UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE TECNOLOGÍAS DE LA INFORMACIÓN, TELECOMUNICACIONES E INDUSTRIAL

CARRERA DE INGENIERÍA INDUSTRIAL EN PROCESOS DE AUTOMATIZACIÓN

Tema:

"DISEÑO DE SISTEMAS DE CONTROL INDUSTRIAL DE ROBOTS BASADOS EN INDUSTRIA 4.0"

Proyecto de Trabajo de Graduación Modalidad: Proyecto de Investigación, presentado previo la obtención del título de Ingeniero Industrial en Procesos de Automatización.

LÍNEA DE INVESTIGACIÓN: Tecnología de la Información y Sistemas de Control

AUTOR: Juan Camilo Escobar Naranjo

TUTOR: Dr. Marcelo Vladimir García Sánchez, Mg.

Ambato – Ecuador

Mayo 2019

APROBACIÓN DEL TUTOR

En mi calidad de tutor del Trabajo de Investigación sobre el tema: "DISEÑO DE SISTEMAS DE CONTROL INDUSTRIAL DE ROBOTS BASADOS EN INDUSTRIA 4.0", del señor Escobar Naranjo Juan Camilo, estudiante de la Carrera de Ingeniería Industrial en Procesos de Automatización, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, considero que el informe investigativo reúne los requisitos suficientes para que continúe con los tramites y consiguiente aprobación de conformidad con el numeral 7.2 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ambato Mayo, 2019

EL TUTOR

Dr. Marcelo Vladimir García Sánchez, Mg.

AUTORÍA

El presente Proyecto de Investigación titulado: "DISEÑO DE SISTEMAS DE CONTROL INDUSTRIAL DE ROBOTS BASADOS EN INDUSTRIA 4.0", es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato Mayo, 2019

Juan Camilo Escobar Naranjo

CC: 1805451760

APROBACIÓN DEL TRIBUNAL DE GRADO

La Comisión Calificadora del presente trabajo conformada por los señores docentes Ing. Mg. Franklin Salazar e Ing. Mg. Patricio Encalada, revisó y aprobó el Informe Final del Proyecto de Investigación titulado "DISEÑO DE SISTEMAS DE CONTROL INDUSTRIAL DE ROBOTS BASADOS EN INDUSTRIA 4.0", presentado por el señor Juan Camilo Escobar Naranjo, de acuerdo al numeral 9.1 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ing. Elsa Pilar Urrutia Urrutia Mg.
PRESIDENTA DEL TRIBUNAL

Ing. Franklin Salazar Mg.
DOCENTE CALIFICADOR

Ing. Patricio Encalada Mg.
DOCENTE CALIFICADOR

DEDICATORIA:

*A mi **MADRE**, quien con todo su amor y dedicación ha logrado fomentar en sus tres hijos un espíritu de lucha basado en la humildad, principios y valores, una persona que ha sido capaz de lograr grandes cosas a lo largo de su vida, motivándome a seguir sus pasos y a culminar mi carrera de la manera más satisfactoria posible. Gracias por su esfuerzo y sacrificio invaluable.*

Juan Camilo Escobar Naranjo

AGRADECIMIENTO:

*Principalmente a **Dios**, por permitirme despertar apreciando un nuevo día, brindándome la oportunidad de vivir experiencias inigualables como hijo, hermano, amigo y ahora como profesional.*

*A mis **Padres**, por su apoyo incondicional, por creer y confiar en mí y en que podré cumplir con mis sueños, por haber estado dispuestos a darme palabras de aliento en los momentos más difíciles, gracias por cada consejo y por ser la guía en mi camino a alcanzar mis metas.*

*Al **Dr. Marcelo García,** por presentarme un reto que supo llevarme a investigar campos que nunca me habría imaginado cursando, pero que ahora cumplido, me ha quedado claro que para la dedicación no existen límites.*

Juan Camilo Escobar Naranjo

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN EJECUTIVO

En el entorno actual de crecimiento tecnológico, el término “Industria 4.0” se ha convertido en sinónimo de innovación a nivel de producción y control industrial, pues en las últimas décadas la automatización de los procesos productivos ha dado lugar a la necesidad de desarrollar sistemas y equipos capaces de realizar actividades complejas, peligrosas y repetitivas de una forma eficiente, de forma que cubran las exigencias industriales, la robótica industrial forma parte esencial en el ambiente de la producción masiva debido a que permite que esta sea más activa incorporando una mayor flexibilidad en los procesos productivos.

Durante los últimos años se han presentado varios avances tecnológicos, lo cual lleva a la miniaturización de equipos y a la reducción de costos, obteniendo de igual forma un mayor dominio sobre los sistemas de control, haciendo necesario la aplicación de sistemas ciber-físicos de bajo costo, mismos que incorporan una mayor integración de los procesos productivos, a través de la comunicación directa entre microcontroladores con sensores o actuadores.

Con estas premisas, la investigación realizada presenta la implementación física y desarrollo de un sistema de control de robots, basado en la tarjeta Raspberry Pi y del robot manipulador Kuka YouBot.

El procedimiento mencionado, se implementó sobre un sistema operativo basado en Debian, lo cual permite la generación de aplicaciones de tipo Open Source, de igual manera se diseñó el entorno de simulación que represente de la manera más exacta posible las características físicas del entorno real, así como la lectura de sensores y el envío de mensajes a través de ROS.

El sistema de control desarrollado permite la modularidad y la integración multiplataforma a través del metasistema ROS, la relevancia del diseño radica en que investigadores y empresas dedicadas a la robótica tendrán más opciones para el control de robots y la programación de sus aplicaciones, sin tener que encontrarse con problemas en la compatibilidad del hardware, contando además con una amplia variedad de herramientas como librerías y paquetes que se encuentran disponibles para las tarjetas empotradas.

Palabras Clave: Industria 4.0, Sistemas CPS, Robótica, Sistemas Empotrados.

ABSTRACT

In the current environment of technological growth, the term "Industry 4.0" has become synonymous with innovation at the level of production and industrial control, since in recent decades the automation of production processes has led to the need to develop systems and equipment capable of carrying out complex, dangerous and repetitive activities in an efficient way, in order to meet industrial demands, industrial robotics is an essential part of the mass production environment because it allows it to be more active, incorporating greater flexibility in the productive processes.

During the last years several technological advances have been presented, which leads to the miniaturization of equipment and the reduction of costs, obtaining in the same way a greater control over the control systems, making necessary the application of low-cost cyber-physical systems , which incorporate a greater integration of production processes, through direct communication between microcontrollers with sensors or actuators.

With these premises, the research carried out presents the physical implementation and development of a robot control system, based on the Raspberry Pi card and the Kuka YouBot manipulator robot.

The aforementioned procedure was implemented on an operating system based on Debian, which allows the generation of Open Source applications, in the same way the simulation environment was designed that represents as accurately as possible the physical characteristics of the real environment, as well as reading sensors and sending messages through ROS.

The control system developed allows modularity and multiplatform integration through the ROS metasystem, the relevance of the design lies in the fact that researchers and companies dedicated to robotics will have more options for controlling robots and programming their applications, without having to encounter problems in hardware compatibility, with a wide variety of tools such as libraries and packages that are available for embedded cards.

Keywords: Industry 4.0, CPS Systems, Robotic, Recessed Systems.

CAPÍTULO I
MARCO TEÓRICO

1.1 Antecedentes Investigativos

El control de procesos mediante sistemas embebidos se encuentra atravesando una etapa de crecimiento gracias a la apertura de las industrias hacia los sistemas cuyas capacidades de seguimiento y control han mejorado su modo de operación a pesar de su bajo costo, como se detalla en[1], los avances de las aplicaciones basadas en Internet permiten entornos de producción distribuida mediante la interacción entre sistemas computacionales integrados y los ambientes físicos, dando lugar dentro de procesos automáticos y de control a los Sistemas Ciber Físicos de Producción (CPPS), demostrando el alto potencial que tiene en el sector productivo, además se debe tener en cuenta que un sistema completo debe contar con varios requerimientos, los cuales tienen diferentes aplicaciones con sus propias barreras. Los avances tecnológicos han sido parte fundamental para el desarrollo de sistemas con estructuras diferentes, cuya arquitectura y comunicación trabajan bajo un mismo estándar.

En la investigación presentada en[2], se menciona la importancia que tiene el uso de plataformas de bajo costo, mostrándose como un problema habitual el elevado costo de los controladores de sistemas electrónicos, mecánicos y robóticos. Se indica además que por su versatilidad y bajo precio las tarjetas microcontroladoras han llegado a tener peso en la robótica educacional y en la investigación, se presenta la utilización de los micro ordenadores, los cuales promueven el desarrollo de aplicaciones complejas dentro del campo académico, el aspecto de mayor importancia es su unidad programable ya que permite mediante sus pines de entrada y salida la lectura de sensores y el cambio de estado en actuadores, además de sus puertos de comunicación los cuales generalmente son de alta velocidad, permitiendo el control de estaciones complejas, por varias razones como las mencionadas, las tarjetas embebidas son consideradas como potentes alternativas ante ordenadores que permitan el mismo desarrollo de actividades con similar consumo de recursos pero a un precio mucho más reducido.

La investigación descrita en[3], presenta la utilización de ROS como un framework implementable dentro de los Sistemas Ciber-físicos, pues simplifica la comunicación y monitoreo del estado del control de robots, este llamado software se instala sobre GNU/Linux permitiendo incorporar varios dispositivos a una red distribuida, donde un nodo maestro permite al resto de nodos comunicarse a través TCP/IP, además dichos elementos que forman parte de la arquitectura de CPS se pueden asegurar mediante la implementación de un servidor VPN, donde se obtienen varias ventajas de esta red de nodos ROS, como la modularidad de la arquitectura permitiendo la adaptación a otros sistemas, es posible la redundancia debido al esquema publicación/suscriptor de ROS, además de permitir el acceso multiplataforma y el control remoto mediante otro elemento conectado a la misma red. La única restricción presente es que el sistema de control se encuentre desarrollado bajo Debian, es decir sea de tipo Open Source, debido a la compatibilidad existente entre sistemas operativos.

Como un ejemplo de la utilización de ROS sobre sistemas embebidos de bajo costo se tiene el ejemplo desarrollado en[4], donde se menciona que al momento de elegir plataformas compatibles con ROS se debe considerar las limitaciones en la comunicación, además que al momento de desarrollar software para robots existe una gran variedad de lenguajes de código abierto como C++, Python o Java. También se presenta que la tarjeta SBC (Single Board Computer) Raspberry Pi es una de las mejores elecciones en cuanto a la utilización de ROS se refiere, debido a su capacidad de procesamiento, además de su bajo costo, posee repositorios con comunidades desarrolladoras, donde un usuario puede obtener asesoría de forma casi inmediata, sin embargo, se debe realizar una minuciosa selección debido a que no todos los softwares montables en la tarjeta son compatibles y poseen los paquetes y dependencias necesarias.

Otro ejemplo de la aplicación de ROS se presenta en[5], donde se menciona el control de navegación de un robot utilizando la tarjeta Raspberry, se enuncia que los paquetes ROS deben construirse desde fuente dependiendo de la distribución que se utilice, además se describe el funcionamiento general del entorno a través de la utilización de nodos y topics, se establece el funcionamiento de la red modular que presentan los nodos conectados al maestro ROS, el cual es arrancado para hacer posible la suscripción a los topics que sean publicados por un nodo, permitiendo así la lectura de sensores y la activación de actuadores.

1.2 Objetivos

1.2.1 Objetivo General

Implementar métodos de control utilizando tarjetas empotradas de bajo costo para el brazo robótico manipulador Kuka youBot.

1.2.2 Objetivos Específicos

- Desarrollar el entorno de simulación para el brazo robótico del manipulador Kuka youBot.
- Identificar los parámetros ROS del brazo robótico para su control mediante mensajes enviados por la tarjeta empotrada.
- Definir el método de comunicación para habilitar el control para el entorno real.

CAPÍTULO II
METODOLOGÍA

2.1 Materiales

2.1.1 Ubuntu WSL

Ubuntu es conocido como un sistema operativo de tipo Open Source para computadoras y tarjetas embebidas, es parte de las distribuciones de Linux y está basado en la arquitectura Debian de tipo Intel, AMD y ARM, está compuesto mediante software libre lo cual permite la integración por parte de desarrolladores, siendo el único servicio pagado el soporte técnico[6], actualmente se encuentra disponible la última versión la cual cuenta con mejores características debido a parches para bugs encontrados en versiones anteriores, Ubuntu 18.10 fue lanzado el 18 de octubre del 2018 y se encuentra disponible en la forma WSL para Windows 10.

WSL (Windows Subsystem for Linux) es parte del programa "entorno para programadores" desarrollada por Microsoft para correr ejecutables de Linux nativamente en Windows 10[7], este sistema provee una interfaz que arranca únicamente un terminal de comandos de Ubuntu pero con todas las características que las aplicaciones orientadas a este puedan tener, al no tener entorno gráfico GUI, este necesita de programas externos que permitan abrir ventanas mediante el uso de un servidor local de la misma Pc, esta herramienta se encuentra únicamente disponible para Windows 10 de 64 bits y se puede descargar la versión de Ubuntu 18.01 gratuitamente desde la tienda de Microsoft[8].

2.1.2 Framework ROS

Es un espacio de trabajo de código abierto (Open Source) que facilita y agilita la construcción de aplicaciones robóticas como envío de mensajes, gestión de paquetes y controladores del dispositivo hardware.

Originalmente fue diseñado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford, con el objetivo de simplificar los sistemas distribuidos, se define los bloques de construcción de una aplicación ROS como nodos, donde cada uno es una entidad

que puede comunicarse con otros nodos que son parte de un proceso del sistema operativo, los nodos soportados por ROS son principalmente escritos en C++ y Phyton.

Existen varias distribuciones de ROS, estas se definen como un conjunto de diferentes paquetes, su propósito es permitir a los desarrolladores trabajar con una base de código estable, cada distribución se basa en la corrección de errores y de mejoras no interrumpidas.

Los nodos en un entorno ROS pueden comunicarse de tres formas como: Publicando mensajes a un tema, escuchando los mensajes publicados sobre un tema y llamando a un servicio proporcionado por otro nodo[4].

Los mensajes representan estructuras que pueden ser transmitidas entre nodos, y la información que estos contienen puede ser de cualquier tipo de dato primitivo, por ejemplo: enteros, booleanos, números de coma flotante o tipo texto.

