



**UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN**

INFORME DE TRABAJO MONOGRÁFICO

**SISTEMA OPERATIVO PARA ROBOTS (ROS): APLICACIÓN EN EL
DESARROLLO DE UN LABORATORIO VIRTUAL PARA EL ESTUDIO
DE LOS FUNDAMENTOS DE LA ROBÓTICA INDUSTRIAL**

Presentado por: Br. Yeser Alfredo Morales Calero

Tutor: MSc. Alejandro Alberto Méndez Talavera

Agosto 30, 2019

DEDICATORIA

Dedico este trabajo monográfico a mis seres queridos quienes me apoyan incondicionalmente en todos los hitos de mi vida, a mis amigos y a mi tutor de UNI por la inspiración y paciencia otorgada.

“All things are difficult before they are easy.”

~Thomas Fuller

RESUMEN

En la actualidad la robótica industrial posee una presencia cada vez más importante dentro del sector industrial y ha contribuido significativamente en el incremento de la producción a nivel mundial. En Nicaragua, la robótica tiene poca presencia ya que son pocas las industrias que poseen robots industriales en sus cadenas de producción. En la educación superior en Nicaragua, pocas instituciones poseen laboratorios de robótica industrial adecuados para la enseñanza de tan importante campo de la ingeniería lo cual es debido, entre otras razones, a que la inversión requerida para adquirir los robots, garantizar la infraestructura, instalar y mantener los robots, es considerablemente alta. El personal capacitado para conducir el proceso enseñanza-aprendizaje de la robótica también es escaso en el país.

La falta de infraestructura para el desarrollo e investigación en el campo de la robótica no es un problema existente solo en Nicaragua, es común en la educación superior latinoamericana. Una alternativa para disminuir los efectos negativos de la carencia mencionada es el uso herramientas virtuales como apoyo para el proceso enseñanza-aprendizaje de la robótica.

En el presente documento se describen la estructura y las herramientas de software utilizadas para desarrollar un laboratorio virtual que pretende servir de apoyo en el proceso de enseñanza de los fundamentos de la robótica industrial, específicamente de los robots antropomórficos. El laboratorio fue desarrollado teniendo como elemento fundamental el Sistema Operativo para Robots (ROS, por sus siglas en inglés) el cual se ha convertido en un estándar de facto para los profesionales que se dedican al desarrollo de sistemas robóticos. ROS posee muchas herramientas de software y simuladores que permiten para el análisis de una aplicación robótica sea esta real o simulada.

Con el objetivo de dar a conocer ROS, debido a la importancia que tiene en el campo de la robótica, en el documento son presentados sus aspectos más relevantes tal como es la terminología, la estructura de una aplicación ROS, y los pasos para instalarlo en una PC.

En este informe se presentan las principales características del laboratorio virtual desarrollado y se presentan evidencia de su efectividad para la realización de actividades en lo concerniente a las herramientas matemáticas para el análisis de la posición y orientación del efecto final del robot, a la morfología de los robots, cinemática directa e inversa y programación de robots.

ABSTRACT

Nowadays, industrial robotics has an increasingly important presence within the industrial sector and has contributed significantly in increasing the production worldwide. In Nicaragua, robotics has little presence since there are few industries that own industrial robots in their production chains. When it comes to higher education in Nicaragua, a few institutions have suitable industrial robotics laboratories for teaching such an important field of engineering. This occurs because of, among other reasons, the high investment required to acquire the robots, guarantee the infrastructure, and install and maintain them. Additionally, the personnel trained to conduct the teaching-learning process of robotics is scarce in the country.

The lack of infrastructure for development and research in the field of robotics is not a problem that only exists in Nicaragua; it is also common in the Latin American higher education. An alternative to reduce the negative effects of the aforementioned need is the use of virtual tools to support the teaching-learning process of robotics.

This document describes the structure and software tools used to develop a virtual laboratory that aims to support the process of teaching the fundamentals of industrial robotics, specifically anthropomorphic robots. The laboratory was developed with the Robotic Operating System (ROS) as its fundamental element, which has become a de facto standard for professionals dedicated to the development of robotic systems. ROS has many software tools and simulators that allow the analysis of a robotic application, whether it is real or simulated.

In order to promote the use of ROS because of its importance to the field of robotics, the document presents its most relevant aspects, such as, terminology, the structure of a ROS application, and the steps to install it in a PC.

In this report, the main characteristics of the virtual laboratory developed are presented. In addition, the document discusses evidence of the effectiveness of carrying out activities regarding the mathematical tools for the analysis of the position and orientation of the final effector of the robot, the morphology of the robots, forward and inverse kinematics, and robot programming.

INDICE

INTRODUCCIÓN.....	1
OBJETIVOS	4
JUSTIFICACIÓN	5
CAPÍTULO I: MARCO TEÓRICO	7
1.1 Tipos de laboratorios	8
1.1.1 Laboratorio Físico.....	8
1.1.2 Laboratorio Virtual	8
1.1.3 Laboratorio Remoto online	8
1.1.4 Realidad virtual 3D	9
1.2 Sistema operativo para robot (ROS).....	11
1.2.1 Conceptos básicos de ROS	11
1.2.1.1 Nodos (Nodes)	12
1.2.1.2 Mensajes (Messages)	12
1.2.1.3 Tópico (Topic)	13
1.2.1.4 Maestro (master)	13
1.2.1.5 Servicios (services)	13
1.2.1.6 Bolsas (bags)	13
1.2.2 Librerías de ROS	15
1.2.3 Herramientas de visualización de datos en ROS	16
1.2.3.1 RViz.....	16
1.2.3.2 RQT.....	17
1.2.3.3 Movelt!.....	17
1.2.4 Lenguajes de programación soportados por ROS	18
1.3 Laboratorio virtual para robótica industrial	19
1.3.1 Interfaz gráfica de usuario (GUI)	20
1.3.2 Fundamentos de la robótica industrial.....	22
1.3.2.1 Morfología del robot.....	22
1.3.2.2 Herramientas matemáticas.....	22
1.3.2.3 Cinemática directa e inversa	23
1.3.2.4 Programación y simulación del robot.....	25
1.4 Uso del laboratorio virtual de robótica industrial	29
CAPÍTULO II: DISEÑO E IMPLEMENTACIÓN DEL LABORATORIO VIRTUAL DE ROBÓTICA INSDUSTRIAL	30
2.1 Recursos de software	30

2.1.1	Sistema operativo Ubuntu	30
2.1.2	ROS	31
2.1.2.1	Instalación y puesta en marcha de ROS	32
2.1.2.2	Estructura de ROS	34
2.1.2.3	Compilación de Paquetes de ROS.....	37
2.1.3	IDE QT-Creator.....	40
2.1.3.1	Plugin de ROS para Qt-Creator.....	41
2.1.3.2	Widgets QT	42
2.1.3.3	Widgets QML.....	44
2.1.4	Librería Orocó	44
2.1.5	Open Rave IkFast	45
2.1.6	Reuleaux	46
2.1.7	QCustomPlot.....	47
2.2	Diseño del LVR.....	48
2.2.1	Modelos de los robots	49
2.2.1.1	Modelos de los Fabricantes de robots.....	50
2.2.1.2	Modelos Creado por el usuario	50
2.2.1.2.1	URDF	50
2.2.1.3	Soporte de ROS – Industrial.....	52
2.2.1.3.1	Estructura de un paquete ROS-I	52
2.2.1.3.1.1	Archivos CAD de los robots industriales.....	53
2.2.2.1	Recursos Bibliográficos	54
2.2.2.2	Recursos Visuales.....	55
2.2.2.3	Ejecución de Procesos de ROS	55
2.2.3	Diseño de la Interfaz Modelación de Robots.....	56
2.2.4	Diseño de la Interfaz de Matemática y Cinemática de Robots	56
2.2.4.1	Matemática de los robots	58
2.2.4.2	Denavit - Hartenberg	60
2.2.4.3	Cinemática de Robots	61
2.2.5	Diseño del módulo de Programación de los Robots.....	63
2.2.5.1	Celda de trabajo con GAZEBO	63
2.2.5.2	Diseño de la interfaz interprete de comandos.	64
2.3	Implementación del LVR.....	66

2.3.1	Implementación de la interfaz principal	66
2.3.1.1	Recursos Bibliográficos	68
2.3.1.2	Recursos Visuales.....	69
2.3.1.3	Ejecución de Procesos de ROS	70
2.3.2	Implementación de la Interfaz Modelación de Robots.....	72
2.3.2.1	Creación de Interfaz en Qt.	72
2.3.2.2	Programación de los recursos de ROS	73
2.3.3	Implementación de la Interfaz de Matemática y Cinemática de Robots	
	77	
2.3.3.1	Creación de Interfaz en Qt.	77
2.3.3.2	Programación de los recursos de ROS	78
2.3.3.2.1	Visualizador 3D de robots virtuales	78
2.3.3.2.2	Selección de los robots dentro de la interfaz.....	81
2.3.3.2.3	Análisis de la Matemática del robot.....	82
2.3.3.2.4	Comprobación de Cinemática Directa	84
2.3.3.2.5	Comprobación de Cinemática Inversa.....	85
2.3.3.2.6	Algoritmo de Denavit Hartenberg	87
2.3.3.2.7	Gráficos de los movimientos de Joints.	88
2.3.4	Implementación del módulo de Programación de los Robots.	90
2.3.4.1	Implementación de la Celda de Trabajo con Gazebo.....	90
2.3.4.1.1	Plugin ros_control.....	91
2.3.4.2	Implementación de la Interfaz de Programación del Robot....	93
CAPÍTULO III: ANÁLISIS DE RESULTADOS		96
CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES		107
BIBLIOGRAFÍA.....		110
ANEXOS.....		114
A: Instalación de Recursos de Software para el Laboratorio Virtual.....		115
A.1 Instalación de UBUNTU 16.04 Xenial		115
A.2 Instalación de ROS Kinetic		117
A.3 Espacio de trabajo Catkin		118
A.4 Instalación de QT – Creator.....		119
A.5 Instalación del Plugin ROS para QT – Creator.		119
A.6 Instalación de Gazebo		121

Lista de Figuras

Capítulo I

Figura 1.1: Esquema de Funcionamiento de la Arquitectura de ROS	14
Figura 1.2: Comunicación utilizando Tópicos. En Topics por ROBOTIS Co., Ltd 2017.....	14
Figura 1.3: Nodo robot_localization recibiendo tópicos de un robot físico y mostrando su localización en el ambiente virtual. En robot_localization por Charles River Analytics Inc 2017	15
Figura 1.4: RVIZ y visualización de modelo de robot y datos provenientes de mensajes 3D	16
Figura 1.5: Gráfico de línea XY de datos provenientes de nodos.....	17
Figura 1.6: Modelo de robot, sistemas de referencia. En “Robótica” (P.6), por J. Craig, 2006.....	23
Figura 1.7: Cinemática directa e inversa En “Fundamentos de robótica” (P.119), por A. Barrientos et al, 2007	24
Figura 1.8: Modelo de utilización del LVR.	29

Capítulo II

Figura 2.1 Logo del sistema operativo para robots.....	31
Figura 2.2: Logo de la distribución de ROS, Kinetic Kame.....	32
Figura 2.3: roscore servidor maestro	34
Figura 2.4: Organización del software en ROS.	35
Figura 2.5: Organización de un paquete de software en ROS	36
Figura 2.6: Espacio de Trabajo Con Catkin.....	37
Figura 2.7: Archivo CMakeLists con parámetros para compilación	38
Figura 2.8: Archivo package.xml con etiquetas de referencia de paquetes.....	39
Figura 2.9: Logo de QT-Creator	40
Figura 2.10: Recursos de QT para el Desarrollador.....	40
Figura 2.11: Plugin CATKIN con QT-Creator Creación del Workspace de Trabajo.....	42
Figura 2.12: Widgets UI con conexión a Proceso ROS	42
Figura 2.13: Widgets Qt. En Qt Widget Gallery por QT Wiki 2019.	43
Figura 2.14 Código escrito en QML.....	44
Figura 2.15: Método de Newton-Raphson	45
Figura 2.16: Clase de la librería KDL para resolución de Problema Cinemático. En ChainIkSolverPos_NR_JL por orocos_kdl 2019.....	45
Figura 2.17: Uso del módulo IKFast para la Cinemática de robots.....	46

Figura 2.18: Proceso de ROS para gestionar la Visualización del espacio de trabajo	46
Figura 2.19: Espacios de Trabajo con Reuleaux. En Reuleaux: Robot base placement by reachability analysis por Makhal, A., et al	47
Figura 2.20: Modelo de laboratorio de robótica físico.....	48
Figura 2.21: Modelo del laboratorio Virtual de Robótica Industrial.....	49
Figura 2.22: Modelo de Robot obtenido utilizando el software Robot Studio de ABB.....	50
Figura 2.23: Representación de Links y Joints en un URDF.....	51
Figura 2.24: Estructura del lenguaje de marcado URDF	51
Figura 2.25 Robot 2 grados de libertad usando URDF.....	52
Figura 2.26: Paquete de Modelo de un Robot Kuka	52
Figura 2.27: Modelos de CAD en 3D	53
Figura 2.28: Diseño de interfaz principal por capa de software.....	54
Figura 2.29: Procesos a ejecutarse dentro de la interfaz principal	55
Figura 2.30: Diseño de la interfaz de Modelación de Robots.	56
Figura 2.31: Conceptos a desarrollar en interfaz de matemática y cinemática del robot.....	57
Figura 2.32 Proceso de Interacción con la interfaz de Matemática y cinemática.	57
Figura 2.33: Objeto puesto a rotación bajo coordenadas de orientación o Ángulos de Euler XYZ.....	59
Figura 2.34 Aplicación de Algoritmo Denavit - Hartenberg	60
Figura 2.35: Interacción de Gazebo con ROS e interfaz de usuario.....	63
Figura 2.36: ros_control y su interacción con el mundo de Gazebo. Tomado de: http://gazebosim.org/tutorials/?tut=ros_control	64
Figura 2.37: Diseño Interfaz Principal.....	66
Figura 2.38: Muestra de parte del código principal.....	67
Figura 2.39: Pantalla de Bienvenida al LVR	67
Figura 2.40: Muestra de la pantalla principal.....	68
Figura 2.41: Configuración de la librería PDF.Js en QML	69
Figura 2.42: Recurso Bibliográfico dentro del LVR.....	69
Figura 2.43: Propiedades del Módulo de Reproducción de Video.....	70
Figura 2.44: Contenido Multimedia en la interfaz del LVR.....	70
Figura 2.45: Clases en C++ y Componentes de QML	71
Figura 2.46: Código de la clase en C++ Registrada en QML	72
Figura 2.47: Tipos de Joints	72
Figura 2.48: Palabras reservadas dentro de la interfaz.....	73
Figura 2.49: Vista de RViz Por Default	74
Figura 2.50: Representación secuencial del modelo de software de Modelación de Robots.....	74
Figura 2.51: Nodos y Tópicos en la Interfaz interactuando bajo ROS	75
Figura 2.52: Interfaz Modelación de Robots	76

Figura 2.53: Ejemplo de la edición de los parámetros de un robot con un script URDF	76
Figura 2.54: Señales y Slots con funciones dentro del código de la interfaz principal	77
Figura 2.55: Herramientas del panel 3D de RViz	78
Figura 2.56: Herramientas de Visualización	79
Figura 2.57: Configuración del plugin Reuleaux	79
Figura 2.58: Reuleaux generando archivo para mostrar espacio de trabajo.	80
Figura 2.59: Espacio de Trabajo de un Robot Kuka.	80
Figura 2.60: Aplicación en C++ de la Matemática del Robot dentro del LVR	82
Figura 2.61: Posición del Modelo 3D	83
Figura 2.62: Rotación del Modelo en 3D del Avión.....	84
Figura 2.63: Obtención de los datos de Cinemática directa con KDL.....	85
Figura 2.64: Visualización del robot y su cinemática directa respectiva.	85
Figura 2.65: Obtención de los datos de Cinemática inversa con KDL.....	86
Figura 2.66: Visualización del robot y su cinemática inversa respectiva.	86
Figura 2.67: Programación en C++ con ingreso de datos de Denavit – Hartenberg	87
Figura 2.68: Modelos de Robot Generado con DH.....	88
Figura 2.69: Modelo de robot ABB IRB 120 con DH.....	88
Figura 2.70: Gráficos del movimiento del robot.	89
Figura 2.71: Nodos de ROS de la interfaz de Matemática y Cinemática.....	89
Figura 2.72: Modelo de la celda de Trabajo.	90
Figura 2.73: Archivo launch lanza los procesos para la carga de la celda de Trabajo.....	91
Figura 2.74: Servicios de configuración del PID de los Joints del robot.	92
Figura 2.75: Celda de Trabajo del robot.	93
Figura 2.76: Servicios de configuración de ros_control de Cada PID del Robot.94	
Figura 2.77: Robot ABB IRB120 simulado en Gazebo y controlado con Interfaz de Programación.....	94
Figura 2.78: Nodos de Gazebo e interfaz de Programación.....	95

Capítulo III

Figura 3.1: Robots con diferentes Morfologías en el LVR	97
Figura 3.2: Pose 1 del Robot ABB IRB2600 a) Robot Studio b) Visualizador Plugin RVIZ.....	98
Figura 3.3: Pose 2 del Robot ABB IRB2600 a) Robot Studio b) Visualizador Plugin RVIZ.....	99
Figura 3.4: Pose 1 del Robot ABB IRB120 a) Robot Studio b) Visualizador Plugin RVIZ.....	100
Figura 3.5: Pose 2 del Robot ABB IRB120 a) Robot Studio b) Visualizador Plugin RVIZ.....	100
Figura 3.6: Representación de dos tramas.....	101
Figura 3.7: Trama {B} Trasladada y rotada.....	101
Figura 3.8: Ejercicio de DH En “Fundamentos de robótica” (P.173), por A. Barrientos et al, 2007	103
Figura 3.9 Ingreso de los parámetros en Interfaz del LVR	104
Figura 3.10: Aplicación del Robot en Simulación y robot en físico.	106

Capítulo IV

Figura 4.1 ROS y Sistemas WEB	109
-------------------------------------	-----

ANEXOS

Anexo A

Figura A.1: IDE QT-Creator Activación del Plugin Catkin, para el espacio de Trabajo.....	120
Figura A.2: Ubicación del espacio de Trabajo.	121
Figura A.3: Espacio de trabajo Creado.....	121

Lista de Tablas

Capítulo I

Tabla 1.1: Mensajes ROS.....	12
Tabla 1.2: Comparación Técnica entre Simuladores.....	27

Capítulo II

Tabla 2.1 Comandos ROS.....	33
Tabla 2.2: Estructura de ROS.....	35
Tabla 2.3: Contenido de un Package	36
Tabla 2.4: Comandos Catkin.	37
Tabla 2.5 Conversión de coordenadas de posición y orientación.....	58
Tabla 2.6: Representación Matemática de Coordenadas de Posición y Orientación.....	59
Tabla 2.7: Parámetros de Denavit - Hartenberg	60
Tabla 2.8: Tabla de Paquetes de la interfaz de Matemática y Cinemática de Robots.....	62
Tabla 2.9: Paquetes utilizado para la programación y simulación del Robot en Gazebo	65
Tabla 2.10: Robots seleccionados dentro de la interfaz	81

Capítulo III

Tabla 3.1: Validación de Modelo ABB IRB2600	98
Tabla 3.2:Validación de Modelo ABB IRB120	99
Tabla 3.3: Ejercicio de representación de los sistemas de referencia	102
Tabla 3.4 Confirmación de algoritmo DH del robot del ejercicio 4	104

Anexos

Anexo A

Tabla A.1: Instalación de UBUNTU 16.04 Xenial	115
Tabla A.2: Instalación de ROS Kinetic.....	117
Tabla A.3 Creación del Espacio de Trabajo Catkin.	118
Tabla A.4: Instalación del IDE QT- Creator.	119
Tabla A.5: Instalación de Plugin ROS QT.....	119
Tabla A.6 Instalación de Gazebo.....	121

INTRODUCCIÓN

La robótica es uno de los campos de la tecnología que se desarrolla velozmente y encuentra cada día nuevas áreas de aplicación de tal forma que la clasificación de los robots debe ser ajustada constantemente. En la actualidad encontramos aplicaciones de la robótica en el campo personal y doméstico, servicios profesionales, investigación y desarrollo. Unos de los primeros campos donde los robots entraron en acción fue la industria en la cual destaca el uso de estos en el sector automotriz (soldadura y/o manejo de materiales). A pesar de que el uso de los robots en la industria data de la década de los 60s, en Nicaragua el uso de estos es relativamente bajo. El hecho mencionado tiene un impacto negativo en el desarrollo de la industria nicaragüense. Son pocas las industrias que poseen robots industriales y, de igual forma, son pocas las instituciones de educación superior que cuentan con la infraestructura adecuada, y el recurso humano, para la enseñanza e investigación de la robótica. Una de las razones es que un laboratorio físico, para experimentar en los diferentes aspectos asociados con los robots industriales, requiere de una inversión relativamente alta dado que es necesario garantizar el espacio físico, adquirir los robots y su mantenimiento.

La falta de infraestructura para el desarrollo e investigación de la robótica ha sido experimentada en la mayoría de las instituciones de educación superior latinoamericana muchas de las cuales han recurrido a la utilización laboratorios virtuales, diseñados e implementados por dichas instituciones o adquiridos en el mercado internacional. MATLAB, software ampliamente utilizado en el campo de las ingenierías, cuenta con un toolbox para robótica (Corke, 1996) mediante el cual se puede modelar y simular el comportamiento de un manipulador, sin embargo, el mismo tiene limitaciones. Simscape Multibody (antes llamado SimMechanics), de MATLAB y Simulink, permite la simulación de sistemas mecánicos en 3D, tales como robots. El software mencionado requiere de una licencia y el costo de esta aumenta en dependencia del número de usuarios. También se puede utilizar el software de programación de robots, versión

evaluación, de algunos fabricantes, tales como Robot Studio de ABB. Dicho software, además de limitaciones temporales, solo puede ser utilizado en los robots fabricados por la empresa. Para trabajar con diferentes robots, es necesario adquirir y aprender los lenguajes de programación de cada uno de ellos.

En la actualidad se busca estandarizar el trabajo con los robots y una de las herramientas disponibles para tal fin es el middleware “Sistema Operativo para Robot (ROS, por sus siglas en inglés)” el cual se ha convertido en el estándar de facto a nivel industrial tanto así que ya existe una versión denominada ROS Industrial (2017). ROS (s.f.) es un middleware, open-source, para el desarrollo a gran escala de complejos sistemas robóticos. Se trata de una colección de herramientas, librerías y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento complejo y robusto en una amplia variedad de plataformas robóticas.

Para dar a conocer ROS y mostrar la importancia de este, para el desarrollo de la robótica, se desarrolló un laboratorio virtual de robótica industrial (LVR) para el estudio de los fundamentos de la robótica industrial.

Para el desarrollo del laboratorio, además de ROS, se utilizaron herramientas de simulación tales como Gazebo, visualizadores como **RViz** y **rqt**. La integración de los diferentes nodos se logró utilizando el lenguaje de programación C++, para la modelación de los robots se utilizó el formato de descripción unificado de robots (URDF, por sus siglas en inglés), y para la creación de ambientes y escenarios fue empleado el formato de descripción de la simulación (SDF, por sus siglas en inglés).

El laboratorio virtual cuenta con los recursos necesarios para que los estudiantes estudien y realicen prácticas relacionadas a la morfología de los robots, herramientas matemáticas necesarias para el estudio de los robots, cinemática de los robots y programación de manipuladores industriales. Un elemento de mucha importancia en el LVR es el simulador que incorporará el cual permitirá manipular el modelo de un robot en un escenario particular, en respuesta a un programa

utilizando instrucciones parecidas a las que incorporan los lenguajes de programación de fabricantes tales como ABB.

El laboratorio virtual de robótica cuenta con un manual en el cual es presentada información de los recursos disponibles en el LVR, de forma tal que los docentes puedan diseñar prácticas que permitan a los estudiantes afianzar los fundamentos de la robótica industrial, e información general sobre ROS y herramientas asociadas, a un nivel tal que el usuario podrá implementar un ambiente de desarrollo básico. El profesor podrá elaborar prácticas de laboratorio relacionadas con los cuatro aspectos mencionados utilizando para ello los modelos de robots presentes en el LVR. Por ejemplo, el profesor podrá diseñar una práctica para que los estudiantes apliquen el procedimiento de Denavit y Hartenberg para determinar la matriz de transformación homogénea. En su versión inicial, el LVR cuenta con dos guías para realizar algunas actividades que permitan a los futuros usuarios apropiarse del procedimiento necesario para utilizar efectivamente los recursos del LVR.

En el presente documento son presentados los principales aspectos relacionados con el desarrollo del laboratorio virtual, así como los resultados obtenidos.

OBJETIVOS

Objetivo general

- Desarrollar un laboratorio virtual para el estudio de los fundamentos de la robótica industrial utilizando, para su implementación, el sistema operativo para robots (ROS).

Objetivos específicos

- Identificar las principales características/recursos/estructura de ROS enfatizando aquellas aplicables directamente a la robótica industrial.
- Identificar las herramientas de software requeridas, además de ROS, para el desarrollo del laboratorio virtual para el estudio de los fundamentos de la robótica industrial.
- Desarrollar la interfaz gráfica de usuario (GUI) del laboratorio virtual, según los requerimientos establecidos.
- Verificar la efectividad de los recursos del laboratorio virtual mediante la realización de dos prácticas de laboratorio.
- Elaborar un manual para el instructor donde se describa los recursos, y las características, del laboratorio virtual

JUSTIFICACIÓN

El campo de la robótica ha tenido, principalmente en las últimas décadas, un desarrollo vertiginoso y está impactando de forma positiva, principalmente en países desarrollados, en áreas como la medicina, la industria, y la investigación, entre otras. En Nicaragua, país en vías de desarrollo, el uso de la robótica es relativamente bajo, sin embargo, dicha tecnología tendrá mayor presencia en el futuro cercano y por consiguiente es necesario contar con profesionales con las competencias requeridas para realizar efectiva y eficientemente las diferentes tareas asociadas con la aplicación, el desarrollo y la investigación de la robótica.

La formación de profesionales en el campo de la robótica requiere de recursos humanos e infraestructura apropiada para tal fin (Jara, 2011a). En Nicaragua pocas instituciones de educación superior cuentan con recursos básicos mínimos para la formación de los estudiantes en el campo de la robótica (Universidad Tecnológica La Salle, 2017), siendo uno de los principales obstáculos para tener mejores condiciones la imposibilidad de garantizar la alta inversión requerida para la adquisición de los robots, software, accesorios asociados, instalación y mantenimiento de acuerdo con un estudio de Castellanos, & Martínez (2010). Compañías como FESTO ofrecen Kits denominados modulares, para prácticas de un proceso automatizado utilizando un robot Mitsubishi, que cuesta alrededor de 41,500 euros.

En el campo industrial, muchos fabricantes ofrecen manipuladores industriales los cuales tienen precios altos y, además, cada uno ofrece su propio software para la programación de los robots (Owen, 2016). Lo anterior significa que los estudiantes o especialistas en robótica industrial tendrían que aprender a utilizar el software correspondiente al manipulador en turno lo cual es imposible lograr si no se cuenta con la infraestructura adecuada. Se han realizado intentos para estandarizar los aspectos relacionados con la programación de los robots, hacerla independiente del fabricante, y uno de los resultados más destacado en la última década es el middleware “Sistema Operativo para Robot” (ROS, por sus siglas en inglés). En

la actualidad es considerado el estándar de facto para la aplicación, desarrollo e investigación de la robótica. Una muestra de la importancia de ROS es que en el campo industrial crearon el ROS-Industrial.

El desarrollo de un laboratorio virtual en robótica contribuiría en la mejora de las condiciones requeridas para el estudio de los fundamentos básicos de esta ya que brindaría a los estudiantes, y docentes, una herramienta que podría ser utilizada en cualquier lugar y en cualquier momento.

El laboratorio virtual sería desarrollado tomando como elemento principal el middleware ROS lo cual, entre otras cosas, permitiría a los estudiantes experimentar con modelos de robots de diferentes fabricantes tales como ABB, KUKA, FANUC, entre otros.

La inversión en el desarrollo e implementación del laboratorio virtual siempre será inferior a la requerida para tener un laboratorio físico ya que la mayoría de las herramientas utilizadas, tales como C++, ROS, GAZEBO, Qt creator son de código abierto (Open Source).

Los principales beneficiarios de los resultados de este proyecto son los docentes y los estudiantes. Los primeros dispondrán de una herramienta a partir de las cuales podrán diseñar experimentos, con su guía apropiada, relacionados con los fundamentos básicos de los robots industriales. Los estudiantes, por su parte, contarán con una herramienta flexible, que podrán utilizar en cualquier momento y en cualquier lugar, la cual les posibilitará verificar los fundamentos teóricos relacionados con los robots industriales abordados en programas de asignatura relacionados así programar los robots para que realicen tareas básicas.

El laboratorio virtual de robótica desarrollado debe ser considerado como el primer paso hacia un laboratorio virtual con mayores recursos en el cual se pueda experimentar con diversos tipos de robots, no solo los antropomórficos. Lo anterior contribuiría considerablemente en el estudio del funcionamiento de los robots y sus aplicaciones y presentaría un espacio para el desarrollo e investigación en el campo de la robótica.

CAPÍTULO I: MARCO TEÓRICO

Las instituciones académicas, principalmente las de educación superior (IES), buscan constantemente mejorar la calidad de la educación y una forma de hacerlo es integrando nuevas técnicas y herramientas orientadas a mejorar los resultados del proceso enseñanza-aprendizaje. En la actualidad, en muchas IES, a la tradicional clase presencial se suman otras formas de enseñanza-aprendizaje las cuales se fundamentan en las TIC, permitiendo de esta manera nuevas formas de enseñar, aprender, generar y compartir conocimiento.

Un recurso de mucho impacto en la formación de los futuros profesionales y cuya presencia crece día a día, en los procesos de enseñanza aprendizaje, es el laboratorio virtual (LV), el cual puede presentar altos niveles de flexibilidad y la inversión es muy inferior a la requerida para instalar un laboratorio real.

Uno de los campos en el cual el desarrollo y utilización de un laboratorio virtual reviste mucha importancia, dada la inversión alta requerida para tener un laboratorio físico, es el de la robótica en general y en particular, la robótica industrial.

En la introducción del presente documento se estableció que el trabajo de monografía desarrollado pretende dar a conocer el middleware ROS, su importancia en el desarrollo de la robótica industrial, y dar muestra de sus posibilidades mediante el diseño y construcción de un laboratorio virtual que sirva de base para el estudio de los fundamentos de los manipuladores industriales tipo serie.

En los siguientes apartados se presentan los elementos teóricos-tecnológicos relacionados con el desarrollo del laboratorio virtual.

1.1 Tipos de laboratorios

La experimentación de un fenómeno físico, o mediante la simulación, requiere de entornos de trabajos que reúnan los equipos o herramientas necesarias para el desarrollo efectivo de una práctica. En los ambientes académicos podemos encontrar diferentes tipos de laboratorios tales como los descritos a continuación.

1.1.1 Laboratorio Físico

Es ampliamente utilizado en universidades con modelos clásicos de enseñanza y son los espacios apropiados para desarrollar experimentos que requieren de la presencia física tanto del tutor como del alumno. Cabe destacar que la interacción directa con los equipos de un laboratorio físico aporta una experiencia difícil de igualar debido a que los alumnos perciben los resultados directamente lo cual pone en juego los cinco sentidos (vista, tacto, audición, olfato e incluso, a veces el gusto) (Calvo, 2009a).

1.1.2 Laboratorio Virtual

Es una alternativa al laboratorio físico y utiliza recursos computacionales haciendo uso de modelos matemáticos y recursos de visualización como modelos CAD y animaciones gráficas. Se pretende, mediante de este tipo de laboratorio, que el estudiante pueda estudiar el comportamiento de la realidad mediante el uso de modelos, sencillos o complejos, de la misma. Los recursos de un laboratorio virtual pueden ser ampliados en el tiempo permitiendo la realización de prácticas asociadas a temas de mayor complejidad.

1.1.3 Laboratorio Remoto online

Nacen bajo la necesidad de dar acceso a los estudiantes a un espacio de trabajo, de forma online, que combina los recursos de un laboratorio físico con recursos de software. El estudiante accede al laboratorio por medio de una página web a través de la cual puede controlar los recursos disponibles y llevar a cabo el experimento de interés. Su aprovechamiento está basado en la accesibilidad, por parte de un usuario, ya que los horarios pueden ser muy flexibles.

1.1.4 Realidad virtual 3D

Es la integración de recursos de hardware y software para acercar aún más al usuario a la experimentación de una teoría estimulando los sentidos tanto de la vista al exponerlos a entornos virtuales 3D y audición bajo altavoces con sonido envolvente.

En la actualidad se puede encontrar diferentes recursos para la enseñanza de la robótica Industrial tales como kits de robótica industrial, laboratorios virtuales de robótica de categoría libre o de pago para ejecutar cierta práctica. El desarrollo de un laboratorio remoto requiere contar con acceso a un laboratorio físico y enlazarlo a un servidor para que el estudiante pueda acceder vía remota garantizándole una plataforma amigable y multiplataforma como pueden ser los navegadores WEB. Candelas, F. et al. (2004) analizan las ventajas del uso de un laboratorio remoto ROBOLAB, proyecto desarrollado por ellos mismos bajo el nombre de grupo de investigación AUROVA. El proyecto utiliza herramientas gráficas para modelado y visualización de objetos 3D de un brazo robótico que coincide con el mismo modelo de un robot físico, permitiendo al estudiante realizar sus prácticas al observar y configurar el modelo virtual y una vez alcanzado un nivel de aprendizaje aceptable, proceder a interactuar con el robot físico utilizando la misma plataforma web. Ellos concluyen que al utilizar el laboratorio Remoto ROBOLAB “*La mayoría de los alumnos prefieren disponer de un laboratorio en la universidad dónde trabajar con la ayuda de los compañeros y el apoyo didáctico del profesor, pero también hay muchos alumnos que reciben con agrado la opción de un laboratorio virtual que les ofrezca unos horarios flexibles en los que realizar los experimentos*”. (Candelas, et al, 2004b).

En resumen, un laboratorio virtual es un elemento importante en el proceso enseñanza-aprendizaje ya que destacan, entre otras, las siguientes ventajas según Calvo, et al (2009b):

1. El estudiante se familiariza con el experimento evitando acudir al aula sin conocimiento previo.

2. Comparación del comportamiento de modelos matemáticos ante una simulación permitiendo extraer sus propias conclusiones de cierta práctica.
3. Manejo de herramientas informáticas contemporáneas para la formación integral de un estudiante.
4. Repetitividad de los experimentos realizados por el estudiante que permitirá reproducir cuantas veces desee hasta consolidar el conocimiento.
5. Disminución de riesgos y accidentes que pueden ocasionar una mala práctica o configuración de un equipo físico.
6. Multiplicidad de experimentos simultáneos realizados ya que cada estudiante podrá ejecutar la práctica indicada en su computador asignado, además de esta forma se favorecen los procesos colaborativos como el de “Lluvia de ideas” al opinar cada alumno sobre su percepción adquirida al ejecutar la práctica.

Considerando lo anterior, se decidió, para dar a conocer la importancia y posibilidades de ROS, desarrollar un laboratorio virtual el cual tendrá a ROS como elemento integrador. Es importante destacar la inversión relativamente baja requerida para implementar el laboratorio virtual, la posibilidad de ampliación de este y la posibilidad que el mismo ofrecerá a los docentes para diseñar nuevas prácticas a partir de los recursos disponibles.

1.2 Sistema operativo para robot (ROS)

El sistema operativo para robot (ROS, por sus siglas en inglés), elemento fundamental del presente trabajo, es un middleware ampliamente utilizado en el mundo creciente de la robótica. Originalmente fue desarrollado en el 2007 en los laboratorios de Inteligencia Artificial de Stanford y en el 2008, cedieron el derecho de desarrollo al instituto de investigación de robótica Willow Garage (2010), donde la filosofía es proporcionar bibliotecas y herramientas libres para ayudar a los desarrolladores de software a crear aplicaciones para robots

En este trabajo monográfico, se promueve la importancia de ROS en el campo de la robótica y se hace mediante el diseño y construcción, utilizando los recursos proporcionados por este, de un Laboratorio Virtual de Robótica, en el cual será posible realizar experimentos sobre los fundamentos de la robótica industrial específicamente de los manipuladores industriales tipo serie.

Una característica importante de ROS es su código abierto (Open Source) lo que ha llevado al desarrollo, por muchos colaboradores a nivel mundial, de una amplia colección de herramientas, librerías y convenciones. Los recursos que ofrece ROS permiten el desarrollo, a gran escala, de sistemas robóticos complejos, tanto físicos como simulados.

Los desarrolladores de ROS han establecido un conjunto de convenciones y buenas prácticas en el desarrollo de software de forma tal que se fomente la reutilización de código para robots, trabajando sobre una arquitectura robótica totalmente sólida y funcional. De esta forma se garantiza, entre otras cosas, que el proyecto no sea fallido y que sus resultados satisfagan la calidad requerida en este tipo de trabajos.

