ya sea por reflejos o porque resultaría en un trabajo físico abrumador.

Figura 4.1: Robots de clasificación industrial

En el trabajo conjunto de *Alejandro Gómez Rubio* [1], *Alberto Lázaro Enguita* [2] y *Javier Moriano Martín* [3] se desarrolló una aplicación de calificación de objetos para un entorno de trabajo estático. En dicho proyecto, el proceso de agarre de los objetos

era único para todos los objetos: posicionarse en la vertical del objeto a una altura lo suficientemente alta como para no chocar la herramienta contra el objeto, descender el efector hasta un punto lo suficientemente cercano con el objeto para poder agarrarlo, y proceder a cerrar los dedos de la BarrettHand según fuera el objeto detectado.

En este proyecto, gracias al nuevo paquete de agarre, es posible mejorar dicha aplicación: mediante una petición directa sobre el paquete de agarre, el sistema puede desplazarse directamente desde la situación de reposo a la situación de agarre, en menos tiempo, de manera más eficiente y dando la posibilidad de introducir condicionantes al agarre, como menor distancia para realizar el agarre, o máxima calidad.

Debido a la filosofía del robot calificador, cuya principal virtud es la capacidad que tiene para realizar tareas de forma veloz, se ha optado por adoptar la condición de "mínima distancia de agarre" en la mejora de esta aplicación, de tal forma que el robot pueda cumplir con la mayor cantidad de objetivos en el menor tiempo.

La aplicación de clasificación tiene el siguiente esquema de funcionamiento:

Figura 4.2: Diagrama de la aplicación de clasificación

El paquete de visión aporta la situación espacial y la forma del objeto al que se dispone el agarre. Esta información es tramitada por el paquete de agarre, que planea una solución de agarre válido una vez reciba la situación de partida del robot. Mediante el paquete de control de la mano robótica, los ángulos aportados por la solución de agarre se articulan. Para evitar posibles impactos a lo largo del recorrido, los dedos son plegados relativamente, y al llegar a la situación de agarre, son articulados por completo con los valores angulares requeridos. Las órdenes de desplazamiento son enviadas mediante el Socket de control al IRB120, que mediante una planificación de rutas en MoveIt!, se desplaza por la zona de trabajo. El desplazamiento realizado al trasladar el objeto entre el punto de agarre y el punto de clasificación se realiza mediante desplazamiento cartesiano para garantizar la integridad del objeto durante el movimiento planificado.

Una vez apoyado el objeto la rutina de clasificación se reinicia y el sistema vuelve a esperar a que un objeto conocido entre dentro de la zona de trabajo.

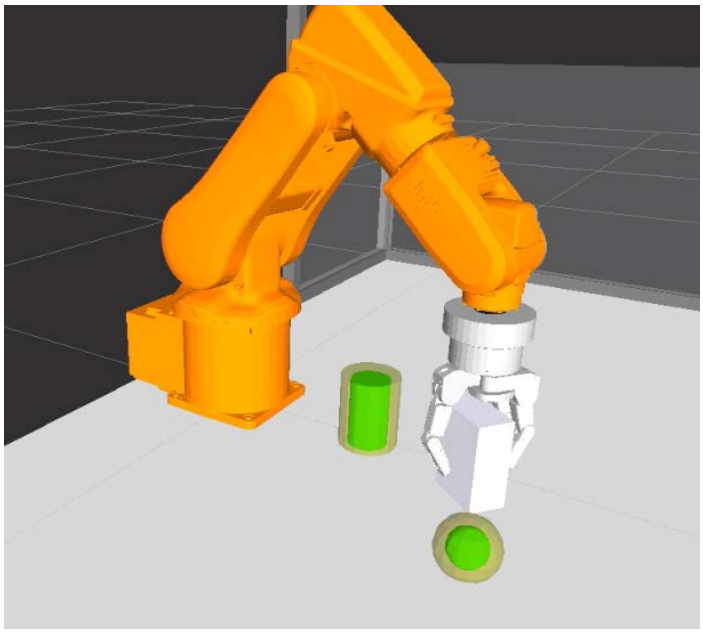


Figura 4.3: Solución al agarre del brick de leche

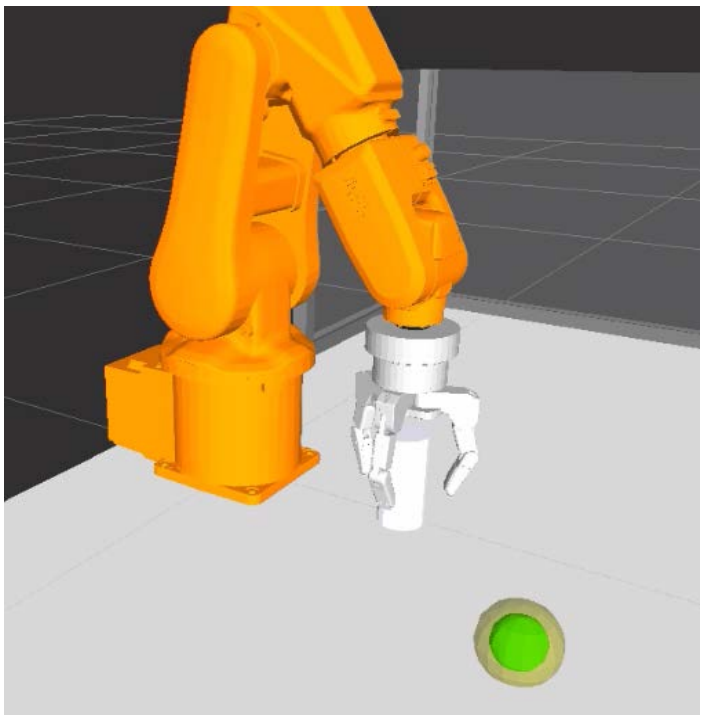


Figura 4.4: Solución al agarre de la lata de refresco

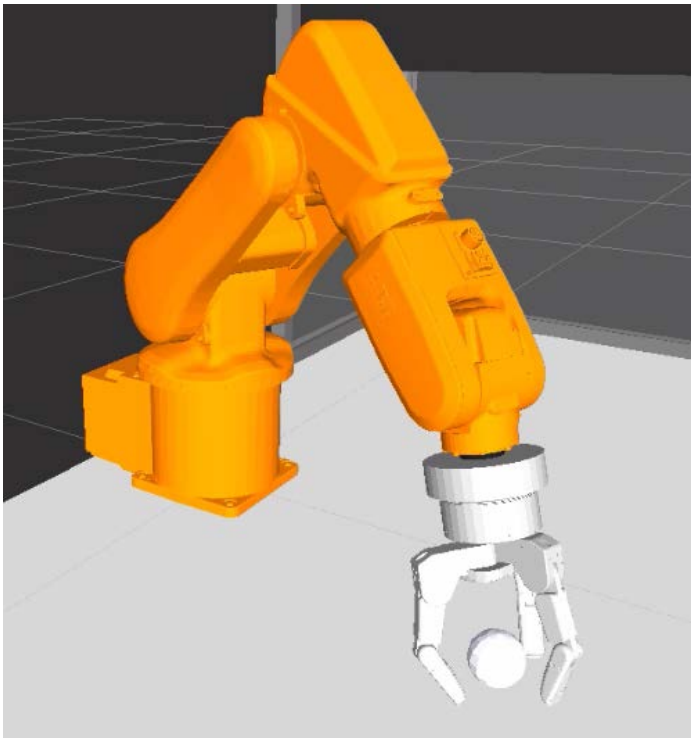


Figura 4.5: Solución al agarre de la pelota de tenis

Un detalle a destacar respecto al paquete de agarre es su relación con el paquete de visión artificial. En la anterior imagen, se puede apreciar que no existe contacto entre la herramienta BarrettHand y la pelota, aun cuando se había simulado específicamente la base de datos para una esfera de mismas dimensiones. Esto es debido a la representación en el entorno de Rviz y la información obtenida por la visión artificial. La representación de los objetos reales difiere a su relación respecto al entorno simulado, debido a que en este, su representación es extraída de la forma primitiva del objeto, por lo que esta puede ser interpretada por el paquete de visión con un menor volumen. No obstante, en el entorno real, la aplicación aferraría el objeto tal y como se ha simulado dentro de GraspIt!.