Los topics o temas de ROS representan canales con nombre sobre los cuales un tipo particular de mensaje puede ser transferido, el topic provee comunicación unidireccional entre cualquier número de publicaciones y nodos consumidores.

Un nodo que publica un topic o tema es llamado publicador o editor, estos corren continuamente para proporcionar la lectura de sensores u otra información periódica a otros nodos, sin embargo, es posible que un editor publique un mensaje una sola vez. Un nodo que escucha los mensajes de un tema es llamado suscriptor, el cual especifica que tema es el que quiere escuchar así también como el tipo de mensaje para dicho tema, además registra una función de devolución de llamada (callback) que se ejecuta siempre y cuando se reciba un mensaje[9].

Finalmente un nodo puede proporcionar un servicio a otro nodo, donde una llamada de servicio en ROS se define como un flujo de trabajo remoto, un nodo cliente envía una solicitud al nodo del proveedor de servicios, mientras que una función de devolución de llamada registrada en el nodo proveedor realiza la acción apropiada y el proveedor de servicio envía una respuesta de regreso al cliente, dado que una llamada de servicio es un intercambio entre un nodo cliente y un proveedor de servicios no se emplean temas, sin embargo los mensajes de petición de servicio y de respuesta se definen de manera similar a los mensajes estándar. Se obtiene un gran flexibilidad

en las acciones ya que un nodo puede realizar cualquier combinación de las acciones de comunicación[4].

En la figura N°1, se puede observar como el maestro ROS permite el intercambio de información dentro del protocolo publicación/suscriptor.

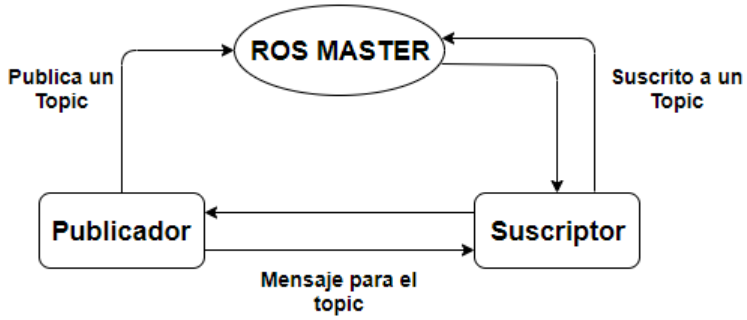

Figura N° 1: Arquitectura ROS

Fuente: Elaborado por el Investigador

Inicialmente para la gestión del sistema de comunicación se debe iniciar o crear un ROS master, el cual es encargado de permitir que los nodos puedan encontrarse e intercambiar mensajes, el uso de temas o topics se define como rutas de conexión lógicas, donde los nodos pueden publicar temas o bien, suscribirse a ellos[10].

Actualmente es compatible con Raspbian Jessie la distribución de ROS Indigo, sin embargo, su instalación debe realizarse por fuente debido a la ausencia de paquetes pre-compilados, es decir se deben clonar todas las dependencias necesarias para su posterior compilación.

2.1.3 Entorno de Simulación Gazebo

Es un simulador de robots múltiple desarrollado por Open Source Robotics Foundation(OSRF), diseñado para ambientes al aire libre, el sistema es compatible con ROS[11], por lo cual es una buena opción para representar el envío de mensajes hacia el robot, adicionalmente se presenta una simulación física del cuerpo rígido que permite la detección de colisiones y la manipulación de objetos, además Gazebo permite la simulación de sensores permitiendo incorporar la funcionalidad completa de Kinect y las cámaras de techo en la simulación, permite el control mediante funciones API con aplicaciones programadas en C, C++, Python, Java, Matlab.

El entorno permite simular tanto el brazo como la base del robot manipulador Kuka YouBot como se presenta en la figura N°2 y ambos pueden ser controlados dentro de

la simulación, se pueden agregar y mover objetos con los que el robot puede interactuar.

Gazebo posee un amplio repositorio con la mayoría de los robots comerciales, posee un motor de renderizado avanzado, lo cual permite el reconocimiento de imagen, que se integra con el soporte para plugins y con el software operativos para robots basado en Linux (ROS).

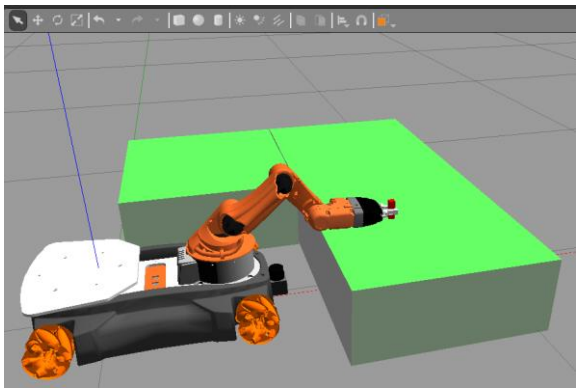

Figura N° 2: Kuka youBot en Gazebo
Fuente: Elaborado por el Investigador

2.1.4 Manipulador Kuka youBot

Se define como un manipulador móvil desarrollado principalmente para la educación e investigación, formado por dos partes que se pueden utilizar de forma independiente[12], un brazo robótico de cinco grados de libertad con un gripper como efector final y de una base omnidireccional como se muestra en la figura N°3, su brazo cuya interfaz de programación de aplicaciones (API) es compatible con ROS se puede comunicar a través de Ethernet y EtherCAT, además en su base posee una PC mini ITX integrado con su CPU[13].

Figura N° 3: Robot Manipulador Kuka youBot

El brazo robótico como se muestra en la figura N°4 consta de cinco juntas rotatorias y una pinza de dos dedos en paralelo como herramienta final, está diseñado para transportar una carga de hasta de 0.5kg y se puede acceder a sus juntas rotatorias a través de Ethernet y EtherCAT.

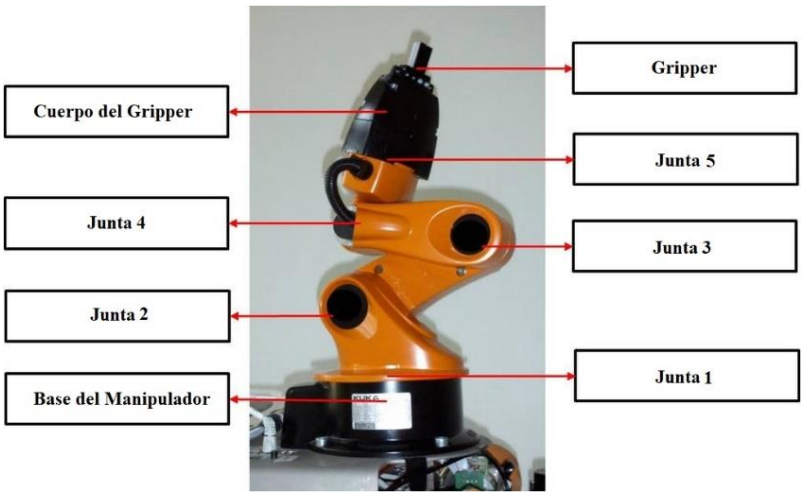

Figura N° 4: Partes del brazo Robótico del manipulador YouBot

Fuente: Elaborado por el Investigador

El protocolo de comunicación definido para el Kuka YouBot utiliza la red estándar Ethernet de tipo Simple Open Source EtherCAT Master (SOEM), el cual permite la implementación de un controlador de código abierto para el maestro.

2.1.5 Raspberry Pi

Mundialmente conocido como un ordenador de placa única o reducida (SBC) de bajo costo, con el objetivo de desarrollar sistemas de enseñanza en informática fue lanzado por la Fundación Raspberry Pi el modelo A del microcontrolador, expresamente no se define como hardware libre sin embargo al ser de código abierto (Open Source) cualquiera puede aportar haciendo modificaciones en la programación del software original, la tarjeta permite el uso de varios sistemas operativos GNU/Linux como Ubuntu, Raspbian e incluso Windows 10[14].

En cualquier versión de la tarjeta se puede encontrar un procesador Broadcom, HDMI, memoria RAM, puertos USB, Ethernet, una GPU, 40 pines GPIO y un conector para cámara, ninguna edición posee memoria por lo cual se debe colocar una MicroSD de alta velocidad para no afectar el rendimiento de la tarjeta[15].

Se considera como una potente alternativa ante los ordenadores actuales pues al poseer arquitectura armhf es de bajo costo, además al tener software basado en Linux, sus aplicaciones son de tipo Open Source, la tarjeta puede servir como un microcontrolador donde a partir de repositorios el usuario podrá tratar con una amplia multitud de programas.

En la tabla N°1 se detalla las características técnicas de la tarjeta Raspberry Pi 3, modelo B.

Tabla 1: Especificaciones Técnicas de la tarjeta SBC Raspberry Pi 3 B

Detalle	Raspberry Pi 3 B
Dimensiones	85 x 56 mm
Procesador	Broadcom BCM2837
Memoria RAM	1 GB
Pines I/O	40
Puertos USB	4 puertos / 2.0
Puerto Ethernet	10/100
SO's estables	GNU/Linux: Ubuntu, Raspbian y Windows 10
Consumo Eléctrico	2.5A y 5V

Se seleccionó la tarjeta Raspberry Pi 3 modelo B, debido a su gran capacidad de procesamiento, además de contar una mayor cantidad de Sistemas Operativos estables, permitiendo la comunicación de forma cableada e inalámbrica, adicionalmente cuenta con la mayor comunidad de desarrolladores y se dispone de una amplia cantidad de repositorios que permiten el intercambio seguro de archivos, paquetes y dependencias.

En la figura N°5 se presenta la tarjeta SBC Raspberry Pi 3 modelo B.

Figura N° 5: Raspberry Pi 3 Modelo B

2.1.6 Raspbian Jessie

Es un sistema operativo GNU/Linux basado en Debian para el uso exclusivo de Raspberry Pi, es considerado oficialmente como el SO principal para esta familia de SBC, se encuentra en continuo desarrollo proporcionando versiones más recientes y estables, Raspbian está optimizado para el uso exclusivo en CPU's de tipo ARM[16], al ser de tipo Open Source cuenta con gran soporte por parte de la comunidad de desarrolladores, además se encuentra compuesto por un entorno LXDE y un administrador de ventanas Openbox[17].

2.1.7 Catkin Workspace

Se define como un espacio de trabajo catkin a una carpeta o directorio dentro de la cual se permite la modificación, creación e instalación de paquetes catkin, dentro de este se encuentra el espacio fuente el cual contiene el código raíz de los paquetes que se desea compilar.

Una vez que se han compilado los paquetes se crea de forma automática el espacio de construcción, el cual invoca a la instancia CMake para la construcción de los paquetes catkin[18].

Se trabaja en espacios de trabajo catkin debido a que permiten una mejor organización y composición de los paquetes a utilizar, además se ejecutan de forma rápida y simultanea la construcción de numerosos e independientes proyectos, los cuales no necesitan compartir o ser parte del mismo proyecto.

El principal objetivo de catkin es organizar las dependencias de manera que al compilar el código en C++ este pueda encontrar fácilmente todos los ficheros o directorios que fueron declarados en la programación, una vez compilados se distribuye los paquetes de manera que ROS esté disponible para ubicar y correr los ejecutables necesarios.

2.2 Caso de Estudio

Para la presente investigación se ha propuesto el desarrollo de un sistema de control para robots basado en la utilización de ROS ejecutándose sobre la tarjeta SBC Raspberry Pi y del robot manipulador Kuka YouBot.

El desarrollo del trabajo comienza con la simulación del robot, para el cual existen varios entornos de simulación, sin embargo, el predilecto por Kuka es Gazebo, mismo

que a su vez permite el envío de mensajes a través de ROS, mediante la construcción de plugins que habilitan la comunicación entre el robot y el terminal de comandos, cabe recalcar que todo el caso de estudio planteado se desarrolla sobre el sistema operativo Linux, por las ventajas que el desarrollo Open Source ofrece.

El entorno de simulación permite determinar los rangos rotacionales de las juntas que posee el brazo robótico y el tipo de dato que debe ser enviado hacia los motores, estos parámetros son similares a los existentes en el entorno real debido a que en Gazebo el robot Kuka YouBot existe dentro de la biblioteca del simulador, el cual cuenta con todas las características físicas necesarias.

Como caso de estudio se plantea una tarea de posicionamiento de objetos de nombre "pick and place", la cual es una de las más comunes en los procesos industriales, partiendo de los rangos rotacionales obtenidos, se determina para cada junta el valor que debe ser asignado de forma que llegue a dos posiciones diferentes, las cuales corresponden a donde el brazo robótico recogerá el objeto y en donde lo depositará, una vez obtenidas las posiciones finales de la pinza del robot, como parte de la matemática en la robótica se debe aplicar la cinemática directa para determinar la matriz de transformación homogénea, de forma que se obtenga una referencia desde la herramienta terminal hacia la base del manipulador.

Para el desarrollo en el entorno real, ROS debe estar instalado sobre Raspbian, al igual que los controladores y paquetes necesarios para el control del brazo robótico, se debe incluir el paquete de compensación gravitatoria, el cual realiza un control PD en las juntas del brazo, consiguiendo que estas trabajen de un modo más seguro al no ejercer grandes cantidades de pares de torsión que podrían dañar los motores, evitando que incluso los eslabones lleguen a ser vencidos por acción de su propio peso.

Se desarrolla la tarea de posicionamiento planteada dentro de un nuevo espacio de trabajo y a continuación, se inicializa un maestro ROS, el cual permite la suscripción de los nodos hacia un cierto topic, finalmente el control del brazo robótico se da mediante ROS corriendo en la tarjeta SBC, esta solución integrada viene dentro de los sistemas ciber-físicos de bajo costo para procesos industriales.

En la figura N°6 se presenta el esquema final de la arquitectura desarrollada para el sistema de control.