1.2.1 Conceptos básicos de ROS

ROS fue diseñado bajo una estructura distribuida y modular con el propósito de que los usuarios puedan usar los recursos requeridos según la aplicación a desarrollar. Es decir, el usuario puede seleccionar los recursos de software necesarios para implementar la solución.

En su operación, ROS presenta una red de **procesos** que se ejecutan simultáneamente en una estructura **peer-to-peer** donde cada nodo o elemento del sistema puede actuar al mismo tiempo como cliente y como servidor. Para una aplicación distribuida de ROS se requiere que los nodos peer to peer estén sincronizados bajo un middleware el cual genera un orden temporal para los eventos generados en el sistema, además de utilizar la comunicación XML/RPC que define la información de registro de cada nodo al **ROS-Master**. Una vez registrados los nodos se procede a ejecutar la comunicación peer to peer entre los nodos utilizando el protocolo **TCP/IP**. Entre los muchos conceptos básicos relacionados con la comunicación destacan los siguientes:

1.2.1.1 Nodos (Nodes)

Son procesos que realizan cálculos. Por ejemplo, en un sistema robótico, un nodo realiza la planificación de la trayectoria y otro puede suministrar una vista gráfica del sistema.

1.2.1.2 Mensajes (Messages)

Los nodos se comunican entre ellos enviándose mensajes. Un mensaje es simplemente una estructura de datos, que contiene campos de un solo tipo de datos. Tipos primitivos de datos (enteros, booleanos, punto flotante, etc.) son soportados, así como lo son arreglos de tipos primitivos.

Tabla 1.1: Mensajes ROS

Grupo de Mensajes	Descripción del Tipo de mensaje
actionlib_msgs	Mensajes para interacción con un servidor Acción y cliente Acción
geometry_msgs	Mensajes Para manejo de datos geométricos con puntos, vectores y Estados o poses de orientación por coordenadas.
sensor_msgs	Mensajes usados para conectar Sensores, incluyendo cámaras y laser para scanner PCL.
trajectory_msgs	Mensajes para definir la trayectoria de los robots, ya sea cinemática, dinámica o Posición y orientación en un espacio.

1.2.1.3 Tópico (Topic)

El tópico es un nombre usado para identificar el contenido de un mensaje. Un nodo envía un mensaje publicándolo en un tópico dado. Los mensajes son enrutados vía un sistema de transporte con una semántica de publicación/suscripción. Los tópicos son nombres utilizados para identificar el contenido de un mensaje. Un nodo que está interesado en cierta clase de datos debe suscribirse al tópico apropiado. (Ver Figura 1.2)

1.2.1.4 Maestro (master)

El máster proporciona los elementos necesarios para que los nodos se encuentren los unos a los otros, intercambien mensajes o invoquen servicios.

1.2.1.5 Servicios (services)

Es un modelo de comunicación cliente-servidor, es una manera en el que los nodos se pueden comunicar. Estos permiten que se envíen solicitudes y se reciban respuestas

1.2.1.6 Bolsas (bags)

Formato para guardar y reproducir datos de un mensaje proveniente de un nodo de ROS. Son un importante mecanismo para almacenar datos, tales como datos de sensores, que son difíciles de obtener pero que son necesarios para el desarrollo y prueba de algoritmos.

La arquitectura básica de comunicación de ROS, usando 3 nodos como ejemplo simplificado es mostrada en la Figura 1.1. Para establecer comunicación con diferentes nodos es necesario la utilización de registros, cuyos nombres son suministrados por el maestro.

Como se muestra en la Figura 1.1 el Nodo 1, el Nodo 2 y el Nodo 3 se registran con el Máster usando comunicación XML/RPC el cual, luego de conocer cada nodo registrado, permite a los diferentes nodos intercambiar la información entre ellos utilizando para esto el protocolo TCP/IP.

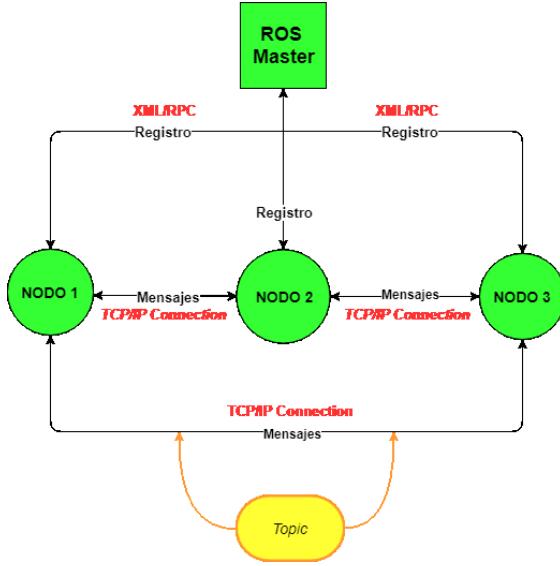


Figura 1.1: Esquema de Funcionamiento de la Arquitectura de ROS

Dado que cada nodo puede compartir múltiples mensajes, a estos se les clasifica como tópicos lo cual permite la centralización de información proveniente de varios procesos de un sistema robótico ya sea este físico o simulado. En la Figura 1.2 podemos observar el comportamiento básico de un nodo y sus tópicos.

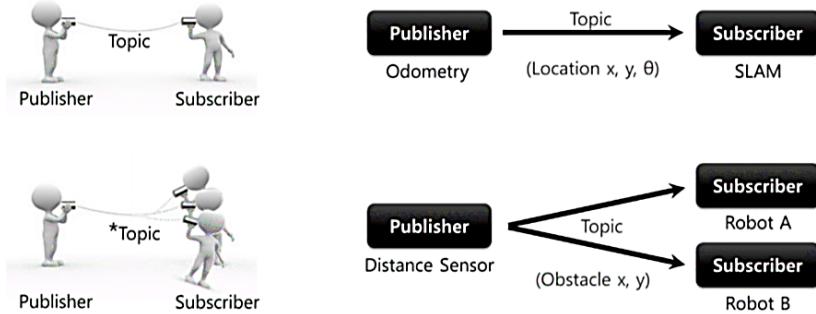


Figura 1.2: Comunicación utilizando Tópicos. En Topics por ROBOTIS Co., Ltd 2017

Un ejemplo del funcionamiento de la arquitectura de ROS es mostrado en la Figura 1.3 en la cual se identifica el **nodo** "robot_localization." El nodo mencionado recibe mensajes desde varios nodos bajo los **tópicos**: IMU, Camera, GPS, Detección 3D (Sensing 3D) y odometria (odometry). Los datos suministrados en los **mensajes** son procesados por el nodo y como resultado comparte bajo un único tópico, y varios mensajes, al visualizador 3D RViz las coordenadas de localización del objeto físico para que este pueda ser representado de manera virtual.

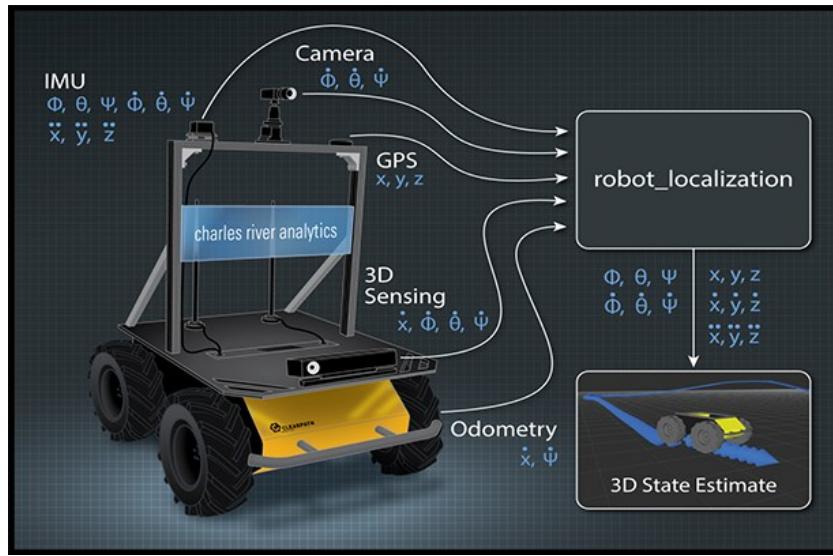


Figura 1.3: Nodo `robot_localization` recibiendo tópicos de un robot físico y mostrando su localización en el ambiente virtual. En `robot_localization` por Charles River Analytics Inc 2017.

Un nodo podemos considerarlo como un ejecutable dentro del paquete de ROS y utiliza la librería cliente, lista de nodos disponibles de ROS, para comunicarse con otros nodos, y a estos a su vez se les configura para publicar o suscribirse a un tópico, o algunos procesos de ROS tales como servicios o almacenamiento de datos (Bags).

1.2.2 Librerías de ROS

El sistema operativo para robots (ROS) presenta una serie de características que llaman la atención de las personas dedicadas a la robótica principalmente a la hora de diseñar e implementar sistemas basados en robots. Una de las características principales es que puede acceder a una amplia colección de librerías las cuales varían de acuerdo con la distribución de ROS (turtle, kinetic, etc.) con la que se deseé trabajar. Las librerías de las distribuciones más recientes cuentan con librerías con códigos más estables e incorporan nuevas librerías probadas y recomendadas por la comunidad ROS.

Para implementar un nodo en ROS, se pueden utilizar diferentes librerías tales como **roscpp** para C++ o **rospy** para Phyton.

ROS dispone de las librerías necesarias para la implementación de las actividades que deben ser desarrolladas en un sistema robótico tales como el intercambio de mensajes entre nodos o la estructuración de datos para descripción de trayectorias.

1.2.3 Herramientas de visualización de datos en ROS

Un elemento importante en cualquier laboratorio virtual de robótica es la posibilidad de visualizar el comportamiento del modelo de un robot en un escenario dado. También es fundamental que el mismo cuente con los recursos necesarios para facilitar el análisis de los datos generados por el sistema robótico bajo estudio. ROS dispone de una serie de herramientas para la visualización de los mensajes de los sistemas robóticos ya sean estos físicos o virtuales. Entre el sinnúmero de herramientas disponibles destacamos **RViz**, **rqt**, y **MoveIt**, nodos en el sistema de suscripción de ROS con sus tópicos y servicios para permitir la entrada y salida de los datos a procesar. Las herramientas mencionadas fueron utilizadas en el desarrollo del laboratorio virtual propuesto.

1.2.3.1 RViz

Es una herramienta de visualización 3D que permite combinar en una misma pantalla modelos de robots, datos de sensores (cámara, láser, etc.) y otros datos provenientes de mensajes con información de coordenadas 3D. RViz es utilizado en el desarrollo del laboratorio virtual principalmente para visualizar la morfología y estudiar la cinemática de los robots. La Figura 1.4 muestra una imagen generada utilizando RViz.

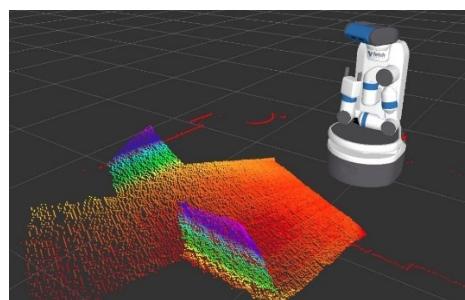


Figura 1.4: RVIZ y visualización de modelo de robot y datos provenientes de mensajes con coordenadas 3D.

1.2.3.2 RQT

La herramienta rqt ofrece varios plugins, entre los cuales destacan ‘**rqt_image_view**’, ‘**rqt_graph**’, ‘**rqt_plot**’ y ‘**rqt_bag**’, los cuales pueden ser utilizados para la introspección y visualización de datos provenientes de procesos de ROS. Por ejemplo, si es necesario visualizar los mensajes provenientes de una cámara en un sistema robótico, podríamos utilizar el plugin rqt-image_view. rqt expone los nodos, y las conexiones entre ellos, lo que permite entender la estructura del sistema, su funcionamiento, así como su fácil depuración. En la Figura 1.5 se muestra el resultado de utilizar rqt_plot para visualizar los datos provenientes de un nodo.

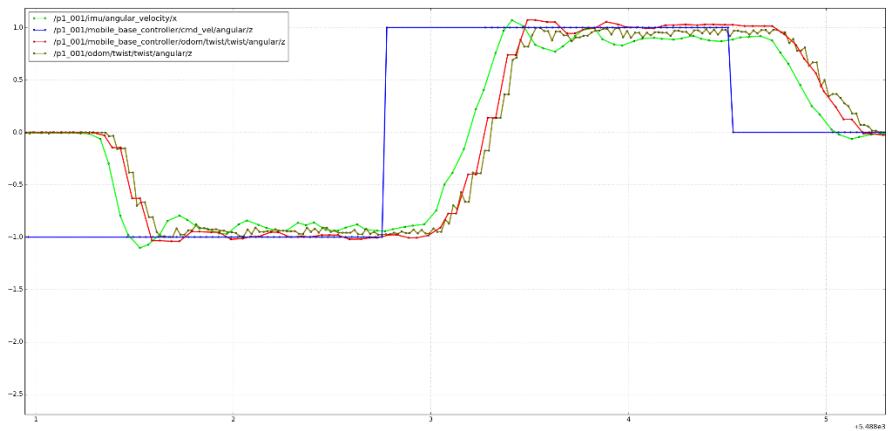


Figura 1.5: Gráfico de línea XY de datos provenientes de nodos

En el laboratorio virtual de robótica, rqt fue utilizado para graficar la estructura de datos correspondiente a la posición del robot durante su movimiento.

1.2.3.3 MoveIt!

Es una herramienta de software, escrita en C++, que permite la planeación de trayectorias para un robot industrial, así como la visualización de los datos y del modelo 3D del mismo, valiéndose de un plugin que es incorporado a la aplicación de visualización de datos RViz. Actualmente, MoveIt (s. f.) es utilizado en diversas industrias para hacer la planeación de las trayectorias para diferentes tipos de manipuladores industriales, adoptando la visión de ROS de facilitar la programación de diferentes tipos de robots con los mismos recursos de Software.

El sistema desarrollado se le puede suministrar mayor funcionalidad al utilizar ROS ya que el mismo permite la integración y utiliza la información generada por aplicaciones externas de código abierto muy populares tales como Gazebo y OpenCV. Por ejemplo, es posible utilizar OpenCV para hacer reconocimiento de imágenes y tomar decisiones a partir de los datos suministrados. Más adelante en el documento se presentarán las características de Gazebo dado que es el simulador que se incorporó en el LVR en las prácticas de programación del robot.

1.2.4 Lenguajes de programación soportados por ROS

En el desarrollo del software para una aplicación de robótica, ROS permite el uso de distintos lenguajes de programación. De manera oficial soporta **Python**, **C++** y **LISP**. **Java** podría ser soportado en el futuro, pero en la actualidad se encuentran en una fase experimental, apoyada por Google. De igual forma **C#** se encuentra en una fase experimental utilizando UNITY 3D, el cual es un motor de video juegos multiplataforma que permite simular robots importando el **URDF** como un GameObject en UNITY3D y obtener de este una simulación más realista con el renderizado de UNITY3D.

Con ROS es posible utilizar nodos creados en diferentes lenguajes de programación tales como Phyton y C++ y garantizar el intercambio de información entre estos.

En el desarrollo de la GUI y el intercambio de información entre los nodos ROS se utilizó el lenguaje C++ y el compilador **CMake** el cual fue diseñado para construir, probar y empaquetar software. CMake es el compilador oficial soportado por la comunidad ROS y el mismo desplazó al anterior compilador llamado **rosbuild**.

El ambiente integrado de desarrollo **Qt-Creator** posee un plugin para ROS (ROS Industrial, 2016) que cuenta con los recursos necesarios (por ejemplo, templates) para crear nodos, servicios, y archivos URDF, entre otros. Qt-Creator permite al desarrollador construir los recursos de software necesarios, desde el IDE, para un sistema robótico.

1.3 Laboratorio virtual para robótica industrial

En una institución de educación superior, que dispone de un laboratorio físico para la enseñanza de la robótica, el robot es parte de un escenario que simula un ambiente industrial y las prácticas generalmente están relacionadas con la programación del robot para que realice una tarea determinada. El sistema no presenta la flexibilidad necesaria para que los estudiantes mejoren su comprensión acerca de los fundamentos de la robótica industrial. De hecho, aprenden a programar el robot utilizando el lenguaje específicamente diseñado para este.

El laboratorio virtual para robótica desarrollado incorpora los recursos necesarios para que los estudiantes estudien o realicen experimentos relacionados con temas tales como la morfología del robot, la cinemática del robot, la programación de robots de diferentes fabricantes independientemente del lenguaje específico desarrollado para cada uno. Se incorporaron recursos que permitirán al estudiante entender las herramientas matemáticas utilizadas en el campo de la robótica industrial, específicamente manipuladores industriales tipo serie.

En el desarrollo de software es común el uso de los términos **Back-end** y **Front-end** los cuales aportan una perspectiva de la clasificación de tipo de Software a utilizar en la construcción del propio laboratorio virtual. ROS fue utilizado para de manejo de datos en back-end y el uso de librerías de software libre para desarrollar la GUI del lado de front-end.

En el LVR el estudiante, para la realización de los experimentos, puede utilizar modelos de robots de fabricantes tales como ABB, FANUC o KUKA. De igual forma, cuenta con las herramientas y la guía para desarrollar sus modelos propios.

Los principales elementos del laboratorio virtual son:

- Interfaz gráfica de usuario (GUI) para interacción, programación, visualización, etc.
- Modelos de robots industriales de ABB, KUKA, y Cartesianos.
- Escenarios para la realización de tarea específicas

- Documentación sobre aspectos fundamentales sobre robótica industrial
- Programación de Robots bajo el entorno de simulación de Gazebo.
- Gráficos de los movimientos de los Joints del Robot.

La interacción entre los diferentes elementos del laboratorio virtual es garantizada mediante el middleware ROS.

A continuación, son presentados algunos de los conocimientos o recursos tecnológicos requeridos para el desarrollo del LVR, específicamente para la implementación de la interfaz gráfica de usuario (GUI) y la construcción de algunos recursos para el estudio de los fundamentos de los manipuladores industriales.

1.3.1 Interfaz gráfica de usuario (GUI)

Una interfaz gráfica de usuario (GUI, por sus siglas en inglés) facilita la explotación de una aplicación mediante la incorporación de elementos destinados para tal fin. La GUI debe comportarse de una manera comprensible y predecible, de modo que un usuario sepa qué esperar cuando realiza una acción. La interfaz gráfica de usuario del laboratorio virtual cuenta con los elementos necesarios (botones, sliders, editor de texto, pantalla de visualización, etc.) para realizar las prácticas sobre los fundamentos de la robótica industrial, así como para acceder a documentos de interés tales como libros, guías de laboratorio, entre otros. La GUI está diseñada de tal forma que el estudiante podrá usar los recursos de manera transparente sin necesidad de tener que configurar aspectos básicos de ROS tales como su inicialización y el lanzamiento de nodos.

En la actualidad existen muchos lenguajes que pueden ser utilizados para desarrollar una interfaz de usuario y entre ellos destacan C++, C#, Java, Phyton, Visual Basic. Un criterio para utilizar un lenguaje en particular es que el mismo sea multiplataforma ya que la interfaz podría ser utilizada bajo diferentes sistemas operativos.

Hoy día están disponibles un sinnúmero de ambientes integrados de desarrollo (IDE, por sus siglas en inglés) que facilitan la programación ya que aglomeran todas las herramientas necesarias tales como conjuntos de librerías, preprocesador, depurador, y compiladores. Entre los IDEs más conocidos destacan, entre otros, Visual Studio, Eclipse, NetBeans, CLion, y Qt Creator. Para la construcción de la GUI del laboratorio virtual de robótica industrial se utilizó el IDE Qt Creator, debido, principalmente, a que este posee un plugin de soporte para ROS.

Qt Creator fue utilizado para crear las aplicaciones o nodos de ROS y se exploraron las capacidades que posee QT Creator para la creación de aplicaciones de escritorio. Qt Creator cuenta con **QML**, un estándar propio del IDE, que permite crear aplicaciones dinámicas y más elaboradas capaces de incorporar recursos WEB. Esa línea de creación se utilizó para incorporar un lector de PDF con el objetivo que los estudiantes dispongan en el LVR de documentación de interés tales como libros o guías de laboratorio.

En el desarrollo de la GUI del LVR se utilizaron los widgets de Qt Creator (slider, botones, etc.) los cuales están escritos en C++. Esta línea de creación es muy importante para lograr la visualización en el LVR ya que es muy fácil incorporar las librerías de ROS, escritas en C++, a una aplicación utilizando widgets. Un ejemplo es el desarrollo de una aplicación de escritorio personalizada que se acople a los requerimientos del LVR al incorporar las librerías de RViz para visualizar los modelos de los robots.

Es importante mencionar que en el laboratorio virtual se realizan múltiples procesos en el back-end, con el objetivo de garantizar la transparencia en el uso de los sus recursos por los usuarios, que iniciarán la ejecución de las interacciones en la interfaz de usuario tales como la visualización de los robots y datos, programación y simulaciones de los robots. Dichos procesos están basados en nodos que son lanzados o eliminados de acuerdo con la lógica implementada en el propio LVR.

1.3.2 Fundamentos de la robótica industrial.

El laboratorio virtual de robótica desarrollado tiene un alcance limitado ya que de los temas considerados en un curso básico de robótica industrial el mismo solo contempla recursos para la experimentación en temas como herramientas matemáticas, morfología del robot, cinemática y programación de robots. El trabajo pretende mostrar la potencialidad del sistema operativo para robots (ROS) y lo hace poniendo en marcha el diseño, la programación y el uso del LVR. La información suministrada acerca de ROS y sobre el laboratorio desarrollado deben servir como base para ampliar los recursos de este.

1.3.2.1 Morfología del robot

El laboratorio virtual de robótica cuenta con modelos creados utilizando **URDF** y los mismos podrán ser visualizados utilizando la herramienta RViz. El estudiante puede visualizar el espacio de trabajo, para una morfología dada, con la ayuda de plugins desarrollados para la visualización. Interactuar con diferentes tipos de articulaciones ya sean prismáticas o de revolución y robots de diferentes grados de libertad. El estudiante tendrá la posibilidad de experimentar con modelos construidos por él mismo utilizando el formato URDF.

RViz es una abreviación para ROS visualización y es una herramienta poderosa para la visualización en 3D. Permite al usuario ver el modelo simulado del robot, obtener información de los sensores del robot, y reproducir la información obtenida de los sensores. Mediante la visualización de lo que ve y hace el robot el usuario puede eliminar los errores de la aplicación robótica desde entradas de sensores hasta acciones planificadas o no planificadas.

1.3.2.2 Herramientas matemáticas

El modelado cinemático de un robot busca las relaciones entre las variables articulares y la posición (expresada normalmente en forma de coordenadas cartesianas) y orientación del extremo del robot (expresada como matrices de rotación, ángulos de Euler o algún otro de los métodos establecidos para tal fin).

Entender las herramientas matemáticas utilizadas para determinar la localización espacial de los diferentes elementos de un robot industrial, especialmente del efecto final, presenta cierta dificultad a los estudiantes, principalmente cuando se utiliza solamente la pizarra para su explicación. Para contribuir al entendimiento y aplicación de dichas herramientas el laboratorio virtual de robótica incorpora los recursos necesarios para que los estudiantes puedan analizar la posición de un objeto utilizando coordenadas esféricas y cilíndricas, así como la orientación de este mediante la matriz de rotación, los ángulos de Euler, y los cuaternios. Las operaciones requeridas para el desempeño de los recursos estarán soportadas por la librería **TF2**, disponible para C++ y Phyton, propia de la distribución de ROS, que permite hacer conversiones de sistemas de coordenadas.

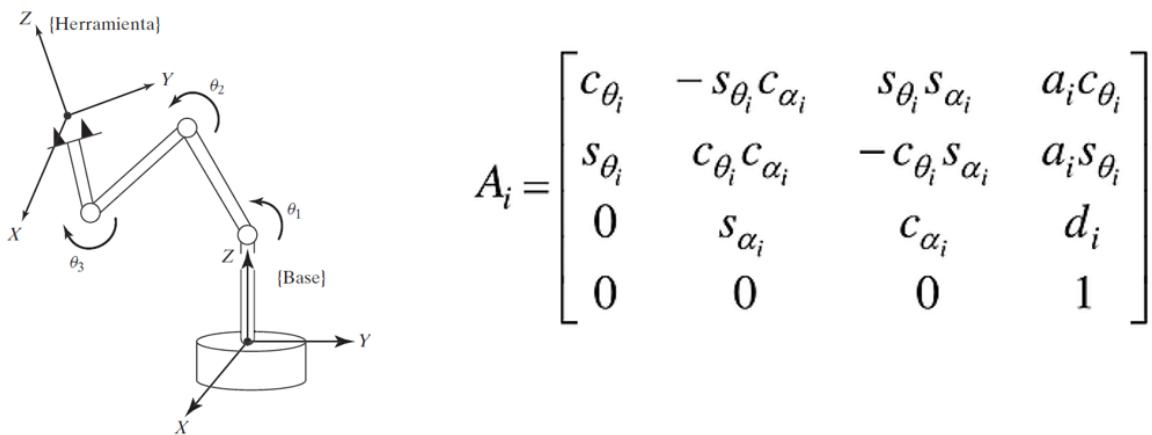


Figura 1.6: Modelo de robot, sistemas de referencia. En “Robótica” (P.6), por J. J. Craig, 2006

1.3.2.3 Cinemática directa e inversa

La cinemática del robot estudia el movimiento de este con respecto a un sistema de referencia sin considerar las fuerzas que intervienen. Así, la cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo, y en particular por las relaciones entre la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares. Dentro de la cinemática se estudian la posición, velocidad, aceleración

y todas las derivadas de mayor orden de las variables de posición (respecto al tiempo o a cualquier otra variable).

En muchas ocasiones es necesario conocer en cuál posición se encontrará el efecto final del robot dadas ciertas posiciones de las articulaciones y dicho problema es abordado por la cinemática directa. En otras ocasiones es necesario conocer cuál debe ser la posición de las articulaciones para que el efecto final sea ubicado en una posición deseada. El último problema es abordado por la cinemática inversa. La Figura 1.7 muestra la relación entre las variables para cada uno de los tipos de cinemática.



Figura 1.7: Cinemática directa e inversa En “Fundamentos de robótica” (P.119), por A. Barrientos et al, 2007

El laboratorio virtual de robótica desarrollado cuenta con recursos que permitirán al estudiante realizar experimentos sencillos que le ayudarán a comprender las características de la cinemática directa e inversa.

A partir del modelo de un robot, construido utilizando el formato URDF, es posible obtener toda la información necesaria de la estructura de este utilizando la librería **KDL**. La información obtenida al utilizar el KDL en combinación con la información suministrada por el usuario (por ejemplo, los ángulos de los joints) mediante ciertos widgets de la GUI permiten manipular y/o determinar la posición del robot. La librería KDL también puede ser utilizada para determinar la matriz de transformación homogénea la cual es utilizada para determinar la posición y rotación del efecto final del robot. Los datos obtenidos serán visualizados mediante widgets del GUI desarrollados para la representación de datos.

1.3.2.4 Programación y simulación del robot

En la robótica, en general, uno de los más aspectos de mayor importancia es el relacionado con la programación de los robots. Durante la ejecución de un programa se interacciona con la memoria del sistema, leyendo y actualizando el contenido de las variables utilizadas en el programa; con el sistema de control cinemático y dinámico del robot, encargados de dar la señal de mando a los accionamientos del robot a partir de las especificaciones del movimiento que se les proporciona; y con las entradas-salidas del sistema, consiguiéndose así la sincronización del robot con el resto de las máquinas y elementos que componen su entorno.

El sistema de programación es, por tanto, la herramienta con que cuenta el usuario para acceder a las diversas prestaciones del robot, existiendo una relación directa entre las características y posibilidades del sistema de programación y las del robot en sí mismo.

La programación puede ser **guiada** o **textual**. Es importante destacar que en la actualidad es muy frecuente que los sistemas de programación de robots tiendan a combinar los dos modos básicos (guiado y textual), permitiéndose desarrollar el programa mediante la escritura de las instrucciones, y utilizando la posibilidad de guiado en línea en aquellos momentos en que sea necesario. Sistemas como RAPID de ABB, VAL II de Staübli y V+ de Adept Technology son ejemplos de esta dualidad.

Uno de los problemas relacionados con la programación de los robots es que no existe un estándar y el programador debe aprender el lenguaje específico para los robots de diferentes fabricantes. ROS brinda la posibilidad de programar robot de diferentes fabricantes desde el mismo entorno.

Supongamos que es necesario que dos robots de diferentes fabricantes, pero la misma morfología, deben seguir la misma trayectoria. En el ambiente de ROS, es posible escribir mensajes de ROS (que contiene la posición, velocidad, aceleración, etc., de cada articulación del robot), escritos en C++ o Python. El driver de cada robot, una vez recibido el mensaje, generará los comandos para que su robot siga la trayectoria indicada.

El laboratorio virtual de robótica permite al usuario escribir un programa (script, programación textual) que adoptará ciertas convenciones, utilizadas en el software de programación de robots de ABB Rapid, para que uno de los robots (modelos disponibles en el LVR) realice una tarea sencilla.

Una forma mediante la cual podemos verificar si el robot realiza la tarea indicada correctamente es la simulación. En la actualidad existen muchos simuladores para robots, sencillos y complejos. También podemos clasificarlos en comerciales y de código libre. (Ver Tabla 1.2)

Entre los comerciales destaca **V-Rep** el cual es un simulador dinámico con un entorno de desarrollo integrado que permite desde su interfaz lenguajes de programación tales como C, C++, Phyton, entre otros.

V-Rep es un simulador con entorno de desarrollo integrado donde cada objeto/modelo se puede controlar individualmente a través de un script de programación que puede ser escrito usando lenguajes de programación como C/C++, Python, Java, Lua, Matlab u Octave. V-REP se utiliza para el desarrollo rápido de algoritmos, simulaciones de automatización de fábricas, prototipado rápido y verificación y educación relacionada con la robótica.

De los simuladores open-source uno de los más utilizados es **Gazebo**, el cual provee simulaciones dinámicas donde los robots pueden interactuar con el entorno (pueden coger, empujar, rodar, deslizarse por el suelo), capacidad de manipular las características del entorno tal como la gravedad del mundo virtual.

Tabla 1.2: Comparación Técnica entre Simuladores.

Simulador	Lenguaje de Programación	Formatos Soportados	Extensión de Software	Middleware	GUI
Gazebo	C++	SDF/URDF	Plugins C++	ROS, Player.	Si
V-Rep	LUA	Object, STL, Collada, URDF	API, Addons, Plugins	Sockets, ROS	Si
Webots	C++	WBT	Plugins C++, API	ROS, URBI, NaoQI	Si

Gazebo es simulador dinámico 3D que tiene la habilidad para simular poblaciones de robots, de forma exacta y eficiente, en ambientes complejos internos y externos. Aunque similar a máquinas de juego, Gazebo ofrece una simulación física a un grado de fidelidad mayor, un conjunto de sensores, e interfaces tanto para programas como para los usuarios. Usos típicos de Gazebo incluyen:

- Prueba de algoritmos robóticos
- Diseño de robots
- Realización de pruebas de regresión con escenarios realísticos.

Algunas características claves de Gazebo:

- Máquinas físicas múltiples
- Una librería de modelos de robots y ambientes.
- Una amplia variedad de sensores

Aunque ambos simuladores tienen la posibilidad de comunicarse utilizando ROS, se seleccionó Gazebo como el simulador a incorporar en el LVR debido a que este es open-source. El estudiante podrá observar en el simulador Gazebo el

comportamiento del modelo del robot en respuesta al programa (script) escrito por el usuario. El script incorporará el conjunto de instrucciones las cuales serán procesadas por un nodo el cual enviará a Gazebo, utilizando el plugging de conexión correspondiente, los mensajes apropiados para lograr el movimiento de las articulaciones del robot en el mundo.

SDF es un formato XML utilizado para describir objetos y ambientes para simuladores de robots, visualización, y control. SDF tiene capacidad para describir todos los aspectos de un robot, objetos estáticos y dinámicos, iluminación, terreno y también la física. En el laboratorio virtual el lenguaje SDF se utiliza para modelar los escenarios de trabajo en el ambiente de Gazebo.

Podrán ser utilizados los siguientes parámetros:

- ✓ **Escena:** Iluminación ambiental, propiedades del cielo, sombras.
- ✓ **Física:** Gravedad, paso del tiempo, motor de la física.
- ✓ **Modelos:** Colección de enlaces, objetos de colisión, articulaciones y sensores.
- ✓ **Luces:** Punto, espacio y fuentes de luz direccionales.
- ✓ **Plugins:** plugins soportado por gazebo, plugins del mundo, del modelo, del sensor y del sistema.

1.4 Uso del laboratorio virtual de robótica industrial

Los diferentes elementos involucrados en el uso, y posible ampliación del laboratorio virtual, son mostrados en la Figura 1.8. El laboratorio virtual incluye un manual de usuario cuyo objetivo es presentar al usuario los recursos disponibles y un par de guías de laboratorio para mostrar cómo utilizarlos. Como se aprecia en la Figura 1.8, el docente podrá, partiendo de los temas relacionados con los fundamentos de la robótica industrial que desee desarrollar y de los recursos disponibles en el laboratorio, elaborar las guías que utilizarán los estudiantes para realizar las actividades de estudio o las prácticas de laboratorio correspondientes.

Se propone, al margen del trabajo monográfico, desarrollar un taller, para los docentes interesados, con el objetivo de brindarles los detalles de diseño y herramientas utilizadas en la implementación del LVR para que tengan una base para la ampliación de los recursos de este.

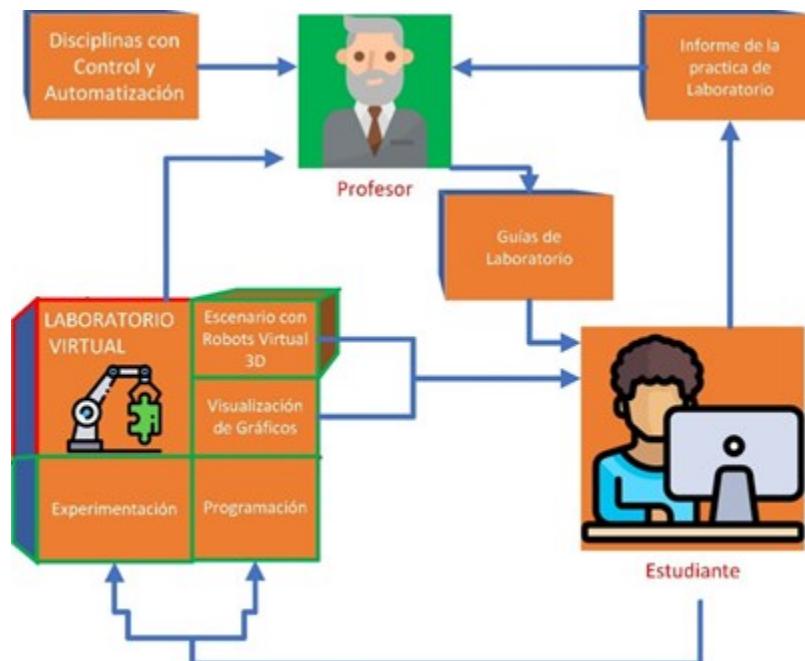


Figura 1.8: Modelo de utilización del LVR.

CAPÍTULO II: DISEÑO E IMPLEMENTACIÓN DEL LABORATORIO VIRTUAL DE ROBÓTICA INSNDUSTRIAL

En este capítulo son presentados los aspectos técnicos relacionados con el diseño e implementación del software desarrollado, API y plugins utilizados para el desarrollo del Laboratorio virtual de Robótica, cumpliendo los requerimientos planteados en la sección 1.4 para la utilización del LVR.

2.1 Recursos de software

A continuación, se explicará el proceso para adquirir los recursos de software necesarios para lograr el desarrollo del laboratorio virtual, se detallará la importancia de estos, enfatizando en aspectos particulares que son necesarios dominar para trabajar con el sistema operativo Ubuntu, ROS, el IDE y librerías de importancia para este trabajo monográfico.

2.1.1 Sistema operativo Ubuntu

El sistema operativo sobre el que se desarrolló este trabajo monográfico es una distribución de Linux, precisamente la distribución **Ubuntu** 16.04 LTS (Long Terminal Support), es libre y con compatibilidad con la versión más estable de ROS, la distribución Kinetic LTS. Cabe destacar que esta versión se mantiene con actualizaciones hasta el 2020, actualmente está disponible la versión estable ROS Melodic. No hace falta migrar a la versión Melodic debido a que todos los recursos de software que serán necesarios para este trabajo están bajo la versión instalada Kinetic (Ver anexo A1 Instalación de Ubuntu).

Debido a que el sistema operativo Linux no es ampliamente utilizado, representa una fragilidad, esto porque actualmente el sistema operativo dominante en las computadoras personales es Windows. Es bien conocido que para acceder a los recursos de software en las plataformas Linux es necesario tener un cierto manejo y conocimiento de líneas de comandos (Shell) en Linux para movernos por el entorno, esto representa una falta de popularidad comparada con Windows. Es importante dar a conocer que esto no representará un problema en el futuro

cercano en los trabajos desarrollados con ROS ya que Microsoft anuncio la colaboración con ROS en adoptar la compatibilidad del Middleware ROS a Windows (Microsoft 2018). El resultado de este trabajo monográfico posee la capacidad de ser actualizado a una versión en Windows.