Otra cualidad que permite el paquete de agarre es la capacidad de expandir la zona de trabajo. En cadenas industriales, donde los robots han sido implementados como medida para modernizar la planta, reconstruir infraestructura relacionada con cintas mecánicas o palés para adecuarlos al robot sería un coste añadido a la instalación de este. Dado que mediante el paquete de agarre ahora se pueden albergar agarres oblicuos, como en el ejemplo al agarre de la pelota de tenis, el sistema puede llegar a agarrar objetos a distancias más lejanas y más complejas que para la primera versión de la aplicación.

La gran mayoría de soluciones de agarre aportadas por el paquete de agarre, debido a la condición consensuada de “mínima distancia”, han sido agarres superiores a los objetos, tal y como se esperaba. La implementación de un paquete específico para el planteamiento del agarre ha agilizado la velocidad con la que se clasifica, siendo 4 segundos por objeto, de media, más rápido que la primera versión de la aplicación. Este factor de mejora es deseable en la mayoría de aplicaciones de clasificación, puesto que el tiempo es limitado y la cantidad de objetos a clasificar es elevada. Uno

de las mayores repercusiones que puede tener el factor de velocidad es la capacidad de evitar tener que disponer de robots adicionales para clasificar un proceso. De esta forma, si un único robot es capaz de realizar toda su carga de trabajo él solo, se evita la compra, mantenimiento y suministro de robots adicionales que complementen su trabajo.

4.3 APLICACIÓN DE LABORATORIO ROBOTIZADO

Esta aplicación integra un robot dentro del espacio de trabajo de un laboratorio a modo de herramienta de seguridad entre operario y ensayo. Mediante el paquete de agarre desarrollado, el robot puede manipular probetas o vasos de precipitados, de forma que estas puedan ser vertidas de forma remota, estableciendo una barrera de contención adicional respecto al operario.

La máxima prioridad a la que se atiende para implementar un brazo robot en el entorno de trabajo es la seguridad del operario: Los dos campos científicos donde esta aplicación podría ser más relevante serían en la industria química y en la investigación biológica.

4.3.1 Investigación biológica

En el entorno de la investigación médica y biológica el mayor peligro respecto a seguridad es el factor de riesgo biológico. Dependiendo del nivel de riesgo, los laboratorios son asociados a un nivel de riesgo, desde riesgo mínimo en nivel 1, hasta riesgo extremo en nivel 4. Estos niveles están recogidos dentro del Real Decreto 664/1997 de la jurisdicción española [15]. En función del riesgo al que se asociada un laboratorio este se equipa con mayor cantidad de barreras internas y su protocolo de uso es más estricto. Un pequeño resumen los factores a los que se exponen los trabajadores en su entorno:

NIVEL 1	E. coli, varicela, Hepatitis canica, Bacillus Subtilis
NIVEL 2	Hepatitis B, hepatitis C, gripe, salmonelas, VIH
NIVEL 3	Ántrax, viruela, tuberculosis, tifus, dengue
NIVEL 4	Fiebre hemorrágica del Ébola, virus de Lassa

Tabla 4.1: Riesgos biológicos

Figura 4.6

Desde el punto de medidas de seguridad para el operario, evitar el contacto directo con la piel y la inhalación sigue siendo la prioridad para evitar propagaciones y enfermedades relacionadas.

Desde el aspecto funcional, al igual que para la industria química, es deseable la condición de esterilidad del medio al ensayar: de esta manera se consigue que los ensayos no sean influenciados por condiciones externas, ni que el entorno de trabajo se corrompa con las pruebas. El brazo robot como herramienta permite ser esterilizado, lo que facilita su integración dentro del medio de trabajo.

4.3.2 Industria química

Dentro de un laboratorio químico existe peligro real por el operario al manejar reactivos. Estos componentes pueden resultar perjudiciales por separado, permanecer en el entorno de trabajo como partículas en suspensión, producir gases tóxicos, etc. Debido a que el operario realiza manualmente la mayor parte de su trabajo [16], es importante saber el riesgo asociado a la manipulación de cada sustancia.

Los factores de riesgo más comunes se listan a continuación:

	Corrosivo Causa destrucción de tejidos vivos y/o materiales inertes. Evitar inhalación, contacto con la piel, ojos y ropas.
	Explosivo Sustancias y preparaciones que pueden explotar bajo efecto de una llama. Evitar que se golpee, sacuda, friccione o acerque a fuentes de calor.
	Comburente Sustancias que tienen la capacidad de incendiar otras sustancias. Evitar que contacten con materiales combustibles.
	Inflamable Esta propiedad agrupa mayoritariamente a sustancias que tienen facilidad para ser prendidas mediante una reacción química. Evitar su contacto con materiales ignitivos.
	Tóxico Sustancias y preparaciones que, al ser absorbidas por el organismo, pueden implicar riesgos graves, agudos o crónicos de salud. Evitar el contacto con el cuerpo humano.
	Nocivo Sustancias y preparaciones que, al ser absorbidas por el organismo, pueden implicar riesgos a la salud de forma temporal o alérgica. Evitar la inhalación de los gases y su contacto con la piel.

Tabla 4.2: Riesgos químicos

Un factor común de seguridad es constante respecto a los anteriores peligros: se debe evitar la inhalación y el contacto directo con la piel. En la actualidad, los laboratorios cuentan con entornos aislados con el exterior, de modo que el operario pueda realizar ensayos controlados.

Figura 4.7: Barreras de protección convencionales para experimentación química

Existen factores propios de la infraestructura que dificultan el éxito en el ensayo y que exponen al operario a peligro: los componentes son introducidos manualmente por el usuario, el espacio es limitado, existe el impedimento propio de la infraestructura de trabajo: altura de la mesa, guantes sensibles a ser afectados por el ensayo llevado a cabo, esterilidad del medio, entre otros.

Figura 4.8

El caso más extremo entre corrupción del espacio de trabajo y riesgo del operario es debido a elementos radioactivos. Estos elementos pueden alterar de forma grave su entorno, con capacidad de producir cáncer o alteraciones celulares.

Una posible implementación de un sistema real respecto al propuesto en este proyecto, sería establecer el sistema robótico dentro de un entorno con barreras de hormigón laminados de plomo, y controlar todo el proceso mediante el módulo de visión artificial.

La implementación del brazo robot dentro del entorno controlado del laboratorio químico mejoraría las prestaciones de seguridad respecto al operario, puesto que se podrían realizar tareas de ensayo sin que el operario interviniese personalmente en el proceso, evitando los riesgos por inhalación o contacto. Además cabe destacar que el brazo robot se ha convertido a lo largo del tiempo en una herramienta precisa, por lo que puede abordar aplicaciones donde se requiera cierto nivel de pulcritud.

Planteando los aspectos de seguridad recogidos por las anteriores disciplinas, se ha simulado el entorno de trabajo de un robot operador en un laboratorio genérico, cuya misión es verter el contenido de un recipiente dentro de un vaso de precipitados.

Para ello, la aplicación detecta el tipo de recipiente de laboratorio que se ha colocado encima de la mesa, lo agarra, y lo trasfiere hasta la situación de vertido. Una vez ha finalizado, vuelve a dejar el recipiente en su sitio y se queda a la espera de volver a reiniciar la rutina.

El esquema de ejecución respondería al siguiente diagrama:

Figura 4.9: Diagrama de la aplicación para el laboratorio robotizado

Una vez la aplicación está en marcha, el brazo robot se desplaza a una posición de reposo concreta para indicar que la aplicación está en ejecución, a la espera de un objeto conocido adquirido por la visión artificial.

El proceso empieza detectando el recipiente correspondiente a ser agarrado. Debido al carácter normalizado de las dimensiones de los recipientes, bastaría con generar el objeto simulado en GraspIt! y extraer la base de datos de sus agarres, siguiendo los pasos previamente mencionados. De esta forma, mediante ficheros .txt se podría dar cabida a todos los instrumentos de trabajo del laboratorio. Para este proyecto se utiliza la base de datos obtenida para la lata de refresco de la aplicación anterior, puesto que tiene similares dimensiones a un vaso de precipitados genérico. En esta aplicación el paquete de agarre utiliza la condición de “calidad”, por varios motivos:

-<u>Baja fricción del vidrio:</u> Para evitar que el recipiente resbale debido a la fuerza ejercida por su contenido.