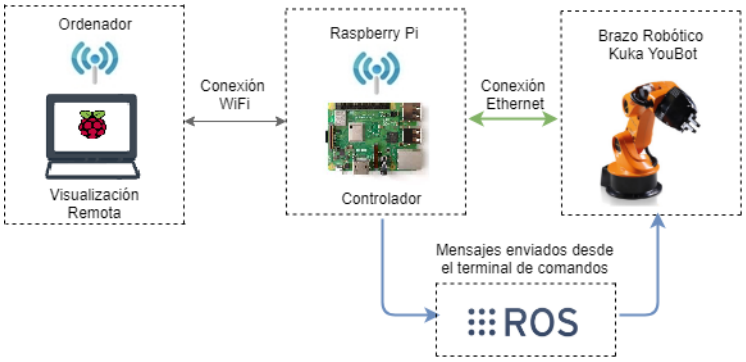

Figura N° 6: Arquitectura del Caso de Estudio

Fuente: Elaborado por el Investigador

2.2.1 Matriz de Transformación Homogénea

Para determinar la matriz homogénea del efector final en las dos posiciones de la tarea de posicionamiento "Pick and Place" se hace uso de la cinemática directa, la cual se refiere a la aplicación de las ecuaciones cinemáticas de un robot a partir de valores especificados para los parámetros requeridos por las juntas.

Existen diferentes métodos para determinar las posiciones de un brazo robótico, sin embargo, para el modelo del Kuka youBot se aplica el algoritmo Denavit-Hartenberg (D-H).

El brazo robótico consta de cinco juntas rotatorias equivalentes a 5 grados de libertad, en la figura N°7 se puede observar la longitud de cada eslabón así también como los limites angulares de cada junta o articulación[19].

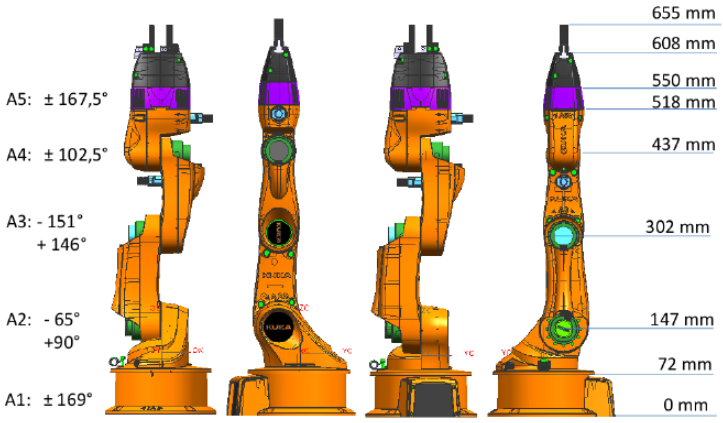

Figura N° 7: Dimensiones longitudinales y rotacionales del brazo robótico

Generalmente los manipuladores se definen como una cadena cinemática abierta donde en la mayoría de los casos se cumple que **n +1** eslabones están conectados a **n** juntas. La figura N°8 muestra el diagrama cinemático de un manipulador que posee 5

juntas rotatorias y una base móvil sobre la cual se apoya, de igual forma que el robot manipulador Kuka YouBot[20].

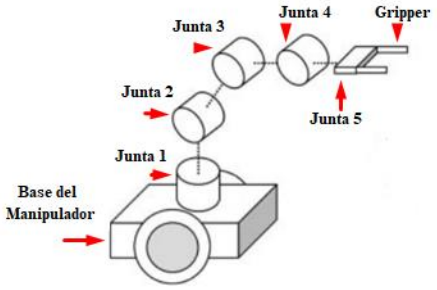

Figura N° 8: Diagrama cinemático del manipulador Kuka YouBot

El método D-H convencionalmente utiliza cuatro parámetros que especifican completamente la posición del efector final, en la figura N°9 se representa la definición original del algoritmo mencionado[21][22].

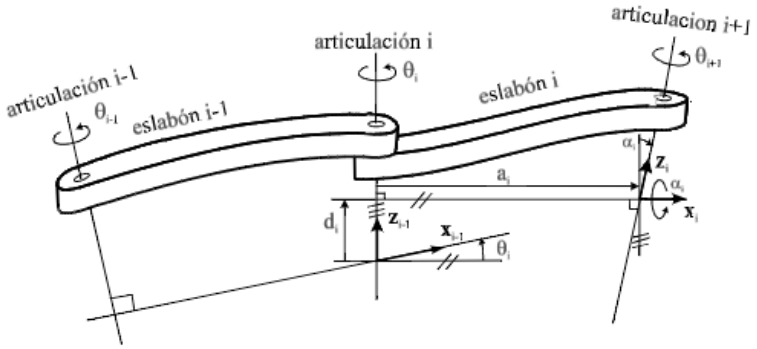

Figura N° 9: Principio de Denavit-Hartenberg

Donde:

ai = Distancia entre el origen de dos juntas O1, O2.

di = Coordenada del origen O1 a lo largo de Zi.

αi = El ángulo entre los ejes Zi-1 y Zi, en referencia al eje Xi, se toma positivo cuando la rotación es en sentido horario.

qi = El ángulo entre los ejes Xi-1 y Xi, en referencia al eje Zi-1, se toma positivo cuando la rotación es en sentido horario.

Para definir los parámetros necesarios se debe principalmente establecer sistemas de referencia sobre el brazo del manipulador Kuka youBot[23][24], para lo cual se desarrolla inicialmente el diagrama cinemático del estado del brazo robótico en las

posiciones a analizar, como resultado se obtendrá dos diagramas para la posición de recolección del objeto y de depósito.

En la figura N°10 se puede apreciar con mayor claridad el posicionamiento de los mencionados marcos referenciales para cada junta rotatoria sobre el diagrama cinemático representado para la primera posición.

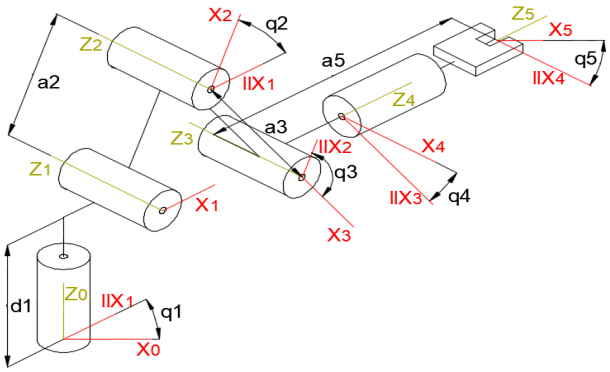

Figura N° 10: Diagrama cinemático de la primera posición

Fuente: Elaborado por el Investigador

Siguiendo los pasos del algoritmo D-H se determina los parámetros necesarios para el desarrollo de la matriz de transformación homogénea, los cuales para la primera posición se encuentran representados en la tabla N°2.

Tabla 2: Parámetros Denavit-Hartenberg

Fuente: Elaborado por el Investigador

Eslabón	qi	αi	ai	di
1	q1	- π/2	0	d1
2	q2	0	a2	0
3	q3	0	a3	0
4	q4	π/2	0	0
5	q5	0	0	d5

Para la segunda posición, la cual se diferencia en una rotación por parte de la primera junta rotatoria, se tiene el siguiente diagrama cinemático como indica la figura N°11.

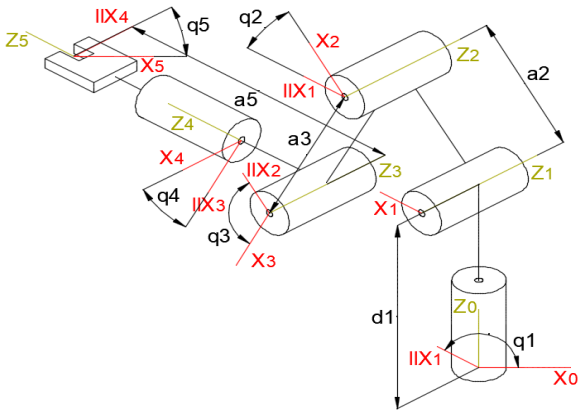

Figura N° 11: Diagrama cinemático en la segunda posición

Fuente: Elaborado por el Investigador

Para esta segunda posición el único cambio que tendrán los parámetros D-H será en el valor del ángulo **q1**, el cual incrementa su valor al tener una mayor rotación en base al sistema de referencia ortogonal principal, por ende, no se modifican la tabla con los parámetros resultados obtenidos.

Una vez determinados los cuatro parámetros correspondientes al método D-H, se procede con la obtención de la matriz de transformación homogénea, la cual es parte de este proceso sistemático que describe la estructura cinemática de un brazo robótico[24], se debe tener en cuenta que tres de los cuatro parámetros son constantes y dependen únicamente de la relación geométrica entre articulaciones, mientras que el valor de cada ángulo **q** es variable para cada articulación, siendo el ángulo de giro del eje **Xi-1,** alrededor del eje **Zi-1**, hasta llegar a **X**[25][21].

La matriz homogénea basada en el método D-H es resultado de realizar una rotación alrededor **Zi-1**, una traslación a lo largo de **Zi-1**, una traslación a lo largo de **Xi,** y una rotación alrededor del eje **Xi,** esto según el convenio de conexión de elementos contiguos de Denavit-Hartenberg, se define mediante la ecuación N°1[26].

$$\boldsymbol{A_i^{i-1} = T(z, q1) * T(0, 0, d_i) * T(a_i, 0, 0) * T(x, \alpha_i)} \qquad (1)$$

$$A_i^{i-1} = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 & 0 \\ \sin(q_i) & \cos(q_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_i^{i-1} = \begin{bmatrix} \cos(q_i) & -\cos(\alpha_i)\sin(q_i) & \sin(\alpha_i)\sin(q_i) & \text{ai} * \cos(q_i) \\ \sin(q_i) & \cos(\alpha_i)\cos(q_i) & -\sin(\alpha_i)\cos(q_i) & \text{ai} * \sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & di \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matriz A_i^{i-1} se expresa para conocer la ubicación de la herramienta terminal con respecto de la base del robot.

Para los parámetros D-H de la tabla N°2, del manipulador youBot de 5 grados de libertad, se determina las matrices de transformación para cada junta.

$$A_1 = \begin{bmatrix} \cos(q_1) & 0 & -\sin(q_1) & 0 \\ \sin(q_1) & 0 & \cos(q_1) & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \cos(q_2) & -\sin(q_2) & 0 & a_2 * \cos(q_2) \\ \sin(q_2) & \cos(q_2) & 0 & a_2 * \sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & a_3 * \cos(q_3) \\ \sin(q_3) & \cos(q_3) & 0 & a_3 * \sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \cos(q_4) & 0 & -\sin(q_4) & 0 \\ \sin(q_4) & 0 & \cos(q_4) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} \cos(q_5) & -\sin(q_5) & 0 & 0 \\ \sin(q_5) & \cos(q_5) & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Para el cálculo de la matriz de transformación homogénea T_5 la cual conecta el sistema de ejes O_5, X_5, Y_5, Z_5, con el sistema de referencia absoluto, situado en la base del robot, se realiza la multiplicación matricial siguiendo la ecuación N°2[21].

$$\boldsymbol{T_5 = A_1 * A_2 * A_3 * A_4 * A_5} \qquad (2)$$

$$T_5 = \begin{bmatrix} c_1 * c_{234} * c_5 + s_1 * s_5 & -c_1 * c_{234} * s_5 + s_1 * c_5 & -c_1 * s_{234} & c_1(-d_5 * s_{234} + a_3 * c_{23} + a_2 * c_2) \\ c_1 * c_{234} * c_5 - s_1 * s_5 & -s_1 * c_{234} * s_5 - c_1 * c_5 & -s_1 * s_{234} & s_1(-d_5 * s_{234} + a_3 * c_{23} + a_2 * c_2) \\ -s_{234} * c_5 & s_{234} * s_5 & -c_{234} & d_1 - a_2 * s_2 - a_3 * s_{23} - d_5 * c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde:

Ci = cos(qi);

Si = sen(qi);

Cijk = cos (qi + qj + qk);

Sijk = sen (si + sj + sk);

La posición del gripper se define mediante los vectores x5, y5, z5, los cuales vienen dados por:

$$\boldsymbol{p_5} = \begin{bmatrix} c_1(-d_5 * s_{234} + a_3 * c_{23} + a_2 * c_2) \\ s_1(-d_5 * s_{234} + a_3 * c_{23} + a_2 * c_2) \\ d_1 - a_2 * s_2 - a_3 * s_{23} - d_5 * c_{234} \end{bmatrix}$$

$$\boldsymbol{x_5} = \begin{bmatrix} c_1 * c_{234} * c_5 + s_1 * s_5 \\ c_1 * c_{234} * c_5 - s_1 * s_5 \\ -s_{234} * c_5 \end{bmatrix}$$

$$\boldsymbol{y_5} = \begin{bmatrix} -c_1 * c_{234} * s_5 + s_1 * c_5 \\ -s_1 * c_{234} * s_5 - c_1 * c_5 \\ s_{234} * s_5 \end{bmatrix}$$

$$\boldsymbol{z_5} = \begin{bmatrix} -c_1 * s_{234} \\ -s_1 * s_{234} \\ -c_{234} \end{bmatrix}$$

2.2.2 Compensación Gravitatoria

El campo del control de la estabilidad de robots manipuladores ha adquirido más importancia desde su propuesta en 1981, pues este solo aplicaba para brazos robóticos que cuenten únicamente con uniones rotativas, para el modelado matemático se utilizan ecuaciones que son de tipo no lineal, sin embargo, se suele usar la teoría de Lyapunov como un medio auxiliar[27].

El desarrollo del control PD con compensación de gravedad para manipuladores, se basa en la suposición de que no existe la dinámica de los actuadores, por lo cual se debe considerar que la señal de control no es el par sino el voltaje que se aplica a los motores, donde el par viene como el resultado de la interacción eléctrica de los actuadores[28].

Sin considerar perturbaciones externas, el modelo matemático de un robot rígido que cuenta con un número **n** de eslabones se describe según la ecuación N°3[27].

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F(\dot{q}) = \tau} \quad (3)$$

Donde:

$\boldsymbol{M(q)}$ = Matriz de inercia simétrica del manipulador

$\boldsymbol{C(q,\dot{q})}$ = Matriz de fuerzas centrípetas y de Coriolis

$\boldsymbol{g(q)}$ = Vector de pares gravitacionales

$\boldsymbol{F(\dot{q})}$ = Pares de Fricción

El algoritmo de control proporcional derivativo con compensación de gravedad este definido mediante la ecuación N°4[27]:

$$\boldsymbol{\tau = k_p\tilde{q} - k_d(\dot{\tilde{q}}) + G(q)} \quad (4)$$

Donde:

$\boldsymbol{k_p\tilde{q}}$ = Ganancia proporcional multiplicada por el error de posición

$\boldsymbol{k_d(\dot{\tilde{q}})}$ = Ganancia derivativa multiplicada por la velocidad de movimiento

$\boldsymbol{G(q)}$ = Compensación del par gravitacional

Se define la compensación gravitatoria como un término proporcional al error de posición, al cual se le adiciona un amortiguamiento que consiste en la velocidad de movimiento, esta parte del control consiste en una retención mecánica que absorbe los sobre impulsos resultantes de la respuesta, suprimiendo picos y oscilaciones, para la etapa estacionaria se estima que la posición del efector o herramienta terminal es constante, asumiendo un valor de cero a la velocidad de movimiento[29].