Par el desarrollo de este trabajo fue de vital importancia conocer el entorno Linux para entender cómo se realizan las compilaciones, ejecutar comandos con debida destreza por consola para manipular parámetros de manera rápida y como mostrar resultados de procesos por consola.

2.1.2 ROS



Figura 2.1 Logo del sistema operativo para robots

Como se mencionó en la sección 1.2, ROS se define como un **middleware** para el desarrollo software de robot. La curva de aprendizaje que toma aprender ROS es algo sinuosa al principio, pero algo que diferencia a ROS de otros middleware como **Orocos**, **playerStage**, etc., es que pone a nuestra disposición multitud de recursos que brinda la propia comunidad, de hecho ser parte de la comunidad complementa la filosofía que dio origen a ROS (Ver sección 2.2) de no reinventar la rueda al compartir ideas de cómo hacer robótica bajo el estándar de trabajo basado en un ecosistema ROS con más de 3,000 paquetes de software, una infraestructura de comunicación modular basada en nodos, tópicos, o mensajes de ROS. En su página cuenta con foros o la wiki de la propia página. En dicha Wiki nos muestran los pasos necesarios para comenzar en ROS en la sección Getting Started (Ros Wiki 2018).

2.1.2.1 Instalación y puesta en marcha de ROS

Dentro del desarrollo de aplicaciones o recursos de software solventados o mantenidos por una comunidad es muy importante definir la versión con la cual se desea trabajar ya que estos recursos de software (package) son actualizados o se agregan nuevas funcionalidades constantemente, si el trabajo investigativo lleva una extensión de tiempo considerable y se desea aprovechar todas las funcionalidades de cierta versión, se debe elegir una distribución LTS. En el desarrollo del LVR, desde el inicio se instaló la distribución Kinetic y se aprovecharon muchos de los recursos bajo esta distribución. Ver Anexo 1.2 Instalación de ROS.



Figura 2.2: Logo de la distribución de ROS, Kinetic Kame.

Al instalar todos los paquetes de ROS Kinetic con el comando

```
$ sudo apt-get install ros-Kinetic-desktop-full
```

Comando BASH

y otras herramientas como visualizadores de datos como, RQT, RViz y el simulador Gazebo obtenemos los recursos de software necesarios para el desarrollo del Laboratorio Virtual de Robótica.

Es necesario conocer los comandos Shell para el desarrollo con ROS, los comandos más comunes son.

Tabla 2. 1 Comandos ROS

Comando	Descripción
roscore	Es necesario ejecutar este comando ya que pone en marcha para que todos los demás funcionen. Es el encargado de gestionar toda la comunicación entre los nodos.
roscd:	Permite cambiar de paquete, directorio o stack. Sin necesidad que sean colindantes de forma jerárquica.
rostopic	[list] Muestra la lista de topics en ejecución
	[echo] Muestra información sobre el topic
	[type] Muestra el tipo del topic
	[pub] Publica un mensaje en un topic dado
rosnode	[list] Muestra la lista de nodos en ejecución
	[info] Muestra información sobre el nodo
	[ping] Comprueba la conexión con el nodo
	[kill] Mata el nodo indicado
rosrun	Ejecuta un nodo. \$ rosrun [PACKAGE_NAME] [NODO_NAME]
rosmsg:	Muestra información sobre los mensajes ROS
roslaunch	Permite lanzar varios nodos y parámetros simultáneamente \$ ros launch [PACKAGE_NAME] [/launch_FILE_NAME]

Si ejecutamos el comando

\$ roscore

Comando BASH

Podemos comprobar el estado de la instalación de ROS y por ende proseguir con el desarrollo de software.

```
roscore http://yesser-dell-system-inspiron-n411z:11311/
yesser@yesser-dell-system-inspiron-n411z:~$ roscore
... logging to /home/yesser/.ros/log/9de7fc22-1498-11e9-a202-b5ee938d8173/roslau
nch-yesser-dell-system-inspiron-n411z-14673.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
WARNING: disk usage in log directory [/home/yesser/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://yesser-dell-system-inspiron-n411z:34304/
ros_comm version 1.12.14

SUMMARY
=====
PARAMETERS
  * /rosdistro: kinetic
  * /rosversion: 1.12.14
NODES

auto-starting new master
process[master]: started with pid [14683]
ROS_MASTER_URI=http://yesser-dell-system-inspiron-n411z:11311/
setting /run_id to 9de7fc22-1498-11e9-a202-b5ee938d8173
process[rosout-1]: started with pid [14696]
started core service [/rosout]
```

Figura 2.3: roscore servidor maestro

En la Figura 2.3 muestra información que indica el excelente estado de la instalación con el servidor maestro.

2.1.2.2 Estructura de ROS

Como se especificó en la sección 1.2.1 “Conceptos Básicos de ROS”, los nodos en ROS se comunican entre sí mediante tópicos intercambiando mensajes los cuales son administrados y enrutados a sus nodos correspondientes por el ROSMaster. Podemos ver a estos nodos como ejecutables dentro de paquetes de software ya sea escritos en C++ o python. Para entender la estructura de ROS podemos ver la Figura 2.4 con su respectiva Tabla 2.2 que describe la clasificación de los elementos.

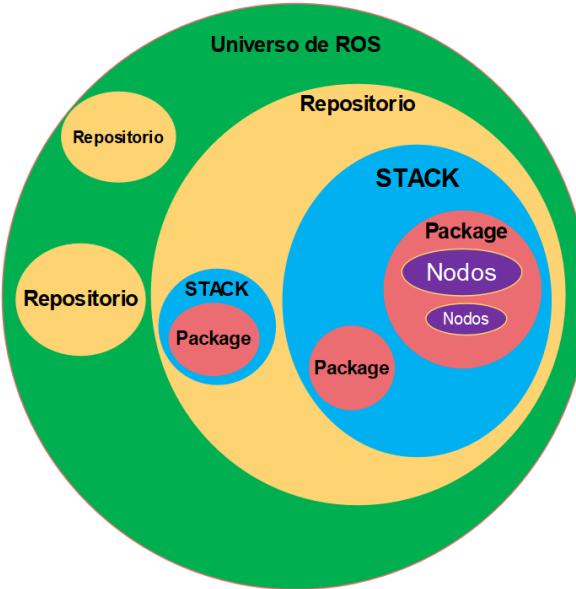


Figura 2.4: Organización del software en ROS.

Tabla 2.2: Estructura de ROS

Clasificación	Descripción
Repository	Es en general un proyecto que puede tener diversos paquetes de software que interactúan intercambiando información bajo los nodos de ROS. El laboratorio Virtual de Robótica es considerado un Repository ya que en él van diferentes recursos de software como Widgets de Qt-Creator, QML con Qt Creator, librerías Boost, Librerías RViz y librerías ROS.
Stack	Las pilas o stacks contienen ciertos paquetes relacionados entre sí
Packages	Los paquetes contienen dentro la definición de los mensajes que utilizan los nodos asociados a estos, por ejemplo, un paquete de un stack puede enviar información procesada de la cinemática de un robot vía mensajes a un nodo de otro paquete en otro Stack que no cuenta con este procesamiento y por ende los recursos de software.

Para entender mejor la estructura de un paquete y su relación con un Stack podemos ver la estructura de este en la Figura 2.5, relacionando directamente a el tipo de recurso que posee entre los diferentes paquetes.

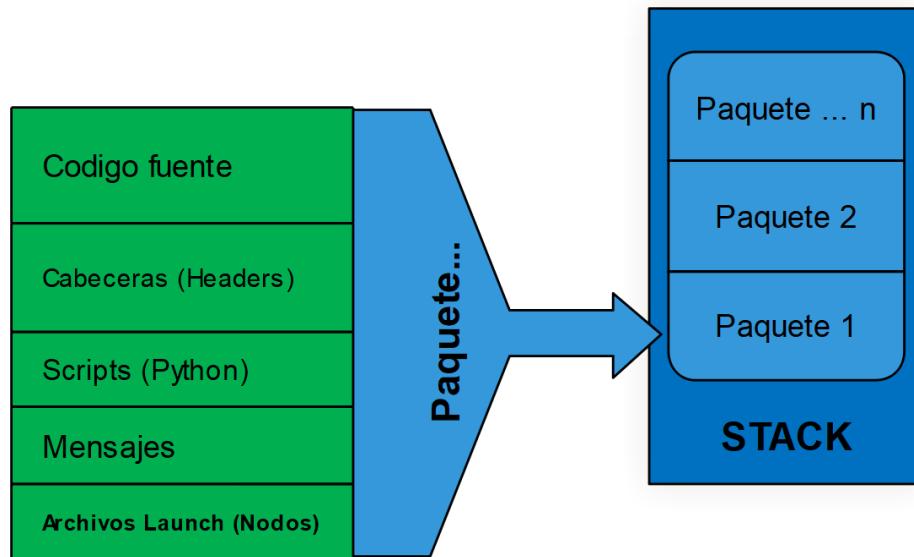


Figura 2.5: Organización de un paquete de software en ROS

Dentro de los paquetes encontramos los siguientes elementos que son fundamentales entender para el desarrollo de cualquier aplicación con ROS.

Tabla 2.3: Contenido de un Package

Elementos	Descripción
Package.xml	Este archivo de lenguaje descriptivo contiene todos los paquetes de los que depende el mismo paquete en cuestión. Se definen aquellos que van a ser construidos y los que van a ejecutarse bajo las etiquetas <code><build_depend></code> y <code><run_depend></code>
CMakeFile	Este archivo se encarga de compilar el paquete. Deben incluirse todas las dependencias y códigos que se hayan desarrollado si existiesen. Estos códigos tendrán librerías específicas que hacen referencia a paquetes de ROS por ello es necesario definir cada uno de estos paquetes para que el compilador lo tenga en cuenta
Carpetas	Normalmente se designa las siguientes carpetas: <i>src</i> , <i>launch</i> , <i>bag</i> . <i>src</i> se guarda el código que se desea compilar y ejecutar como nodo. <i>launch</i> se guardan los lanzadores de nodos.

Para crear un paquete utilizamos las instrucciones de línea de comando del compilador **Catkin** y para esto debemos crear un espacio de trabajo para el compilador (Ver Anexo A.3 Espacio de trabajo Catkin) donde estarán almacenados todos los paquetes que vayamos creando. Es necesario conocer 3 comandos importantes al compilar con Catkin.

Tabla 2 4: Comandos Catkin.

Comandos	Descripción
catkin_init_workspace	Inicializa el espacio de trabajo creado
catkin_create_pkg	Crea un paquete con la dependencia de una librería u otro paquete de ROS. [PACKAGE_NAME] [DEPENDENCY_PACKAGE1]
catkin_make	Compila todos los paquetes dentro del Espacio de Trabajo.

En la Figura 2.6 se muestra la estructura de carpetas que conforman nuestro espacio de trabajo con Catkin.

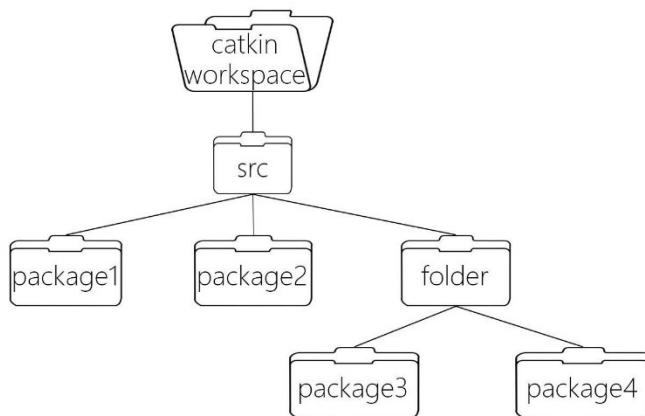


Figura 2.6: Espacio de Trabajo Con Catkin.

2.1.2.3 Compilación de Paquetes de ROS.

Todo paquete debe contener dos archivos, los cuales son de importancia para el compilador, **CMakefile.txt** y **package.xml**. En los archivos mencionados se

definen los recursos de software de un paquete para que este sea compilado correctamente. Los archivos CMakefile y package.xml poseen varios parámetros de configuración a tener en cuenta, pero en nuestro caso, tendremos en cuenta los siguientes parámetros.

find_package

CMakeLists.txt

Este parámetro del archivo CMakeLists permite agregar los paquetes necesarios para la compilación.

Indicamos el mismo paquete en las dependencias de catkin.

catkin_package

CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8.3)
2  project(my_first_node_lab_UNI)
3
4  find_package(catkin REQUIRED COMPONENTS
5    roscpp
6  )
7
8  catkin_package(
9    # INCLUDE_DIRS include
10   # LIBRARIES myworkcell_core
11   CATKIN_DEPENDS
12    roscpp
13   # DEPENDS system_lib
14 )
```

Version de Catkin

Nombre del paquete

Dependencia del paquete o project

roscpp
dependencia para creacion de nodos

Dependencia del paquete o project

Figura 2.7: Archivo CMakeLists con parámetros para compilación

Es necesario configurar correctamente este archivo debido a que la compilación de un paquete a ser usado bajo ROS depende de lo indicado en el archivo CMakeLists. En la Figura 2.7 se muestra parte de los atributos necesarios a ser configurados.

El archivo package.xml es un complemento de CMakeLists.txt donde también se agregan las dependencias solo que en formato con etiquetas en un archivo .XML. Dentro de estos parámetros tenemos los más destacados a analizar a continuación.

Todas estas etiquetas en el archivo de marcado son atributos importantes para hacerle referencia al compilador de cuáles recursos serán ocupados en el paquete y además como serán utilizados. Por ejemplo, el <build_depend> se utiliza para hacer el llamado de las librerías a utilizar en el paquete a compilar.

```
<build_depend>roscpp</build_depend>
```

package.xml

En cambio el <run_depend>, del mismo paquete, se utiliza para incluir los vínculos de los archivos objetos a ejecutar dentro del paquete con los comandos de **rosrun** (Ver Tabla 2. 1 Comandos ROS).

```
<run_depend>roscpp</run_depend>
```

package.xml

```

1  <?xml version="1.0"?>
2  <package>
3      <name>my_first_node_lab_UNI</name>
4      <version>0.0.0</version>
5      <description>The my_first_node_lab_UNI package</description>
6      <maintainer email="myalfredo03@ieee.org">Yeser A. Morales</maintainer>
7      <license>GPLv2</license>
8
9      <buildtool_depend>catkin</buildtool_depend>
10     <build_depend>roscpp</build_depend>
11     <run_depend>roscpp</run_depend>
12
13     <!-- The export tag contains other, unspecified, tags -->
14     <export>
15         <!-- Other tools can request additional information be placed here -->
16
17     </export>
18 </package>

```

Version de .XML	Linea 1
Nombre del paquete	Linea 3
Información del paquete (Autor) (Licencia)	Lineas 5-7
Compilador del paquete	Linea 9
Dependencia del paquete a compilar	Linea 10
Dependencia del paquete a Ejecutar	Lineas 11-12

Figura 2.8: Archivo package.xml con etiquetas de referencia de paquetes.

Al crear un paquete siguiendo las indicaciones plasmadas en el anexo A.3 Espacio de trabajo Catkin”, se garantiza que podamos crear la lógica de programación dentro de un archivo en C++, crear un nodo, hacer traspasos de mensajes vía tópicos usando los recursos de ROS o en todo caso hacer uso de Widgets para la creación de interfaces de usuario, donde las interfaces a crearse para el laboratorio se pueden definir como un paquete con recursos de ROS que puede mostrar información al usuario.

2.1.3 IDE QT-Creator.

En la Sección 1.3.1 se mencionan las tecnologías más conocidas para el diseño de una interfaz de usuario. QT-Creator posee varias ventajas que fueron útiles en el desarrollo del trabajo monográfico destacando su característica multiplataforma, además, anteriormente en la Sección 2.1.1 se dieron a conocer los detalles del porque se usó el sistema operativo Ubuntu y la posibilidad de migrar este trabajo monográfico a un sistema operativo de uso común como lo es Windows. Al usar QT - Creator junto con la futura versión de ROS para Windows (Windows 10 + ROS, 2019) es posible la migración gracias a su característica multiplataforma.



Figura 2.9: Logo de QT-Creator

Qt utiliza el lenguaje de programación C++ como principal herramienta de desarrollo, pero adicionalmente puede ser utilizado en otros lenguajes de programación a través de “bindings” o plugins como son conocidos normalmente. Los plugins más conocidos son **PyQt** para el desarrollo de interfaz en Python, **QtRuby**, **PHP-QT**, entre otras.

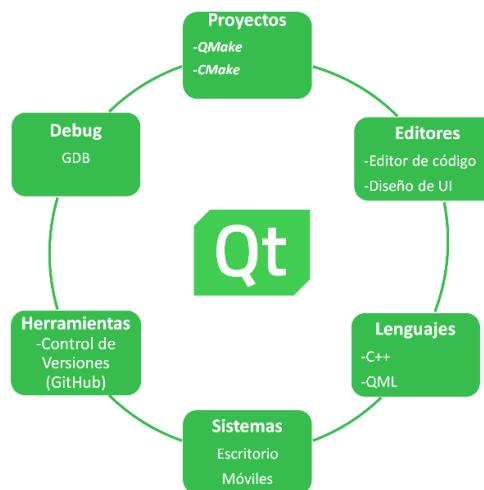


Figura 2.10: Recursos de QT para el Desarrollador.

En los recursos mostrados en la Figura 2.10, al usar el IDE Qt, podemos compilar con CMake (Ver sección 2.2.2.3 Compilación de Paquetes de ROS.), usar lenguajes como C++ para trabajar con ROS y usar el lenguaje QML para el desarrollo de interfaces. El IDE Qt adopta todos los recursos de software necesarios para un desarrollo ágil y centralizado.

QT provee de recursos a otros lenguajes de programación y posee la capacidad de adoptar muy fácilmente varios recursos que poseen otros lenguajes dentro del propio IDE. Por ejemplo, diseñar una interfaz de usuario (GUI) con capacidad para cargar una página WEB o vincular recursos de base datos. Dichos recursos son utilizados en el laboratorio virtual desarrollado.

Tres ejes principales de Qt-Creator se aprovecharon para el desarrollo del laboratorio virtual los cuales son en primer lugar el uso del plugin de ROS para el IDE, en segundo lugar, el uso de Widgets para aplicaciones de interfaz de usuario nativas en C++ y por último el uso de Widgets QML para aplicaciones de interfaz de usuario donde el lenguaje QML presta ciertas bondades parecidas a usar CCS al diseñar páginas Web. Se aporta dinamismo y mayores prestaciones visuales al desarrollar una aplicación con QML.

2.1.3.1 Plugin de ROS para Qt-Creator

La versión de Qt-Creator usada en el trabajo monográfico es la **5.7**, compatible con el **Plugin de ROS** desarrollado por el Consorcio de ROS – Industrial. Este plugin permite la agilización de creación de paquete de ROS, mensajes, servicios, acciones, así como la incorporación de la estructura básica de programación en C++ de un nodo.

Al instalar el IDE (Ver Anexo A.4) se puede proceder a la instalación y configuración del plugin para el IDE QT (Ver Anexo A.5). De la misma forma en que se agiliza la creación de un paquete también es posible compilar y hacer la búsqueda de errores del paquete de ROS dentro del IDE, ver Figura 2.11.

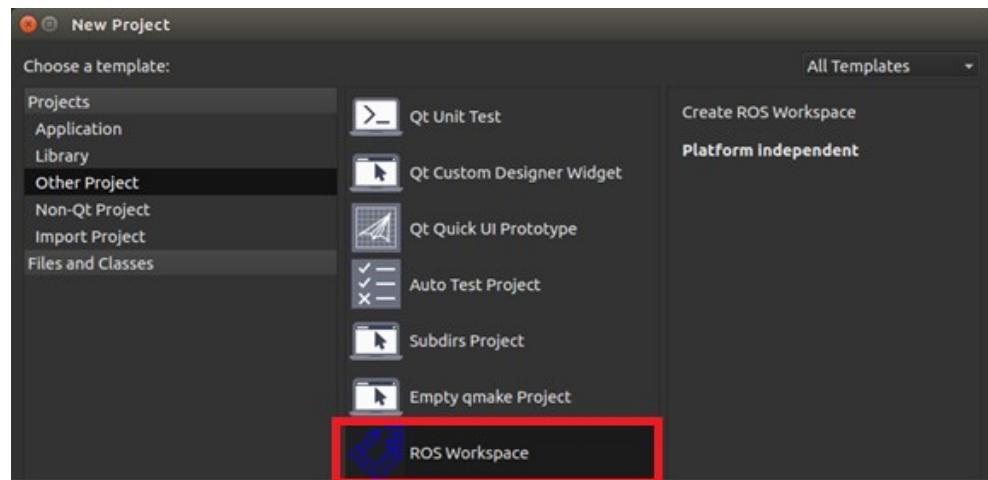


Figura 2.11: Plugin CATKIN con QT-Creator Creación del Workspace de Trabajo.

2.1.3.2 Widgets QT

Widgets y formas creadas con **Qt Designer** generan un formato de archivo .UI el cual se integra a la perfección con código programado bajo ROS. Mediante el uso de las señales de Qt todo esto bajo el lenguaje C++.

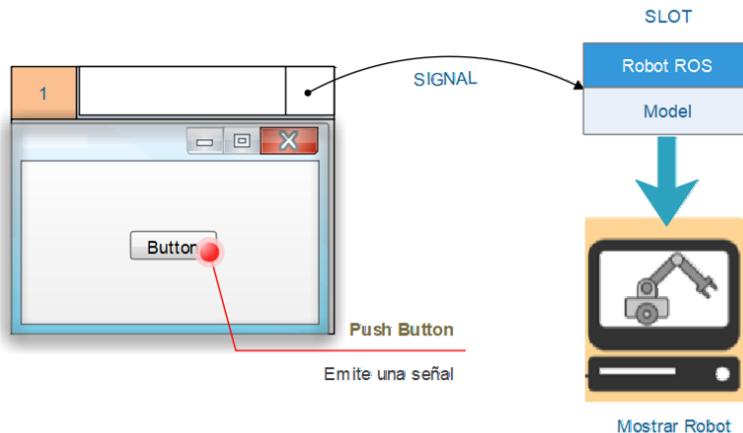


Figura 2.12: Widgets UI con conexión a Proceso ROS

En la Figura 2.12 se muestra un botón el cual es un widget dentro de Qt, que al ser accionado (presionado) emite una señal que es referenciada a una función dentro del código en C++ el cual ejecuta un proceso. En el ejemplo de la figura al presionar el botón se ejecuta el proceso de mostrar el modelo del robot a simular. Cabe destacar que Qt posee los widgets comunes que cobran importancia para

el manejo de datos para toda interfaz de usuario, como slider, pushbuttons, checkbox, tablas y otros. La Figura 2.13 se muestra una recopilación de los widgets más comunes en una interfaz de usuario y entre ellos destacan los siguientes.

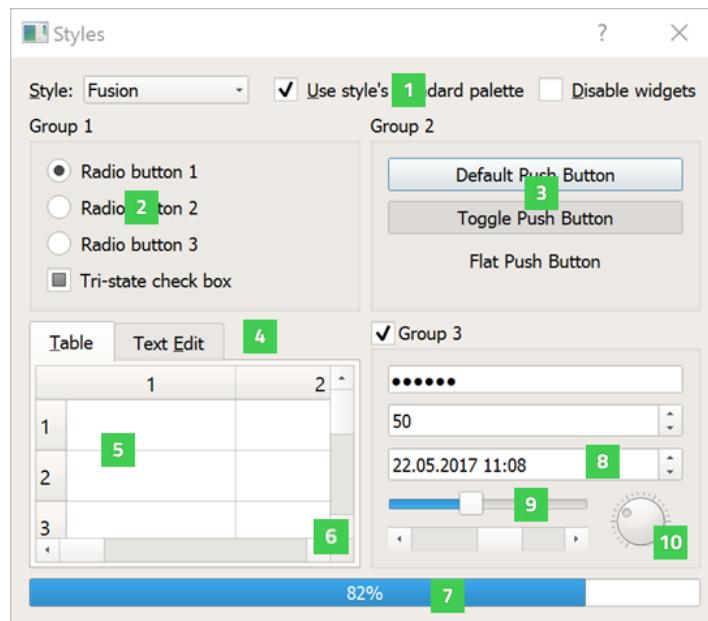


Figura 2.13: Widgets Qt. En Qt Widget Gallery por QT Wiki 2019.

- [1] QCheckBox, provee un CheckBox con una etiqueta de texto.
- [2] QRadioButton Provee un Radio button Con una etiqueta de Texto.
- [3] QPushButton Provee un botón con etiqueta.
- [4] QTabWidget Provee un Espacio con widgets apilados.
- [5] QTableWidget Provee una table de datos.
- [6] QScrollBar Provee una barra vertical y horizontal deslizante.
- [7] QProgressBar Provee una barra horizontal progresiva.
- [8] QDateTimeEdit Provee un Widget Para edición de fechas y horas.
- [9] QSlider Provee un Slider horizontal.
- [10] QDial, Provee un control Redondo tipo potenciómetro.

2.1.3.3 Widgets QML

La programación en QML describe un árbol de elementos (tipo HTML), estos elementos son los widgets los cuales poseen animaciones más fluidas y agradables para la experiencia del usuario.

La Figura 2.14 se muestra un ejemplo de código escrito en QML con el resultado, Hello, que se mostraría por pantalla al correr la aplicación.



```
import QtQuick 2.0

Rectangle {
    width: 400; height: 300
    Text {
        text: qsTr("Hello")
        font.pointSize: 25;
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
    }
}
```

Figura 2.14 Código escrito en QML

2.1.4 Librería Orocó

Es una librería de software libre, de propósito general, para el control de dispositivos robóticos. Soporta cuatro bibliotecas desarrolladas en C++:

- ✓ Real-Time Toolkit (RTT)
- ✓ Kinematics and Dynamics Library (KDL)
- ✓ Bayesian Filtering Library (BFL)
- ✓ Orocó Component Library (OCL)

La librería Kinematics and Dynamics Library (KDL) fue utilizada en el desarrollo del LVR. Con ella podemos dotar el software con la funcionalidad para hacer cálculos relacionados con los problemas cinemáticos directo e inverso. Debido a las muchas soluciones que posee la cinemática inversa, se utilizan algoritmos más sofisticados, como es el método de obtención de raíces Newton-Raphson, para obtener una aproximación al punto deseado para el robot. (Ver Figura 2.15)

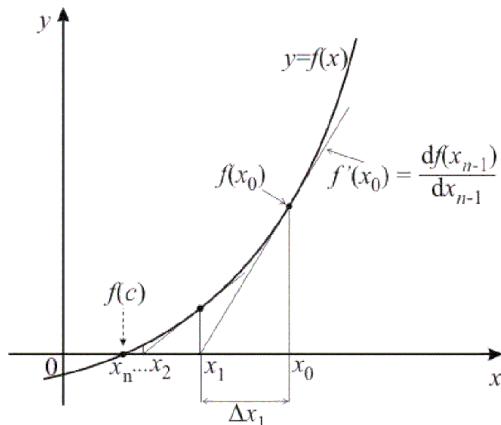


Figura 2.15: Método de Newton-Raphson

Los métodos software utilizados son **CarToJnt()** para el cálculo de la cinemática inversa y **JntToCart()** para el cálculo de la cinemática directa. En la Figura 2.16 se muestra el método **CartToJnt()** de la API de KDL. Este método espera los datos de entrada como la posición articular actual del manipulador (**q_init**) y posición cartesiana deseada (**p_in**) y como salida de datos nos da la posición articular del manipulador (**q_out**).

```
int KDL::ChainIkSolverPos_NR_JL::CartToJnt ( const JntArray & q_init,
                                              const Frame & p_in,
                                              JntArray & q_out
                                            )
                                            [virtual]
```

Calculates the joint values that correspond to the input pose given an initial guess.

Parameters:

- q_init** Initial guess for the joint values.
- p_in** The input pose of the chain tip.
- q_out** The resulting output joint values

Figura 2.16: Clase de la librería KDL para resolución de Problema Cinemático. En *ChainIkSolverPos_NR_JL* por orocos_kdl 2019.

2.1.5 Open Rave IkFast

IKFast es un módulo de **OpenRAVE**, el cual especifica que es un compilador para cinemática de robots (OpenRAVE 2018). IKFast puede resolver analíticamente las ecuaciones de la cinemática de diferentes tipos de robots generando un archivo

C++ el cual especifica la cinemática de un robot en particular. De esta forma se agiliza el desarrollo de una aplicación robótica, concentrándonos nada más en la lógica de uso de los datos de la cinemática.

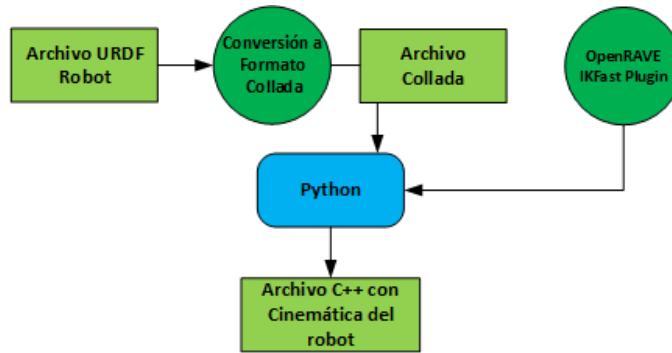


Figura 2.17: Uso del módulo IKFast para la Cinemática de robots

2.1.6 Reuleaux

Reuleaux es un plugin para RViz para visualizar **el espacio de trabajo** de diferentes tipos de robots desarrollada por usuarios de la comunidad de ROS, Makhale, A., et al (2018). Utilizan IKFast (Ver sección 2.1.5) para el mapeo de accesibilidad de un modelo de robot dado, creando poses en el entorno y calculando soluciones de IK válidas para las poses proveniente del archivo IKFast.

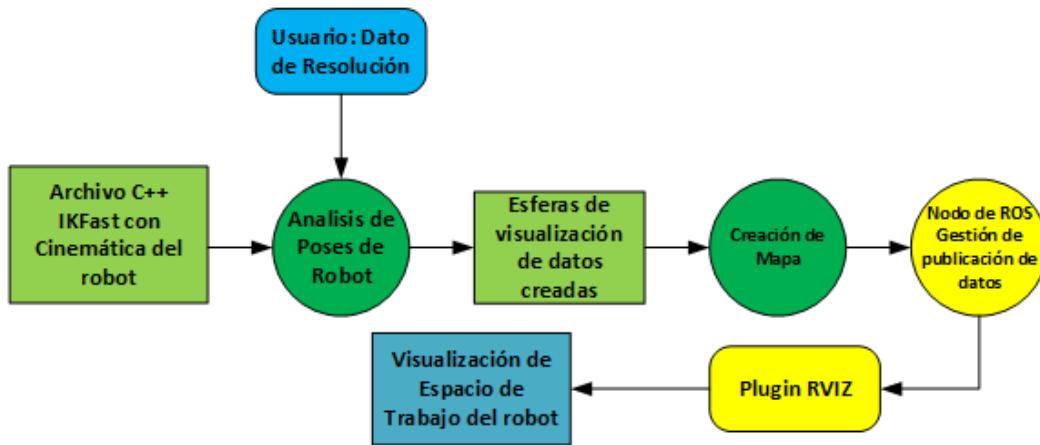


Figura 2.18: Proceso de ROS para gestionar la Visualización del espacio de trabajo

Las poses a las que puede acceder el robot están asociadas a esferas discretizadas. La accesibilidad de cada esfera en el entorno está parametrizada por un índice de alcance de 1 a 100, cambiando de colores de la esfera de acuerdo a su accesibilidad y la resolución que el usuario desea que se analice en el espacio de trabajo. El formato de archivo hdf5 guarda los detalles sobre todas las poses accesibles y esferas discretizadas. (Ver Figura 2.19)

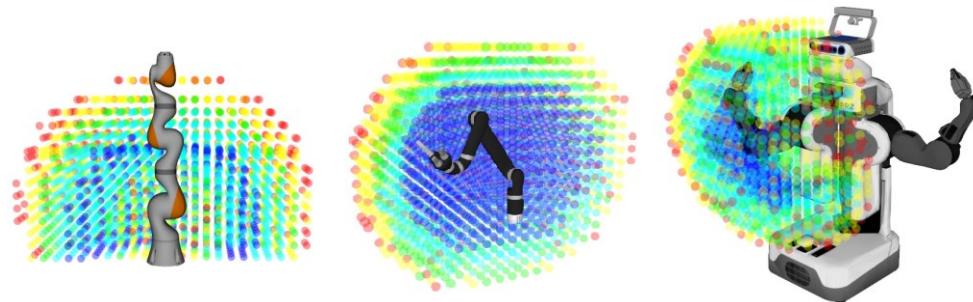


Figura 2.19: Espacios de Trabajo con Reuleaux. En Reuleaux: Robot base placement by reachability analysis por Makhale, A., et al.

2.1.7 QCustomePlot

Dentro de las interfaces de usuario que se requerían para mostrar los fundamentos de robótica era necesario contar con un graficador de datos para diferentes procesos de movimiento de los robots bajo consideración. Para esto se usó una biblioteca de expansión para QT, llamada **QCustomeplot**. Esta biblioteca de gráficos se enfoca en crear gráficos, así como en ofrecer un alto rendimiento para aplicaciones de visualización en tiempo real. Para el uso de esta librería es necesario agregar los archivos **qcustomplot.cpp** y **qcustomplot.h** al proyecto en QT para el desarrollo y configuración de las gráficas para los datos provenientes desde los nodos de ROS.

2.2 Diseño del LVR

El laboratorio virtual de robótica desarrollado pretende, entre otras cosas, recrear los recursos básicos que se encuentran en un laboratorio de robótica industrial real y brindar los recursos necesarios para la utilización de estos en la realización de experimentos. En la Figura 2.20 se muestran los elementos que generalmente se encuentran en un laboratorio real.

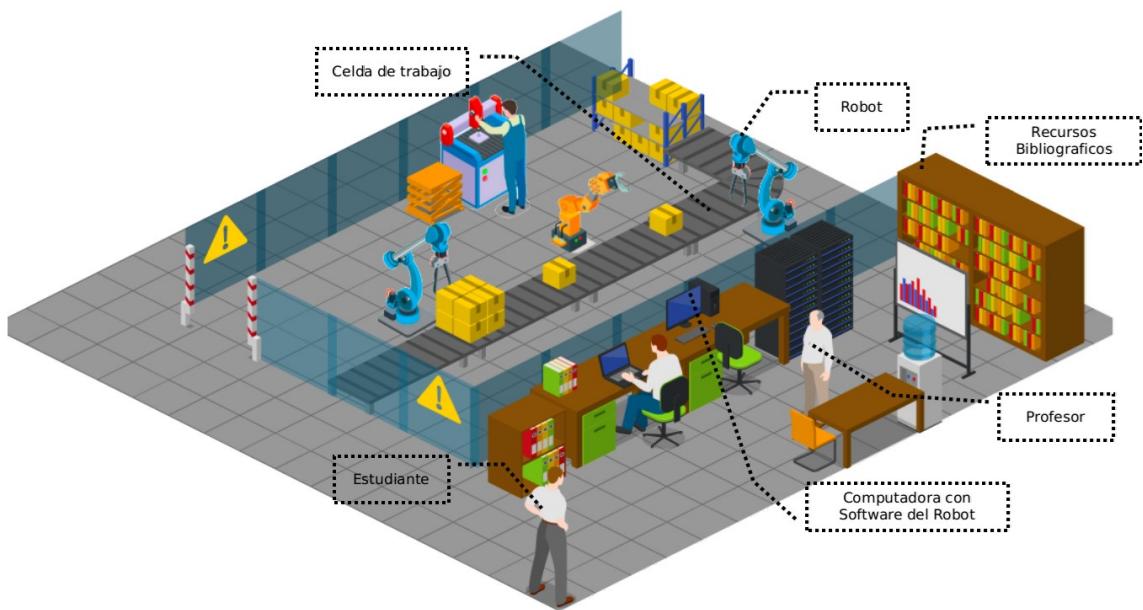


Figura 2.20: Modelo de laboratorio de robótica físico.

Básicamente se cuenta con un robot, una computadora con el software correspondiente para la programación de este, una celda de trabajo, y un espacio donde se almacena la documentación correspondiente al robot, por ejemplo, manual de usuario y/o manual de instalación, lenguaje de programación, etc.

En la revisión de literatura, y reconocimiento de los laboratorios en algunas instituciones de educación superior, encontramos que la principal actividad que se realiza en este tipo de laboratorios es la programación del robot, usando un lenguaje específico, para que ejecute una tarea determinada.

El laboratorio virtual desarrollado, ver Figura 2.21, cuenta con los recursos de un laboratorio básico de robótica e incorpora otros recursos que permiten realizar

otras actividades como comprobación de cinemática del robot yendo más allá de la simple programación del modelo del robot bajo consideración.

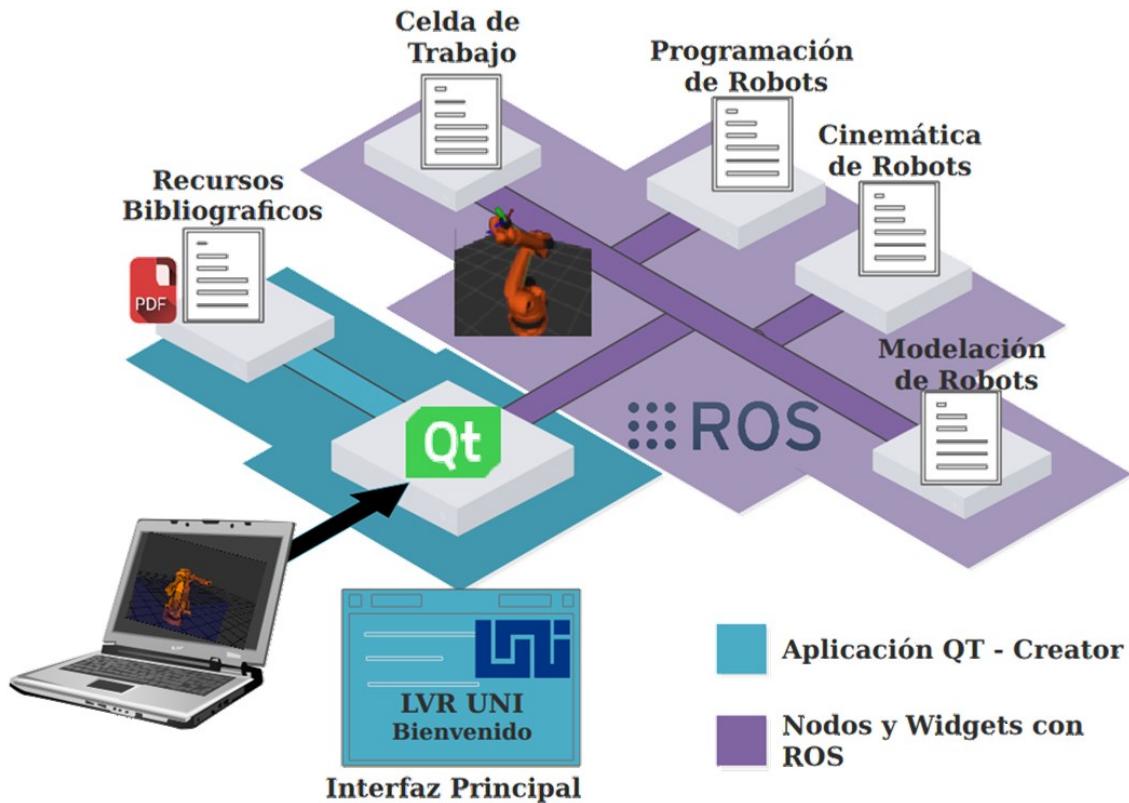


Figura 2.21: Modelo del laboratorio Virtual de Robótica Industrial.

2.2.1 Modelos de los robots

Simulación de robots puede tener significados distintos para diferentes profesionales de la Robótica. Para muchos de los que trabajan con robots la simulación de dichos sistemas abarca la visualización del movimiento del robot en su espacio de trabajo. Estos simuladores están ampliamente basados en diseños CAD y herramientas de visualización gráfica. Por lo tanto, es necesario contar con los modelos CAD de los robots para integrarlos a una simulación. Los fabricantes de robots tienen sus propios simuladores de robots, (ver Figura 2.22) y para ello utilizan archivos CAD mismos que se encuentran disponibles en sus páginas WEB. Fabricantes reconocidos como ABB, FANUC, KUKA tienen disponibles estos archivos con mayor o menor resolución como son STL, 3DS, VRML1, DXF, IGES, GoogleSketchUp, SLDASM, entre otros.

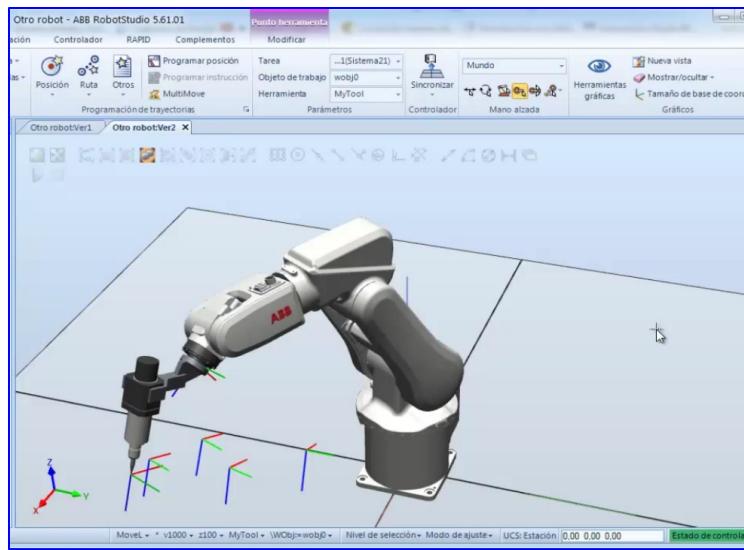


Figura 2.22: Modelo de Robot obtenido utilizando el software Robot Studio de ABB

2.2.1.1 Modelos de los Fabricantes de robots

Dentro del consorcio de empresas que usan ROS, llamado ROS-Industrial, se cuenta con la colaboración de empresas como ABB, FANUC, Yaskawa, KUKA, entre otras, que colaboran proveyendo repositorios de libre acceso a la comunidad de roboticistas y dentro de esta colaboración se tiene acceso a los modelos de robots elaborados por estas empresas con lenguaje descriptivo URDF que describen fielmente al robot tal como si se usara el modelo en el simulador del fabricante del robot.

2.2.1.2 Modelos Creado por el usuario

La modelación de un robot es posible mediante el lenguaje descriptivo URDF el cual consta de una serie de etiquetas XML que permiten especificar una serie de parámetros para cada eslabón y cada articulación del robot. En los eslabones es posible indicarle una etiqueta con referencia a un archivo CAD y proveer la información de su posición y orientación en el espacio.

2.2.1.2.1 URDF

Es necesario conocer que el modo descripción de etiquetas en un archivo URDF es prácticamente genérico para todo tipo de robots, desde un vehiculó aéreo no tripulado hasta un brazo manipulador. Sin embargo, existen ciertas limitaciones

que hacen imposible la descripción de ciertos robots. Los robots deben ser tipo series (estructura de árbol), descartando las cadenas cinemáticas cerradas.

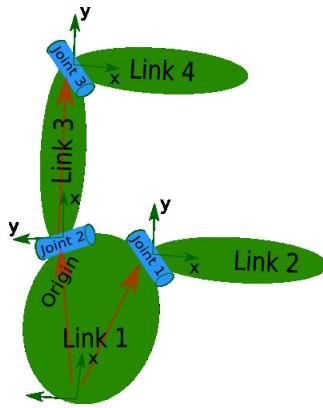


Figura 2.23: Representación de Links y Joints en un URDF.

El robot se interpreta como un conjunto de eslabones (links) unidos mediante un conjunto de articulaciones (joints) tal y como se muestra en la Figura 2.23. De esta forma el archivo URDF poseerá etiquetas de lenguaje de marcado de la forma y clasificación como se muestra en la Figura 2.24. La etiqueta **robot** es la etiqueta general de un archivo URDF