-<u>Precisión de la maniobra:</u> Es esencial que el líquido no se vierta por accidente al ir a verterlo, por lo que es necesario que la rotación entre la mano robótica y el vaso sea lo menos excéntrica posible.

<u>-Posibilidad de verter:</u> Todas las soluciones de agarre aportadas por el paquete de agarre son válidas, pero no todas valen para verter el contenido de un recipiente. Esta resolución se manifestó para las soluciones de agarre superiores al cilindro. Dado que es necesario que el robot pueda realizar un agarre de confianza desde y hasta cualquier punto del entorno de trabajo, la condición de calidad, para la cual respecto al cilindro, todas las soluciones eran de agarre lateral, resultó ser la opción más lógica.

Una vez el objeto ha sido identificado, una espera de 5 segundos permite verificar al operario que se ha detectado correctamente, y sirve a modo de espera para que si el operario lo desea, el sistema entre en modo manual.

4.3.2.a Rutina opcional de la aplicación ROBLAB–Modo manual

El modo manual es un primer paso para demostrar cómo pueden cooperar máquina y operario para desempeñar una tarea de precisión de manera remota. Aplicando la filosofía de aprendizaje usada para generar una base de datos de agarres, el control directo del operario, y la seguridad facultativa del control telemático, se pretende así que el brazo robot pueda ser usado como una extensión al operario.

El medio propuesto en este proyecto para lograrlo es mediante el teclado. Usándolo, el operario puede desplazar el brazo robot a lo largo de la zona de trabajo libremente. El sistema solo restringe al usuario cualquier movimiento que lleve al sistema a colisionar de manera deliberada.

Mediante el paquete KEYBOARD de la página de ROS, se han registrado dentro de la aplicación ROBLAB una serie de teclas para dicho control, de tal modo, que si mientras se realiza la espera de 5 segundos se presiona una tecla, la aplicación se queda a la espera de la pulsación de las teclas para realizar los desplazamientos adecuados. Para ello previamente es necesario haber lanzado el paquete "*keyboard*", y es necesario que para que las pulsaciones sean detectadas, estas deben ser producidas dentro de la interfaz emergente que se despliega al ejecutarlo.

En modo manual, se ha optado por componer las órdenes más básicas para controlar el robot dentro del entorno de trabajo. Los controles se muestras a continuación:

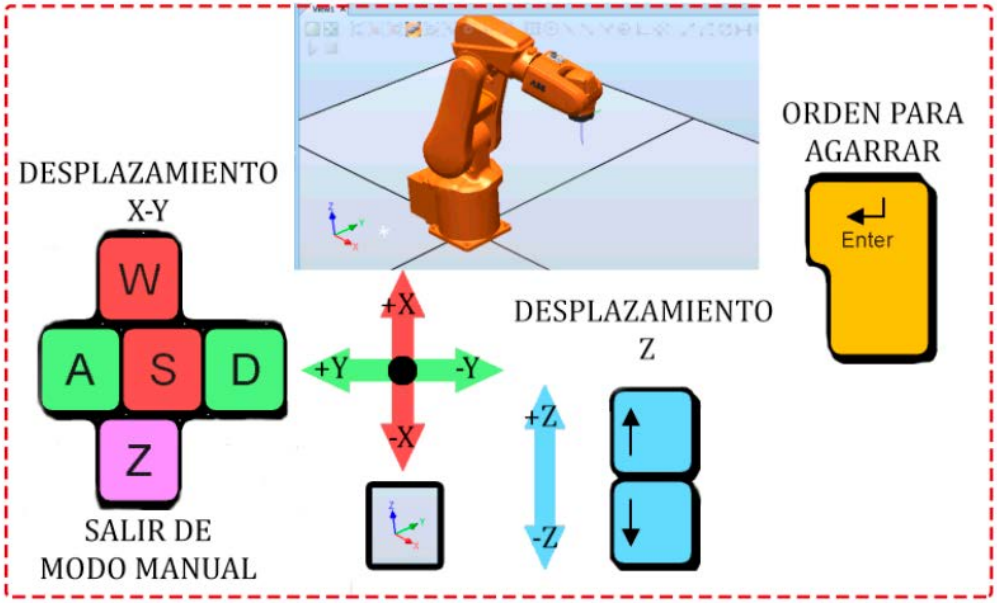


Figura 4.10: Controles para modo manual

Cuando se da la orden para agarrar, el paquete de agarre obtiene la nueva situación de partida para realizar el agarre, y el sistema sale del modo manual.

4.3.2b Continuación del proceso de rutina

Una vez se sale del control manual, o por el contrario, el sistema pasa 5 segundos sin recibir una orden por teclado, la aplicación publica la situación del robot, que es usada por el paquete de agarre para otorgar una solución de agarre. Entonces, el robot se desplaza hasta la situación otorgada.

Una vez que las articulaciones de los dedos aferran el recipiente, el robot pasa a ser portador del recipiente. Por motivos de seguridad y para preservar la integridad de la mesa de trabajo, todos los desplazamientos que realiza el robot mientras es portador son de carácter cartesiano. Esto evita que el planificador de desplazamiento trace malas planificaciones, tales como rotar la muñeca mientras se desplaza o moverse de manera errática por el entorno de trabajo. De esta forma, al moverse de forma cartesiana, el robot mantiene su orientación terminal, por lo que evita las agitaciones del recipiente, y realiza un desplazamiento preciso.

Una vez ha llegado al punto para verter, este es, encima del vaso receptor, a una altura lo suficientemente alta como para no colisionar entre ellos, el recipiente se vuelca con una rotación respecto a la normal de 163º, de forma que se asegure que todo el líquido es vertido y que el caudal es continuo para evitar salpicaduras.

Pasado un tiempo, por defecto 5 segundos, el sistema endereza el vaso de precipitados, supuesto ya todo su contenido vertido, y vuelve a desplazarlo cartesianamente hasta su posición inicial. La variación de peso respecto al recipiente lleno o vacío puede desplazar ligeramente la posición de apoyo, siendo por norma general un factor despreciable.

La rutina finaliza depositando el recipiente en su situación original, y devolviendo el brazo robot a su situación de reposo desplazada, para seguir indicando que su rutina sigue en marcha. El sistema se reinicia en 20 segundos, tiempo suficiente como para que el operario pueda extraer o añadir a la zona de trabajo nuevas herramientas de forma ágil. También se puede cancelar el nodo en ejecución en este instante sin peligro para el usuario de desplazamiento del robot.

Los pasos realizados por la aplicación se muestran a continuación mediante las capturas realizadas durante su ejecución:

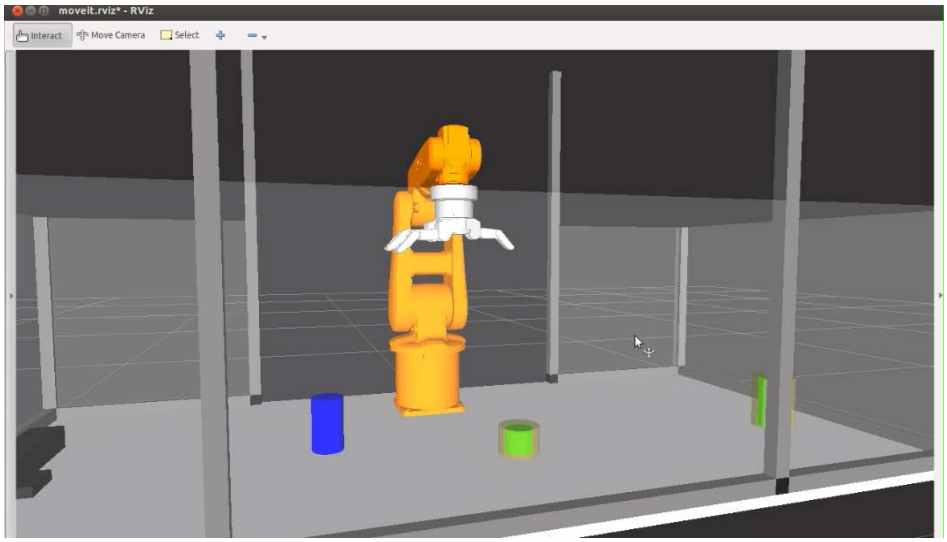


Figura 4.11: Situación de reposo primaria

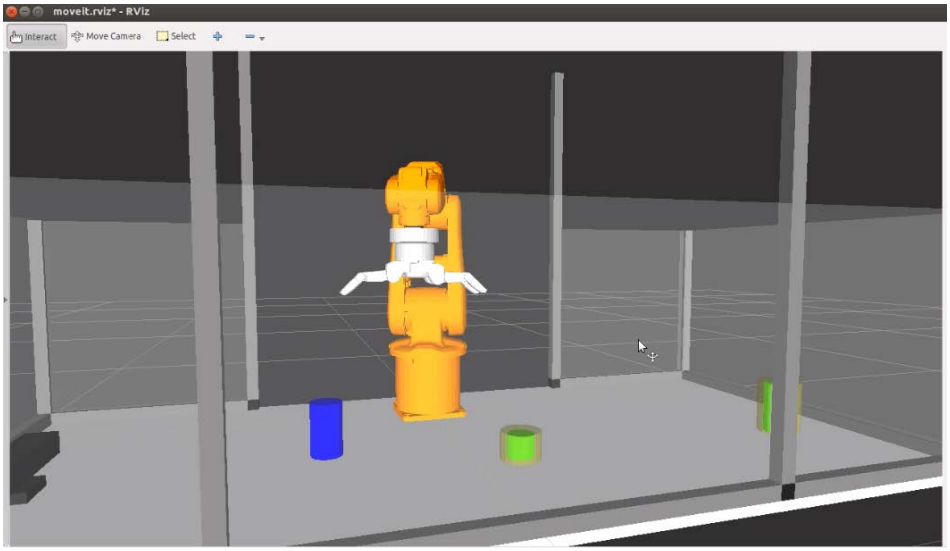


Figura 4.12: Sistema en espera, segunda situación de reposo

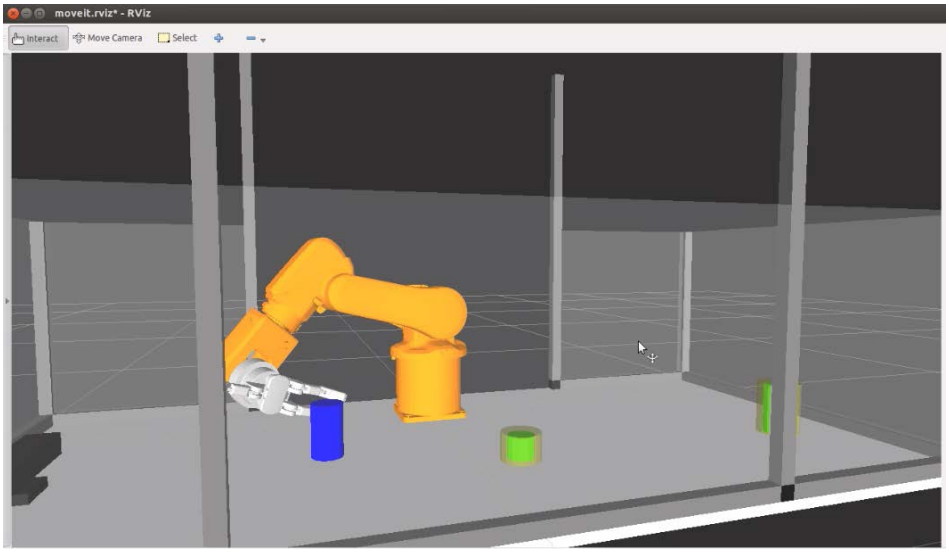


Figura 4.13: Detección del recipiente, aproximación

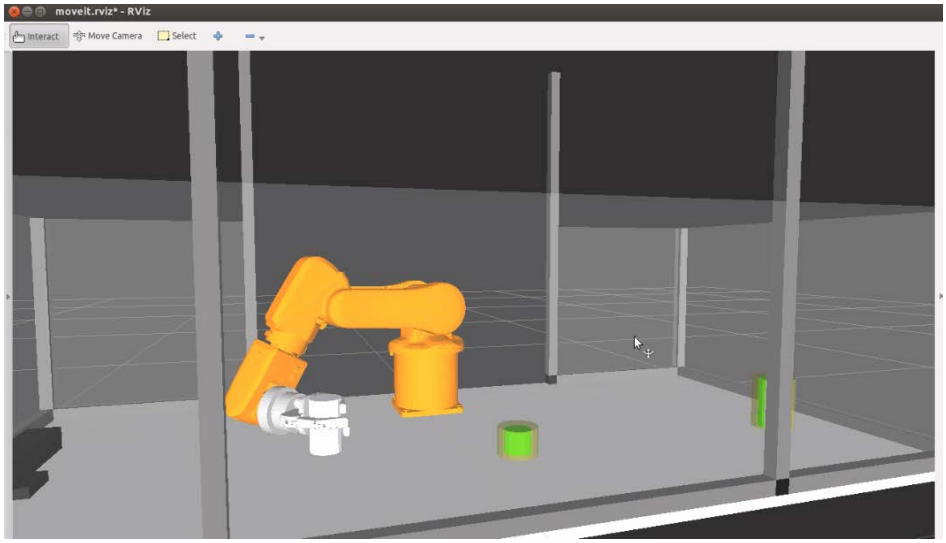


Figura 4.14: Agarre de máxima calidad al recipiente

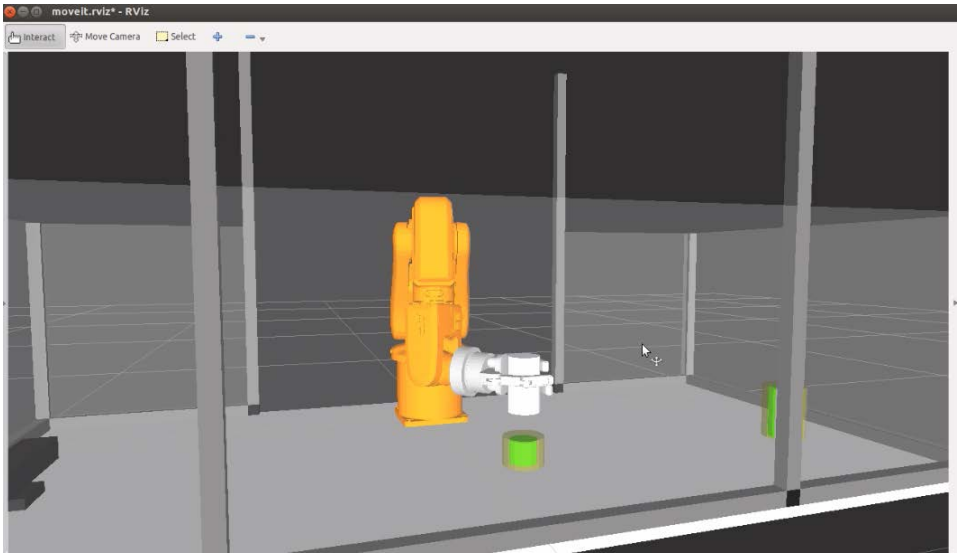


Figura 4.15: Desplazamiento cartesiano hasta el vaso

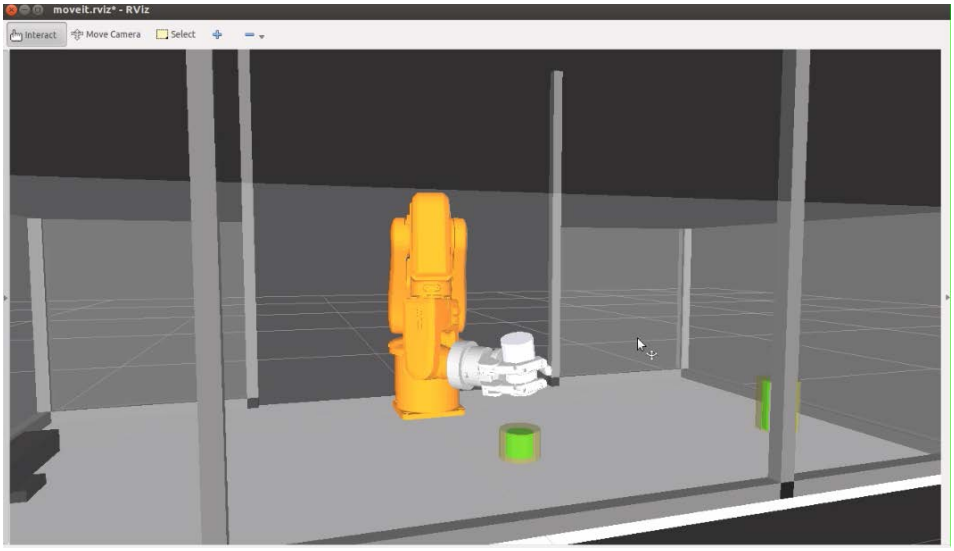


Figura 4.16: Vertido del recipiente en el vaso

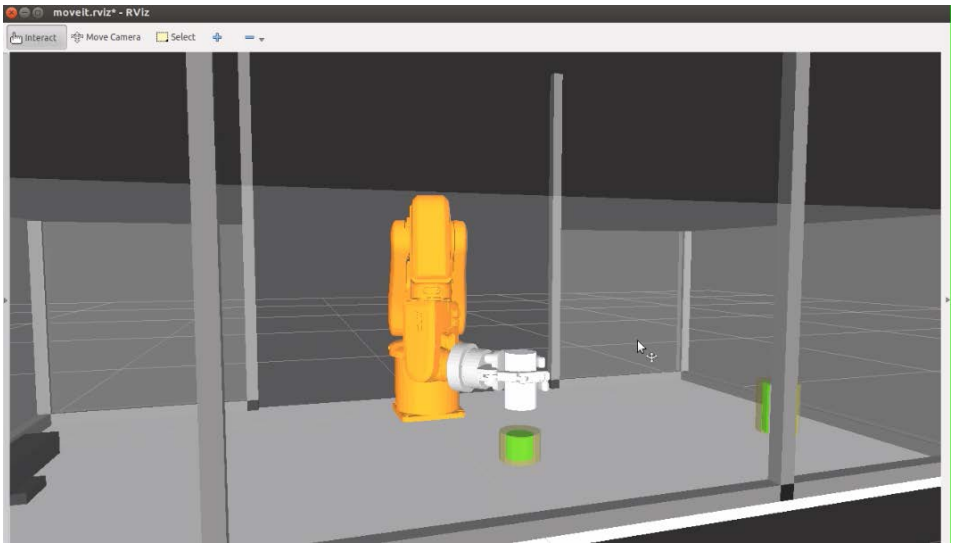


Figura 4.17: Fluido vertido, enderezar el recipiente

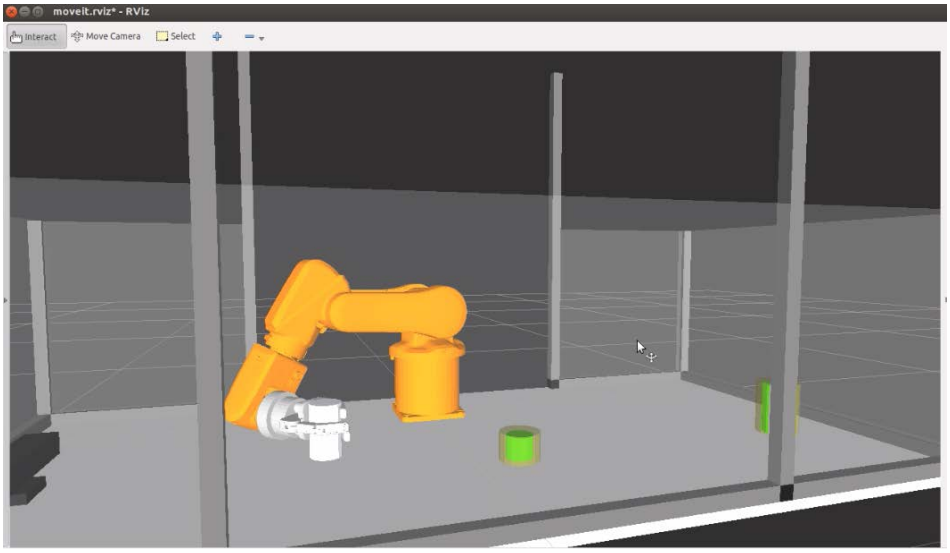


Figura 4.18: Desplazamiento cartesiano del recipiente a su situación inicial

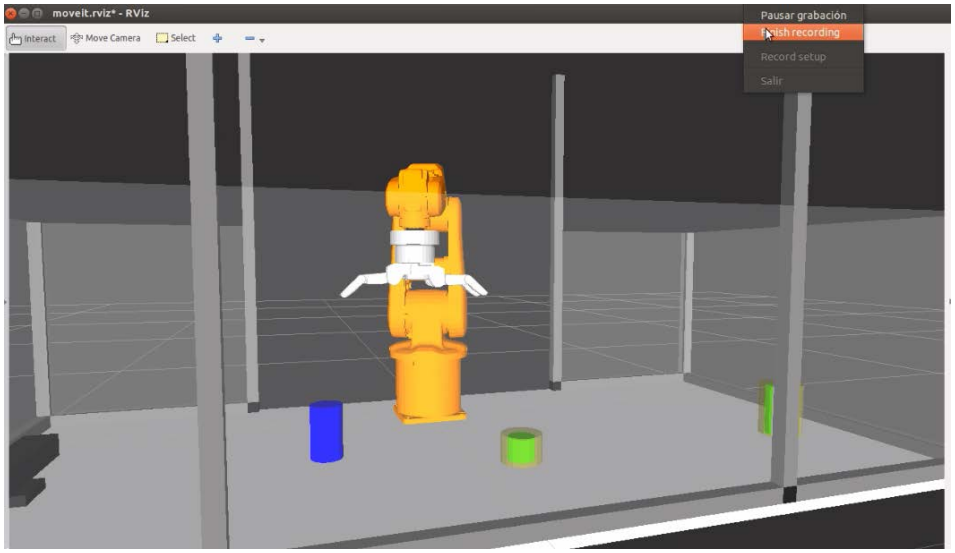


Figura 4.19: Fin de la rutina del laboratorio

CAPÍTULO 5

RESULTADOS Y CONCLUSIONES FINALES

5.1 RESULTADOS

-Evaluación de la funcionabilidad paquete de agarre -

La concepción de la base de datos y el estudio previo de los planificadores de agarre asociados a GraspIt! ha demostrado la relación entre tiempo y esfuerzo para generar soluciones. Los resultados de la base de datos han sido lo suficientemente variados como para permitir al sistema poder responder de manera óptima y fluida.

El desarrollo de la base de datos ha demostrado ser funcional para esfera, cilindros y prismas apoyados por su base.

Debido a su fuerte carácter matemático para obtener la solución de agarre más acertada, para la mayoría de simulaciones, ante un mismo objeto para una situación espacial concreta se correspondió con la misma solución de agarre invariante a la cantidad de simulaciones realizadas.

La probabilidad de reiteración de solución de agarre ante un caso concreto en el 99% de los casos.

-Respuesta del paquete de agarre para las aplicaciones realizadas -

Dada la situación frontal del brazo robot respecto al entorno de trabajo, para el conjunto de agarres realizados, las soluciones con orientación comprendidas entre -45º y +45º respecto al eje Z fueron las más comunes, puesto que tanto para las dos condiciones de agarre diseñadas se valoraba la distancia mínima entre punto de agarre y base del robot.

La distribución de las soluciones aportadas respecto al eje Z muestra una respuesta de distribución gaussiana.