A diferencia de otros sistemas de control, para el tipo PD con compensación gravitatoria, la selección de las constantes integral y derivativa son totalmente arbitrarias y no dependen de ningún modelo dinámico, en la figura N°12, se presenta el diagrama de bloques del control para robots manipuladores[29].

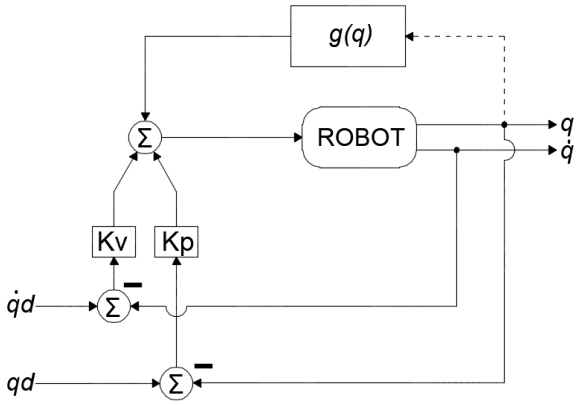

Figura N° 12: Diagrama de bloques para el lazo de control del brazo robótico.

La ecuación que define el comportamiento la acción del manipulador se define de la igualdad entre las ecuaciones N° 3 y 4, presentando el resultado en la ecuación N°5[29].

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + F(\dot{q}) = k_p\tilde{q} - k_d(\dot{\tilde{q}}) + G(q)} \quad (5)$$

La ecuación N°5 se define por ser una ecuación diferencial no lineal y no autónoma, esta propiedad nace debido a que la ecuación depende de variables en función del tiempo.

Adicionalmente se debe realizar el cálculo de las ecuaciones de movimiento de Lagrange, de forma que se pueda obtener todos los parámetros de torque necesarios, se considera un robot de **n** eslabones, del cual la energía total ε es el resultado de la sumatoria de sus energías cinética κ y potencial Ư, como se muestra en la ecuación N°6[29].

$$\boldsymbol{\varepsilon\big(q(t),\dot{q}(t)\big) = \kappa\big(q(t),\dot{q}(t)\big) + Ư\big(q(t)\big)} \quad (6)$$

El lagrangiano $L(q,\dot{q})$ de un manipulador de **n** grados de libertad, viene dado como la diferencia entre su energía cinética κ y potencial Ư cómo se presenta en la ecuación N°7[30].

$$\boldsymbol{L\big(q(t),\dot{q}(t)\big) = \kappa\big(q(t),\dot{q}(t)\big) - Ư\big(q(t)\big)} \quad (7)$$

Las ecuaciones de movimiento de Lagrange para el manipulador de n grados de libertad viene dado por la siguiente ecuación[30]:

$$\frac{d}{dt}\left[\frac{\partial L(q,\dot{q})}{\partial \dot{q}}\right] - \frac{\partial L(q,\dot{q})}{\partial \dot{q}} = \tau \quad (8)$$

Al obtener los valores de torque se pueden reagrupar las ecuaciones de Lagrange de modo que se puedan diferenciar los componentes y encontrar el vector resultante con las fuerzas gravitatorias, como se presenta en la ecuación N°9[27].

$$\tau \begin{bmatrix} 4\ x\ 1 \\ vector \end{bmatrix} = Kp \begin{bmatrix} 4\ x\ 4 \\ matriz \\ diagonal \\ positiva \end{bmatrix} \tilde{q} + Kd \begin{bmatrix} 4\ x\ 4 \\ matriz \\ diagonal \\ positiva \end{bmatrix} (\dot{\tilde{q}}) + g \begin{bmatrix} 4\ x\ 1 \\ vector \end{bmatrix}$$

Parámetros físicos como el largo de los eslabones, masas e inercias que son utilizados para el cálculo del vector g(q) y los niveles de torque máximos permitidos para cada junta se pueden observar en la tabla N°3.

Tabla 3: Parámetros físicos del robot Kuka YouBot

Parámetro del Actuador	Junta 1	Junta 2	Junta 3	Junta 4	Junta 5
Voltaje nominal (V)	24	24	24	24	24
Corriente nominal (A)	2.32	2.32	2.32	1.07	0.49
Torque Nominal (mNm)	82.7	82.7	82.7	58.8	-
Inductancia Terminal (mH)	0.573	0.573	0.573	2.24	7.73
Resistencia Terminal (Ω)	0.978	0.978	0.978	4.48	13.7
Momento de Inercia (kg*mm^2)	13.5	13.5	13.5	9.25	3.5
Rango de velocidad (RPM)	5250	5250	5250	2850	2800
Peso (g)	110	110	110	75	46

2.3 Desarrollo de la Propuesta

2.3.1 Creación del Entorno de Simulación

El envío de mensajes hacia el robot simulado se realiza mediante el llamado a un fichero ejecutable a través de ROS, en la figura N°13 se muestra la arquitectura desarrollada para el entorno de simulación elaborado en Gazebo.

Ubuntu WSL no tiene interfaz gráfica, por lo que necesita de Xlaunch para el arranque del GUI de Gazebo, los mensajes a los nodos ROS conectados se envían desde un nuevo terminal diferente al que contiene al maestro, se puede probar la conexión entre nodos mediante “rostopic list” e incluso obtener información acerca de un nodo especifico mediante “rosnode info”.

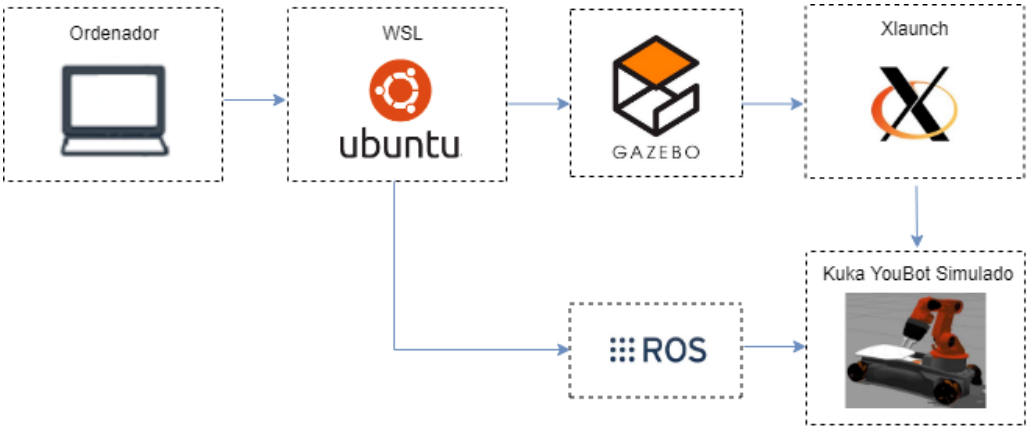

Figura N° 13: Arquitectura de la Simulación en Gazebo

Fuente: Elaborado por el Investigador

Se debe utilizar el comando "***touch***" dentro de un nuevo terminal en el espacio de trabajo catkin para crear dos ficheros, uno con extensión .world y otro .launch, estos corresponden al entorno de simulación y al ejecutable de éste respectivamente, ambos trabajan simultáneamente de forma que el uno se ejecuta bajo la orden o llamado del otro, el primero de estos contendrá todo lo referente a la simulación y será escrito bajo formato SDF en la versión 1.6, el lenguaje utiliza netamente librerías descritas en C++.

Para la edición de los dos ficheros necesarios para la simulación, dentro de la consola de comandos se usa "***nano***" más el nombre del archivo, para el entorno se debe incluir varios componentes básicos como: iluminación, fuerza de gravedad, una superficie de apoyo para el robot a simular y de ser necesario coeficientes de fricción.

Adicionalmente en la simulación se debe añadir al entorno el robot manipulador Kuka YouBot, extrayéndolo del repositorio, al seleccionarlo del repositorio de Gazebo se crea dentro del fichero .world todo el código necesario para el funcionamiento en el entorno a simular, con excepción del plugin que permita la lectura de sensores y escritura de ordenes mediante comando por consola.

El fichero .launch o .launcher deberá contener todas las instrucciones para el arranque de la simulación, entre las cuales se encuentran: el tiempo de simulación, la interfaz gráfica de usuario, si se encuentra arrancada o pausada, la dirección del archivo .world que se desea simular y finalmente el nombre de éste.

Los componentes adicionales que se deban incluir en la simulación pueden ser agregados de dos formas, extrayendo del repositorio los elementos deseados o desarrollando el código del objeto mediante el formato SDF, en la figura N°14 se

presentan los pasos necesarios a seguir para la creación de un entorno de simulación en Gazebo.

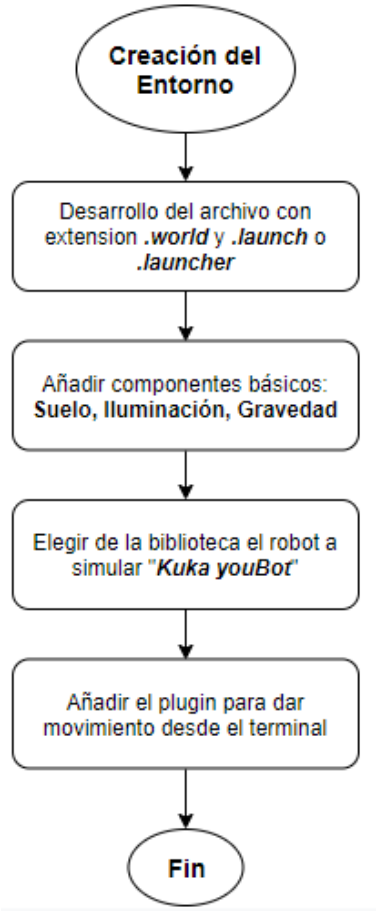

Figura N° 14: Diseño y creación del entorno de simulación en Gazebo

Fuente: Elaborado por el Investigador

La creación de plugins en Gazebo se basa en la utilización del API de este simulador, para la cual se deben crear ficheros que permitan leer datos de la simulación y escribir datos sobre el robot manipulador.

El API de Gazebo incluye diferentes clases y librerías que permiten la programación de la interfaz gráfica del modelo, estas pueden ser acerca de eventos que afectan el modelo a simular, así también como la implementación de sensores, el renderizado 3D de los elementos simulados, además de incluir las clases de física y dinámica para la aplicación de fuerzas e inercias.

El plugin al tener todos sus archivos programados en C++ puede llamar o referenciar a varias clases pertenecientes a diferentes librerías que existen dentro del mismo

espacio de trabajo, lo que permite que el desarrollo de este sea dependiente, es decir que un fichero requiera de otro y a su vez este último se integre con otros externos.

En la figura N°15 se establece el orden necesario para el desarrollo de un plugin en Gazebo mediante la programación en la API de este.

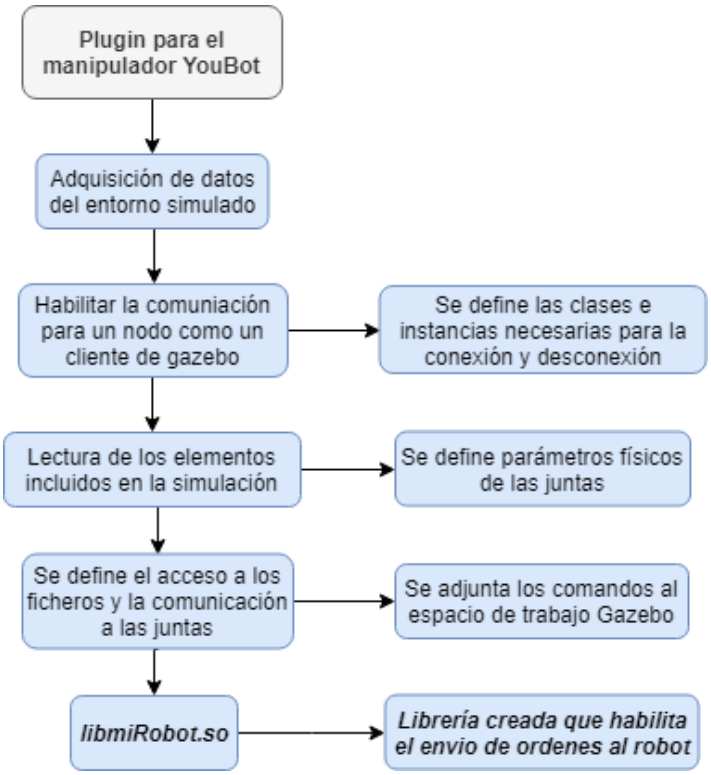

Figura N° 15: Desarrollo del Plugin en Gazebo

Fuente: Elaborado por el Investigador

La creación de los ficheros pertenecientes al plugin para el manipulador Kuka YouBot son parte de la librería "**libmiRobot.so**", creada automáticamente en el directorio **devel** del espacio de trabajo catkin, esta permite el envío de comandos que generen movimiento sobre el robot, ya sea el accionamiento de una sola junta o de una tarea propuesta a desarrollar.

La librería desarrollada se ingresa directamente en el fichero .world y se compila todo el espacio de trabajo para que finalmente esta sea ejecutable.

2.3.2 Realización de la tarea "pick and place" dentro del entorno de Simulación

Se desarrolla un fichero programado en C++ estableciendo valores numéricos para las posiciones que se enviarán a las juntas rotatorias, previamente en la parametrización de las características físicas se establece cada una de ellas dentro de un vector de 5

elementos, permitiendo el acceso a la información que estos poseen, en la figura N°16 se presentan los pasos necesarios para el desarrollo del fichero ejecutable con la operación planteada.