```
<robot name="manipulador">
  <link name="link_1"> ... </link>
  <link name="link_n"> ... </link>
  <joint name="joint_1" type="revolute"> ... </joint>
  <joint name="joint_n" type="prismatic"> ... </joint>
</robot>
```

Figura 2.24: Estructura del lenguaje de marcado URDF

La Figura 2.25 muestra el modelo de un robot creado usando simples etiquetas URDF. Aunque el modelo corresponde a un robot con dos eslabones, modelos más complejos, para robots con un mayor número de grados de libertad, pueden ser construidos utilizando el URDF.

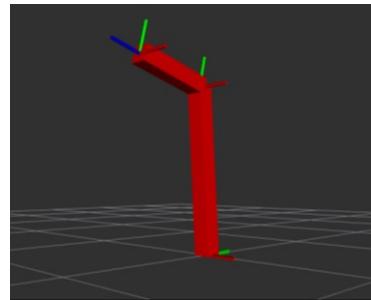


Figura 2.25 Robot 2 grados de libertad usando URDF

2.2.1.3 Soporte de ROS – Industrial

ROS – Industrial (ROS – I) es un consorcio de empresas que utilizan los recursos de ROS a nivel industrial. Las implementaciones de software para aplicaciones industriales han dado origen a nuevos paquetes de software de ROS tales como MoveIt para la planeación de trayectorias para los robots industriales, además de la colaboración de los desarrolladores de las propias empresas como ABB, FANUC, KUKA, Motoman, entre otras, para proveer de paquetes que interactúen con ROS, por ejemplo, la capacidad de ROS – I de comunicarse vía un nodo de ROS con el controlador (CPU) de un Robot Industrial.

2.2.1.3.1 Estructura de un paquete ROS-I

Dentro del desarrollo de este trabajo se utilizaron los paquetes de ROS Industrial (Ver Figura 2.26) estos poseen archivos URDF de robots industriales, archivos de configuración de los joints del robot, archivos launch para aplicaciones con robots reales, archivos CAD (meshes) y archivos de prueba para la puesta en marcha de un robot.

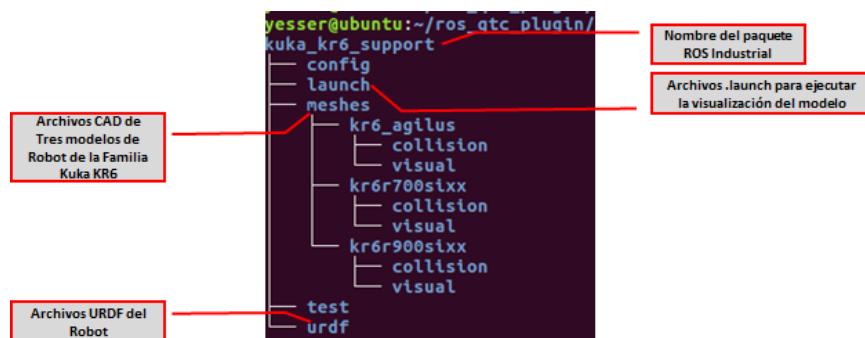


Figura 2.26: Paquete de Modelo de un Robot Kuka

Los archivos URDF pueden ser configurados por el usuario tal y como se menciona en la sección 2.2.1.2 pero para el LVR se usaron tambien los archivos URDF que han parametrizado las empresas unidas al consorcio de ROS tales como ABB, FANUC, Motoman etc., garantizando de esta manera un robot simulado que cumple a cabalidad la hoja de datos proveída por el fabricante. En estos cumplimientos están la máxima apertura de los Joints en radianes, la inercia de los eslabones del robot en $\text{Kg} * \text{m}^2$.

2.2.1.3.1.1 Archivos CAD de los robots industriales

El uso de archivos CAD en este trabajo tiene mucha importancia ya que brinda a los estudiantes el acercamiento a un robot modelado en 3D. Con los paquetes de ROS Industrial se obtiene el modelo del robot bajo las mismas dimensiones que el robot físico, este detalle cobra importancia al ejecutar análisis de cinemática y en análisis más avanzados dentro de la dinámica del modelo en 3D. Además, dentro de este paquete se encuentran dos tipos de CAD; la **categoría Visual** para obtener la representación más fidedigna y la **categoría Colisión** para análisis de trayectorias y dentro del Simulador avanzado de Gazebo Colisionar con objetos y ver su respuesta. (Ver Figura 2.27)

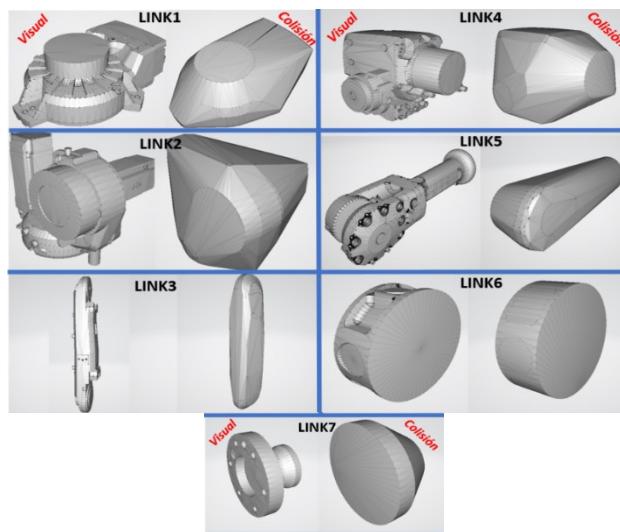


Figura 2.27: Modelos de CAD en 3D

2.2.2 Diseño de la interfaz principal.

La interfaz principal está diseñada bajo el lenguaje **QML**, de QT Creator, creado para desarrollar aplicaciones dinámicas. Para el diseño de la interfaz se tomaron en cuenta los siguientes requerimientos, que debe garantizar la interfaz, tal como se especifica en la Figura 2.28

- ✓ Centralización de todos los recursos del LVR en una interfaz principal.
- ✓ Visualización de Recursos bibliográficos y visuales (Videos).
- ✓ Manejo de Procesos (Thread) para ejecutar los nodos de ROS sin necesidad de usar línea de comandos por terminal Linux.

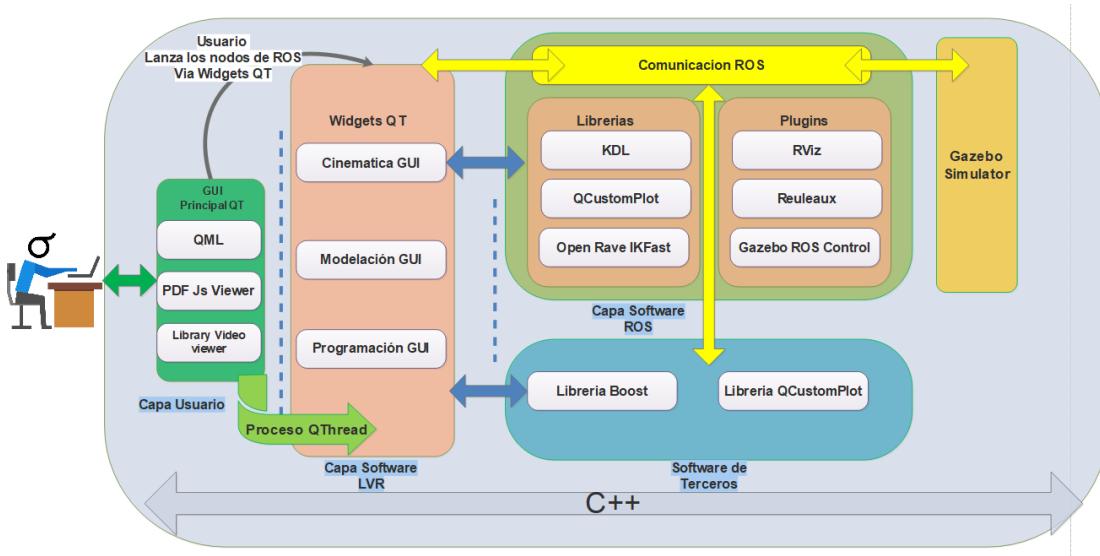


Figura 2.28: Diseño de interfaz principal por capa de software.

2.2.2.1 Recursos Bibliográficos

Los recursos bibliográficos del LVR están relacionados con los acápite de la sección 1.3.2 “Fundamentos de la robótica industrial. Los mismos están basados en los primeros 4 capítulos del libro de Barrientos et al (2007) el cual fue utilizado en la asignatura control aplicado cuando surgió la idea de desarrollar un laboratorio virtual para robótica industrial. En los recursos bibliográficos se

encuentran otros libros de robótica que son considerados complementarios al de Barrientos y que serán de mucha utilidad a los usuarios del laboratorio.

El espacio “recursos bibliográficos” fue creado utilizando el lenguaje QML y un visor de archivos pdf el cual es lanzado vía una página Web bajo el plugin del visor **PDF Js Viewer**. Todo esto dentro de la misma Interfaz de usuario, garantizando además la centralización de la información para el estudiante.

2.2.2.2 Recursos Visuales

Desde la interfaz de usuario principal el estudiante podrá acceder a medios visuales como imágenes o videos. Esta funcionalidad se logró por medio de la librería para QML **QtMultimedia 5**. Lo anterior permitirá al estudiante adoptar un concepto de robótica con medios visuales que le describan de manera más detallada el concepto en la práctica.

2.2.2.3 Ejecución de Procesos de ROS

La interfaz de usuario principal fue diseñada para que se realicen procesos con solo ejecutar un Widget (Push button o Check box) y lanzar las demás aplicaciones, nodos de ROS, que usará el estudiante, (ver Figura 2.29). Para lanzar los nodos ROS se usa la clase de Qt llamada **QProcess** la cual permite iniciar programas externos bajo la administración del usuario y eliminar procesos que estaban en ejecución para evitar conflictos.

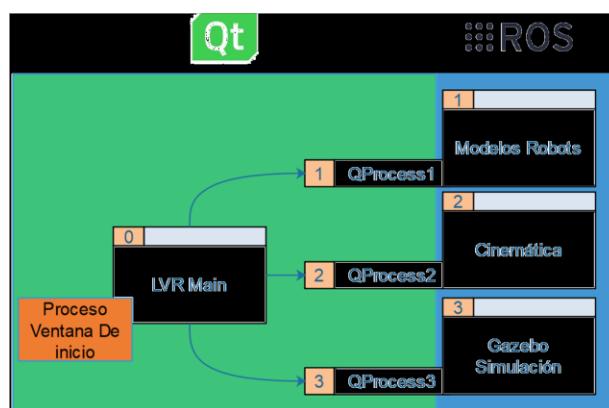


Figura 2.29: Procesos a ejecutarse dentro de la interfaz principal

2.2.3 Diseño de la Interfaz Modelación de Robots

La morfología del robot es un concepto muy importante y fue abordado en la Sección 1.3.2 “Fundamentos de la robótica industrial. En la interfaz Modelación de Robots se utilizan Widgets de Qt y Plugins de ROS los cuales permitirán al estudiante ejecutar prácticas con el lenguaje de marcado URDF (ver sección 2.2.1.2.1 URDF). El estudiante tendrá la posibilidad de modelar cualquier robot antropomórfico que desee. En la Figura 2.30 se muestra el diseño general de la interfaz y las herramientas de ROS que destacan para la implementación como son RViz y KDL, garantizando de esta forma que el estudiante cuente con una herramienta de software para construir modelos personalizados de diferentes robots

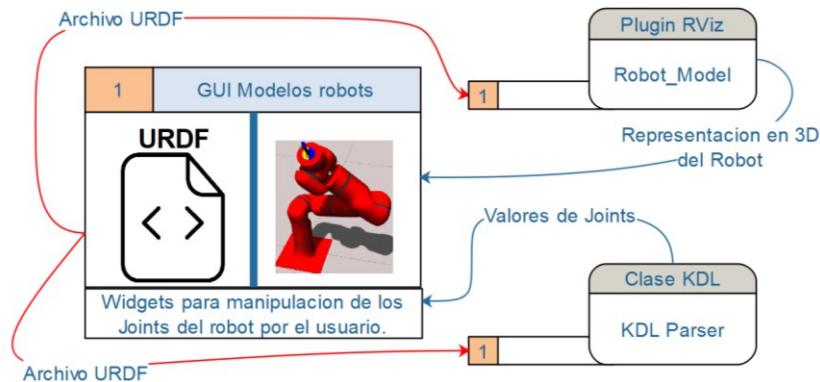


Figura 2.30: Diseño de la interfaz de Modelación de Robots.

2.2.4 Diseño de la Interfaz de Matemática y Cinemática de Robots

El diseño de esta interfaz considera varios aspectos relacionados con las herramientas de matemática y con elementos de la cinemática de los robots antropomórficos o series (ver Figura 2.31). Se usa el plugin de RViz para la visualización de los archivos CAD de los robots, herramientas de interacción del usuario con el robot, visualización del espacio de trabajo de los robots, además del uso de la librería KDL para la conversión matemática de las coordenadas de posición de los robots, el análisis cinemático y la conversión de marcos de referencia del algoritmo Denavit-Hartenberg. Se incorporó además la librería **QCUSTOMPLOT** para graficar los movimientos del robot analizado.



Figura 2.31: Conceptos a desarrollar en interfaz de matemática y cinemática del robot.

El uso de la interfaz Matemática y Cinemática del robot comienza con la inicialización de esta lo cual permitirá seleccionar el modelo del robot que será analizado y dará acceso a las subinterfaces, Matemática, Cinemática y Representación de Denavit-Hartenberg. Una vez en la subinterfaz el estudiante podrá hacer cambios con los widgets para manipular los ángulos de los Joints, la posición del efecto final, entre otras cosas y observar el cambio correspondiente en la posición del modelo del robot. En la figura 2.32 se muestra el camino a seguir.

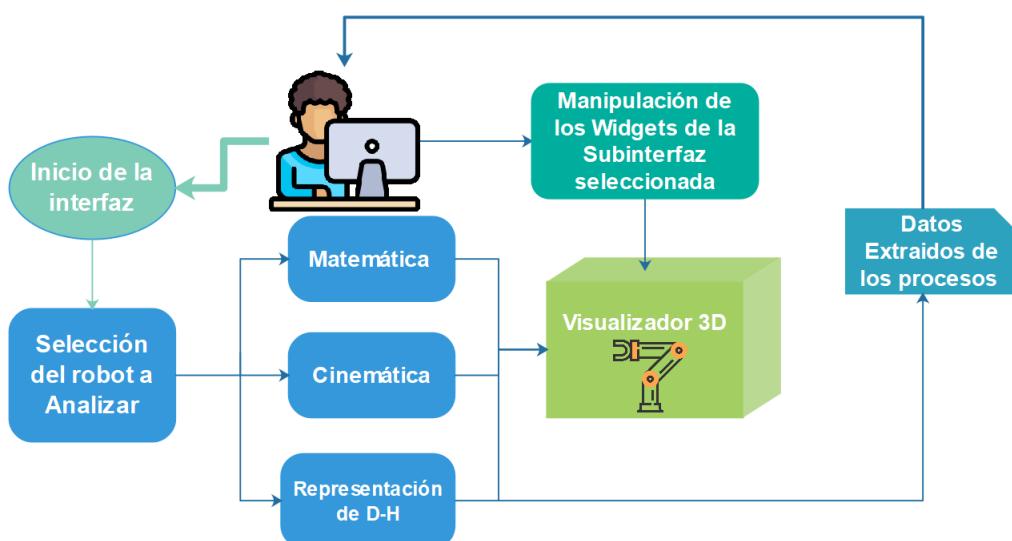


Figura 2.32 Proceso de Interacción con la interfaz de Matemática y cinemática.

2.2.4.1 Matemática de los robots

Para entender como un robot se mueve en su espacio de trabajo es necesario entender el concepto de posición y orientación de los elementos a manipular con respecto a la base del robot o mundo. En la subinterfaz de matemática se abordan conceptos de conversión de coordenadas de posición y orientación de un sólido en un ambiente 3D como se puede observar en la Figura 2.33, esto con el fin de proveer al estudiante de una herramienta útil que le permita asimilar con facilidad dichos conceptos. Las Tablas 2.5 y 2.6 muestran las relaciones existentes entre las diferentes representaciones de coordenadas desde la Matriz de Transformación Homogénea que engloba la posición y orientación de un robot hasta la representación singular de diferentes tipos de coordenadas.

Tabla 2.5 Conversión de coordenadas de posición y orientación

Coordenadas de Posición y Orientación				
Matriz de Transformación Homogénea	Conversion	Matriz de posicion	Conversion	Coordenadas cilíndricas
		Matriz de Orientación		Coordenadas Esféricas
	Conversion		Conversion	Ángulos de Euler
				Representación de Cuaternios

Tabla 2.6: Representación Matemática de Coordenadas de Posición y Orientación

Representación	Ecuación
Matriz de Transformación Homogénea	$R_i = \begin{pmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{pmatrix} = R_i = \begin{pmatrix} \text{Rotacion} & \text{Traslacion} \\ \text{Perpectiva} & \text{Escalado} \end{pmatrix} =$ $R_i = \begin{pmatrix} r11 & r12 & r13 & px \\ r21 & r22 & r23 & py \\ r31 & r32 & r33 & pz \\ 0 & 0 & 0 & 1 \end{pmatrix}$
Matriz de Orientación	$R_i = \begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$
Matriz de posición	$R_i = \begin{pmatrix} px \\ py \\ pz \end{pmatrix}$
Ángulos de Euler XYZ	$\theta = \text{Atan2}(-r31, \sqrt{r31^2 + r21^2})$ $\psi = \text{Atan2}\left(\frac{r21}{\cos \theta}, \frac{r11}{\cos \theta}\right)$ $\phi = \text{Atan2}\left(\frac{r32}{\cos \theta}, \frac{r33}{\cos \theta}\right)$
Representación de Cuaternios	$\epsilon_0 = \frac{1}{2}\sqrt{1 + r11 + r22 + r33}$ $\epsilon_1 = \frac{r32 - r23}{4\epsilon_0}$ $\epsilon_2 = \frac{r13 - r31}{4\epsilon_0}$ $\epsilon_3 = \frac{r21 - r12}{4\epsilon_0}$
Coordenadas Cilíndricas	$r = \sqrt{px^2 + py^2} \quad \theta = \tan^{-1} \frac{py}{px} \quad z = pz$
Coordenadas Esféricas	$\rho = \sqrt{px^2 + py^2 + pz^2} \quad \theta = \tan^{-1} \frac{py}{px} \quad \varphi = \cos^{-1} \frac{pz}{\sqrt{px^2 + py^2 + pz^2}}$

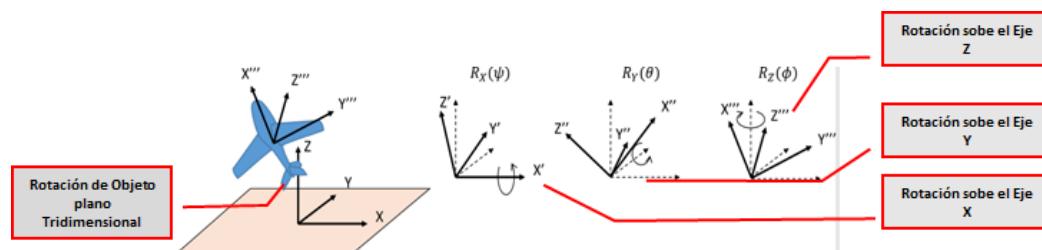


Figura 2.33: Objeto puesto a rotación bajo coordenadas de orientación o Ángulos de Euler XYZ

2.2.4.2 Denavit - Hartenberg

La subinterfaz de Denavit – Hartenberg se diseña de tal forma que el estudiante sea capaz de introducir los valores de los parámetros de Denavit - Hartenberg del robot y en respuesta confirmar la validez de estos mediante la visualización del modelo del robot en un ambiente 3D. En el capítulo 4.1.3 del libro “Fundamentos de Robótica” de Barrientos se describe el algoritmo de Denavit-Hartenberg que permite obtener los valores de los parámetros de dicho modelo. Los parámetros son descritos en un apartado posterior del documento.

Tabla 2.7: Parámetros de Denavit - Hartenberg

Θ_i :	Rotación alrededor del eje z_{i-1} .
Vector $d_i (0, 0, d_i)$:	Traslación a lo largo de z_{i-1} una distancia d_i .
Vector $a_i (a_i, 0, 0)$:	Traslación a lo largo de x_i una distancia a_i .
α_i :	Rotación alrededor del eje x_i

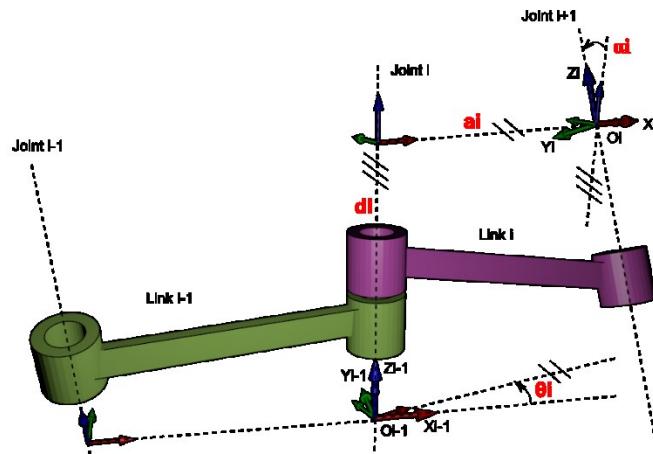


Figura 2.34 Aplicación de Algoritmo Denavit - Hartenberg

En el desarrollo del laboratorio virtual se utilizó la librería KDL la cual posee funciones que permiten la construcción del modelo de un robot utilizando los parámetros de Denavit-Hartenberg. El modelo del robot bajo consideración estará conformado por una cadena de eslabones cuya efectividad puede ser comprobada desde la interfaz tanto de forma visual como matemática.

2.2.4.3 Cinemática de Robots

La interfaz correspondiente al tema Cinemática Directa e Inversa, fue diseñada, utilizando las herramientas y plugins de ROS listados en la Tabla 2.8 para facilitar al estudiante la comprensión de los problemas de la cinemática directa e inversa considerados en la sección 1.3.2.3. Se asocian los diferentes modelos elaborados por el desarrollador en URDF y los modelos elaborados por el fabricante y mediante el uso de diferentes librerías se realizan un conjunto de procesos que permiten obtener los datos de interés relacionados con los problemas mencionados.

Cabe destacar que existen varios métodos para la solución del problema cinemático directo siendo el más común, y más práctico, el de Denavit-Hartenberg el cual permite obtener un modelo simplificado del robot sin importar el número de joints a utilizar. Otro método es el de uso de Cuaternios para encontrar la cinemática directa, el cual no será abordado en este diseño, pero el cual es válido mencionar.

Para obtener una solución al problema de la cinemática inversa es necesario el uso de algoritmos más sofisticados ya que para una posición y orientación de un robot hay un sinnúmero de posibles soluciones de acuerdo con la morfología del robot. Para hacer un análisis el LVR utiliza KDL de forma general e IkFast para análisis de la accesibilidad (Espacio de Trabajo) de un robot obteniendo como resultado el espacio de trabajo del robot.

Tabla 2.8: Tabla de Paquetes de la interfaz de Matemática y Cinemática de Robots

Paquete	Categoría	Propósito de Aplicación	Capítulo vinculado
Orocos_KDL	Librería de Cinemática	Cinemática Directa e inversa a través de URDF.	2.6.1, 2.6.2, 2.6.3
TF ROS	Librería de Transformaciones	Cambio de marcos de referencia del robot.	2.6.1
Reuleaux	Plugin de Rviz	Muestra el espacio de Trabajo de robots industriales	2.6.3
IkFast	Módulo de Open Rave	Generador de Archivo que describe la cinemática del robot	2.6.3
RViz	Visor de datos 3D de ROS	Se usa el código fuente (Librerías y Plugins) del visor de ROS para incorporarlo en la Interfaz del LVR	2.5, 2.6.1, 2.6.2, 2.6.3
QCustomPlot	Librería de Gráficos	Uso para las gráficas de movimientos de los Joints	2.6.3

2.2.5 Diseño del módulo de Programación de los Robots.

Uno de los aspectos más importantes en la robótica es la programación y simulación de los robots. La simulación permite verificar que el robot realiza adecuadamente la tarea para la cual es programado antes de probar su funcionamiento en el mundo real. En el laboratorio virtual se utiliza Gazebo para realizar simulación de los robots y es posible enviar desde una interfaz de usuario los parámetros de configuración del robot a controlar.

Cabe destacar que cada robot posee su propia morfología, cinemática y dinámica característica y para el control de cada modelo es necesario ajustar los parámetros del control PID para cada Joint del robot usando el plugin ROS Control. (Ver Figura 2.35)

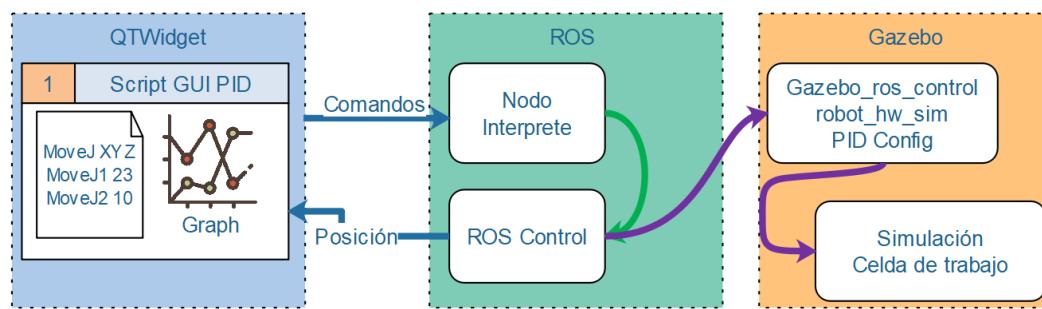


Figura 2.35: Interacción de Gazebo con ROS e interfaz de usuario

2.2.5.1 Celda de trabajo con GAZEBO

La celda de trabajo fue diseñada tratando de recrear lo más exacto posible una celda de trabajo real con la intención de permitir a los estudiantes una idea lo más real posible del ambiente de trabajo en el cual interviene un robot.

Al mundo simulado en Gazebo es posible añadirle modelos 3D, así como definir las colisiones mediante las cuales el robot puede experimentar acciones como las que se presentan en el mundo real al colisionar con objetos que se encuentren alrededor del robot, por eso es necesario conocer el espacio de trabajo al cual el robot puede acceder. Al modelo del robot cargado se le define (vincula) el plugin

de **ros_control** el cual controlará los Joints del robot para que garantizar la interacción de este en la celda de trabajo. Ver Figura 2.36.

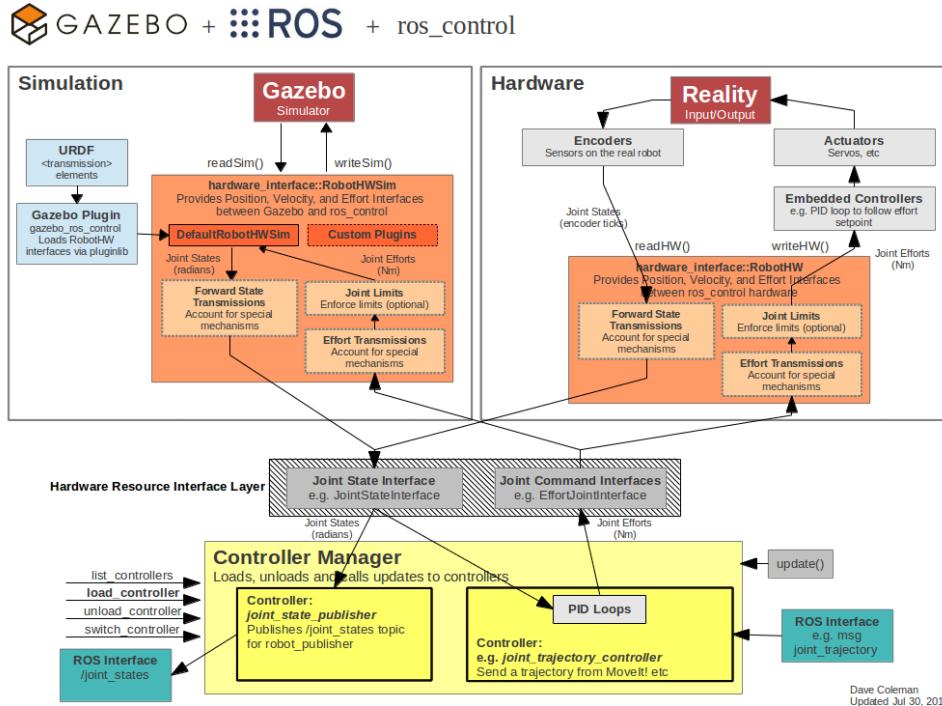


Figura 2.36: *ros_control* y su interacción con el mundo de Gazebo. Tomado de: http://gazebosim.org/tutorials/?tut=ros_control

Con el plugin de **ros_control** es posible configurar los parámetros del robot simulado y el del robot real simultáneamente usando control PID. En el presente trabajo solo se realiza la simulación del robot dentro del mundo de Gazebo

2.2.5.2 Diseño de la interfaz interprete de comandos.

La interacción del usuario con el modelo del robot a simular se realizará a través de una interfaz de usuario donde se escribirá el **Script de Control** de cada uno de los Joints del robot a controlar. Una opción es configurar el PID de cada elemento del robot y visualizar gráficamente el movimiento de los Joints del robot, como se muestra en el diagrama de la Figura 2.35. En la Tabla 2.9 se muestra los paquetes y librería de ROS y las herramientas del simulador GAZEBO utilizados.

Tabla 2.9: Paquetes utilizado para la programación y simulación del Robot en Gazebo

Software	Categoría	Propósito de Aplicación	Capítulo vinculado
Gazebo	Simulador	Modelación del mundo virtual para simular el robot.	2.7
ros_control	Librería de control	Controlar los Joints del robot utilizando PID	2.7
Kuka ros_industrial	Modelos de robot	Robot Kuka a simular dentro del mundo	2.7
Joint_state messages	Mensajes de ROS	Mensajes de la posición del robot que son tomados de la simulación	2.7

2.3 Implementación del LVR

En el acápite anterior se describieron los diferentes recursos del laboratorio virtual. A continuación, se explica el procedimiento y el uso de las herramientas para la implementación del LVR con sus diferentes subsistemas de software.

2.3.1 Implementación de la interfaz principal.

La implementación de la interfaz principal, tal y como fue descrito en la sección 2.2.2, fue realizada utilizando Qt Creator.

En la Figura 2.37 se muestra el diagrama de la interfaz principal.

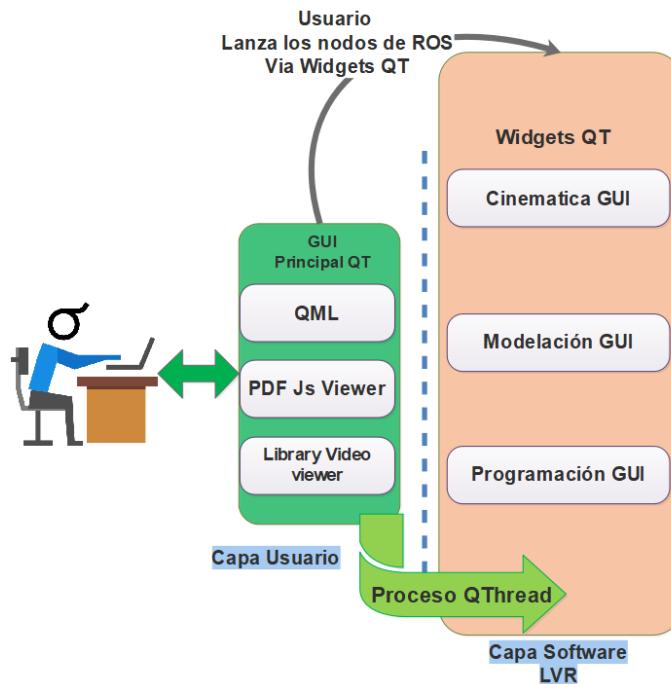


Figura 2.37: Diseño Interfaz Principal

La interfaz principal fue desarrollada utilizando QML, lenguaje de programación de Qt-Creator muy parecido a un archivo HTML. Con el lenguaje QML se define un encabezado principal (header) y un cuerpo que contiene los diferentes widgets; botones, barras deslizantes, cajas de texto, etc., de la interfaz. El uso de llamadas o vínculos a otros módulos escritos en QML muestran una interfaz gráfica dinámica y atractiva.