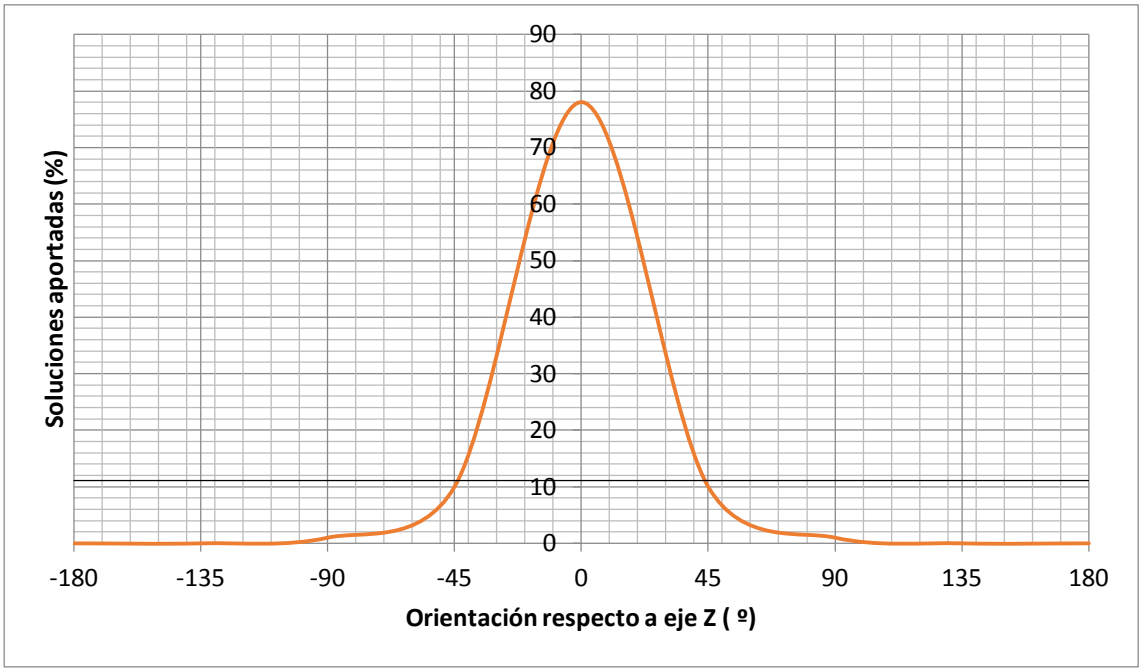


Figura 5.1: Orientación de soluciones aportadas para la clasificación respecto el eje Z

Debido al carácter de la simulación, no se pudo evaluar el par de agarre ejercido por los dedos de la BarrettHand para los objetos aferrados. Respecto a su disposición, los dedos efectores cumplieron respecto a la simulación en GraspIt!, rodeando de forma correcta las formas dispuestas al agarre en el entorno de trabajo.

La herramienta BarrettHand ha demostrado, mediante simulación una buena respuesta angular. Queda a expensas de ser realizado un test de fuerzas que evalué el par de agarre.

A lo largo del desarrollo de este proyecto, se ha percatado que es complejo alterar la naturaleza del planificador MoveIt! para planificar desplazamientos que no entren en conflicto de manera indirecta con la solución concreta de agarre. Una pequeña cantidad de simulaciones resultaron en colisiones fortuitas, es decir, colisiones entre herramienta y elementos del entorno de trabajo que no son percibidos por el planificador y que, en una situación real, harían fracasar a la aplicación. Puesto que este aspecto depende del paquete de agarre, se hace alusión al total del éxito de agarre.

La probabilidad de éxito para obtener un agarre válido desde el paquete de agarre ronda el 95% de los casos.

Respecto a la calidad de los agarres realizados, ha sido evidente una clara relación entre velocidad de agarre y calidad. La condición de "mínima distancia" generó soluciones de agarres con bajo factor de calidad y rápida ejecución, mientras que los específicos para "máxima calidad" tuvieron un alto factor de calidad y una velocidad promedio más lenta hasta llegar a la situación de agarre.

Dicha velocidad promedio es un valor orientativo, puesto que ROS no es un sistema en tiempo real.

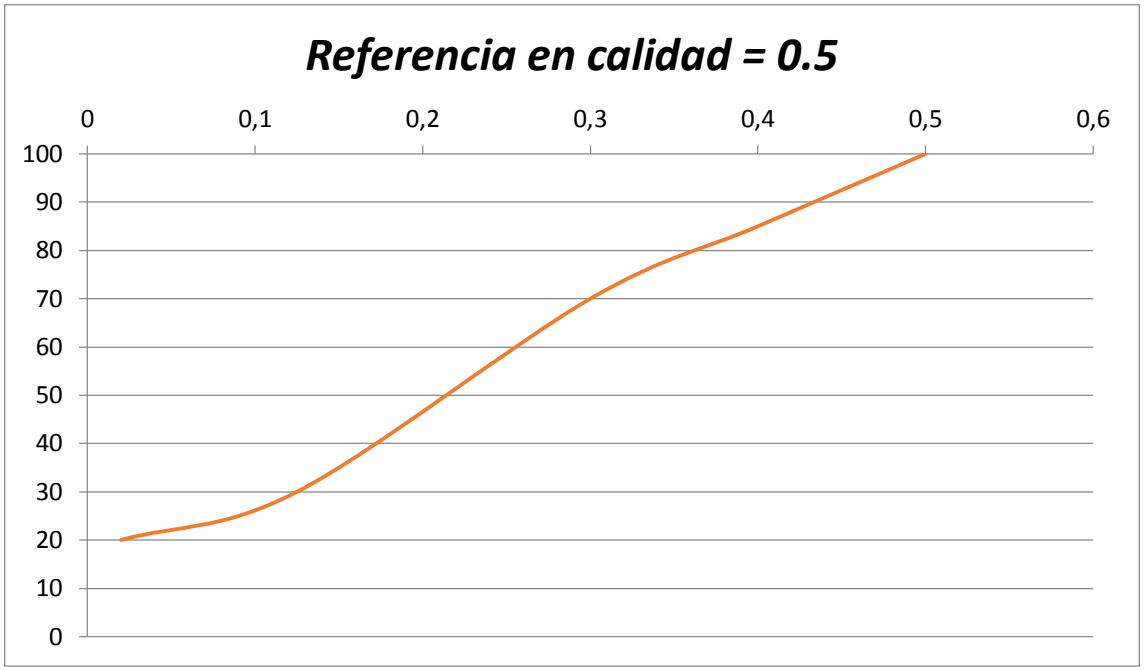


Figura 5.2: Tiempo empleado (%) Vs. Calidad de agarre

5.2 CONCLUSIONES

El paquete de agarre ha resultado ser práctico y funcional en todas las situaciones expuestas para las aplicaciones realizadas. Debido a las circunstancias por las cuales se apoyaban los objetos en el espacio de trabajo, el mayor partido que se le hubiera podido sacar a la base de datos hubiera sido para el brazo robot anclado al techo por su base, puesto que de esta forma hubiera tenido mucha más flexibilidad al realizar agarres, debido principalmente a que los condicionantes al agarre respecto a la seguridad del entorno hubieran sido menos restrictivas.

Respecto a la aplicación de clasificación, la incorporación del paquete de agarre ha sido una mejora necesaria: sólo depende de información trivial del sistema y ofrece un buen servicio a nivel funcional.

En el caso de la aplicación para el laboratorio, se ha demostrado la precisión, calidad y seguridad que puede aportar el paquete de agarre ante un entorno de trabajo controlado. La implementación del paquete de agarre abre una nueva frontera para el desarrollo entre robot operado y operario, pudiendo dar paso al desarrollo de redes neuronales o protocolos de interacción.

5.3 TRABAJOS FUTUROS

5.3.1 Respecto al paquete de agarre:

-Generar una base de datos más completa en GraspIt!, abarcando más volúmenes y formas de agarre respecto a la herramienta BarrettHand.

-Encontrar un sustituto a GraspIt! para planificar de forma dinámica sobre otras herramientas distintas a la BarrettHand.

-Proporcionar al algoritmo lógico más condiciones para evaluar situaciones de agarre complejas, como podría ser establecer una condición de agarre evaluando teóricamente el peso o el material del objeto objetivo [17].

5.3.2 Respecto al sistema de aplicaciones completo:

-Añadir un módulo de cinéticas, para poder realizar aplicaciones de agarre a objetos en movimiento.