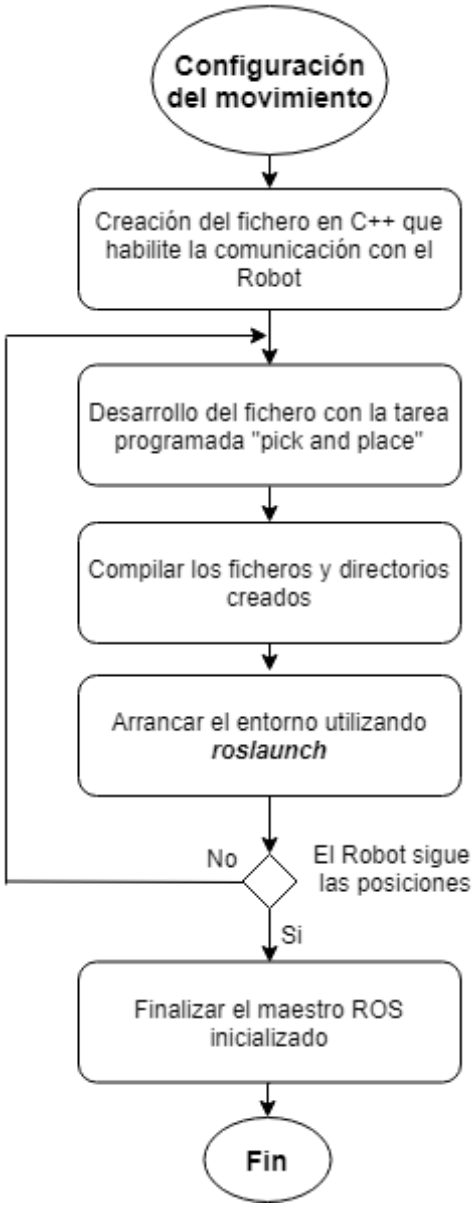

Figura N° 16: Creación de tareas para el Robot

Fuente: Elaborado por el Investigador

Se debe tener en cuenta que a las juntas rotatorias no se las debe forzar, es decir se debe evitar de cualquier forma el envío de valores posicionales que superen los límites ya definidos por el sistema, en la tabla N°4 se muestra los rangos posicionales y la denominación proporcionada por las diferentes clases de la API para cada junta perteneciente al brazo robótico.

Tabla 4: Límites rotacionales de cada junta

Fuente: Elaborado por el Investigador

Número de Junta	Denominación	Límite Inferior	Límite Superior
1	arm_joint_1	0.0100692	5.84014
2	arm_joint_2	0.0100692	2.61799
3	arm_joint_3	-5.02655	-0.015708
4	arm_joint_4	0.0221239	3.4292
5	arm_joint_5	0.110619	5.64159

Se define dentro del programa desarrollado todas las posiciones en el orden establecido, adicionalmente se debe incluir descansos entre movimientos para que estos no sean bruscos ya que en el entorno real se pueden dar fallos sobre los motores del manipulador, para la tarea planteada se configura cada una de las juntas de forma que el brazo robótico alcance las posiciones requeridas.

A través de la clase JointPositions incluida en el API del brazo robótico, se establece como resultado de la simulación los valores numéricos deseados para cada junta, los cuales corresponderán a las dos posiciones requeridas por el proceso “pick and place”, posteriormente se define para cada posición mencionada el vector resultante además del estado “Home”.

$$Home = \begin{vmatrix} arm\_joint\_1 \\ arm\_joint\_2 \\ arm\_joint\_3 \\ arm\_joint\_4 \\ arm\_joint\_5 \end{vmatrix} = \begin{vmatrix} 0.11 \\ 0.11 \\ -0.11 \\ 0.11 \\ 0.111 \end{vmatrix}$$

$$Posición\ 1 = \begin{vmatrix} arm\_joint\_1 \\ arm\_joint\_2 \\ arm\_joint\_3 \\ arm\_joint\_4 \\ arm\_joint\_5 \end{vmatrix} = \begin{vmatrix} 3 \\ 2.1 \\ -1 \\ 1 \\ 2.9 \end{vmatrix}$$

$$Posición\ 2 = \begin{vmatrix} arm\_joint\_1 \\ arm\_joint\_2 \\ arm\_joint\_3 \\ arm\_joint\_4 \\ arm\_joint\_5 \end{vmatrix} = \begin{vmatrix} 1.5 \\ 2.1 \\ -1 \\ 1 \\ 2.9 \end{vmatrix}$$

Una vez ejecutado el entorno de simulación y con los paquetes construidos y compilados se ejecuta mediante “rostopic pub” el envío de comandos desde el terminal hacia los nodos disponibles.

En la figura N°17 se puede observar el robot Kuka YouBot simulado, finalizando la tarea de posicionamiento diseñada.

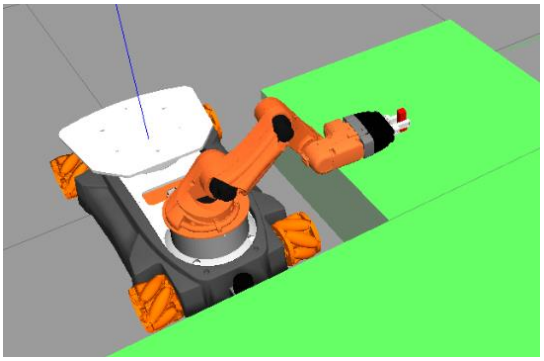

Figura N° 17: Brazo robótico depositando el elemento en su segunda posición

Fuente: Elaborado por el Investigador

Para visualizar el flujo de información, además de los topics y nodos presentes en la simulación, se utiliza la herramienta “rqt_graph”, parte del paquete de instalación de ROS, la cual crea una ilustración dinámica del funcionamiento de un sistema, en la figura N°18 se presenta el gráfico resultante del sistema desarrollado en Gazebo para el proceso “pick and place” planteado.

Figura N° 18: Gráfico del flujo de información en la simulación.

Fuente: Elaborado por el Investigador

Se puede observar los nodos ROS en colores azul y verde, los cuales corresponden a la orden enviada desde el terminal a través del plugin desarrollado y a la interfaz gráfica de Gazebo respectivamente, en color rojo se presenta el topic youbot, que corresponde al robot dentro del entorno de simulación, sobre el cual se publica el envío de instrucciones.

2.3.3 Construcción de Paquetes para la Raspberry Pi

La mayoría de los paquetes requeridos por el controlador del robot manipulador Kuka youBot se han instalado simultáneamente con ROS Indigo, sin embargo, se debe clonar y construir mediante fuente en un diferente espacio de trabajo catkin los paquetes disponibles en repositorios de Git-Hub, los cuales corresponden directamente a los drivers del robot y a paquetes que causan dependencia sobre estos.

Para establecer la comunicación con el brazo robótico se desarrolla la programación mediante la API del manipulador, esta requiere de varios controladores que deben ejecutarse simultáneamente y de forma continua para que no exista desconexión al momento del envió de ordenes desde la tarjeta empotrada, dado que no existen paquetes pre-compilados para Raspbian se debe instalar y construir todos los necesarios, pero de forma individual e inicialmente clonándolos de los repositorios originales de Kuka.

En la imagen N°19, se describe los paquetes necesarios para el funcionamiento del robot manipulador mediante el control diseñado bajo la API de youBot.

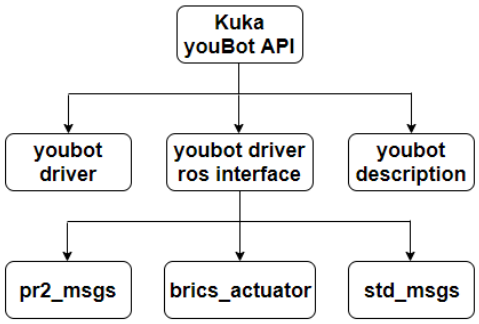

Figura N° 19: Requerimientos para el uso de la API de YouBot

Fuente: Elaborado por el Investigador

- **Kuka YouBot API**

El API proporciona las librerías principales y los diferentes ficheros de configuración para el sistema de control del robot, esta se representa como una combinación de subsistemas funcionales desacoplados, es decir se define el brazo y la plataforma como el conjunto de diferentes uniones, dentro de la API se encuentran tres clases principales utilizadas para el control del Manipulador, la Base y Ambas simultáneamente.

Adicionalmente el API proporciona una jerarquía donde se incluye el diagnóstico de los esclavos conectados al puerto Ethercat, brindando la posibilidad de obtener

parámetros eléctricos sobre los motores de las juntas rotatorias, de forma que se pueda trabajar siempre bajo óptimas características, procurando el correcto funcionamiento del brazo robótico.

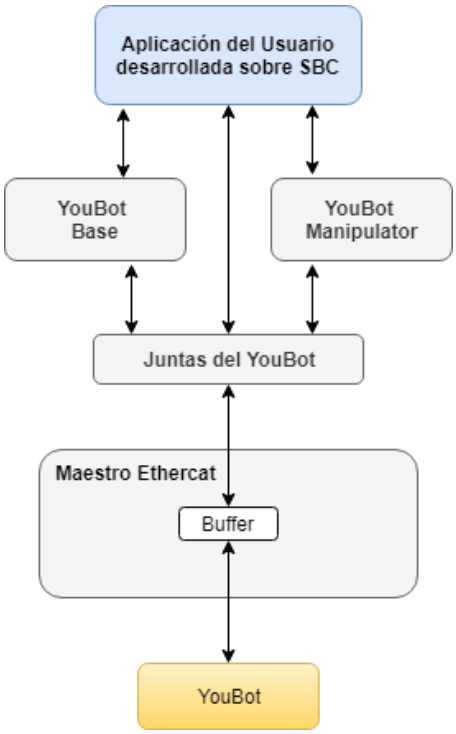

Figura N° 20: Flujo de comunicación desde la tarjeta SBC hacia el robot

Fuente: Elaborado por el Investigador

En la figura N°20, se representa de forma resumida la arquitectura del funcionamiento de la API de YouBot, las flechas representan el flujo de datos el cual se da a través de la comunicación EtherCAT, la transferencia de datos entre el usuario y el robot se realiza mediante un buffer no bloqueante, es decir se puede leer, escribir y obtener respuesta de forma inmediata.

- **Youbot driver**

  Representa la clase principal de la API del youBot, permite el control del puerto Ethernet de la tarjeta SBC mediante el fichero de configuración de la comunicación EtherCat, además referencia el acceso a los subsistemas de la plataforma, del brazo y del gripper por medio de la definición de los parámetros de las juntas rotatorias.

- **Youbot driver ROS Interface**

  Define las clases y paquetes del driver de youBot para el control a través de ROS, además contiene el fichero ejecutable que permite la inicialización del driver para el handshake entre la tarjeta SBC y el brazo robótico a través de TCP.

- **brics_actuator**

  Paquete que contiene todos los tipos de mensajes que pueden ser enviados hacia las juntas del robot, entre los principales se encuentran los de posición, torque y velocidad.

- **std_msgs**

  Contiene el tipo de dato que se va a enviar mediante las aplicaciones de usuario desarrolladas en C++, estos pueden ser de tipo string, flotante, bool, entero, entre otros.

- **pr2_msgs**

  Este paquete define mensajes y tipos de servicio específicos de PR2, forma parte de los componentes requeridos por catkin.

- **Youbot Description**

  La integración de ROS sobre el Kuka youBot incluye un modelo del robot en Formato de Descripción de Robot Unificado (URDF), este paquete describe la geometría de varios elementos y cadenas cinemáticas.

2.3.4 Control del brazo robótico

Previo al control del brazo robótico se definen los nodos y topics ROS que serán parte del sistema, donde cuentan la aplicación del usuario, el nodo controlador y el topic del hardware de youBot, en la figura N°21 se presenta la estructura desarrollada.

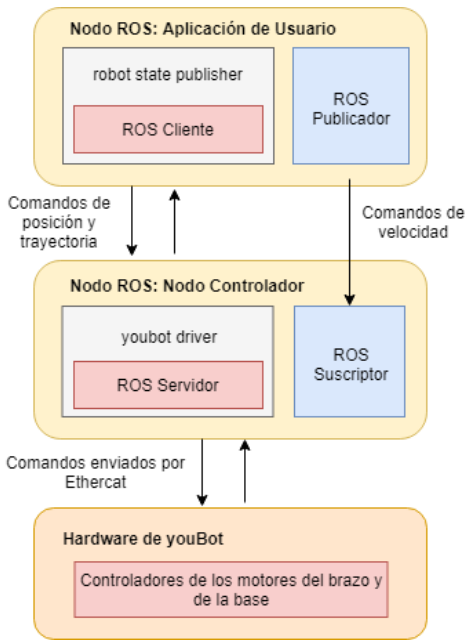

Figura N° 21: Nodos existentes dentro del sistema planteado

Fuente: Elaborado por el Investigador

Para dar movimiento al brazo robótico dentro del espacio de trabajo catkin donde se han clonado y construido los paquetes mencionados en la figura N°17, se crea un nuevo paquete ROS, donde se deberá hacer la recursividad dentro del fichero CMake con los paquetes previamente instalados, adicionalmente se desarrolla el archivo ejecutable con la programación requerida para el envío de ordenes hacia el robot, en la figura N°22 se muestran los pasos requeridos para habilitar el control del brazo robótico.