```

1 import QtQuick 2.9
2 import QtQuick.Controls 2.2
3 import QtQuick.Window 2.3
4 import QtQml.Models 2.1
5 import "qml"
6
7 Window {
8     id:mainsc
9     visible: true
10    width: Screen.width-45
11    height: Screen.height
12    title: qsTr("Virtual Laboratory")
13
14    Welcome {
15        id:welcome
16        ...
17    }
18
19    MainLab{
20        id:mainLab
21    }
}

```

Figura 2.38: Muestra de parte del código principal.

En la Figura 2.38 se muestra el código requerido para agregar dos ítems “Welcome” y “Main Lab”, dos archivos QML gestionados para acceder a la pantalla de bienvenida y a la pantalla de navegación entre los diferentes elementos del LVR.

En el elemento “Welcome” (pantalla inicial) se muestra una introducción al estudiante de los diferentes recursos del LVR. Esta interfaz (Figura 2.39) cuenta con diálogos, botones, imágenes planas e imágenes Gif, un Script en JavaScript que gestiona la hora y el tiempo que el usuario utiliza en el LVR.

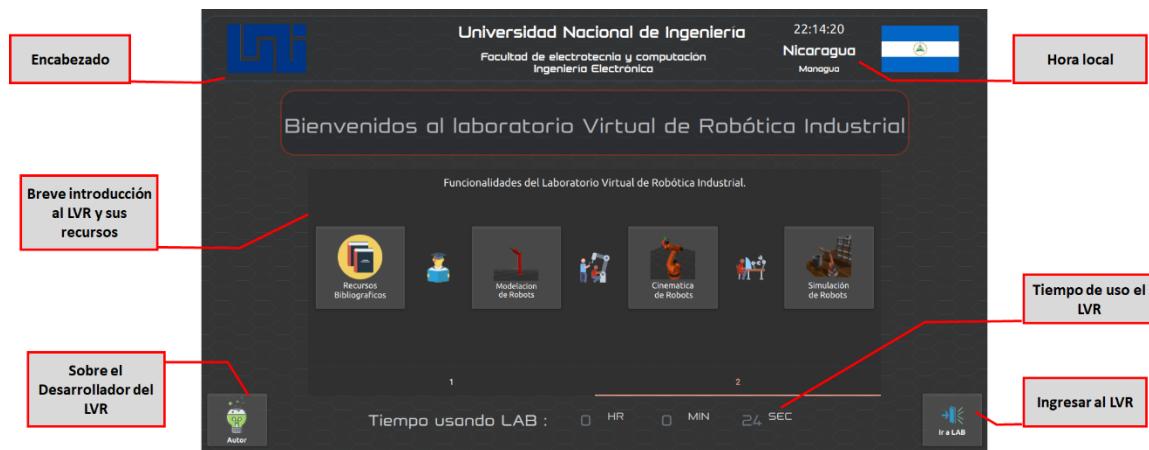


Figura 2.39: Pantalla de Bienvenida al LVR

Cuando el usuario ingresa al LVR (elemento MainLab) se puede acceder al menú principal de los recursos de lectura, visuales y animados y lanzar de manera transparente los nodos de ROS para las prácticas. La interfaz consta de cuatro categorías que son Recursos bibliográficos y visuales, Morfología y modelación de robots, Cinemática de robots y simulación de robots tal y como se aprecia en la Figura 2.40



Figura 2.40: Muestra de la pantalla principal.

2.3.1.1 Recursos Bibliográficos

Para mostrar los recursos bibliográficos dentro de la interfaz gráfica de usuario se utilizó un visor de archivos en PDF de libre acceso llamado PDF.Js, gestionado y administrado por **Mozilla Labs**. La librería necesita de un navegador Web para mostrar los archivos referenciados vía un enlace Web ya sea apuntando a un archivo que se encuentre localmente en una PC o en un servidor FTP, en nuestro caso lo hacemos localmente, referenciando a los diferentes archivos PDF que son necesarios para el material de robótica de este LVR.



Figura 2.41: Configuración de la librería PDF.Js en QML

En la Figura 2.41 se muestra el uso del **QtWebEngineView** de QML el cual es un módulo de Qt de Renderizado de páginas Web basado en la versión libre de Google Chromium, el cual soporta cualquier tecnología Web. En la programación del módulo se le referencia el link del URL a cargar dentro del proceso Web, permitiendo de esta manera mostrar los recursos en PDF, Ver Figura 2.42.



Figura 2.42: Recurso Bibliográfico dentro del LVR.

2.3.1.2 Recursos Visuales

Los recursos visuales en el LVR son muy importantes ya que permiten al estudiante la visualización e interacción con el objeto de estudio. El desarrollo de dicha funcionalidad se logró mediante el uso de **QtMultiMedia 5**, un módulo nativo de QML que al editar las propiedades (Ver Figura 2.43) se le puede programar como un reproductor de video dentro de la interfaz.

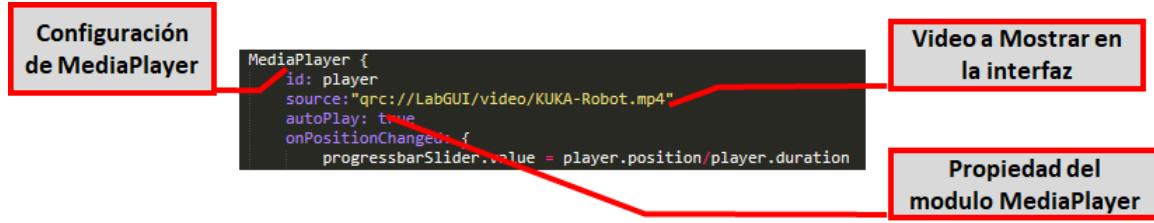


Figura 2.43: Propiedades del Módulo de Reproducción de Video.

En la Figura 2.44 se muestra contenido multimedia del laboratorio virtual de robótica. El reproductor de contenido cuenta con una barra de herramientas para el control y volumen de la reproducción. En el laboratorio virtual de robótica contiene diferentes tipos de videos, ya sean tutoriales o ejemplos, donde se muestra el uso de robots. La mayoría de los videos son provenientes de YouTube y Vimeo.

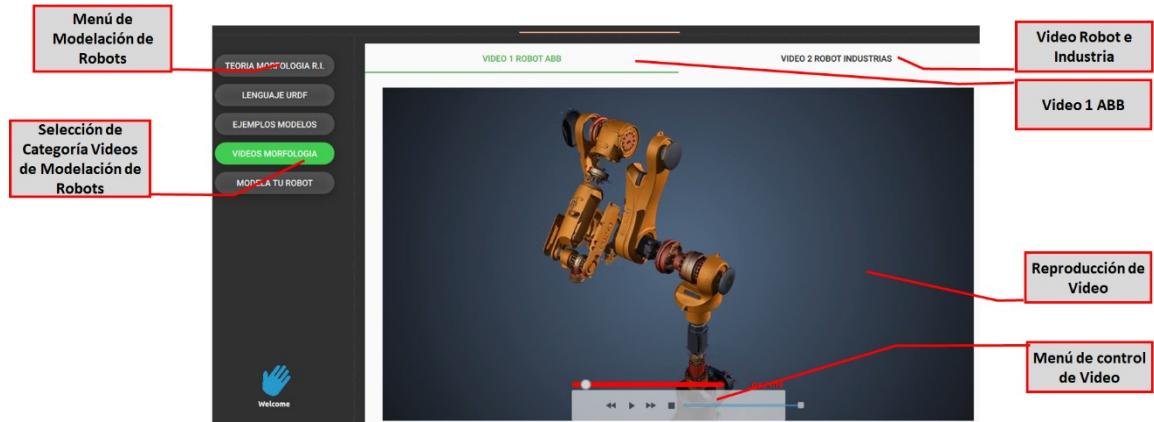


Figura 2.44: Contenido Multimedia en la interfaz del LVR.

2.3.1.3 Ejecución de Procesos de ROS

Como se muestra en la Figura 2.29, es necesario que la interfaz principal cuente con la capacidad de lanzar y administrar los nodos de ROS en caso de que se tenga que detener el proceso para dar chance a la realización de otra práctica. Los procesos hijos de la interfaz principal deben poder ser iniciados por separado en algún punto del código.

La clase utilizada se llama QProcess la cual está escrita en C++ y, debido a que la interfaz principal está escrita en QML, se aprovecha la capacidad de QT para poder importar una clase de C++ a la interfaz, demostrando de esta manera que se puede implementar con QML una interfaz dinámica y capaz de comunicarse con cualquiera de las numerosas librerías escritas en C++.

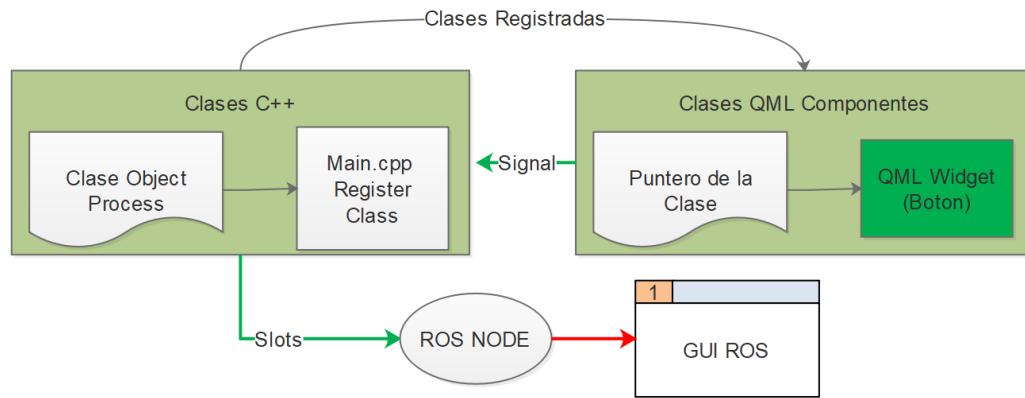


Figura 2.45: Clases en C++ y Componentes de QML

En la Figura 2.45 se muestra el proceso realizado para ejecutar clases de C++ con QML donde un Widget (botón) emite una señal al objeto de la clase a ejecutar la cual en nuestro caso es un proceso hijo de la aplicación principal que lanza un nodo de ROS. En la Figura 2.46 se especifica la lógica del objeto en C++, donde primero se eliminan (Kill) los procesos relacionados con una práctica lo cual evita abrir por ejemplo la interfaz de cinemática repetidas veces o tener nodos repetidos con diferente flujo de información. Se configura el proceso a ejecutar y se apunta a la función (Start) dentro de la clase QProcess, la cual espera un comando en String para ejecutar un Proceso. En la misma Figura 2.46 se observa la definición del proceso hacia un nodo ROS con el comando `rosrun` (Ver **¡Error! No se encuentra el origen de la referencia.**).

```
process->start("Comando")
```

Launcher.cpp



Figura 2.46: Código de la clase en C++ Registrada en QML

2.3.2 Implementación de la Interfaz Modelación de Robots

En la sección 2.2.3 “Diseño de la Interfaz Modelación de Robots” se especifica el proceso de escritura de un URDF que puede ser editado por el usuario cambiando las dimensiones de los eslabones, el tipo de Joints (Figura 2.47) o especificando en grados las aperturas de los ángulos de los mismos. Una vez editados los cambios del URDF estos son mostrados automáticamente en el visor de robots RVIZ.

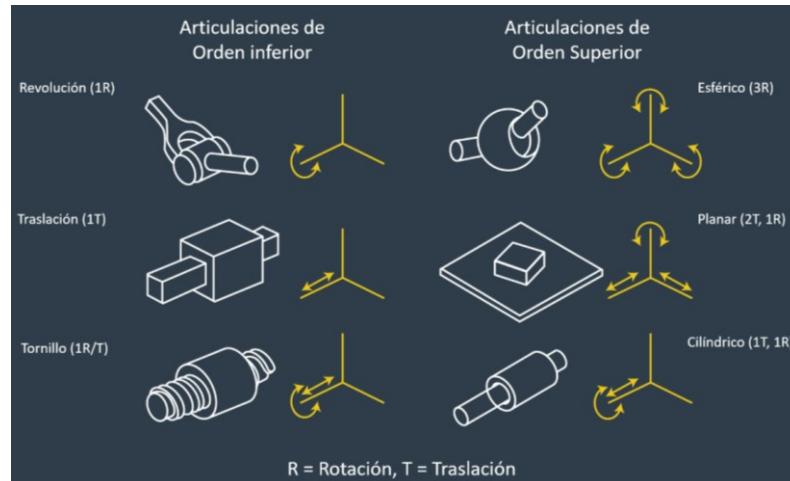


Figura 2.47: Tipos de Joints

2.3.2.1 Creación de Interfaz en Qt.

La creación de la interfaz Qt cumple análogamente el lineamiento de desarrollo Web, en el que se cuenta con el Front End y el Back End mencionado en la

sección 1.3.1 “Interfaz gráfica de usuario (GUI)”. Se implementó la Ventana de Modelación de Robots utilizando Widgets QT, con cuatro secciones de importancia:

- 1) Visualización del Script en URDF.
- 2) Un Panel de herramientas para gestionar el Guardado y Visualización del Robot.
- 3) Un Widget donde se vincula el Visor 3D RViz para poder Ver el Robot
- 4) Un panel de Widgets Slider para el control de los eslabones.

Todas estas funcionalidades están especificadas en la Figura 2.30: Diseño de la interfaz de Modelación de Robots.

Dentro de la programación de la interfaz, al usar los Widgets más comunes, se decidió agregarle un complemento visual a la edición de los parámetros del URDF, asemejándose a los programas de edición de software básicos, como el autocompletado y resaltado (HighLight) de palabras reservadas. (Ver Figura 2.48)

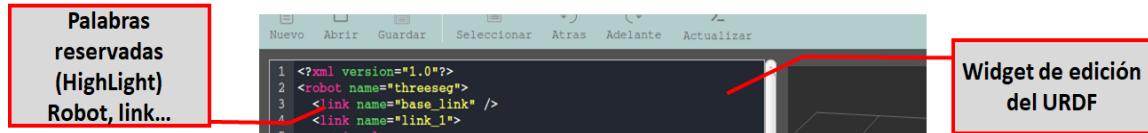


Figura 2.48: Palabras reservadas dentro de la interfaz.

2.3.2.2 Programación de los recursos de ROS

La programación de los recursos de ROS en la modelación de los robots, cinemática y programación se enfoca en crear un sistema no convencional al usar ROS, con esto se hace referencia al hecho de garantizar al estudiante un sistema transparente.

Por ejemplo, una vista común del uso del visualizador de robots RViz es la mostrada en la Figura 2.49 el cual muestra la ventana de visualización típica de

RViz con muchos plugin cargados que podrían dificultar el uso si se hubiese hecho un LVR con las mínimas configuraciones y lógica de programación en C++.

En cambio, en la Figura 2.52 se muestra el desarrollo de la ventana de Modelación de Robots enfocada a un proceso de aprendizaje específico, valga la redundancia Modelar Robots solamente.

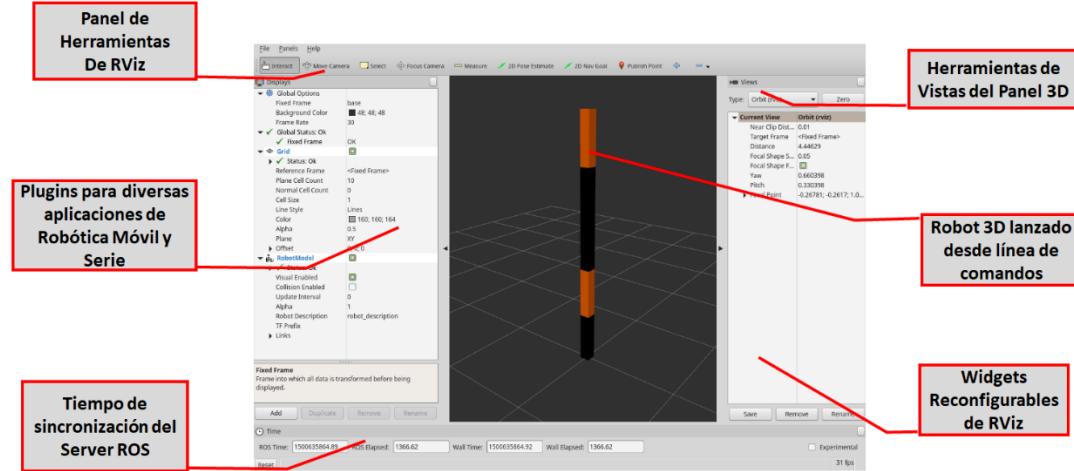


Figura 2.49: Vista de RViz Por Default.

La interfaz gráfica de usuario, programada con el lenguaje C++, posee un conjunto de archivos (Cmake, Package, URDFs, etc) los cuales se encuentran en el espacio de trabajo de ROS como un paquete que debe ser compilado (ver sección 2.1.2.3). Los widgets incorporados en la interfaz gráfica de usuario crean varios procesos con ROS los cuales son mostrados en la figura 2.50.

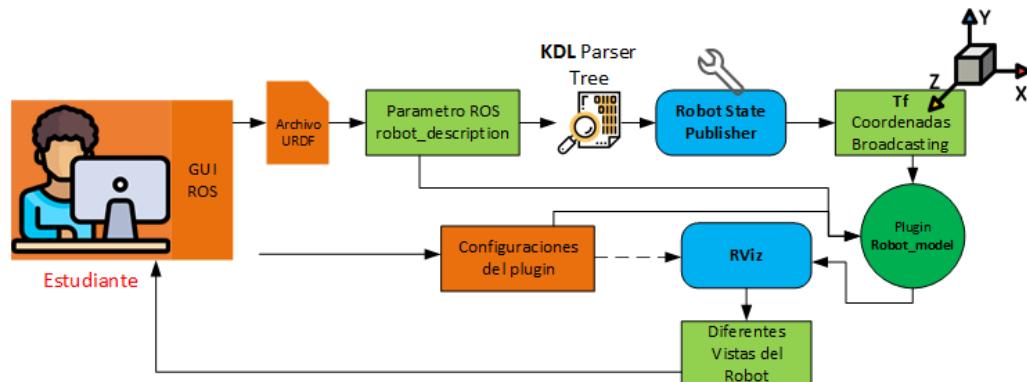


Figura 2.50: Representación secuencial del modelo de software de Modelación de Robots.

En la figura se puede observar el modelo de software para ver el modelo de robot creado y personalizado por el estudiante, usando URDF. Si el modelo tiene una sintaxis de lenguaje de marcado xml con las propiedades de URDF, se registra como un parámetro dentro de ROS bajo el tópico *robot_description* (*ver código en la parte inferior de este párrafo*). El tópico es leído por una función de la clase KDL_Parser, la cual crea un objeto KDL_Tree, donde se especifica la estructura del robot definido en el archivo URDF editado. Este Objeto KDL_Tree es publicado en un tópico llamado ***robot_state_publisher*** el cual publica en ROS las coordenadas de cada elemento y joint del robot.

```
robot_model_->subProp("TF Prefix")->setValue("robot_editor");
robot_model_->subProp("Robot Description")->setValue("robot_description");
robot_preview.cpp
```

Para definir el nombre del marco de referencia al que está anclado el robot se utiliza el puntero ***robot_state_pub_*** tal como se muestra el código mostrado. Lo anterior permite el uso del plugin de **RViz** ***robot_model*** con el cual se muestra una representación visual del modelo 3D publicado como parámetro en ***robot_description***. El plugin *robot_model* es activado dentro de la clase de RViz *render_panel* la cual es incorporada dentro del widget que espera el escenario 3D.

```
robot_state_pub_->publishFixedTransforms("robot_editor");
robot_editor.cpp
```

Los nodos y tópicos de la interfaz y su interacción bajo ROS son mostrados en la Figura 2.51.

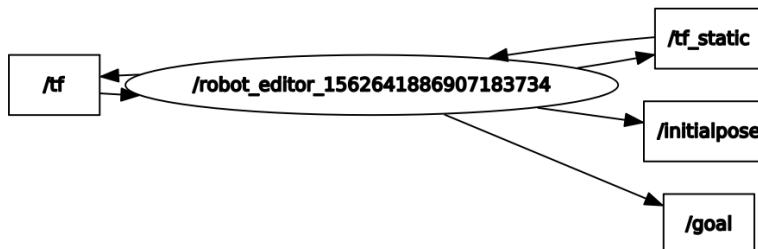


Figura 2.51: Nodos y Tópicos en la Interfaz interactuando bajo ROS

El resultado del desarrollo de esta interfaz es mostrado en la figura 2.52 en la cual podemos apreciar que se cuenta con una ventana de edición con autocompletado y resaltado de las palabras reservadas (highlight), un conjunto de widgets mediante los cuales el robot se mueve a la posición correspondiente de acuerdo con los ángulos de los joints ingresados por el estudiante, el panel de visualización del robot en 3D y un conjunto de herramientas para control de la ventana. Esta ventana es un nodo de ROS que es lanzado desde la aplicación principal tal y como fue definido en la sección 2.3.1.3 “Ejecución de Procesos de ROS”.

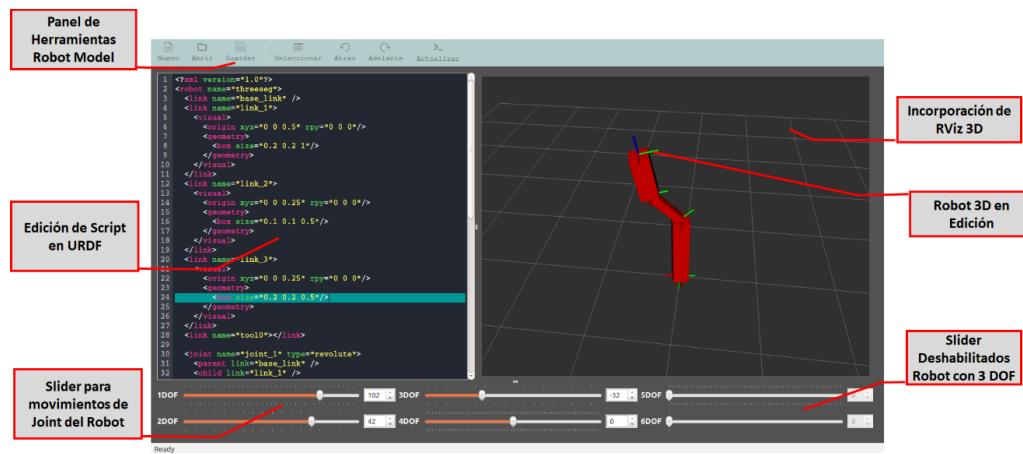


Figura 2.52: Interfaz Modelación de Robots.

En la Figura 2.53 se puede observar el resultado cuando se alteran las dimensiones de un robot cilíndrico en el archivo URDF correspondiente el cual ha sido cargado previamente en la interfaz de visualización.

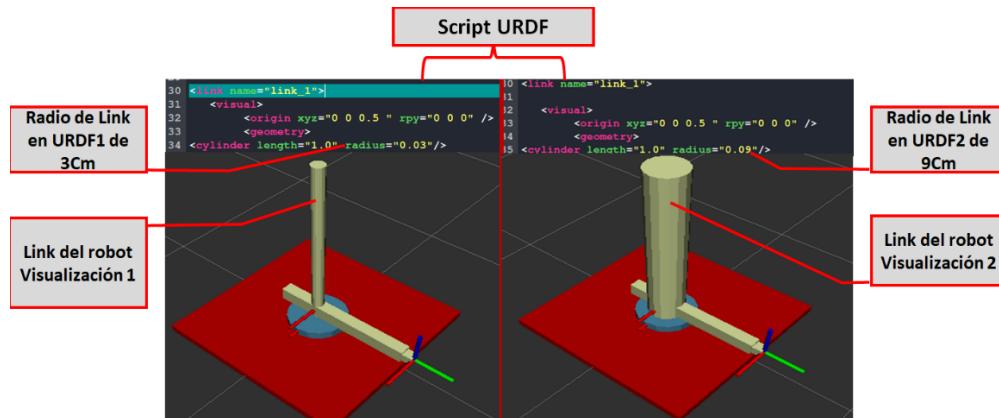


Figura 2.53: Ejemplo de la edición de los parámetros de un robot con un script URDF.

2.3.3 Implementación de la Interfaz de Matemática y Cinemática de Robots

El proceso de desarrollo de la interfaz relacionada con la matemática y la cinemática del robot y la visualización del modelo del robot fue realizado siguiendo el procedimiento indicado en la sección 2.3.2 “Implementación de la Interfaz Modelación de Robots”. En esta interfaz el estudiante deberá cargar un modelo de los existentes en el LVR, los cuales fueron creados por el desarrollador del trabajo monográfico u creados por los fabricantes (ABB, KUKA, MOTOMAN) según los paquetes de ROS industrial. Para garantizar la efectividad de la interfaz se incorporaron más plugins y librerías de ROS en C++.

2.3.3.1 Creación de Interfaz en Qt.

Para la implementación de esta interfaz fue necesario definir varios procesos de software en C++, con características de comportamiento estándar, de manejo de los diferentes modelos de robots cargados, intercambio de información de los nodos, configuración de los plugin de ROS, etc., donde la incorporación de los Widgets de Qt cobra importancia para hacer de la interfaz lo más transparente posible.

Cada widget utilizado posee señales y slots de funciones para la realización de una acción dentro del código, cada interacción con la GUI del LVR es usando estos recursos de Qt. (Ver Figura 2.54)

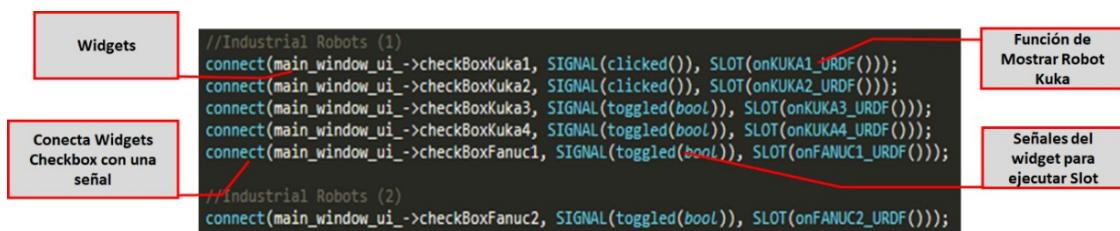


Figura 2.54: Señales y Slots con funciones dentro del código de la interfaz principal.

Las secciones de la interfaz desarrollados bajo los widgets de Qt son:

- 1) Visualizador 3D de robots virtuales
- 2) Selección de los robots dentro de la interfaz.
- 3) Análisis de la Matemática del robot.

- 4) Comprobación de Cinemática Directa.
- 5) Comprobación de Cinemática Inversa.
- 6) Modelar robot con Algoritmo de Denavit Hartenberg
- 7) Gráficos de los movimientos de Joints.

2.3.3.2 Programación de los recursos de ROS

Cada subinterfaz en Qt posee nodos, servicios o vinculación de plugin de RViz que permiten mostrar de forma visual la información de robótica industrial necesaria para este LVR.

2.3.3.2.1 Visualizador 3D de robots virtuales

Para visualizar el robot se sigue el mismo algoritmo, plasmado en la Figura 2.50, de la interfaz de modelación de robots. También se configuran las herramientas del visualizador RViz como: zoom, selección de sólidos, herramienta de medición del espacio 3D, y se agrega el plugin de Reuleaux para el análisis del espacio de trabajo. La interfaz provee al estudiante una serie de widgets que le permitirán usar las herramientas de RViz y activar o desactivar el plugin de Reuleaux para que se muestre el espacio de trabajo del robot.

Al agregar el robot virtual en el ambiente 3D, como se plantea en la sección 2.3.2 “Implementación de la Interfaz Modelación de Robots”, se puede proceder a agregar más funcionalidades al visualizador 3D. Se agregan las librerías de RViz para la programación de las herramientas del visualizador 3D como se observa en la Figura 2.55. Se programa un puntero en C++, de la clase de RViz, llamado **ToolManager** que administra la activación de las herramientas dentro del panel 3D.

El diagrama ilustra la ejecución de un código C++ que crea un puntero a la clase **ToolManager**. El código es el siguiente:

```

76 // Create an interaction tool...
77 tm=new rviz::ToolManager(manager_);
78 tm = manager_->getToolManager();
79 interactTool_ = tm->addTool("rviz/Interact");
80 pointTool_ = tm->addTool("rviz/PublishPoint");
81 measureTool_ = tm->addTool("rviz/Measure");
82 selectTool_ = tm->addTool("rviz>Select");
83 moveCamera_ = tm->addTool("rviz/MoveCamera");
84 focusCamera_ = tm->addTool("rviz/FocusCamera");

```

Un cuadro rojo rodea la línea 77 y tiene la etiqueta "Herramientas agregadas al panel 3D de RViz". Un cuadro rojo rodea la línea 78 y tiene la etiqueta "Creación de Puntero de la Clase ToolManager". Una flecha roja apunta desde el cuadro de la línea 78 hacia el cuadro de la línea 77.

Figura 2.55: Herramientas del panel 3D de RViz

Las herramientas dentro del panel 3D facilitan el estudio del modelo del robot cargado (ver Figura 2.56). Por ejemplo, al cargar el modelo de un robot industrial se puede utilizar la herramienta de medición para conocer o confirmar las dimensiones de este. Otro ejemplo es la posibilidad de ver el robot bajo la herramienta TF permitiendo ver el robot sin los archivos CAD cargados. Esto es útil para estudiar el robot y conocer los puntos de intersección de los Joints y eslabones.

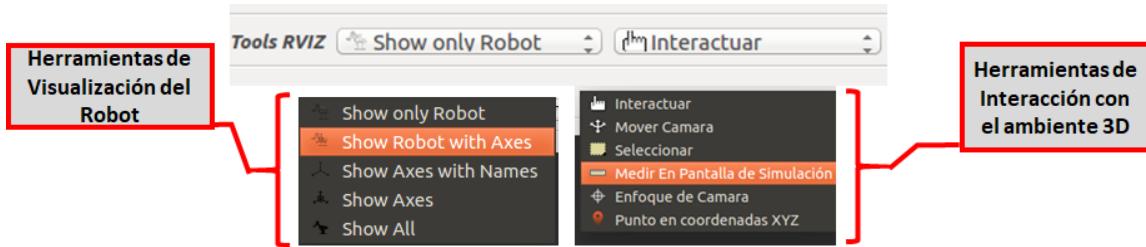


Figura 2.56: Herramientas de Visualización

Otro plugin dentro de la visualización del ambiente en RViz es **Reuleaux** (ver sección 2.1.6, Figura 2.18: Proceso de ROS para gestionar la Visualización del espacio de trabajo”). En la sección mencionada se describe la obtención de los datos del espacio de trabajo de cada robot dentro del ambiente 3D cargado en la interfaz de matemática y cinemática.

Reuleaux es un plugin de propósito general, diseñado particularmente para RViz, y este fue configurado para incorporarlo al laboratorio virtual con el objetivo de que el estudiante pueda utilizarlo de forma transparente, ver el código de la configuración en la Figura 2.57. Se incorpora el plugin dentro del visualizador del LVR y se añaden herramientas que pueden ser utilizadas por medio de Widgets de Qt para desactivar o activar el espacio de trabajo, o cambiar el índice de alcance del robot dentro de este.

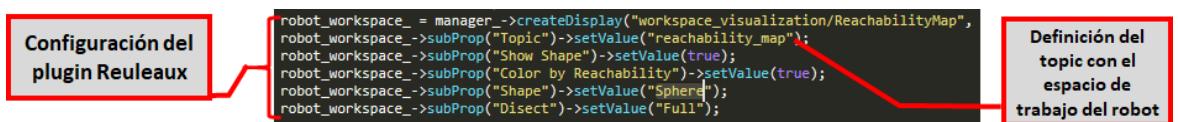


Figura 2.57: Configuración del plugin Reuleaux

Cada robot posee su espacio de trabajo característico por lo cual usando los archivos en C++ generados con IKFast del módulo de OpenRAVE (2.1.5 Open Rave IkFast), se genera vía las nodos de Reuleaux archivos H5 (Arreglo de datos en forma jerárquica) que guardan todas las posiciones posibles de la cinemática especificada en el archivo en C++ de IKFast. En la Figura 2.58 se muestra el proceso de creación del archivo H5.