-Desarrollar en profundidad el control manual para la aplicación del laboratorio robotizado, incorporando una interfaz al usuario y/o la capacidad de accionarse mediante joystick.

-Actualizar el sistema de visión artificial, de tal forma que se puedan evaluar objetos compuestos, pudiendo obtener las primitivas de sus formas estructurales y se pueda evaluar el agarre de manera personalizada para cada sección del objeto.

CAPÍTULO 6

PRESUPUESTO

Se lista la cuantía de gastos que debería asumirse en el caso de instalar un sistema completo para realizar cualquiera de las aplicaciones de este proyecto (*21% IVA incluido*):

SOFTWARE

PRODUCTO	**PRECIO UNIDAD**	**CANTIDAD**	**TOTAL (EUR)**
Ubuntu 12.04 LTS	Desde "0"	1	0
Robot Studio 5.15 (incluido con IRB120)	0	1	0
ROS Hydro Medusa	0	1	0
GraspIt!	0	1	0
		TOTAL	0

HARDWARE

PRODUCTO	**PRECIO UNIDAD**	**CANTIDAD**	**TOTAL (EUR)**
Portátil ASUS K550C	640	1	640
BarrettHand BH8-262	35000	1	35000
ABB IRB 120 + IRC5	12000	1	12000
Kinect para Windows V1	120	1	120
Ordenador para simulación	300	1	300
		TOTAL	48060

RECURSOS HUMANOS

PROCESO	**SALARIO MENSUAL**	**MESES EMPLEADOS**	**TOTAL (EUR)**
Documentación del sistema	1400	1	1400
Redacción de memoria y diseño gráfico de figuras	1300	1	1300
Implementación del sistema	1500	4	6000
		TOTAL	8700

IMPORTE TOTAL

PROCESO	**TOTAL (EUR)**
SOFTWARE	0
HARDWARE	48060
RECURSOS HUMANOS	8700
GASTOS DE IMPRENTA	150
TOTAL	56910

CAPÍTULO 7

MANUAL DE USUARIO

Para recrear el sistema es necesario seguir unos parámetros de instalación, asegurarse de que todas las dependencias de los paquetes están descargadas y disponer de los paquetes necesarios diseñados en este proyecto y en trabajos anteriores [1] [2] [3].

A continuación se presentan las líneas que hay que ejecutar en la terminal de Ubuntu para descargar las dependencias relacionadas con el paquete de agarre.

Premisa: Ubuntu 12.04 LTS ha sido instalado satisfactoriamente y mantener una conexión a internet estable.

7.1 Instalación de ROS HYDRO MEDUSA

Instancie los siguientes comandos en una terminal de Ubuntu:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add –
$ sudo apt-get update
$ sudo apt-get install ros-hydro-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ source /opt/ros/hydro/setup.bash
```

Advertencia: wget http://... Envía una petición de validación al servidor de ROS. Asegúrese de que el servidor le ofrezca respuesta operativa al requerirla, puesto que si no recibe la clave algunas características de ROS se le denegaran.

En este punto ROS Hydro Medusa se habrá instalado. A continuación genere el directorio de trabajo “Workspace”:

```
$ mkdir -p ~/ros_catkin_ws/src
$ cd ~/ros_catkin_ws/src
$ catkin_init_workspace
$ cd ~/ros_catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Consulte el directorio “home” de Ubuntu. Deberá haber aparecido una carpeta llamada “ros_catkin_ws” que contiene tres carpetas llamadas “build”, “devel” y “src”, tal y como se describieron en este proyecto, en el apartado de “paquetes de ROS”.

Si lo desea, puede generar las carpetas de forma manual, respetando la estructura anterior.

7.2 Instalación de ROS openni

Paquetes de configuración básica para la cámara Kinect:

$ sudo apt-get install ros-hydro-openni-camera
$ sudo apt-get install ros-hydro-openni-launch

7.3 Instalación de MoveIt!

Instale el planificador de movimientos para el brazo robot:

$ sudo apt-get install ros-hydro-moveit-full

7.4 Instalación de GraspIt!

Obtenga la última versión comprimida del instalador de GraspIt desde http://graspit.sourceforge.net/ . En este proyecto se usó la versión correspondiente a:

graspit-2.2.tar.gz

Descomprima el fichero y guarde su contenido en la carpeta deseada para realizar su instalación.

Antes de instalarlo, descargue las siguientes dependencias:

$ sudo apt-get install libqt4-dev
$ sudo apt-get install libqt4-opengl-dev
$ sudo apt-get install libqt4-sql-psql
$ sudo apt-get install libcoin60-dev
$ sudo apt-get install libsoqt4-dev
$ sudo apt-get install libblas-dev
$ sudo apt-get install liblapack-dev
$ sudo apt-get install libqhull-dev

**Advertencia*: las dependencias descritas pueden presentar ciertos fallos de compatibilidad con ROS Hydro Medusa, debido a las actualizaciones propias de ROS y de los propios paquetes. Si surgiera fallo, es necesario instalar su dependencia compatible, usualmente, la siguiente versión superior del paquete en cuestión.*

Una vez las dependencias han sido instaladas, diríjase mediante el comando "*cd*" en el terminal de Ubuntu hasta la carpeta donde están descomprimidos los archivos de GraspIt!, y construya la aplicación ejecutando:

$ qmake graspit.pro
$ make

7.5 Instalación de las dependencias relacionadas con el control de la BarrettHand

El paquete de agarre funciona de manera independiente al control de la herramienta, por lo que no es necesario disponer de las dependencias de control para su uso. Consulte la guía de usuario de Alejandro Gómez Rubio [1] para instalar las dependencias de control de la herramienta BarrettHand.

7.6 ESTACIÓN VIRTUAL EN ROBOTSTUDIO (Recomendado)

A continuación se explican los pasos para simular un brazo robot IRB120 en un ordenador personal. En este proyecto se usaron dos ordenadores portátiles para simular el comportamiento del sistema: uno en función de operador y otro en función de robot simulado. Este apartado no es necesario a nivel funcional de paquetes, pero es recomendable que se realice, a fin de tener un robot simulado para realizar pruebas en él.

Para instalar la estación de manera automática, dentro del CD del proyecto se adjunta la carpeta del sistema ya configurado, usado para el proyecto (v5.15 de RobotStudio). Edite la IP del socket, como se explicará a continuación, y su estación estará configurada completamente.

A continuación se explica paso a paso como crear la estación virtual.

PARA INSTALAR LAS DEPENDENCIAS DE ROS EN UN BRAZO ROBOT REAL, VAYA AL APARTADO "INSTALACIÓN DE SOCKET"

Abra RobotStudio, preferiblemente la versión 5.15.

- Pinche en Estación vacía, Crear.
- Inicio, Biblioteca ABB, IRB120, Aceptar.
- Inicio, Sistema de robot, Desde Diseño.
- Saltará una ventana. Edite el nombre de su estación, guárdela con su versión de RobotStudio, y pinche en Siguiente.
- La ventana mostrará el nombre del robot en la tabla Mecanismos. Pinche en Siguiente.
- Después de un ligero retardo, aparecerán las Opciones del sistema. En esta pantalla, pinche en Opciones…
- Aparecerá una lista desplegable para cambiar las opciones. Seleccione las opciones 616-1 PC Interface y 623-1 Multitasking. Aceptar y Finalizar.

Hasta aquí ha creado una estación básica en RobotStudio. Ahora hay que añadirle los programas controladores, en formato RAPID.

- Dentro de la carpeta de RobotStudio, abra la carpeta Systems y abra la carpeta con el nombre de su estación. A continuación, pegue el contenido de la carpeta HOME dentro del directorio, de tal forma que la carpeta HOME almacene dos ficheros .sys y la carpeta ROS.
- A continuación es necesario que sepa la IP de su ordenador para controlar el robot simulado.
- En Windows, abra la aplicación Ejecutar, instancie cmd para abrir la consola, y pinche en Aceptar. Una vez aparezca la consola cmd.exe , instancie en el

terminal ipconfig y pulse intro. Se desplegarán sus conexiones, copie la IP referente a la dirección IPv4, por defecto.