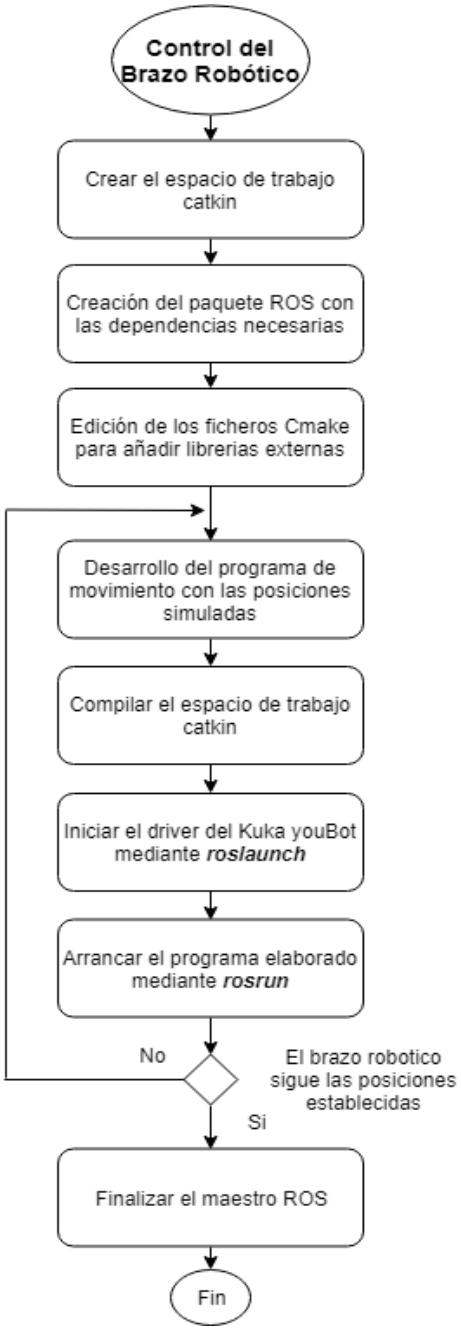

Figura N° 22: Ejecución de ordenes sobre el robot manipulador

Fuente: Elaborado por el Investigador

Se considera al brazo robótico de 5 juntas rotatorias como un vector de 5 elementos, donde cada uno corresponde a cada junta, a estos se les debe asignar valores numéricos, de tipo float, para el control del gripper se debe enviar el mismo valor a cada parte de la pinza, es decir el mismo valor deberá ser asignado a las partes izquierda y derecha del efector final.

```
brics_actuator::JointPositions createArmPositionCommand(std::vector<double>& newPositions) {
        int numberOfJoints = 5;
        brics_actuator::JointPositions msg;
        if (newPositions.size() < numberOfJoints)
                return msg;
        for (int i = 0; i < numberOfJoints; i++) {
         brics_actuator::JointValue joint;
         joint.timeStamp = ros::Time::now();
         joint.value = newPositions[i];
         joint.unit = boost::units::to_string(boost::units::si::radian);
         std::stringstream jointName;
         jointName << "arm_joint_" << (i + 1);
         joint.joint_uri = jointName.str();
                 msg.positions.push_back(joint);
          }
        return msg;
}
```

A

```
brics_actuator::JointPositions createGripperPositionCommand(double newPosition) {
         brics_actuator::JointPositions msg;
         brics_actuator::JointValue joint;
         joint.timeStamp = ros::Time::now();
         joint.unit = boost::units::to_string(boost::units::si::meter);
         joint.value = newPosition;
         joint.joint_uri = "gripper_finger_joint_l";
         msg.positions.push_back(joint);
         joint.joint_uri = "gripper_finger_joint_r";
         msg.positions.push_back(joint);
         return msg;
}
```

B

Figura N° 23: Creación de Mensajes

Fuente: Elaborado por el Investigador

El código presentado en la figura N°23 muestra la creación de mensajes para el envío de posiciones hacia las juntas rotatorias y el gripper.

Figura 23.A. Se crea el mensaje de tipo brics_actuator, haciendo uso de la librería JointPositions, debido a que se tienen 5 grados de libertad, se desarrolla un vector de 5 elementos, donde cada elemento deberá estar en formato ***double***, ya que cada posición dentro de la API se designa en radianes, adicionalmente se le asigna un nombre a cada elemento correspondiente a cada una de las juntas, estos nombres deben coincidir con los presentados en el driver, finalmente al enviarse un espacio del vector vacío, se le asigna automáticamente el valor de cero.

Figura 23.B. El mensaje para el gripper es diferente debido a que sus unidades son en metros en lugar de radianes como lo es para todas las juntas del brazo robótico, el tipo de dato que se enviará será de tipo ***double***, por la existencia de decimales, el valor que se enviará se replica para ambos dedos del gripper, de forma que se muevan en la misma cantidad y de forma simétrica.

Se realiza la conexión de la tarjeta empotrada y del brazo robótico del manipulador Kuka youBot, además de la alimentación de estos, en la figura N°24 se presenta el resultado del medio físico de comunicación.

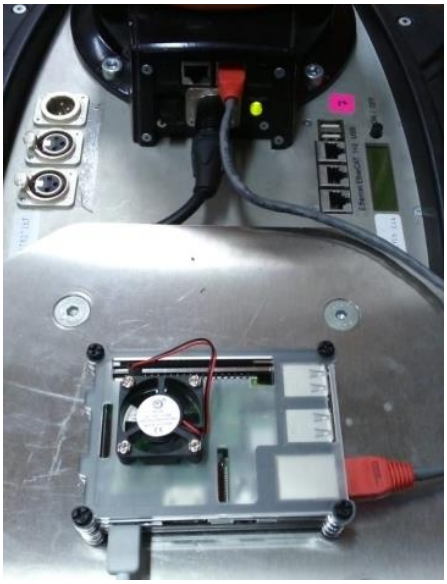

Figura N° 24: Conexión resultante del caso de estudio planteado

Fuente: Elaborado por el Investigador

Para probar el estado de la comunicación, inicialmente se puede enviar directamente en el terminal mensajes haciendo uso del paquete brics_actuator, referenciando el cambio de posición en las juntas, esta acción controla una junta rotatoria a la vez y se le asignara un valor numérico que representa la posición expresada en radianes.

De igual manera que en la simulación se desarrolla un programa en C++ con posiciones ya establecidas, se utiliza al igual que el entorno simulado un vector de 5 elementos para el acceso a cada junta, para que se produzca movimiento sobre el brazo se envía los mismos valores numéricos simulados al elemento correspondiente a cada junta, se añaden ligeros descansos entre ejecuciones para que el brazo no genere un movimiento brusco que lleve a la falla o daño sobre sus motores internos.

Finalmente se realiza la conexión de alimentación y Ethernet entre el brazo robótico y las Raspberry Pi, posteriormente en el terminal de comandos se ejecuta el controlador, lo cual permite el arranque del maestro ROS, además de la lectura y calibración de las juntas disponibles, si todo el sistema ha sido compilado sin errores, en el terminal actual se deberá presentar el mensaje de inicialización del brazo robótico denominado por el driver como "youbot-manipulator", como se muestra en la figura N°25.

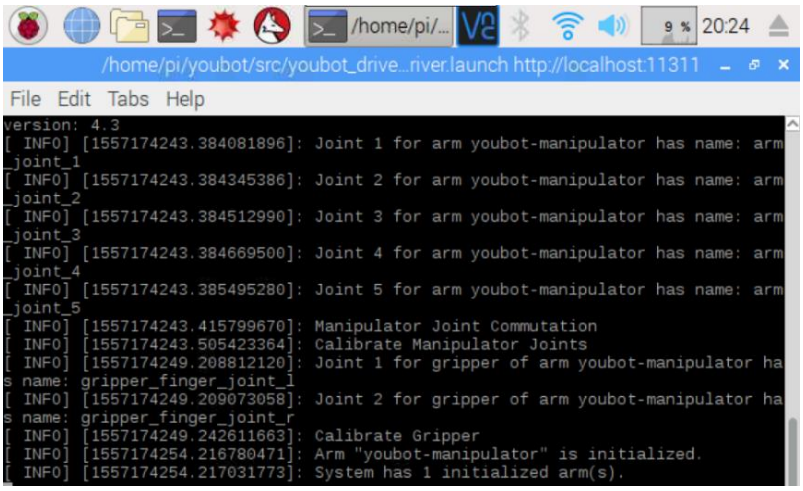

Figura N° 25: Inicialización del Brazo Robótico en Raspbian

Fuente: Elaborado por el Investigador

Una vez que se ha realizado una correcta conexión con el brazo robótico y se ha ejecutado el fichero desarrollado para el proceso "pick and place" se ejecuta la herramienta "rqt_graph" para obtener el gráfico dinámico del flujo de información a través de los nodos inicializados mediante la suscripción y publicación de topics, en la figura N°26 se presenta el diagrama resultante de los nodos ROS.

Figura N° 26: Gráfico del Flujo de comunicación sobre el brazo robótico físico

Fuente: Elaborado por el Investigador

De igual forma al gráfico obtenido de la simulación, se presentan los nodos en colores azul y verde, estos corresponden al driver del youBot y al nodo de servicio que concatena el estado de las juntas con el envío de mensajes hacia las mismas, respectivamente, en color rojo se visualiza el topic, a este se encuentra suscrito el youbot driver, y a través de este se genera la publicación de mensajes de tipo JoinPositions.

2.4 Métodos

Proyecto de investigación aplicada, debido a la dependencia de información y conocimientos previos necesarios para el desarrollo de este, requiriendo de las siguientes modalidades de investigación:

2.4.1 Investigación de Campo

Debido a que la implementación se realizará en los laboratorios de la Facultad de Ingeniería en Sistemas Electrónica e Industrial de la Universidad Técnica de Ambato, utilizando el respectivo manipulador Kuka youBot.

2.4.2 Investigación Bibliográfica – Documental

Porque se buscó información en artículos científicos, tesis, libros y en páginas web, que ayudarán al cumplimiento de los objetivos planteados.

2.4.3 Investigación Experimental

Debido a que se realizó pruebas de funcionamiento hasta obtener el correcto funcionamiento del sistema de control implementado sobre el brazo robótico del Kuka youBot.

CAPÍTULO III
RESULTADOS Y DISCUSIÓN

3.2 Análisis y Discusión de los Resultados

De un total de 20 ejecuciones de la tarea “Pick and Place”, enviadas desde la tarjeta empotrada Raspberry Pi hacia el brazo robótico del robot manipulador Kuka youBot, se obtuvo el tiempo de cumplimiento del total del proceso, es decir desde el momento en que la orden es enviada hasta que el brazo cumple con la recolección y depósito del elemento a posicionar.

En la tabla N°5 se presenta el total de ejecuciones efectuadas con su respectivo tiempo obtenido por ejecución.

Tabla 5: Tiempos obtenidos por Ejecución

Fuente: Elaborado por el Investigador

Ejecuciones	Tiempo por Ejecución	Ejecuciones	Tiempo por Ejecución
1	26,68	11	26,67
2	26,78	12	26,67
3	26,62	13	26,86
4	26,68	14	26,68
5	26,71	15	26,69
6	26,67	16	26,71
7	26,79	17	26,54
8	26,66	18	26,67
9	26,78	19	26,71
10	26,63	20	26,70

Con los datos recolectados del proceso “Pick and Place” se calcula la desviación estándar para determinar la variación o dispersión existente, cabe recalcar que al utilizar una conexión de Red entre la tarjeta SBC y el brazo robótico mediante Ethernet, la velocidad de transmisión de datos se limita al de la tarjeta, la cual en su versión Raspberry Pi 3 modelo B, posee Ethernet de alta velocidad, lo cual permite

una capacidad de procesamiento teórica de 100 Mbps (megabits por segundo), junto a la utilización de ROS, la comunicación se vuelve bidireccional entre nodos, evitando así la perdida de información y la desconexión entre el controlador y el brazo robótico.

En la figura N° 27 se puede observar gráficamente la cantidad de tiempo obtenido para cada ejecución enviada hacia el brazo robótico.

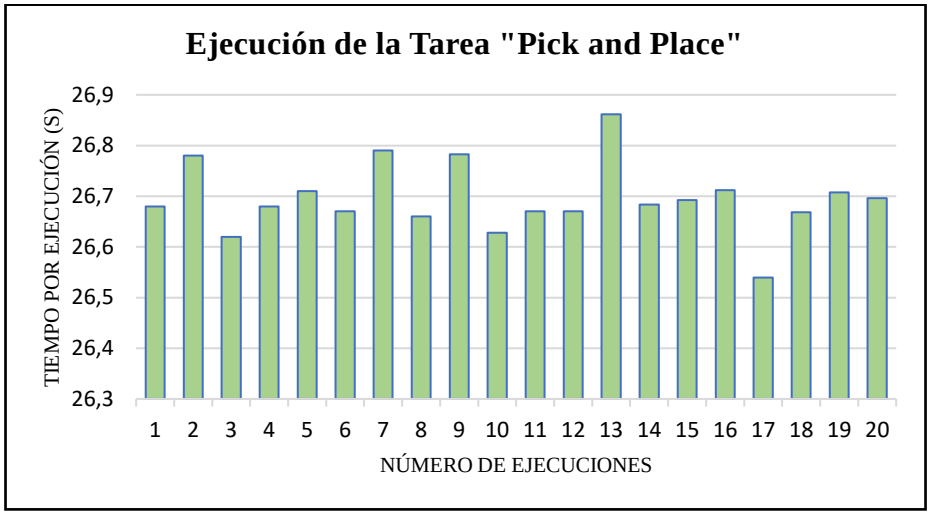

Figura N° 27: Representación gráfica del Tiempo obtenido por Ejecución

Fuente: Elaborado por el Investigador

3.2.1 Desviación Estándar

Se utiliza para determinar la dispersión entre los datos recolectados respecto de la media, donde un valor bajo de este indica una menor dispersión, siendo un buen indicador de que el proceso se ejecuta correctamente.

$$\boldsymbol{S} = \sqrt{\frac{\sum_{\boldsymbol{n}}^{\boldsymbol{i=1}}(\boldsymbol{xi} - \bar{\boldsymbol{x}})^{\boldsymbol{2}}}{\boldsymbol{n-1}}} \qquad (9)$$

Donde:

n = Tamaño de la muestra

$\bar{x}$ = Media aritmética de la muestra

Mediante la ecuación N°9 y de los datos obtenidos se obtiene el valor de la desviación estándar, adicionalmente en la figura N°28 se representa gráficamente la dispersión de los datos.

$$\boldsymbol{S} = 0.06871$$

El valor resultante indica que el brazo robótico tiende a ejecutar la tarea de recolección y posicionamiento de objetos de una forma adecuada y precisa, esto se debe a la alta tecnología de comunicación que posee el sistema de control implementado, con lo cual se demuestra que la variación en los datos es en mayor parte por el método de cronometraje de la duración del proceso.

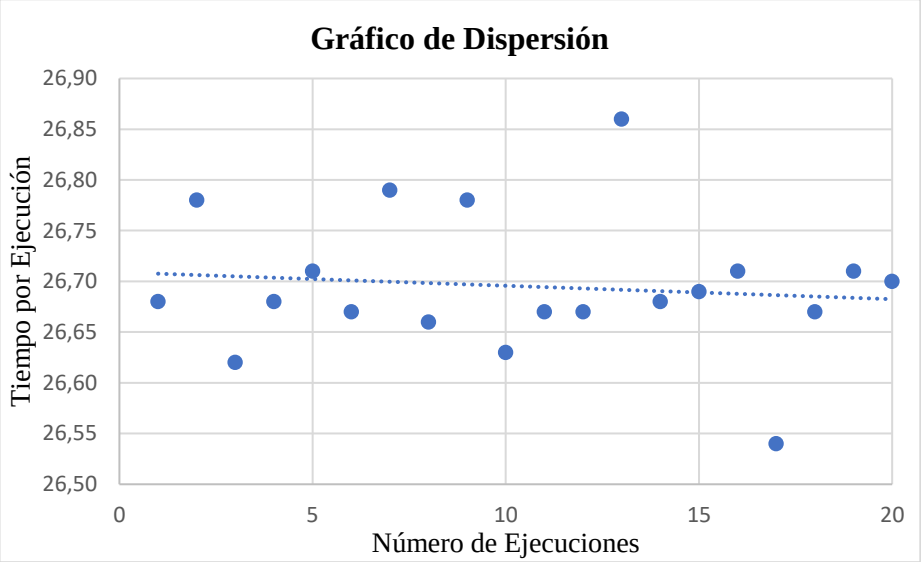

Figura N° 28: Representación gráfica de la dispersión de los datos obtenidos

Fuente: Elaborado por el Investigador

Se puede observar que no existe una gran dispersión en los datos obtenidos, esto debido a que la diferencia entre el mayor y menor tiempo existente es de 0.32seg.

3.2.1 Desempeño de la Raspberry Pi con la integración de ROS

De manera resumida se presenta el consumo de los recursos hardware y software de la tarjeta Raspberry Pi 3B, se monitorea el rendimiento del CPU integrado durante un periodo de 45 minutos dentro de los cuales inicialmente se arranca el maestro ROS con el driver del Kuka YouBot configurado para el control únicamente del brazo robótico, y de manera constante se ejecuta la tarea de posicionamiento de objetos, se debe destacar el óptimo funcionamiento del sistema desarrollado puesto que no se finaliza el enlace de comunicación en ningún momento y la tarjeta únicamente se exige como máximo hasta la tercera parte de su capacidad de procesamiento.

La figura N°29 representa de forma resumida el consumo de recursos CPU de la Raspberry Pi, durante 45min de ejecución de la operación “Pick and Place” planteada como caso de estudio.

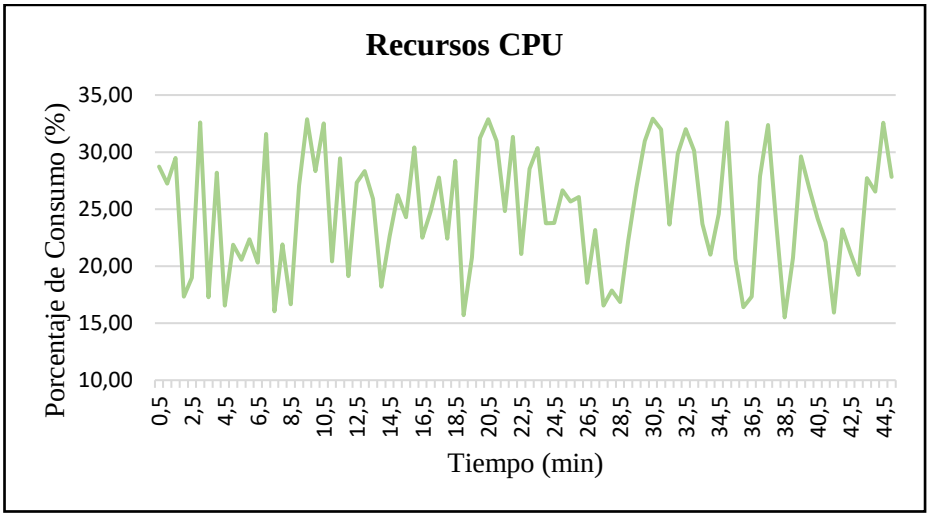

Figura N° 29: Consumo del CPU de la Raspberry Pi 3

Fuente: Elaborado por el Investigador

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- La implementación de la tarjeta empotrada Raspberry Pi como dispositivo controlador del brazo robótico del robot manipulador Kuka YouBot, el cual se planteó como caso de estudio, presenta un medio eficaz de control pues permite la integración de dos sistemas diferentes cuyas plataformas son compatibles únicamente debido a que trabajan bajo Open Source y se encuentran basadas en software libre, lo cual hace posible la interoperabilidad y el uso multiplataforma de manera que no se necesita de recursos hardware desarrollados exclusivamente por una específica casa comercial, convirtiendo a la tarjeta SBC en una alternativa confiable y de bajo costo para el control de varios procesos cuyas características son de tipo industriales, dando apertura al desarrollo de sistemas ciber-físicos más robustos que sustenten las necesidades de la Industria 4.0.

- Mediante la simulación del robot manipulador Kuka YouBot desarrollada en Gazebo, se obtuvo los valores numéricos requeridos para el posicionamiento del brazo robótico y del objeto a manipular, los rangos rotacionales de las juntas y la topología de las mismas, además este simulador permite entornos físicamente reales, por lo cual la ejecución de la tarea planteada como caso de estudio se realizó sobre el brazo robótico bajo condiciones similares a las simuladas, además de simular el trabajo realizado por el robot manipulador, se habilitó la comunicación a través de ROS, por lo cual el envío de mensajes hacia el robot simulado fue codificado exactamente igual que para el sistema de control desarrollado para el brazo robótico, solo diferenciándose en la utilización de los paquetes construidos para Raspbian, de esta forma la simulación fue necesaria para el control final debido a que ambos entornos son estructuralmente funcionales bajo la misma arquitectura.

- La instalación de ROS sobre Raspbian y de los drivers de la interfaz ROS del robot manipulador Kuka YouBot permite el establecimiento de la comunicación entre estos, dando lugar al control del brazo robótico mediante la tarjeta SBC, estos deben satisfacer todas las dependencias y librerías necesarias para una compilación exitosa, el sistema de control desarrollado ejecuta ROS dentro de la Raspberry, a pesar de ser un framework con gran cantidad de paquetes, el consumo de memoria sobre la tarjeta es mínimo como se puede observar de los resultados del desempeño con la integración de ROS, dando lugar a una óptima comunicación con el brazo robótico, permitiendo que la comunicación entre nodos ROS sea adecuada para la suscripción y publicación de topics, la tarjeta microcontroladora es encargada de procesar varias acciones además de inicializar el maestro ROS, debe dar lectura de los nodos conectados mediante el puerto Ethernet y permite el acceso a las juntas definidas en la configuración del driver del Kuka YouBot.

- Debido a que Gazebo trabaja con ROS y además cuenta con el robot Kuka YouBot en su repositorio, se obtiene del API los rangos rotacionales de las juntas, valores propios del diseño del brazo robótico, de las pruebas realizadas en el entorno de simulación, se obtuvo los valores numéricos para cada junta, de forma que se cumpla con las trayectorias requeridas por el proceso planteado “pick and place”, esto se define con la premisa de no forzar los motores del brazo robótico físico, para el entorno real se desarrolló dentro del espacio de trabajo catkin un paquete ROS, dentro del cual se define las librerías externas necesarias para el envío de ordenes hacia cada junta mediante el driver del Kuka YouBot, al igual que en la simulación se crea un vector de cinco elementos, dentro del cual cada elemento corresponderá a cada una de las juntas, a su vez el terminal que se encuentra ejecutando el controlador del brazo robótico permite la lectura constante de los valores numéricos asignados, lo que permite tener un mayor control sobre el sistema desarrollado.

4.2 Recomendaciones

- Al trabajar con ROS el cual es un framework que se actualiza cada año para la corrección de errores y mejorar el rendimiento de sus aplicaciones, se requiere de paquetes y dependencias actualizadas, lo cual no ocurre con la mayoría de repositorios, los mismos que se han mantenido con las primeras versiones que fueron desarrollados, al estar discontinuados los controladores del robot manipulador Kuka YouBot se recomienda el generar una investigación que actualice los drivers, lo cual conlleva al nivel más básico de programación, es decir re escribir los paquetes disponibles de manera que pueda trabajar con versiones de ROS más estables, permitiendo el desarrollo de mejores y más novedosas aplicaciones.

- Para que la tarjeta Raspberry Pi se mantenga trabajando de forma óptima, evitando el sobrecalentamiento del CPU y un posible colapso del procesamiento, se recomienda acondicionar disipadores de calor, en el caso de que se requiera varias horas de trabajo continuo, de igual forma el acoplamiento de carcasas protectoras, esto dependiendo del área de trabajo donde la tarjeta será instalada, al tratar del sistema operativo se recomienda que el trabajo de compilación se ejecute un proceso a la vez, ya que si se ejecutan varios, estos puede colapsar la memoria de la tarjeta y concurrir en el apagado de la misma, lo cual daña los paquetes que no se hayan construido en su totalidad.

- Al trabajar con ROS se recomienda tener conocimiento en el desarrollo de espacios de trabajo catkin, pues es la herramienta más utilizada para la creación y compilación de paquetes de tipo Open Source, gracias al orden que brinda y al alto nivel de corrección de errores en ficheros individuales y en enlaces entre librerías y dependencias, adicionalmente se recomienda la utilización de software de código abierto para el desarrollo de aplicaciones que puedan de ser tipo multiplataforma.

C. MATERIALES DE REFERENCIA

Bibliografía

[1] L. Wang, M. Törngren, and M. Onori, "Current status and advancement of cyber-physical systems in manufacturing," *J. Manuf. Syst.*, vol. 37, pp. 517–527, 2015.

[2] A. Valera, A. Soriano, and M. Vallés, "Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica," *RIAI - Rev. Iberoam. Autom. e Inform. Ind.*, vol. 11, no. 4, pp. 363–376, 2014.

[3] P. González-Nalda, I. Calvo, I. Etxeberria-Agiriano, E. Zulueta, and J. M. López-Guede, "Hacia un framework basado en ROS para la implementación de Sistemas Ciberfísicos," *XXXVI Jornadas de Automática*, pp. 1050–1057, 2015.

[4] J. G. Pérez, "Introducción a ROS en Raspberry Pi," Universidad Abierta de Cataluña, 2017.

[5] B. Abia and Z. Casanova, "Desarrollo de un sistema de navegación para un robot móvil," 2014.

[6] M. Gerwitz, "GNU/kWindows," 2016. [Online]. Available: https://mikegerwitz.com/2016/04/gnu-kwindows. [Accessed: 05-Apr-2019].

[7] M. Harsh, "Run Bash on Ubuntu on Windows - Windows Developer Blog," 2016. [Online]. Available: https://blogs.windows.com/buildingapps/2016/03/30/run-bash-on-ubuntu-on-windows/. [Accessed: 05-Apr-2019].

[8] K. Finley, "Why Microsoft Making Linux Apps Run on Windows Isn't Crazy | WIRED," 2016. [Online]. Available: https://www.wired.com/2016/03/microsoft-making-linux-apps-run-windows-isnt-crazy/. [Accessed: 05-Apr-2019].

[9] A. Araújo, D. Portugal, M. S. Couceiro, and R. P. Rocha, "Integrating Arduino-Based Educational Mobile Robots in ROS," *J. Intell. Robot. Syst. Theory Appl.*, vol. 77, no. 2, pp. 281–298, 2014.

[10] E. Dominguez, "UNA INTRODUCCIÓN AL SISTEMA OPERATIVO DEL

ROBOT (ROS) - NOTICIAS - 2019," 2016. [Online]. Available: https://es.electronics-council.com/an-introduction-robot-operating-system-92765. [Accessed: 10-Apr-2019].

[11] M. Alajlan and A. Koubâa, *Writing global path planners plugins in ROS: A tutorial*, vol. 625, no. Volume 1. 2016.

[12] T. Duc, "Philosophy Doctoral Thesis in Information Engineering USING KUKA YOUBOT FOR TEACHING ASSISTANCE Supervisors : Professor Luca Iocchi and Professor Massimo Mecella," no. November, 2014.

[13] Kuka, "KUKA youBot Research & Application Development in Mobile Robotics youBot Arm."

[14] E. Upton and G. Halfacree, *Raspberry pi guía del usuario 1*, Segunda Ed. 2016.

[15] HardZone, "Análisis: Raspberry Pi 3 Modelo B+ - HardZone." [Online]. Available: https://hardzone.es/reviews/perifericos/analisis-raspberry-pi-3-modelo-b/. [Accessed: 05-Apr-2019].

[16] Raspberry Foundation, "RaspbianAbout - Raspbian," 2016. [Online]. Available: https://www.raspbian.org/RaspbianAbout. [Accessed: 10-Apr-2019].

[17] S. Long, "Introducing PIXEL - Raspberry Pi," 2016. [Online]. Available: https://www.raspberrypi.org/blog/introducing-pixel/. [Accessed: 10-Apr-2019].

[18] Wiki ROS, "catkin/workspaces - ROS Wiki," 2017. [Online]. Available: http://wiki.ros.org/catkin/workspaces. [Accessed: 10-Apr-2019].

[19] Locomotec, *KUKA youBot User Manual*, 1st ed. 2012.

[20] C. López-Franco, J. Hernández-Barragán, A. Y. Alanis, N. Arana-Daniel, and M. López-Franco, "Inverse kinematics of mobile manipulators based on differential evolution," *Int. J. Adv. Robot. Syst.*, vol. 15, no. 1, pp. 1–22, 2018.

[21] F. G. Sales, "Control de trayectoria de la simulación de un brazo robot de 5 grados de libertad, controlado mediante la plataforma C2000 Piccolo

LAUNCHXL-F28027F," 2017.

[22] J. Guadalupe, Z. Villalpando, S. Alfonso, G. Blanco, J. José, and G. Mandujano, "Cinemática inversa , Fanuc LR Mate 200ic," no. 103, pp. 202–227, 2013.

[23] J. Hernández-Barragán, C. López-Franco, A. Y. Alanis, N. Arana-Daniel, and M. López-Franco, "Dual-arm cooperative manipulation based on differential evolution," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 1, pp. 1–20, 2019.

[24] V. N. Iliukhin, K. B. Mitkovskii, D. A. Bizyanova, and A. A. Akopyan, "The Modeling of Inverse Kinematics for 5 DOF Manipulator," *Procedia Eng.*, vol. 176, pp. 498–505, 2017.

[25] Y. Zhang, Y. Li, and X. Xiao, "A novel kinematics analysis for a 5-DOF manipulator based on KUKA youBot," *2015 IEEE Int. Conf. Robot. Biomimetics, IEEE-ROBIO 2015*, pp. 1477–1482, 2015.

[26] A. Medez, "Algoritmo de Denavit Hartenberg," 2017. [Online]. Available: https://www.studocu.com/es/document/universidad-complutense-madrid/sistemas-inteligentes/apuntes/algoritmo-de-denavit-hartenberg/1935064/view. [Accessed: 10-Jun-2019].

[27] A. Yarza, "Control de Robots Manipuladores : Análisis de Estabilidad vía Perturbaciones Singulares," no. 3, 2009.

[28] V. Santib, "Control PD de Robots : Din ´ amica de Actuadores y Nueva Sinton ´ ı a," vol. 5, pp. 62–68, 2008.

[29] R. Kelly and V. Santibáñez, "Control de Movimiento de Robots Manipuladores," in *MMW Fortschritte der Medizin*, Pearson., vol. I, 2003, pp. 143–146.

[30] L. Jon and R. Martinez, "Tema 6. dinámica de robots y control," *Open Course Ware*, p. 46, 2018.

Anexos

Anexo 1: Instalación de ROS en Raspbian

Al no existir paquetes pre-compilados para Raspbian, se debe instalar por código fuente todos y cada uno de los necesarios incluyendo ROS, los controladores del robot manipulador Kuka youBot son únicamente compatibles con las distribuciones Fuerte, Electric, Hydro e Indigo, siendo esta ultima la única de sus antecesoras en tener soporte hasta el 30 de abril del 2019,

Para la instalación de ROS Indigo se deben ingresar en un nuevo terminal la siguiente serie de comandos, es recomendable la instalación “Full Desktop” debido a que tiene una mayor cantidad de componentes.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu jessie main" > /etc/apt/sources.list.d/ros-latest.list'

wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -
```

Actualizamos la lista de paquetes disponibles para la distribución de ROS.

```
sudo apt-get update

sudo apt-get upgrade
```

Antes de empezar la instalación de ROS se debe instalar las dependencias necesarias.

```
sudo apt-get install python-pip python-setuptools python-yaml python-distribute python-docutils python-dateutil python-six

sudo pip install rosdep rosinstall_generator wstool rosinstall
```

Una vez terminados los pasos anteriores se inicializa el repositorio de ROS donde se encuentran las distribuciones disponibles.

```
sudo rosdep init

rosdep update
```

A continuación, se debe descargar y construir todos los paquetes con sus librerías.

```
mkdir ~/ros_catkin_ws

cd ~/ros_catkin_ws
```

Para la versión Desktop:

```
rosinstall_generator desktop --rosdistro indigo --deps --wet-only --exclude roslisp --t
ar > indigo-desktop-wet.rosinstall

wstool init src indigo-desktop-wet.rosinstall
```

Estos comandos descargarán por fuente los paquetes de ROS Indigo y añadirán al espacio de trabajo catkin los necesarios según la versión de ROS requerida.

Antes de construir los paquetes en el espacio de trabajo catkin se debe tener descargadas todas las dependencias necesarias, con la herramienta rosdep se puede obtener aquellas que no se encuentran disponibles en los repositorios, todas estas deben ser construidas manualmente.

Para Raspbian Jessie se necesita únicamente la instalación extra del paquete "**collada-dom-dev**".

Los siguientes comandos instalan el paquete necesario.

```
cd ~/ros_catkin_ws/external_src

sudo apt-get install libboost-filesystem-dev libxml2-dev

wget http://downloads.sourceforge.net/project/collada-dom/Collada%20DOM/Collad
a%20DOM%202.4/collada-dom-2.4.0.tgz

tar -xzf collada-dom-2.4.0.tgz

cd collada-dom-2.4.0

cmake .

sudo checkinstall make install
```

Al momento que se solicite el cambio de nombre paquete externo se debe renombrar a "**collada-dom-dev**", de otra forma el comando rosdep no lo encontrara posteriormente.

Verificamos las dependencias de la carpeta fuente dentro del espacio de trabajo e instalamos las que se hayan quedado perdidas.

Una vez que se hayan descargado y resuelto todo lo referente a las dependencias necesarias, procedemos a construir los paquetes catkin.

```
sudo ./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Rel
ease --install-space /opt/ros/indigo
```

Ahora ROS Indigo debe estar finalmente instalado, se debe añadir el entorno ROS al espacio bash, esto permitirá sea cargado automáticamente en cada nuevo terminal.

Adicionalmente se debe tener en cuenta que la tarjeta empotrada Raspberry Pi tiene 1 GB de memoria RAM, por lo cual no podrá compilar los paquetes de ROS con los comandos predeterminados, debido a que estos inicializan las banderilla "-j4", lo cual significa que se compilan 4 procesos en paralelo, es decir simultáneamente, por ende se debe ingresar el siguiente comando que compilará 1 proceso a la vez manteniendo el rendimiento del procesador dentro de los límites normales de funcionamiento, sin saturarse ni sobrecalentarse.

```
export ROS_PARALLEL_JOBS=-j1
```

Anexo 2: Instalación de Gazebo en Ubuntu WSL

Gazebo al requerir de gráficos de tipo 3D es un entorno software que consume muchos recursos en una computadora, por lo cual para el desarrollo de la simulación del Kuka youBot y la ejecución de la tarea “pick and place” previamente planteada, se ha decidido utilizar Ubuntu 18.04 montado sobre Windows, lo cual se denomina como Ubuntu WSL, con esta herramienta para programadores se tiene a disposición un espacio en la memoria del computador principal y la posibilidad de utilizar el terminal al igual que en sistemas Linux.

Para la instalación de Gazebo se sigue las instrucciones disponibles en la página principal en el apartado de tutoriales.

Se añade la descarga del repositorio.

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Obtenemos las contraseñas actualizadas para el acceso al repositorio privado.

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -

sudo apt-get update
```

Se instala la versión 9 de Gazebo, la cual es la más actual al momento y trabaja específicamente bajo ROS Melodic.

```
sudo apt-get install gazebo9
```

Al no tener entorno grafico la versión utilizada de Ubuntu, se debe apoyar sobre un software capaz de abrir ventanas a partir de servidores locales, sobre los cuales se deberán ejecutar los entornos de simulación, se instala la versión compatible con Windows 10 de Xming.

Anexo 3: Programa “Pick and Place” desarrollado como caso de estudio

El fichero es de tipo ejecutable y requiere para su llamado el uso de “rosrun”, comando que viene como parte de la instalación principal de ROS.

```
//Incluimos las librerias necesarias para la creacion de mensajes
#include "ros/ros.h"
#include "boost/units/systems/si.hpp"
#include "boost/units/io.hpp"
#include "brics_actuator/JointPositions.h"
#include "geometry_msgs/Twist.h"

ros::Publisher armPublisher;
ros::Publisher gripperPublisher;

// Creamos el mensaje de tipo brics_actuator para el brazo

brics_actuator::JointPositions
createArmPositionCommand(std::vector<double>& newPositions) {
      int numberOfJoints = 5;
      brics_actuator::JointPositions msg;
      if (newPositions.size() < numberOfJoints)
            return msg;
      for (int i = 0; i < numberOfJoints; i++) {
            brics_actuator::JointValue joint;
            joint.timeStamp = ros::Time::now();
            joint.value = newPositions[i];
            joint.unit =
boost::units::to_string(boost::units::si::radian);
            std::stringstream jointName;
            jointName << "arm_joint_" << (i + 1);
            joint.joint_uri = jointName.str();
            msg.positions.push_back(joint);
      }
      return msg;
}

//creamos el mensaje de tipo brics_actuator para el gripper
definiendo el mismo para las 2 partes de la pinza

brics_actuator::JointPositions createGripperPositionCommand(double
newPosition) {
      brics_actuator::JointPositions msg;

      brics_actuator::JointValue joint;
      joint.timeStamp = ros::Time::now();
```

```
	joint.unit = boost::units::to_string(boost::units::si::meter);
	joint.value = newPosition;
	joint.joint_uri = "gripper_finger_joint_l";
	msg.positions.push_back(joint);
	joint.joint_uri = "gripper_finger_joint_r";
	msg.positions.push_back(joint);

	return msg;
}

// abrir gripper

void abrirGripper() {

	brics_actuator::JointPositions msg;
	msg = createGripperPositionCommand(0.011);
	gripperPublisher.publish(msg);

}


// Rotamos juntas 1[0] y 5[4]

void Rotar() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[0] = 2.95;
	jointvalues[4] = 2.95;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);

	ros::Duration(4).sleep();
}

//Acercamos el brazo a la posicion deseada

void Acercar() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[1] = 2.1;
	jointvalues[2] = -1;
	jointvalues[3] = 1;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);
```

```
	ros::Duration(3).sleep();

}

// cerrar  gripper
void moveGripper() {

	brics_actuator::JointPositions msg;
	msg = createGripperPositionCommand(0);
	gripperPublisher.publish(msg);

	ros::Duration(3).sleep();

}

//Subimos levemente la junta 4 [3]

void Posicion() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[3] = 0.1;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);

	ros::Duration(4).sleep();

}

//Giramos el brazo a la posicion 2

void Girar() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[0] = 1.5;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);

	ros::Duration(5).sleep();
}

//Bajamos el brazo junto con la pieza

void Bajar() {
```

```
	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[3] = 1;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);
	ros::Duration(5).sleep();
}

//Abrimos el gripper para soltar la pieza

void Soltar() {
	brics_actuator::JointPositions msg;
	msg = createGripperPositionCommand(0.011);
	gripperPublisher.publish(msg);

	ros::Duration(2).sleep();
}

//Retiramos el brazo despues de dejar la pieza en su lugar

void Retiro() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);
	jointvalues[1] = 1;
	msg = createArmPositionCommand(jointvalues);
	armPublisher.publish(msg);

	ros::Duration(2).sleep();
}

//Para finalizar el proceso lo enviamos al home

void toHome() {

	brics_actuator::JointPositions msg;
	std::vector<double> jointvalues(5);

	jointvalues[0] = 0.11;
	jointvalues[1] = 0.11;
	jointvalues[2] = -0.11;
	jointvalues[3] = 0.11;
	jointvalues[4] = 0.111;

	msg = createArmPositionCommand(jointvalues);
```

```
	armPublisher.publish(msg);

	ros::Duration(5).sleep();
}

//Cerramos el gripper y terminamos el proceso

void Cerrar() {
	brics_actuator::JointPositions msg;
	msg = createGripperPositionCommand(0);
	gripperPublisher.publish(msg);
}

int main(int argc, char **argv) {
	ros::init(argc, argv, "pick");
	ros::NodeHandle n;

armPublisher =
n.advertise<brics_actuator::JointPositions>("arm_1/arm_controller/p
osition_command", 1);

gripperPublisher =
n.advertise<brics_actuator::JointPositions>("arm_1/gripper_controll
er/position_command", 1);
	sleep(1);

	abrirGripper();
	Rotar();
	Acercar();
	moveGripper();
	Posicion();
	Girar();
	Bajar();
	Soltar();
	Retiro();
	toHome();
	Cerrar();

	sleep(1);
	ros::shutdown();

	return 0;
}
```

Anexo 4: Fichero CMake necesario para la compilación del ejecutable

```
cmake_minimum_required(VERSION 2.8.3)
project(pick)

find_package(catkin REQUIRED COMPONENTS
  brics_actuator
  geometry_msgs
  roscpp
)

catkin_package()

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

include_directories(
  ${catkin_INCLUDE_DIRS}
)

add_executable(pick src/pick.cpp)

target_link_libraries(pick
  ${catkin_LIBRARIES}
)

install(TARGETS pick pick
  ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
  RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Anexo 5: Fichero XML, parte del paquete desarrollado, se incluyen los paquetes necesarios para la compilación del fichero ejecutable.

```
<?xml version="1.0"?>
<package><name>pick</name>
<version>0.0.0</version>
<description>TESIS</description>
<maintainer email="none">Camilo</maintainer>
<license>LGPLv3</license>
<url type="website">http://www.youbot-store.com</url>
<buildtool_depend>catkin</buildtool_depend>
<build_depend>brics_actuator</build_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>roscpp</build_depend>
<run_depend>brics_actuator</run_depend>
<run_depend>geometry_msgs</run_depend>
<run_depend>roscpp</run_depend>
</package>
```

Anexo 6: Imágenes de las pruebas realizadas sobre el robot

Se presenta las imágenes adquiridas mediante las pruebas del control realizado sobre el robot manipulador Kuka YouBot.

En la ilustración 1 se presenta el robot manipulador Kuka YouBot en su posición de calibración, además del objeto a posicionar mediante la tarea planteada en el caso de estudio.

Ilustración 1: Posición Home

En la ilustración 2 se muestra el brazo robótico recogiendo el objeto entre su pinza, la cual además es la primera posición de trabajo.

Ilustración 2: Brazo robótico en la primera posición

En la ilustración 3 se puede observar el brazo robótico depositando el objeto en la segunda posición de trabajo.

Ilustración 3: Brazo robótico en la segunda posición

En la ilustración 4 se presenta el montaje final del caso de estudio planteado.

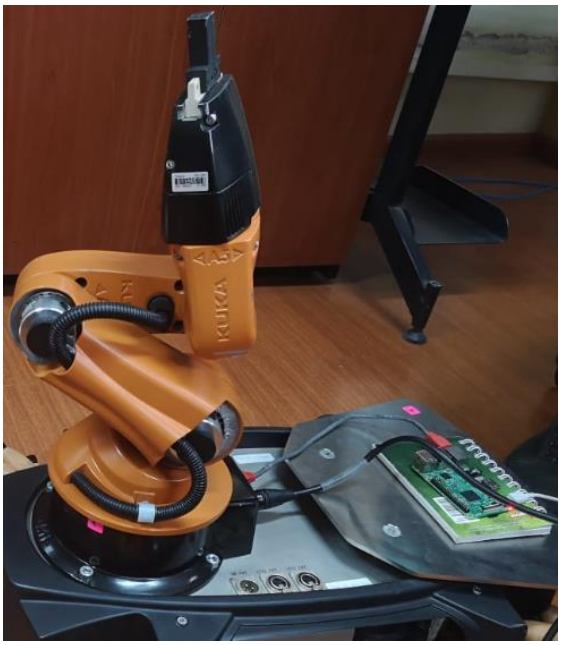

Ilustración 4: Montaje Final