```

[Nodo en Reuleaux genera el archivo H5 con IKFast]
[You can start planning now!]
[INFO] [1547158469.023995825]: Ready to take commands for planning group manipulator.
[INFO] [1547158469.039168693]: Initial workspace has 108 spheres and 5400 poses
[INFO] [1547158469.241242846]: Ready to take commands for planning group manipulator.
[INFO] [1547158469.241976356]: Processing sphere: 1 / 108
[INFO] [1551330959.791118766]: Creating Directory
[INFO] [1551330959.791295544]: Saving map cylindrical_manipulator_0.9_reachability.h5
[INFO] [1551330959.875315792]: Saving poses in reachability map
[INFO] [1551330959.875315792]: Saving poses in reachability map

```

Figura 2.58: Reuleaux generando archivo para mostrar espacio de trabajo.

En la Figura 2.59 se muestra, en el ambiente 3D, el espacio de trabajo de un robot KUKA, gestionado por el plugin Reuleaux al cargar esferas según las coordenadas en el archivo H5. Los colores de las esferas dependen del índice de alcance del robot. Azul indica múltiples poses por parte del robot de alcanzar la posición, Verde y amarillo con pocas poses y rojo una pose que se alcanza por el robot solo con al menos 2 posiciones posibles.

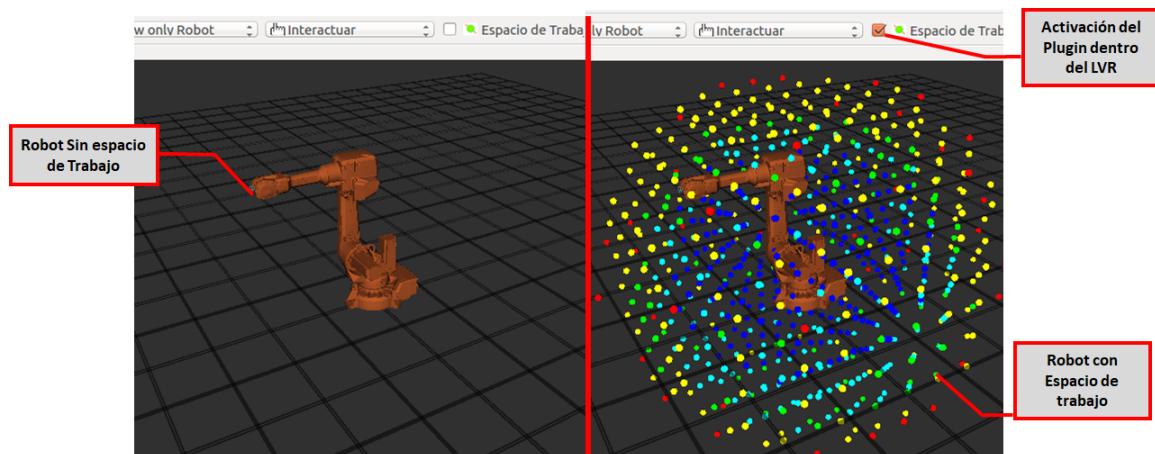
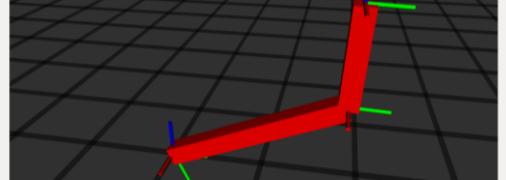
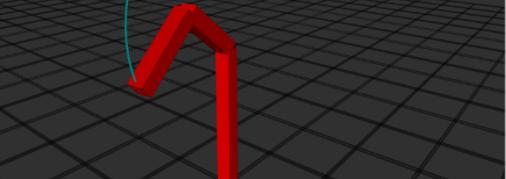
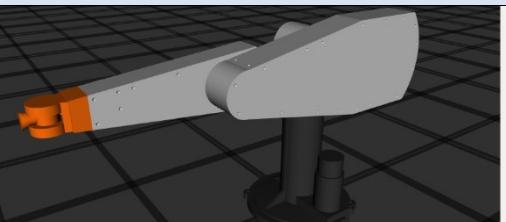
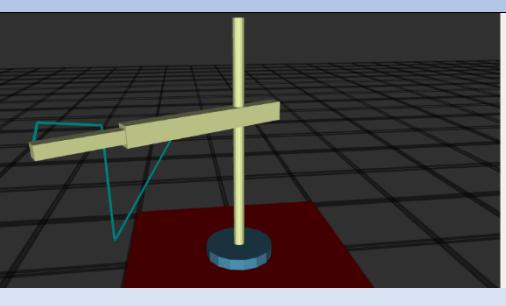


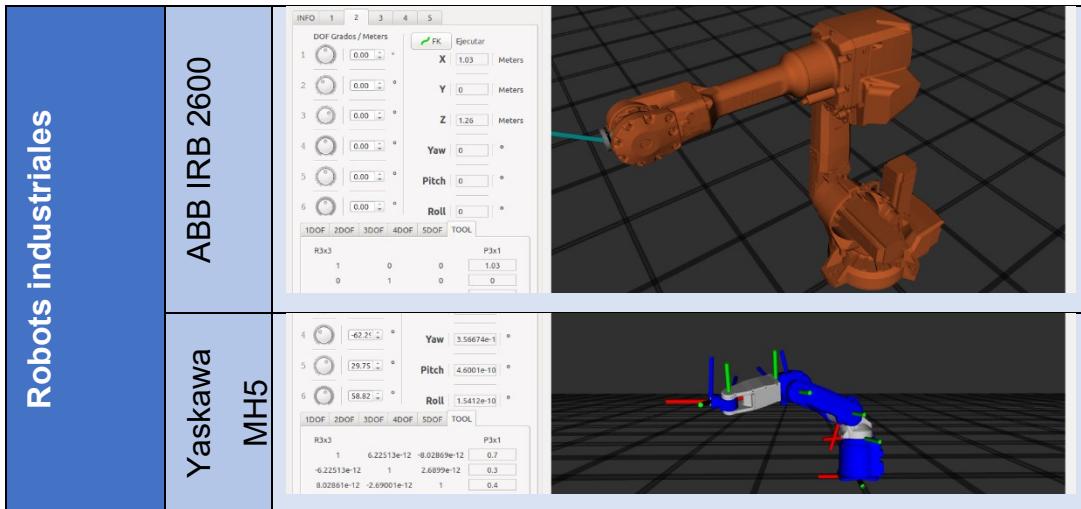
Figura 2.59: Espacio de Trabajo de un Robot Kuka.

2.3.3.2.2 Selección de los robots dentro de la interfaz.

Como se menciona en la sección 2.2.1 “Modelos de los robots”, en este laboratorio virtual hay modelos elaborados por el desarrollador del LVR y modelos de fabricantes de robots, todos modelados en URDF, los cuales son cargados de forma transparente siguiendo diferentes algoritmos en C++ utilizados para actualizar los plugins de visualización del robot, espacio de trabajo y cálculos de la cinemática para cada robot tal como se muestra en la Tabla 2.10.

Tabla 2.10: Robots seleccionados dentro de la interfaz

Robots Clásicos	Robots Simples	<table border="1"> <thead> <tr> <th colspan="2">2 grados de libertad</th> </tr> </thead> <tbody> <tr> <td> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 </td><td> <input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0 Meters Y: 0.937955 Meters Z: 0.890427 Meters Yaw: 0 ° Pitch: 0 ° </td></tr> </tbody> </table>  <table border="1"> <thead> <tr> <th colspan="2">3 grados de libertad</th> </tr> </thead> <tbody> <tr> <td> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 </td><td> <input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO Z: 0.986631 Meters Yaw: 7 ° Pitch: 0 ° Roll: 131 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1 0.992546 0.121869 0 0.0933431 0.0799535 -0.651169 0.75471 -0.760219 0.091476 -0.749084 -0.156659 0.486611 </td></tr> </tbody> </table> 	2 grados de libertad		<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0 Meters Y: 0.937955 Meters Z: 0.890427 Meters Yaw: 0 ° Pitch: 0 °	3 grados de libertad		<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO Z: 0.986631 Meters Yaw: 7 ° Pitch: 0 ° Roll: 131 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1 0.992546 0.121869 0 0.0933431 0.0799535 -0.651169 0.75471 -0.760219 0.091476 -0.749084 -0.156659 0.486611
2 grados de libertad										
<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0 Meters Y: 0.937955 Meters Z: 0.890427 Meters Yaw: 0 ° Pitch: 0 °									
3 grados de libertad										
<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO Z: 0.986631 Meters Yaw: 7 ° Pitch: 0 ° Roll: 131 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1 0.992546 0.121869 0 0.0933431 0.0799535 -0.651169 0.75471 -0.760219 0.091476 -0.749084 -0.156659 0.486611									
Puma 560	<table border="1"> <thead> <tr> <th colspan="2">3 grados de libertad</th> </tr> </thead> <tbody> <tr> <td> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 </td><td> <input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0.86389 Meters Y: 0.12954 Meters Z: 0.62357 Meters Yaw: -2.80557e-1 ° Pitch: 2.80554e-1 ° Roll: 7.62587e-3 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1 </td></tr> </tbody> </table> 	3 grados de libertad		<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0.86389 Meters Y: 0.12954 Meters Z: 0.62357 Meters Yaw: -2.80557e-1 ° Pitch: 2.80554e-1 ° Roll: 7.62587e-3 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1					
3 grados de libertad										
<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: 0.86389 Meters Y: 0.12954 Meters Z: 0.62357 Meters Yaw: -2.80557e-1 ° Pitch: 2.80554e-1 ° Roll: 7.62587e-3 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1									
Cilíndrico	<table border="1"> <thead> <tr> <th colspan="2">3 grados de libertad</th> </tr> </thead> <tbody> <tr> <td> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 </td><td> <input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: -0.385107 Meters Y: 0.694751 Meters Z: 0.6666012 Meters Yaw: 29 ° Pitch: 0 ° Roll: 0 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1 </td></tr> </tbody> </table> 	3 grados de libertad		<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: -0.385107 Meters Y: 0.694751 Meters Z: 0.6666012 Meters Yaw: 29 ° Pitch: 0 ° Roll: 0 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1					
3 grados de libertad										
<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6	<input checked="" type="checkbox"/> EJECUTAR MOVIMIENTO X: -0.385107 Meters Y: 0.694751 Meters Z: 0.6666012 Meters Yaw: 29 ° Pitch: 0 ° Roll: 0 ° 1DOF 2DOF 3DOF 4DOF 5DOF TOOL R3x3 P3x1									



2.3.3.2.3 Análisis de la Matemática del robot.

En esta subinterfaz del laboratorio virtual de robótica el estudiante podrá utilizar los diferentes widgets y modelos 3D, cargados al plugin RViz, para comprobar las coordenadas de posición y orientación descritas en la Tabla 2.6: Representación Matemática de Coordenadas de Posición y Orientación”. Para lograrlo se usan las librerías de TF para alterar la posición y orientación de la base del objeto a analizar dentro de la interfaz de Matemática y Cinemática como se muestra en la Figura 2.60

Aplicación de Ecuaciones en C++

```
//Cartesian
Z = rSph * cos(thSph);
Y = rSph * sin(thSph) * sin(phiSph);
X = rSph * sin(thSph) * cos(phiSph);

phiC = phiSph;
roC = rSph * sin(thSph);

odom_trans.transform.translation.x = X;
odom_trans.transform.translation.y = Y;
odom_trans.transform.translation.z = Z;

odom_trans.header.frame_id = "my_lab_world";
odom_trans.child_frame_id = "my_lab_world/base_link";
odom_trans.header.stamp = ros::Time::now();
broadcaster.sendTransform(odom_trans);
```

Puntero en C++ odom_trans

Publicación del Puntero para alterar la posición del objeto

Figura 2.60: Aplicación en C++ de la Matemática del Robot dentro del LVR

Se describen a continuación dos ejemplos de uso de la matemática del robot usando un Modelo de un cubo en 3D para observar la traslación de un objeto en el ambiente 3D y el Modelo de un avión en 3D para mostrar el ejemplo clásico de los ángulos de Euler.

Ejemplo 1

El cubo mostrado en la Figura 2.61 está ubicado en las coordenadas XYZ lo cual es mostrada en la Matriz de Transformación Homogénea junto con las transformaciones a coordenadas esféricas y cilíndricas correspondientes. El cubo es mostrado en el visualizador del LVR en la posición correspondiente.

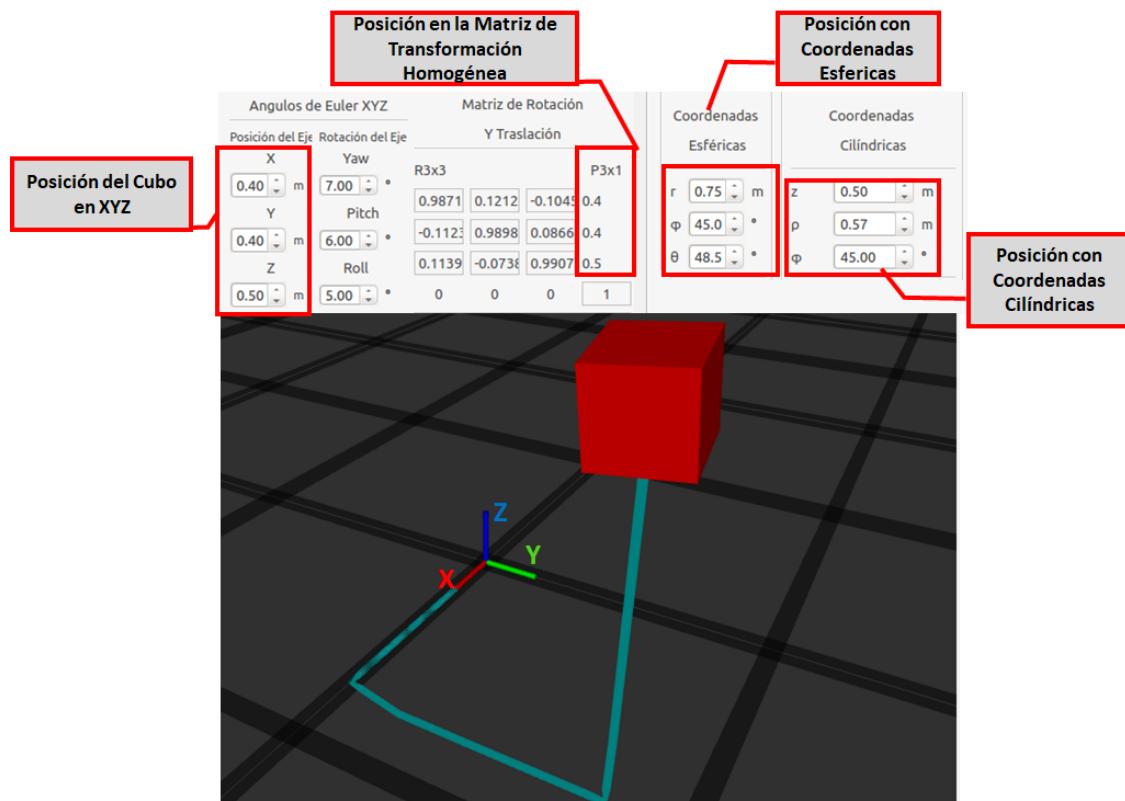


Figura 2.61: Posición del Modelo 3D

Ejemplo 2

En la Figura 2.62 se muestra el modelo del avión aplicando los ángulos de Euler y representando los datos respectivos con la matriz de transformación homogénea, así como con los cuaternios.

Los ejemplos presentados consideran aspectos relacionados con la posición y orientación de un objeto en el espacio los cuales son de mucha importancia en el estudio de la robótica donde es importante conocer la posición y orientación del efecto final.

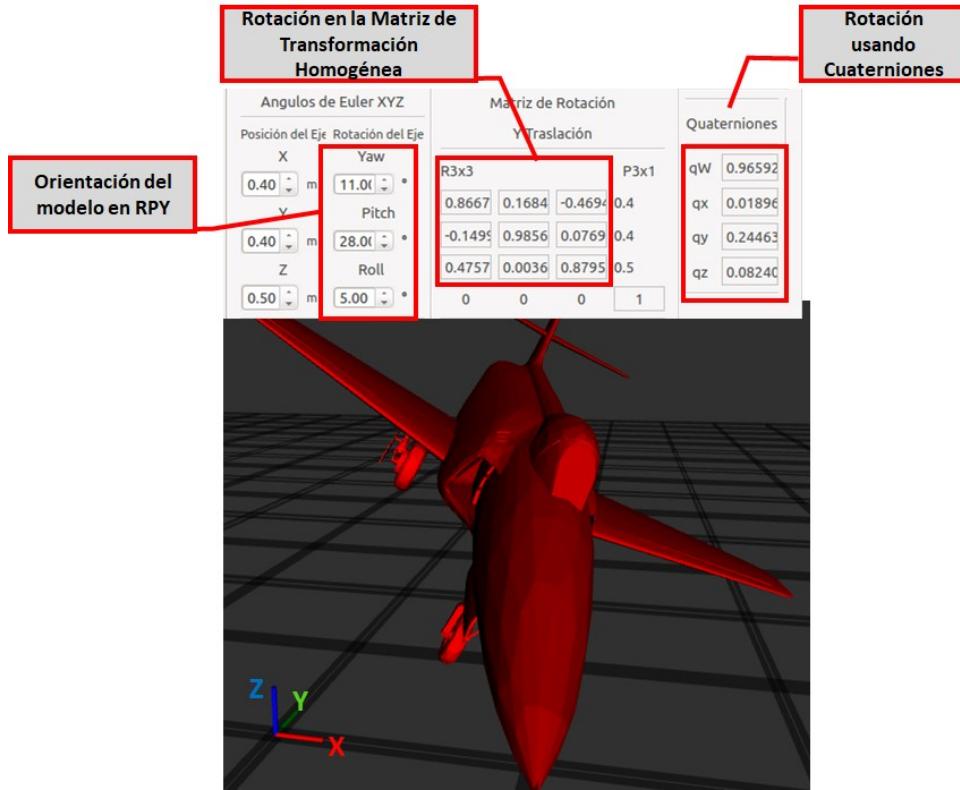


Figura 2. 62: Rotación del Modelo en 3D del Avión

2.3.3.2.4 Comprobación de Cinemática Directa

Para abordar el problema de la cinemática directa relacionada con un robot modelado usando URDF, en el LVR se utiliza la librería KDL. Se implementa la librería KDL para encontrar la cinemática directa del modelo cargado a la interfaz desde un archivo URDF. Para esto se ejecuta un parser del archivo URDF definiéndole a la librería KDL los límites de la cadena cinemática a analizar. Para todos los modelos se usan *base_link* y *Tool0*, se obtiene un objeto de la clase KDL_Chain del cual se obtiene un objeto con la solución de la cinemática y por último se somete este objeto anterior a diferentes vectores de ángulos para obtener la posición de la herramienta con la función *JntToCart* (Joints a Coordenadas cartesianas). (Ver Figura 2.63)

```

//> Parámetros base_link y Tool0 de la cadena Cinemática
//> Puntero Con archivo URDF en C++
//> objeto con la Cadena Cinemática
//> Datos de la Cinemática Directa

nh_.deleteParam("root_link");
nh_.deleteParam("tip_link");
nh_.setParam("root_link","base_link");
nh_.setParam("tip_link","tool0");

/// Get KDL TREE
if (!kdl_parser::treeFromUrdfModel(urdf_model, kdl_tree)){
    ROS_ERROR("Failed to construct kdl tree");
    nh.shutdown();
}

/// GET KDL CHAIN
if(!kdl_tree.getChain("root_link", "tip_link", kdl_chain6))
{
    ROS_ERROR("Error getting KDL chain");
    nh.shutdown();
}

fkSolver6= new KDL::ChainFkSolverPos_recursive(kdl_chain6);

double roll = 0.0 , pitch = 0.0 , yaw = 0.0;
fkSolver6->JntToCart(q6, result6,njnt);

```

Figura 2.63: Obtención de los datos de Cinemática directa con KDL.

En la parte izquierda de la Figura 2.64 se encuentran los widgets mediante los cuales el estudiante ingresa los valores de los ángulos de cada joint y a la derecha los resultados obtenidos para las coordenadas XYZ (posición del efecto final). En el centro el modelo del robot utilizado, MH5 de Motoman, ubicado en la posición correspondiente con los ángulos ingresados por el estudiante. También se muestran los ángulos de Euler RPY, demostrando así que con la interfaz de matemática y cinemática es posible analizar diferentes tipos de robots industriales.

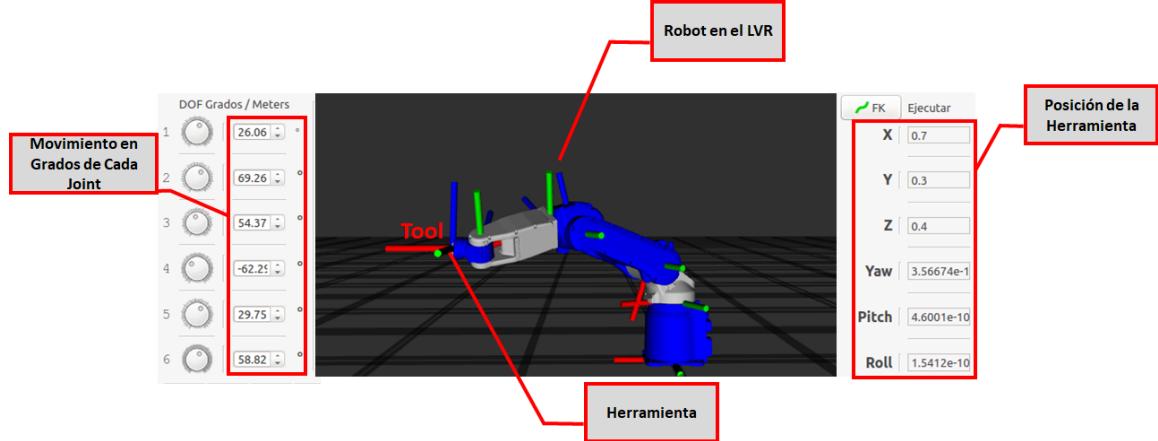


Figura 2.64: Visualización del robot y su cinemática directa respectiva.

2.3.3.2.5 Comprobación de Cinemática Inversa.

Al igual que en el caso de la cinemática directa, para abordar el problema de la cinemática inversa se utiliza la librería KDL para obtener los ángulos en que deben posicionarse los diferentes joints del robot para unas coordenadas XYZ dadas del

efector final. Se sigue el mismo algoritmo de la cinemática directa utilizando otra clase para describir la cinemática inversa utilizando el modelo matemático de Newton Raphson, tal como se plantea en la sección 2.1.4 “Librería Orocó”, el cual es una optimización no-lineal, basado en funciones de algoritmos. Para ejecutar el proceso con la librería KDL es necesario pasarle como argumento la cadena cinemática bajo consideración, así como el **solver** del algoritmo de la cinemática directa y un solver para la cinemática inversa en velocidad el cual será utilizado en el método de Newton-Raphson de KDL. (Ver Figura 2.65)

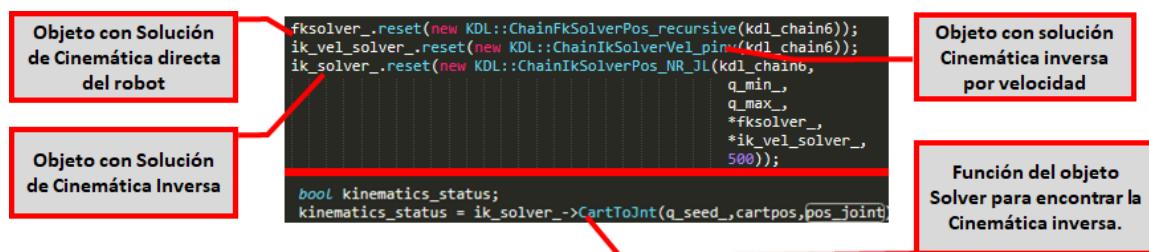


Figura 2.65: Obtención de los datos de Cinemática inversa con KDL.

En la Figura 2.66 se muestran los widgets para el ingreso de las posiciones en XYZ y ángulos RPY en que queremos situar la herramienta para un modelo MH5 de Motoman del cual nos interesa conocer la posición final de los ángulos de cada Joint el robot.

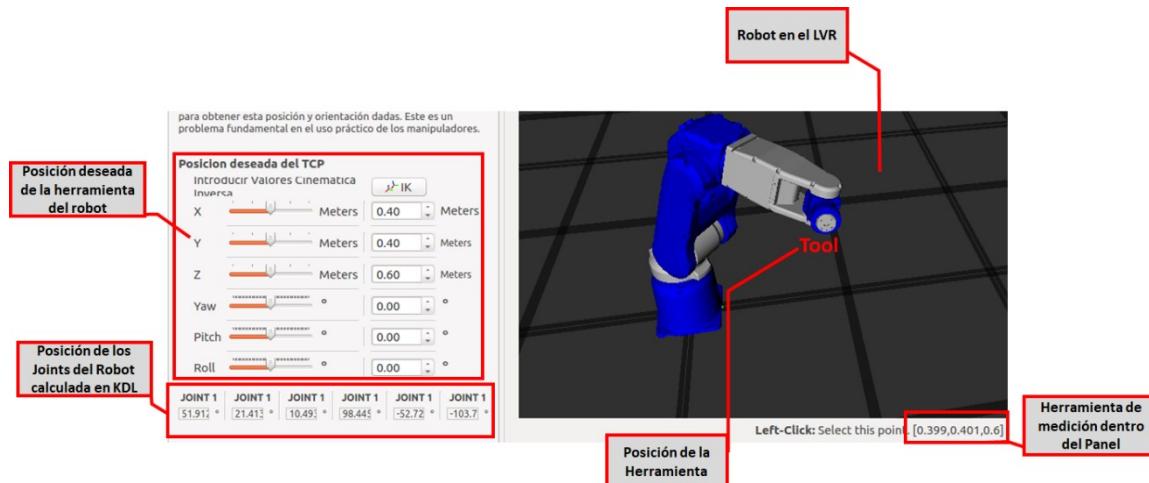


Figura 2.66: Visualización del robot y su cinemática inversa respectiva.

El laboratorio virtual de robótica suministra una interfaz transparente que permite la realización de prácticas relacionadas tanto con la cinemática directo como con la inversa utilizando cualquiera de los modelos proporcionados por este. Es una interfaz apropiada para el estudio de las diferentes morfologías de los robots.

2.3.3.2.6 Algoritmo de Denavit Hartenberg

La relación que existe entre dos eslabones contiguos se puede establecer haciendo uso de cualquier sistema de referencia ligado a cada elemento. Sin embargo, la forma habitual de hacerlo es utilizando la representación de Denavit-Hartenberg (D-H) la cual permite reducir el número de grados de libertad. El estudio del algoritmo de D-H es importante en cualquier curso de robótica básica y el laboratorio virtual de robótica cuenta con los recursos para verificar los resultados de aplicarlo. Lo anterior se logra mediante la librería KDL con la cual se asocian los datos ingresados por el usuario mediante la utilización de widgets tal y como se aprecia en la Figura 2.67. El modelo obtenido se muestra en la Figura 2.68.

```

KDL::Chain chain_dh_robot;
chain_dh_robot.addSegment(KDL::Segment("base_link",
chain_dh_robot.addSegment(KDL::Segment("link_1",
chain_dh_robot.addSegment(KDL::Segment("link_2",
chain_dh_robot.addSegment(KDL::Segment("link_3",
Joint(Joint::None),RotBase));
Joint("joint_1",Joint::RotZ);
Joint("joint_2",Joint::RotZ),
Joint("joint_3",Joint::RotZ),
Frame::DH(DH.data[0], DH.data[1]/ToG, DH.data[2], DH.data[3]/ToG));
Frame::DH(DH.data[4], DH.data[5]/ToG, DH.data[6], DH.data[7]/ToG));
Frame::DH(DH.data[8], DH.data[9]/ToG, DH.data[10], DH.data[11]/ToG));

```

Annotations:

- Objeto con Cadena a definir**: Points to the variable `chain_dh_robot`.
- Cadena Con robot y sus elementos**: Points to the segments added to the chain: `base_link`, `link_1`, `link_2`, and `link_3`.
- Robot de 3 Joints, 3 Grados de Libertad.**: Points to the joints defined: `Joint(Joint::None), RotBase)`, `Joint("joint_1", Joint::RotZ)`, `Joint("joint_2", Joint::RotZ)`, and `Joint("joint_3", Joint::RotZ)`.
- Valores de Denavit Hartenberg**: Points to the DH matrix values: `DH.data[0]` through `DH.data[11]`.
- Joint de la cadena Cinemática**: Points to the joints listed in the DH matrix row: `Joint::DH(DH.data[0], DH.data[1]/ToG, DH.data[2], DH.data[3]/ToG))`, `Joint::DH(DH.data[4], DH.data[5]/ToG, DH.data[6], DH.data[7]/ToG))`, and `Joint::DH(DH.data[8], DH.data[9]/ToG, DH.data[10], DH.data[11]/ToG))`.

Figura 2.67: Programación en C++ con ingreso de datos de Denavit – Hartenberg

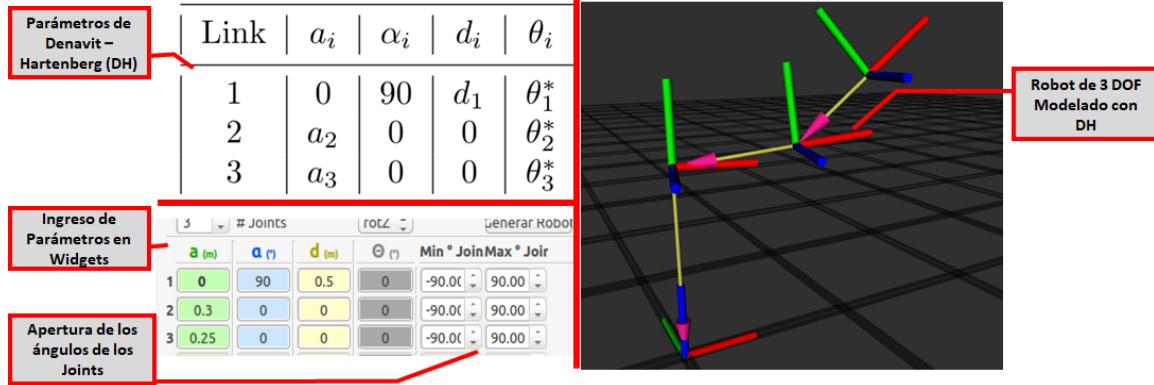


Figura 2.68: Modelos de Robot Generado con DH.

Mediante la subinterfaz suministrada por el LVR es posible modelar robots, usando los parámetros de D-H, de hasta seis grados de libertad, lo cual cumple con los fundamentos de robótica que indican que un robot de más de 6 grados de libertad se vuelve redundante y no óptimo. En la Figura 2.69 se muestra el robot IRB 120 de ABB de 6 grados de libertad modelado con DH.

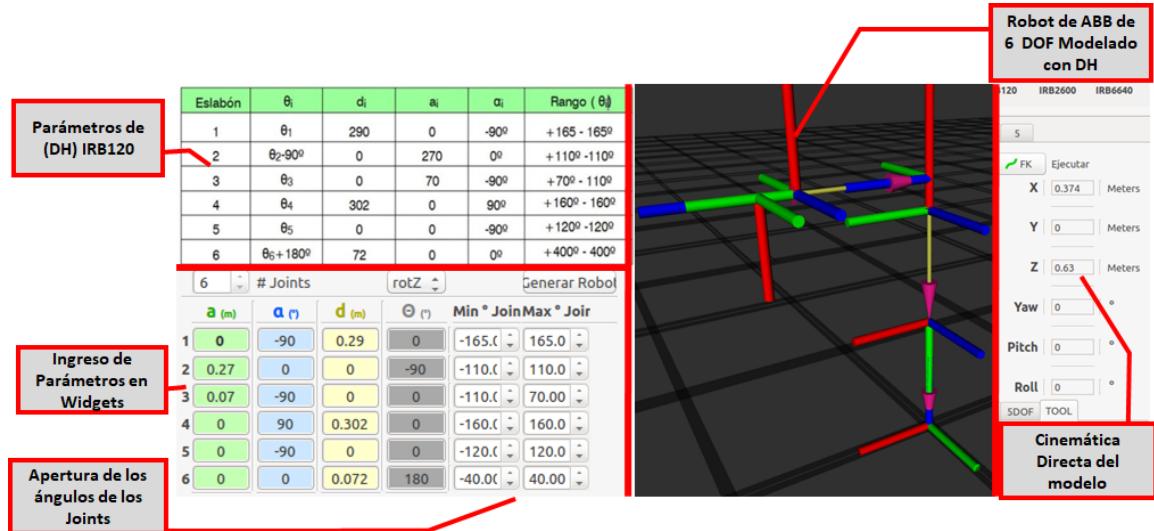


Figura 2.69: Modelo de robot ABB IRB 120 con DH.

2.3.3.2.7 Gráficos de los movimientos de Joints.

La interfaz correspondiente a la matemática y cinemática del robot presenta la posibilidad de visualizar gráficamente la posición angular de cada joint del modelo de robot bajo consideración. Lo anterior se logra mediante el uso de la librería

QCustomplot (ver sección 2.1.7) para la obtención de los gráficos. En la figura 2.70 se observan los gráficos de los datos de los tópicos de ROS con la información de los joints.

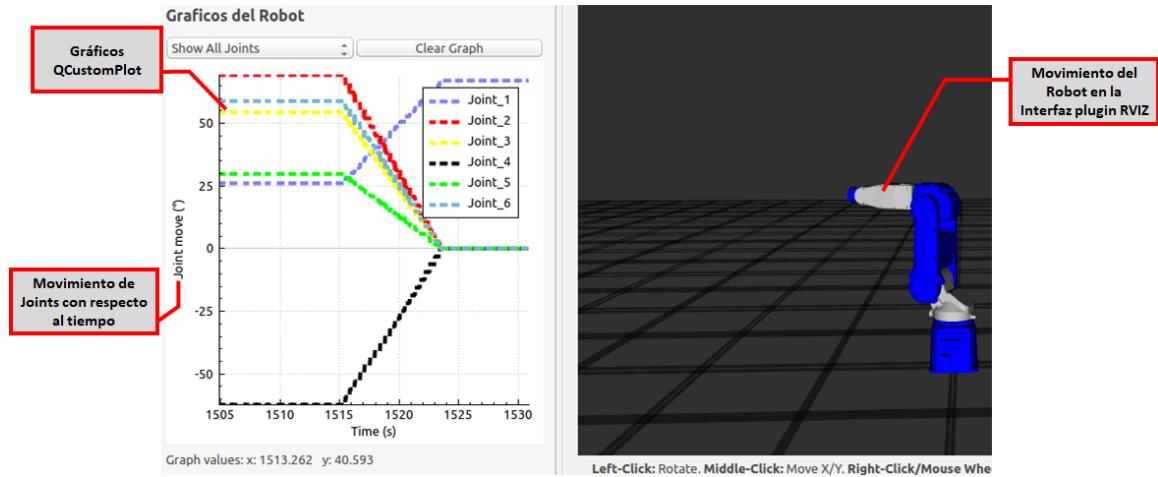


Figura 2.70: Gráficos del movimiento del robot.

En la figura 2.71 se muestran los nodos y publicadores de ROS, en el laboratorio virtual de robótica, correspondientes a la interfaz de matemática y cinemática del robot

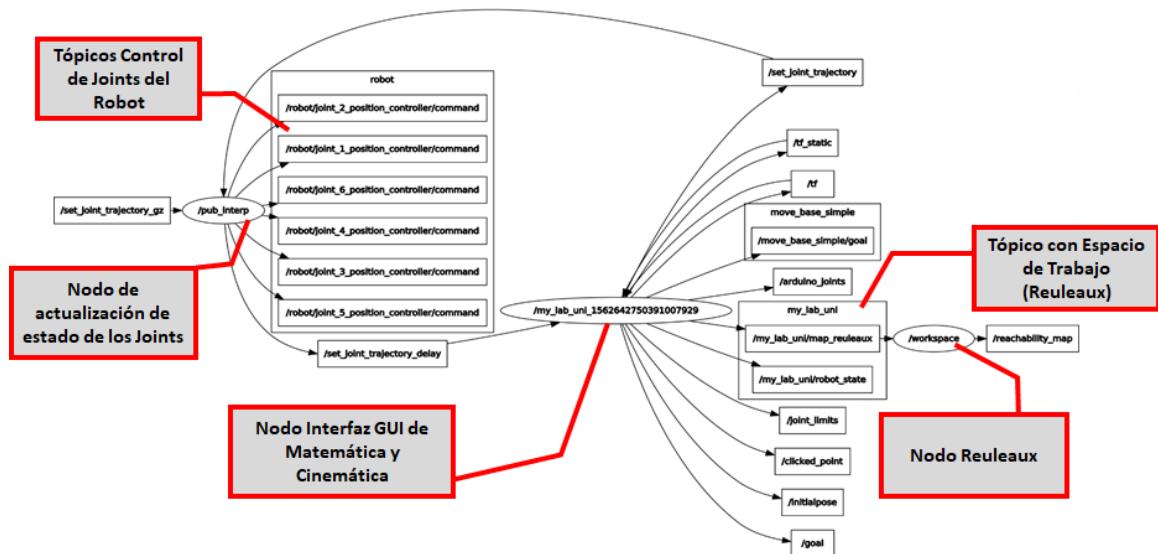


Figura 2.71: Nodos de ROS de la interfaz de Matemática y Cinemática.

2.3.4 Implementación del módulo de Programación de los Robots.

El módulo de programación, para verificar su efectividad, requiere de la creación de una celda de trabajo virtual, así como de una interfaz para la interpretación de las instrucciones requeridas para programar la tarea del robot.

2.3.4.1 Implementación de la Celda de Trabajo con Gazebo

Un laboratorio físico de robótica cuenta, entre otras cosas, con un robot el cual, según su estructura, define un espacio de trabajo en el cual interactuará con diferentes objetos. En general, podemos decir que el robot junto con los otros elementos, requeridos para realizar una tarea dada, conforman una celda de trabajo. Para simular la celda de trabajo se utilizó el simulador Gazebo.

La descripción del mundo en Gazebo se realiza usando SDF mediante el cual se define la física, los plugins y el modelo del robot con sus parámetros físicos. En la figura 2.72 se muestra el modelo de la celda de trabajo simulada.

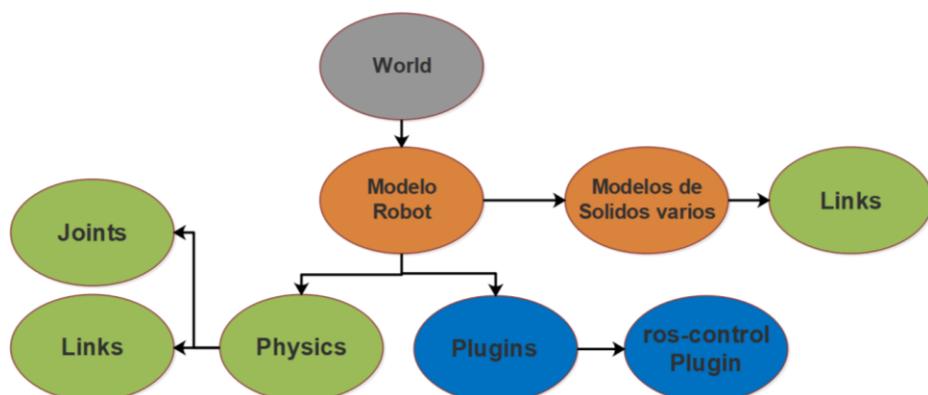


Figura 2.72: Modelo de la celda de Trabajo.

Mediante el archivo Launch se lanza el servicio de Gazebo, ver figura 2.73, el cual ejecuta los nodos correspondientes para cargar la celda de trabajo. También se lanzan los procesos de Gazebo cliente que muestra la interfaz de usuario con la simulación del robot.

2.3.4.1.1 Plugin ros_control

Un aspecto fundamental es el sistema para controlar el modelo del robot que será simulado. Dicho sistema de control se define utilizando el plugin **ros_control**. En el modelo del robot debe indicarse el tipo de actuador de cada joint lo cual es definido con la etiqueta *transmission* en el archivo SDF.

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/MYROBOT</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>

  <transmission name="${prefix}tran1">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="${prefix}joint_1">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor1">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>
</gazebo>
```

xml WORLD Gazebo

El script anterior muestra parte del archivo WORLD de Gazebo que debe ser configurado para agregar el plugin de **ros_control** al mundo, así como el tipo de actuador de los joints del robot.

El archivo launch mostrado en la Figura 2.73 muestra los parámetros para lanzar los nodos en ROS y el servidor de Gazebo, cada vez que se carga un modelo, este posee valores de SetPoint iniciales y ganancias del PID del plugin **ros_control** que hace que el robot se mueva levemente buscando el SetPoint.

The diagram shows a terminal window displaying a launch file. Three red boxes with arrows point to specific parts of the code:

- A box labeled "Carga del modelo IRB120 a la celda de trabajo" points to the line: `<include file="$(find gazebo_ros)/launch/empty_world.launch">`
- A box labeled "Archivo a Lanzar con el mundo de Gazebo." points to the line: `<arg name="world_name" value ="$(find robot_gz)/world/Workcell.world" />`
- A box labeled "Archivo con configuraciones del Plugin ros_control." points to the line: `<param name="robot_description" command="$(find xacro)/xacro.py '$(find irb120_description)/urdf/irb120.xacro'" />`

```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
<arg name="world_name" value ="$(find robot_gz)/world/Workcell.world" />
<arg name="debug" value = "$(arg debug)" />
<arg name="gui" value = "$(arg gui)" />
<arg name="paused" value = "$(arg paused)" />
<arg name="use_sim_time" value = "$(arg use_sim_time)" />
<arg name="headless" value = "$(arg headless)" />
<arg name="verbose" value = "$(arg verbose)" />
</include>
<param name="robot_description" command="$(find xacro)/xacro.py '$(find irb120_description)/urdf/irb120.xacro'" />
<include file="$(find robot_gz)/launch/irb120_control.launch"/>
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"/>
```

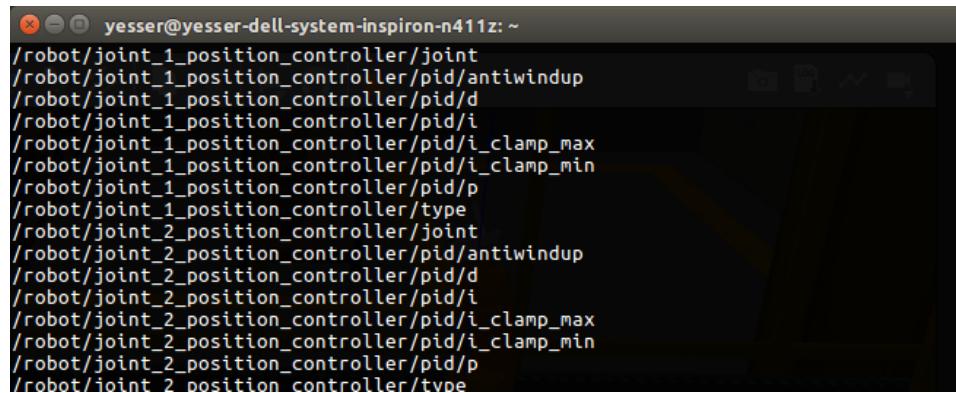
Figura 2.73: Archivo *launch* lanza los procesos para la carga de la celda de Trabajo.

Para mover cada articulación se añade un controlador de ROS, el cual está indicado en la etiqueta *transmission*. Este controlador consiste principalmente de un mecanismo retroalimentado (normalmente un bucle PID), el cual recibe una referencia y controla la salida usando la retroalimentación de los actuadores. El controlador de ROS se comunica con la interfaz de hardware. La función principal de la interfaz de hardware es actuar como sistema mediador entre los controladores de ROS y el hardware real o el simulador. De este modo, el mismo controlador y código puede accionar un robot simulado o un robot real (ver Figura 2.36). Para definir los controladores se define un archivo de tipo yaml (que es un estándar de archivos con un formato específico), que posee los parámetros del PID cargado a cada joint del robot.

```
robot:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 100
  joint_1_position_controller:
    type: position_controllers/JointPositionController
    joint: joint_1
    pid: {p: 10.0, i: 0.01, d: 0.1}
```

Archivo YAML

Cuando se carga la celda de trabajo se pueden observar, vía el comando **ROSService**, los diferentes servicios para el control del Robot (ver figura 2.74). Tales servicios son de importancia para comunicar la celda de trabajo con la interfaz de programación del robot.



A terminal window titled 'yesser@yesser-dell-system-inspiron-n411z: ~' displays a list of ROS services. The services listed are related to the robot's joints, specifically joint_1 and joint_2. The services include joint controllers for both joints, along with their respective PID parameters (antiwindup, p, i, d) and clamp values (i_clamp_max, i_clamp_min).

```
yesser@yesser-dell-system-inspiron-n411z: ~
/robot/joint_1_position_controller/joint
/robot/joint_1_position_controller/pid/antiwindup
/robot/joint_1_position_controller/pid/d
/robot/joint_1_position_controller/pid/i
/robot/joint_1_position_controller/pid/i_clamp_max
/robot/joint_1_position_controller/pid/i_clamp_min
/robot/joint_1_position_controller/pid/p
/robot/joint_1_position_controller/type
/robot/joint_2_position_controller/joint
/robot/joint_2_position_controller/pid/antiwindup
/robot/joint_2_position_controller/pid/d
/robot/joint_2_position_controller/pid/i
/robot/joint_2_position_controller/pid/i_clamp_max
/robot/joint_2_position_controller/pid/i_clamp_min
/robot/joint_2_position_controller/pid/p
/robot/joint_2_position_controller/type
```

Figura 2.74: Servicios de configuración del PID de los Joints del robot.

Una vez que Gazebo y el plugin de ros_control son puestos en marcha, se puede agregar el SetPoint para cada Joint con su respectivo PID de control. En la figura 2.75 se muestra la celda del trabajo con un robot el cual será controlado mediante un programa elaborado desde la interfaz de programación, descrita en el siguiente capítulo.

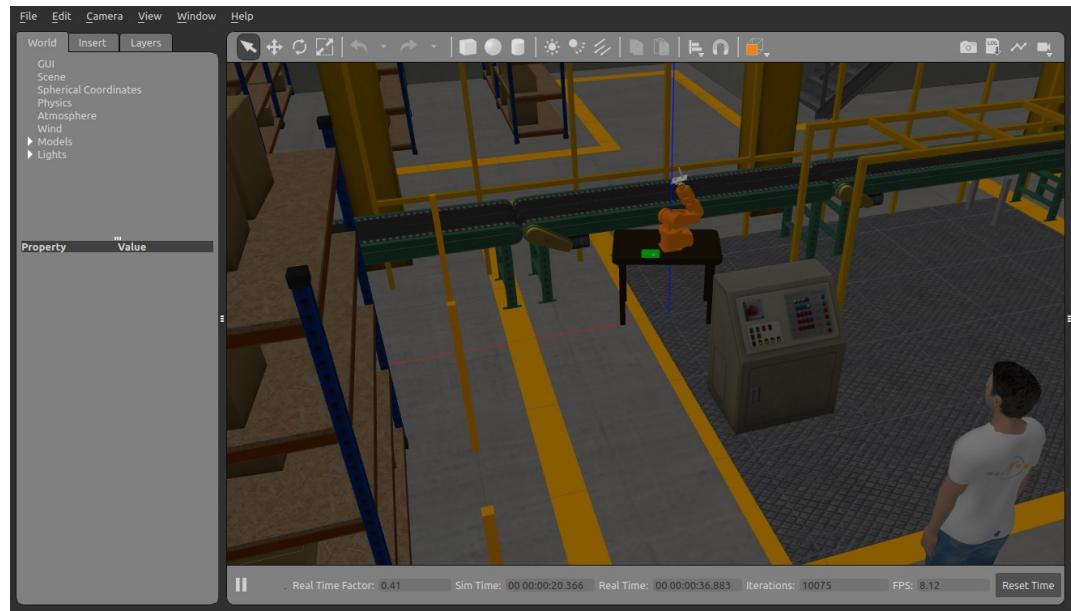


Figura 2.75: Celda de Trabajo del robot.

2.3.4.2 Implementación de la Interfaz de Programación del Robot.

La interfaz de programación fue desarrollada utilizando widgets de Qt Creator y librerías de control de texto plano mediante las cuales el usuario define las instrucciones las cuales deben ser procesadas y traducidos a comandos para el controlador PID de cada joint del robot.

Por ejemplo, el comando

`Move_J1 G 50 W 800`

Script LVR

indica una rotación del Joint 1 (Move_J1) de 50 grados positivos (G 50) y con un tiempo de ejecución de 800ms (W 800). La interfaz de programación acepta un máximo de 6 Joints (Move_J1, Move_J2, Move_J3... Move_J6.) y cada comando en grados es un SetPoint para el PID.

Los valores de las ganancias de los PIDs (KP, KD, KI) se ingresan vía widgets dentro de la interfaz de programación, estos valores luego son pasados a los servicios de configuración del PID de cada Joint a parametrizar (ver Figura 2.76)

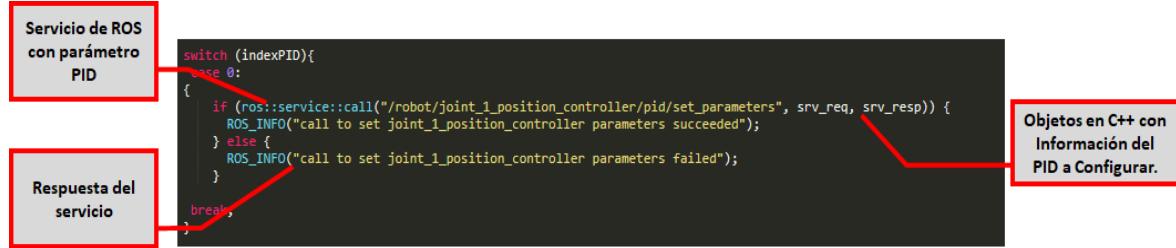


Figura 2.76: Servicios de configuración de ros_control de Cada PID del Robot.

Cada Joint del robot posee un tópico dentro del plugin ros_control que describe la posición en grados, esta información es usada para graficar la respuesta del controlador PID.

En la Figura 2.77 se pueden apreciar las gráficas del movimiento de los joints en respuesta al control aplicado al robot ABB IRB120 por medio de la interfaz de programación. A la izquierda se aprecia las instrucciones para el movimiento de cada joint. También se pueden apreciar los widgets asociados con el controlador PID.

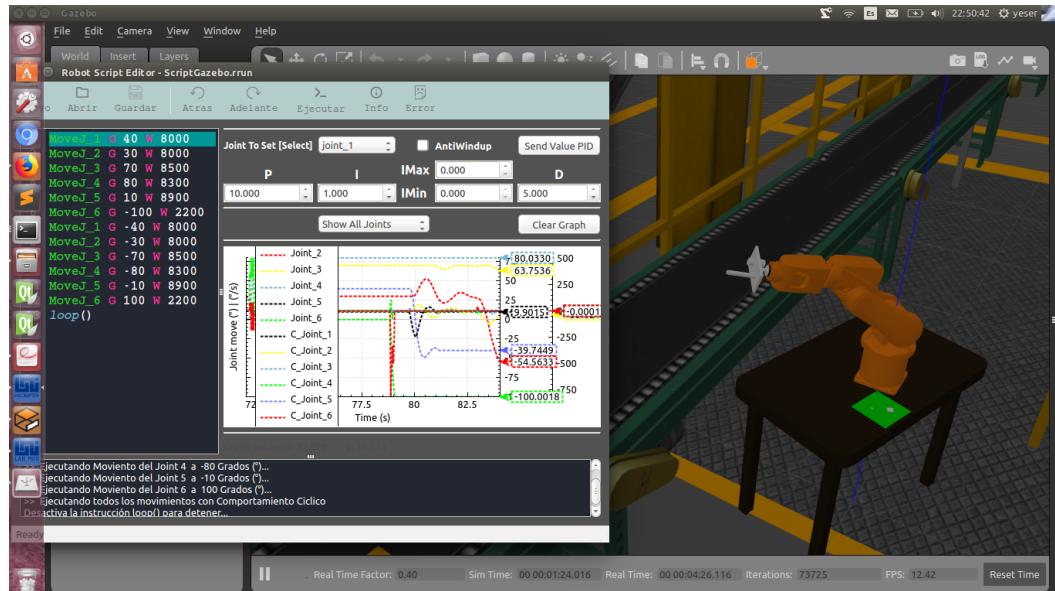


Figura 2.77: Robot ABB IRB120 simulado en Gazebo y controlado con Interfaz de Programación

En la figura 2.78 se muestran los nodos asociados al simulador Gazebo y a la interfaz de programación.

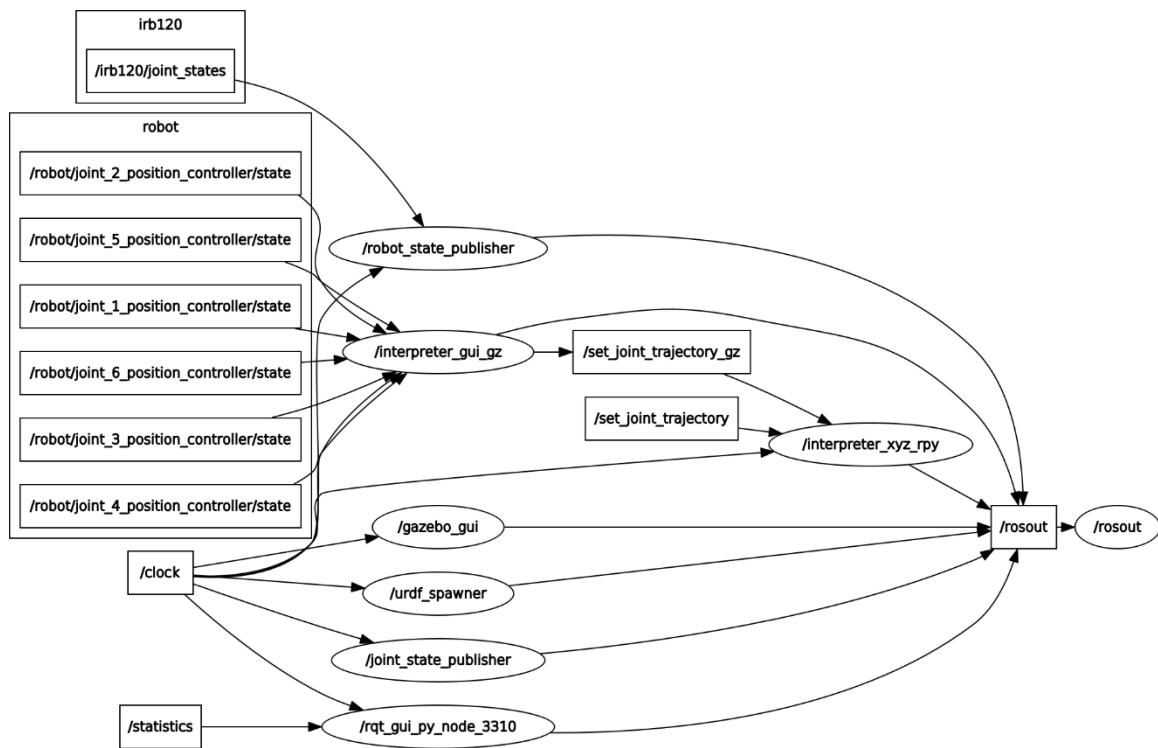


Figura 2.78: Nodos de Gazebo e interfaz de Programación.

CAPÍTULO III: ANÁLISIS DE RESULTADOS

El trabajo monográfico produjo como resultado un laboratorio virtual para robótica industrial el cual consta de varias interfaces las cuales fueron implementadas utilizando ROS, QT y herramientas, plugins y librerías asociadas. Un aspecto a destacar es la puesta en marcha del simulador Gazebo el cual permite verificar diferentes aspectos relacionados con la robótica desde una celda de trabajo considerando muchos de los elementos presentes en el mundo de un robot real. A continuación, se describen algunas actividades, y sus resultados, relacionadas con los cuatro elementos que integran los recursos del laboratorio virtual de robótica como son morfología del robot, herramientas matemáticas, cinemática del robot y programación del robot.

Morfología del robot:

Los robots manipuladores son cadenas cinemáticas de diferentes tipos de joints (rotación, prismáticos) y en el laboratorio virtual el estudiante podrá modelar, desde la interfaz de modelación de robots, usando el lenguaje descriptivo URDF robots con diferentes tipos joints. De igual forma, el estudiante podrá, desde la interfaz de matemática y cinemática, cargar diferentes modelos de robots, propuestos por el desarrollador de este trabajo, para el análisis de la morfología de estos.

En el libro de Barrientos et al (2007), se deja claro que los robots manipuladores son cadenas cinemáticas de diferentes tipos de Joints por lo tanto en la interfaz de Modelación de Robots el estudiante es capaz de modelar un robot usando el lenguaje descriptivo de URDF usando diferentes Joints. Además, en el LVR en la interfaz de Matemática y Cinemática el estudiante es capaz de interactuar con diferentes tipos de modelos de Robots ya cargados previamente por el desarrollador y hacer el análisis respectivo de su morfología.

En la figura 3.1 se puede apreciar los modelos desarrollados en el laboratorio virtual para diferentes morfologías de robots.

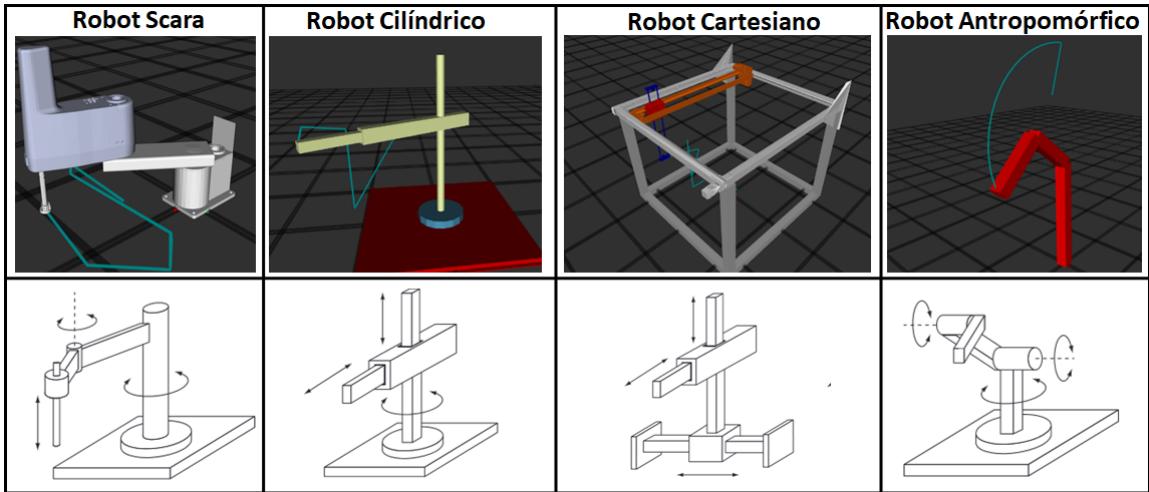


Figura 3.1: Robots con diferentes Morfologías en el LVR

Modelos de robots en ambiente propietario y en el LVR

Para validar la efectividad del laboratorio virtual para representar el modelo de un robot se realizó una comparación de robots en el simulador del fabricante, modelo IRB2600 y el IRB120 de ABB en este caso, y en el LVR. Los joints de los robots fueron ajustados para dos configuraciones diferentes, en el simulador de ABB y en el LVR desarrollado. En ambos casos la configuración de los robots y la posición de sus efectores finales fue prácticamente el mismo.

Los ángulos de los Joints y su diferencia son mostrados en la Tabla 3.1 y 3.2, así como sus poses en las Figuras 3.4 – 3.5.

Tabla 3.1: Validación de Modelo ABB IRB2600

Posiciones de Robot en LVR vs Robot en ABB Robot Studio					
Joints	LVR y Robot Studio	Coordenadas	Posición del TCP en LVR	Posición del TCP en Robot Studio	Error de Posición del TCP
	Angulo en Grados		Distancia en Metros	Distancia en Metros	Error en Metros
Pose 1 Robot IRB2600					
1	-82.61	X	0.228112	0.22815	0.000038
2	35.43				
3	-0.14	Y	-1.2978	-1.29784	0.00004
4	83.09				
5	44.64	Z	0.609155	0.60912	0.000035
6	160.0				
Pose 2 Robot IRB2600					
1	84.35	X	-0.0200323	-0.02004	0.0000077
2	-66.01				
3	39.28	Y	0.229933	0.22983	0.000103
4	-71.50				
5	-31.88	Z	1.23508	1.23515	0.00007
6	160.0				

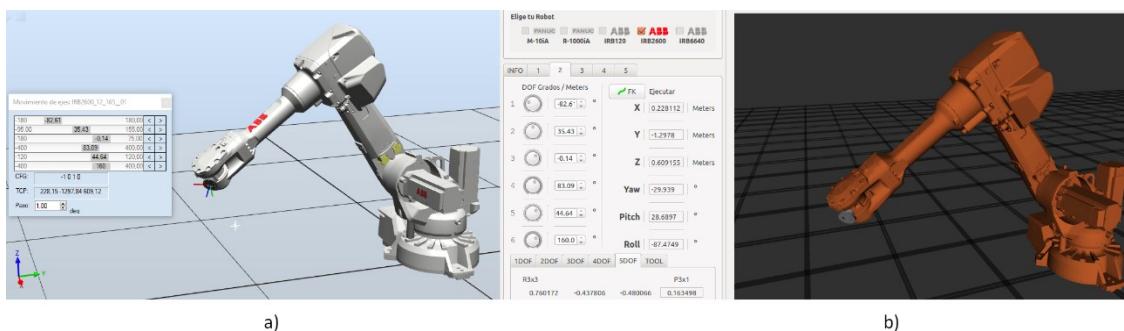
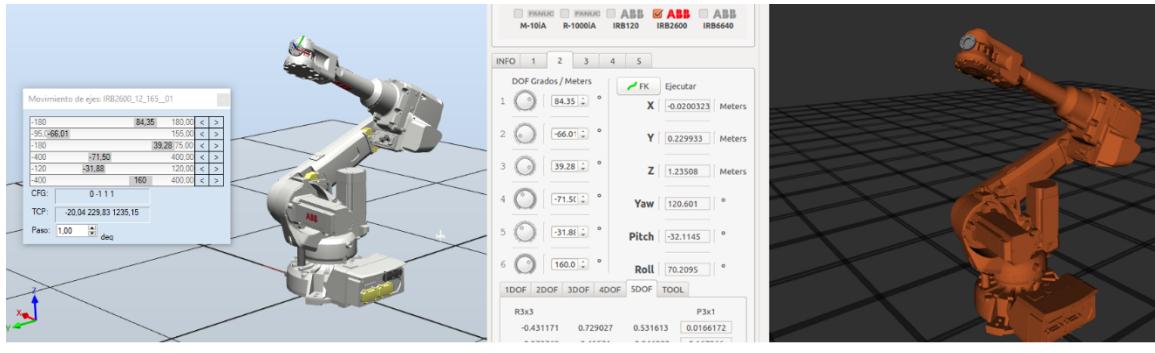


Figura 3.2: Pose 1 del Robot ABB IRB2600 a) Robot Studio b) Visualizador Plugin RVIZ



a)

b)

Figura 3.3: Pose 2 del Robot ABB IRB2600 a) Robot Studio b) Visualizador Plugin RVIZ

Tabla 3.2: Validación de Modelo ABB IRB120

Posiciones de Robot en ROS Contra Robot en ABB Robot Studio					
Joints	LVR y Robot Studio	Coordenadas	Posición del TCP en LVR	Posición del TCP en Robot Studio	Error de Posición del TCP
			Angulo en Grados	Distancia en Metros	Error en Metros
Pose 1 Robot IRB120					
1	104.00	X	0.0677014	0.0702	0.0024986
2	22.85				
3	46.04	Y	0.340683	0.34018	0.000503
4	16.23				
5	-56.23	Z	0.265651	0.26566	0.000009
6	0.00				
Pose 2 Robot IRB120					
1	78.91	X	0.113571	0.1137	0.000129
2	49.42				
3	-90.00	Y	0.382627	0.38268	0.00053
4	-137.00				
5	50.43	Z	0.775921	0.77582	0.000101
6	0.00				

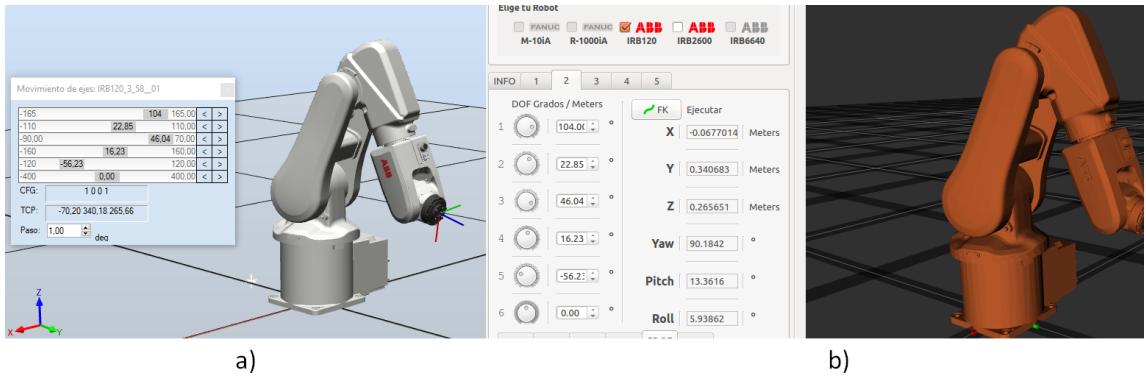


Figura 3.4: Pose 1 del Robot ABB IRB120 a) Robot Studio b) Visualizador Plugin RVIZ

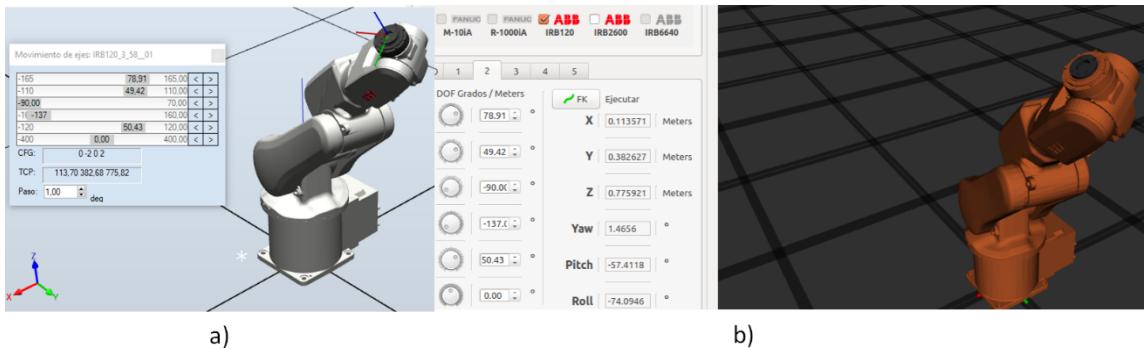


Figura 3.5: Pose 2 del Robot ABB IRB120 a) Robot Studio b) Visualizador Plugin RVIZ

La diferencia de posición del TCP listados en la tabla 3.1 y 3.2 son en el orden de decimas de milímetros, además se puede observar en las Figuras 3.4 – 3.5 las mismas posiciones en el ambiente 3D del Robot.

Herramientas Matemáticas:

Uno de los aspectos fundamentales en la robótica es el estudio para la posición y orientación de un robot, haciéndose necesario contar con herramientas que faciliten la comprensión de los diferentes elementos relacionados a dichos temas. El laboratorio virtual de robótica incorpora herramientas y una interfaz apropiada para realizar estudios básicos sobre la posición y orientación del efecto final del robot. Por ejemplo, en el LVR el estudiante será capaz de ajustar la posición y orientación de un sólido y observar su posición en el espacio. En la interfaz se podrán apreciar las conversiones correspondientes entre los diferentes sistemas de referencia.

En la figura 3.6 se muestra la trama $\{A\}$ la cual es la representación del eje de referencia base y la trama $\{B\}$ que es un punto de referencia que la herramienta del robot desea alcanzar. La posición se puede representar en el sistema de referencia de transformación homogénea, ángulos de Euler y Cuaternios.

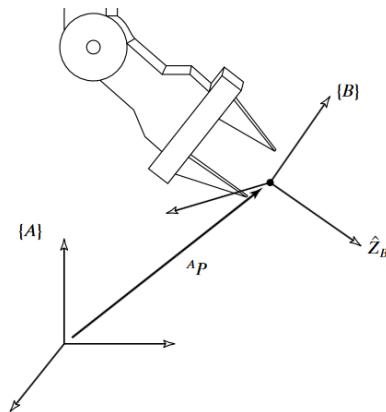


Figura 3.6: Representación de dos tramas

Para mostrar la funcionalidad del laboratorio virtual consideremos el siguiente ejercicio.

Ejercicio: Si la trama $\{B\}$ se gira 30 grados en forma relativa a la trama $\{A\}$ sobre el eje Z, se trasladada 1 m en X_A y 0.5 m en Y_A , encuentre la posición de la trama $\{B\}$.

Usando la interfaz de matemática y cinemática del laboratorio virtual obtenemos la representación visual (Ver figura 3.7) y los valores de los sistemas de referencia de este ejercicio (ver Tabla 3.3).

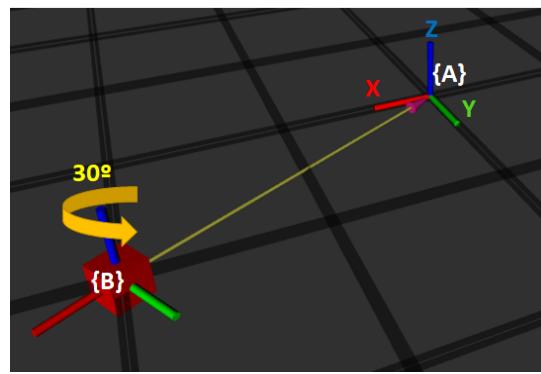


Figura 3.7: Trama $\{B\}$ Trasladada y rotada.

La interfaz de matemática y cinemática evita al estudiante el tedioso trabajo de resolver las matrices de rotación y traslación y a la vez le permite obtener una representación visual del robot y su efecto final.

Tabla 3. 3: Ejercicio de representación de los sistemas de referencia

Representación de la posición														
Trama {B} Trasladada en X = 1m, Y = 0.5m														
Coordenadas XYZ	Coordenadas Esféricicas	Coordenadas Cilíndricas												
Posición del Eje X <input type="text" value="1.00"/> m Y <input type="text" value="0.50"/> m Z <input type="text" value="0.00"/> m	Coordenadas Esféricicas r <input type="text" value="1.12"/> m φ <input type="text" value="26.5"/> ° θ <input type="text" value="90.0"/> °	Coordenadas Cilíndricas z <input type="text" value="0.00"/> m ρ <input type="text" value="1.12"/> m φ <input type="text" value="26.57"/> °												
Representación de la rotación														
Trama {B} rotada 30° sobre el eje Z														
Ángulos de Euler RPY	Matriz de Rotación	Representación de Cuaternios												
Rotación del Eje Roll <input type="text" value="0.00"/> ° Pitch <input type="text" value="0.00"/> ° Yaw <input type="text" value="30.00"/> °	R3x3 <table border="1"> <tr> <td>0.8660</td><td>0.5</td><td>0</td></tr> <tr> <td>-0.5</td><td>0.8660</td><td>0</td></tr> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> </table>	0.8660	0.5	0	-0.5	0.8660	0	0	0	1	0	0	0	Quaterniones qW <input type="text" value="0.96592"/> qx <input type="text" value="0"/> qy <input type="text" value="0"/> qz <input type="text" value="0.25881"/>
0.8660	0.5	0												
-0.5	0.8660	0												
0	0	1												
0	0	0												

Cinemática Directa e Inversa

Un aspecto fundamental en la robótica tiene que ver con la capacidad para poder determinar tanto la posición del efecto final de un robot dado los ángulos de los diferentes joints así como el valor que deben tener los Joints del robot para alcanzar una posición XYZ dada. Los problemas mencionados son conocidos como problema de la cinemática directa y problema de la cinemática inversa.

En el laboratorio virtual desarrollado es posible hacer análisis relacionados con la cinemática directa e inversa. Se pueden obtener los resultados obtenidos para cada caso independientemente del tipo de robot manipulador utilizado. Mediante el uso de la librería KDL se pueden mostrar los datos obtenidos para cada cadena cinemática tal y como se muestra en las figuras 2.64 y 2.66.

El algoritmo de Denavit – Hartenberg (DH) también es considerado un análisis cinemático ya que el estudiante puede describir usando los parámetros de DH cualquier cadena Cinemática con un máximo de 6 grados de libertad en el LVR. (Ver Figura 2.68 y 2.69).

Tomando como referencia el ejercicio 4.2 del Libro de Barrientos et al (2007) el cual muestra un robot de 3 grados de libertad (ver figura 3.8) con los parámetros de D-H, verificaremos la efectividad del laboratorio virtual. Los parámetros de D-H se introducen en el LVR utilizando los widgets de la interfaz tal como se muestra en la figura 3.9.

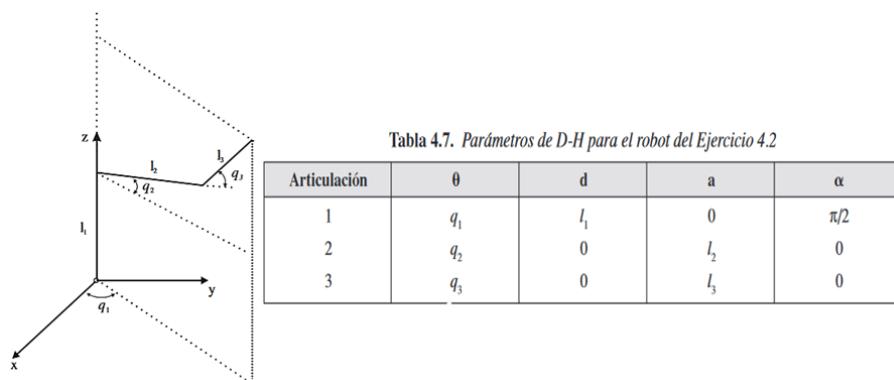


Figura 3.8: Ejercicio de DH En “Fundamentos de robótica” (P.173), por A. Barrientos et al, 2007

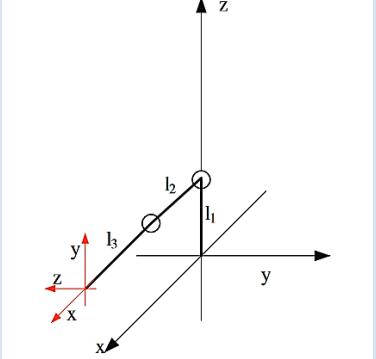
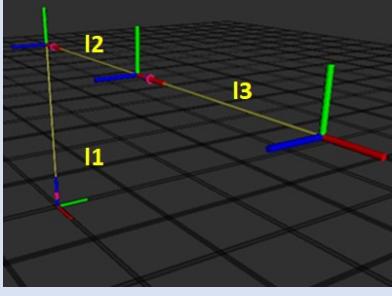
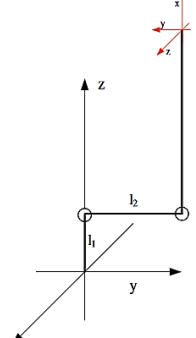
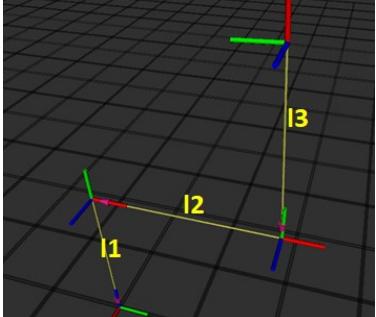
A los valores l_1 , l_2 , l_3 se le asigna un valor de 1 metro.

3 # Joints		rotZ	Generar Robot	
a (m)	d (m)	θ (°)	Min ° Jo	Max ° Jo
1 0	90	1	0	-180.0 180.0
2 1	0	0	0	-180.0 180.0
3 1	0	0	0	-180.0 180.0

Figura 3.9 Ingreso de los parámetros en Interfaz del LVR

Al presionar en el botón generar robot el modelo 3D del robot es mostrado permitiendo alterar los grados de los joints y seguir el ejercicio del libro y confirmar las poses del robot tal como se muestra en la tabla 3.2

Tabla 3.4 Confirmación de algoritmo DH del robot del ejercicio 4

Ángulos q1, q2, q3	Representación en 3D por el libro	Representación en 3D por el LVR
$q_1 = 0$, $q_2 = 0$, $q_3 = 0$		
$q_1 = 90$, $q_2 = 0$, $q_3 = 90$		

Programación del Robot

Para que el robot pueda ejecutar una tarea el mismo debe ser programado. En el libro de Barrientos et al (2007) destaca que “*Los escasos intentos de unificar en cierta medida los procedimientos de programación de robots no han tenido hasta la fecha el reconocimiento y la aceptación necesarios, encontrándose que cada fabricante ha desarrollado su método particular, válido únicamente para sus propios robots*” con el uso de ROS se pretende erradicar este problema dentro de la programación de robots industriales, anteriormente en el capítulo **¡Error! No se encuentra el origen de la referencia.** se menciona el soporte de ROS – Industrial el cual ha tenido resultados dentro de la comunidad de profesionales de la robótica al programar robots industriales físicos. Por lo tanto, en este LVR se puede programar usando la misma interfaz de programación diferentes modelos simulados en Gazebo, siendo digno de recalcar que al usar el plugin de *ros_control* se puede conectar a un robot real sin ningún problema a como se describe en la Figura 2.36.

Normalmente al programar un robot se usan scripts de programación como Rapid de ABB, en esta interfaz de LVR se usan scripts que describen los comandos de acción para el robot (ver capítulo **¡Error! No se encuentra el origen de la referencia.**).

Un aspecto importante del LVR es que desde la misma interfaz y con las mismas instrucciones se pueden programar robots de diferentes fabricantes y para demostrar la efectividad del laboratorio virtual en el aspecto de la programación se procedió conectar un robot físico de 5 grados de libertad y se programó por medio de la interfaz de programación el robot simulado.

Se le programó una rutina que alcanzara un objeto y lo depositara en un recipiente para tal actividad se le programó 10 comandos incluyendo la activación del comando *loop()* que permite dejar el robot en un comportamiento cíclico ejecutando la rutina. (Ver Figura 3.10)

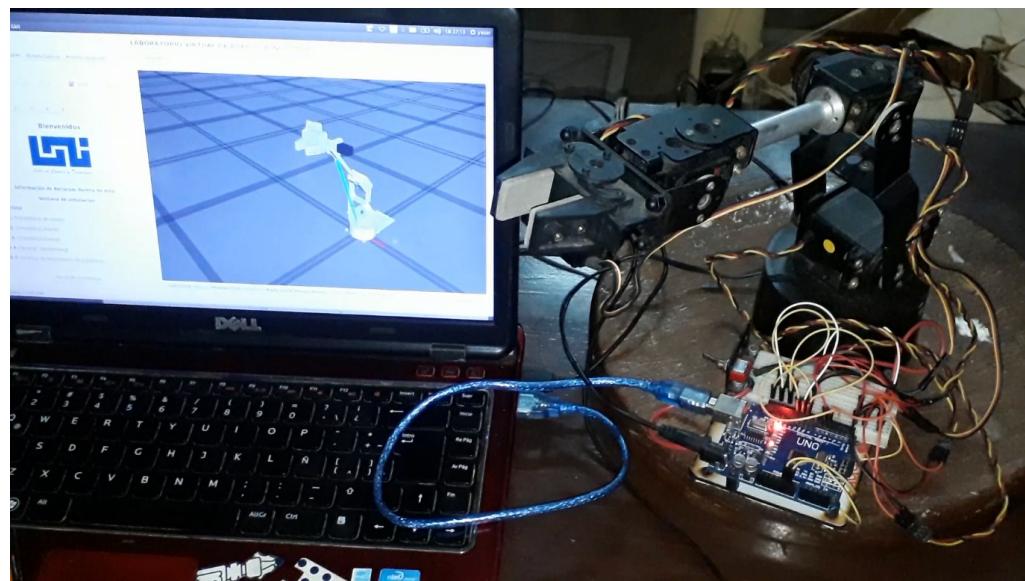


Figura 3.10: Aplicación del Robot en Simulación y robot en físico.

CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

En el trabajo monográfico desarrollado fue diseñado e implementado, con el objetivo de mostrar la potencialidad de ROS, un laboratorio virtual de robótica industrial, usando el middleware ROS. El LVR puede ser utilizado como recurso didáctico en el proceso enseñanza-aprendizaje en temas relacionados con los fundamentos de la robótica industrial.

El laboratorio virtual de robótica cuenta con los recursos necesarios para realizar actividades relacionadas con la morfología de los robots, herramientas matemáticas, cinemática directa e inversa, así como con la programación de robots. El laboratorio fue diseñado de forma tal que fuese lo más transparente posible para el usuario.

En el capítulo anterior fueron mostradas algunas de las posibilidades del LVR, realizando una demostración para cada uno de los componentes. Los resultados mostrados demuestran la funcionalidad y efectividad del laboratorio.

A continuación, son presentadas conclusiones sobre los objetivos específicos establecidos para lograr el objetivo general.

- El haber identificado plenamente los recursos de software de ROS y sus herramientas llevó al diseño e implementación exitosa del laboratorio virtual de robótica industrial.
- La selección adecuada de las herramientas de software requeridas para complementar a ROS, tal como el IDE de programación Qt Creator, posibilitaron el desarrollo de todas las interfaces de usuario del LVR.
- Las diferentes interfaces de usuario como son la interfaz principal, la interfaz para la modelación de robots, la interfaz de matemática y

cinemática, y la interfaz de programación y simulación de robots fueron desarrolladas exitosamente.

- El laboratorio virtual funciona satisfactoriamente lo cual fue verificado con los resultados presentados en el capítulo anterior, así como con los resultados obtenidos al realizar las dos prácticas que se muestran en los anexos.
- Se desarrollo un manual de uso del LVR donde se describen los recursos y características del laboratorio.

RECOMENDACIONES

A continuación, se presentan algunas recomendaciones que pueden contribuir a mejorar las prestaciones del laboratorio virtual desarrollado en este proyecto.

- Incrementar los recursos del laboratorio teniendo en cuenta para ellos que la comunidad hace que ROS evolucione constantemente mediante la incorporación de nuevas herramientas de software. Lo anterior conllevaría a que el laboratorio pueda ser utilizado en otros temas de la robótica de mayor complejidad, como la dinámica de los robots, por ejemplo.
- La importancia de ROS ha llevado al desarrollo de una versión en Windows 10 y es recomendable migrar a Windows, dada la amplia aceptación de este, cuando sea liberada una versión oficial de ROS.
- Usar Protocolo WEB para la ejecución de las prácticas de laboratorio usando la versión de prueba de RViz Web y Gazebo WEB. Lo anterior permitirá que el LVR corra sin problemas de compatibilidad por el sistema operativo. En la Figura 4.1 se muestra la arquitectura de software implementada en un trabajo con ROS conectado a un sistema Web.

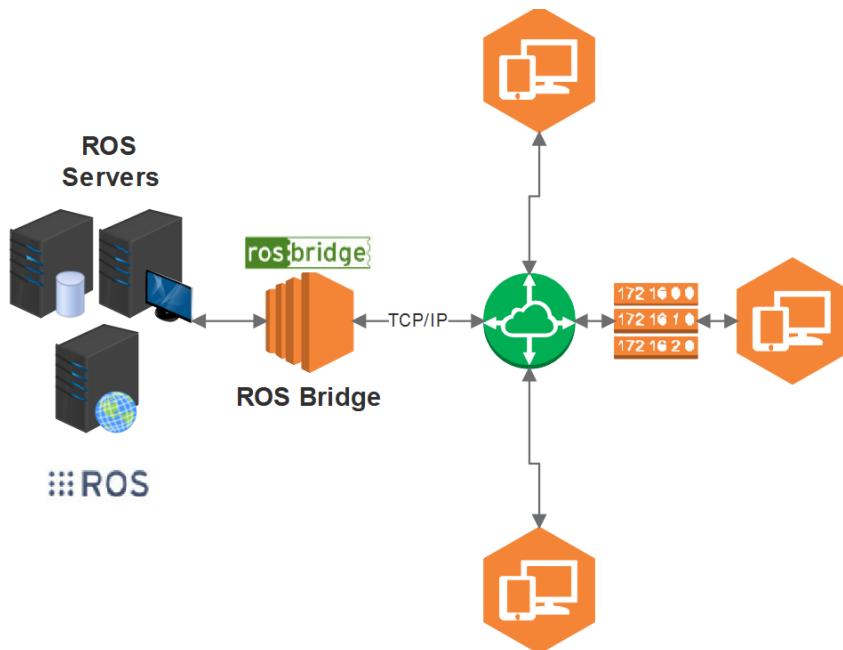


Figura 4. 1 ROS y Sistemas WEB

Bibliografía

- Barrientos, A., Peñín, L. F., Balaguer, C., Aracil, R. (2007). Fundamentos de robótica. Madrid, España. McGraw-Hill/Interamericana de España, S. A. U.
- Craig, J. J. (2006). Robótica, México. Pearson Educación de México, S.A. de C.V.
- Jara, C., et al. (2011). Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory. *Computers & Education*, 57(4), 2451–2461. Obtenido de <https://doi.org/10.1016/j.compedu.2011.07.003>
- Candelas, F., Torres, F., Gil, P., Ortiz, F., Puente, S., & Pomares, J. (2004). Laboratorio virtual remoto para robótica y evaluación de su impacto en la docencia. *RIAI: Revista Iberoamericana de Automática e Informática Industrial*, 1(2), 49–57. Obtenido de <http://rua.ua.es/dspace/handle/10045/4609>
- Castellanos, F., & Martínez, O. (2010). Laboratorios virtuales (LV) como apoyo a las prácticas a distancia y presenciales en Ingeniería. *INGE CUC*, 6(1), 267-280. Obtenido de <http://revistascientificas.cuc.edu.co/index.php/ingecuc/article/view/311>
- Ortega, J., Sánchez, R., González, J., & Reyes, G. (2016). Virtual laboratories for training in industrial robotics. *IEEE Latin America Transactions*, 14(2), 665-672. doi 10.1109/TLA.2016.7437208
- Makhal, A., & Goins, A. K. (2018, January). Reuleaux: Robot base placement by reachability analysis. En 2018 Second IEEE International Conference on Robotic Computing (IRC) (pp. 137-142).
- CNU (2018, abril) Repositorios Nicaragua. En Wellcome to the university repository of the CNU Obtenido de <http://repositorio.cnu.edu.ni/>
- Ramírez, K. (2018, abril) Material del Curso CI-2657. En Lista de Presentaciones Obtenido de <http://www.kramirez.net/ci-2657/materialci2657/>
- COSTARICAMAKERS. (2018, abril) Tag Archives: ROS: YA CONTAMOS PERSONAS... AHORA HAGAMOS ALGO CON PYTHON Obtenido de <http://costaricamakers.com/?tag=ros>
- Valverde, S. (2015). “Robótica inteligente: Implementación de sensores 3D para desenvolvimiento de robots móviles y vehículos autónomos” Obtenido de https://repositoriotec.tec.ac.cr/bitstream/handle/2238/6932/robotica_inteligente_implementaci%C3%B3n_sensores_desenvolvimiento.pdf?sequence=1&isAllowed=y.

Rosas, L., Fuentes, M., Samaniego, C., Alvarado, R., & Valencia, J. (2013). MODELADO Y CONTROL DEL ROBOT MÓVIL ROBOTNIK SUMMIT XL. Obtenido de <http://cerescontrols.com/wp-content/uploads/2013/10/SUMMIT-XL-Informe-Final.pdf>

Justina (2017, mayo) Justina. En user_manual Obtenido de https://github.com/RobotJustina/JUSTINA/blob/master/user_manual/user_manual.pdf

Eagle X (2018, marzo) Un robot construido por alumnos del TEC. En Vinculación y prestigio Obtenido de <https://tec.mx/es/noticias/queretaro/vinculacion-y-prestigio/un-robot-construido-por-alumnos-del-tec>

Juárez, G. (2008) IMPLEMENTACIÓN DE UN LABORATORIO VIRTUAL CON LA AYUDA DE LABVIEW, AL CURSO DE CIRCUITOS ELÉCTRICOS 1. Obtenido de http://biblioteca.usac.edu.gt/tesis/08/08_0148_ME.pdf

Prieto, R., Zaldivar, U., & Bernal, R. (2010) Creación de un Laboratorio Virtual para Optimizar el uso de un Laboratorio de Robótica Real. Obtenido de https://www.academia.edu/374898/Creaci%C3%B3n_de_un_Laboratorio_Virtual_para_Optimizar_el_uso_de_un_Laboratorio_de_Rob%C3%BCtica_Real

Ayala, J., Pupo, L., & Salazar, L. (2016) LABORATORIO VIRTUAL DE INSTRUMENTACIÓN Y CONTROL Obtenido de <http://www.informaticahabana.cu/sites/default/files/ponencias/EDU095.pdf>

Owen, A. (14 de marzo de 2016). What is the Best Programming Language for Robotics? [Entrada en un blog] Robotiq. Recuperado de <https://blog.robotiq.com/what-is-the-best-programming-language-for-robotics>

Tellez, R. (2017). A thousand robots for each student: Using cloud robot simulations to teach robotics. En *Robotics in Education*, 143-155. Springer, Cham.

MoveIt! (s. f.) Concepts. En Documentacion Obtenido de <http://moveit.ros.org/documentation/concepts/>

Corke, P. I. (1996). A robotics toolbox for MATLAB. *IEEE Robotics & Automation Magazine*, 3(1), 24-32. doi 10.1109/100.486658

ROS (s. f.) Core components. En Communications Infrastructure Obtenido de <http://www.ros.org/core-components/>

Windows 10 + ROS (2019.) Windows 10 IoT + ROS. En Get Started Obtenido de <https://microsoft.github.io/Win-RoS-Landing-Page/#/>

Charles River Analytics Inc (2018). [fotografía]. robot_localization Recuperado de <https://www.cra.com/work/case-studies/robotlocalization>

Orocoss_kdl Class Reference (2019). ChainIkSolverPos_NR_JL Recuperado de http://docs.ros.org/indigo/api/orocos_kdl/html/classKDL_1_1ChainIkSolverPos__NR__JL.html

OpenRAVE (2019). IKFast Recuperado de <http://openrave.org/docs/0.8.2/openravepy/ikfast/>

Pitzer, B., Osentoski, S., Jay, G., Crick, C., & Jenkins, O. C. (2012, May). Pr2 remote lab: An environment for remote development and experimentation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on* 3200-3205. IEEE.

Casañ, G. A., Cervera, E., Moughlbay, A. A., Alemany, J., & Martinet, P. (2015). *ROS-based online robot programming for remote education and training*. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* 6101-6106.

Universidad Tecnológica La Salle (2017) Ingeniería en Mecatrónica y Sistemas de Control. En Plan de estudio Obtenido de <http://www.ulsa.edu.ni/index.php/ingenieria-en-mecatronica-y-sistemas-de-control>

Calvo, I., Zulueta, E., Gangoiti, U., López, J. M., Cartwright, H., & Valentine, K. (2009). *Laboratorios remotos y virtuales en enseñanzas técnicas y científicas* (Vol. 3, No. 3, pp. 1-21). Ikastorratza.

Pérez, X., Cerda, A., & Incer, W. (2016) Desarrollar un laboratorio virtual para la realización de prácticas de laboratorio en la disciplina de control automático y automatización industrial

ROS Industrial (2016, noviembre) ROS QTC Plugin. En Repositorio Obtenido de https://github.com/ros-industrial/ros_qtc_plugin

ROS-Industrial (2017, Marzo) En ROS-Industrial Obtenido de <http://wiki.ros.org/Industrial/Tutorials>

Willow Garage. (2010, Abril) En Comic: Reinventing the Wheel [Entrada en un blog] Obtenido de <http://www.willowgarage.com/blog/2010/04/27/reinventing-wheel>

Merino, E. (2015) DISEÑO DE UN SIMULADOR DE COMPILADOR PARA PLATAFORMA MOODLE E IMPLEMENTACIÓN DE UN LABORATORIO VIRTUAL PARA LA ENSEÑANZA DE PROGRAMACIÓN.

Guerrero, L., Gómez, D., Sandoval, E., Thomson, P., Marulanda, J. (2014). SISMILAB, UN LABORATORIO VIRTUAL DE INGENIERÍA SÍSMICA, Y SU IMPACTO EN LA EDUCACIÓN.

ROBOTIS Co., Ltd. (2017). [fotografía]. *ROS Robot Programming*. Obtenido de <https://community.ros.org/t/download-the-ros-robot-programming-book-for-free/51>

QT Wiki (2019). [fotografía]. Qt Widget Gallery Recuperado de <https://doc.qt.io/qt-5/gallery.html>

ANEXOS

A: Instalación de Recursos de Software para el Laboratorio Virtual.

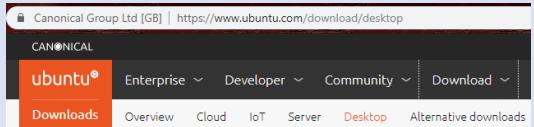
La instalación de todos los recursos de software expuestos fue realizada utilizando una laptop Intel Core I3 con 8GB de RAM y 120 GB de espacio libre de almacenamiento en el disco duro.

A.1 Instalación de UBUNTU 16.04 Xenial

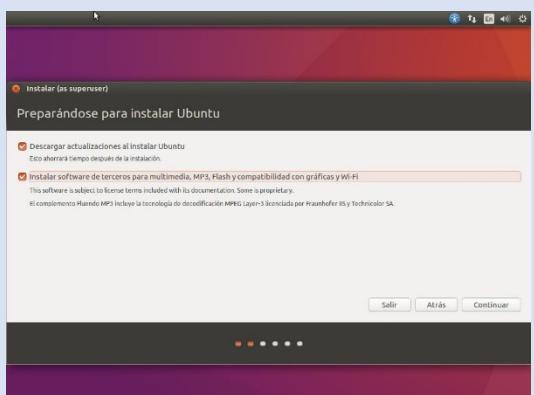
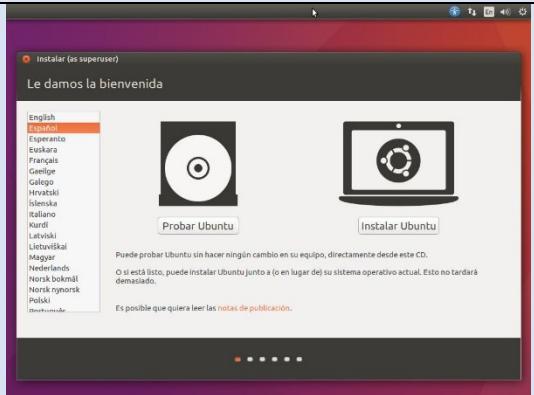
La Tabla A.1 detalla el proceso de instalación:

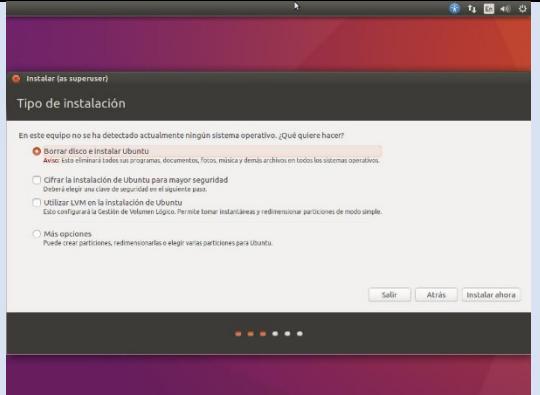
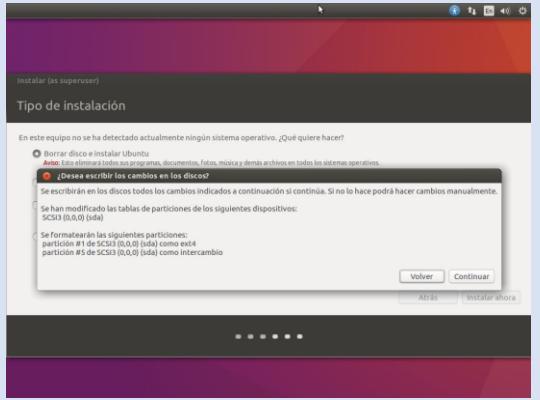
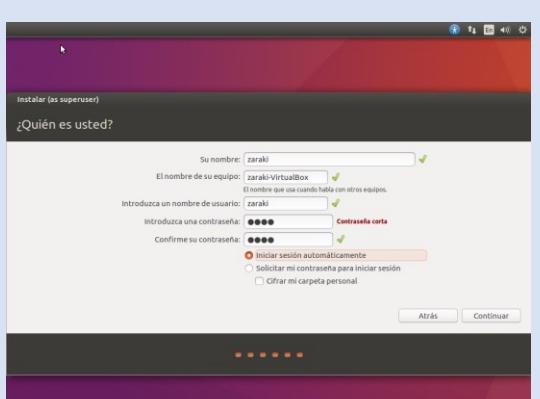
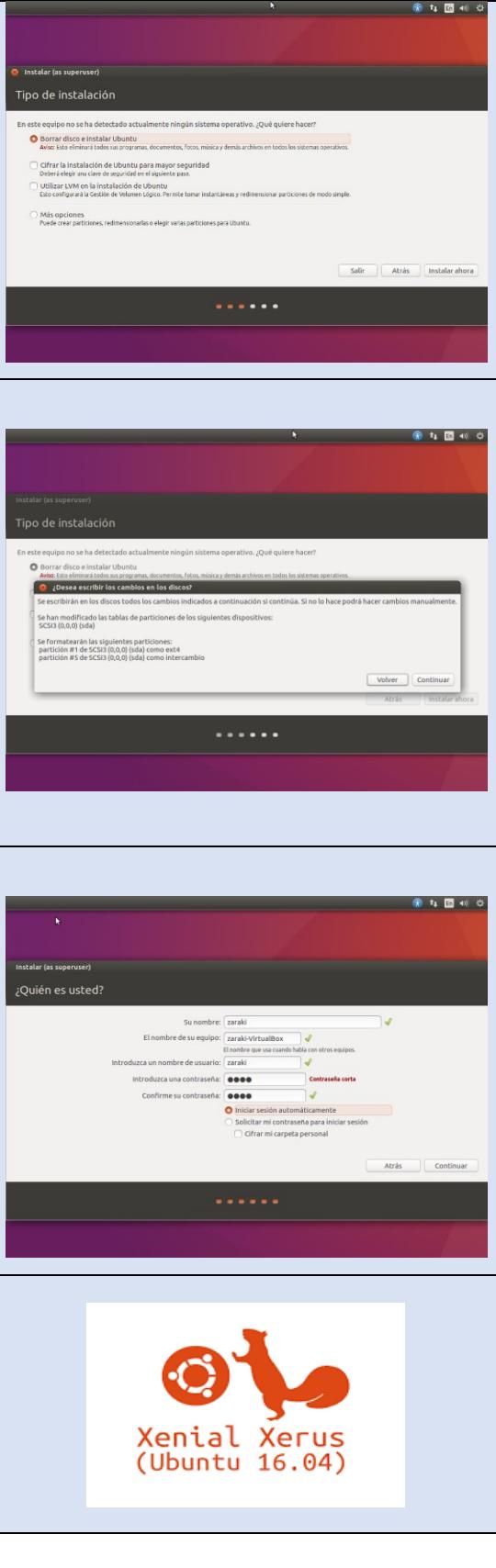
Tabla A.1: Instalación de UBUNTU 16.04 Xenial

Requisitos Preliminares	
Paso 0	<ul style="list-style-type: none">✓ Disponer de al menos 10 GB de espacio de almacenamiento libre.✓ Tener acceso a un DVD o una unidad flash USB que contenga la versión de Ubuntu 16.04✓ Descargar ISO ingresando al siguiente link: https://www.ubuntu.com/download/desktop
Paso 1	<p>Arranque desde una unidad flash USB</p> <p>Configurar la BIOS de la computadora para que se inicie automáticamente desde USB. Basta con insertar la unidad flash USB y encender el ordenador o reiniciarlo. Aparecerá la ventana de bienvenida, solicitando que elija el idioma e instale o pruebe Ubuntu.</p>
Paso 2	<p>Preparado para instalar Ubuntu</p> <p>Después de elegir instalar Ubuntu desde la ventana de bienvenida, se le preguntará sobre las actualizaciones y el software de terceros.</p> <p>Se aconseja activar la descarga de las actualizaciones y la instalación de software de terceros</p>



Download Ubuntu Desktop



	<h3 style="text-align: center;">Asignar Espacio</h3> <p>Utilizar las casillas de verificación para elegir si desea instalar Ubuntu junto con otro sistema operativo.</p> <p>Se aconseja disponer de un espacio de almacenamiento mayor a 10GB esto por las herramientas de visualización de ROS que luego serán instaladas como Gazebo.</p>	
Paso 3	<h3 style="text-align: center;">Inicio de Instalación de Ubuntu en la Unidad de Disco Asignada.</h3>	
Paso 4	<p>Después de configurar el almacenamiento, haga clic en el botón "Instalar ahora". Aparecerá un pequeño panel con una visión general de las opciones de almacenamiento que ha elegido, con la posibilidad de volver si los detalles son incorrectos. Haga clic en Continuar para iniciar el proceso de instalación o en volver para modificar.</p>	
Paso 5	<h3 style="text-align: center;">Detalles de registro.</h3> <p>El nombre del equipo es la forma en que su computadora aparecerá en la red y ROS lo ocupará como identificador dentro del servidor ROSMaster, mientras que el nombre de usuario será su nombre de usuario y de cuenta. A continuación, ingrese una contraseña segura. El instalador le informará si es demasiado débil.</p>	
Paso 6	<h3 style="text-align: center;">Instalación Completa.</h3> <p>Después de que todo se ha instalado y configurado, una pequeña ventana aparecerá pidiendo que reinicie la máquina. Hacer clic en Reiniciar ahora y extraer el DVD o la unidad flash USB</p>	

A.2 Instalación de ROS Kinetic

La tabla A.2 detalla el proceso de instalación:

Tabla A.2: Instalación de ROS Kinetic

Paso 1	Permitir todas las Fuentes de ROS
	<p>El paquete ROS no está soportado oficialmente en el conjunto de paquetes de UBUNTU para eso es necesario instalarlo desde un link externo. Se habilita el computador a que acepte paquetes de <code>packages.ros.org</code></p> <pre>\$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu \$(lsb_release -cs) main" > /etc/apt/sources.list.d/ros-latest.list'</pre>
Paso 2	Configuración del key
	<p>Se crea el directorio ros-latest.list para el soporte de los paquetes que conforman ROS además que ciertos repositorios necesitan de instalación de Claves de autorización o .key</p> <pre>\$ wget http://packages.ros.org/ros.key -O - sudo apt-key add -</pre>
Paso 3	Instalación general
	<p>Primero se debe asegurar que Ubuntu este actualizado, mediante:</p> <pre>\$ sudo apt-get update</pre> <p>Luego que la actualización se complete, se puede instalar la versión completa de Kinetic, la cual es la más recomendada, mediante</p> <pre>\$ sudo apt-get install ros-kinetic-desktop-full</pre>
Paso 4	Inicialización de ROS
	<p>rosdep habilita la fácil instalación de dependencias en el código a compilar y además es requerido para correr algunas dependencias de los componentes de ROS, Ejecutar los comandos siguientes, uno a la vez:</p> <pre>\$ sudo rosdep init \$ rosdep update</pre>
Paso 5	Configuración del entorno de trabajo de ROS para Ubuntu
	<p>Existen usuarios que poseen varias distribuciones de ROS con estos comandos permite inicializar con cual se trabaja. Permitiendo que las variables del entorno ROS sean automáticamente añadidas a la sesión bash cada vez que abrimos una nueva ventana Shell, Ejecutar los comandos siguientes, uno a la vez:</p> <pre>\$ source /opt/ros/kinetic/setup.bash \$ source ~/catkin_ws /devel/setup.sh</pre>

A.3 Espacio de trabajo Catkin

La tabla A.3 muestra el proceso de creación de un espacio de trabajo para compilar los paquetes de ROS:

Tabla A. 3 Creación del Espacio de Trabajo Catkin.

Creación del Espacio de trabajo ROS	
Paso 1	El compilador necesita que se le indique la ruta de acceso a los archivos a ser compilados para esto se construye un entorno de trabajo (catkin workspace), <i>Ejecutar en una terminal</i> \$ mkdir -p ~/catkin_ws/src
Inicialización del Espacio de trabajo	
Paso 2	Con las carpetas creadas accedemos a la carpeta donde estarán almacenados los archivos fuentes de C++ a ser compilados. Ejecutamos los dos comandos uno a la vez. \$ cd ~/catkin_ws/src \$ catkin_init_workspace
Construcción del Espacio de trabajo (Building)	
Paso 3	Ya que se ha inicializado el espacio de trabajo procedemos a compilar este mismo, debido a que no hemos agregado ningún paquete a compilar este no dilatará en crear todo el directorio necesario del espacio de trabajo. \$ cd ~/catkin_ws/ \$ catkin_make
Creación de un paquete.	
Paso 4	Para la creación de un paquete nuevo utilizamos Catkin con el cual podemos hacer referencia a los recursos de software de otros paquetes necesarios para nuestros requerimientos, los paquetes más comunes son roscpp, sensor_msgs, geometry_msgs \$ catkin_create_pkg -D “Descripción del paquete” -a Yeser --rosdistro KINETIC mi_robot roscpp sensor_msgs

A.4 Instalación de QT – Creator.

La tabla A.4 muestra el proceso de Instalación del IDE QT – Creator.

Tabla A.4: Instalación del IDE QT- Creator.

Obtención del Instalador	
Paso 1	Ingresamos a la Pagina WEB www.QT57Download.com o via comando procedemos. <i>Ejecutar en una terminal</i> \$ wget http://download.qt.io/official_releases/qt/5.7/5.7.0/qt-opensource-linux-x64-5.7.0.run
Proveer Permisos al archivo.	
Paso 2	Procedemos a garantizarle los permisos al archivo descargado para la instalación. \$ chmod +x qt-opensource-linux-x64-5.7.0.run
Instalación	
Paso 3	Dentro de la instalación se abrirá una ventana en la cual se indicará si se desea una instalación completa o parcial del IDE, Preferiblemente indicarle instalación completa para no tener problemas con librerías de QT que puedan ser relevantes. \$./qt-opensource-linux-x64-5.7.0.run

A.5 Instalación del Plugin ROS para QT – Creator.

La Tabla A.5 muestra el proceso de Instalación del Plugin de ROS desarrollado por ROS – Industrial para el IDE QT – Creator.

Tabla A.5: Instalación de Plugin ROS QT.

Instalación de dependencias	
Paso 1	Instalamos via terminal las dependencias de el repositorio de Levi Armstrong para Qt versión Xenial. <i>Ejecutar en una terminal los comandos uno a la vez.</i> \$ sudo add-apt-repository ppa:levi-armstrong/qt-libraries-xenial \$ sudo add-apt-repository ppa:levi-armstrong/ppa
Instalación del Plugin	
Paso 2	Actualizamos e instalamos el plugin de ROS para la versión de Qt5.7 \$ sudo add-apt-repository ppa:levi-armstrong/qt-libraries-xenial

	Configuración del sistema
Paso 3	<p>Se configura el IDE de QT a reconocer las librerías de la versión de Qt 5,7, cabe destacar que esto se hace debido a que nativamente Ubuntu posee librerías Qt para ejecutar ciertos procesos. Editamos vía comando el archivo .conf de QT</p> <pre>\$ sudo gedit /usr/lib/x86_64-linux-gnu/qt-default/qtchooser/default.conf</pre> <p>Reemplazamos los parámetros:</p> <pre>/usr/lib/x86_64-linux-gnu/qt4/bin /usr/lib/x86_64-linux-gnu</pre> <p>Por:</p> <pre>/opt/qt57/bin /opt/qt57/lib</pre>
Paso 4	Clonación del espacio de trabajo Recomendado para el Plugin
	<p>Como se ha mencionado en la sección 2.2 de instalación de recursos es necesario el uso de un espacio de trabajo de compilación de ROS, el repositorio ros_qtc_plugin es recomendado para una buena instalación del plugin</p> <pre>\$ git clone -b master https://github.com/ros-industrial/ros_qtc_plugin.git</pre>
Paso 5	Iniciación del workspace y de las librerías de ROS.
	<p>Una vez descargado el repositorio procedemos a inicializar El espacio de trabajo con el cual el IDE Qt Podrá compilar vía CMake los paquetes de ROS</p> <pre>\$ cd ~/ros_qtc_plugin-devel \$ bash setup.sh -d</pre>

Con los pasos descritos en la tabla A.5 podemos abrir el IDE Qt Creator e inicializar el Plugin tal como se muestra en las siguientes imágenes.

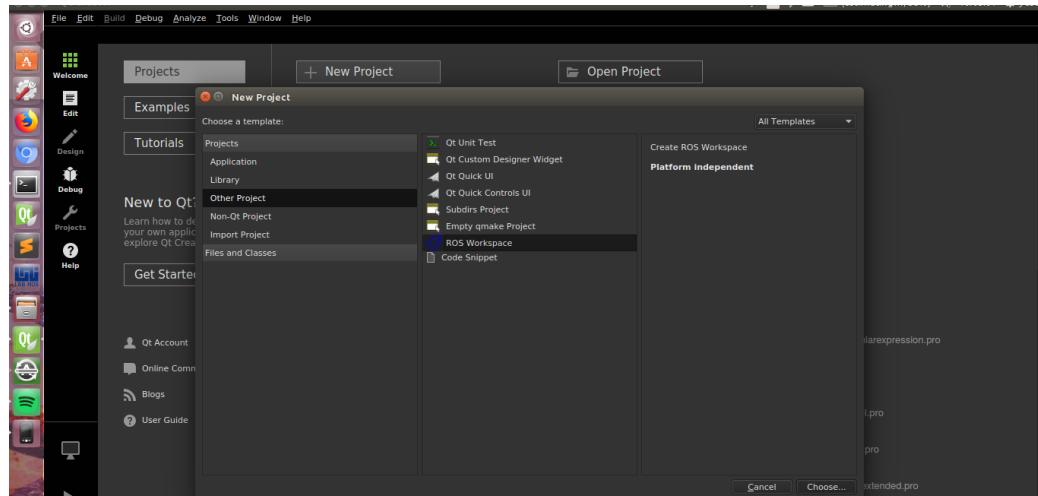


Figura A.1: IDE QT-Creator Activación del Plugin Catkin, para el espacio de Trabajo.

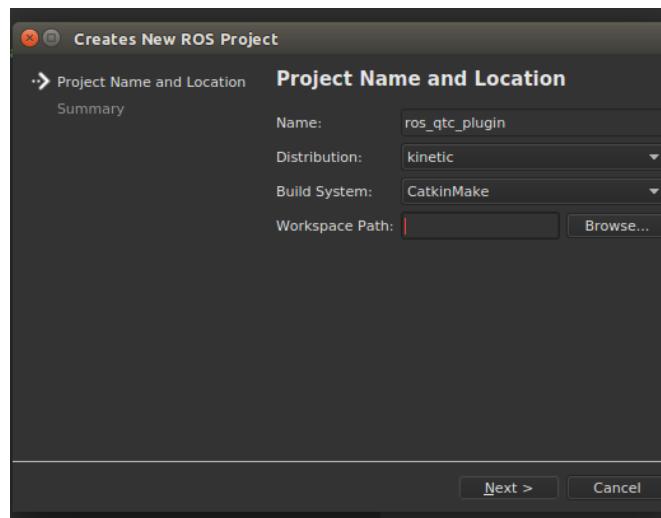


Figura A.2: Ubicación del espacio de Trabajo.

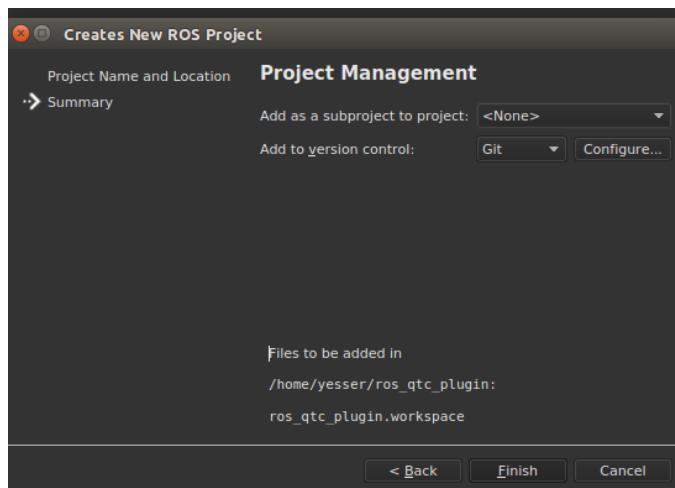


Figura A.3: Espacio de trabajo Creado.

A.6 Instalación de Gazebo

La Tabla A.6 muestra el proceso de instalación de Gazebo.

Tabla A.6 Instalación de Gazebo.

	Instalación de dependencias
Paso 1	Instalamos via terminal las dependencias base para instalar Gazebo \$ sudo apt-get install ros-kinetic-ros-base
Paso 2	Instalación de Gazebo
	Instalamos via terminal los paquetes de Gazebo \$ sudo apt-get install ros-kinetic-gazebo9-ros-pkgs ros-kinetic-gazebo9-ros-control ros-kinetic-gazebo9*