- Vuelva a la carpeta ROS que ha generado hace dos pasos. Abra el fichero ROS_socket.sys con un editor, y cambie la IP por la nueva adquirida.

Hasta este punto, ha configurado los accesos más básicos al brazo robot.

7.7 INSTALACIÓN DEL SOCKET

Ahora es necesario cargar dentro del controlador FlexPendant los controladores entre ROS y robot. Este apartado se realiza tanto para el sistema virtual como para el real, en sus respectivos medios. Se explicará a continuación para el caso virtual.

Dentro del entorno de RobotStudio, en la ventana del Controlador, pinche en FlexPendant.

Dentro de la carpeta RAPID se hayan los archivos que deben de ser cargados en la memoria del controlador. Para poder realizar cambios en el controlador FlexPendant, entre en modo manual:

Figura 7.1: Activación de modo manual en FlexPendant

Una vez en modo manual, desde el menú pinche en:

- ABB, Control Panel, Configuration, Topics, Controller, Task *(doble click)*

Añada y edite los siguientes módulos, respetando la siguiente configuración:

NAME	***TYPE***	***TRUST LEVEL***	***ENTRY***	***MOTION TASK***
ROS_StateServer	SEMISTATIC	NoSafety	Main	NO
ROS_MotionServer	SEMISTATIC	SysStop	Main	NO
T_ROB1	NORMAL		Main	YES

Tabla 7.1: Configuración de las tareas del socket

Una vez realizado, desplácese a:

- ABB, Control Panel, Configuration, Topics, Controller, Automatic loading of Modules *(doble click)*

Añada los siguientes archivos, respetando la siguiente configuración:

FILE	TASK	INSTALLED	ALL TASKS	HIDDEN
HOME:ROS/ROS_common.sys		NO	YES	NO
HOME:ROS/ROS_socket.sys		NO	YES	NO
HOME:ROS/ROS_messages.sys		NO	YES	NO
HOME:ROS/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO
HOME:ROS/ROS_motionServer.mod	ROS_MotionServer	NO	NO	NO
HOME:ROS/ROS_motion.mod	T_ROB1	NO	NO	NO

Tabla 7.2: Configuración de los módulos del socket

Una vez instalados estos archivos, el brazo robot, simulado o real, está listo para poder realizar las aplicaciones desde el entorno de ROS.

7.8 CARGA DE LOS PAQUETES

Dentro del CD del proyecto se haya el paquete de agarre y las aplicaciones puestas en marcha que usan sus ventajas. Las aplicaciones aportadas dependen del trabajo previo realizado sobre el brazo robot [1] [2] [3] . Asegúrese de poseerlos para poder ejecutar las aplicaciones correctamente.

Para compilar y construir los paquetes, cópielos dentro de la carpeta *"src"* del directorio de trabajo *"ros_catkin_ws".* Una vez copiados, navegue mediante la terminal hasta *"ros_catkin_ws"*, y ejecute:

$ catkin_make

**Advertencia*: Debido a problemas relacionados con la adquisición de la licencia en la instalación de ROS Hydro, esta instrucción puede estar limitada a nivel de usuario. Si tiene problemas al construir paquetes debido a errores en la búsqueda de dependencias, se recomienda que actué en modo superusuario:*

$ sudo su

$ catkin_make

Actué con precaución respecto al usar el terminal en este modo.

7.9 LANZAR LAS APLICACIONES

Para ejecutar las aplicaciones pertinentes, siempre se abrirá una terminal en Ubuntu, se instanciará el comando "**roscore**" y a continuación se podrán a empezar a lanzar los nodos necesarios. Puede ir abriendo nuevas pestañas en la terminal mediante el comando *CTRL +SHIFT+T.*

Desplácese en el terminal a la ubicación de su entorno de trabajo

$ cd ros_catkin_ws/

Cada vez que vaya a lanzar una aplicación en una terminal, es necesario que se especifique la ubicación de la fuente. Si se ha desplazado a la ubicación del entorno de trabajo antes de abrir una nueva pestaña de terminal, el directorio se mantendrá.

Determine la fuente

$ source devel/setup.bash

Repita este proceso cada vez que se disponga a lanzar un nuevo nodo en una terminal.

Realice la conexión con el robot:

$ roslaunch abb_common robot_interface.launch robot_ip:=XXX.XXX.X.XX

**Advertencia*: Si realiza esta conexión con un robot simulado, asegurase de que el cortafuegos le permita la conexión.*

Ejecute el nodo de planificación y el entorno virtual:

$ roslaunch hand_moveit_config move_group_fast.launch

Ejecute el nodo de la mesa de trabajo:

$ rosrun table table

Hasta aquí, tendrá el entorno de trabajo listo en ROS. Una vez logrado, siga los pasos correspondientes a cada aplicación para ejecutarla.

7.9.1 APLICACIÓN DE CLASIFICACIÓN

Ejecute el nodo de visión artificial

$ rosrun threeDOR threeDOR

Ejecute el nodo para mostrar los objetos en el espacio de trabajo

$ rosrun object_atta object_atta

Ejecute el nodo de agarre

$ rosrun grasp grasp

Ejecute el nodo de la aplicación

$ rosrun gttach gttach

7.9.2 APLICACIÓN DE ROBOT EN LABORATORIO

Ejecute el nodo referente a los recipientes del laboratorio

$ rosrun probetas probetas

Ejecute el nodo para mostrar los objetos en el espacio de trabajo

$ rosrun object_atta object_atta

Ejecute el nodo de agarre

$ rosrun grasp grasp

(Opcional) Ejecute la opción de teclado para entrar en modo manual

$ rosrun keyboard keyboard

Ejecute el nodo de la aplicación

$ rosrun roblab roblab

BIBLIOGRAFÍA

[1] Alejandro Gómez Rubio, “Control de la mano robótica BarrettHand BH8-262 en entorno ROS”, TFG, UAH, 2014

[2] Alberto Lázaro Enguita, “3D Object recognition and pose estimation using VFH descriptors”, TFG, UAH, 2014

[3] Javier Moriano Martín, “Trayectory planning for the IRB120 robotics arm using ROS”, TFG, UAH, 2014

[4] Origen y desarrollo de la robótica, http://proton.ucting.udg.mx/materias/robotica/r166/r63/r63.htm [Activo el 01/08/2015]

[5] Video promocional de ABB, https://www.youtube.com/watch?v=SOESSCXGhFo

[Activo el 01/08/2015]

[6] Robots usando grippers en la actualidad, https://www.youtube.com/watch?v=Bi0o-ai6-yM ; http://www.technologyreview.com/news/537646/robots-start-to-grasp-food-processing/

[Activo el 01/08/2015]

[7] Juan Hernández Sánchez, “Técnicas Modernas en Automática”, Parte III: “Robótica”

[8] Wiki oficial de ROS, http://wiki.ros.org/

[9] Jason M. O'Kane, “A Gentle Introduction to ROS”, https://cse.sc.edu//~jokane/agitr/ [Activo el 01/08/2015]

[10] Página oficial de Barrett Technology Inc, http://www.barrett.com/products-hand.htm [Activo el 01/08/2015]

[11] Página oficial del ABB IRB 120 http://new.abb.com/products/robotics/es/robots-industriales/irb-120 [Activo el 01/08/2015]

[12] Página oficial de GraspIt! http://www.cs.columbia.edu/~cmatei/graspit/ [Activo el 01/08/2015]

[13] Relación entre cuaternios y ángulos de Euler https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles [Activo el 01/08/2015]

[14] Propiedades de los cuaternios https://es.wikipedia.org/wiki/Cuaterni%C3%B3n [Activo el 01/08/2015]

[15] Ley orgánica de prevención de riesgos biológicos http://www.insht.es/InshtWeb/Contenidos/Documentacion/FichasTecnicas/NTP/Ficheros/301a400/ntp_376.pdf [Activo el 01/08/2015]

[16] Artículo de la página NOP online http://www.oc-praktikum.de/nop/es/articles/pdf/HazardSymbols_es.pdf [Activo el 01/08/2015]

[17] Andrew T.Miller, “GraspIt!: A Versatile Simulator for Robotic Grasping” , Columbia University, 2001

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá