UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

Uso de software libre para la implementación y simulación de un robot autónomo

T E S I S

QUE PARA OPTAR POR EL TÍTULO DE

LICENCIADO EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

P R E S E N T A :

MARTÍN TORRES GALICIA

D I R E C T O R A

M. EN C. MAGALI CORTEZ VÁZQUEZ

Ciudad de México, noviembre de 2023.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Resumen de la tesis de **Martín Torres Galicia**, presentado como requisito parcial para la obtención del grado de **LICENCIADO EN INGENIERÍA en SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES**. Ciudad de México, noviembre de 2023.

Uso de software libre para la implementación y simulación de un robot autónomo

En este documento se describe el desarrollo de un robot autónomo de búsqueda de personas utilizando herramientas de software libre, que cuenta con una unidad de procesamiento abordo y un sistema de comunicación basado en Wifi. Este proyecto se implementa con un robot Mindstorm EV3 de Lego, una Raspberry Pi 4, un módulo de cámara Raspberry Pi, dos tarjetas de desarrollo de comunicaciones NodeMCU con módulos ESP8266 y una PC de escritorio. De igual manera, se presenta la implementación del robot, así como un modelo de simulación de este usando el software Gazebo.

El robot lleva abordo su sistema de procesamiento, que es una Raspberry Pi que cuenta con ROS, y con un módulo de cámara para realizar el reconocimiento facial. El proyecto está en base a [1], en el que el control del robot es por medio de scripts programados en lenguaje Python y el reconocimiento facial se hace con ayuda de la librería OpenCV, estos son ejecutados en ROS, en este caso en la versión Noetic. Este trabajo aporta un sistema cliente-servidor que, tras identificar un rostro, envía

un mensaje a un servidor web local que guarda en su base de datos. Para hacer esto posible se hace uso de dos módulos NodeMCU, uno tiene la función de crear una red LAN y el otro sirve como puente entre Raspberry Pi y el servidor web local. La parte principal de este trabajo es el desarrollo de un sistema de comunicación para enviar datos del robot a un servidor web, así como la implementación del procesamiento abordo.

Palabras clave: Mindstorm EV3 Lego, ROS, Gazebo, ESP8266, Xampp, rosserial, RPyC.

Dedicatoria

A mi familia, amigos, mi Lety H.U.,
seres queridos que dejaron este plano terrenal
y todos aquellos quienes creyeron en mí.

Agradecimientos

Gracias a este proyecto educativo que es la Universidad Autónoma de la Ciudad de México, quien me dio la oportunidad de continuar con mis estudios superiores y culminar mi licenciatura. Así mismo esta misma institución me brindo apoyo con las becas de: realización de servicio social, elaboración de trabajo recepcional y para la impresión/empastado de trabajo recepional. También la UACM me proporciono aulas, laboratorios, espacios de estudio, compañeros, compañeras, amigos y amigas quienes apoyaron y ayudaron en mi estudio y estancia en la universidad.

Mi reconocimiento a todos los profesores, profesoras, maestros y maestras quienes me compartieron de sus conocimientos, experiencias, formas de pensar y me bridaron todo su apoyo para formar mi perfil académico, social y humano. En especial mi más grande reconocimiento y agradecimiento a mi maestra y directora de tesis la M. en C. Magali Cortez Vázquez quien me guio, me apoyo, ayudo y me dio herramientas necesarias para la realización de este trabajo de tesis y mi formación académica.

Por otra parte, agradezco a mis lectores: el profesor Ricardo, Joel Yazbek y a Luis René quienes me brindaron de su tiempo para la revisión de este trabajo.

Mi más especial agradecimiento a mis padres Fidel, Silvia y mis hermanos Benjamín y Brenda por todo su cariño, su compresión, su apoyo incondicional y sus palabras de aliento. También agradezco a Dios por darme la vida, la paciencia, el conocimiento y la oportunidad de culminar mi licenciatura.
Agradezco infinitamente a mi Lety Hernández Ulloa, mujer increíble, maravilloso ser humano, e increíble persona, lo más importante de mi vida y quien amo mucho. Ella es quien siempre estuvo a mi lado a pesar de las adversidades, jamás dejo de creer en mí, quien, con su energía, calidez, palabras, brillo, alegría, amor y cariño me dieron fortaleza, fuerza decisión y optimismo para no rendirme.

Contenido

Lista de figuras

Lista de tablas

Capítulo I Introducción

En la actualidad, el uso de robots en todo el mundo ha captado la atención y con el paso de los años se ha buscado mejorarlos, ofreciendo una amplia gama de aplicaciones para el beneficio de la humanidad. Tras los años, se han desarrollado diferentes tipos de robots, los cuales han sido clasificados por generaciones: Primera Generación, de Robots manipuladores; Segunda Generación, de Robots de aprendizaje; Tercera Generación, de Robots con sensores; Cuarta Generación, de Robots móviles, y Quinta Generación, de Robots inteligentes [2]. Este trabajo se enfoca a la cuarta generación de robots. Un robot móvil se diferencia de los demás por ser un robot que no está anclado y se puede mover libremente por diferentes entornos, cabe mencionar que a este tipo de robots también se le denomina robots autónomos. Pueden existir robots autónomos de bajo o elevado costo, así como

pueden existir robots sencillos o complejos, por ello, dentro de esta cuarta generación se pueden encontrar robots emblemáticos del tipo androides, zoomorfos, entre otros. Actualmente las aplicaciones de robots móviles más comunes son: inspección de volcanes, aviones no tripulados para desactivar explosivos, o para aplicaciones muy específicas. Por otra parte, hoy en día existen varias herramientas para el desarrollo de robots, como son simuladores de robots, software y hardware. Dentro de la amplia gama de software para programar y simular robots, en este trabajo se usa software y simuladores de licencia libre, ya que se busca elaborar un robot autónomo de bajo costo, por lo tanto, de igual manera se usará hardware de bajo costo. Cabe resaltar que muchas de estas herramientas están disponibles para cierta población con conocimientos en el área y por lo general no existe documento que recopile toda la información necesaria para iniciar en el tema, que incluya ejemplos y la solución a los errores más comunes que se pueden llegar a encontrar al usar las herramientas de software y hardware que se encuentren al alcance del público en general. Es por ello, que en este trabajo se busca generar un material de apoyo para todo el público interesado en la simulación, control y programación de un robot autónomo con la capacidad de detectar rostros humanos. En particular, se usa un robot Lego Mindstorm EV3, el software ROS en su versión Noetic Ninjemys y el simulador Gazebo. Cabe señalar que el presente proyecto es la continuación y mejora del trabajo de tesis desarrollado por un egresado de la UACM en 2019 [1].

1.1 Descripción del problema y propuesta de solución

Como se menciona anteriormente este proyecto busca mejorar y actualizar el trabajo de 2019 [1]. Para este caso se implementa un sistema de procesamiento que irá a bordo del robot, así como un sistema de comunicación para enviar datos del robot a un servidor web, es por ello que también se implementa un servidor web que se encarga de recibir y almacenar datos que recibe del robot. El dato que se envía, recibe y almacena es información que confirma el reconocimiento de un rostro facial que detecta el robot EV3.

Por otro lado, existen muchos recursos, plataformas y software enfocado al diseño, simulación e implementación de robots, pero no hay una guía con ejemplos para robots de bajo costo, de manera que este documento busca generar el aprendizaje autodidacta para un público que inicia en proyectos robóticos, así como para público con mayor habilidad técnica y conocimiento en el tema.

1.2 Objetivos

Mejorar y actualizar el sistema de búsqueda de personas, agregando una unidad de procesamiento que va a bordo del robot y hacer uso de software más reciente. Así mismo, se implementa un sistema de comunicación que cumple con enviar información del robot a un servidor web y que este mismo almacena información en su base de datos.

Por otra parte, crear un documento de referencia que describe cómo generar un modelo de simulación de un robot autónomo usando software libre, tanto para simulación del sistema en Gazebo[1], así como programar el comportamiento del robot de hardware EV3[2] usando ROS[3] y Python.

Objetivos particulares

- ✓ Desarrollar un modelo de simulación de un robot usando herramientas de software libre.
- ✓ Crear un compendio de ejemplos para analizar los conceptos involucrados en el proceso de programar el comportamiento de un robot y para interactuar con el hardware; esto es con el afán de reducir el tiempo de desarrollo e investigación y que el interesado se concentre más en la implementación y análisis de los algoritmos de proyectos robóticos.
- ✓ Instalar ROS en su versión más reciente, Noetic Ninjemys, en una tarjeta Raspberry Pi 4 modelo B para ejecutar los scripts del sistema de búsqueda.
- ✓ Mejorar la aplicación del robot EV3 dedicado a la búsqueda de personas agregando un sistema de comunicación, un servidor web y una unidad de procesamiento abordo que ejecuta ROS.

[1] Simulador de robótica en 3D de código abierto.
[2] Kit de educativo de robótica de la marca LEGO.
[3] Es un conjunto de bibliotecas y herramientas que ayudan a crear aplicaciones para proyectos robóticos.

1.3 Aportaciones

Se crea un documento que es una guía práctica de temas que son referidos a la iniciación de proyectos robóticos.

Se mejora la autonomía del robot EV3 que se encarga de realizar búsqueda de personas. Estás mejoras cumplen con los objetivos planteados del presente trabajo, todo esto haciendo uso de herramientas de software e implementación de hardware recientes.

Se evalúan módulos de comunicación que son compatibles con ROS, con el fin de desarrollar un sistema de comunicación entre la unidad de procesamiento del robot y el servidor web.

1.4 Metodología

Para el desarrollo del presente trabajo y de acuerdo con los temas que se abordarán, se seguirá la siguiente metodología, la cual ayuda en la comprensión de las etapas del proyecto como son: el aprendizaje del funcionamiento e implementación del robot educativo EV3, de la tarjeta inalámbrica Xbee junto con su adaptador para la Raspberry, de la tarjeta de desarrollo NodeMCU y de la minicomputadora Raspberry Pi 4 modelo B. Así mismo, conocer la programación y lenguaje que se ocupa para programar las acciones que debe realizar el robot, y para familiarizarse con el software que se ocupa, en este caso ROS (por sus siglas en inglés de *Robot Operating System*) y Gazebo. La metodología se puede resumir en los siguientes pasos:

1.- Recopilación de información acerca del tema.

2.- Seguimiento de tutoriales para familiarizarse y reforzar los temas aprendidos en la búsqueda de información.

3.- Desarrollar un modelo de simulación del robot EV3, modificar y programar el algoritmo que se encarga de controlar al robot.

4.- Implementar una computadora a bordo del sistema móvil, que será el cerebro del robot, usando el software libre ROS y la tarjeta de implementación Raspberry PI.

5.- Dotar de comunicación inalámbrica al robot, con el objetivo de enviar datos recabados por medio de sensores, para ser procesados y almacenados en una computadora remota.

6.- Reportar y documentar de manera concisa el desarrollo de la programación, simulación e implementación del robot EV3.

Capítulo II Marco teórico

En el presente capítulo se abordan temas relevantes para la comprensión de los conceptos teóricos para crear y simular un proyecto robótico usando software de licencia libre; en este caso ROS Noetic y Gazebo. Así mismo, se hace mención sobre el origen de estas herramientas y las motivaciones para elegirlos. También se da un breve resumen sobre la estructura y funcionamiento de éstos, para que el lector se familiarice con dichas herramientas.

2.1 Simulación de sistemas robóticos

Las herramientas de simulación son parte fundamental para toda aquella comunidad que se involucra en el desarrollo de las diferentes ramas de la robótica. Es por ello que existe un gran número de herramientas de simulación, que son de licencia libre o de paga. Por lo general se les conoce como herramientas de simulación dinámica.

La simulación dinámica es aquella que estima características físicas del prototipo o modelo del robot a desarrollar, así como las características del entorno en que se simulan. Este tipo de simulación es para robots que contienen elementos articulares, es la más precisa en la evaluación de fuerzas y momentos que se generan en las articulaciones de un robot cuando se realiza una trayectoria descrita dentro de un entorno, por lo tanto, se considera que la simulación dinámica se asemeja mucho a la realidad [3, p. 4].

Estas herramientas han tenido un gran auge en el campo de la robótica por lo que se usan para evaluar el comportamiento de robots sencillos hasta proyectos complejos como robots humanoides. Hasta el momento existe una gran variedad de simuladores, el más adecuado depende del propósito y uso que les de cada usuario o desarrollador.

2.1.1 Herramientas de simulación

Estas herramientas están conformadas por dos elementos muy importantes que son: Motores de Físicas y Simuladores de Sistemas. Los Motores de físicas se

implementan con software dedicado a simular sistemas físicos que pueden ir desde sistemas sólidos, rígidos, de fluidos y de partículas, en los cuales se evalúa la dinámica, el movimiento y la elasticidad en estos sistemas. Cabe mencionar que el origen de estos motores viene de la creación de los videojuegos debido a que, por la gran demanda de los usuarios, los videojuegos pretenden llegar al realismo para proporcionar una mejor experiencia de juego. Es por ello, que los motores de físicas han evolucionado de manera exponencial y a la par con la tecnología de los videojuegos. Los motores de físicas se dividen en dos grupos, de los cuales ambos calculan las fuerzas de contacto, pero en uno de ellos se considera a la fuerza de contacto como una restricción bilateral o unilateral [3, p. 6]. Dentro de los ejemplos de motores de físicas se puede encontrar el software XDE, Bullet y ODE, siendo estos los más representativos.

Por otro lado, los Simuladores de Sistemas son aquellos softwares que están compuestos por un motor de físicas, para así iniciar la simulación dinámica de los proyectos robóticos en un entorno, además de proporcionar herramientas como: simulación de sensores, edición de modelos a simular, así como facilitar la simulación de control de los proyectos. Esto último, es con el objetivo de proveer la simulación y el control, de manera que no existan problemas al pasar el código que se ejecutó en la simulación al robot real. Dentro de esta categoría, los siguientes son ejemplos de softwares populares de Simuladores de Sistemas: OpenHRP, Gazebo, V-Rep y RoboTran.

2.1.2 Herramientas de simulación disponibles

Como se mencionó anteriormente, debido al gran avance de la tecnología y específicamente al campo de la robótica, se han desarrollado muchas herramientas para la simulación de este tipo de proyectos. Sin embargo, en el documento [3, p. 10], se presenta un estudio estadístico para determinar cuáles simuladores son más populares, en dónde son más utilizados y las características en las cuáles los usuarios basan su elección. Se concluye que, por lo general, se emplean en universidades, centros de investigación, instituciones y empresas. Así mismo, la elección de herramientas de simulación tiende a que los usuarios trabajen con un Sistema Operativo (SO) basado en GNU/Linux, por lo que la elección de uso de herramientas se enfoca a software de código abierto, así como el uso de un middleware[4] robótico y con una tendencia alta en el uso de lenguaje de programación de C++. Otro punto relevante es que los usuarios buscan un simulador que sea estable, que la simulación sea realista, que sea de código abierto, que la simulación sea rápida y que el código no difiera entre la simulación y la implementación en el robot real. De acuerdo a esos parámetros, los tres simuladores más populares son: Gazebo, en primer lugar, V-Rep, en segundo y Webots en tercero [3, p. 12].

[4] Software que brinda servicios y funciones comunes a las aplicaciones, además de lo que ofrece el sistema operativo. Generalmente, se encarga de la gestión de los datos, los servicios de aplicaciones, la mensajería, la autenticación y la gestión de las API.

Con el resultado del estudio anterior y de acuerdo con toda la información, ejemplos, proyectos funcionales y tutoriales disponibles en la red, la herramienta de simulación que se va a usar en este trabajo es el Simulador de Sistemas **Gazebo**.

2.2 Software para robótica ROS

En los inicios, para el desarrollo de un proyecto robótico, los creadores se topaban con diversas dificultades, dado que un robot es un sistema complejo. La primera dificultad era contar con habilidades y conocimientos en mecánica, electrónica y programación, y como segunda dificultad, dedicar un tiempo considerable para desarrollar un software que comunicara todo el hardware del robot y que a su vez realizara ciertas tareas, además de desarrollar el hardware necesario. Con todo esto, los diseñadores, investigadores y pioneros en el marco de la robótica, se dieron a la tarea de crear diversas herramientas que facilitaran el diseño, simulación y desarrollo de robots.

Conforme se iban creando proyectos robóticos, los desarrolladores se percataron de que el software que controlaba y comunicaba los subsistemas de un robot eran parecidos entre diversos proyectos que hasta el momento se habían creado. Esto dio como resultado una herramienta más importante, un software que funcionaba como el SO del robot y cada desarrollador, si lo quería, compartía nuevas funcionalidades de este a toda la comunidad, para que todo aquel que iniciara un proyecto no se viera en la necesidad de comenzar desde cero. Por lo tanto, este

software o sistema operativo debe ser colaborativo, público, multiplataforma y que tuviera una gama alta en cuanto a la expansión y desarrollo a nivel de software.

Entre los diversos softwares o SO creados se encuentran: Microsoft Robotics Developer Studio (Microsoft RDS), NAOqi, URBI, CARMEN, PLAYER, Mobile Robot Programming Toolkit (MRPT), Lightweight Communications and Marshalling (LCM), entre otros. De todos los SO que se desarrollaron, el que cumple con todas las necesidades que buscaban los desarrolladores, es el SO llamado ROS (Robotic Operating System).

ROS se lanzó en el 2007, este se desarrolló como uno de los tantos proyectos en Stanford Artificial Intelligence Laboratory (SAIL) de la Universidad de Stanford. Tiempo después, en el 2008, ROS es respaldado por una empresa privada que tiene vínculos con la Universidad de Stanford, esta empresa es Willow Garage, la cual fue formada en 2006 por Scott Hassan, uno de los empleados que participaron en los inicios de Google. En 2013 ROS pasa a formar parte de las Open Source Robotics Fundation (OSRF), actualmente sigue vigente su afiliación.

Las ventajas de ROS son las siguientes:

- ✓ Cuenta con herramientas y bibliotecas que ayudan al fácil desarrollo de programación de algoritmos de un robot.
- ✓ Dado que es el SO más usado por desarrolladores, este cuenta con una gran comunidad, colaboradores y por lo tanto existe una gran variedad de ejemplos que están abiertos a todo el público.

- ✓ Cuenta con versiones actualizadas que mejoran y corrigen errores de versiones previas del SO.

De acuerdo con la literatura se define a ROS como un SO o un framework para crear aplicaciones robóticas que permite reutilizar código entre proyectos robóticos. Se define como un sistema operativo porque tiene una gran cantidad de herramientas, bibliotecas y colección de paquetes que ayudan al desarrollo de software de un robot. A grandes rasgos, ROS es una herramienta que ayuda al desarrollo de aplicaciones enfocadas a la robótica, ya sean sencillas o complejas. Siendo ROS un framework dedicado a la tecnología y enfocado a la robótica, este tiene que ir evolucionando continuamente, mejorando o corrigiendo errores en su sistema para así dar un mejor funcionamiento. Cada mejora genera una nueva distribución. En la figura 1 se indican algunas de las distribuciones de ROS.

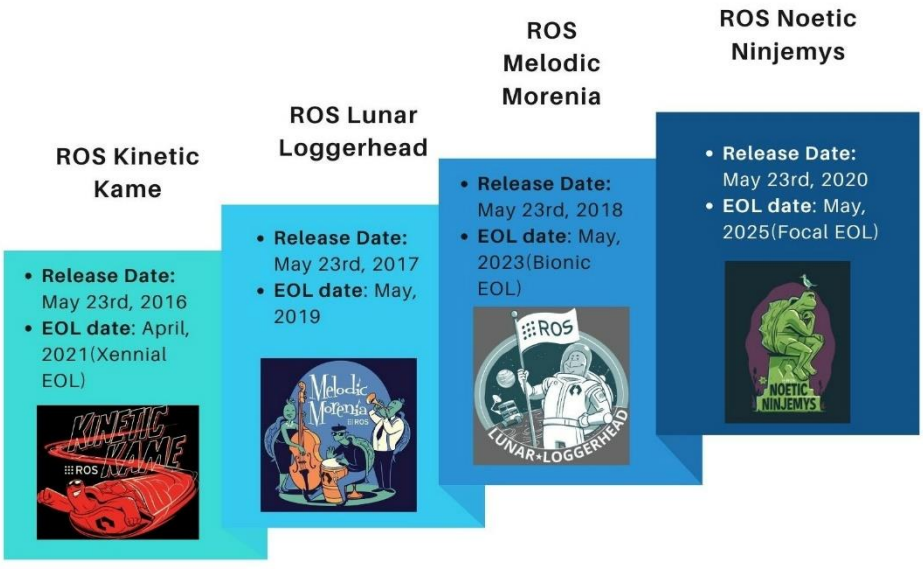


Figura 1. Distribuciones de ROS. Con referencia [4].

2.2.1 Filosofía de desarrollo ROS

Los principios generales más importantes en ROS son los siguientes:

- ✓ **Peer-to-peer.** Los proyectos (paquetes) elaborados en ROS se encuentran formados por programas, que en este framework son llamados como "**nodos**". Éstos se comunican y se interconectan con una topología **peer to peer** (punto a punto) para así comunicarse por medio de un continuo intercambio de mensajes, los cuales son recibidos (suscritos) o enviados (publicados) por los mismos nodos, más adelante se hace mención sobre esto.
- ✓ **Multilingual.** Cada desarrollador elabora sus códigos en diversos lenguajes dependiendo de sus necesidades. Por ello ROS es capaz de leer, ejecutar y comunicarse entre sí, independientemente del lenguaje que se haya usado [5, p. 13]. Actualmente los lenguajes soportados en ROS son: *C++, Python y Lisp*, los cuales a su vez tiene bibliotecas experimentales con lenguajes como: *Java y Lua.*
- ✓ **Based on tools.** ROS está diseñado como un microkernel (micronucleo), lo que da la posibilidad de realizar tareas como: navegar y explorar el código, visualizar las interconexiones, graficar y visualizar, datos, almacenar datos, etc.
- ✓ **"Ligth".** Todo el código se encuentra estructurado en bibliotecas pequeñas que son independientes, esto ayuda a la reutilización de los proyectos.

- ✓ **Free and open source.** ROS es un software gratuito bajo licencia BSD[5].

ROS tiene un diseño de micronúcleo, en el que organiza su sistema de archivos en tres niveles.

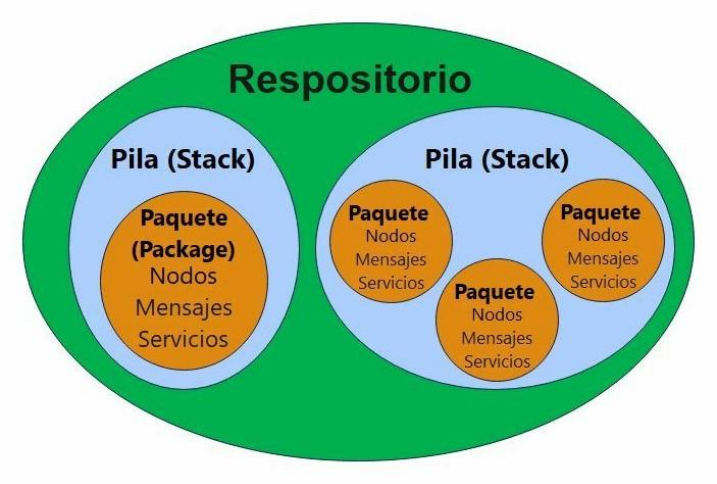


Figura 2. Contenido de un Repositorio de ROS. Con referencia [7].

La figura 2 muestra el nivel de organización más bajo, que es el **Package** (paquete), enseguida está el nivel intermedio, que es el **Stack** (pila) y por último se encuentra el nivel que engloba a todos, que es el **Repository** (repositorios). Lo siguiente es un breve resumen sobre el contenido de cada nivel de archivos:

1. *Paquetes.* Contienen archivos como: nodos, modelos de robot, tipos de mensajes y servicios, herramientas o biblioteca; mismos que se definen más adelante.

[5] BSD viene de las siglas Berkley Software Distribution, esta licencia se le adjudica a los SO que su código fuente se encuentra derivado de código abierto del sistema operativo AT&T's Research UNIX [6].

2. *Pilas.* Están formadas por grupos de paquetes que conforman librerías de alto nivel. Las pilas pueden agrupar paquetes que se complementan entre ellos.
3. *Repositorios.* Son agrupaciones de pilas que fueron descargadas o instaladas por medio de una línea de comandos, así como lo hace Linux.

Una característica común entre los *Paquetes* y las *Pilas* es que ambos tienen un archivo que da información específica de su contenido, su función y los paquetes de los que dependen. Para las *Pilas,* el archivo se llama **stack.xml** y para los *Paquetes* se llama **manifest.xml**.

2.2.2 Organización de directorio y archivos en el espacio de trabajo de ROS

ROS se instala en la dirección ***/opt/ros****,* donde se encuentran los archivos restringidos y por lo tanto no podemos modificar sus paquetes, esto es porque podemos realizar daños al sistema operativo que tengamos instalado, el cual puede ser alguna distribución de Linux. Para no dañar el sistema, ROS habilita y nos ayuda a crear un espacio de trabajo, en este caso en una carpeta denominada **workspace**, la cual contiene los paquetes de proyectos que se desarrollan.

Teniendo un espacio de trabajo, ya se puede crear un paquete, el cual está formado por carpetas y algunos archivos. Las carpetas que se encuentran por lo general dentro de un paquete de ROS están organizadas como se muestra en las figuras 3 y 4, y se describen en los siguientes párrafos.

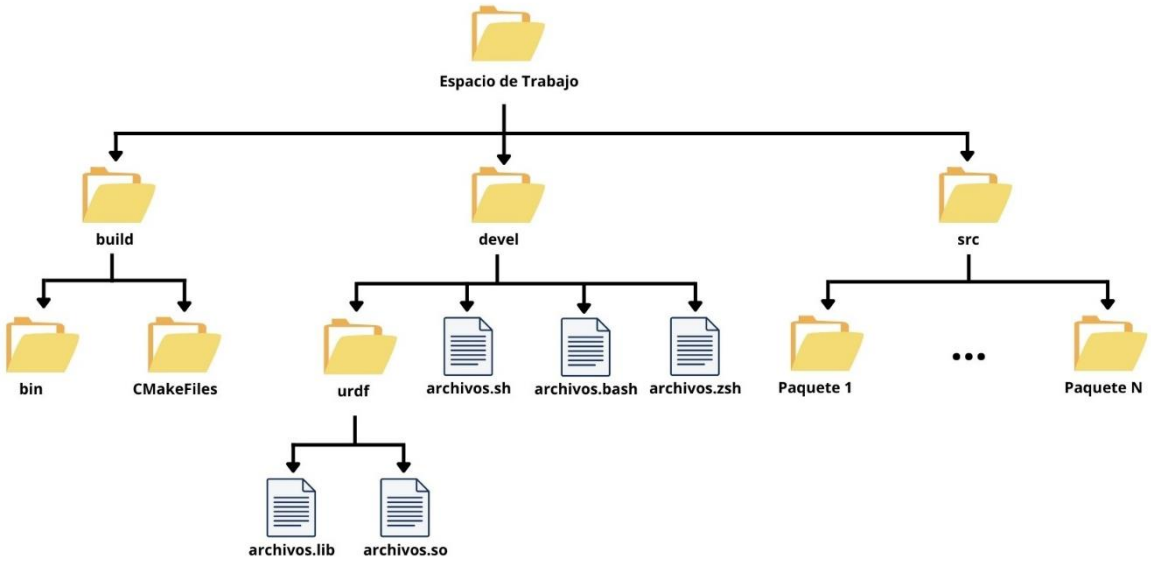


Figura 3. Contenido del espacio de trabajo.

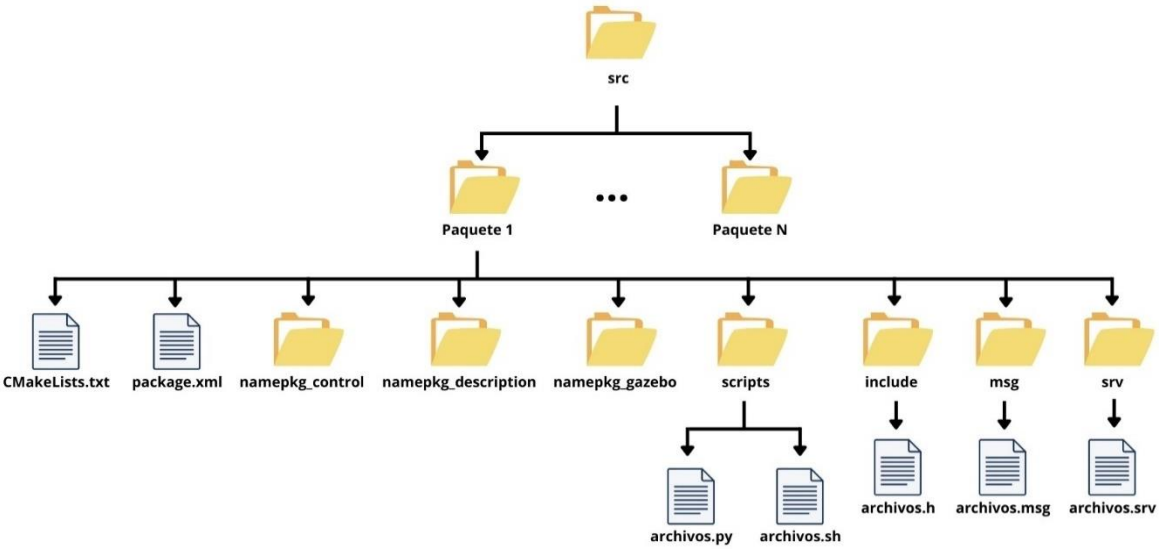


Figura 4. Contenido de manera general de la carpeta src.

- **Carpeta build.** Se crea la primera vez que compilamos el paquete y contiene los archivos de compilación del proyecto.

- **Carpeta bin.** Contiene archivos ejecutables del paquete, estos son lanzados como nodos, mismos que se explican posteriormente.
- **Carpeta lib.** Contiene bibliotecas de archivos con extensión **.lib** y .**so**.
- **Carpeta src.** Contiene los códigos de programación del proyecto o paquete a desarrollar, en lenguajes según la conveniencia del desarrollador. Una vez que se compilan se crearan ejecutables que se almacenan en la carpeta **bin**.
- **Carpeta include.** Contiene los archivos con extensión **.h** de los programas que conforman los paquetes.
- **Carpeta launch.** Aquí se encuentran archivos que inician uno o más nodos de los paquetes, estos archivos tienen extensión **.launch**.
- **Carpeta yaml.** Su contenido está conformado por archivos de lista de parámetros, la extensión de estos archivos es **.yaml**. Esta carpeta suele nombrarse **config**.
- **Carpeta msg.** Aquí yacen los tipos de mensajes que se definieron en el paquete. Son archivos de tipo **.msg**.
- **Carpeta msg_gen.** Contiene archivos con extensión **.h** que se autogeneraron al compilar los tipos de servicio definidos en el paquete.
- **Carpeta srv.** Su contenido tiene archivos con extensión **.srv**, estos corresponden a los tipos de mensajes de los servicios que son definidos en el paquete.

- **Carpeta srv_gen.** Aquí se encuentran archivos con extensión **.h**, los cuales se generan de manera automática como resultado de la compilación de los tipos de servicios definidos en el paquete.

Aparte de incluir el anterior conjunto de carpetas, también se encuentran normalmente presentes los siguientes dos archivos, que componen a cada paquete de ROS (proyecto de ROS) creado:

- **Archivo CMakeLists.txt.** La información que se encuentra aquí es un listado donde se especifica al compilador cuáles programas deben compilarse del paquete, llámense: nodos, mensajes, servicios, librerías, etc.
- **Archivo manifest.xml.** Aquí se encuentra información referida a la descripción del paquete como, por ejemplo: función, autor, licencia, etc. Así mismo, contiene un listado de los paquetes de los que depende el proyecto. Por lo general, este archivo se encuentra como: *`package.xml`*.

2.2.3 Conceptos básicos de ROS

ROS implementa un modelo de programación concurrente [8], multi-propósito y concurrente multi-hilo. El procesamiento de datos en ROS está formado por una red de nodos, la cual se denomina gráfico de procesamiento. Los conceptos más sobresalientes dentro de este gráfico son: Nodos ROS, Master, Servidor de parámetros, Mensajes, Topics, Servicios y Bags, ver figura 5. Estos conceptos en conjunto con los paquetes relacionados con la comunicación de ROS y las bibliotecas de cliente principales, como ***roscpp*** y ***rospython***, forman parte de un

stack llamado ***ros_comm***. Este stack tiene herramientas como: **rostopic**, **rosparam**, **rosservice** y **rosnode**; los cuales son de ayuda para el análisis de conceptos que se encuentran en el gráfico de procesamiento. Así mismo, el stack **ros_comm** contiene paquetes de comunicación de ROS denominados **ROS Graph Layer.**

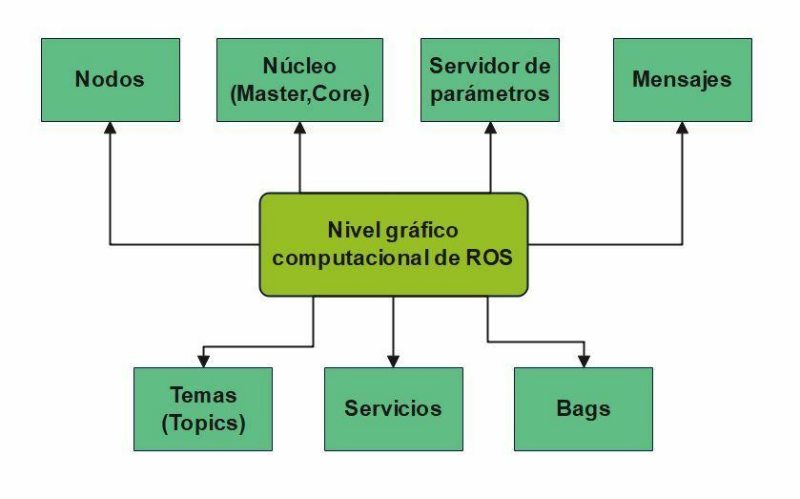


Figura 5. Elementos que componen el nivel gráfico computacional de ROS. Con referencia [9].

De la figura 5, los elementos más sobresalientes que componen la ROS Graph Layer son:

- **Nodos.** Son subsistemas del proyecto robótico que realiza procesamiento de datos, para reducir la complejidad del proyecto, aumentando la capacidad de depuración, esto es referido a que si el robot tiene una falla se puede ir revisando cada nodo hasta encontrar la falla. Por lo tanto, un nodo es un subsistema que realiza procesamiento de datos, el cual tiene la capacidad de enviar y recibir información a otros nodos. Tomar en cuenta que cada nodo debe de tener un nombre único, ya que es el que lo identifica y lo

distingue de los otros nodos que conforman el proyecto. Por ello, un robot puede tener muchos nodos de los cuales cada uno puede tener un sensor para cumplir el objetivo del robot, ya sea para enviar o para hacer el procesamiento de los datos obtenidos. Los nodos pueden estar codificados, ya sea en C++ (roscpp) o en Python (rospy), ya que ROS ofrece compiladores para el uso de esos tipos de lenguaje de programación. La comunicación que se realiza entre nodos es por medio de topics, servicios y parámetros, más adelante se hará mención sobre los topics.

- **Master.** Es el núcleo del sistema, con el nombre de "*roscore*". Siendo este un nodo, tiene la función parecida a un servidor DNS, ya que lleva el registro del nombre de todos los nodos existentes, así como el registro de los servicios y parámetros. Cabe destacar que este nodo **Master**, es de vital importancia, ya que sin él no existiría la comunicación entre los nodos y transferencia de datos entre ellos.

- **Topics.** Los nodos se comunican e intercambian mensajes por medio de topics (temas), que son denominados "buses". Cuando un nodo se encuentra enviando un mensaje, se dice que el nodo se encuentra publicando en un topic y cuando un nodo recibe un mensaje a través de un topic, se dice que el nodo se está suscribiendo a un topic. Cabe mencionar que tanto el nodo publicador como el suscriptor no conocen su existencia, dado que es el nodo

maestro quien hace el intercambio de mensajes. El manejo de la información permite el caso en el que varios nodos se encuentren publicando o suscribiendo a un mismo topic, siempre y cuando haya concordancia con el tipo de topic. Por ello, al declarar el topic es importante definir el tipo de topic que se va a usar, así como definir un nombre único con el que cualquier nodo pueda suscribirse o publicar datos. Por lo tanto, la comunicación entre nodos sería de la siguiente manera: el nodo debe de comunicar al **Maestro** (nodo maestro o core de ROS) que va a publicar o suscribirse a un topic, ya sea para enviar o recibir información. Después, el nodo comienza a publicar la información a través de un topic, mientras que el otro nodo deberá suscribirse al topic para así recibir la información. La figura 6 muestra un ejemplo de este proceso. Se trata de un grafo que se puede obtener con la herramienta de ROS llamada **rqt**. Los óvalos simbolizan los nodos, los rectángulos representan los topics y las flechas la dirección de comunicación, estas son unidireccionales.

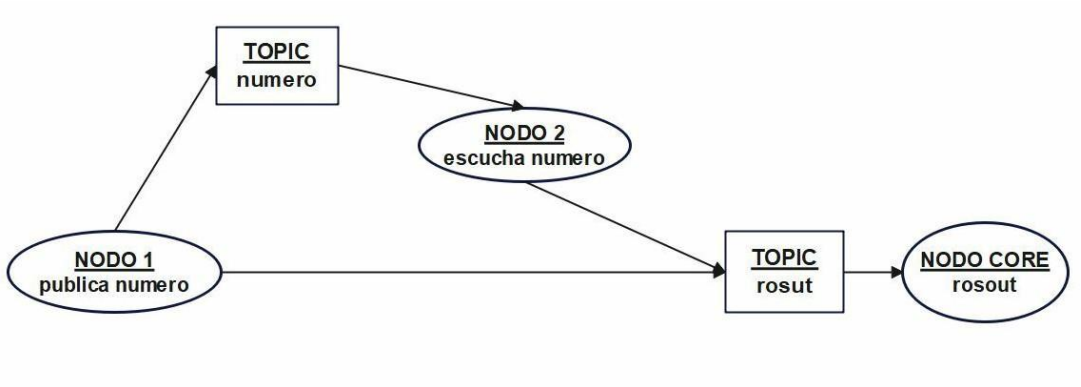


Figura 6. Grafo de un ejemplo de proyecto de ROS.

- **Mensajes.** La manera en que un nodo se comunica con otros es publicando un mensaje por medio de un topic, mientras que el otro simplemente se suscribe al topic. Los mensajes son datos con una estructura simple de los cuales ROS lee dos tipos, que son: *datos primitivos estándar y matrices de tipo primitivos*. Cabe mencionar que los datos son declarados en forma de lista, declarando el tipo y el nombre, separados por un espacio. ROS tiene un paquete llamado *std*_msgs, en el que se encuentran definidos los tipos de datos compatibles.

2.2.4 Integración en ROS de un ambiente gráfico usando RVIz y Gazebo

Las herramientas graficas más comunes que ayudan a la simulación para observar gráficamente el comportamiento de un proyecto en un entorno son los siguientes: RViz y Gazebo.

2.2.4.1 RVIz

RViz es una herramienta de visualización que está dentro de los stacks de ROS, para ser más específicos, en el **stack de visualización**. Con RViz se pueden visualizar modelos de robots en 3D descritos en un archivo URDF, por sus siglas en inglés de *Unified Robot Description Format*, y en ese mismo mostrar datos y tipos de mensaje de ROS obtenidos por sensores, como captar o transmitir lo que capta una cámara y realizar escaneos con sensores láser.

2.2.4.2 Gazebo

Por otro lado, Gazebo es un software de simulación de sistemas que tiene como complemento principal la integración de cuatro motores de físicas. Este simulador tiene la capacidad de simular múltiples robots de manera eficiente y precisa, todo esto en un entorno complejo, llámese interiores y exteriores. Además de contar con un conjunto de motores de físicas completo, también cuenta con interfaces programables y una alta calidad de gráficos. Este software está disponible de manera gratuita para Windows, Mac y Linux en sus distribuciones: Ubuntu, Debian y Fedora. En la figura 7 se muestran las características principales de Gazebo.

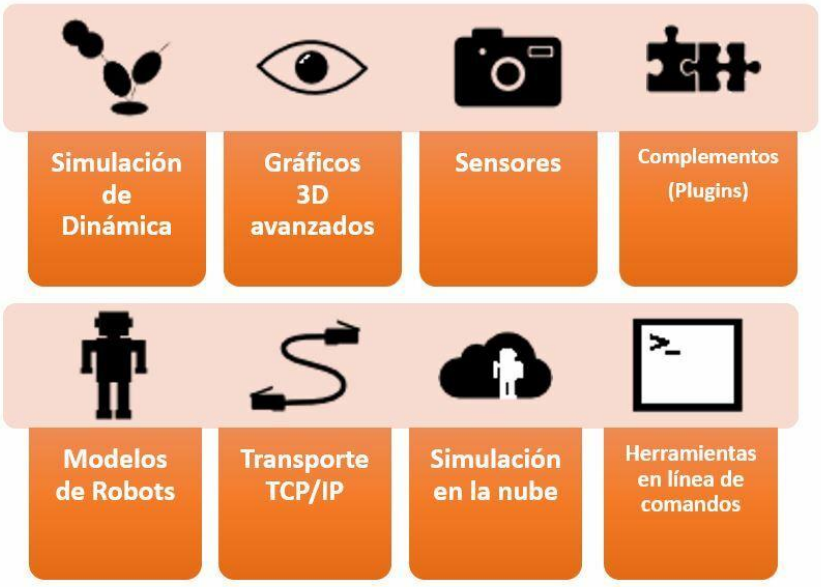


Figura 7. Características de Gazebo. Con referencia [10].

La simulación dinámica esta referida al uso múltiple de motores de físicas, Gazebo cuenta con los siguientes motores: ODE, Bullet, Simbody y DART. Contiene graficas avanzadas en 3D porque este es uno de los simuladores más realistas en cuanto

al entorno y al robot a modelar. Este simulador cuenta con el manejo de plugins que pueden personalizarse, estos son de gran ayuda para el control de robots, sensores y el entorno. Gazebo tiene gran variedad de ejemplos de modelos robóticos, de los cuales el usuario puede crear el propio en formato SDF. Este simulador se puede comunicar por TCP/IP con servidores remotos para realizar alguna simulación, su comunicación se encuentra administrada por Boost ASIO y con ayuda de Google Protofus se encarga de enviar mensajes. Con CloudSim se puede ejecutar Gazebo desde un navegador de Internet, esto es gracias a los servidores GzWeb y Amazon AWS. Y para aquellos que prefieren ejecutar funciones por medio de líneas de comandos, en Gazebo es posible ya que cuenta con un inmenso compilado de líneas de comandos.

Al momento de desarrollar una simulación, existen elementos que son de suma importancia y que son necesarios recordar, algunos de ellos son:

- **Archivo World.** Este archivo tiene extensión **.world** y en su contenido se describen elementos que describen el entorno en el que se encuentre el robot a modelar. Aquí se definen: la iluminación, el tipo de suelo, propiedades físicas del entorno, objetos estáticos, sensores y robots. También se pueden definir escenarios hechos y modelados, por ejemplo: un cuarto amueblado, bosques o gasolineras; estos y más modelos se encuentran disponibles en la red, que son compartidos por la comunidad.
- **Archivo Model.** Archivo con contenido en formato SDF, similar a un archivo **.world**. Contine información que describe un entorno con sus propiedades

físicas. Su finalidad es simplificar el contenido extenso de los archivos **.world** para describir un entorno y que estos modelos sean reutilizados por los usuarios. Gazebo cuenta con gran número de modelos disponibles para la comunidad.

- **Servidor gzserver.** Es el encargado de leer el archivo “world” (mundo) y enseguida se comienza a simular el mundo, esto es con ayuda de los motores de físicas. Este servidor no proporciona una interfaz gráfica. Para ejecutar el servidor simplemente en una terminal se ejecuta el comando:

  ```
  gzserver nombre_archivo_world.world
  ```

  donde `nombre_archivo_world` hace referencia al archivo **.world** que ha creado el usuario.
- **Cliente gzclient.** Este servidor se conecta con el servidor **gzserver** que se encuentra activo para que de manera grafica se visualice la simulación del paquete o proyecto. Por lo tanto, gzclient provee una interfaz gráfica (GUI), la cual cuenta con una herramienta para tener la posibilidad de modificar la simulación.
- **Plugins.** Proporcionan una manera fácil de comunicarse con Gazebo, estos mismos pueden ser ejecutados en líneas de comando o especificarse en los archivos que describirán al robot (esto más adelante se explicará a detalle).

2.3 Integración de ROS-Gazebo

Dado que ROS y Gazebo hacen una dupla muy eficaz para el modelado, diseño y desarrollo de proyectos robóticos, se crea un conjunto de paquetes o metapaquetes que son de suma importancia para realizar una simulación en Gazebo por medio de mensajes, servicios y reconfiguración dinámica de ROS. El metapaquete se llama gazebo_ros_pkgs, para mayor información se recomienda [11].

Figura 8. Estructura de MetaPaquete gazebo_ros_pkgs. Con referencia [11].

Como se observa en la figura 8, gazebo_ros_pkgs está compuesto por tres categorías de paquetes: Paquetes ROS, Plugins Gazebo y Paquetes de simulator_gazebo. A continuación, un breve resumen del contenido de los paquetes más importantes.

- **gazebo_ros.** Este paquete proporciona funcionalidades entre ROS y Gazebo. Aquí también yacen dos plugins muy importantes que son *gazebo_ros_api_plugin* y *gazebo_ros_paths_plugin*. Por otro lado, contiene archivos que son herramientas como las mencionadas anteriormente, por ejemplo: *gzclient*, *gzserver*, *spaw-model*, entre otros.
- **gazebo_msgs.** Contiene archivos que definen la estructura y los tipos de contenido de cada mensaje o servicio que se realice entre ROS y Gazebo.
- **gazebo_plugins.** Aquí se proporciona una ampliación dedicada a los plugins que van enfocados a los actuadores, sensores o reconfiguraciones. Estos plugins que son agregados a un modelo de robot proporcionan una funcionalidad. Los más relevantes son: *gazebo_ros_diff_drive* permite controlar el modelo indicando su velocidad lineal y angular; *gazebo_ros_joint_state_publisher* proporciona la posición de las articulaciones del modelo.
- **gazebo_dev.** Este paquete funciona como intermediario entre ROS y Gazebo. Está conformado por archivos **.CMake**, que proporcionan la configuración de Gazebo con respecto a la distribución de ROS.

Con todos los paquetes anteriores es como se tiene la comunicación y un trabajo conjunto entre ROS y Gazebo.

Cada vez que se ejecuta Gazebo y ROS por medio del paquete gazebo_ros, se crean un conjunto de topics y el nodo **/gazebo**, los cuales se comunican con ROS. Todos estos elementos se crean con ayuda del plugin gazebo_ros_api_plugin.

2.4 Descripción del robot en Gazebo

Una vez creado el espacio de trabajo, pasaremos a modelar al robot. El modelo del robot puede crearse de dos formas: la primera es modelar el robot con formato **SDF** y la segunda es modelar mediane un paquete URDF.

La primera es la forma común de modelar cuando solo se trabaja con la herramienta Gazebo. El formato SDF debe colocarse en el directorio ***~/.gazebo/models***. La segunda opción, es la más adecuada para modelar un robot cuando se utiliza ROS-Gazebo, ya que se modela como un paquete **URDF***.* Es un formato nativo de esta herramienta, la cual ayuda a describir todos los elementos de un robot, la descripción se encuentra como un archivo XML. La estructura del robot será en forma de árbol, por lo que el robot estará conformado de enlaces (links) y uniones (joints). El URDF está compuesto por un grupo de etiquetas de XML especiales. El modelar un robot de esta manera tiene desventajas, por ejemplo, el paquete URDF contiene todo el código que describe al robot en un solo archivo. Con lo anterior es mejor modelar al robot como un conjunto de archivos **xacro**. El **Xacro** significa XML Macros, xacro es similar a un paquete URDF, pero incluye más complementos [9,

p. 74]. Se utiliza esta manera de modelar porque se tiene la ventaja de hacer un archivo URDF más corto y que facilita la descripción de robots complejos. Uno de los atributos más sobresalientes de usar xacro es que si existe un error en el archivo XML cuando se construye el archivo URDF, en la terminal se muestra en cuál línea está el error. De manera resumida el usar xacro ayuda a modelar el robot de forma simplificada y eficiente. Para mayor información consultar [9, p. 86]. La extensión de los archivos es **.xacro** y siguen usando formato XML.

2.4.1 Lenguaje XML

El lenguaje de programación XML, de sus siglas en inglés de eXtensible Markup Language, fue desarrollado por W3C (World Wide Web Consortium), generalmente es usado en diseño y desarrollo de páginas web ya que este lenguaje tiene soporte a bases de datos. Un documento XML se caracteriza por ser jerárquico y es legible tanto por una computadora como por un usuario con ayuda de un editor de textos. Dado que un archivo XML contiene la información de manera jerárquica y empaquetada, toda esta información se encuentra ordenada de manera específica, es por ello que la estructura del contenido de un archivo XML está definida de la siguiente manera.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nota>
    <para>Lety</para>
    <de>Martin</de>
    <titulo>Recordatorio</titulo>
    <contenido>A las 2:00 pm en la puerta del auditorio de la
    universidad</contenido>
 </nota>
```

Se observa que, de manera general, el contenido de un archivo XML está compuesto por un **prólogo** y el **cuerpo**.

El **prólogo** define el tipo de archivo, la versión, el autor, entre otros. Cabe mencionar que esta línea no es obligatoria, pero se recomienda colocarla para dar fiabilidad de la información contenida.

Por otro lado, el **cuerpo** contiene información relevante del archivo XML. Todos los documentos XML tienen un cuerpo con una estructura llamada “árbol”, esta estructura contiene un elemento “raíz”, del cual los demás elementos se encuentran contenidas en ella. En el cuerpo se definen elementos, los cuales tienen etiquetas que los identifican y además de eso, cada uno de ellos pueden contener varios atributos que corresponden a sus nombres y valores que los relacionan, por ejemplo:

<universidad pais = “CDMX”> UACM **</universidad>**

Con lo anterior, se observa que siempre en XML se tiene una **etiqueta inicio** en el que lleva el nombre del elemento, todo esto dentro de los símbolos **<>**, dentro de estos signos también suelen definirse los atributos del elemento y el valor del atributo se encuentra entre comillas. En seguida se encuentra el contenido del elemento y para cerrar se coloca una **etiqueta fin**. Esta etiqueta se encuentra formada por los símbolos **<>** y dentro de ella va el símbolo **/** y el nombre del elemento.

En los archivos XML se pueden agregar comentarios ya sea para definir o explicar alguna línea. Para realizar un comentario simplemente se escribe dentro de los símbolos **<>** un signo de exclamación seguido de dos guiones **<! --** y **-->**.

2.4.2 Etiquetado en el modelado de robots con archivos .xacro

Antes de realizar una descripción del robot se deben entender los siguientes conceptos, los cuales se utilizan en el etiquetado para la descripción del proyecto:

- **<xacro:property>:** generalmente se usa cuando se modela con xacro. Esta etiqueta ayuda a declarar constantes o propiedades del robot a modelar, ya sean valores o dimensiones que describen al robot. El beneficio de usar esta etiqueta es solo mandar a llamar por medio de ésta las constantes o dimensiones, si se requiere una modificación solo hacerlo desde la etiqueta **property** y no en cada línea que aparezca la constante. En el código siguiente, se observa cómo se declara la constante π y las dimensiones de un objeto a modelar en 3D, esto se hace usando la etiqueta **xacro:property**.

```
<xacro:property name="PI" value="3.1415926535897931" />

<xacro:property name="bloqueLarX" value="0.44" />
<xacro:property name="bloqueAnchY" value="0.35" />
<xacro:property name="bloqueAltZ" value="0.2" />
<xacro:property name="bloqueMass" value="25" />
```

  Por otro lado, en el código siguiente se aprecia cómo se utilizan esas dimensiones del objeto y estas son invocadas de la siguiente manera: **${nombre de la propiedad}**, por ejemplo, **${bloqueLarX}.**

```
<link name="bloque_ev3">
    <visual>
      <origin xyz="0 0 ${ruedagdRadio}" rpy="0 0 0"/>
      <geometry>
        <box size="${bloqueLarX} ${bloqueAnchY} ${bloqueAltZ}" />
      </geometry>
      <material name="white"/>
    </visual>
```

- **<xacro:include>:** está etiqueta se utiliza para importar contenido de otro archivo .xacro o .gazebo. La finalidad es no saturar de código un solo archivo con la descripción del robot. Por ello, es mejor dividir el modelado del robot en varios archivos .xacro y al final fusionarlos utilizando esta etiqueta. En el siguiente código se muestra un ejemplo.

```
<xacro:include filename="$(find ev3_description)/urdf/ev3.gazebo" />
<xacro:include filename="$(find ev3_description)/urdf/materials.xacro" />
<xacro:include filename="$(find ev3_description)/urdf/macro.xacro" />
```

El código anterior muestra la forma en que se ocupa la etiqueta **include**, esta se declara de la siguiente manera:

<xacro:include filename = "$ (find nombre_del_paquete_description**)** ruta de donde se encuentran los archivos a incluir, en este caso la carpeta **urdf /** archivo a incluir **"/>**

- **<xacro:macro>:** por lo general esta etiqueta ayuda a reducir la complejidad de la descripción del robot. Usualmente los macros se usan para definir las propiedades inerciales de partes geométricas del robot y también la descripción de partes simétricas del mismo. Estas macros pueden estar

definidas en otros archivos .xacro y cuando se requieran solo se invocan. Un ejemplo de ello se aprecia en el siguiente código[6].

```
<?xml version="1.0"?>
<robot name="mybot" xmlns:xacro="http://www.ros.org/wiki/xacro">
<!-- Put here the robot description -->

  <xacro:macro name="box_inertia" params="m x y z">
      <inertia  ixx="${m*(y*y+z*z)/12.0}" ixy = "0.0" ixz = "0.0"
                iyy="${m*(x*x+y*y)/12.0}" iyz = "0.0"
                izz="${m*(x*x+z*z)/12.0}"/>
  </xacro:macro>

  <xacro:macro name="cylinder_inertia" params="m r h">
    <inertia  ixx="${m*(3*r*r+h*h)/12.0}" ixy = "0.0" ixz = "0.0"
              iyy="${m*(3*r*r+h*h)/12.0}" iyz = "0.0"
              izz="${m*r*r/2.0}"/>
  </xacro:macro>

  <xacro:macro name="sphere_inertia" params="m r">
      <inertia  ixx="${2*m*r*r/5}" ixy = "0" ixz = "0"
                iyy="${2*m*r*r/5}" iyz = "0"
                izz="${2*m*r*r/5}" />
  </xacro:macro>

  <xacro:macro name="wheel" params="lr tY">


  <link name="${lr}_wheel">
     <collision>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <geometry>
           <cylinder length="${wheelWidth}"
radius="${wheelRadius}"/>
        </geometry>
     </collision>

     <visual>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <geometry>
           <cylinder length="${wheelWidth}"
radius="${wheelRadius}"/>
```

[6] Código con referencia a [12], se corrige en la macro name = “box inertia” la definición de iyy que tiene referencia en [13]

```
        </geometry>
        <material name="black"/>
     </visual>

     <inertial>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <mass value="${wheelMass}"/>
        <xacro:cylinder_inertia m="${wheelMass}" r="${wheelRadius}"
h="${wheelWidth}"/>
     </inertial>
  </link>
```

En el código anterior se aprecia el ejemplo del contenido de un archivo **macro.xacro** y el uso de la etiqueta **<xacro:macro>** para definir la propiedad de inercia de una caja y una esfera. En ambos casos el código de inercias se escribe de la siguiente manera:

```
<xacro:macro name= nombre_de_la_macro param=
“los_parámetros_del_código_de_inercia” >
   <intertia   -------Definición de inercia--------- />
</xacro:macro>
```

Para el caso de las partes simétricas del robot, en este caso `wheel`, que corresponde a las llantas del robot, también el código está entre las etiquetas **xacro:macro**. A continuación, un ejemplo de esto[7]:

```
<link name="caster_wheel">
  <collision>
     <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
     <geometry>
        <sphere radius="${casterRadius}"/>
     </geometry>
  </collision>

  <visual>
    <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
    <geometry>
```

[7] Código con referencia a [12]

```
      <sphere radius="${casterRadius}"/>
    </geometry>
    <material name="red"/>
  </visual>

  <inertial>
    <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
    <mass value="${casterMass}"/>
    <xacro:sphere_inertia m="${casterMass}" r="${casterRadius}"/>
  </inertial>
</link>


<xacro:wheel lr="left" tY="1"/>
<xacro:wheel lr="right" tY="-1"/>
```

En el código anterior se define una parte del robot que tiene geometría esférica, más adelante se invoca la propiedad de inercia de la esfera, la cual se encuentra ya definida en otro archivo .xacro. Para llamar a esta macro se escribe lo siguiente:

<xacro: `el_nombre_de_la_macro` parámetros que ocupa la macro (en este caso **m** que corresponde a la masa del objeto y **r** al radio del objeto) ***/>***

En las siguientes líneas se invoca la macro de `wheel`, utilizando el mismo formato para llamar a la macro. El objetivo es usar el mismo código para modelar la rueda derecha e izquierda y así no generar un código que modele a cada una de las llantas.

Por lo general, los robots modelados de esta manera contienen los siguientes archivos: `nombre_del_robot.xacro`, `macros.xacro`, `materials.xacro` y `nombre_del_robot.gazebo`. Esto solo es un ejemplo de un robot simple ya que,

entre más complejo sea el robot, mayor será el número de archivos dentro del paquete URDF y en ocasiones, para dar una mejor organización, el robot se divide en módulos por cada parte del robot. Cabe mencionar que la instrucción `include` manda a llamar otros archivos xacro para el modelado del robot.

La estructura básica del contenido de los archivos anteriormente mencionados es de la siguiente manera:

```
<?xml version="1.0"?>
<robot name="nombre_del_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
          <!-- Colocar la descripción del robot -->
</robot>
```

La primera línea especifica el lenguaje que usaremos, en este caso es XML. La siguiente línea define el nombre del robot, se recomienda que sea el mismo nombre del paquete que se crea para modelar al robot. Seguidamente la línea: **`xmlns:xacro="http://www.ros.org/wiki/xacro`**, se escribe por que se modela al robot usando xacro. Por último, se cierra esta estructura con la etiqueta **`</robot>`**. Cabe mencionar que la descripción del robot, llámese: propiedades físicas, dimensiones, enlaces, articulaciones, transmisión, entre otras; van entre las etiquetas **`</robot name = ...>`** y **`</robot>`**. Esta estructura básica debe de estar en todos los archivos que estén dentro de la carpeta URDF, así como en los 4 archivos anteriormente mencionados. A continuación, una descripción del contenido y la relación entre los archivos: `nombre_del_robot.xacro, macros.xacro, materials.xacro y nombre_del_robot.gazebo.`

- **`nombre_del_robot.xacro:`** aquí se definen propiedades físicas y visuales de partes del robot y sensores, cámaras o dispositivos para recolectar datos

(dispositivos de entrada); así como dimensiones de partes y módulos del robot. Por lo general, en proyectos complejos estos últimos datos no se suelen definir aquí, ya que se usa una herramienta de diseño 3D para modelar. En el caso de ROS hay diferentes opciones para el diseño 3D, uno de los más comunes es COLLADA (COLLAborative Design Activity), que exporta la descripción del proyecto en un archivo con formato .DAE, llamado "mesh" [14, p. 44]. Cabe destacar que en robots complejos suele nombrarse un archivo por cada parte del robot como: `nombre_de_la_parte_del_robot.xacro.`

- **`macros.xacro:`** de igual manera que en el archivo `nombre_del_robot.xacro`, también se definen propiedades y dimensiones, ya sea de partes del robot, sensores o dispositivos de entrada. Pero específicamente se definen las macros para invocarse desde los archivos .xacro. Por lo general aquí se suelen definir los cálculos para la matriz rotacional inercial[8], pero eso ya depende del diseñador, porque también se suele definir en el archivo principal de la descripción del robot o en la descripción de la parte del robot.
- **`materials.xacro:`** el contenido de este archivo lleva información acerca de los materiales ocupados para el modelado del robot, en este caso se definen los colores en RGBA, donde el valor de "A" corresponde a la

[8] También conocido como Tensor Inercial de un sólido, es aquel que determina la relación que existe entre el momento cinético de un sólido a un punto y su vector rotacional. Así mismo, para cada sólido definido, el Tensor de Inercia es diferente para cada punto en el espacio.

opacidad, su valor está entre 1 (visible) y 0 (transparente). Por lo tanto, los colores se declararán de la siguiente forma:

```
<material name="orange">
   <color rgba="255 108 10 1"/>
</material>
```

En la primera etiqueta de material se define el nombre del color que va entre comillas, en la siguiente línea va la etiqueta que define el color en valores de RGB y la opacidad con la letra A, por último, se cierra con diagonal y la etiqueta material. En este archivo se definen todos los colores necesarios para modelar al robot, así entonces, `materials.xacro` trabaja conjuntamente con `macros.xacro` y `nombre_del_robot.xacro`. Estos dos últimos archivos contienen el etiquetado **`<visual></visual>`**, aquí se define la etiqueta material donde se especifica el color a utilizar, mismo que ya se definió en RGBA en `materials.xacro`. Lo anterior se puede observar en el siguiente código[9]:

```
<visual>
 <origin xyz="0 0 ${wheelRadius}" rpy="0 0 0"/>
  <geometry>
   <box size="${chassisLength} ${chassisWidth} ${chassisHeight}"/>
  </geometry>
  <material name="orange"/>
</visual>
```

- **`nombre_del_robot.gazebo:`** el código escrito en este archivo es información que lleva el etiquetado **`<gazebo></gazebo>`**, en el cual se encuentra código referente a materiales, propiedades físicas y controladores

[9] Código con referencia a [12]

de sensores, cámaras, entre otros dispositivos de entrada. Cuando se incluyen datos referidos al material o color de las piezas del robot, esto es para visualizar al robot en la herramienta Gazebo. El formato para declarar los colores para los links o piezas del robot se define de la siguiente manera:

```
<gazebo reference="chassis">
      <material>Gazebo/Orange</material>
</gazebo>
```

Se observa que, en la primera línea después de la etiqueta **gazebo**, está el código **`reference= "chassis".`** Aquí se escribe el nombre del link o pieza del robot que se modela, en este caso chassis; al que se le asigna un color.

Por otra parte, se declaran las propiedades físicas del suelo en Gazebo por el que se desplazara el robot en la simulación. En el código que sigue se muestra un ejemplo[10]:

```
<gazebo reference="${lr}_wheel">
  <mu1 value="1.0"/>
  <mu2 value="1.0"/>
  <kp  value="10000000.0" />
  <kd  value="1.0" />
  <fdir1 value="1 0 0"/>
  <material>Gazebo/Black</material>
</gazebo>
```

El código declara las etiquetas nuevas como son: `mu1, mu2, kp, kd` y `fdir1`, mismas que se definen de la siguiente manera [15]:

- `kp`: es la rigidez de contacto.
- `kd`: es la amortiguación de contacto.

[10] Código con referencia a [12]

- `mu1`: es la fricción de la superficie sobre el plano x.
- `mu2`: es la fricción de la superficie en el plano y.
- `fdir1:` es un vector que especifica la dirección de `mu1` con referencia a la colisión.

Por último, se tendrá código referente a los controladores de los sensores, cámaras y demás dispositivos de entrada que se hayan definido en el archivo `macros.xacro` y `nombre_del_robot.xacro`. Además de declarar el controlador, también se especifica que es un plugin, así como parámetros que son declarados de acuerdo con el tipo de complemento a utilizar. En el siguiente código es un ejemplo de ello[11]:

```
<gazebo reference="camera">
  <material>Gazebo/Blue</material>
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller"
filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>mybot/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
```

[11] Código con referencia a [12]

```
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

En el código se observa claramente la estructura en la que entre las etiquetas de **<gazebo></gazebo>**, donde se define todo sobre el dispositivo de entrada llamado “camera”.

Con todo lo anterior se tiene información básica para crear un proyecto robótico a partir de un tutorial o crear un proyecto robótico propio con la herramienta ROS y además crear la simulación del robot con ayuda de Gazebo.

2.5 Comunicación para el sistema robótico

Una vez teniendo los conceptos básicos para simular y crear un proyecto robótico en ROS y Gazebo es momento de dar un repaso a conceptos que son necesarios para la comprensión de la parte más relevante del robot que es la comunicación de éste con sistemas que ayudan a que el robot sea lo más autónomo posible.

El sistema robótico requiere comunicarse con un punto de monitoreo al que envíe los datos recabados durante la misión de exploración. En la figura 9 se muestra un esquema de la comunicación del robot. Es por ello por lo que la primera red de comunicación es una red cableada que se encarga de comunicar el robot EV3 con

la computadora a bordo, en este caso con la tarjeta Raspberry Pi que tiene instalado ROS. La segunda red es la comunicación serial que se encarga de enlazar el módulo de comunicación con la computadora abordo y así sacar los datos obtenidos por el robot. Por último, está la red inalámbrica local (WLAN) para que el módulo de comunicación envié los datos al servidor web y éstos se guarden en la base datos del mismo servidor. También esta última red ayuda a tener una conexión remota con la Raspberry Pi y así no hacer uso de un monitor, teclado y mouse.

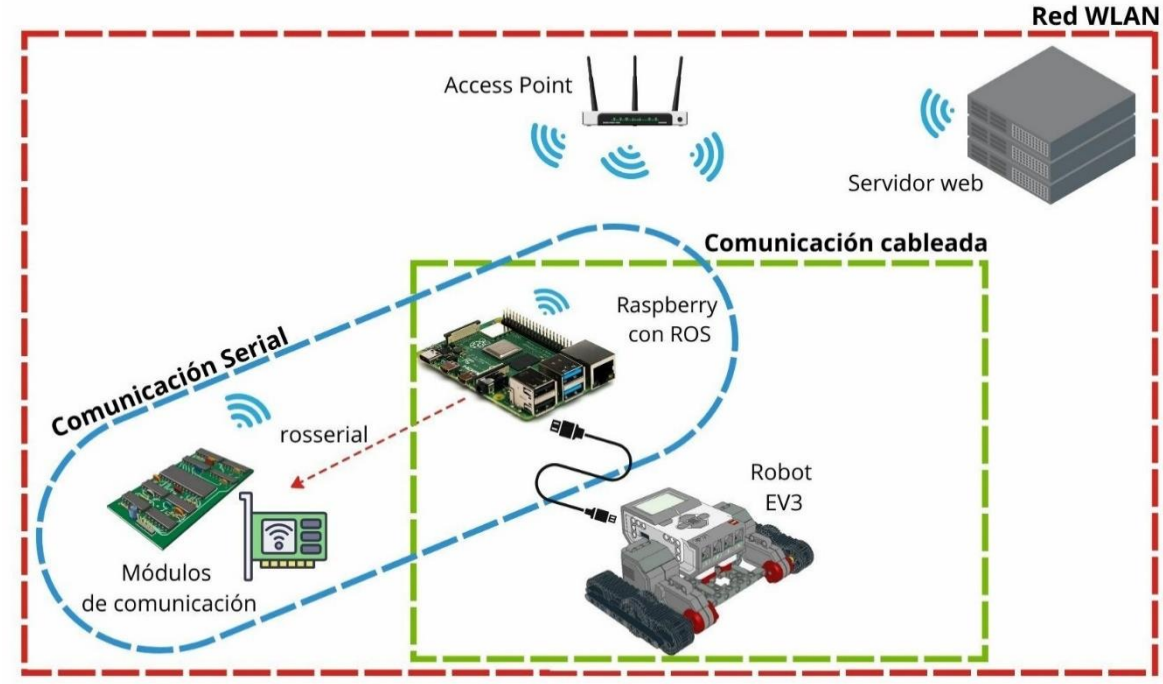


Figura 9. Esquema de la red WLAN para el Sistemas de búsqueda de personas.

Existen varias alternativas de comunicación, pero el presente trabajo se enfoca solo en dos de ellas: redes de área personal (PAN) y redes de área local (LAN).

Con el paso de los años la comunicación ha ido evolucionando y ha sido parte fundamental de la vida de los seres humanos. Es por ello que se han desarrollado

una inmensidad de sistemas de comunicación que van desde los más rudimentarios hasta los más complejos, que ya son conocidos como sistemas de telecomunicaciones. El termino telecomunicaciones se define como toda emisión, transmisión y recepción de signos, señales, escritos e imágenes, sonidos e informaciones de cualquier naturaleza, por cable, radiofrecuencia, medios ópticos u otros sistemas electromagnéticos, esto es de acuerdo a la Unión Internacional de Telecomunicaciones (International Telecommunication Union, ITU) [16]. Estos sistemas siguen evolucionando, desarrollándose y creando nuevos sistemas, los más emblemáticos son: red telegráfica, red telefónica y la red de datos o computadoras. Una red de comunicaciones está definida como la interconexión o enlace por medios alámbricos o inalámbricos entre dos o más sistemas de comunicación. Actualmente las redes de computadoras son los sistemas más usados en todo el mundo, ya que estos ayudan a la comunicación y a la transferencia de datos con la gran mayoría de todos los países.

2.5.1 Redes de datos

De acuerdo con lo que se tiene planeado en el presente trabajo, se va a implementar una red de datos entre el robot y un punto de monitoreo. Este tipo de redes se pueden clasificar por la técnica de transmisión de datos que se use, las cuales pueden ser unicast, broadcast y multicast. En el unicast se define un enlace de punto a punto en el que solo hay un par de computadoras que pueden ser un receptor o un transmisor que envían mensajes cortos (paquetes) por un solo medio

de transmisión. Por otra parte, en una red broadcast las computadoras que forman parte de la red comparten el mismo medio de comunicación, por lo que cuando se envían paquetes a todas les llegan, más sin embargo los paquetes tienen un campo de dirección que especifica a quién va dirigido, por lo que cada equipo revisa el campo de dirección. Si es para éste, entonces procesa el paquete, de lo contrario solo lo ignora. Por último, multicast es una técnica que en una red broadcast a su vez manda información a otro subconjunto de equipos. Cabe mencionar que estas redes de datos pueden usar enlaces (medios de transmisión) cableados o inalámbricos, ambos tienen ventajas y desventajas. Por lo general la elección del medio de transmisión va de acuerdo con las necesidades de los proyectos o de los usuarios.

De igual manera otra forma de clasificar las redes de datos es por su extensión geográfica o, como en algunos textos se menciona, por la escala de la red. Esta clasificación surge de acuerdo con la medición de distancia entre equipos o procesadores de estos y la ubicación de estos dentro de un espacio, figura 10.

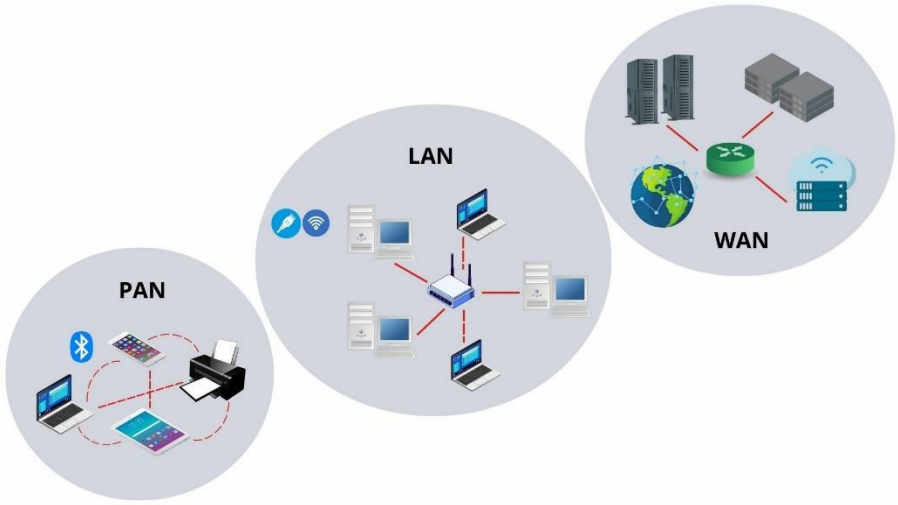


Figura 10. Esquema de tipos de redes de computadoras.

Distancia entre procesadores	Procesadores ubicados en el (la) mismo (a)	Red de datos
1 m	Metro cuadrado	PAN (Personal Area Network)
10 m	Cuarto	LAN (Local Area Network)
100 m	Edificio	
1 km	Campus	
10 km	Cuidad	MAN (Metropolitan Area Network)
100 km	País	WAN (Wide Area Network)
1000 km	Continente	
10000 km	Planeta	

Tabla 1. Clasificación de redes de computadoras [17].

En la figura 10 se observan 4 tipos de redes que existen según su escala, las más relevantes son las redes PAN, LAN y WAN.

Las redes de área personal (Personal Area Network, PAN), de acuerdo con la figura 10, permiten la comunicación dentro de un área de un metro cuadrado. Actualmente un ejemplo de esto es la red inalámbrica que existe entre un smartphone y los audífonos bluetooth. Para este tipo de redes bluetooth se define un nodo maestro y un nodo esclavo, donde el maestro es el que permite la transferencia de

información, define las direcciones de los esclavos, el tiempo en que pueden transmitir, entre otras funciones [17, p. 16]. Además del bluetooth existen otras tecnologías para desarrollar una red PAN como son: IrDA, RFID y Zigbee.

De acuerdo con la figura 10, la siguiente red de datos, de acuerdo con su expansión geográfica, es la red LAN (Local Area Network). En la literatura una red de área local (LAN) se encuentra definida como una red de comunicación que asocia varios dispositivos y facilita un medio por el cual permite el intercambio de datos entre los dispositivos que forman parte de la red LAN [18, p. 17]. Las redes LAN se caracterizan por:

Red LAN	Contar con una cobertura pequeña, a lo mucho un conjunto de edificios cercanos.
	Las velocidades de transmisión de una LAN son mayores en comparación de una red de área amplia (WAN).
	La gestión de la red radica en un único usuario.

Tabla 2. Características relevantes de una red LAN.

De acuerdo con su configuración una red LAN puede ser: LAN cableada y LAN inalámbrica. Para este caso, solo se enfocará en la red LAN inalámbrica, como su nombre lo dice utiliza tecnología inalámbrica y por lo tanto los equipos que componen la red hacen uso de enlaces de radiofrecuencia. La gran ventaja de este tipo de LAN es que permiten la movilidad y la fácil instalación. Dado que en este caso el medio por el que se transmiten las ondas electromagnéticas se propaga por

el aire se puede decir que se usa una topología bus ya que solo existe un solo medio (aire).

Anteriormente, en las primeras redes LAN sólo estaba permitido que uno de los dispositivos (servidor) tuviera la oportunidad de ofrecer recursos a los demás dispositivos dentro de la red, siendo estos últimos solo clientes. Actualmente se ha desarrollado software dedicado a hacer que los dispositivos de las redes puedan tener la característica de ser servidores y clientes al mismo tiempo [19].

La siguiente red de datos es la WAN (Wide Area Network) la cual cubre una gran área geográfica, esto puede ser un país o un continente (redes de computadoras 5ta edicion). Por otro lado, otros definen una WAN como aquella que abarca más de un edificio o simplemente una empresa internacional que cuenta con muchas sucursales en diferentes partes del mundo y que es necesario que se comuniquen entre ellas. Así mismo se puede decir que la red WAN más grande que existe en el mundo es **Internet**, ya que Internet conecta o enlaza muchas redes por todo el mundo entre sí [20]. Para este tipo de red se involucran más dispositivos que son los nodos conmutadores (router) y es que la función principal de las redes WAN es el enrutamiento que tiene que ver con el servicio de conmutación, así como proporcionar servicios de conexión y acceso [21, p. 26].

Dentro de las redes de datos más relevantes que se mencionaron anteriormente, en este trabajo se van a enforcar a dos tipos de redes: redes LAN y PAN, los cuales son soluciones para resolver las necesidades de este proyecto robótico.

2.6 Servidor web

En el campo de las telecomunicaciones, un servidor se encuentra definido como: aquel equipo de cómputo que se encuentra en una red y que ofrece servicios a otros equipos. Dado que la tecnología va en crecimiento y la necesidad de los usuarios es demandante, existen una variedad de servidores que se definen de acuerdo con su función y a su contenido. A continuación, se mencionan los servidores más relevantes: de impresiones, de correo, de fax, de telefonía, proxy, de acceso remoto (RAS), servidor web, de base de datos, de seguridad, de la nube, entre otros [22].

En este trabajo se crea un servidor de base de datos en el que se va a almacenar la información que se obtenga del robot EV3.

Para crear este servidor se usa la herramienta de software llamada XAMPP[12], la cual es gratuita y se encuentra disponible para sistemas operativos Windows, Linux y Mac. Este software es un paquete de aplicaciones que ayudan a la creación de un servidor web local, por lo que al instalar esta herramienta en un equipo de cómputo se cuenta con un servidor web Apache, una base de datos MySQL, PHP y Perl.

Por otro lado, para enviar datos a un servidor web, es necesario entender los siguientes conceptos.

- **HTTP (Hyper Text Transfer Protocol)**. Este es un protocolo que se encarga del transporte de la información entre un servidor y el cliente. El protocolo se

[12] Conjunto de software libre que contine herramientas para la instalación y creación de un servidor web Apache.

enfoca en generar una solicitud y una respuesta, que se llevan a cabo en la capa de aplicación del modelo OSI de redes y, por lo tanto, opera sobre TCP (protocolo de control de transmisión). Aquí también se define el tipo de mensajes que es posible enviar entre el cliente y el servidor y las respuestas que se dan de acuerdo a los mensajes [17, p. 587].

- **URL (Uniform Resource Locator)**. Es un identificador de las páginas web para que sean localizadas en cualquier parte del mundo en Internet. Las URL's se encuentran estructuradas en 3 partes: protocolo, nombre DNS y ruta del archivo a leer de la página web (html, php, etc). Un ejemplo es la URL siguiente, que es un aviso antes de entrar al Sistema de estudiantes de la Universidad Autónoma de la Ciudad de México (UACM).

https://www.uacm.edu.mx/Portals/0/enlaces/mensaje.html

- Aquí se aprecia que el protocolo es **https**, el nombre DNS es **www.uacm.edu.mx** y la ruta es **/Portals/0/enlaces/mensaje.html** siendo **mensaje.html** el archivo a leer.
- **GET.** Este es un método de solicitud para páginas web. GET hace uso de la URL para solicitar la página web al servidor. Aquí los datos son visibles al usuario, cabe mencionar que los datos a enviar son valores de variables que pueden enviarse a una base de datos sin necesidad de hacer un formulario.
- **POST.** También es un método de solicitud enfocado para envió de formularios, éste de igual manera hace uso de una URL para recuperar una

página y a su vez enviar datos al servidor [17, p. 590]. Este tipo de método envía la información de manera oculta al usuario.

- **Estados de respuesta HTTP.** Así como hay métodos de solicitud a un servidor, también existen respuestas a dichas solicitudes, es por ello por lo que existe un listado de respuestas que son informativas. Estas respuestas o códigos de estado están formados por tres dígitos que confirman si la solicitud se atendió y si no fue así se dice el por qué. Dentro de los tres dígitos el primero es utilizado para clasificar al tipo de grupo de respuesta; existen 5.

Estado	Significado	Casos
1xx	Información	100 = el servidor acepta manejar la solicitud del cliente.
2xx	Éxito	200 = la solicitud es exitosa; 204 = no hay contenido
3xx	Redirección	301 = se movió la página; 304 = la página en caché aún es válida.
4xx	Error del cliente	403 = página prohibida; 404 = no se encontró la página.
5xx	Error del servidor	500 = error interno del servidor; 503 = intentar más tarde.

Tabla 3. Respuestas HTTP [17].

De acuerdo a la tabla 3, las respuestas con estado 1xx no son muy utilizadas, los estados 2xx confirman que la solicitud se atendió de manera exitosa y también da información sobre el contenido enviado, los estados 3xx indican al cliente si la página se movió o cambió de URL, para el caso de los estados 4xx proporcionan información sobre un problema con el cliente o con la página web a consultar, por ejemplo, si ya no existe; y por último están los estados 5xx, que indican si el servidor tiene un error interno o presenta una sobrecarga temporal [17, p. 591].

2.7 Comunicación de la Raspberry Pi (ROS) con el servidor web

Por otra parte, se busca la solución para que los datos recolectados del robot sean enviados a un servidor web. La solución por la que se opta es usar dos dispositivos de comunicación. Así mismo, se investiga sobre cómo realizar la comunicación de estos dispositivos con ROS y con el servidor.

2.7.1 Comunicación serial en ROS

ROS tiene un paquete de software que proporciona la comunicación serial. Este ayuda a que exista comunicación entre ROS y un dispositivo con puerto serie o socket. El paquete de nombre **rosserial** [23] contiene un conjunto de protocolos que proporcionan comunicación de ROS con dispositivos externos y viceversa. La función principal de este paquete es convertir los datos enviados entre ROS y un dispositivo para que exista la transferencia de datos. Eso se hace porque ROS usa como medio de comunicación el protocolo TCP/IP y los dispositivos usan UART por su puerto serie. Los datos que se envían y reciben pueden ser mensajes, tópicos o servicios, figura 11.

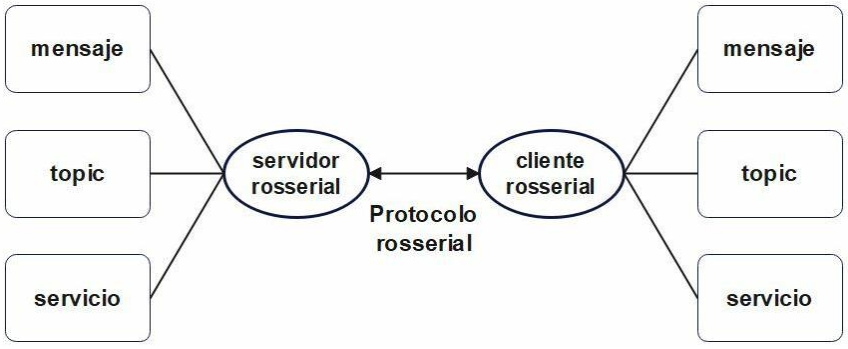


Figura 11. Diagrama sobre el paquete y protocolo rosserial. Con referencia [24].

La figura anterior nos da un ejemplo de cómo es la comunicación entre ROS y un dispositivo. Del lado izquierdo se interpreta que es una PC con ROS instalado, que se define como un servidor rosserial, por otro lado, el dispositivo conectado a la PC toma el papel de cliente rosserial. Por medio del paquete y protocolo rosserial es que es posible el envío de datos entre una PC con ROS y un dispositivo externo.

El servidor rosserial es un nodo de ROS que permite la comunicación con dispositivos agregados usando el protocolo rosserial, por su lado, el servidor necesita herramientas para decodificar los mensajes, los tópicos y servicios que se estén transmitiendo. Estas herramientas se implementan en los siguientes paquetes: *rosserial_python*, *rosserial_server* y *rosserial_java*. De estos paquetes, el más recomendado para el manejo de datos es rosserial_pyhton y es el que se implementa en el presente trabajo.

Por otra parte, el dispositivo externo hace uso de la biblioteca rosserial_client para que el dispositivo se convierta en un cliente. La biblioteca rosserial_client está conformada por un conjunto de sub-bibliotecas que facilita el trabajo con diversas plataformas con las que cuente el dispositivo externo. La biblioteca que se ocupa en este trabajo es rosserial_arduino, la cual ayuda a que sea posible programar el dispositivo desde Arduino IDE y que ROS pueda comprender esta comunicación sin dificultad. Cuando el dispositivo toma el papel de cliente rosserial, este obtiene la cualidad de funcionar como un nodo más de ROS, y a su vez puede suscribirse y publicar en otros nodos.

2.7.2 Comunicación con módulos Xbee Serie 1 Pro

Específicamente para la librería o paquete rosserial_xbee que se encuentra disponible para ROS Noetic, los módulos compatibles son solo los Xbee Serie 1. Los módulos Xbee son fabricados por la empresa Digi International. Estos módulos permiten conexión inalámbrica mediante radiofrecuencia y utilizan el protocolo de red IEEE 802.15.4 y la especificación Zigbee. Así mismo, estos radios se caracterizan por el bajo consumo de energía, facilidad de realizar conexiones: punto a punto, punto a multipunto y mesh, además de un buen manejo de tráfico alto de datos. Es importante mencionar que dependiendo del tipo de la serie del radio se puede realizar cierto tipo de conexión. Cabe mencionar que los módulos XBee cuentan con antenas, las cuales pueden ser del tipo: Chip Antena, Wire Antena y u.FL Antena. En este trabajo se ocupan módulos de la Serie 1 Pro, que son radios fáciles de usar y se recomiendan para usuarios principiantes, no son compatibles con los módulos serie 2 y solo proporcionan conectividad punto a punto. En la guía práctica se explica a detalle la comunicación entre una PC y una Raspberry Pi por medio de módulos Xbee[13].

2.7.3 Comunicación con tarjeta NodeMCU con modulo ESP8266

Este tipo de tarjetas son de bajo costo en comparación con otras que proveen de

[13] Sección H de la guía práctica se encuentra disponible en: https://drive.google.com/drive/folders/1_SP2056VzYJxuA_QH-YiUsUdoUCIPUYf?usp=sharing

comunicación WiFi, tiene mejores características de comunicación en comparación con tarjetas Arduino y además los NodeMCU son de gran utilidad para proyectos enfocados en Internet de las cosas [25]. Esta tarjeta de desarrollo cuenta con un SoC ESP8266, USB UART CH340 que permite conexión a la PC, cuenta con memoria flash externa de 4 MB, conectividad Wi-Fi a 2.4 GHZ y la característica más sobresaliente es que se puede descargar el nucleo o core de la tarjeta para programar con Arduino IDE. Para este caso se tiene el modelo NodeMCU Rectangular basado en ESP-12E/F.

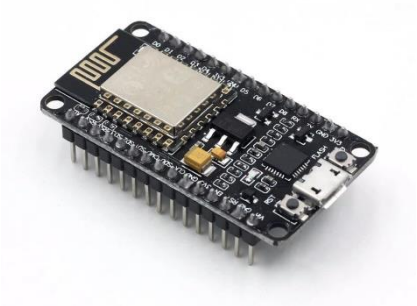


Figura 12. NodeMCU con ESP8266.

El microcontrolador ESP8266 permite diversas soluciones en cuanto a redes Wi-Fi, en este sentido, esta tarjeta puede configurarse en los siguientes modos: Modo de estación (STA), Modo de punto de acceso (AP), Modo combinado (STA) + (AP) y el Modo Wi - Fi apagado. De acuerdo con las características que se proporcionan en el procesamiento de datos y el almacenamiento, el NodeMCU permite trabajar directamente con sensores agregados.

2.8 Comunicación del robot con ROS

Dado que se busca que el robot sea lo más autónomo posible, se propone una minicomputadora que lo controle y vaya con el mismo robot, por ello, se opta por la Raspberry Pi 4, y se investiga sobre cómo es posible la comunicación con el robot.

Con el fin de controlar el robot EV3 desde la Raspberry Pi se tiene una comunicación física entre la Raspberry Pi y el Brick del Lego EV3, por lo que se encuentran conectados por medio de un cable usando los puertos USB. Ahora se necesita de una herramienta a nivel de software para que sea posible la comunicación entre estos dispositivos. En este caso es una biblioteca llamada Remote Python Call (RPyC), es una librería de Python para realizar llamadas a procesos remotos de manera simétrica. Por otro lado, esta biblioteca hace uso de object-proxying, una técnica usada en Pyhton que ayuda a que la realización de procesos remotos sea considerada como procesos locales, esto es para el caso de computación distribuida.

Por lo general las opciones más sobresalientes de uso de esta librería de Pyhton son el control y administración de hardware y software de un punto central, acceso remoto a recursos de hardware, distribución de carga de trabajo entre varios equipos, implementación de servicios remotos (como WSDL o RMI), entre otros usos más [26].

RPyC es compatible con el SO ev3dev, sistema operativo que se usa en el Brick del EV3. Dentro del material disponible de ev3dev se hace presente el uso de la biblioteca RPyC, la cual proporciona los siguientes usos [27]:

RPyC	Ejecutar un script o programa de Python en un dispositivo con ev3dev que a su vez controla a otro dispositivo ev3dev.
	Ejecutar un script en una computadora, ya sea de escritorio o portátil, para controlar un dispositivo con ev3dev.

Tabla 4. Características de la biblioteca RPyC.

Para este trabajo se buscó realizar el segundo escenario. Esto es con el objetivo de usar un código o script más robusto para controlar al robot EV3 y usar los recursos en hardware que proporciona la computadora, ya que si el script se ejecutara en el Brick demoraría ejecutar las tareas asignadas al robot, debido a que las características del Brick no son suficientes para ejecutar un script con procesos extensos.

Cabe mencionar que los equipos de cómputo que ejecuten el script de Python, como los dispositivos con EV3 remotos, deben contar con una dirección IP. Una vez que se cuente con eso se tiene la posibilidad de contar con los siguientes escenarios [27]:

Equipos con dirección IP	Múltiples EV3 con ev3dev en una misma red WiFi
	Una computadora portátil y un EV3 en la misma red WiFi
	Dos EV3 conectados por medio de una conexión bluetooth

Tabla 5. Escenarios para uso de RPyC con equipos que cuentan con una IP.

2.9 Sistema de detección del robot

Para usar una cámara que funcione en conjunto con ROS **Noetic**, se deben realizar los pasos que en este bloque se explican. ROS requiere emplear una biblioteca para el procesamiento y análisis de la información capturada por la cámara, llámense imágenes o fotogramas. Esta biblioteca se llama **OpenCV (Open Source Computer Vision Library)**, como lo dicen sus siglas, es de código abierto y tiene las siguientes características: está basado en lenguaje C++, es compatible con lenguajes de programación como: Python, Java y MATLAB y así mismo esta biblioteca es compatible con SO como: Windows, Linux, Android y Mac OS. OpenCV contiene más de 2,500 algoritmos que se encuentran enfocados al aprendizaje y visión artificial por computadora, es por ello, que los algoritmos más sobresalientes son aquellos que realizan funciones como: detección de rostros, identificación de objetos, ya sean estáticos o en movimiento; rastrear movimiento, unión de imágenes para crear una imagen de alta calidad, modelado en 3D de un objeto, búsqueda de imágenes basado en una base de datos, etc. Dado que es de código abierto, existe una comunidad que lo respalda, así como empresas de alto renombre que hacen uso de esta biblioteca, por lo que es recomendable su uso [28].

Para ilustrar el funcionamiento de OpenCV con ROS se realiza un pequeño ejemplo para comprender, configurar, observar y saber hacer uso de esta biblioteca. Como

primera instancia, se crean dos nodos:

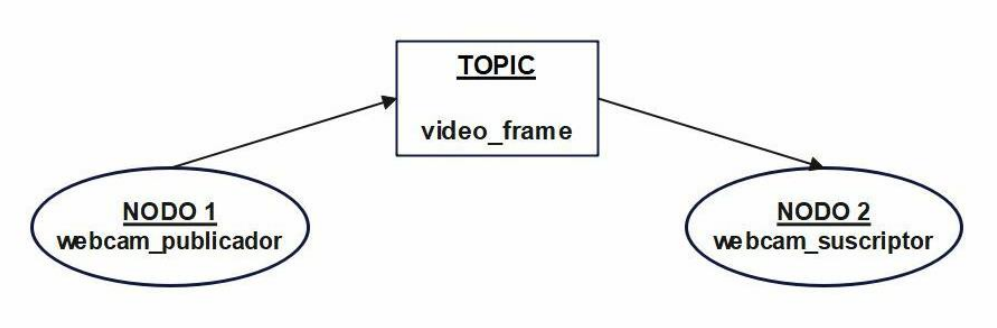


Figura 13. Diagrama sobre los nodos a crear y el topic por el que se comunican.

El nodo publicador, se encarga de publicar las imágenes captadas por la cámara a través de un topic. Por otro lado, el nodo suscriptor, se suscribe al topic e imprime las imágenes captadas por la cámara.

Teniendo el espacio de trabajo llamado *"catkin_et"*, se agrega el paquete que se encargará de capturar video a través de un sensor, que en este caso es una cámara. Este ejemplo se encuentra descrito de manera detallada en la guía práctica[14]

Los conceptos y herramientas descritas en este capítulo fundamentan la implementación de un proyecto robótico, cuyo desarrollo se describe en el capítulo III.

[14] Sección E de la guía práctica.

Capítulo III Implementación

En este capítulo se describe la simulación y la implementación física del proyecto robótico. Para ello, se describe la creación de un espacio de trabajo en ROS, directorio en el que se guardan, se editan y se compilan los scripts que controlan al robot y la descripción de la simulación del robot en Gazebo. Seguidamente se explica la implementación por hardware usando un robot que se diseñó con ayuda del kit Lego Mindstorm Education EV3. Además de los sensores que proporciona el kit, se agrega una cámara que realiza la función de reconocimiento. Todos los datos recolectados se procesan y envían a una tarjeta de desarrollo Raspberry Pi 4, que tendrá instalado ROS para realizar el control del robot. Y por último los datos se envían a una base de datos usando módulos de comunicación compatibles con ROS.

3.1 Robot Lego EV3 Mindstorm

LEGO Mindstorms Education EV3 es un kit educativo, que tiene como objetivo que los usuarios creen, diseñen, analicen y comprendan el funcionamiento de un robot básico de tercera generación, figura 14.

Figura 14. Kit LEGO Mindstorm Education EV3.

El set contiene un Brick, que es el cerebro del robot, el cual cuenta con un procesador ARM9 a 300 MHz, una memoria flash de 16 MB, una memoria RAM de 64 MB. Además, tiene precargado un SO basado en Linux. Cuenta con ranura microSD para la expansión de memoria hasta un máximo de 32 GB, tiene un altavoz, una pantalla de 178 x 128 pixeles, puerto USB 2.0, puerto USB 1.1, 4 puertos de entrada y 4 puertos de salida con conectores RJ12. El Brick puede ser alimentado por 6 pilas AA o por un módulo de batería recargable especial de Lego. También tiene una tarjeta de comunicación inalámbrica Bluetooth (IEEE 802.15.1) y cuenta con la posibilidad de comunicación Wi-Fi (IEEE 802.11) con ayuda de un adaptador que se conecta al puerto USB del Brick, figura 15.

Figura 15. Brick del robot LEGO.

Por otra parte, el kit contiene tres motores de dos tipos. Dos motores son del tipo grande con un torque de rotación de 20 Ncm, funcionan a 160 - 170 rpm, tienen un sensor de rotación incorporado con resolución de 1 grado para un control preciso [29, p. 13]. Por otro lado, cuenta con motores medianos a 240 - 250 rpm, torque de rotación de 20 Ncm, también cuenta con sensor de rotación incorporado, este motor por ser pequeño y ligero tiene la característica de responder más rápido en comparación con el motor grande, figuras 16 y 17.

Figura 16. Motor grande.

Figura 17. Motor mediano.

El set también incluye sensores, en este caso sensor de infrarrojo y sensor de toque. El sensor infrarrojo es digital, el cual detecta la luz infrarroja reflejada en objetos sólidos o puede localizarlas, figura 18. Este se puede configurar en tres modos: modo proximidad, modo de baliza y modo remoto, más información a este respecto en [29, p. 18].

Figura 18. Sensor infrarrojo para robot LEGO EV3.

El sensor de toque es analógico, el cual tiene tres modos: presionado, lanzado o en contacto, figura 19.

Figura 19. Sensor de toque para robot LEGO EV3.

Y por último el kit cuenta con llantas, rines, bandas y piezas de construcción de Lego para crear robots diseñados por Lego o para elaborar un diseño propio.

3.1.1 Software ev3dev

Como se mencionó anteriormente, el Brick del robot EV3 cuenta con un sistema operativo basado en Linux. Para programar al robot, el Brick se conecta a la computadora por medio de los puertos USB y se usa un software que incluye el mismo set. Esta herramienta configura al robot y sus elementos para asignar tareas y dar un cierto comportamiento al robot. El software trabaja por medio de bloques para realizar la programación, se encuentra basado en el software de programación gráfica llamado LabVIEW[15], figura 20. También el software proporciona herramientas para hacer registro, análisis de datos, actualización del hardware, tutoriales para la creación de modelos de robots ya proporcionados por Lego, entre otras. Es un software diseñado para ser muy interactivo.

[15] Más información, visitar https://www.ni.com/es-mx/shop/labview.html

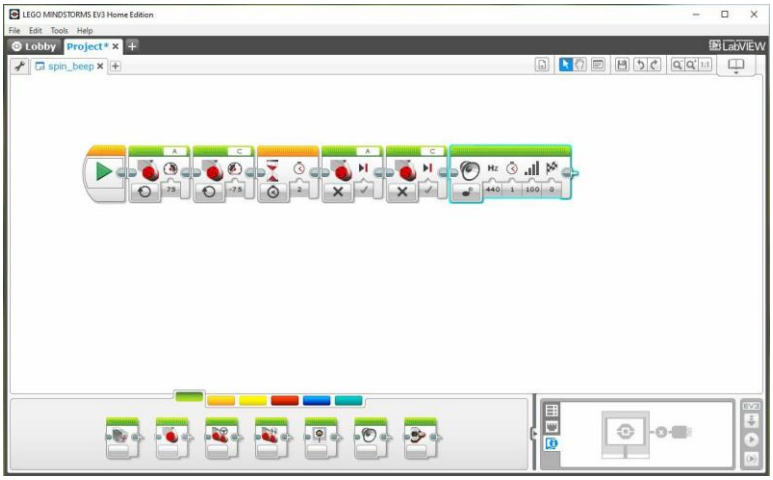


Figura 20. Ambiente grafico de software de programación para robot LEGO EV3.

Para este trabajo, se implementa un SO diferente al que tiene de fábrica el robot EV3. El SO para usar se llama **ev3dev**, es de licencia libre, está basado en Linux en su versión Debian, existe una gran comunidad que lo respalda y por lo tanto tiene una inmensa cantidad de librerías para programar al robot EV3. Por ejemplo, se cuenta con librerías que controlan el hardware del robot, por lo que mover las llantas u obtener información de los sensores se simplifica considerablemente. Este SO permite realizar la programación en lenguajes como Python y C++. Su instalación no es complicada, simplemente se descarga de la página[16] y con ayuda de un software, se guarda el SO en una tarjeta de memoria flash microSD y ésta se inserta en la ranura correspondiente del Brick. Esta modificación del SO no borra datos de fábrica del robot EV3, porque ev3dev inicia desde la microSD.

[16] Para más información consultar la referencia [30]

3.2 Simulación del robot EV3 en Gazebo

Una vez que se sabe lo básico de ROS y Gazebo, se puede comenzar a construir el modelo en 3D del robot EV3, para ello se siguen los siguientes pasos.

Dado que ROS trabaja en conjunto con Gazebo, como primer paso se debe crear un espacio de trabajo, este es el lugar donde se guardan todos los proyectos a desarrollar. Para crea un espacio de trabajo, en una nueva terminal escribir y ejecutar el comando `mkdir` *(make directory*) como sigue:

```
mkdir –p ~/catkin_et/src
```

El comando anterior es para crear la carpeta del espacio de trabajo, la cual se llama **catkin_et**, cada usuario puede nombrar su carpeta de espacio como lo desee. Así mismo, dentro de esta carpeta se va a crear otra carpeta nombrada **src**, ésta contendrá todos los paquetes de los proyectos que se van a desarrollar. Como siguiente paso se ingresa al interior de la carpeta **src** ejecutando el comando `cd` *(change directory)* en la terminal como sigue:

```
cd ~/catkin_et/src
```

Ya que se encuentra dentro de la carpeta **src**, para inicializar el espacio de trabajo escribir y ejecutar el comando:

```
catkin_init_workspace
```

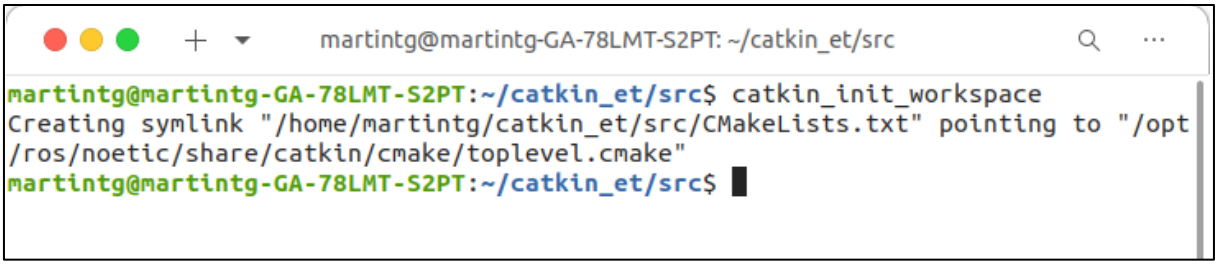


Figura 21. Captura de pantalla de terminal tras ejecutar el comando catkin.

En la figura 21, se observa que al ejecutar el comando se despliega un resultado que indica la creación del archivo `CMakeLists.txt`, este estará registrado en el archivo `toplevel.cmake`, archivo interno de ROS.

Seguidamente se debe hacer una compilación en la carpeta del espacio de trabajo. Con esta compilación se crean dos carpetas muy importantes que son las carpetas **build** y **devel**. Ahora bien, para comenzar la compilación, primero se debe de ubicar en la carpeta del espacio de trabajo, para ello ejecutar:

```
cd ..
```

Ese comando tiene la función de regresar a una ruta anterior del directorio; después se ejecuta el comando:

```
catkin_make
```

Después de ejecutarlo se observa el despliegue de muchas líneas de comandos, (figura 22), esto significa que se está llevando a cabo el proceso de compilación y se están creando las carpetas necesarias.

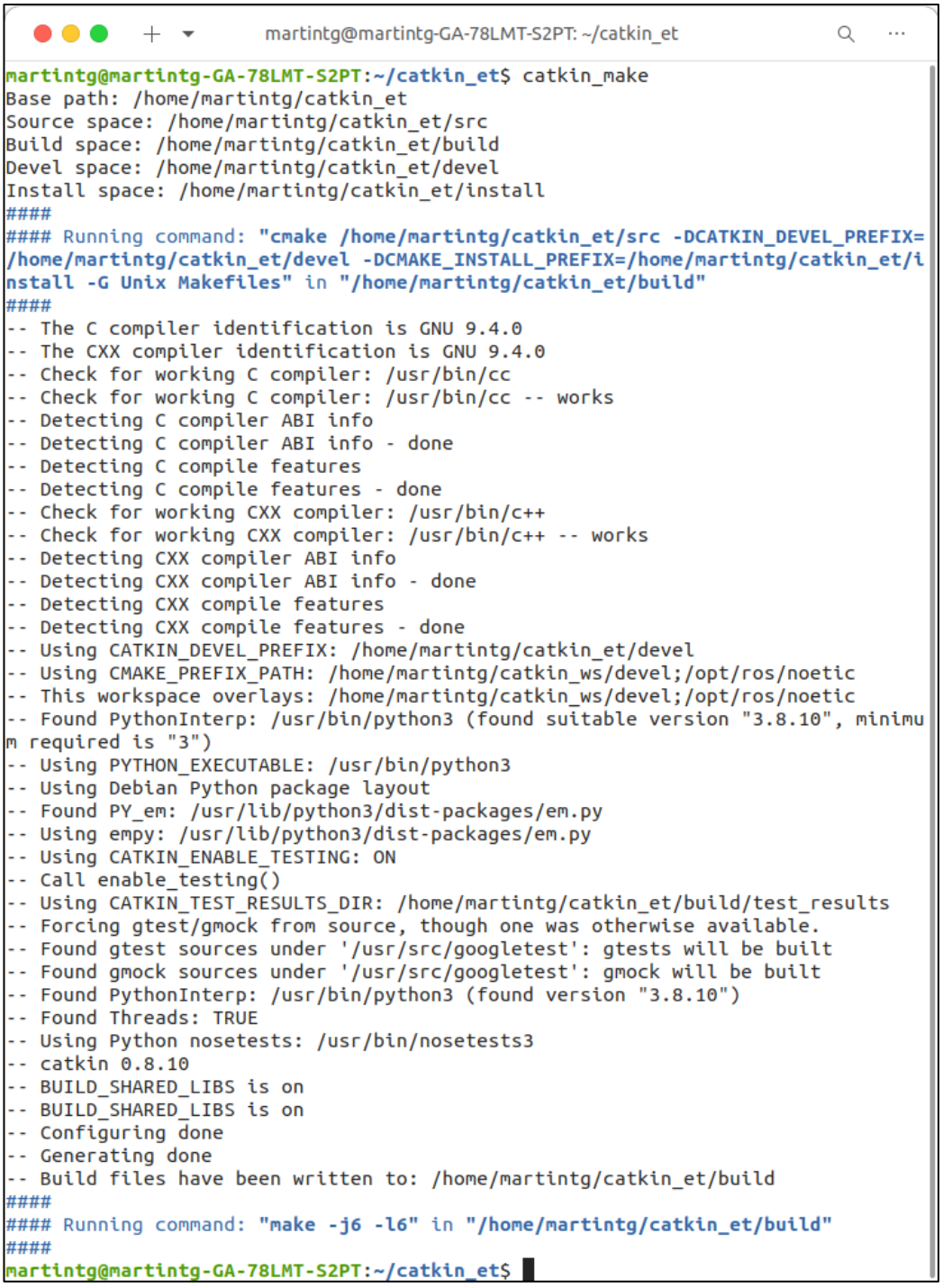


Figura 22. Captura de pantalla de terminal tras ejecutar catkin_make.

Con el proceso anterior se obtiene un espacio de trabajo para desarrollar una infinidad de proyectos enfocados a robots. Una vez que finalice el proceso, se cierra la terminal.

Cabe mencionar que antes de ejecutar, trabajar o simular un proyecto, se debe indicar la fuente de donde se obtendrán los paquetes que se van a ejecutar. Para ello, abrir una nueva terminal y se usa el siguiente comando:

```
source ~/catkin_et/devel/setup.bash
```

Una vez hecho eso se comienza a realizar el modelo en 3D del robot. Los robots para modelar en ROS están conformados por tres paquetes: `nombre_del_proyecto_gazebo`, `nombre_del_proyecto_description` y `nombre_del_proyecto_control`, donde `nombre_del_proyecto` hace referencia al nombre del paquete que haya definido el usuario. A continuación, una breve descripción del contenido de estos paquetes:

- `nombre_del_proyecto_gazebo.` Este paquete contiene archivos que están organizados por directorios, son dos: carpeta launch y worlds. La primera se encarga de iniciar la simulación en 3D del robot y la segunda contiene las descripciones de los entornos donde se simulará el robot.
- `nombre_del_proyecto_description.` Dentro de este paquete se encuentra la descripción del robot en 3D, por ejemplo: sensores, motores, articulaciones, etc. para robots manipuladores.

- `nombre_del_proyecto_control.` Por último, este paquete tiene archivos de programación que interactúan con los elementos del robot y estos se ven reflejados en la simulación.

En esa misma terminal, se dirige a la carpeta del espacio de trabajo y de ahí a la carpeta **src**, ejecutando la siguiente línea:

```
cd ~/catkin_et/src
```

Después ejecutaremos los siguientes comandos que nos ayudan a crear los 3 paquetes mencionados anteriormente. Se coloca línea por línea lo siguiente:

```
catkin_create_pkg myev3_gazebo gazebo_ros

catkin_create_pkg myev3_description

catkin_create_pkg myev3_control
```

Para este caso, el nombre del proyecto es **myev3**. Como resultado se obtiene lo que se muestra en las figuras 23, 24 y 25, esto confirma que se crearon los paquetes con éxito.

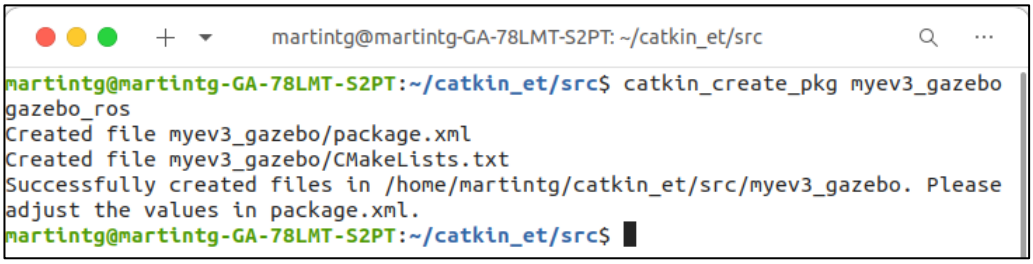


Figura 23. Captura de pantalla de terminal al crear el paquete myev3_gazebo gazebo_ros.

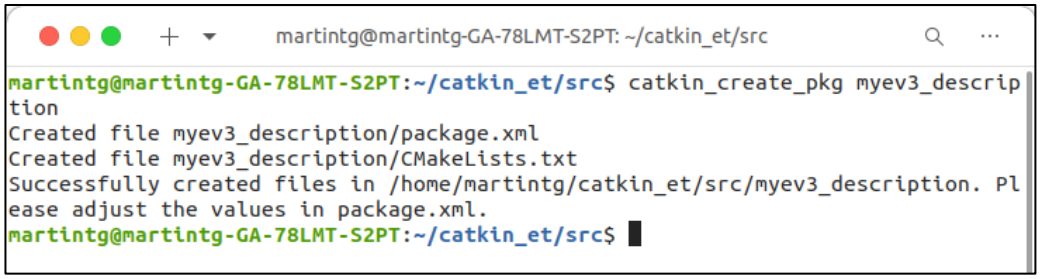


Figura 24. Captura de pantalla de terminal al crear el paquete myev3_description.

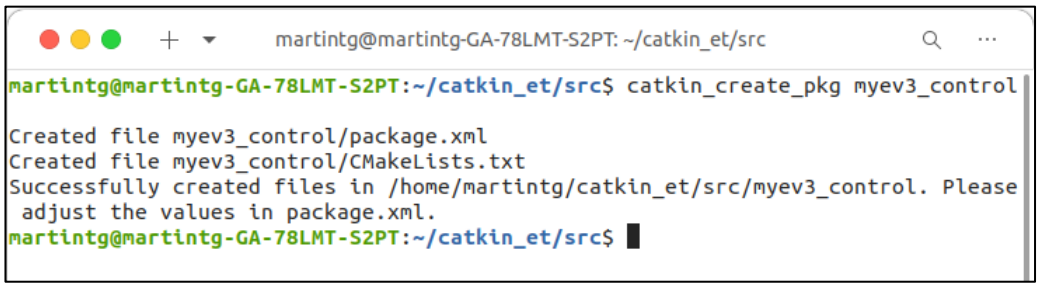


Figura 25. Captura de pantalla de terminal al crear el paquete myev3_control.

Ahora se define el entorno de simulación, lugar donde se encontrará el robot y se le nombra “Mi mundo”.

Ubicar la carpeta del proyecto donde se crean los archivos referentes a los mundos (worlds). En la terminal abierta ejecutar el siguiente comando para ir al directorio:

```
cd myev3_gazebo
```

Ya en el directorio **myev3_gazebo**, crear dos carpetas que se van a necesitar, una es worlds y la otra es launch. Escribir la siguiente línea en terminal para crear las carpetas:

```
mkdir launch worlds
```

Primero se va a crear el mundo, por ello se dirige al interior de la carpeta **worlds**,

```
cd worlds
```

En el interior de la carpeta se crea el archivo que describe y define al mundo. Para esto se requiere de un editor de textos; los más comunes son gedit y nano[17]. En este procedimiento se usará el editor gedit. Se usa la siguiente línea para crear el archivo `myev3.world`:

```
gedit myev3.world
```

Figura 26. Captura de pantalla del editor gedit del archivo myev3.world.

Es importante conocer la estructura básica donde se define al mundo. La siguiente es una estructura básica de un mundo llamado “myworld”.

```
<?xml version="1.0"?>
<sdf version="1.4">
   <world name="myworld">
---información que describe al mundo---
   </world>
</sdf>
```

[17] GNU nano es un editor de textos con la edición básica de texto y con funciones de deshacer/rehacer entre otras. Para más información consultar: https://www.nano-editor.org/dist/latest/nano.html

Lo anterior se parece a la estructura básica para definir y modelar a un robot, como se explica en el Capítulo 2 sección 4, la única diferencia es que todo el contenido donde está descrito el mundo se encuentra entre las etiquetas `<sdf>`, en donde en la etiqueta de apertura se agrega la versión del Formato de Especificación SDF. Así mismo, la otra diferencia es que dentro de la etiqueta **sdf**, se encuentra la etiqueta `<world>`, en donde en la etiqueta de apertura se define el nombre del mundo que se va a crear, todo esto entre comillas. Dentro de estas etiquetas, se agregan todas las características físicas del entorno a definir. Además, se pueden agregar modelos de mundos ya hechos y definidos, así como poner la posición de un objeto dentro del entorno. En este ejemplo solo se agrega definiciones físicas básicas como son un suelo y una fuente de iluminación. Estas características se agregan usando la etiqueta `<include>` y la etiqueta `<URI>`, que sirve como un identificador de recursos o modelos de los cuales ya se encuentran en las bibliotecas de GAZEBO. El tipo de iluminación se define de la siguiente manera:

```
<include>
     <uri> model://sun</uri>
</include>
```

Y para definir el tipo de suelo se escribe:

```
<include>
     <uri> model://ground_plane</uri>
</include>
```

Por lo tanto, la definición del entorno llamado “mimundo” queda de la siguiente manera:

```
<?xml version="1.0"?>
<sdf version="1.4">
<world name="mimundo">
      <include>
           <uri> model://sun</uri>
      </include>
      <include>
         <uri> model://ground_plane</uri>
      </include>
</world>
</sdf>
```

Ya que se define correctamente el entorno, se procede a guardar los cambios que se hicieron en el archivo **myev3.world**. Se cerrará el editor de textos y se sigue en la terminal, se escribe el comando `cd ..` para regresar al directorio **`myev3_gazebo`** y de aquí ahora se dirige al directorio **launch**,

```
cd launch
```

Aquí se crea el archivo que inicializa la simulación en 3D del mundo que se está modelando. Se ejecuta el siguiente comando en la terminal que está abierta.

```
gedit myev3_world.launch
```

De igual manera se abrirá una ventana con el editor de textos gedit y se colocan las siguientes líneas. Estas líneas sirven para heredar las funcionalidades del mundo `empty_world.launch`, mismo que ya existe dentro de Gazebo, de manera que el único parámetro que se modifica es `world_name`, este será evaluado por el nombre del mundo que se crea, en este caso es `myev3.world` y también se observa que se indica la ubicación del archivo world, esto se define en la línea 3 y 4 del siguiente código:

```
<?xml version="1.0" encoding="UTF-8" ?>
<launch>
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
   <arg name="world_name" value="$(find myev3_gazebo)/worlds/
    myev3.world"/>
   <arg name="gui" value="true"/>
 </include>
</launch>
```

Se observa que ahora toda la información relevante se encuentra dentro de la etiqueta `<launch>`.

Para corroborar que todo está correctamente modelado, simplemente se inicia la simulación 3D en Gazebo. Se abre una nueva terminal y se ejecuta el comando:

```
roscore
```

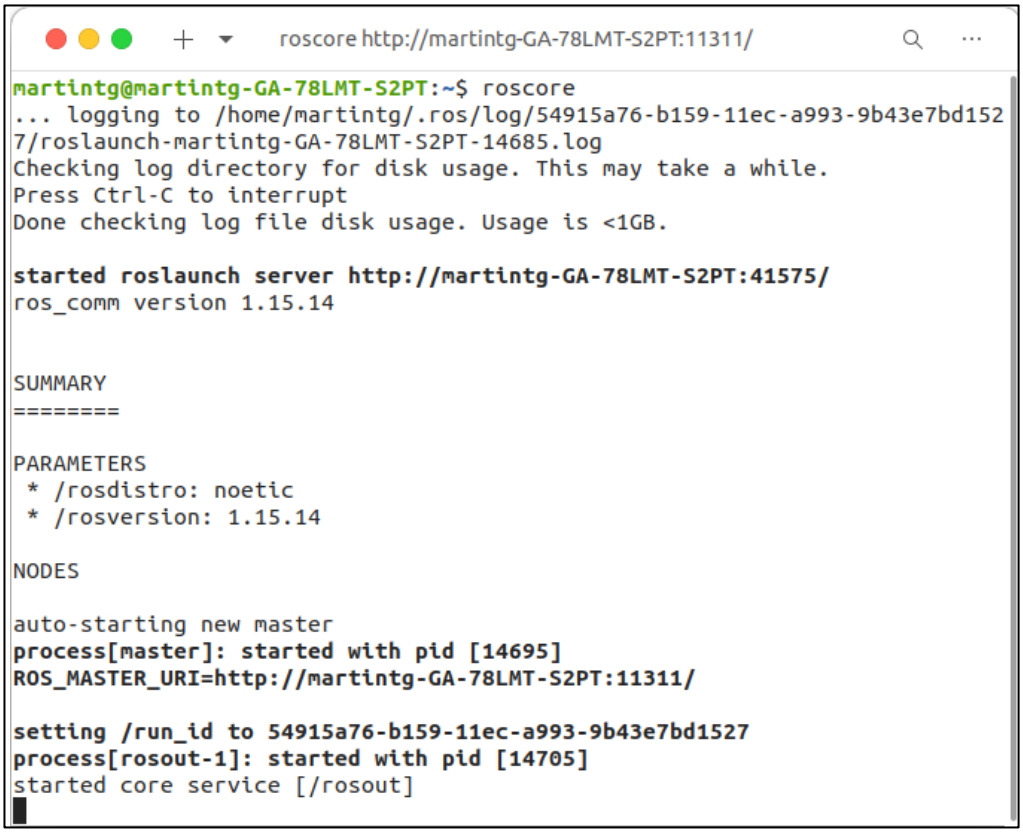


Figura 27. Captura de pantalla de terminal después de ejecutar el comando roscore.

Este comando inicia el núcleo maestro de ROS y así se pueda ejecutar la simulación, figura 27. Se vuelve a la terminal que ya se tenía abierta y se ejecuta la línea:

```
source ~/catkin_et/devel/setup.bash
```

Seguidamente se escribe y se ejecuta la línea de comandos:

```
roslaunch myev3_gazebo myev3_world.launch
```

Se abrirá una ventana nueva, que es la interfaz gráfica del simulador gazebo. La figura 28 muestra las características que se definen para el entorno de simulación y se observan en el panel del lado izquierdo de Gazebo, donde se encuentran las características físicas modeladas. En la figura 29 se aprecia con detalle.

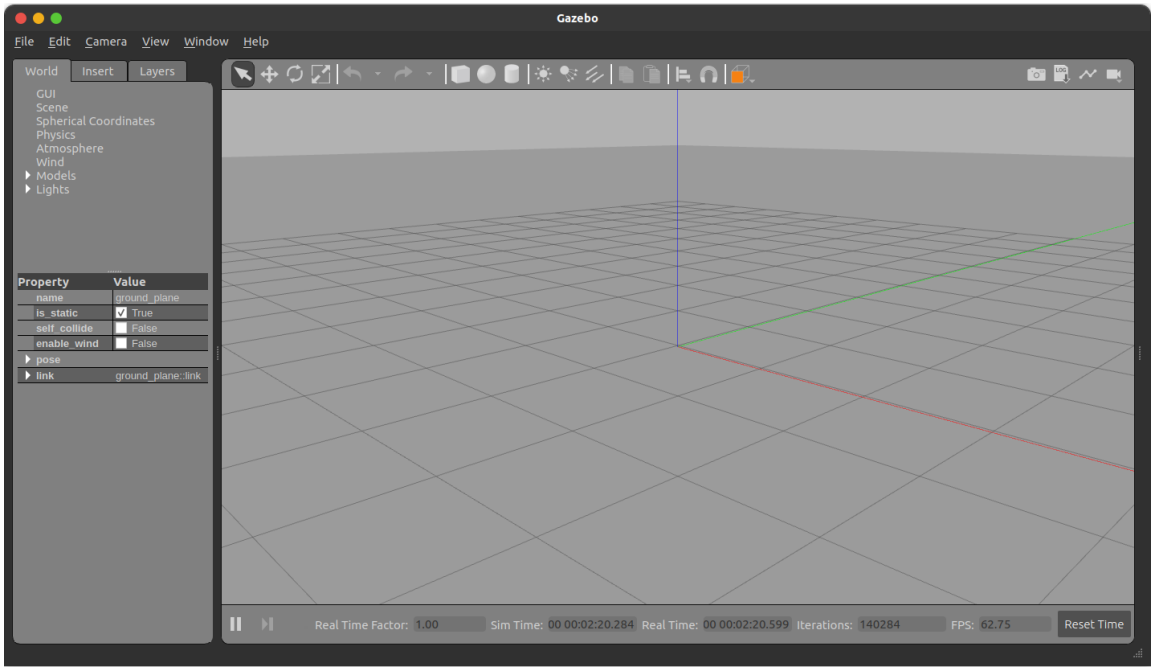


Figura 28. Captura de pantalla de la ventana que corresponde a Gazebo.

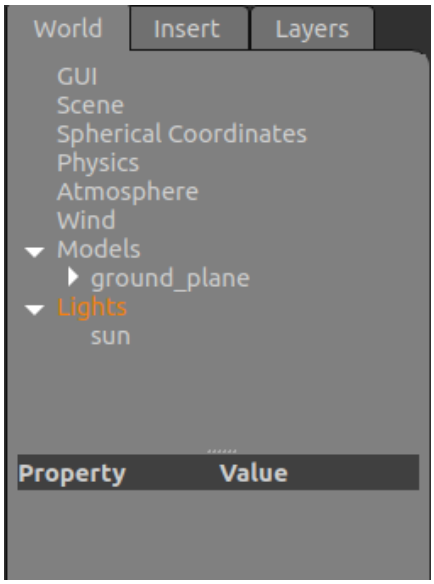


Figura 29. Captura de pantalla de la barra lateral izquierda de gazebo que muestra las propiedades del mundo.

Por el momento cerrar la ventana de Gazebo y en la terminal donde se ejecutó el comando **roslaunch**, se cierra el proceso oprimiendo al mismo tiempo las teclas **Crtl** y **C**, esperar a que se finalicen los procesos y se cierra la terminal.

Continuamos ahora con el modelo en 3D del robot EV3. Se abre una nueva terminal y se dirige al directorio donde se encuentra el proyecto, escribir y ejecutar lo siguiente:

```
cd ~/catkin_et/src
```

Ahora se ubica en el interior de la carpeta `myev3_description`, se ejecuta la línea:

```
cd myev3_description
```

Se crea una carpeta nueva llamada **URDF**:

```
mkdir urdf
```

y se dirige al interior de esta misma con el comando `cd`.

En esta ubicación crear el archivo .xacro, este contendrá la descripción del robot EV3 y se nombra `myev3.`xacro. Ejecutar la siguiente línea, la cual crea y abre una ventana del editor de textos gedit:

```
gedit myev3.xacro
```

Recordando lo que se explicó en el Capítulo 2 en la sección 2.4, primero se agrega al archivo la estructura básica de un archivo con extensión xacro.

Ya que agregamos la estructura básica, se deben analizar y observar las partes del robot como, por ejemplo: el cerebro del robot EV3 (Brick), los motores, las ruedas y sus soportes, figura 30.

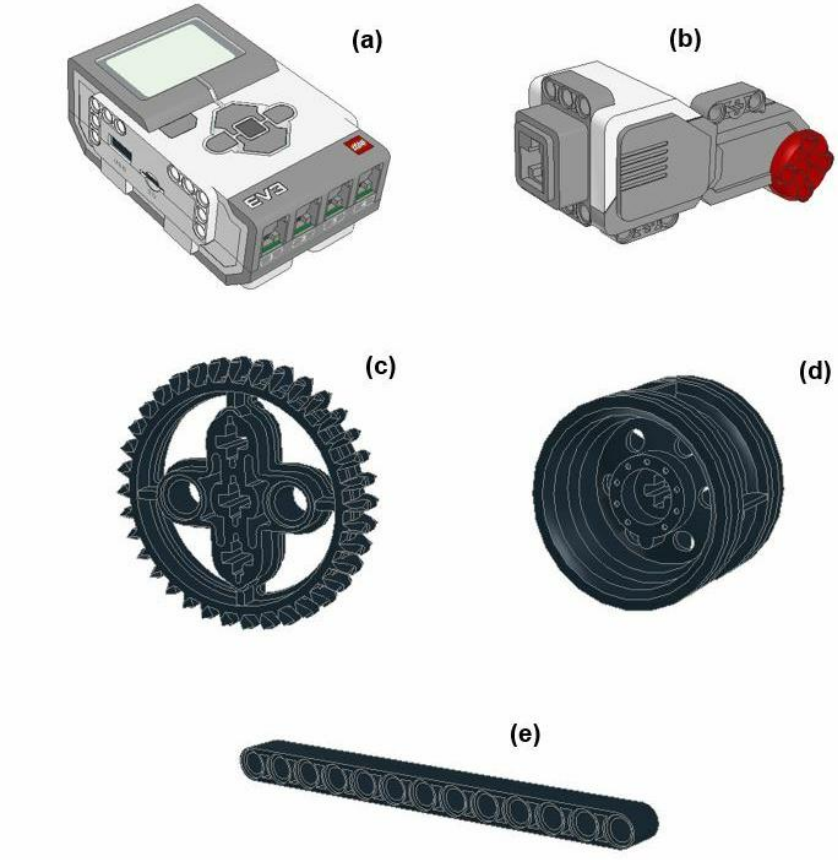


Figura 30. Diseño 3D de piezas del robot lego EV3 con software LeoCad. (a) Brick, (b) Motor grande, (c) Engrane, (d) Rin y (e) Barra de 5 pines.

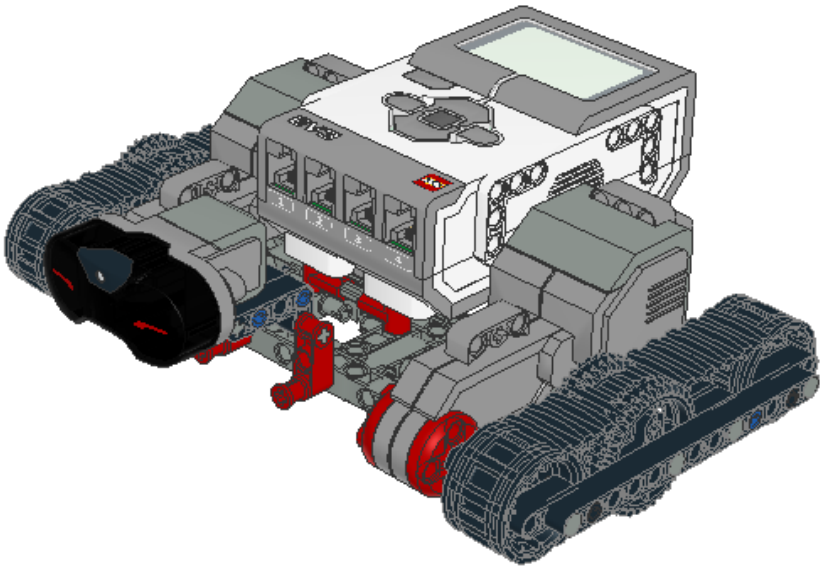


Figura 31. Diseño de Robot EV3, elaborado con software LeoCad.

El modelo en 3D de cada parte del robot se construirá por un conjunto de cuerpos geométricos que se asemejen lo más posible a cada elemento que conforma físicamente al EV3. Así entonces se toma la idea de modelar al robot EV3 como se muestra en el siguiente diagrama de la figura 32.

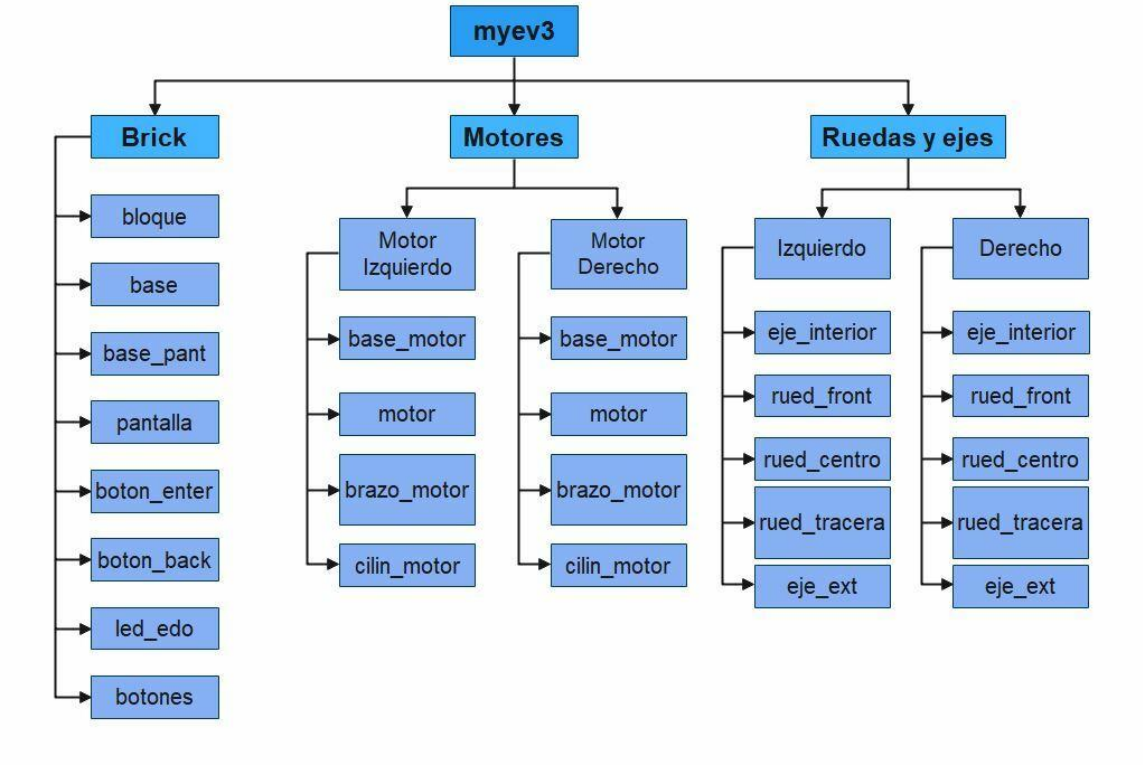


Figura 32. Diagrama de modelado de las piezas que compone el robot EV3.

Se toman medidas aproximadas de las dimensiones de cada parte, ya que estos valores serán necesarios para la simulación, además estos deben estar definidos en el archivo xacro con ayuda de las etiquetas `<xacro:property/>`, esta etiqueta se especifica en el Capítulo 2.

Como primera línea se pone la constante de Pi = 3.1415... Esta constante matemática se usa para realizar giros sobre los ejes x, y y z de las figuras a modelar, esto se verá a detalle más adelante. La forma en que se escribirá dentro de las etiquetas `<xacro:property/>`, es de la siguiente manera:

```
<xacro:property name="PI" value="3.1415926535897931" />
```

Enseguida se comienza a definir las medidas del Brick, que se construirá por un conjunto de tres elementos principales, en este caso tres prismas rectangulares, figura 33.

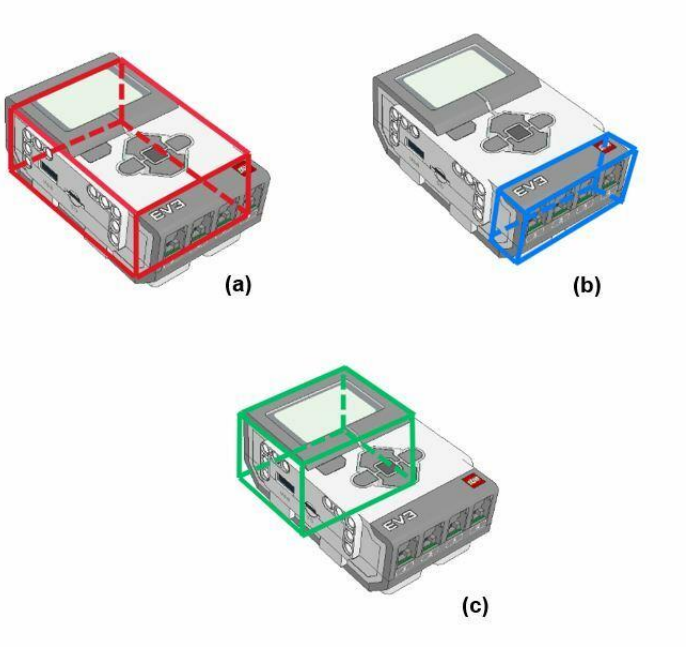


Figura 33. Modelado en 3D del Bick. (a) bloque, (b) base y (c) base_pant.

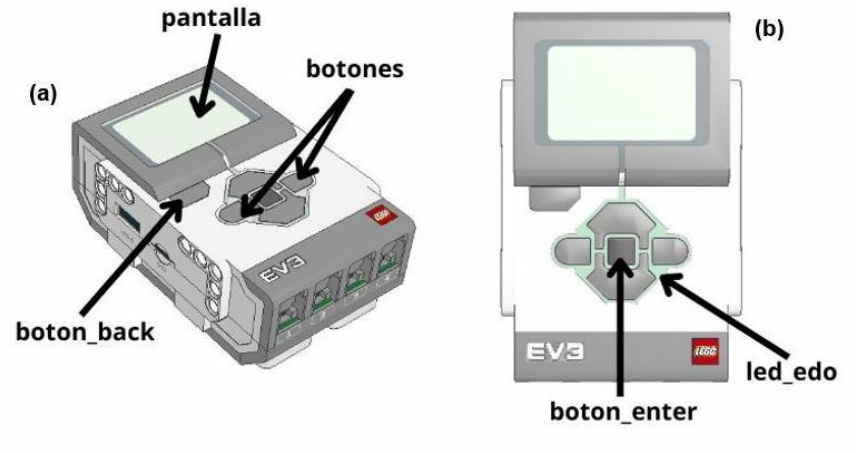


Figura 34. Modelado en 3D de las piezas que componen al Brick. (a) Botones y pantalla y (b) botón central y led de estado del Brick.

De acuerdo con las figuras 32 y 33 los prismas tendrán el nombre de “bloque”, “base” y “base_pant”. Otros elementos que se elaboran son: “pantalla”, “led_edo”, “botones”, “boton_enter” y “boton_back”, figura 34. Cada uno de los objetos estarán definidos por: **largo**, correspondiente al eje x, **ancho** correspondiente al eje y, y la **altura** que corresponde al eje z. También se debe de definir la **masa**. Así entonces uno de los tres prismas se define de la siguiente forma:

```
<xacro:property name="bloqueLarX" value="0.091" />
<xacro:property name="bloqueAnchY" value="0.071" />
<xacro:property name="bloqueAltZ" value="0.0352" />
<xacro:property name="bloqueMass" value="20" />
```

Para elaborar la pantalla, solo se genera un prisma rectangular con un grosor mínimo. Por otro lado, para la elaboración de los botones, serán prismas cuadrangulares, los cuales más adelante se explica cómo se irán acomodando. A continuación, la forma en que se colocan las medidas para hacer estos detalles del Brick es la siguiente:

```
<xacro:property name="pantallaLarX" value="0.031" />
<xacro:property name="pantallaAnchY" value="0.051" />
<xacro:property name="pantallaAltZ" value="0.01" />
<xacro:property name="pantallaMass" value="1" />


<xacro:property name="botonLarX" value="0.01" />
<xacro:property name="botonAnchY" value="0.01" />
<xacro:property name="botonAltZ" value="0.008" />
<xacro:property name="botonMass" value="0.05" />
```

Como siguiente paso se define las medidas de las ruedas, de las cuales se tienen dos diferentes (figura 35). Se nombrarán como ruedapq (rueda pequeña) y ruedadg (rueda grande).

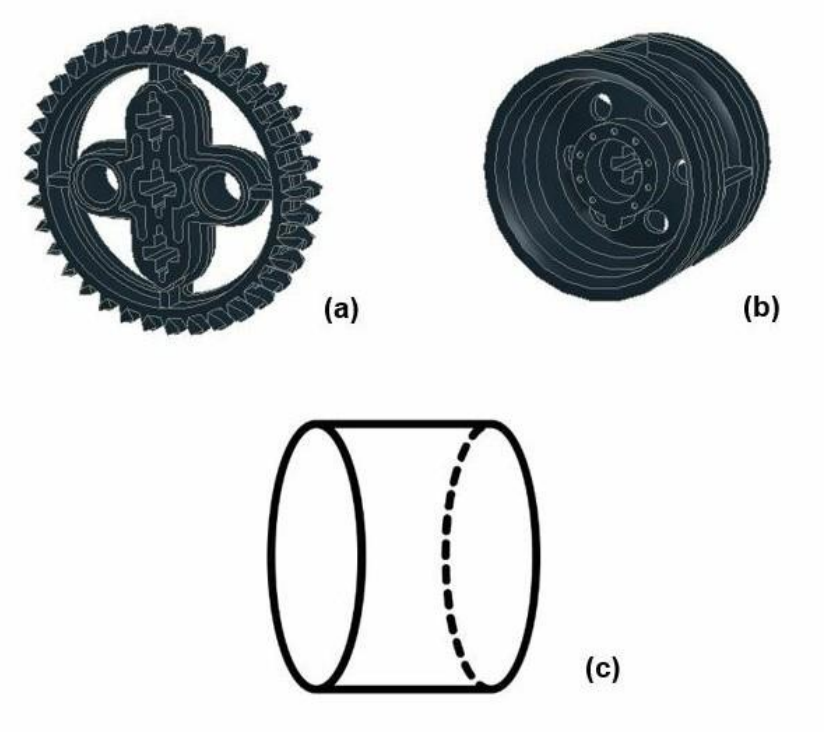


Figura 35. Modelado 3D de las ruedas. (a) engrane, (b) rin y (c) Cilindro, cuerpo geométrico que modela las ruedas del robot.

Las ruedas estarán formadas por cilindros con las siguientes medidas: **radio**, **grosor**, **posición** y **masa**. Por lo tanto, la rueda grande y pequeña se definen como sigue:

```
<xacro:property name="ruedapqRadio" value="0.015" />
<xacro:property name="ruedapqGrosr" value="0.023" />
<xacro:property name="ruedapqMass" value="1" />

<xacro:property name="ruedagdRadio" value="0.018" />
<xacro:property name="ruedagdGrosr" value="0.023" />
<xacro:property name="ruedagdMass" value="1" />
```

El motor grande se conforma por 4 cuerpos geométricos: dos cubos, un prisma cuadrangular y un cilindro, figura 36.

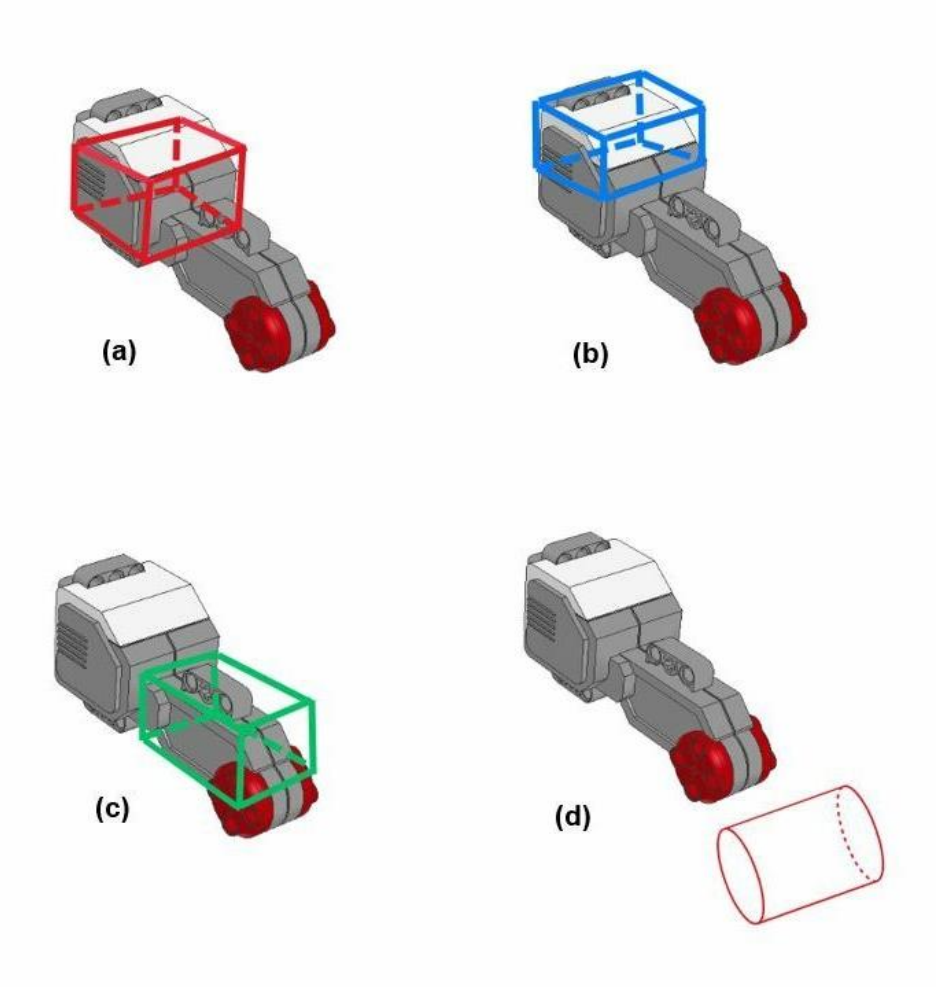


Figura 36. Modelado 3D de los motores. (a) base_motor, (b) motor, (c) brazo_motor y (d) cilin_motor.

Para ello, se definen los siguientes elementos: “`motor`”, “`base_motor`”, “`brazo_motor`” y “`cilin_motor`”. Esta pieza es una combinación de las piezas que se han definido anteriormente, por lo tanto, la forma en que se define uno de los 4 cuerpos geométricos, es la siguiente:

```
<xacro:property name="motorLarX" value="0.035" />
<xacro:property name="motorAnchY" value="0.0352" />
<xacro:property name="motorAltZ" value="0.028" />
<xacro:property name="motorMass" value="10" />
```

Por último, se escriben las dimensiones de los soportes de las ruedas. Para esta pieza, se define solo un prisma cuadrangular de la manera que sigue:

```
<xacro:property name="ejeLarX" value="0.103" />
<xacro:property name="ejeAnchY" value="0.007" />
<xacro:property name="ejeAltZ" value="0.007" />
<xacro:property name="ejeMass" value="0.2" />
```

En el Capítulo 2 se explicó que los robots modelados de esta forma están conformados por 4 archivos importantes (*nombre_del_robot.xacro, macros.xacro, materials.xacro* y *nombre_del_robot.gazebo*), siendo el archivo `myev3.xacro` el que se está editando en esta sección. Cabe mencionar que de aquí en adelante se editará el contenido de estos cuatro archivos simultáneamente. En el Capítulo 2 se dio información sobre la etiqueta `<include>` que se usa para agregar o llamar la información que se requiera de otros archivos. Por lo tanto, se utiliza la siguiente etiqueta para compartir e incluir información de los demás archivos que conforman el robot:

```
<xacro:include filename="$(find myev3_description)/urdf/myev3.gazebo" />
<xacro:include filename="$(find myev3_description)/urdf/materials.xacro"
/>
<xacro:include filename="$(find myev3_description)/urdf/macro.xacro" />
```

Ahora se comienza con la elaboración del modelo del robot, para ello es necesario tener en claro las funciones y usos de algunas etiquetas que se van a utilizar [14], [15], [31], [32], [33], [34].

<link>. Etiqueta que ayuda a describir las características visuales y de inercia de un cuerpo sólido. Esta etiqueta tiene como atributo **name**, que es el nombre que se le asigna al link o pieza del robot a modelar. Así mismo, contiene elementos que son etiquetas como: **<visual>**, **<collision>** e **<inertial>**.

- **<visual>** Como su nombre lo dice esta etiqueta proporciona información de la forma del objeto del link, todo esto con el objetivo de visualizarse en el simulador de 3D Gazebo. Aquí se pueden encontrar etiquetas como:
    - → **<origin>** que es el punto donde estará situado el elemento visual con respecto a un punto de referencia que tenga asignado el link. Este está dado por desplazamiento en los planos con **xyz** y por ángulos de giro con **rpy** en unidades de radianes; razón por la que se declara la constante Pi.
    - → **<geometry>** encargada de asignar la forma del objeto visual. La forma del objeto está definida por etiquetas que ya están por defecto y son las siguientes: **<box>** esto es para formar una caja, tiene como atributo **size** que define los tres lados del cuerpo geométrico, cabe mencionar que el origen del cuerpo geométrico está en el centro; **<cylinder>** con ello se forma un cilindro, el cual tiene como atributos **lenght** y **radius**, que indican la altura y el radio, su origen está en el centro; **<sphere>** con esto generamos una esfera, la cual consta del atributo **radius**, que indica el radio de la esfera, de igual manera su origen yace en el centro de la misma. Por último, para agregar otro

tipo de cuerpos geométricos está la etiqueta **<mesh>**, esta permite cargar figuras tridimensionales desde archivos con extensiones **.dae** y **.stl**, las cuales son las más recomendables de usar.

→ **<material>** con esta etiqueta se especifica el material del objeto visual, este está definido por el atributo **name**, en el que se especifica entre comillas el nombre del material.

- **<collision>** Esta etiqueta contiene información sobre las propiedades de colisión del objeto o link del robot a modelar en 3D. Con esta etiqueta el simulador Gazebo forma un área de colisión, dando así la capacidad al robot simulado de percibir colisiones. Por lo general, las propiedades visuales pueden contener los mismos valores en los atributos **<origin>** y **<geometry>**, esto es con la finalidad de reducir el tiempo de procesamiento en el momento de simularse el robot.
- **<inertial>** Contiene información correspondiente a las propiedades de inercia que se le definirán al robot a simular. Esta etiqueta está conformada por los atributos: **<origin>**, igual a la anterior; **<mass>** etiqueta correspondiente al valor de la masa del objeto; **<inertia>** es una etiqueta que contiene valores de la matriz inercial de 3x3, cabe destacar que son 6 elementos de esta matriz que están definidos y los cuales usan los siguientes atributos: **ixx**, **ixy**, **ixz**, **iyy**, **iyz** e **izz**.

<joint> Es una etiqueta a la que también se conoce como “articulación”, contiene información de las articulaciones sobre propiedades cinemáticas y dinámicas y, lo

más importante, es una etiqueta que determina la unión de dos elementos o links usando otras etiquetas nombradas **parent** y **child**. Sus atributos importantes son **name**, que define el nombre de la articulación que le asigne el desarrollador y **type**, que define el tipo de articulación de acuerdo con la funcionalidad que proporcionará este elemento al robot. Los tipos de articulaciones que existen son:

- **revolute.** Este tipo de articulación es aquella que solo gira un grado de revolución en torno a un eje. Se dice que esta articulación se asemeja a una bisagra, por lo que tiene limitaciones inferiores y superiores.
- **continuous.** Esta articulación se asemeja a la revolute, la diferencia es que aquí gira continuamente sobre el eje, esto implica que no tenga limitaciones.
- **prismatic.** Define una articulación con tan solo un grado de movimiento sobre un eje con un rango limitado.
- **fixed.** Esta no es una articulación, ya que no puede moverse y por lo tanto no necesita de ejes, calibración, dinámica, limites o control de seguridad.
- **floating.** Es una articulación que solo permite 6 grados de libertad.
- **planar.** Articulación que da movimiento sobre un plano que es perpendicular a un eje. Por lo tanto, permite solo 2 grados de movimiento.

Continuando con la etiqueta **<joint>** dentro de esta se encuentran elementos que igual son etiquetas como: **<origin>**, **<parent>**, **<child>**, **<axis>**, **<calibration>**, **<dynamics>**, **<limit>, <mimic>** y **<safety_controller>**. Las siguientes etiquetas son las que se ocupan en este proyecto.

- **<parent>** Etiqueta que tiene como atributo **link**, en el que se especifica el nombre del link a unir con otro objeto a partir de una articulación **joint**. El link definido con esta etiqueta se convierte en la pieza principal de la unión.
- **<child/>** Etiqueta con atributo **link**, que se usa cuando se realiza la unión de dos piezas a partir de una articulación **joint**. Por lo general, el objeto relacionado con esta etiqueta se define como la pieza secundaria dentro de la unión.
- **<origin/>** Etiqueta que define la referencia en el espacio de la articulación, la cual se encuentra en el centro de los objetos a unir, en este caso entre parent y child. Esta se encuentra definida por los mismos parámetros que tiene la etiqueta **origin** dentro de la etiqueta **visual**.
- **<axis/>** Etiqueta que define el eje de rotación o eje de translación para articulaciones del tipo prismatic, mientras que para las articulaciones del tipo planar aquí se define el plano perpendicular al plano de dicha articulación. Para articulaciones tipo fixed o floating esta etiqueta no está definida.
- **<limit/>** Etiqueta que contiene parámetros como límite inferior y superior de movimiento, de velocidad máxima y de esfuerzo máximo que puede existir en una articulación. El límite inferior está definido por **lower**, mientras que el límite superior esta dado por **upper**. Las unidades para estos valores dependerán del tipo de articulación que se haya definido. Por ejemplo, si joint es prismatic, las unidades serán en metros (*m)*; mientras que, si es revolute, las unidades serán radianes (*rad)*. Por otro lado, para el caso de la velocidad,

está definida por el atributo **velocity**, sus unidades de igual forma dependerán del tipo de joint. Entonces *m/s*, será para prismatic y *rad/s* para revolute. Por último, se encuentra el parámetro esfuerzo, que está denominado como **effort** con unidades de *N* para prismatic y *N*m* para revolute.

- **<dynamics/> o <joint_properties/>** Dentro de esta etiqueta se encuentran las propiedades físicas que tienen las articulaciones. Estas propiedades son la fricción y el amortiguamiento.

Con los conceptos mencionados se comienza a crear un link, el cual es y será el punto de referencia del robot. Este link se va a nombrar `referencia_base`, como solo es la referencia tendrá una estructura sencilla que no necesitará de etiquetas principales como: `<visual>`, `<collision>` e `<inertial>`. Seguidamente se comienza a construir el Brick del EV3. Se construyen los links que formarán el brick, se inicia con el link que lleva el nombre de `bloque`; simplemente se modelará como una caja. La estructura del link `referencia_base` y la estructura del link bloque son como sigue:

```
<link name="referencia_base"/>

<link name="bloque">
      <visual>
        <origin xyz="0 0 0.0175" rpy="0 0 0"/>
        <geometry>
          <box size="${bloqueLarX} ${bloqueAnchY} ${bloqueAltZ}" />
        </geometry>
        <material name="white"/>
      </visual>
      <collision>
        <origin xyz="0 0 0.0175" rpy="0 0 0"/>
        <geometry>
          <box size="${bloqueLarX} ${bloqueAnchY} ${bloqueAltZ}"/>
```

```
        </geometry>
      </collision>
      <inertial>
        <origin xyz="0 0 0.0175" rpy="0 0 0"/>
        <mass value="${bloqueMass}"/>
        <xacro:box_inertia          m="${bloqueMass}"          x="${bloqueLarX}"
y="${bloqueAnchY}" z="${bloqueAltZ}"/>
      </inertial>
</link>
```

En el código, se aprecia que el link tiene como atributo **name= "bloque"**, así como este link está construido por las etiquetas **<visual>**, **<collision>** e **<inertial>**. Cabe destacar que para este caso la etiqueta `<origin>` que aparece en las etiquetas `<visual>`, `<collision>` e `<inertial>` tiene los mismos valores en el atributo **xy**, pero en **z** tiene un valor diferente de cero, esto es para que la caja quede elevada a una altura que es igual 0.0175 m. Otro punto para destacar es la etiqueta `<geometry>`, que define la forma del objeto, en este caso una caja, que se crea con la etiqueta `<box>`, la cual a su vez tiene como atributo **size** con tres valores que indican las dimensiones de la caja. La etiqueta `<geometry>` se encuentra presente dentro de la etiqueta `<visual>` y `<collision>`. También se aprecia la etiqueta `<material>` que tiene como atributo **name**, en donde se especifica el nombre del material, en este caso es "`grey`" (policarbonato de color gris).

Por otra parte, el contenido de la etiqueta `<inertial>` tiene la etiqueta `<mass>` con atributo **value** con el valor del peso del objeto dado en kilogramos (kg). Por último, está la etiqueta `<xacro:box_inertia>`, la cual ya se explicó en el Capítulo 2. Para este caso se definen los atributos: **m**, masa del objeto; **x**, largo del objeto; **y**, ancho del objeto; y **z**, altura del objeto.

Ahora se guardan los cambios del archivo `myev3.xacro` y se cierra el editor de texto. En seguida se crea el archivo `materials.xacro`, para ello, se ejecuta en la terminal el comando que sigue:

```
gedit materials.xacro
```

Se abre el editor de textos con un documento vacío y luego se agrega la información siguiente:

```
<?xml version="1.0"?>
<robot>

       <material name="black">
          <color rgba="0.0 0.0 0.0 1.0"/>
        </material>

       <material name="grey">
          <color rgba="0.2 0.2 0.2 1.0"/>
       </material>

       <material name="grey1">
          <color rgba="0.21 0.21 0.21 1.0"/>
       </material>

       <material name="red">
          <color rgba="0.8 0.0 0.0 1.0"/>
      </material>

      <material name="white">
          <color rgba="1 1 1 1"/>
      </material>

      <material name="pantallacolor">
          <color rgba="0.651 0.678 0.592 1"/>
      </material>

</robot>
```

Cabe mencionar que en el Capítulo 2 se explicó el contenido del archivo que se está editando. Aquí yace la definición de los materiales visuales que se van a usar,

por ejemplo: el material `grey, grey1 y white`, son para elaborar parte del Brick, los brazos de los motores; el material red también ayuda a construir parte de los motores. El material `pantallacolor` es para dar la textura de cristal a la pantalla, por último, el material `black` es para los soportes y las ruedas. Terminando de editar el archivo se guardan los cambios y se cierra el editor de textos.

Como siguiente paso se tiene que crear y editar el archivo `macros.xacro`. Entonces, en la terminal se ejecuta la siguiente línea:

```
gedit macros.xacro
```

para abrir el editor de textos con un archivo vacío. Por el momento, aquí se agrega la definición para realizar los cálculos de la inercia del objeto que se está modelando, en este caso el objeto "`bloque`" que es una caja. Se agrega el siguiente código en el archivo.

```
<?xml version="1.0"?>
<robot name="myev3" xmlns:xacro="http://www.ros.org/wiki/xacro">

        <xacro:macro name="box_inertia" params="m x y z">
            <inertia  ixx="${m*(y*y+z*z)/12.0}" ixy = "0.0" ixz = "0.0"
                      iyy="${m*(x*x+y*y)/12.0}" iyz = "0.0"
                      izz="${m*(x*x+z*z)/12.0}"/>
        </xacro:macro>

</robot>
```

De igual manera, el archivo `macros.xacro` lleva la misma estructura básica de los archivos que se han creado hasta el momento y toda la información relevante va dentro de las etiquetas `<robot></robot>`. Como primer punto se define la etiqueta que se usó en la etiqueta `<inertial>` para definir el link y modelar la caja llamada

“bloque”, todo esto en el archivo `myev3.xacro`. Se está hablando de la etiqueta `<xacro:box_inertia>`, esta etiqueta manda a llamar la etiqueta `<xacro:macro>`, que se encuentra definida en el archivo `macros.xacro`. En el Capítulo 2 se explica a detalle sobre la etiqueta `<xacro:macro>`, así como las ventajas de usarla. Continuando con el análisis del código anterior, la etiqueta `<xacro:macro>` cuenta con los siguientes parámetros definidos: **`name= “box_inertia”`**, que corresponde al nombre de la macro que se va a definir y **`params= “m x y z”`**, que son las variables que se definieron en el archivo `myev3.xacro` dentro de la etiqueta `<inertial>`. Por otra parte, en la etiqueta `<xacro:macro>` también se definen los cálculos para obtener los valores de la matriz de inercia, de los cuales **ixy**, **ixz** e **iyz** tienen un valor de cero y los otros tres se encuentran definidos por sus respetivas fórmulas. Es importante recordar que este cálculo de inercia es específicamente para cajas. Terminando de editar, se guardan los cambios y se cierra el editor.

Ahora se vuelve a editar el archivo `myev3.xacro`. En la misma terminal se ejecuta el código siguiente:

```
gedit myev3.xacro
```

se busca el segmento donde se crearon los dos links. Aquí se va a agregar la articulación que unirá a los dos links, para ello se escribe el siguiente código entre las líneas de código que definen a los links:

```
<joint name="bloque_union_referencia_base" type="fixed">
   <parent link="referencia_base"/>
   <child link="bloque"/>
</joint>
```

En el código anterior se aprecian los atributos **name= "bloque_union_referencia"**, el cual es el nombre que se le asigna a la articulación y, por otro lado, el atributo **type= "fixed"**, que define el tipo de articulación, siendo está una articulación estática. Seguidamente están las etiquetas `<parent>` y `<child>` en las que se especifica en su atributo **link**, los nombres de los links que se unirán por medio de esta articulación, en este caso el link **referencia_base** y el link **bloque**. Con esto se da por terminada la edición del archivo `myev3.xacro` para simular la caja que será parte del Brick. Se guardan los cambios y se cierra el editor de textos.

Ahora toca crear y editar el archivo `myev3.gazebo`, que de igual manera en el Capítulo 2 se hizo mención sobre su contenido y función. Para crear este archivo en la terminal que se tiene abierta se ejecuta la siguiente línea:

```
gedit myev3.gazebo
```

Una vez abierto el editor con el archivo vacío, se coloca la siguiente información:

```
<?xml version="1.0"?>
<robot name="myev3" xmlns:xacro="http://www.ros.org/wiki/xacro">

      <gazebo reference="bloque">
            <material>Gazebo/White</material>
      </gazebo>

</robot>
```

Se guardan los cambios y se cierra el editor de textos. Antes de iniciar la simulación y observar en Gazebo la caja que se acaba de crear, se debe de editar el archivo `myev3_world.launch`, para ello, en la terminal se ejecuta la siguiente línea:

```
cd ~/catkin_et/src/myev3_gazebo/launch
```

Esta línea nos dirige al directorio donde se encuentra el archivo que se va a editar.

En la terminal se ejecuta el comando para editar el archivo `myev3_world.launch`:

```
gedit myev3_world.launch
```

Una vez abierto el archivo se agrega el siguiente código que ayuda a que se ejecute la simulación y se pueda observar lo que se ha creado. Este código se debe de colocar dentro de las etiquetas `<launch></launch>` y después de la información de las etiquetas `<include></include>`:

```
<!-- urdf xml robot description loaded on the Parameter Server, converting the xacro into a proper urdf file-->

<param name="robot_description" command="$(find xacro)/xacro '$(find myev3_description)/urdf/myev3.xacro'"/>

<!-- push robot_description to factory and spawn robot in gazebo -->

<node name="myev3_spawn" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen" args="-urdf -model myev3 -param robot_description" />
```

El código se encuentra dividido por dos secciones. La primera sección especifica que el archivo URDF, en formato XML, será leído y cargado al Servidor de Parámetros. La información en este bloque está dentro de la etiqueta **`<param>`**. Este parámetro tiene el atributo **name**, el cual se encuentra definido como `robot_description`, aquí se carga toda la información del URDF. También se encuentra el atributo **command**, el cual proporciona la ruta donde se encuentra el archivo `myev3.xacro` y que se cargará al parámetro `robot_description`. La

segunda sección corresponde a la construcción de la simulación en Gazebo. Esta se construye a partir de la etiqueta **<node>,** la cual tiene como atributo **name= "myev3_spaw"**. Este nodo se encuentra dentro del paquete **gazebo_ros**, que leerá los datos de **robot_description** para que éstos sean simulados en Gazebo. Se observa que se especifica el modelo a simular, en este caso **myev3**.

Terminada la edición del archivo `myev3_world.launch` se guardan los cambios y se cierra el editor de textos. Con todo lo anterior, todo está listo para ver el bloque en el software Gazebo. Como primer paso, se debe verificar si en alguna de las terminales que se tienen abiertas se está ejecutando el nodo maestro. De no ser así, ejecutar el comando `roscore` en una terminal nueva. Por otro lado, en la terminal en que se ejecutó el comando para editar el archivo launch, se ejecuta el siguiente comando:

```
roslaunch myev3_gazebo myev3_world.launch
```

Inmediatamente se abre una ventana nueva que corresponde al software de Gazebo y una vez cargado se obtiene algo como lo mostrado en la figura 37.

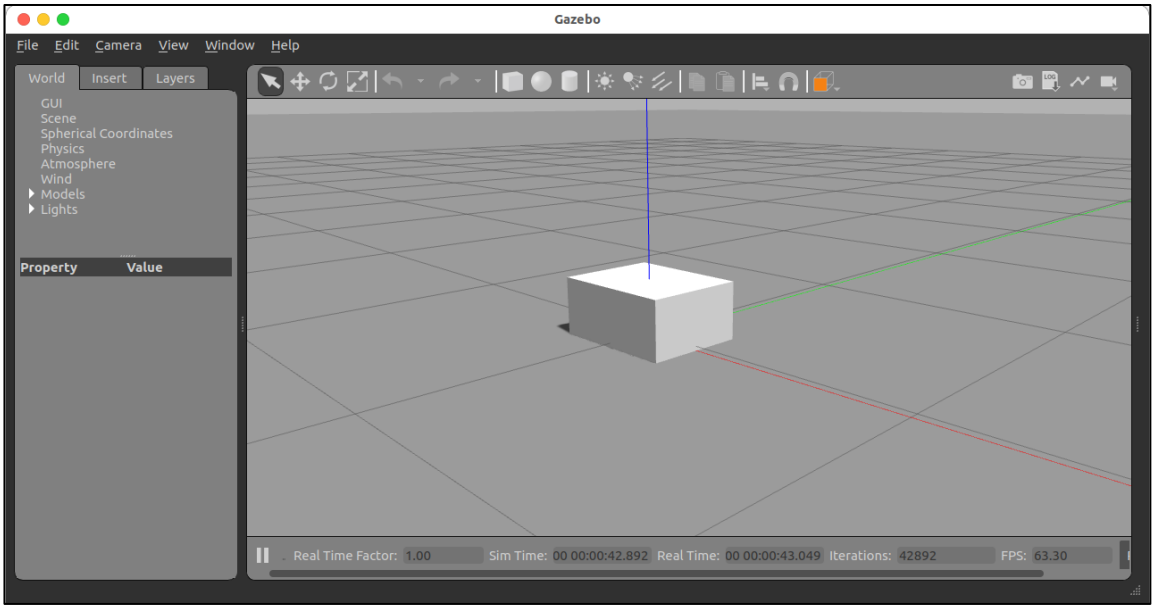


Figura 37. Captura de pantalla de Gazebo mostrando el bloque del Brick.

Se observa que efectivamente se simuló con éxito el bloque de color gris que se modeló. En el panel izquierdo está un listado con las propiedades del mundo, se da clic en **Models** y se despliega más información como se muestra en la figura 38.

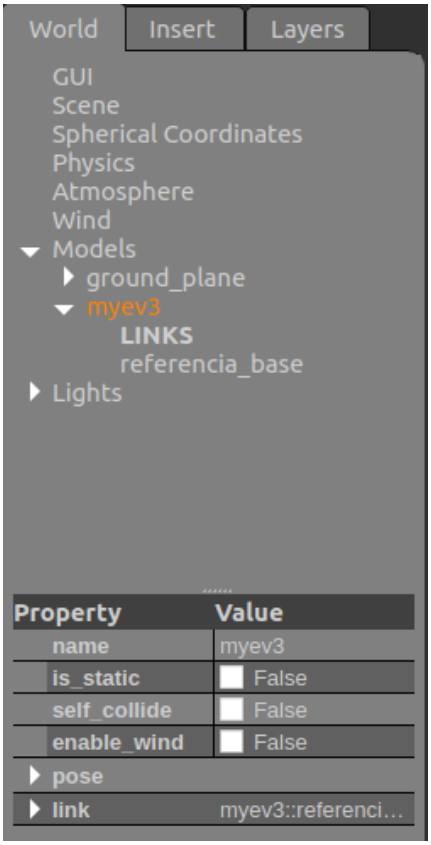


Figura 38. Captura de pantalla de barra lateral izquierda de herramientas de Gazebo.

Se puede apreciar que se encuentra el tipo de suelo y también se encuentra el proyecto myev3. Si se da clic sobre el nombre del proyecto se despliega información referente a los links que lo componen, en este caso solo es el link **`referencia_base`**. Se continúa elaborando los cuerpos geométricos más importantes que conforman el Brick, en este caso son tres cajas que se llaman **`base`**, **`base_pant`** y **`pantalla`**. Para elaborar estos elementos, se vuelve a editar el archivo **myev3.xacro**, se abre y se edita de la misma forma que se ha venido haciendo con ayuda del editor gedit. Así entonces, para agregar estas piezas, se usa la etiqueta link y se colocan los valores requeridos, los links a elaborar tendrán

la misma estructura que la caja que se construyó anteriormente. Algo que se debe de tomar en cuenta, es que la posición de origen de cada caja será diferente para cada uno.

Para el caso del link llamado **base**, su origen se encuentra definido como: **xyz= "0 0 0.02"**, este valor se debe colocar en donde se encuentre la etiqueta **<origin/>**. Por otro lado, para el link llamado **base_pant** su origen está especificado por: **xyz= "0 0 0.125"**. Otro elemento en el que se diferencian es en las dimensiones, que se fijaron en la etiqueta **<xacro:property>**. Es importante colocar la correspondiente etiqueta a cada objeto. También es relevante asignar el material correspondiente a cada caja. Para el caso de la base se ocupa el material **grey**, el cual será el mismo para las dos cajas.

Ahora se crea el modelo de los elementos que son detalles visuales para dar una mejor similitud con el Brick. Estos elementos son: **pantalla**, **led_edo**, **botones**, **boton_enter** y **boton_back**. Cada uno sigue siendo una caja, pero con un grosor, origen, tipo de material y masa diferente. Los elementos más sencillos para modelar son: **pantalla**, **boton_enter** y **boton_back**; estos también se construyen con ayuda de la etiqueta link. Para el caso de la **pantalla**, los atributos **xyz** de la etiqueta origen están definidos como **xyz= "0 0 0.005"**. Por otro lado, para el **boton_enter** y **boton_back** están dados por **xyz= "0 0 0.004".**

Para estas piezas modeladas del robot, los materiales se encuentran definidos como: "grey" para el **boton_back**, "black" para el link **boton_enter** y "cristal" para la **pantalla**.

El modelado de las piezas llamadas **led_edo** y **botones** tiene unos detalles que se explicarán enseguida. Modelar **led_edo** se construye de la misma manera que los links elaborados hasta el momento, también es una caja, solo que tiene un grosor mínimo, por lo que se asemeja más a una lámina. Cabe destacar que esta lámina cuadrada se encontrará girada sobre el eje z, figura 39.

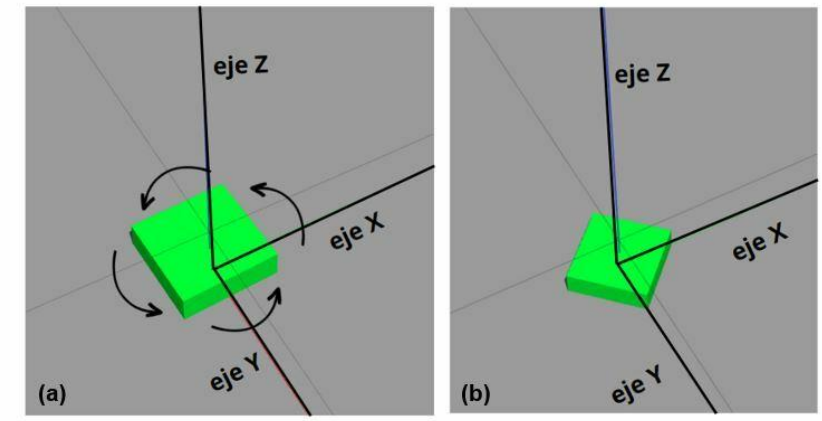


Figura 39. Giro de objeto en eje Z. (a) Objeto perpendicular al eje Y. (b) Giro del objeto hacia la izquierda con respecto al eje Z.

Para obtener ese giro en la etiqueta **<origin>** y en el atributo **rpy** se colocará el valor en radianes para dar el giro que se desea. Para este caso, como solo se requiere giro en el eje z, los demás ejes sin cambio, queda de la siguiente manera **rpy= "0 0 Pi/4"**. Dentro de la misma etiqueta se especifica el atributo **xyz= "0 0 0.004"** para esta pieza. En este elemento igualmente se especifica el **<material>** con un atributo **name= "green"**.

En la elaboración del link **botones**, se observa que se tienen dos botones al costado del **boton_enter**. La forma más rápida de hacerlos es modelar cada botón, es decir, hacer un link para el botón derecho y otro para el botón izquierdo. La desventaja de

modelarlo de esta forma es que se extenderá más el código, siendo que es la misma pieza solo que reflejada. Esto puede resolverse creando un solo link para construir los dos botones, para ello se usará una macro. En esta macro se le dan atributos que ayudan a definir el botón izquierdo y el botón derecho de manera automática, así como la ubicación en el plano de estos elementos.

En el archivo `myev3.xacro` que se está editando, se agregan las siguientes líneas, en donde se manda a llamar la macro "`botones`":

```
<xacro:botones sd="derecha" tY="1"/>

<xacro:botones sd="izquierda" tY="-1"/>
```

La macro llamada **`botones`** tiene los atributos **sd** y **tY**. El atributo **sd** define la orientación de los botones, razón por la cual se tiene los valores en **sd** de izquierda y derecha. Por otra parte, **tY** define la ubicación del objeto sobre el plano **y**, dado que el Brick está centrado en el origen, este ocupa los planos positivos y negativos en **x** y **y**. Por ello se observan los valores de **tY** con –1 y 1. Terminado esto se guardan los cambios y ahora se edita el archivo `macros.xacro`, esto se hace a través de la terminal y ejecutando la línea para editar el archivo con editor gedit.

Una vez abierto el archivo se agregan las siguientes líneas que definirán la macro de botones.

```
<xacro:macro name="botones" params="sd tY">
   <link name="boton_${sd}">
      <visual>
        <origin xyz="0 0 0.004" rpy="0 0 0"/>
        <geometry>
          <box size="${botonLarX} ${botonAnchY} ${botonAltZ}" />
        </geometry>
        <material name="grey"/>
      </visual>
```

```
      <collision>
         <origin xyz="0 0 0.004" rpy="0 0 0"/>
         <geometry>
           <box size="${botonLarX} ${botonAnchY} ${botonAltZ}"/>
         </geometry>
      </collision>

      <inertial>
         <origin xyz="0 0 0.004" rpy="0 0 0"/>
         <mass value="${botonMass}"/>
         <xacro:box_inertia m="${botonMass}" x="${botonLarX}"
y="${botonAnchY}" z="${botonAltZ}"/>
      </inertial>
   </link>

   <joint name="boton_${sd}_union_bloque" type="fixed">
       <parent link="bloque"/>
       <child link="boton_${sd}"/>
       <origin xyz="0.022 ${tY*0.015} 0.0274" rpy="0 0 0" />
   </joint>

   <gazebo reference="boton_${sd}">
       <material>Gazebo/Grey</material>
   </gazebo>
</xacro:macro>
```

El **link** para elaborar los botones se encuentra dentro de la macro que se define dentro de las etiquetas **<xacro:macro>** el cual tiene como atributos **name** y **params**. En **name** se define el nombre de la macro, en este caso "**botones**" y en **params** está definido **sd** y **tY,** que ya se mencionaron.

Como siguiente línea está la etiqueta link, que tiene el atributo **name= "boton_${sd}"**. Este link ayuda a construir a cada botón uno por uno de manera automática, ya que se crea el link **boton_derecha** cuando **sd = derecha** y cuando **sd =izquierda** se crea el link **boton_izquierda**. Continuando con el código, en

la parte donde aparece la etiqueta **<origin>,** tiene el atributo **xyz= “0 0 0.004"**. Se observa que de igual manera dentro de la etiqueta **<link>** están las etiquetas **<visual>**, **<collision>** e **<inertial>**. Dado que estamos haciendo los botones de manera automática usando los parámetros **sd** y **tY**, entonces dentro de la macro se tiene que definir la unión de los botones hacia el link **bloque**. Es por ello por lo que después de construir el link ahora toca elaborar la estructura del **joint**. Así entonces, en la etiqueta **<joint>** se especifica el atributo **name= “boton_${sd}_union_bloque”** y el atributo **type="fixed"**. Tomar en cuenta que donde esté el parámetro **sd** se escribe el valor de izquierda o derecha, según sea el caso. En la etiqueta **<parent>** se coloca el atributo **link= “bloque”**, que es la referencia base para construir el Brick. Y para el caso de la etiqueta **<child>** el atributo link se define como **link= “boton_${sd}”** que es el link de la elaboración de cada botón. Ya como punto final se define el **<origin>** de la unión o `joint` de la siguiente manera:

```
<origin xyz= “0.022 ${tY*0.015} 0.0274” rpy= “0 0 0” />
```

Observamos que los valores son muy diferentes de la etiqueta de **<origin>** que se encuentra dentro de las etiquetas **<visual>**, **<collision>** e **<inertial>**, esto es porque en esas etiquetas solo se está creando el objeto con centro en el origen y que el objeto se encuentre sobre el suelo. Por otra parte, en la etiqueta **<joint>** ya se está colocando el objeto en una ubicación especifica, por lo que la etiqueta **<origin>** tiene valores diferentes. Aquí se aprecia la presencia del parámetro **tY**, en el que cuando **sd = derecha** y **ty = 1** significa que se encuentra el botón en el

plano positivo del eje **y**. Por otro lado, cuando **sd = izquierda** y **ty= -1** el botón se encuentra en el plano negativo del eje **y**.

También dentro de la macro que se está construyendo, se define el tipo de material con las etiquetas **<gazebo>**, en este caso es "Grey". Con esto se conforma la macro para modelar los botones. Se guardan los cambios y se cierra la ventana del editor. Ahora se continúa editando el archivo myev3.xacro, esto se hace por medio de la terminal abierta y con ayuda del editor gedit.

Una vez elaborados los links correspondientes, es momento de definir las uniones o joint. Como recomendación y para tener un mejor orden en el código cada joint debe de estar entre cada link que se va a unir. Para no confundirse con la información que irá en las etiquetas **<parent>** y **<child>**, se define al link llamado **bloque** con la etiqueta **<parent>**, ya que este objeto será la referencia principal o la base donde se estarán agregando los objetos para construir el Brick del robot EV3. Por lo tanto, los demás elementos llevarán la etiqueta **<child>**. Cabe mencionar que esto solo será para el caso de la elaboración del Brick.

Después de haber agregado los joints correspondientes a cada elemento para formar el Brick, se guardan los cambios del archivo `myev3.xacro` y se cierra el editor. Antes de ver en simulación el resultado de la construcción del Brick, primero se debe editar el archivo `myev3.gazebo`, donde se va a agregar el material que se le asignó a cada link. Cabe mencionar que se debe de especificar el material a cada

link que se elaboró, no importa si usan el mismo material. Se define de la siguiente manera:

```
<gazebo reference="base">
   <material>Gazebo/Grey</material>
</gazebo>

<gazebo reference="base_pant">
   <material>Gazebo/Grey</material>
</gazebo>

<gazebo reference="pantalla">
 <material>Gazebo/Pantallacolor</material>
</gazebo>

<gazebo reference="boton_enter">
 <material>Gazebo/Black</material>
</gazebo>

<gazebo reference="boton_back">
 <material>Gazebo/Grey</material>
</gazebo>

<gazebo reference="led_edo">
 <material>Gazebo/Grey</material>
</gazebo>
```

Se guardan los cambios y se cierra el editor. En esa misma terminal que se tiene abierta se escribe el siguiente comando para iniciar la simulación en Gazebo:

```
roslaunch myev3_gazebo myev3_world.launch
```

Es importante verificar si ya se está ejecutando el nodo maestro en otra terminal, sino es así, ejecutar en una terminal el comando `roscore`.

Si todo está correcto se debe abrir una ventana nueva de Gazebo y se debe observar el Brick como en la siguiente figura 40.

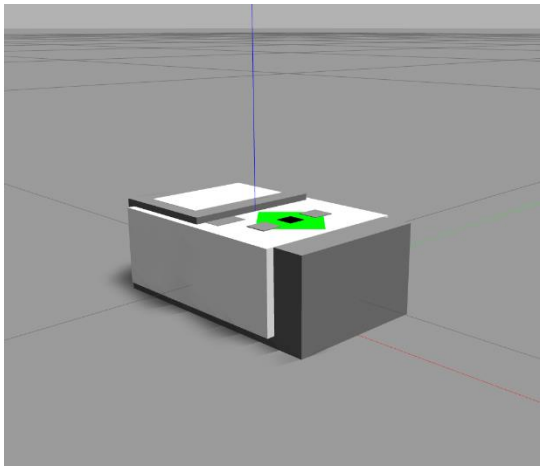


Figura 40. Captura de pantalla de Gazebo que muestra el modelado en 3D del Brick del robot EV3.

Se continua con el modelado del robot y ahora toca el turno de construir los motores. Anteriormente se especificó que esta pieza del robot está conformada por los siguientes cuerpos geométricos: **`base_motor`**, **`motor`**, **`brazo_motor`** y **`cilin_motor`**. Los primeros dos son cajas con un material diferente, el tercero es un prisma rectangular y el último es un cilindro. Al analizar este elemento del robot, se deduce que de igual manera los motores son simétricos.

Se abre y se edita nuevamente el archivo `myev3.xacro`. Primero se comienzan a elaborar los elementos: **`base_motor`**, **`motor`** y **`brazo_motor`**. Para la construcción de estos se usa el mismo método de modelado que el de los botones del Brick. Tener en cuenta que cada elemento tiene su material, tamaño y origen específico, así como recordar que se estarán editando los archivos `myev3.xacro` y `macros.xacro`.

Para el caso del cilindro **cilin_motor**, primero se tiene que editar el archivo `macros.xacro` y definir la macro de inercia para un cilindro, esta se especifica de la siguiente manera:

```
<xacro:macro name="cylinder_inertia" params="m r h">
       <inertia  ixx="${m*(3*r*r+h*h)/12.0}" ixy = "0.0" ixz = "0.0"
                 iyy="${m*(3*r*r+h*h)/12.0}" iyz = "0.0"
                 izz="${m*r*r/2.0}"/>
</xacro:macro>
```

Para llevar orden en el código se recomienda colocar lo anterior debajo de las líneas donde se define la inercia para una caja. Enseguida se continúa editando este archivo agregando la definición de la macro **cilin_motor**. Esta lleva casi la misma estructura que las demás macros, las diferencias son:

- ✓ En la etiqueta **<origin>** lleva los siguientes atributos **xyz="0 0 0.0115" rpy="0 ${PI/2} ${PI/2}"**.
- ✓ Dentro de la etiqueta **<geometry>** se define la etiqueta **<cylinder length="${cilin_motorGrosr}" radius="${cilin_motorRadio}"/>**.
- ✓ En la etiqueta **<material> name="red"**.
- ✓ La etiqueta **<xacro:cylinder_inertia>** se especifica como: **<xacro:cylinder_inertia m="${cilin_motorMass}" r="${cilin_motorRadio}" h="${cilin_motorGrosr}"/>**.
- ✓ Para la etiqueta **<joint>** en **<origin>** se define **xyz="0.01885 ${tY*0.0001} -0.008" rpy="0 0 0"**.

- ✓ Por último, para la etiqueta gazebo solo se especifica el material **Red** dentro de la etiqueta **<material>.**

Se guardan los cambios y se cierra el editor de textos. Ahora se abre y edita el archivo `myev3.xacro` y se colocan las siguientes líneas:

```
<xacro:cilin_motor sd="derecha" tY="1"/>
<xacro:cilin_motor sd="izquierda" tY="-1"/>
```

Con todo lo anterior se tienen todos los componentes para modelar los motores del robot EV3.

Como siguiente paso, se elaboran los soportes de las ruedas: **eje_int** y **eje_ext**. Estas piezas serán dos prismas rectangulares con una longitud larga. De igual manera estas piezas son simétricas y por lo tanto se usa una macro para la construcción de estos elementos, que es el mismo método que se usó para modelar los botones.

El caso de las ruedas también presenta simetría, entonces para crear el modelo se usa una macro y de igual manera se modifican los archivos `myev3.xacro` y `macros.xacro`. Las ruedas serán cilindros y estarán definidas como: **rued_front**, **rued_centro** y **rued_tracera**, estos tienen una orientación y ubicación específica.

Se comienza a definir la **rued_front**: dentro de la etiqueta **<link>** se encuentra la etiqueta **<origin>**, en la que se especifica lo siguiente **xyz="0 0 0.0115" rpy="0 ${PI/2} ${PI/2}"**, en el atributo **rpy** se colocan valores para girar 90° el cilindro. El material de las ruedas se define como **name="grey1"**. Para el apartado de **<joint>**, dado que las ruedas se moverán, este joint será del tipo

type="continuous". En el etiquetado <origin>, se define la ubicación de las ruedas de la siguiente manera **xyz="0.0465 ${tY*0.015} -0.008" rpy="0 0 0"**. Dentro de joint se van a agregar nuevas etiquetas como: <axis>, <limit> y <dynamics>. Para axis se especifica **xyz="0 1 0"** y **rpy="0 0 0"**, esto quiere decir que el eje de giro se encuentra en el plano *y*, por eso el valor de 1 en *y*. En limits se define **effort="100"** y **velocity="100"**. Y en dynamics se especifica los siguientes atributos **damping="0.0" friction="0.0".**

Para este caso los joints permiten asociar un actuador y una transmisión entre ambos para que el objeto se pueda controlar y manipular como tal. Para esto se agrega la siguiente información después del contenido de <joint>:

```
<transmission name="${sd}_trans">
    <type>transmission_interface/SimpleTransmission</type>
      <joint name="rued_front_${sd}_union_cilin">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
      </joint>
      <actuator name="${sd}Motor">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>10</mechanicalReduction>
      </actuator>
</transmission>
```

Se observa que la etiqueta <transmision> contiene el atributo **name**, el cual crea una transmisión dependiendo del valor en **sd**; transmisión izquierda y derecha. En la siguiente línea se especifica el tipo de transmisión, en este caso es **SimpleTransmission**. Se agrega la etiqueta de <joint> para especificar qué unión o articulación tendrá la transmisión y dentro de esta misma etiqueta se define el tipo de interfaz de hardware que se ocupa, en este caso es **EffortJointInterface**. En la etiqueta que sigue se especifica el actuador con el

atributo **name**. De igual manera dentro de esta etiqueta se especifica la **interfaz de hardware** y también la **reducción mecánica**,con valor de **10**. Y con eso termina la sección de la transmisión.

Por otro lado, dentro de la etiqueta `<gazebo>` además de especificar el tipo de material que en este caso será `Grey1`, también se especifican nuevas etiquetas que son plugins o extensiones de Gazebo, esto es con el objetivo de simular los diferentes comportamientos de los objetos en contacto con el suelo. Así entonces el apartado de gazebo queda de la siguiente manera.

```
<gazebo reference="rued_front_${sd}">
      <mu1 value="1.0"/>
      <mu2 value="1.0"/>
      <kp  value="10000000.0" />
      <kd  value="1.0" />
      <fdir1 value="1 0 0"/>
      <material>Gazebo/Grey1</material>
  </gazebo>
```

Para las demás ruedas que son dos las que faltan **rued_centro** y **rued_tracera**, se sigue el mismo procedimiento solo con la diferencia de valores en los atributos **xyz** dentro de la etiqueta **<origin>** que está dentro de la etiqueta **<joint>** y además el tipo de unión para estas ruedas es que serán estáticas **type= “fixed”**.

Y como último paso queda la construcción del eje o soporte exterior de las llantas **eje_ext**, que se elabora de la misma manera que el **eje_int**.

Ya como último paso se ejecuta el escenario de mundo y los archivos correspondientes para iniciar la simulación, recordar que antes de ejecutar el escenario de mundo debe estar activo el nodo maestro en una terminal.

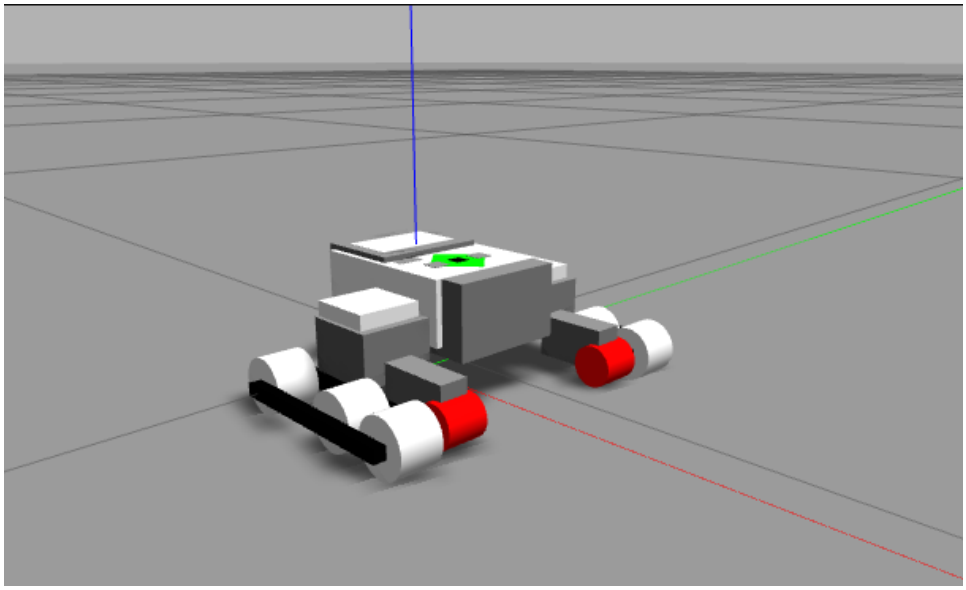


Figura 41. Captura de pantalla de Gazebo que muestra el modelado final del robot EV3 en 3D.

Tras ejecutar el escenario de mundo se abrirá el programa de Gazebo y observaremos el modelo simulado que se asemeja al robot EV3, figura 41.

Si se encuentra un error en terminal, revisar el código fuente para la simulación del robot EV3[18].

3.3 Instalación de ROS en Raspberry Pi

La tarjeta Raspberry Pi versión 4, modelo B, es la más adecuada para este proyecto porque los recursos con los que cuenta dan mayor posibilidad de que funcione ROS en su versión más actual sin ninguna restricción, además de que este tipo de tarjeta es portable y por lo tanto no exige un gran consumo de energía, así como tener la oportunidad de instalar un SO basado en Linux compatible con ROS. Por otro lado,

[18] Los archivos de la simulación se encuentran disponible en: https://drive.google.com/drive/folders/1xQB5UFZ-vJdYrjhX_lqX_VlO-2KZdlIP?usp=sharing

la Raspberry Pi proporciona diversos puertos de entrada, puerto de comunicación, entre otros. Con todo lo anterior y lo que más adelante se describe de la Raspberry Pi son las razones por las cuales se elige trabajar con esta tarjeta de desarrollo.

Actualmente Raspberry Pi cuenta con tres modelos, los cuales solo se diferencian en el tamaño de la memoria RAM [35].

La tarjeta de desarrollo que se ocupa cuenta con un procesador de dos núcleos con arquitectura de 64 bits a 1.5 GHz, así mismo, la tarjeta puede contar con una memoria SDRAM de 2, 4 u 8 GB según sea su versión. Estas características la hacen adecuada para soportar ROS en su versión desktop. Para almacenamiento de información cuenta con una ranura para microSD y se recomienda una capacidad de 32 ó 64 GB, ya que para almacenamiento más grande se tiene que realizar un formateo especial de la microSD. Por la parte de la comunicación, tanto alámbrica como inalámbrica, cuenta con un puerto gigabit Ethernet, tarjeta de bluetooth v 5.0 y con una tarjeta wireless IEEE 802.11ac en bandas de 2.4 y 5 GHz. Para su alimentación cuenta con una entrada USB tipo C y un pin de conexión, ambos aceptan 5 V DC con un mínimo de 3 A. Cabe mencionar que cuenta con 4 puertos USB y 2 puertos micro HDMI. De todos los puertos, solo se va a utilizar el Ethernet para comunicación y acceso a internet. El USB tipo C para la alimentación.

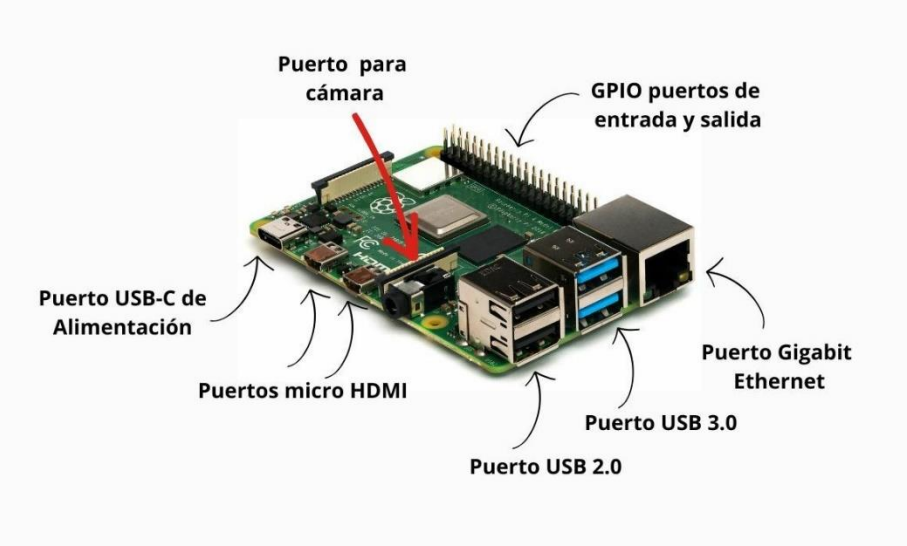


Figura 42. Puertos relevantes de Raspberry Pi 4 modelo B.

Para este trabajo, se usa la versión Raspberry Pi 4 modelo B de 4 GB de RAM. Hasta el momento, la Raspberry Pi modelo 4 en todas sus versiones son las de mayor rendimiento.

Antes de realizar la instalación de ROS Noetic en la Raspberry Pi se toma en cuenta dos puntos relevantes:

1. Dado que la Raspberry Pi 4 es una minicomputadora que tiene limitantes, en este proyecto solo se instalan paquetes tipo **Desktop.** Esta es una de la gran variedad de paquetes de ROS que se pueden instalar.
2. Este paquete de ROS solo es compatible con la versión Búster, distribución de Debian. Esto implica que versiones diferentes como Raspberry Pi OS Debian Bullseye no es compatible con esta versión de ROS.

Los recursos que se necesitan para realizar la instalación de ROS en Raspberry Pi son los siguientes: una PC de escritorio o laptop, una tarjeta microSD de 32 o 64

GB junto con su adaptador USB o SD, un cable UTP (Ethernet), un modem donde se encuentre conectado el equipo de cómputo a Internet, un eliminador de 5 V con mínimo 3 A con su cable tipo C y dos software: Raspberry Pi Imager y VNC Viewer. El siguiente paso y más importante es instalar el SO en la Raspberry PI, como se mencionó anteriormente es la versión Buster. El SO se instala en la microSD con ayuda del software Raspberry Pi Imager. Para más información, revisar la guía práctica[19]**.** Los siguientes pasos solo se realizan cuando por primera vez se inicia la tarjeta Raspberry Pi 4.

Ya que se tiene lista la microSD con el SO adecuado, esta se debe conectar a la PC para habilitar el protocolo SSH de la Raspberry Pi. Este tipo de conexión que significa Secure Shell, permitirá a la Raspberry Pi conectarse vía remota a la PC, esto es con la ventaja de no conectar un monitor, un teclado y un mouse a la tarjeta Raspberry Pi 4. Una vez conectada la microSD a la PC, se ingresa al interior de la microSD usando el explorador de archivos de la computadora. Una vez reconocida la memoria microSD en la PC, se elige la partición que se creó con el nombre de **boot**, la figura 43 muestra un ejemplo del contenido de esta partición.

[19] Sección A de la guía práctica.

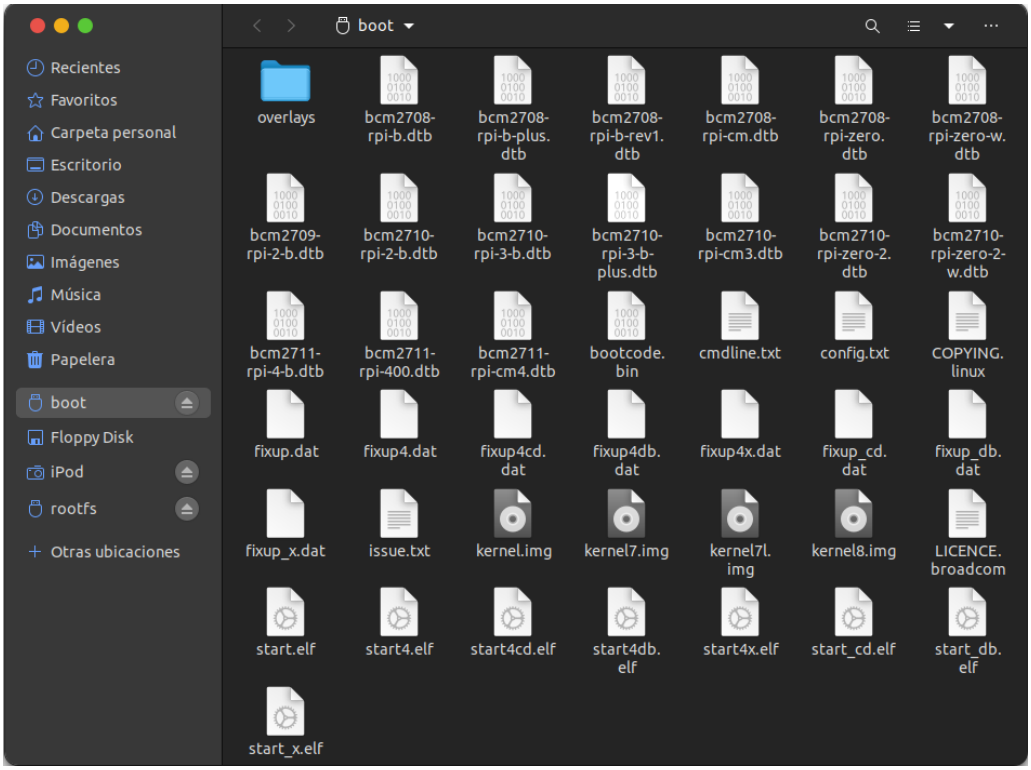


Figura 43. Contenido de la memoria microSD de la partición boot mostrado en el explorador de archivos.

Ya estando en la partición se da clic derecho y se selecciona la opción de abrir en una terminal, se abre automáticamente una terminal nueva y seguidamente se ejecuta el comando:

```
gedit ssh
```

Ya que se abre el editor de textos, solo se da clic en guardar y se cierra la ventana e igual la ventana de la terminal. Así mismo, observar el contenido de la partición que efectivamente se creó el archivo **ssh**.

Ahora, se procede a desmontar la microSD de la PC e insertarla en la ranura de microSD que tiene la Raspberry Pi. La figura 44 muestra gráficamente la conexión que se realiza de la PC a la Raspberry.

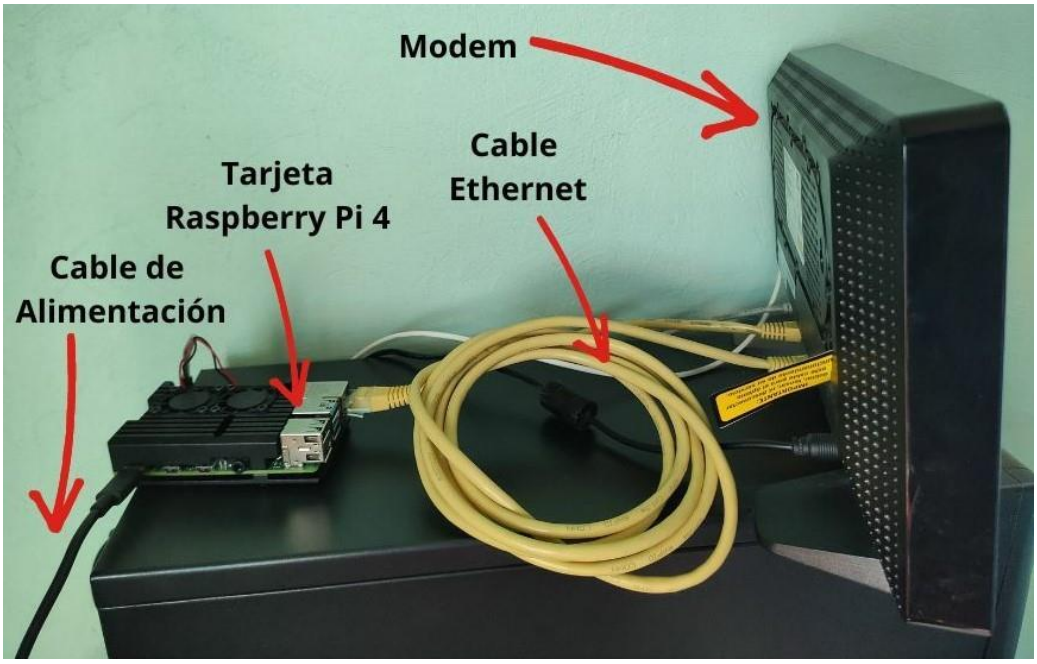


Figura 44. Conexión de la Raspberry Pi 4 con la PC por medio de un modem.

Por lo tanto, un conector del cable UTP va al puerto ethernet de la Raspberry y el otro va conectado al puerto ethernet del modem o router, por último, conectar la alimentación en el puerto tipo C de la Raspberry. Ahora se necesita saber la dirección IP asignada a la Raspberry, para ello simplemente se ingresa a la configuración del router o modem por medio de un navegador de Internet. En la opción de dispositivos conectados mostrará la Raspberry conectada junto con la IP asignada, como se muestra en la figura 45.

Dispositivos Locales

Estado	Tipo de Conexión	Nombre del Dispositivo	Dirección IP	Dirección Hardware	Tipo de Asignación de IP	Borrar
Activo	Inalámbrico(2.4GHz)	XiaoMiRepeater_V3			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Inactivo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Inactivo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	Mi9T-Mi9T			DHCP	Borrar
Inactivo	Inalámbrico(5GHz)	Unknown_			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Inactivo	Inalámbrico(2.4GHz)	Galaxy-A10s			Estática	Borrar
Inactivo	Inalámbrico(2.4GHz)	Home			DHCP	Borrar
Activo	Cable Ethernet	martintg-GA-78LMT-S2PT			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	BRW5C61994009EE			DHCP	Borrar
Activo	Inalámbrico(2.4GHz)	android-2e3cb03d97303b2b			DHCP	Borrar
Inactivo	Inalámbrico(2.4GHz)	Unknown_			DHCP	Borrar
Activo	Cable Ethernet	raspberrypi	192.168.1.82	e4:5f:01:3d:6f:16	DHCP	Borrar

Figura 45. Tabla de dispositivos conectados por Ethernet y Wifi al modem.

Ya que se cuenta con la IP, en este caso **192.168.1.82**, se procede a abrir una terminal nueva para ejecutar la siguiente línea:

```
sudo su ssh pi@192.168.1.82
```

El comando **sudo su** permite tener privilegios de administrador necesarios para conectar vía ssh la PC a la Raspberry. Tras ejecutar la instrucción anterior se pedirá

la contraseña de administrador que se le haya asignado a la PC, se escribe.

La instrucción que sigue después de **sudo su** es la que se encarga de realizar la conexión remota por medio del protocolo SSH de la PC a Raspberry. Seguidamente se despliega un mensaje preguntando si está seguro de continuar con la conexión, solo escribe "*yes*" y después se oprime la tecla enter. Consecutivamente se pide una contraseña, por default los datos de todas las Raspberry tienen como **usuario: pi** y como **contraseña: raspberry**. Por lo tanto, se ingresa la contraseña y se despliegan mensajes de autenticación, se espera hasta ver en la terminal el cambio de nombre de usuario y equipo en la terminal, en este caso se debe mostrar como: **pi@raspberrypi:~$**, esto nos confirma que la conexión por medio de SSH ha sido correcta, como se muestra en la figura 46.

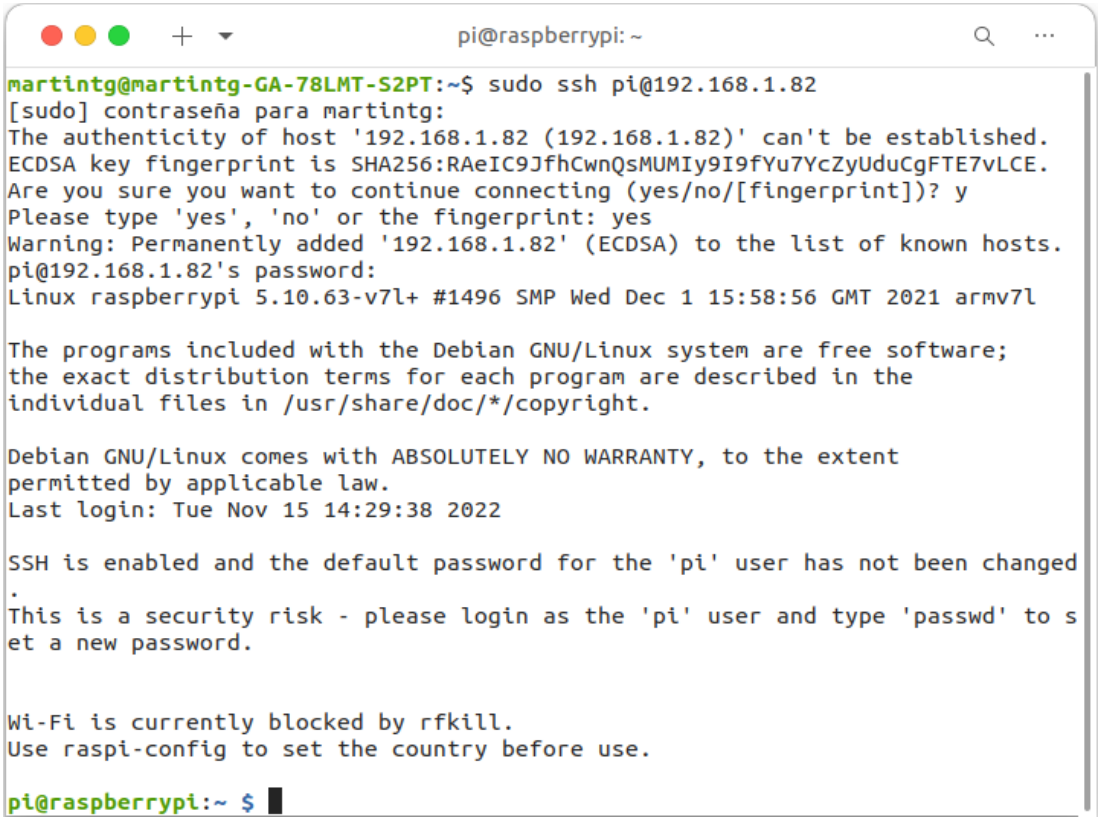


Figura 46. Conexión vía ssh de la PC a la Raspberry vista desde la terminal.

Teniendo comunicación desde la PC a la Raspberry Pi, se puede empezar a realizar las configuraciones pertinentes e instalar ROS, pero resulta ser un poco complejo el realizar todas esas actividades por medio de comandos. Es mejor y más fácil contar con una interfaz gráfica para ver el escritorio del SO de la Raspberry y así realizar las configuraciones de inicio y pruebas pertinentes para el buen funcionamiento de ROS. Para esto usar el software VNC Viewer. Antes de usar este programa primero se debe habilitar **VNC Server** en la Raspberry, para ello se ejecuta la línea de comandos en la terminal abierta:

```
pi@raspberrypi:~$ sudo raspi-config
```

Este comando muestra una ventana con un menú de opciones, que es la

herramienta de configuración del software de Raspberry Pi, un ejemplo de ello se muestra en la figura 47. En este caso, se elige la opción **Interfacing Options** y después la opción **VNC**, en seguida pregunta si se desea habilitar VNC Server, se elige la opción *YES* para confirmar y seguidamente pulsamos la tecla enter. Por último, se muestra un mensaje que notifica que VNC Server fue habilitado, se confirma con OK tecleando de nuevo la tecla enter. Regresando al menú inicial y seleccionamos con las teclas de desplazamiento la opción **Finish** para salir de las configuraciones de la Raspberry y volver a la terminal.

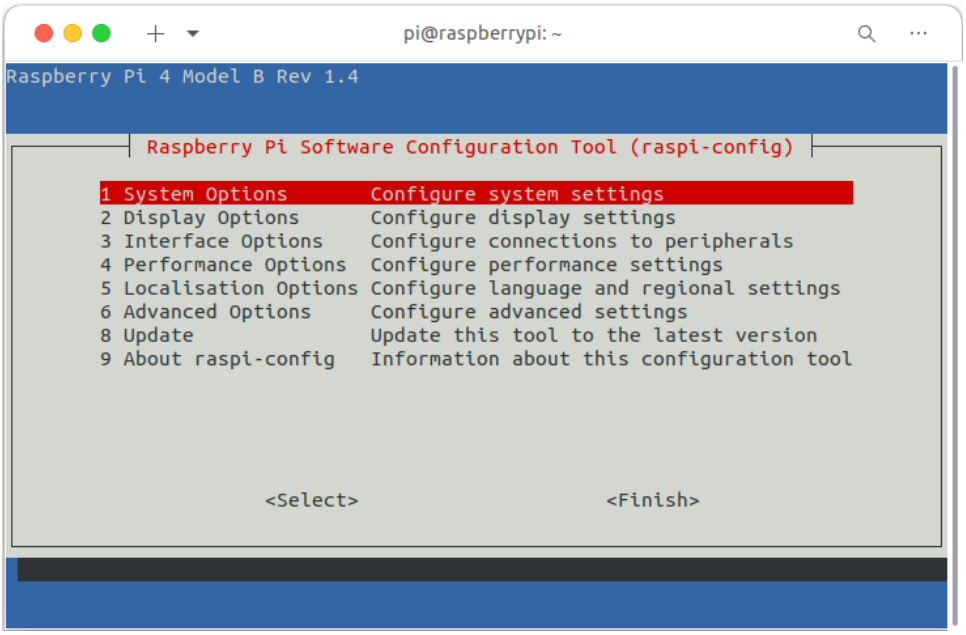


Figura 47. Ventana de herramientas de configuración de Raspberry Pi.

Ahora en la PC se busca en el cajón de aplicaciones, el software VNC Viewer, que previamente se descargó e instaló. Esto es para ver gráficamente el escritorio de Raspberry. Una vez abierto el programa, figura 48, en el área superior de la ventana

donde dice ***Especifique una dirección de VNC Server***, escribir la IP que tiene asignada la Raspberry:

Figura 48. Ventana inicial de software VNC Viewer.

Ahora se abre una ventana de autentificación, figura 49, en donde se pide nuevamente un usuario y su contraseña, se escribe y se elige aceptar.

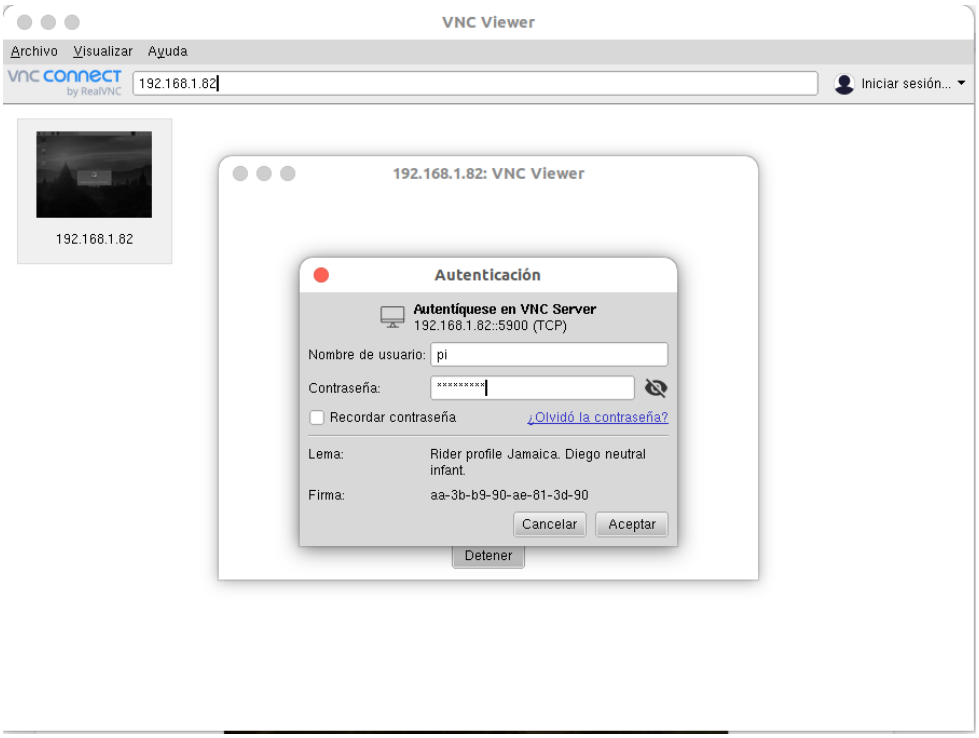


Figura 49. Realizando la conexión remota a la Raspberry Pi desde VNC Viewer.

Seguidamente se abre una ventana para mostrar el escritorio del SO de la Raspberry [20], figura 50.

[20] En caso de no mostrar el escritorio de la Raspberry, consultar la guía práctica Sección B

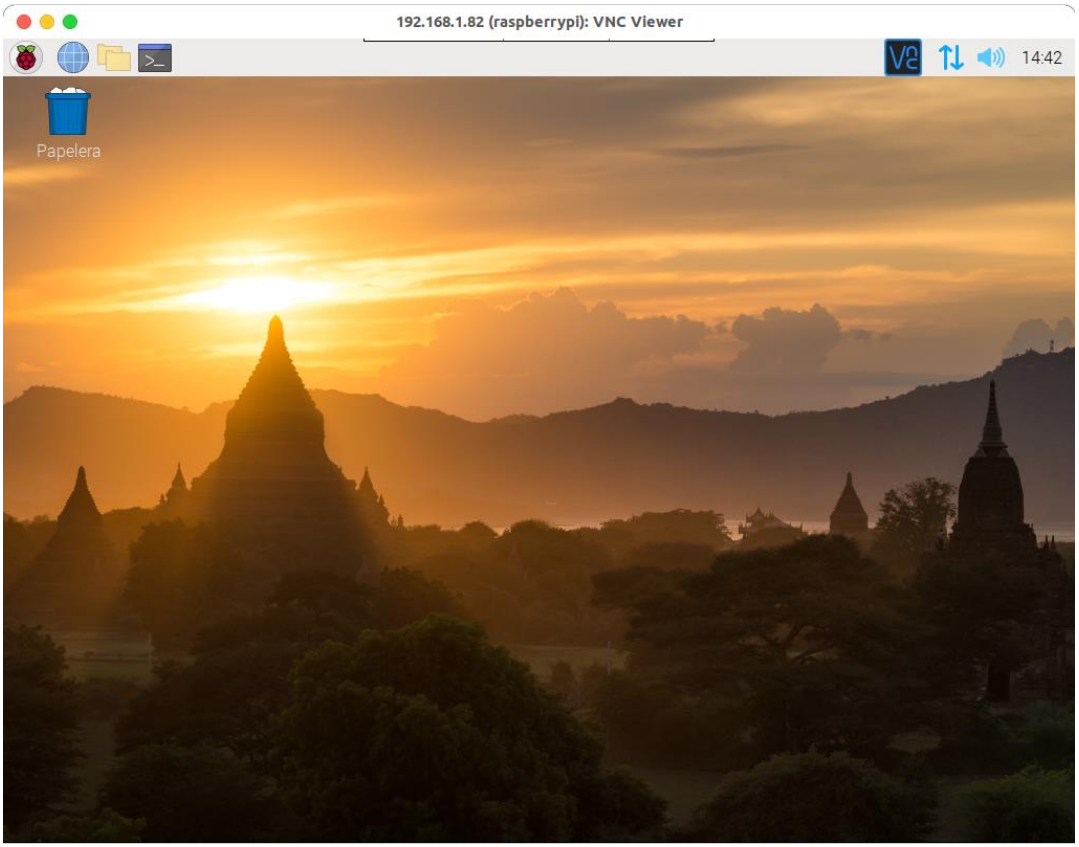


Figura 50. Escritorio de la Raspberry Pi con SO Debian versión Buster.

A continuación, se procede con la instalación de ROS Noetic en la Raspberry. El procedimiento de instalación se lleva a cabo por medio de una serie de comandos que se ejecutan en una terminal del SO. Toda la instalación se resume en los siguientes pasos:

1. Agregar el repositorio de ROS Noetic en la lista de fuentes del SO de la Raspberry.
2. Se ingresa una clave de ROS, esto es para descargar e instalar los paquetes de ROS Noetic de servidores oficiales y confiables, evitando así posibles hackeos ante el tráfico de datos en una red.

3. Se instalan las dependencias de compilación y herramientas necesarias para la compilación de los paquetes de ROS.
4. Se inician las dependencias y se actualizan. Seguidamente se crea un espacio de trabajo en donde se instalarán y compilarán los paquetes de ROS Noetic.
5. Se reúnen los paquetes necesarios y esenciales de Noetic para que estos sean compilados. Actualmente se han probado dos variantes de paquetes: **ROS-Comm** y **Desktop** con un funcionamiento estable.
6. Se procede con la instalación de ROS Noetic en su variante Desktop. Y también se procede a instalar dependencias de ROS.
7. Se aumenta el tamaño de la memoria SWAP para evitar errores durante el proceso de compilación.
8. Por último, se realiza la compilación de todo el contenido del espacio de trabajo.

Para verificar que sea han compilado correctamente los paquetes de ROS, en la misma terminal abierta se ejecuta el comando:

```
source /opt/ros/noetic/setup.bash
```

Seguidamente se ejecuta el comando `roscd`. Como resultado se debe de observar lo que se muestra en la figura 51.

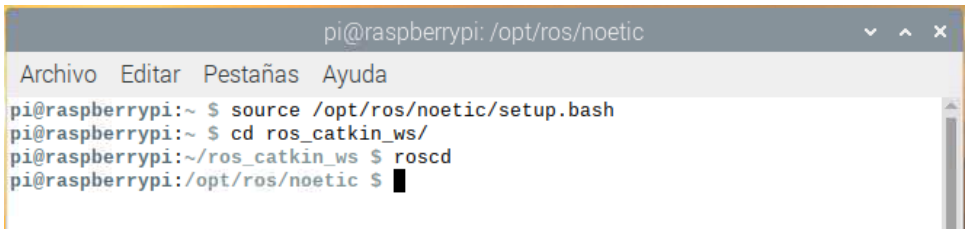


Figura 51. Ejecutando comandos de ROS para verificar la instalación.

Otra forma de proceder es iniciar el núcleo maestro de ROS. Para ello, abrir una terminal nueva y antes de iniciar el núcleo maestro, primero ubicarse en la dirección `~/ros_catkin_ws` con el comando `cd`. Estando en ese directorio, ejecutar los siguientes comandos por separado:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
roscore
```

El primer comando es para cargar las variables del entorno de ROS de manera automática y el segundo es para iniciar el núcleo maestro, así como se muestra en la figura 52.

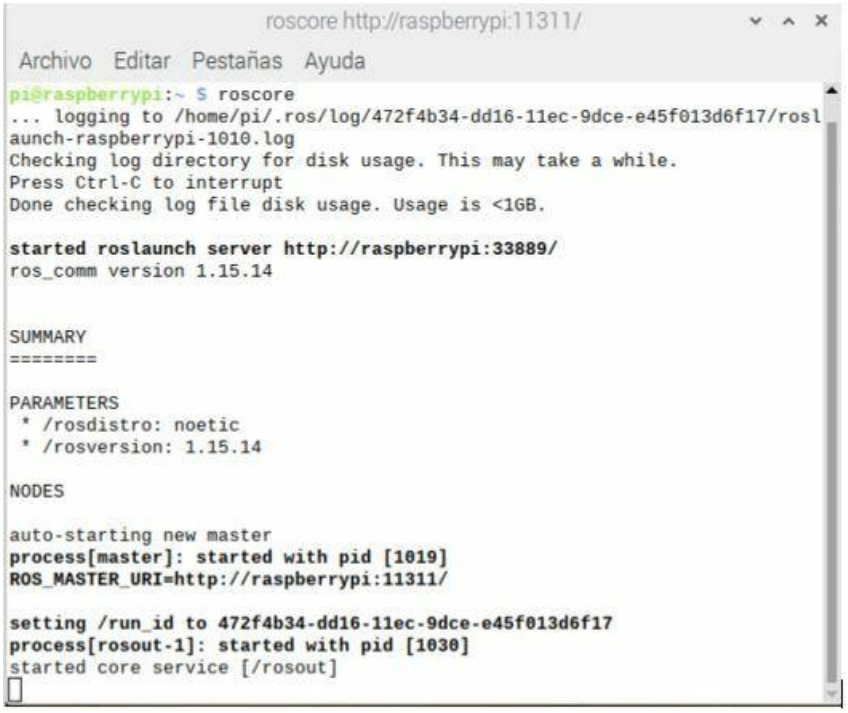


Figura 52. Iniciando el Core maestro de ROS.

Con esto se tiene a ROS Noetic funcionando en la Raspberry PI 4 modelo B. En la guía práctica se encuentra de manera detallada toda la instalación de ROS Noetic en Raspberry Pi, así como dos tutoriales para realizar pruebas básicas para iniciar el paquete turtlesim y otro para crear un nodo publicador y suscriptor, para verificar el funcionamiento de ROS[21].

3.4 Soluciones de comunicación en ROS

En esta se parte del capítulo, se toman en cuenta dos soluciones relacionadas con la comunicación de ROS con dispositivos externos. En este trabajo se pretende que

[21] Sección C de la guía práctica.

el dato que produzca el robot tras identificar un rostro humano se envié a un servidor web.

3.4.1 Comunicación con módulos Xbee Serie 1 Pro

Para realizar la comunicación entre ROS y módulos Xbee se sigue el tutorial que se encuentra en la página oficial de ROS, esto es con el afán de aprender a configurar los módulos Xbee, así como para comprender el funcionamiento del paquete ros-serial.

De manera resumida el tutorial describe a detalle la construcción de una red que consta de: una computadora, en la cual se ejecuta ROS y a su vez tiene conectado un módulo XBee y, por otro lado, se tienen dos nodos, los cuales son dos radios Xbee configurados de manera que envían cierto mensaje a la computadora. Estos radios están conectados por medio de una shield a una tarjeta Arduino la cual se programa para que envíe mensajes [36], figura 53.

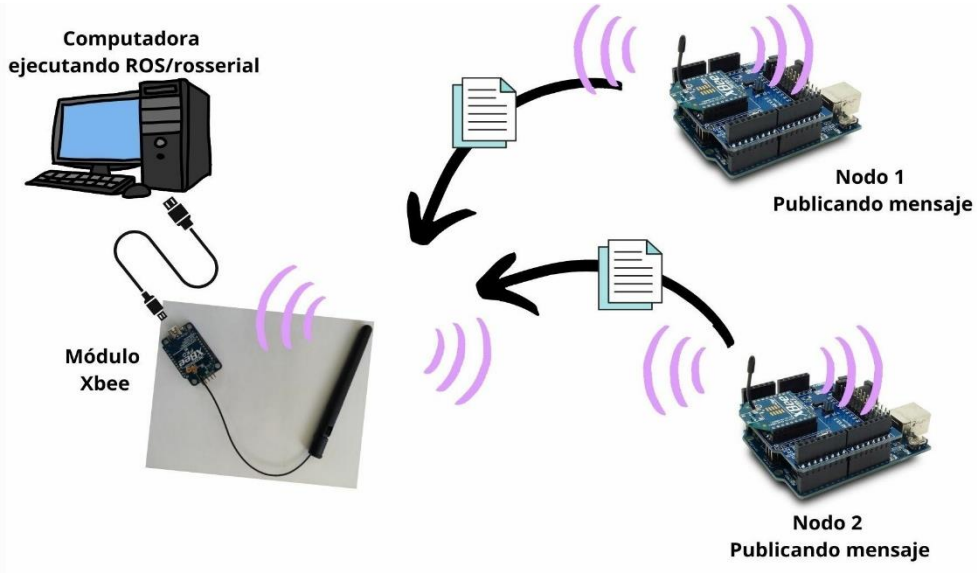


Figura 53. Diagrama de conexión del ejemplo de ROS con módulos Xbee.

Como primer paso, se descarga e instala Arduino IDE en la PC, se descargan los archivos necesarios del protocolo rosserial, así como las bibliotecas necesarias para el buen funcionamiento de la comunicación de ROS y los radios. Ya por último, se instalan las librerías en Arduino IDE para que exista la interacción entre Arduino y ROS.

Se continua con el tutorial y al momento de realizar las configuraciones pertinentes de los módulos Xbee, se mostrará un conflicto con la versión de python que se tiene instalada en la PC, mismo que se resuelve realizando una configuración en la que se puede cambiar de versión de python. Se continua con la programación de Arduino para crear un nodo de ROS que envíe un mensaje. Por último, se inicia la red por medio de la terminal y como resultado solo se observan errores. Se investigó cómo resolverlos, pero al final se detectó que los archivos de rosserial no se encuentran actualizados para la versión ROS Noetic, dado que los scripts de rosserial se encuentran codificados en python 2 y ROS Noetic trabaja con python 3, por lo que existe un gran conflicto y no se puede iniciar la red. Ante ello, se buscaron otras opciones de comunicación.

En este proyecto la conexión que se planteaba realizar usando módulos Xbee se muestra en la figura 54.

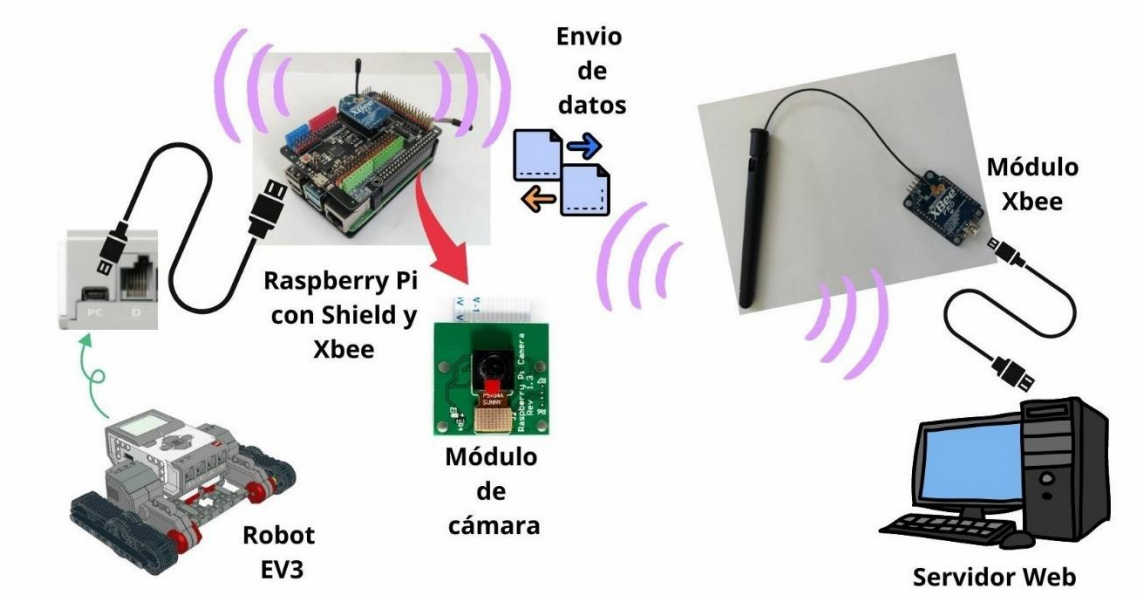


Figura 54. Diagrama de comunicación planeada para este trabajo usando módulos Xbee.

3.4.2 Comunicación usando la tarjeta NodeMCU con un módulo ESP8266

Para este caso se probaron configuraciones del módulo ESP8266 con ROS, esto con la finalidad de que éste pueda enviar datos que son detectados por el robot a una base de datos. En la guía práctica se explican a detalle las configuraciones iniciales de NodeMCU y un ejemplo para que esta tarjeta trabaje con ROS[22].

De acuerdo con lo que se planteaba con los módulos Xbee, se requiere de un servidor web con base de datos que reciba y tenga un historial de la información obtenida por el robot EV3. Esta información será enviada por un NodeMCU, que se encuentra programado como nodo suscriptor y para enviar los datos obtenidos al servidor. Dado que todos los dispositivos deben estar comunicados

[22] Sección I de la guía práctica.

inalámbricamente, todos los dispositivos forman parte de una misma red. Para ello, se elabora una red local, esto se realiza con ayuda de un NodeMCU que se programa en modo Access Point (AP), figura 55.

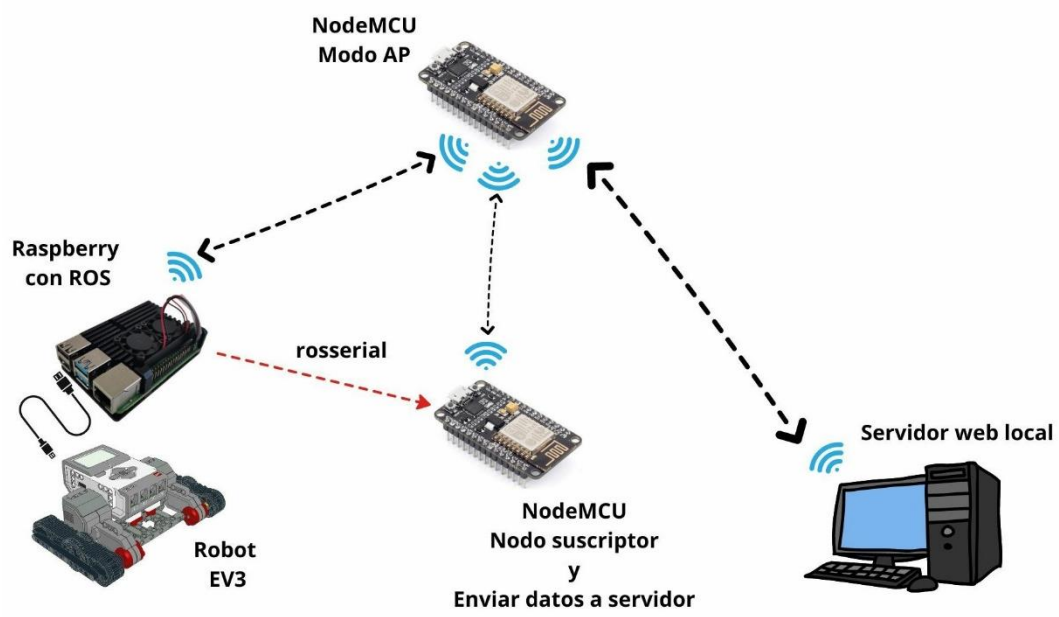


Figura 55. Esquema de la red WLAN para el sistema de búsqueda de personas usando NodeMCU.

De acuerdo con el diagrama anterior se requiere de dos NodeMCU para realizar la comunicación entre el robot y el servidor web local.

Para configurar el NodeMCU como AP se utiliza el código de programación que se muestra en la página web [37]. En este caso solo se modifican los datos que corresponden al nombre de la red local a crear y la contraseña:

*ssid = "**RED_NodeMCU**";

*password = "**Is3tu4cm**";

Se programa el NodeMCU que será AP con ayuda del software Arduino IDE y se carga el algoritmo al NodeMCU. Para corroborar que se programó de manera

correcta, con ayuda de cualquier dispositivo con conexión wifi, en la opción de conexión wifi debe aparecer en el listado de **Redes Disponibles** la red **RED_NodeMCU**. Cabe mencionar que la red por defecto que va a crear el NodeMCU en modo AP es una red de **clase C** con direcciones IP privadas que van de **192.168.4.1** a **192.168.4.254**, siendo la primera IP asignada a el NodeMCU en modo AP.

3.5 Configuración de una cámara en ROS

Para este trabajo se usa la cámara Raspberry Pi Rev. 1.3, figura 56. Esta cámara es pequeña, por lo que es generalmente utilizada para tareas que requieren la captura de imágenes o video en robótica, seguridad, monitoreo y otros usos, con los cuales se trabaje en conjunto con la Rasberry Pi 1 o la Rasbperry Pi 4. La resolución de la cámara es de 5 Mega Pixeles (MP) y con una resolución de captura de video de 1080 pixeles a 30 fotogramas por segundo (fps) [38].

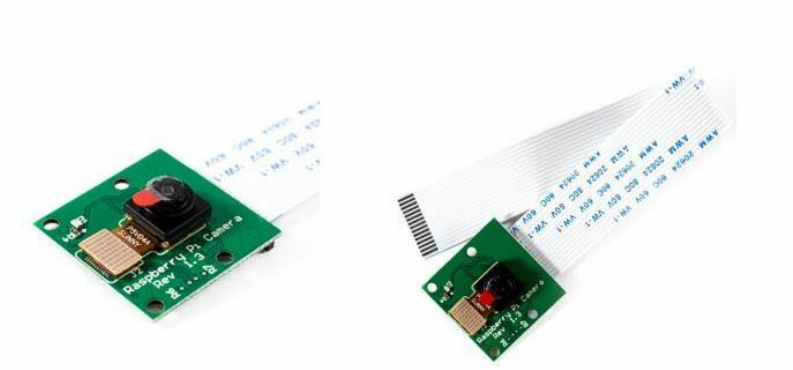


Figura 56. Cámara Raspberry Pi Rev 1.3 5MP con su flex.

La cámara cuenta con una membrana o flex que se conecta a un puerto de la

Raspberry Pi dedicado a la cámara. Para realizar la conexión, configuración y realizar pruebas de la cámara ya conectada con la Raspberry Pi, visitar el tutorial indicado en la referencia [39].

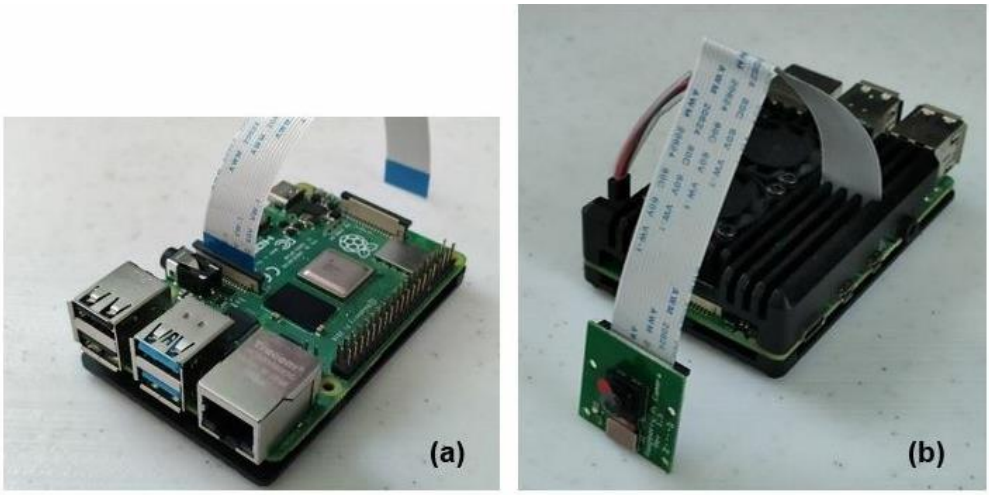


Figura 57. Instalación de la Cámara en la Raspberry Pi. (a) Conectando flex en el puerto asignado de la Raspberry. (b) Instalación finalizada de la cámara.

La Raspberry Pi empleada tiene instalado un case con disipador, entonces para colocar la cámara hay que quitar el case de la Raspberry Pi y después volver a colocarla. Una vez conectada la cámara en la Raspberry Pi, se le coloca un case y soporte de la cámara, figura 58, para tener una mayor protección del dispositivo y así mismo colocarlo de manera fija con el robot EV3.

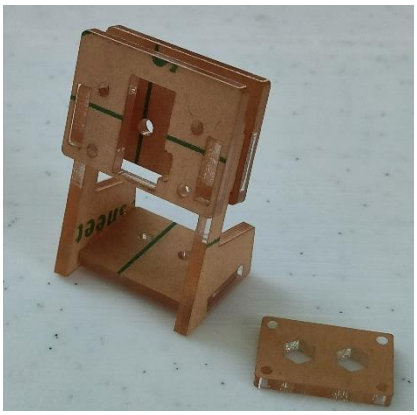


Figura 58. Base y protección para la cámara Raspberry Pi.

Tras la instalación del case y base queda como se muestra en la figura 59.

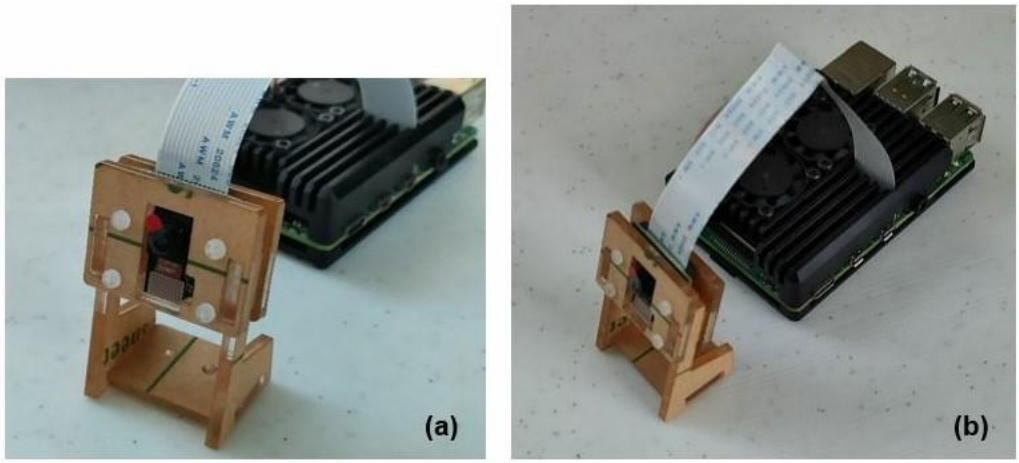


Figura 59. Base y protección instalada en la cámara. (a) Cámara en una posición fija. (b) Vista de lejos de cámara y Raspberry Pi.

El mismo ejemplo que se presenta en la guía práctica se puede realizar y funciona correctamente con esta cámara[23].

[23] Sección E de la guía práctica.

3.6 Conexión de Raspberry Pi y el robot EV3

De acuerdo con las especificaciones del robot EV3, el Brick (cerebro del robot) cuenta con posibilidad de comunicación inalámbrica, ya sea con Bluetooth o Wi-Fi. Con estas características el usuario puede controlar el robot a distancia y de manera remota, pero el uso de estas tecnologías tiene sus pros y contras. Para el caso de Bluetooth su alcance está limitado, pero no tanto para Wi-Fi, pero ambos consumen una cierta cantidad de corriente para su funcionamiento. Hacer uso constante de estas opciones de comunicación implica que la durabilidad de las baterías sea menor, lo que no es conveniente, dado que las baterías se encuentran enfocadas en primera instancia al movimiento del robot, en caso más específico a los motores; en segunda instancia a la recolección de datos mediante los sensores; y en tercera instancia a la comunicación.

Para este caso, se usa un método de comunicación cableado al Brick del EV3 con la tarjeta Raspberry Pi 4 modelo B, dado que se tendrá ésta última “a bordo” del robot, figura 60.

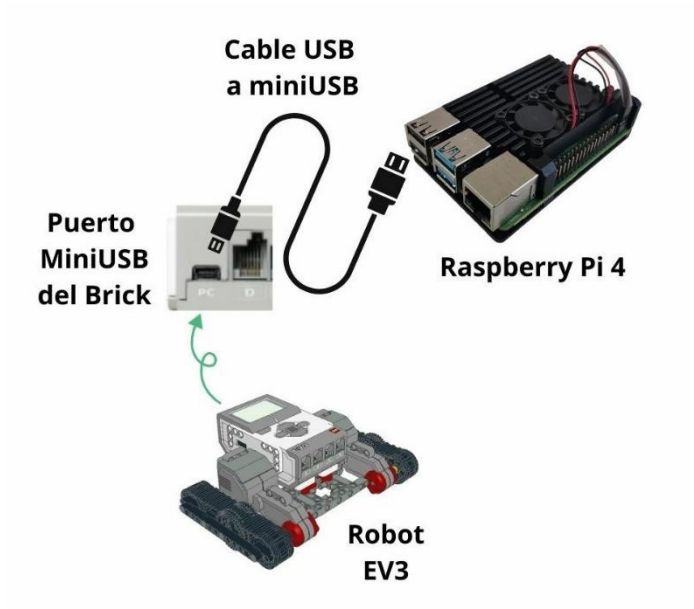


Figura 60. Conexión de EV3 con Raspberry Pi 4 por medio de un cable USB.

De manera más específica, el puerto mini USB del Brick se conecta a un puerto USB de la Raspberry. Como requisito importante el Brick debe de contar con el SO ev3dev, ya que este permite realizar la configuración para tener la comunicación entre el EV3 y la Raspberry Pi.

Este tipo de conexión se configura de tal manera que por medio del puerto USB se asigne una dirección IP, la cuál será estática. En la guía práctica se describen paso a paso las configuraciones pertinentes[24]. También se puede realizar el ejemplo de la guía práctica para hacer uso de RPyC y controlar al robot de manera remota desde a la Raspberry Pi[25].

[24] Sección F de la guía práctica.
[25] Sección G de la guía práctica.

3.7 Servidor web EV3

En este trabajo se elabora un servidor web local dado que la red que se propone es una red LAN inalámbrica formada por el NodeMCU en modo AP. Para crear este servidor se hace uso de la herramienta de software XAMPP y se instala en la PC de escritorio, para mayor información sobre cómo se instala y se inicia el servidor consultar [40].

Una vez instalado y funcionando el servidor, es momento de configurar la base de datos para comenzar a realizar la programación y las pruebas pertinentes con el NodeMCU para que este envié datos al servidor web y se almacenen en la base de datos; todo esto se encuentra detallado en la guía práctica[26].

3.8 Nodos del robot

Como bien se menciona en el Capítulo 2, ROS trabaja con la creación de nodos en los proyectos, esto es con la finalidad de reducir la complejidad y encontrar con mayor facilidad algún error que se presente. Dando el seguimiento a la forma que trabaja ROS, el robot se implementa a partir de subsistemas que son nodos de ROS. Estos nodos se desarrollaron en un trabajo previo [1], los cuales están organizados por seis grupos de nodos: adquisición de datos, generadores de datos y salidas, sistema de referencias, modelado del robot, visualización del robot y el

[26] Sección K de la guía práctica.

nodo launch que ejecuta todas la anteriores. En la figura 61 se muestra un diagrama de los nodos que componen al robot.

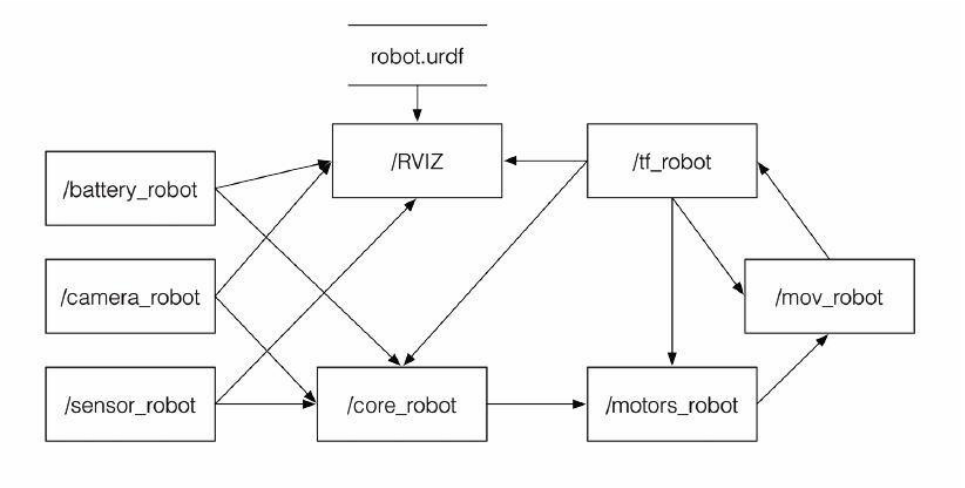


Figura 61. Diagrama que muestra los nodos del sistema de búsqueda [1].

Para el presente trabajo no se usan los nodos de sistema de referencias, modelado del robot y visualización del robot, que se desarrollaron en el trabajo previo, ya que en esta versión la computadora solo controla al robot, pero no se realiza una visualización simultánea del robot en un ambiente gráfico, de la cual se obtendría la ubicación del robot. También cabe mencionar que, de los nodos a usar, se realizaron algunos cambios y además se agregó un nuevo nodo que se encarga de la comunicación y envío de datos de ROS al servidor web local.

Con los cambios realizados y el nuevo nodo incorporado, el robot EV3 realiza la tarea de búsqueda de rostros humanos de la siguiente manera. El robot comienza de un punto de partida y verifica, por medio de la cámara, si se detecta un rostro

humano, de ser así se genera el mensaje de humano detectado, este mismo es enviado al nodo agregado que se encarga de mandar un valor entero al servidor web local, confirmando que se detectó un rostro y con esto finaliza la búsqueda. Si por el contrario no se detecta un rostro humano, el robot verifica, por medio del sensor IR, si existe algún obstáculo, si no es así, el robot avanza cierta distancia y se detiene. Después nuevamente verifica la detección de un rostro, si hay un obstáculo que impide el avance, el robot evade ese obstáculo y continúa avanzando. En todo momento siempre se verifica si se detecta un rostro o si existe algún obstáculo, por lo tanto, el robot sigue avanzando hasta que detecte un rostro y así finalizar la misión, figura 62.

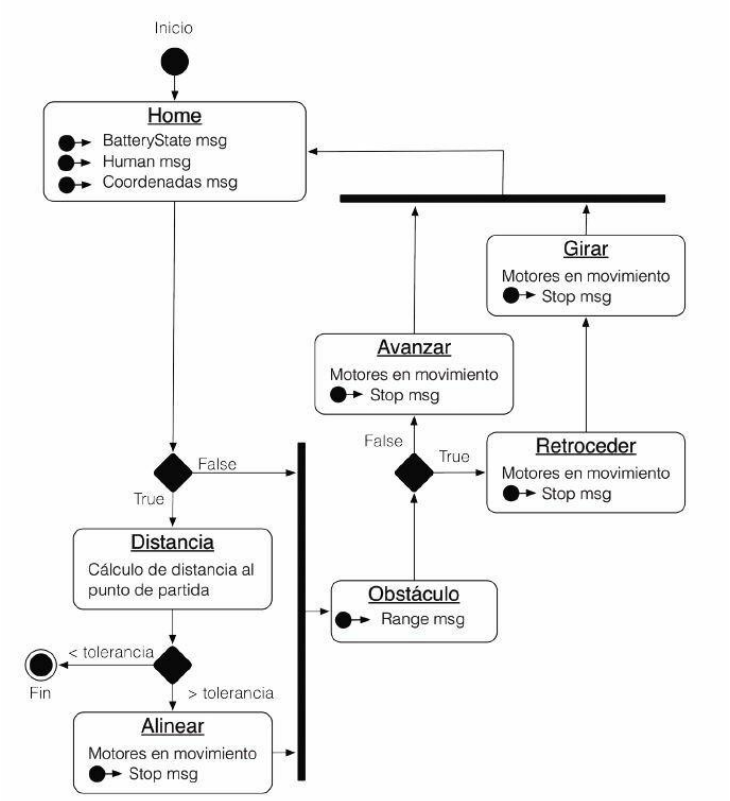


Figura 62. Diagrama de estados del sistema del nodo Core [1].

3.8.1 Nodo Core

Este nodo, como su nombre lo dice, es el núcleo o el nodo principal del robot, el cual se encarga de realizar funciones y toma de decisiones de acuerdo con los datos o información obtenidos por los nodos: batería, sensor y cámara. Es la razón por la que el nodo Core se encuentra regido por un programa **main**, que realiza la función de verificar el valor de una variable y otras funciones más. La variable puede tener dos valores. De acuerdo con lo anterior, el comportamiento del robot se define por dos estados. El primer estado es sobre el inicio de la búsqueda, que corresponde a que el robot avance y verifique si el sensor no encuentra obstáculo alguno y si lo hay lo evada. En este estado el nodo Core se comunica con el nodo sensor y el nodo Motores. El segundo estado es para finalizar la búsqueda, aquí se tienen dos posibilidades: a) cuando se detecta un rostro y b) cuando la alimentación no sea suficiente para continuar con la misión.

3.8.2 Nodo Batería

Este nodo es de gran importancia, ya que el robot es alimentado por 6 pilas recargables tipo AA que se van descargando con las funciones que se le asignan al robot; tales como mover los motores, hacer uso del sensor IR, reproducir un sonido en la bocina integrada en el robot, entre otras funciones básicas. Con todo lo anterior es relevante llevar un monitoreo sobre el estado de nivel de carga de las baterías. Es por ello, que este nodo especifico envía, cada cierto tiempo, un mensaje con el informe sobre el estado de las baterías al nodo Core. Así entonces

el objetivo de este nodo es que el robot cancele la búsqueda cuando detecte bajos niveles de batería y antes de que se quede sin energía, para ello se define un rango de tolerancia. En tal caso el nodo muestra un mensaje de “Baja energía de alimentación” y el mensaje de “Búsqueda cancelada”.

En la programación de este nodo los puntos más importantes son: El primero es definir la conexión entre el Brick del robot y la Raspberry Pi, para ello se hace uso de la librería RPyC y se coloca la IP asignada al puerto USB del Brick. El segundo es obtener la información sobre el estado actual de alimentación del robot, que se almacena en archivos específicos que se encuentran en el sistema de archivos del Brick. Así entonces, para saber la corriente actual, el voltaje actual y el voltaje máximo se debe de ingresar a las siguientes direcciones.

- **Corriente actual:** `/sys/devices/platform/battery/power_supply/lego-ev3-battery/current_now`
- **Voltaje actual:** `/sys/devices/platform/battery/power_supply/lego-ev3-battery/voltage_now`
- **Voltaje máximo:** `/sys/devices/platform/battery/power_supply/lego-ev3-battery/voltage_max_design`

En el nodo se declara una función para obtener la información de la alimentación del robot y esta se asigna a una variable para que sea guardada. La función para obtener los datos es `c.builtins.open()`, dentro del paréntesis se coloca la

dirección del archivo para obtener los datos, esta va entre comillas simples (‘’). Solo para el caso de corriente y voltaje actuales, se le agrega un atributo r, que también va entre comillas simples. Este valor es para indicar que se va a leer la información del archivo especificado, el atributo se coloca después de la dirección agregando antes una coma, por ejemplo:

```
voltage_now =
c.builtins.open('/sys/devices/platform/battery/power_supply/lego-
ev3-battery/voltage_now', 'r')
```

En la figura 63 se muestra el diagrama de flujo par este nodo.

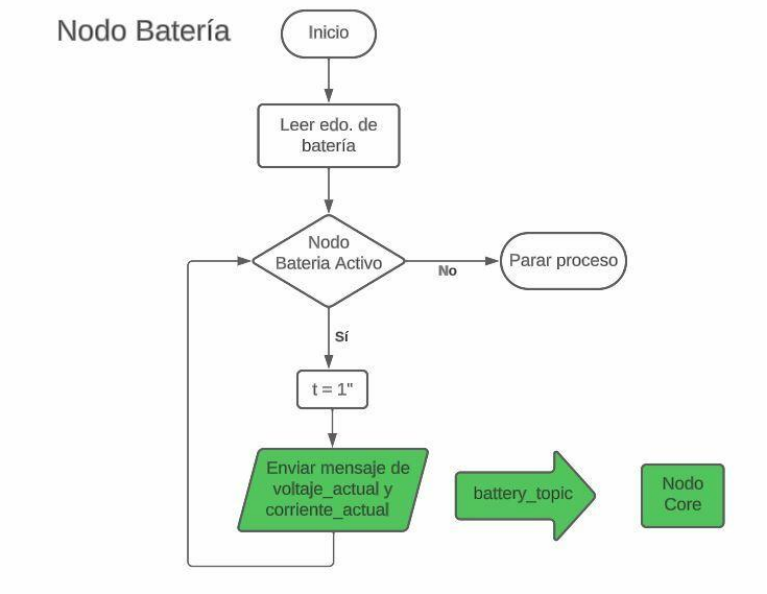


Figura 63. Diagrama de flujo del nodo battery_robot [1].

3.8.3 Nodo sensor Infrarrojo

Este nodo se encarga de realizar la detección de obstáculos. En resumen, la función de este nodo comienza realizando una conexión con el robot y enseguida lee los datos obtenidos por el sensor IR. El sensor se puede encontrar en dos situaciones:

obstáculo detectado y no detectado, esta es la información que se le envía al nodo Core, el cual toma la decisión de ordenar al robot si debe de avanzar o realizar una maniobra para eludir el objeto que impide el avance. La información sobre el estado del sensor se envía en un tiempo definido al nodo Core. En la figura 64 se muestra el diagrama de flujo de este nodo.

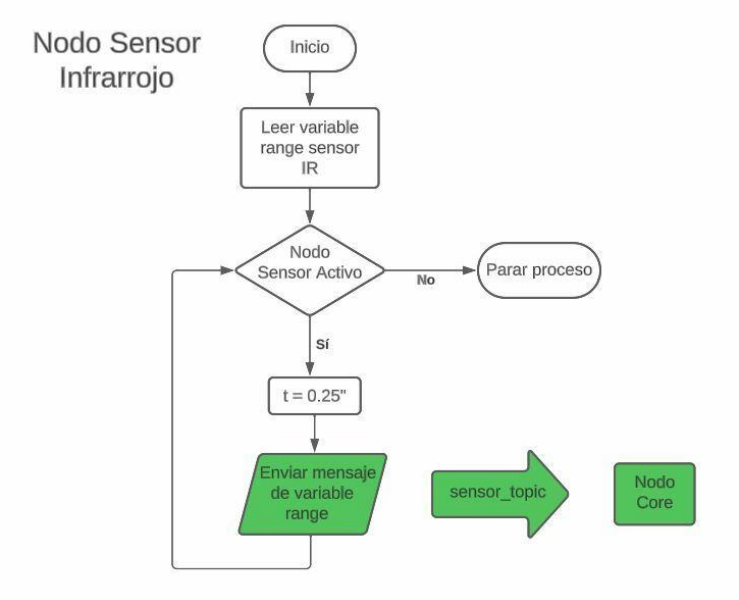


Figura 64. Diagrama de flujo de nodo sensor_robot [1].

3.8.4 Nodo Motores

Este nodo se encarga de realizar el movimiento de los motores del robot, de acuerdo con las ordenes que envíe el nodo Core, es decir, se define en qué momento debe avanzar el robot, girar y retroceder para evadir un obstáculo. Por otra parte, el nodo Motores se comunica con el nodo Movimiento, para que a este se le notifique con base en las rotaciones de los motores cual fue su avance y así mismo en qué momento se detuvieron [1].

3.8.5 Nodo Cámara

Una vez que se revisó y elaboró el ejemplo sobre la librería de OpenCV en ROS[27], se procede a realizar el script del nodo Cámara. Este nodo se encarga de realizar la detección de rostros por medio de una cámara y al momento de identificar un rostro, este mismo nodo envía un mensaje al nodo Core confirmando la detección del rostro. Para hacer posible la detección de rostros se emplea el aprendizaje maquinal (*machine learnig)*[28] en la Raspberry Pi usando el método Haar Cascades. OpenCV proporciona varios clasificadores no solo para reconocer rostros sino para detectar ojos, sonrisas y más[29]. Por otra parte, también se hace uso de una función llamada detecMultiScale que se encuentra dentro del método que se va a usar. Esta librería ayuda a realizar la detección por medio de un rectángulo delimitador, dentro del cual se hace el análisis para identificar un rostro u objeto en una imagen o un video. Es importante saber que la detección se realiza con mayor eficiencia si la imagen o video se encuentra en escala de grises [41]. De acuerdo con lo anterior, en este nodo se hace la transformación a escala de grises de cada frame que es captado por la cámara y una vez hecho eso ahora se procede a realizar la detección de rostros.

[27] Sección E de la guía práctica.
[28] Para más información consultar https://www.ibm.com/mx-es/analytics/machine-learning
[29] Clasificadores (algoritmos de reconocimiento) disponibles en https://github.com/opencv/opencv/tree/master/data/haarcascades

3.8.6 Nodo NodeMCU

Por último, el nodo NodeMCU es el que tiene la tarea de recibir información desde ROS sobre la identificación de un rostro y enviar la notificación a un servidor web local. Una vez que se recibe la información de detección, el nodo debe enviar el valor de acuerdo con la detección para indicar si se identificó o no un rostro. Después este valor es enviado al servidor web local donde es almacenado en su base de datos.

Para que este nodo funcione se requiere que el sistema completo (Raspberry, servidor, NodeMCU) estén conectados a una red LAN, se puede usar como ejemplo la red WLAN de la guía práctica[30].

El módulo NodeMCU se programa como un nodo suscriptor de ROS para recibir la información que envía el nodo Cámara al identificar un rostro. Por otro lado, también se programa para almacenar en una variable, la información de detección que recibió, y por último para enviar la variable al servidor web local[31]. Para más información sobre los algoritmos de programación de los NodeMCU's revisar la dirección web mostrada al pie de página[32].

[30] Revisar Sección K de la guía práctica.
[31] ibidem
[32] Los algoritmos están disponibles en:
https://drive.google.com/drive/folders/1oyfnDHhBTVkLcJCYgUyDlNPSY8vyJooI?usp=sharing

Capítulo IV Evaluación de prestaciones

En este capítulo se describen tres escenarios de prueba diseñados para evaluar las prestaciones de la implementación física del proyecto robótico. Para ello, se describe de manera general cada uno de ellos y se señalan las posibles complicaciones en la implementación, así como futuras modificaciones que se pueden realizar para mejorar el proyecto robótico.

4.1 Escenarios de prueba

El primer escenario es para verificar el funcionamiento de la cámara y del sistema de comunicación al que reporta información de detección. En este escenario se pone en funcionamiento el nodo de cámara y el nodo NodeMCU, mientras el robot se mantiene estático. Se corrobora el funcionamiento y comunicación entre el robot EV3, la Raspberry Pi y el servidor web. El objetivo en este escenario es que al momento que el robot EV3 identifiqué a una persona por medio de la cámara, se envíe un mensaje o dato al nodo NodeMCU, mismo que se guarda y envía a un servidor web local. La información recibida en el servidor web se almacena en su base de datos, la cual cuenta con un historial de los datos recibidos, figura 65.

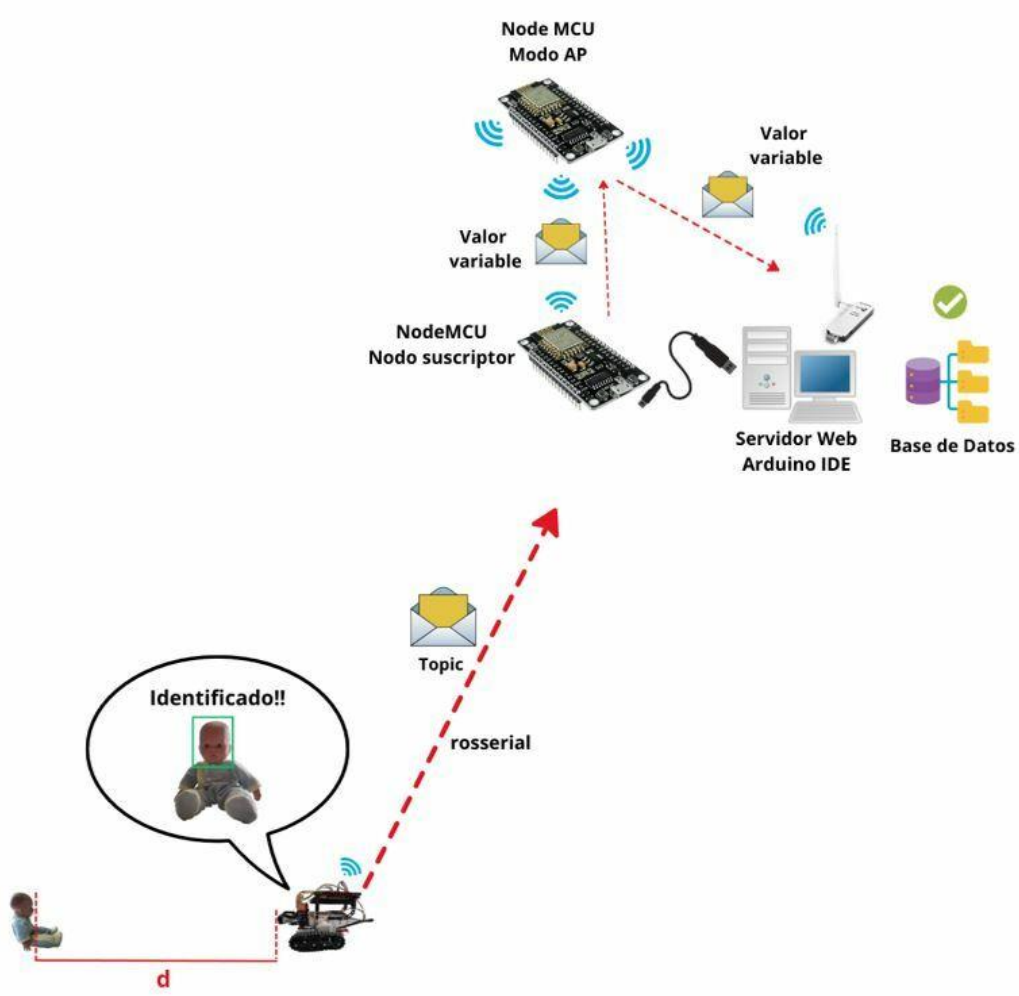


Figura 65. Esquema del funcionamiento del sistema de búsqueda, escenario 1.

En el segundo escenario el robot se mueve en un espacio libre de obstáculos. En esta prueba se utilizan e inician todos los nodos que compone el sistema de búsqueda, así mismo se coloca un bebe de juguete a una cierta distancia de la ubicación del robot EV3. A diferencia del primer escenario, en este caso el robot avanza en línea recta hasta identificar un rostro facial, se detiene y envía un mensaje al servidor web para notificar que se ha detectado un rostro, figura 66.

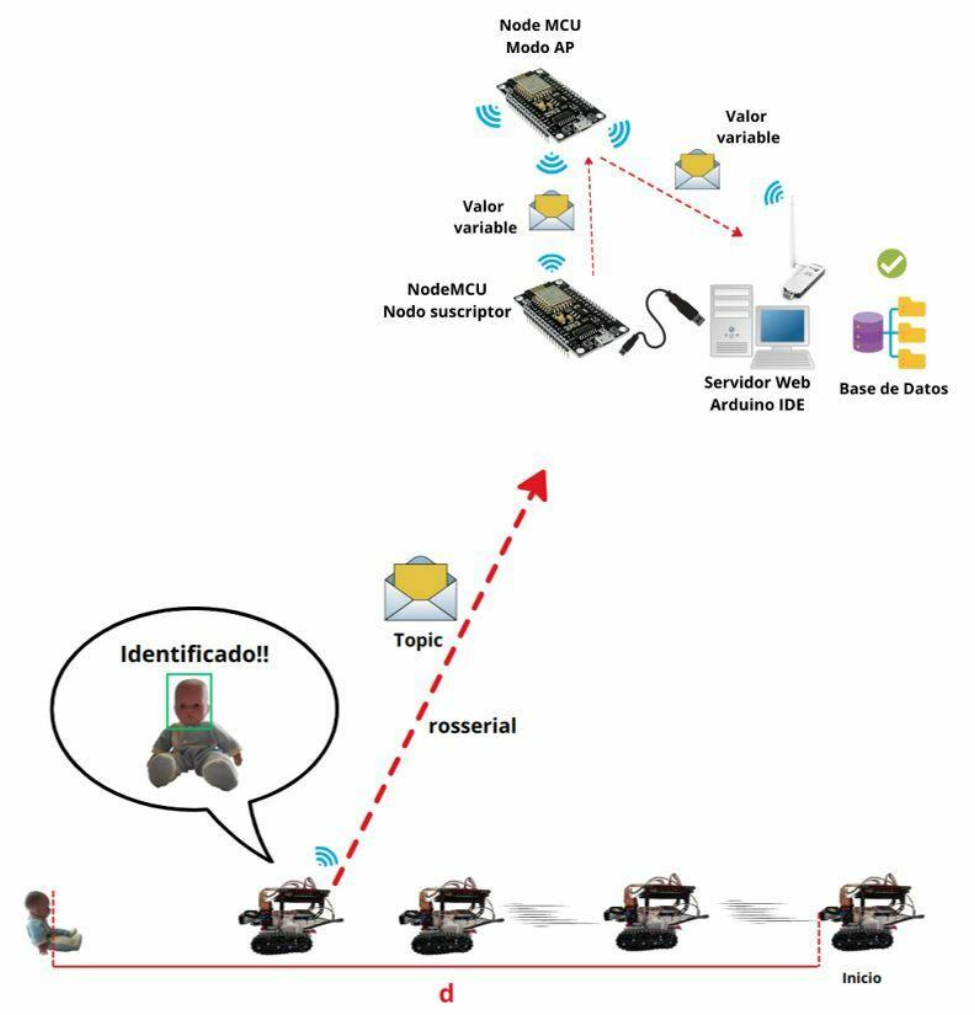


Figura 66. Esquema del funcionamiento del sistema de búsqueda, escenario 2.

El tercer escenario es parecido al anterior, pero ahora el espacio en el que se encuentra el robot cuenta con un obstáculo en su camino. Para la evasión del obstáculo el robot toma la decisión de girar a la izquierda o a la derecha y continuar avanzando hasta identificar un rostro. De igual manera al encontrar un rostro esto se notifica al servidor web local, figura 67.

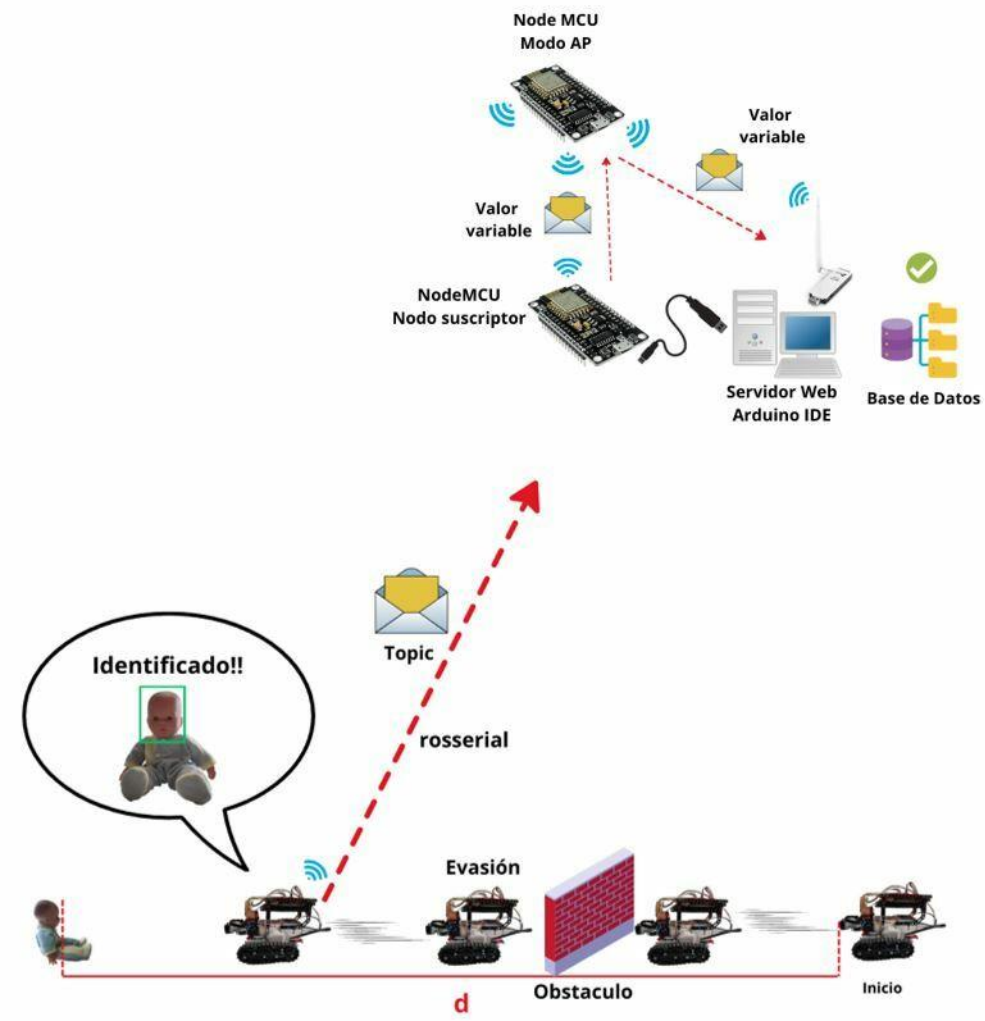


Figura 67. Esquema del funcionamiento del sistema de búsqueda, escenario 3.

4.2 Conexiones y herramientas de software

Previo a realizar los escenarios propuestos, es necesario realizar las conexiones pertinentes entre todos los dispositivos que compone el sistema de búsqueda e iniciar y conocer las herramientas de software que son necesarias.

Para el caso de herramientas de software se hace uso de HeidiSQL, Arduino IDE y VNC Viewer. HeidiSQL es un software libre que edita y crea bases de datos de

computadoras, llámese MariaDB, MySQL entre otros [42]; esta herramienta puede visualizar la información que se guarda y modifica en la base de datos.

Arduino IDE es un software que proporciona herramientas como: escribir, depurar, editar, programar y grabar código en un dispositivo Arduino para que funcione de acuerdo a las necesidades del desarrollador [43]; en este caso el software se usó para programar los NodeMCU's. Sirve además para tener la visualización de los mensajes que recibe el nodo NodeMCU y para verificar su funcionamiento. Por último, VNC Viewer es un software libre del tipo cliente – servidor que permite visualizar e interactuar con un equipo (servidor) desde cualquier pc o dispositivo móvil (cliente) siempre y cuando se encuentren en la misma red. Este software es multiplataforma por lo que ayuda a tener acceso y control remoto desde cualquier equipo [44] [45]. Esta herramienta sirve para tener el control y visualización de la Raspberry Pi de manera remota, para ello es necesario que tanto el equipo servidor y cliente estén dentro de una red. Cabe mencionar que dichas herramientas de software están instaladas y se ejecutan en la PC de escritorio donde también se ejecuta el servidor web y la base de datos.

Para la parte de las conexiones se debe de tener lo siguiente. En uno de los puertos USB de la PC de escritorio se encuentra conectada la tarjeta de desarrollo NodeMCU, que en este trabajo se define como nodo NodeMCU. En uno de los puertos USB de la Raspberry Pi se tiene conectado el Brick del robot Lego EV3.

4.3 Descripción y desarrollo de las pruebas

Como se mencionó anteriormente se proponen cuatro escenarios para observar, analizar y comprobar el comportamiento del robot. Así mismo, estos escenarios comprueban el desempeño de la Raspberry Pi al ejecutar ROS, controlar al robot, tener conexión remota con una PC de escritorio y tener comunicación con la tarjeta de desarrollo NodeMCU.

Es importante mencionar que antes de comenzar a realizar las pruebas necesarias se debe contar con la siguiente red LAN que comunica a todos los elementos que conforman el sistema de búsqueda, figura 68.

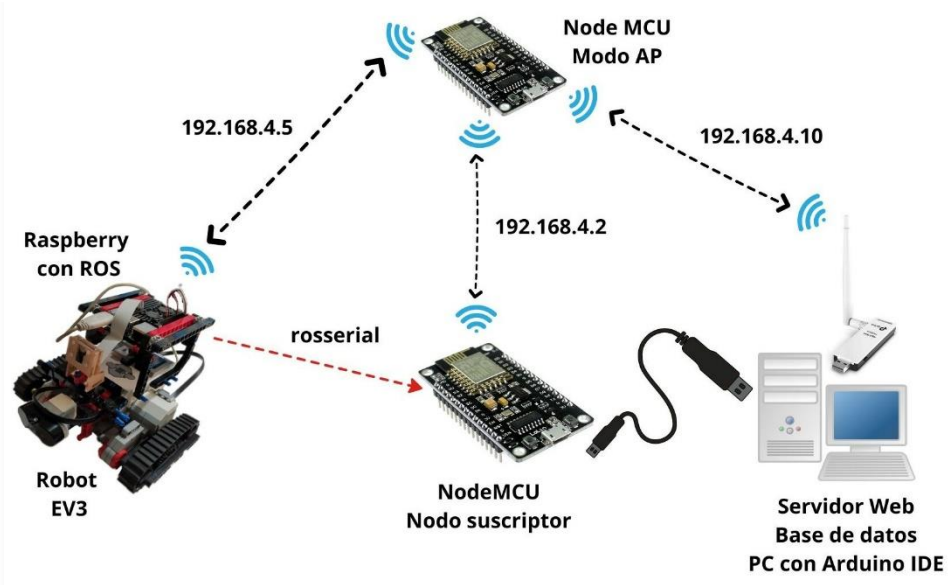


Figura 68. Esquema de red WLAN del sistema de búsqueda.

a) Primer escenario

El robot va a estar estático y solo se ejecuta el nodo Core y el nodo Cámara. Para hacer esto posible se debe modificar el archivo **.launch**, que ejecuta los nodos que componen al paquete o proyecto. Para este caso simplemente se comentan los demás nodos y solo se dejan activos los dos nodos de este escenario.

Se inicia el servidor web local y se abre HeidiSQ, Arduino IDE y VNC Viewer. Seguidamente en Arduino IDE se ejecuta la herramienta monitor serie para observar los mensajes que proporciona la tarjeta de desarrollo NodeMCU. Por otro lado, teniendo encendida la Raspberry Pi 4 y ejecutando VNC Viewer en la PC de escritorio se debe tener la visualización y control de la Raspberry Pi vía remota, figura 69.

Figura 69. Captura de pantalla de monitor serie de Arduino IDE.

La figura 69 muestra una captura de pantalla de Arduino IDE haciendo uso de la herramienta monitor serie, en donde se muestran algunos mensajes después de

conectar la tarjeta de desarrollo a la PC. Los mensajes para destacar son: el NodeMCU está conectado a una red WiFi llamada RED_NodeMCU, se le asigna una IP y, por el momento, aún no se conecta con el servidor rosserial, por eso el mensaje de No Conectado.

Teniendo encendido el Brick del robot EV3, en el escritorio de la Raspberry Pi se abre una nueva terminal, aquí se realiza la conexión SSH con el Brick del robot EV3 y después se inicia el servicio de RPyC para controlar al robot desde la Raspberry. Después se abre una nueva terminal para iniciar el nodo maestro de ROS. Enseguida se abre otra nueva terminal para iniciar el servidor rosserial y como resultado en el monitor serie de Arduino IDE se observa que el nodo NodeMCU se conecta con este servidor, por lo tanto, ya existe comunicación con ROS y con el servidor web local. Después se observa que se realiza el envío de datos sobre la de información de detección. En este caso, sin iniciar el nodo cámara, el valor de detección es 0. Todo lo anterior se observa en la figura 70.

Figura 70. Captura de pantalla de monitor serie de Arduino IDE, tras iniciar servidor rosserial.

Analizando la figura anterior se observa que una vez que se inició el servidor rosserial los mensajes que se muestran en el monitor serie son: **"Conectado"**, porque la tarjeta se encuentra conectada con el servidor rosserial; el mensaje de **"Recibo 100 si se detecta humano"** y abajo el valor de **"0"**, este último significa que aún no se identifica un rostro; el mensaje de **"Se enviaron los datos al servidor? 200 (SI) -1 (ERROR)"** y abajo mostrando el valor de **"200"** confirma que se están enviando los datos al servidor web y, por último, el mensaje que envía **"El servidor web recibió y guardo la info. en la base de datos"**, este mensaje y el siguiente son las respuestas del servidor al guardar los datos en su BD, en este caso el nombre del dispositivo que es **tarjeta1** y el valor de HUMANO que es **0**. Por otra parte, se revisa la base de datos del servidor web usando HeidiSQL, cabe mencionar que la base de datos está compuesta por dos tablas de datos, una que

muestra **el estado actual** de la variable **numROS**, que define si se detectó o no un rostro humano, y la otra tabla que muestra **el historial** de los valores que ha tenido dicha variable, figuras 71 y 72.

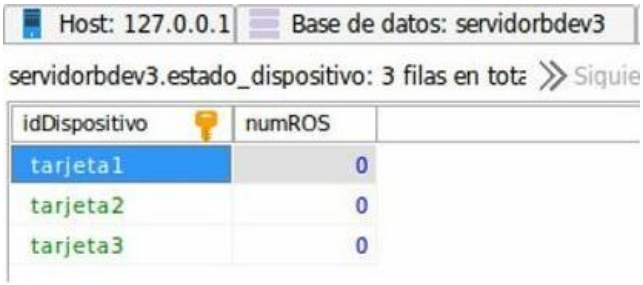


Figura 71. Captura de pantalla de HeidiSQL que muestra la tabla estado_dispositivo de BD.

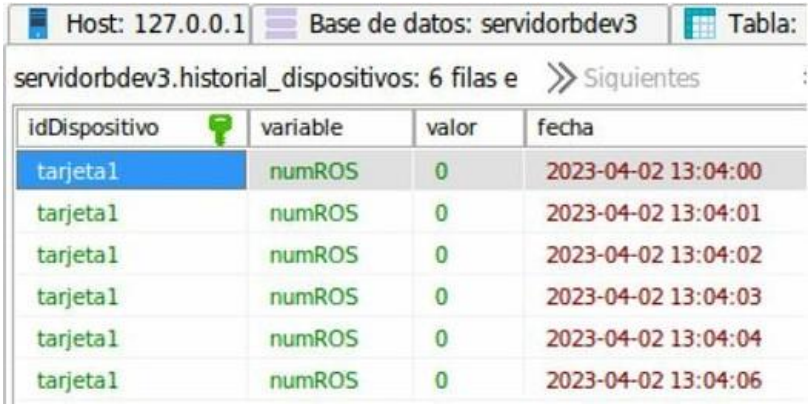


Figura 72. Captura de pantalla de HeidiSQL que muestra la tabla historial_dispositivos de BD.

En la figura 71 se muestra una captura de pantalla de HeidiSQL de la tabla estado_dispositivo. En la columna **idDispositivos** se observa que se definen **tarjeta1**, **tarjeat2** y **trajeta3**, esto es por si en un futuro se hace uso de más NodeMCU's, lo que significa que se tienen más robots para así tener un grupo de

búsqueda; en este caso solo se enfoca a la tarjeta1 por que solo se cuenta con un robot. La siguiente columna corresponde a visualizar y guardar el valor de la variable **numROS** definida en la base de datos, que por el momento se encuentra en **0**.

La figura 72 muestra la tabla historial_dispositivos de la información guardada en la base de datos. La primera columna **idDispositivo** corresponde a mostrar el nombre del dispositivo que se usa, en este caso la tarjeta1, la segunda columna **variable** se muestra el nombre de las variables que modifican su contenido, en este caso solo es la variable numROS, la siguiente columna **valor** define el contenido de la variable numROS y por último la columna **fecha** define año, mes, día, hora, minuto y segundo en que se modificó el contenido de la variable numROS de la tarjeta1 en la base de datos. En este momento el valor de numROS es 0 por que hasta el momento aún no se envía información alguna sobre la identificación de un rostro.

En la Raspberry Pi se abre otra terminal y se ejecuta el launch de proyecto. Enseguida se abre una ventana nueva en el que muestra lo que capta la cámara y en la terminal se imprimen algunos mensajes, todo esto se aprecia en las figuras 73 y 74.

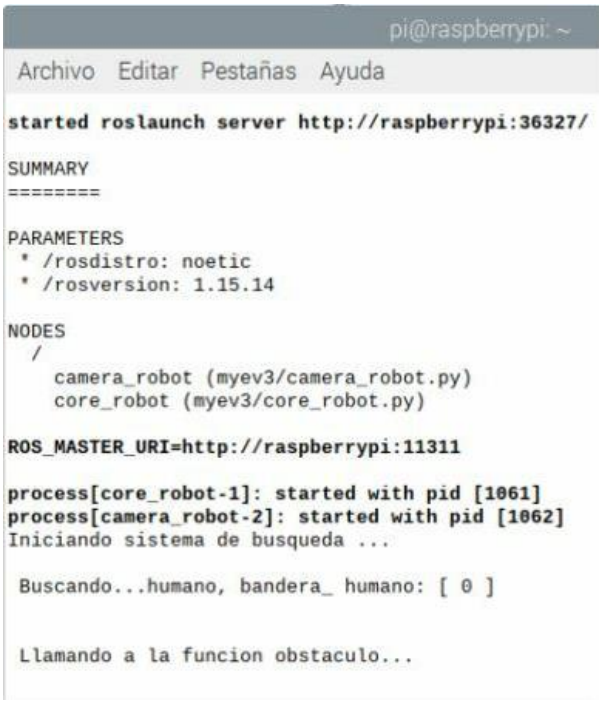


Figura 73. Captura de pantalla de terminal donde se ejecuta el proyecto myev3.

Figura 74. Imagen de la ventana Pi Camara que muestra lo que capta el módulo de cámara.

La figura 73 muestra una captura de pantalla de la terminal y se observan los mensajes de **Buscando...humano, bandera humano [0]**, que indica que hasta el

momento no se ha identificado un rostro. La siguiente línea que se imprime es el mensaje **Llamando a la función obstaculo...**, este mensaje indica el inicio de la función Obstáculo, la cual se encarga de leer los datos que envía el sensor y, de acuerdo con ese valor, se toma la decisión de avanzar, retroceder o girar el robot. Por el momento, se modificó la programación del nodo Core para que solo imprima en la terminal la función **obstacle**, ya que esta misma ocupa los nodos, sensor Infrarrojo y Motores, nodos que por el momento están deshabilitados. Por otra parte, la ventana de la cámara muestra claramente el área o espacio que se tiene frente al robot.

Ahora se coloca en la línea de vista de la cámara un bebe de juguete. El nodo Cámara identifica el rostro y lo encierra en un rectángulo con contorno verde, esto se aprecia en la figura 75 y en la figura 76 se muestra el despliegue de mensajes que confirman la detección del rostro, en este caso son **“Humano encontrado!!”** **“bandera humano [1]”** y **“Búsqueda finalizada...”**.

Figura 75. Imagen de la ventana Pi Camara donde se identifica el rostro del muñeco.

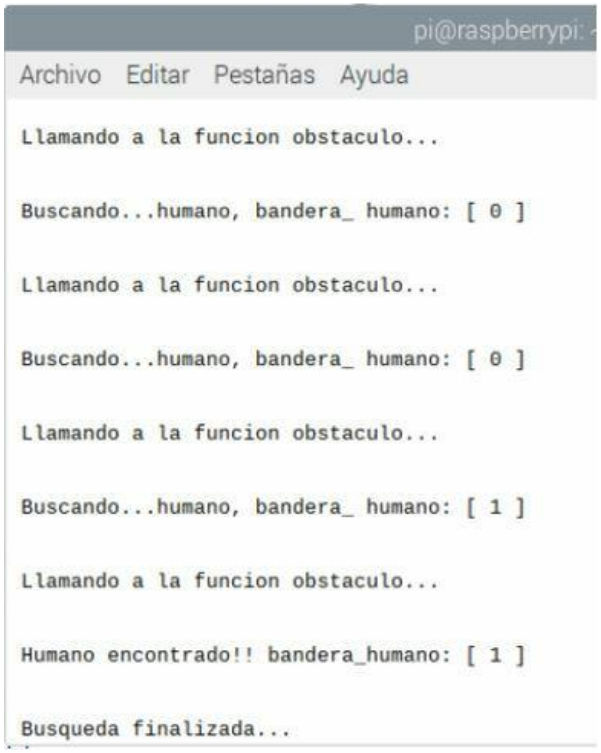


Figura 76. Terminal donde se ejecuta el paquete myev3 que muestra mensajes después de identificar un rostro.

Así mismo, cuando se identifica el rostro humano, en la bocina integrada del Brick del robot se reproduce el mensaje de “human”. Por otra parte, en el monitor serie de Arduino IDE se observan mensajes, mismos que se muestran en la figura 77.

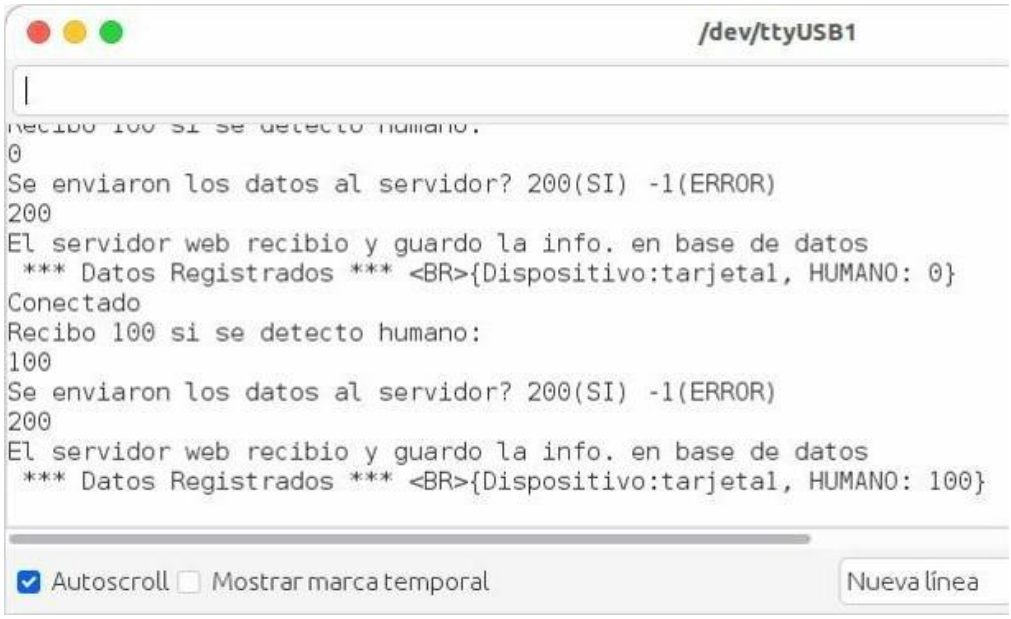


Figura 77. Captura de pantalla de monitor serie que despliega mensajes sobre la identificación de un rostro.

En la figura 77 se aprecia que después del mensaje “Recibo 100 si se detectó humano” está el valor 100, esto confirma que se identificó un rostro, este valor se envía a la base de datos del servidor web. Por último, se despliega el mensaje de los datos guardados en la base de datos, en este caso **“Dispositivo: tarjeta1, HUMANO: 100”**, este mensaje confirma que la información ha sido recibida y guardada en la base de datos del servidor web.

Por otra parte, se observa la tabla de estado_dipsositivo y la tabla de historial_dispositivos de la base datos del servidor web, figuras 78 y 79.

Figura 78. Captura de pantalla de HeidiSQL que muestra la tabla estado_dispositivo de BD tras identificarse un rostro.

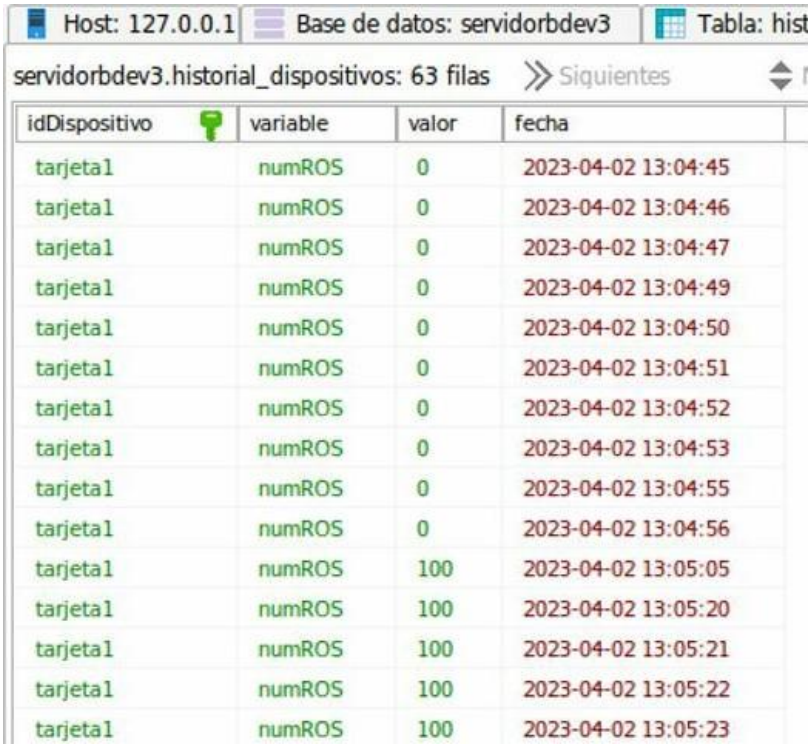


Figura 79. Captura de pantalla de HeidiSQL que muestra la tabla historial_dispositivos de BD tras identificarse un rostro.

En la figura 78 se muestra la captura de pantalla del contenido de la tabla estado_dispositivo, por lo que se observa que, en la segunda columna, corresponde

a la información de la variable numROS, se tiene un valor de 100, este número indica que se identificó un rostro humano.

La figura 79 corresponde a la tabla historial_dispositivos de la base de datos, se observa que sí llegó la información al servidor y esta se guardó en la variable numROS de la base de datos, también se observa la fecha y hora en que se hizo el cambio del valor de la variable numROS.

Este escenario nos confirma el funcionamiento del nodo Core, el nodo Cámara y el nodo NodeMCU, mismos que se apegan a los algoritmos diseñados, esto implica que el nodo Cámara identifica correctamente el rostro y avisa del suceso al nodo Core enviando un mensaje al nodo NodeMCU, para que éste envié el dato de manera inalámbrica a la base de datos de un servidor web local. Por otra parte, se hicieron pruebas para determinar la distancia máxima en la que se puede identificar un rostro, esta distancia es aproximada y se toma a partir del robot a él bebé de juguete. Se obtienen los resultados de la tabla 6.

Distancia metros (m)	Detección
D > 1.10 m	No se detecta rostro
D = 1.10 m	Detección intermitente
D = 0.9 m	Detección intermitente
D = 0.6 m	Detección estable
D < 0.4 m	Detección estable

Tabla 6. Detección de rostros por medio del módulo cámara de acuerdo con determinadas distancias.

Se concluye que la máxima distancia es de 1.10 m. Después de esa distancia, el rectángulo que encierra el rostro en la detección se torna intermitente, así como la reproducción auditiva del mensaje Human en el parlante del EV3 y los datos que

son enviados y guardados en el servidor web. Para distancias menores a 40 cm el rectángulo que encierra el rostro identificado se queda fijo y la reproducción del mensaje en el parlante es constante. Por último, se observa que a distancias mayores a los 1.10 m. no se identifica el rostro. Cabe señalar que es importante que el lugar debe estar bien iluminado para que exista un reconocimiento eficiente.

b) Segundo escenario

Para este caso el robot EV3 se mueve hacia adelante hasta que logra identificar un rostro humano. Para realizar dicha prueba, el muñeco se coloca a una distancia aproximada de 1.98 m del robot, esto es para que el EV3 avance en línea recta y así logre observar e identificar el rostro del muñeco. En este escenario se inician todos los nodos: Core, Sensor, Motors, Cámara, NodeMCU y Batería. Como primer punto, en la Raspberry Pi, con ayuda del explorador de archivos, se ubica el archivo **.launch** y se modifica su contenido para que inicien todos los nodos que componen este proyecto, ya que previamente en el primer escenario se modificó para que solo se ejecutaran dos nodos. Antes de pasar al segundo punto se coloca el robot EV3 en posición de salida y a 1.98 m de él se coloca un muñeco de juguete, figura 80.

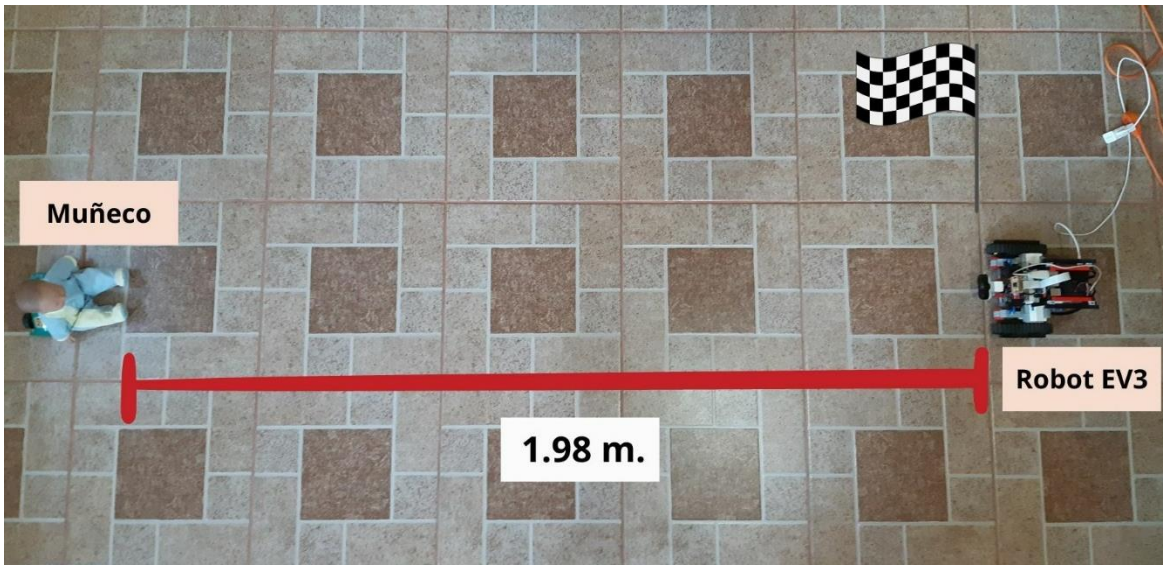


Figura 80. Escenario 2, muestra robot EV3 colocado en punto de inicio y frente a él un muñeco a una distancia.

Por otra parte, se inicia el servidor rosserial y se inicia el monitor serie de Arduino IDE.

Ahora bien, como segundo punto, se abre una terminal nueva en la Raspberry Pi y se inicia el nodo maestro de ROS, seguidamente se abre una nueva ventana para ejecutar el proyecto myev3. Tras ejecutar el proyecto, en la terminal se observan los mensajes de la figura 81 y se abre una ventana nueva, figura 82.

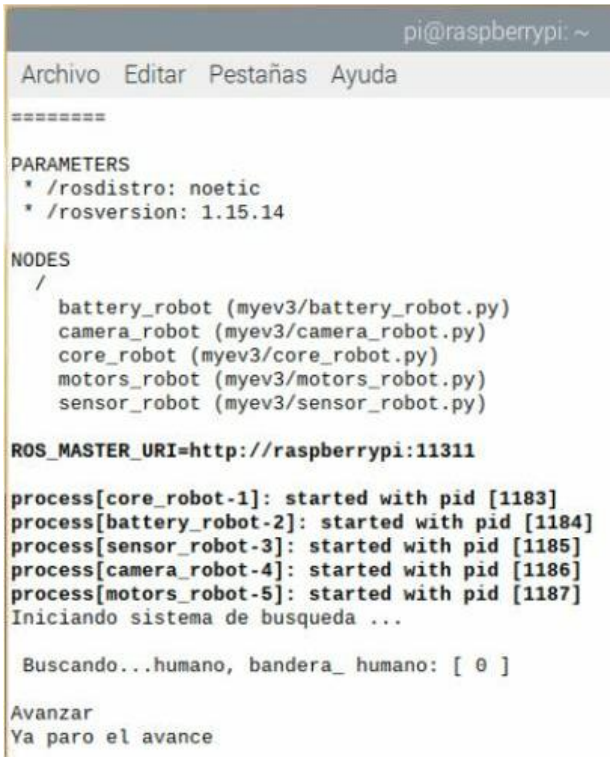


Figura 81. Captura de pantalla de la terminal tras ejecutar el paquete myev3.

Figura 82. Imagen de la ventana Pi Camera que muestra lo que capta el módulo de cámara, al fondo se observa el muñeco.

La figura 81 muestra el inicio de ejecución de los nodos, cada uno con un identificador de proceso *pid*, y seguidamente el mensaje de **“Iniciando sistemas”** y también el primer mensaje de **“Buscando...humano, bandera humano [0]”**. Después comenzará a imprimir en la terminal los mensajes del movimiento del robot, en este caso **“Avanzar”** y **“Ya paro el avance”**. Tras no haber identificado un rostro humano se comienza a publicar el primer mensaje y luego los mensajes del movimiento del robot. En la figura 82 se observa lo que capta la cámara.

Por otra parte, en la PC de escritorio se observa lo que se imprime en el monitor serie de Arduino IDE.

Figura 83. Mensajes mostrados en monitor serie tras ejecutar paquete myev3.

La figura 83 muestra lo que está recibiendo y enviando el nodo NodeMCU al servidor web local. En este caso se envió un valor de 0, que significa que no se ha identificado un rostro, mismo que el servidor ya recibió y guardó en la base de datos.

Si se revisa el registro de la base de datos del servidor web local se observa que se recibió y guardó el valor de 0 en la base de datos. El robot EV3 sigue avanzando en línea recta y todo el registro del avance se imprime, así como también el mensaje de **"Buscando...humano, bandera humano [0]"**, figura 84. Estos mensajes se muestran en la terminal de la Raspberry donde se ejecutó el proyecto.

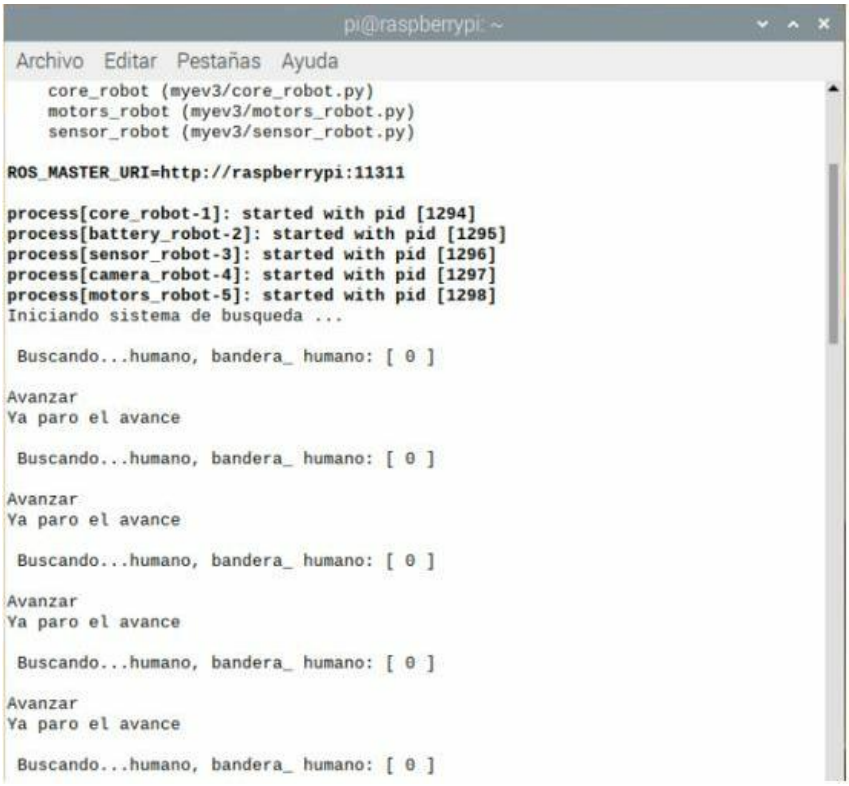


Figura 84. Terminal que muestra mensajes del robot tras avanzar.

Cuando el robot avanzó una distancia aproximada de 1.17 m medidos desde el punto de partida, el nodo Cámara comienza a identificar un rostro de manera intermitente, poco después lo identifica de manera constante a una distancia de 1.36 m desde el punto de inicio del robot, lo anterior se puede observar en el historial

de la base de datos. Lo primero que sucede es que el parlante del robot emite la palabra **human**, seguidamente en la terminal donde se muestran los mensajes del movimiento del robot en la Raspberry se puede ver el nuevo mensaje de "**Humano encontrado!! , bandera humano [1]"** y el mensaje de **"Búsqueda finalizada..."**, esto se aprecia en la figura 85. Por otro lado, la ventana que muestra lo que capta la cámara se observa en la figura 86.

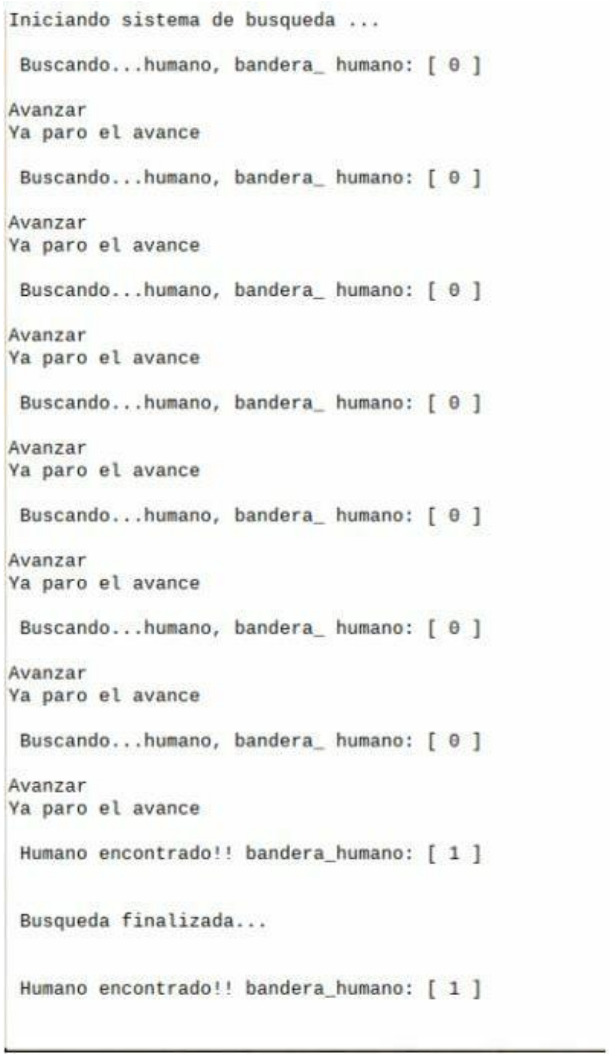


Figura 85. Terminal que muestra los mensajes en el avance del robot y los mensajes tras reconocer un rostro.

Figura 86. Imagen de la ventana Pi Cámara que muestra el muñeco e idéntica su rostro.

En la figura 85 se observa que después de varios mensajes que indican el movimiento del robot se identifica el rostro del muñeco y se envía el mensaje de **“Humano encontrado!! , bandera humano [1]”** y **“Búsqueda finalizada...”** Por otro lado, la figura 86 muestra la imagen que la cámara está capturando, en este caso se observa el muñeco y como su rostro se encuentra encerrado en un recuadrado de contorno verde.

En la base de datos, haciendo usó de HeidiSQL, lo que se observa en las tablas estado_dispositivo y historial_dispositivos se muestra en las figuras 87 y 88.

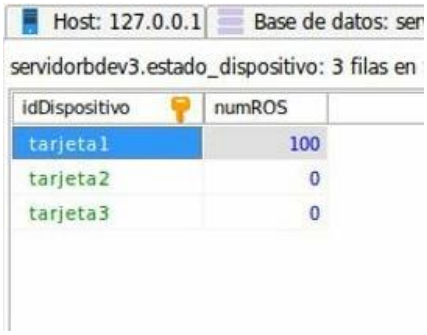


Figura 87. Con HeidiSQL se muestra la tabla estado_dispositivo de BD tras identificarse un rostro.

Figura 88. Con HeidiSQL se muestra la tabla historial_dispositivos de BD tras identificarse un rostro.

La figura 87 muestra el valor actual, que es 100, de la variable numROS para la tarjeta1, esto indica que se identificó un rostro y se guardó en la base de datos del servidor. Por otro lado, para el caso de la figura 88 se muestra el historial de registro de la variable numROS, las flechas rojas indican como se identifica el rostro de manera intermitente y en la flecha azul, ya se identifica el rostro de manera constante.

Con todo lo anterior se observa que la programación y todas las configuraciones son correctas para que el sistema funcione de acuerdo con los objetivos que se buscan. El robot avanza e identifica el rostro humano y envía una señal en forma de dato a un servidor web local para que se confirme que se encontró un humano, esta información se guarda en la base de datos y a su vez se lleva un registro de esto. Así mismo, en este escenario se observa que después de siete movimientos del robot se identifica el rostro del muñeco, esto se pudo ver en la figura 85. Por otro lado, se observa que la intermitencia del reconocimiento se ve reflejado en el historial de la variable numROS, figura 88, ya que en un momento tiene un valor de 100, luego 0, después otra vez 100, nuevamente 0 y al final el valor de 100 se vuelve constante cuando se tiene una distancia de aproximadamente 0.6 m del robot al muñeco, esto se corrobora con la tabla 6 que se obtuvo del primer escenario.

c) Tercer escenario

En este escenario se coloca un obstáculo en la ruta de avance lineal que realiza el robot. El algoritmo de evasión de obstáculo consta de dos movimientos[33], moverse de reversa y después realizar un giro de 60 grados. Para el caso del giro se recuerda que los giros los hace de forma aleatoria, por lo que el robot puede dar un giro a su lado izquierdo o al derecho. Teniendo en cuenta los giros aleatorios del robot, para este escenario en especial se hace uso de dos bebés de juguete. El escenario consta de los siguientes elementos: un obstáculo, que es una caja de cartón, se coloca a una distancia de 0.77 m con respecto al robot y dos muñecos en los extremos derecho e izquierdo a una distancia de 1.95 m y en un ángulo de 60 grados respecto del robot. En la figura 89 se muestra el escenario.

[33] En el presente trabajo el algoritmo de evasión de obstáculos no es uno de los objetivos de este proyecto, razón por la cual este algoritmo es un área de oportunidad para mejorar y explorar para trabajos futuros.

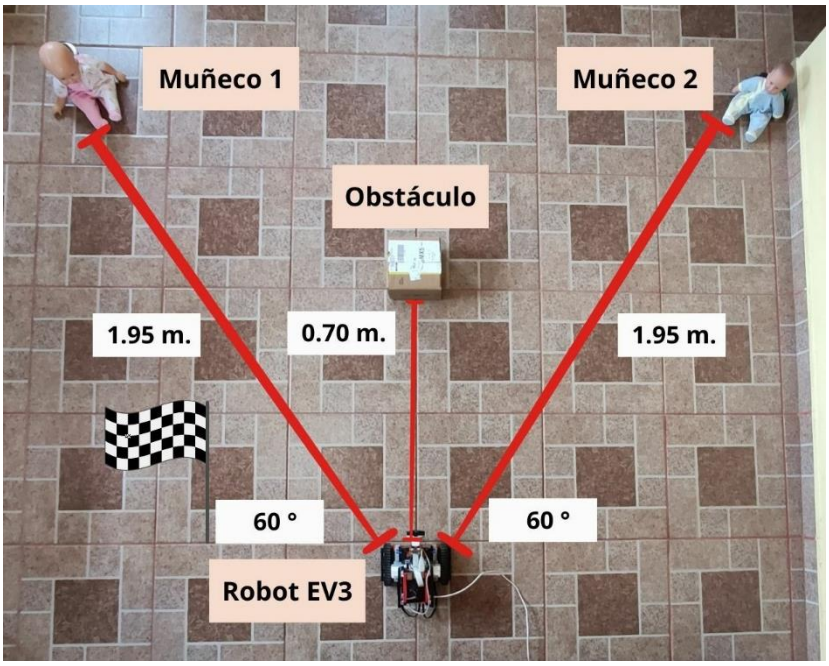


Figura 89. Escenario 3, muestra el robot EV3 en punto de inicio, el obstáculo y los muñecos.

Una vez colocados todos los elementos en su lugar, se procede a iniciar todos los elementos que componen este sistema de búsqueda. Se ejecuta en la Raspberry el paquete myev3 para que comience la búsqueda, en las figuras 90 y 91 se observa lo que se muestra en la terminal y en la cámara.

```
pi@raspberrypi: ~
Archivo  Editar  Pestañas  Ayuda
NODES
  /
    battery_robot (myev3/battery_robot.py)
    camera_robot (myev3/camera_robot.py)
    core_robot (myev3/core_robot.py)
    motors_robot (myev3/motors_robot.py)
    sensor_robot (myev3/sensor_robot.py)

ROS_MASTER_URI=http://raspberrypi:11311

process[core_robot-1]: started with pid [2581]
process[battery_robot-2]: started with pid [2582]
process[sensor_robot-3]: started with pid [2583]
process[camera_robot-4]: started with pid [2584]
process[motors_robot-5]: started with pid [2585]
Iniciando sistema de busqueda ...

 Buscando...humano, bandera_ humano: [ 0 ]

Avanzar
Ya paro el avance

 Buscando...humano, bandera_ humano: [ 0 ]
```

Figura 90. Terminal que muestra mensajes tras ejecutar el paquete myev3.

Figura 91. Imagen de la ventana Pi Cámara que muestra lo que capta la cámara.

En la figura 90 se muestran los resultados obtenidos en la terminal tras ejecutar el paquete myev3, se observa que nuevamente se inician los nodos y se muestran los mensajes que indican que aún no se detecta a un humano y los mensajes de movimiento del robot. Por otro lado, la figura 91 muestra que frente al robot se encuentra el obstáculo, que es una caja de cartón. Por el momento, no se detecta este obstáculo ya que se coloca a una distancia mayor al rango en que el sensor comienza a detectar un obstáculo. El robot sigue avanzando en línea recta hasta que el obstáculo es detectado por el sensor, las siguientes figuras ilustran la terminal y la ventana que corresponde a lo que capta la cámara.

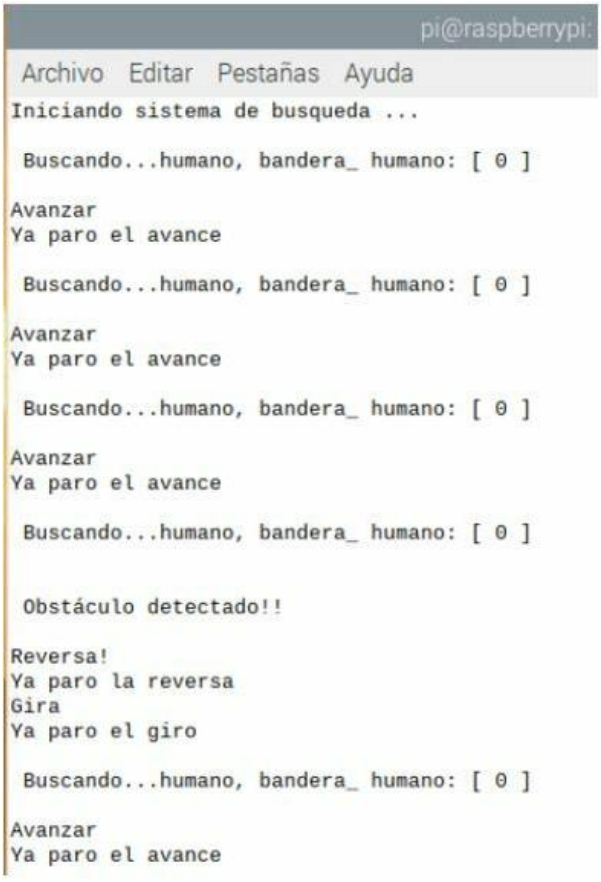


Figura 92. Terminal que muestra mensajes tras detectar un obstáculo frente al robot.

Figura 93. Imagen de la ventana Pi Cámara que muestra el obstáculo muy cerca.

En la figura 92 se observa en la terminal el mensaje **“Obstáculo detectado!”**, este indica que el sensor detectó un objeto; seguidamente el mensaje **“Reversa!”**, indica que el robot se mueve de reversa; después **“Ya paró reversa”**, significa que ya terminó de moverse el robot de reversa; **“Gira”**, avisa que el robot está realizando el giro y por último **“Ya paro giro”**, indica que el robot dejó de girar. Después de esos mensajes se ve el mensaje que indica que aún no se detecta a un humano y se sigue con la búsqueda. Después, nuevamente se imprimen los mensajes de avance del robot, esto significa que se evadió el obstáculo. Mientras tanto, en la figura 93 se observa que cuando se identifica al obstáculo, la cámara capta la caja de cartón muy cerca que hasta se puede apreciar el texto del nombre de una tienda

en línea. También, tras identificar el obstáculo, el robot reproduce en su parlante integrado el mensaje de **“Obstacle!”**.

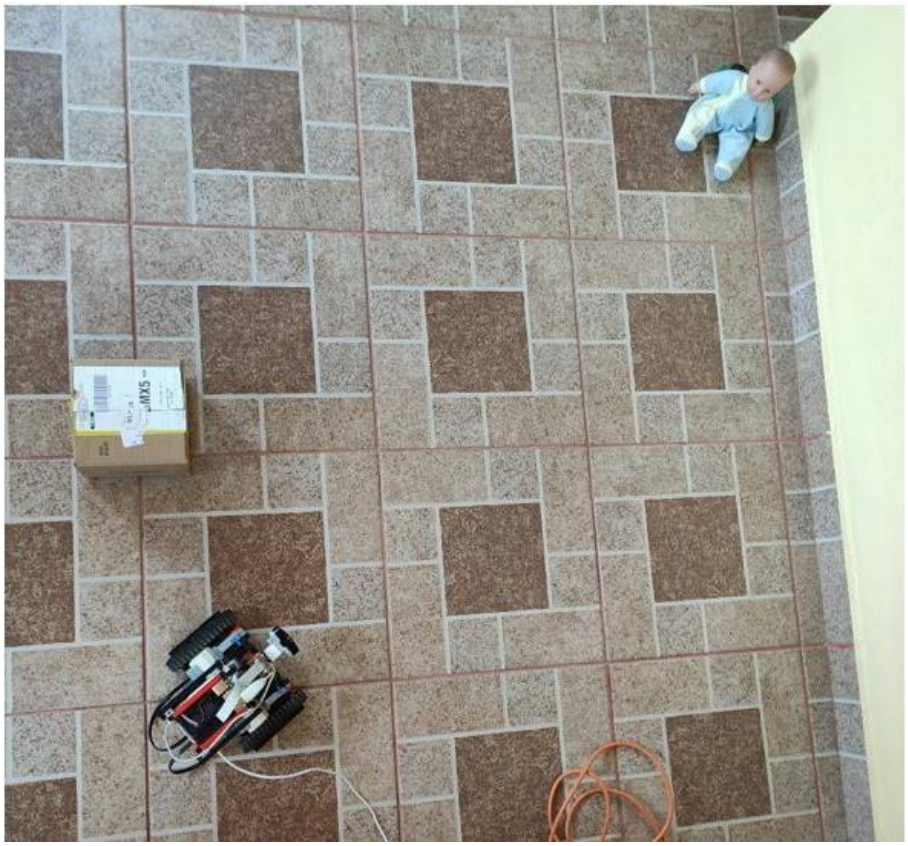


Figura 94. Robot evadiendo el obstáculo.

En la figura 94 se muestra al robot que evade la caja de cartón y sigue con su avance lineal. También se observa que, en esta ocasión, el giro aleatorio fue hacia el lado derecho del robot. El robot sigue con su avance hasta que a unos 0.99 m se comienza a realizar de manera intermitente el reconocimiento, sigue avanzando y a una distancia de 0.8 m se hace la detección de rostro constante, por lo que se obtienen los resultados de las figuras 95 y 96.

```
 Obstáculo detectado!!

Reversa!
Ya paro la reversa
Gira
Ya paro el giro

 Buscando...humano, bandera_ humano: [ 0 ]

Avanzar
Ya paro el avance

 Buscando...humano, bandera_ humano: [ 0 ]

Avanzar
Ya paro el avance

 Buscando...humano, bandera_ humano: [ 0 ]

Avanzar
Ya paro el avance

 Buscando...humano, bandera_ humano: [ 0 ]

Avanzar
Ya paro el avance

 Humano encontrado!! bandera_humano: [ 1 ]


 Busqueda finalizada...


 Humano encontrado!! bandera_humano: [ 1 ]


 Busqueda finalizada...
```

Figura 95. Mensajes en terminal tras identificar un rostro.

Figura 96. Imagen de la ventana Pi Cámara que muestra muñeco y la identificación de su rostro.

Nuevamente, en terminal se muestran los mensajes para indicar que se detectó el rostro y por lo tanto se finaliza la búsqueda, figura 95. Por otro lado, en la ventana de la cámara se observa el rostro del bebé de juguete enmarcado en un recuadro con contorno verde, figura 96.

Se revisa la base de datos y se obtienen los resultados mostrados en la figura 97 y 98.

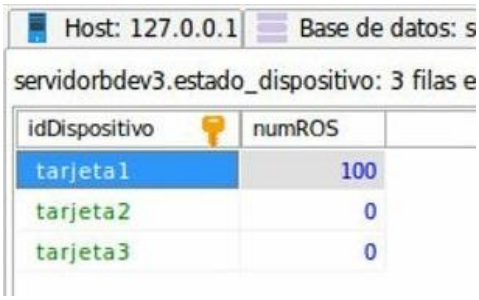
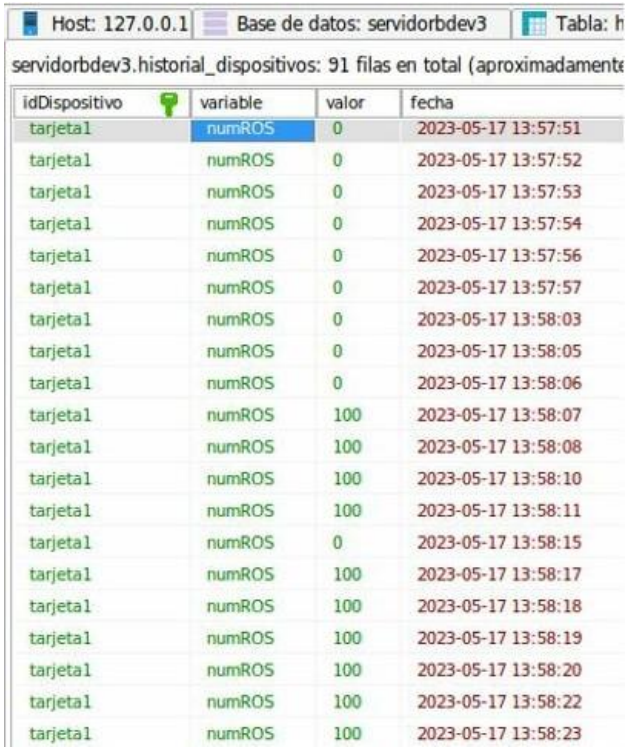


Figura 97. Tabla estado_dispositivo de BD tras identificarse un rostro.



idDispositivo	variable	valor	fecha
tarjeta1	numROS	0	2023-05-17 13:57:51
tarjeta1	numROS	0	2023-05-17 13:57:52
tarjeta1	numROS	0	2023-05-17 13:57:53
tarjeta1	numROS	0	2023-05-17 13:57:54
tarjeta1	numROS	0	2023-05-17 13:57:56
tarjeta1	numROS	0	2023-05-17 13:57:57
tarjeta1	numROS	0	2023-05-17 13:58:03
tarjeta1	numROS	0	2023-05-17 13:58:05
tarjeta1	numROS	0	2023-05-17 13:58:06
tarjeta1	numROS	100	2023-05-17 13:58:07
tarjeta1	numROS	100	2023-05-17 13:58:08
tarjeta1	numROS	100	2023-05-17 13:58:10
tarjeta1	numROS	100	2023-05-17 13:58:11
tarjeta1	numROS	0	2023-05-17 13:58:15
tarjeta1	numROS	100	2023-05-17 13:58:17
tarjeta1	numROS	100	2023-05-17 13:58:18
tarjeta1	numROS	100	2023-05-17 13:58:19
tarjeta1	numROS	100	2023-05-17 13:58:20
tarjeta1	numROS	100	2023-05-17 13:58:22
tarjeta1	numROS	100	2023-05-17 13:58:23

Figura 98. Tabla historial_dispositivos de BD tras identificarse un rostro.

El estado actual de la variable numROS tiene un valor de 100, indicando que se encontró un humano, figura 97. En la figura 98, el historial se observa el valor de numROS con cambios de 0 a 100, así mismo se observa la intermitencia de la detección, es por ello que unas veces numROS es 100 luego 0 y después ya se queda con 100, indicando que la detección es constante.

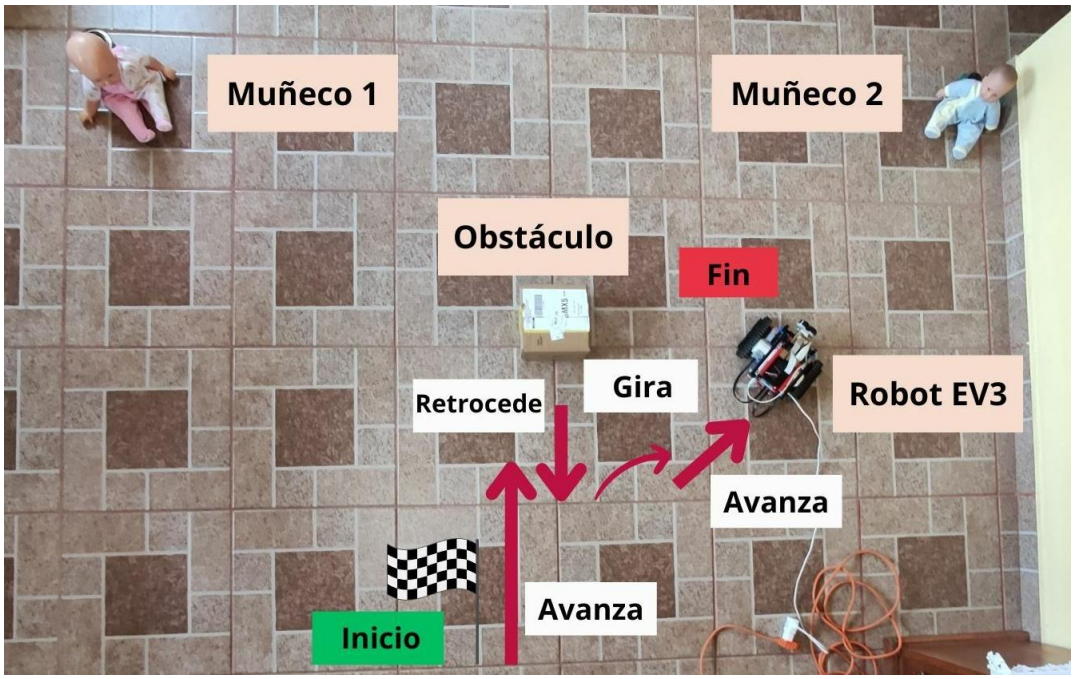


Figura 99. Escenario 3 tras finalizar la búsqueda.

Con todo lo anterior el algoritmo sigue los objetivos de acuerdo con lo programado, ya que el robot avanza, identifica un obstáculo, evade el obstáculo retrocediendo y girando ya sea a la izquierda o a la derecha de manera aleatoria, sigue avanzando, identifica el rostro humano y envía una señal en forma de dato a un servidor web local para que se confirme que se encontró un rostro humano; esta información se guarda en una base de datos y a su vez se lleva un registro de esto. Este escenario ayuda a verificar el funcionamiento de todos los nodos que compone este proyecto de búsqueda, así como todos los sistemas de comunicación que se involucran.

Conclusiones

En los siguientes párrafos se escriben las conclusiones obtenidas en la elaboración y cierre de este trabajo, en el que se retoma y mejora un sistema de búsqueda usando un robot autónomo y software libre. Así mismo, se escriben las conclusiones personales respecto a la formación recibida en la licenciatura de Ingeniería en Sistemas Electrónicos y de Telecomunicaciones de la UACM. Por último, se mencionan las propuestas de mejoras futuras para este proyecto robótico de búsqueda.

I. Conclusiones generales

Con la elaboración de este trabajo se cumplieron los objetivos planteados en la implementación de un sistema de búsqueda de personas, que se apoya de un robot autónomo de bajo costo. El sistema tiene la capacidad de detectar un rostro humano en tres escenarios diferentes: a) de manera estática al colocarle enfrente a un muñeco de juguete, b) de manera dinámica cuando el robot se mueve en línea recta hacia el muñeco y c) al colocarle un objeto como obstáculo para que realice la tarea de evadirlo y buscar el rostro humano en la nueva dirección.

Con base en el trabajo [1], se visualiza el robot con el software RVIZ que sólo es una herramienta de visualización de ROS. En este trabajo se modela el robot con el software libre Gazebo, que incorpora un motor de físicas. Por otra parte, se mejora el sistema agregando módulos de comunicación y la creación de un servidor web en el que se almacena el registro de la detección de rostros humanos; este sistema se comunica por medio de una WLAN. El robot es un Lego EV3 Mindstorm que lleva a bordo un sistema de procesamiento empleando una tarjeta Raspberry Pi 4, que hace uso de ROS para realizar la ejecución de algoritmos que controlan al robot. Se usan son los módulos de comunicación ESP8266 que son compatibles con las librerías de ROS, para este caso se hace uso de las tarjetas de desarrollo NodeMCU. Así entonces, todos los elementos que componen este sistema de búsqueda se configuraron con el fin de cumplir con los objetivos mencionados en este trabajo. Aunado al desarrollo de la implementación se generó una guía que explica de manera detallada los elementos necesarios para la realización de

proyectos robóticos de bajo costo haciendo uso de ROS, Raspberry Pi, NodeMCU, el robot Lego Mindstorms EV3, entre otros elementos. La guía ayuda a la compresión sobre el tema para aquellos usuarios interesados y quienes apenas inician en proyectos robóticos como este. Siendo ésta una de las principales aportaciones del proyecto.

II. Conclusiones personales

En el desarrollo de este proyecto se aplican conocimientos que se obtuvieron en los cursos que se imparten en la UACM para la licenciatura de Ingeniería en Sistemas Electrónicos y de Telecomunicaciones. De acuerdo con los planes de estudio, esta licenciatura se divide en ciclo básico y ciclo superior. Así entonces, del ciclo básico se retoman conocimientos de mecánica, algebra lineal, cálculo vectorial, programación e inglés, este último se debe a que la mayoría de la información requerida se encuentra en idioma inglés. Por otra parte, del ciclo superior se hace uso de habilidades desarrolladas para el manejo de SO basados en Linux y su amplio listado de comandos para trabajar desde una terminal. Se retoman temas sobre tipos de redes de computadoras, estándares de IEEE y todo lo relacionado con las redes. También se usan conocimientos sobre servidores web, así como de base de datos. De manera general, la formación académica obtenida en la UACM se usa para encontrar, proponer y desarrollar diversas soluciones para resolver los inconvenientes encontrados durante la elaboración de este trabajo.

Por otro lado, se aprendió que en la actualidad existe demasiada información disponible en Internet y se requiere la habilidad para optar por fuentes fiables. En el caso del uso de software y hardware se debe de investigar a detalle sobre la compatibilidad y funcionamiento entre éstos dado que ambos elementos tecnológicos, con el paso de los años, generan nuevas versiones que no son necesariamente compatibles. Todo lo anterior ha enriquecido mi formación académica para: investigar, desarrollar y resolver problemas, proponer, documentar y exponer soluciones, además de organizar y comprender distintas temáticas de sistemas y telecomunicaciones, aunado al desarrollo de más habilidades que se obtuvieron en mi estancia en la UACM, para así enfrentarme a los retos y necesidades que se encuentran actualmente en el ámbito de la tecnología.

III. Trabajo Futuro

Como se mencionó anteriormente, en el presente documento se cumplen los objetivos planteados. Mas sin en cambio en el transcurso del desarrollo del proyecto se encontraron nuevas oportunidades de mejora de este sistema de búsqueda. Estas mejoras están enfocadas para aquellos usuarios que estén interesados en continuar con este proyecto. A continuación, se describen dichas áreas.

En primera instancia, se requiere agregar los nodos que en el trabajo [1] se definieron para que el robot, una vez que identificó un rostro humano, regrese al punto de inicio, a la vez de que se realiza la simulación en tiempo real del robot EV3. Para que se realice la simulación en tiempo real sin ninguna dificultad se propone usar una Raspberry Pi 4 de mayor capacidad de memoria RAM.

Como segundo punto se propone la adición de un módulo de alimentación para la unidad de procesamiento, en este caso para la Raspberry. Resolviendo este punto se obtendría el 100 % de la autonomía del robot EV3.

Como tercer punto, se requiere configurar de mejor manera el algoritmo de detección de rostros Haar Cascades, ya que la configuración actual del algoritmo tiende a demorar en la detección de un rostro si no existe una buena iluminación de éste, en este caso los rostros de los muñecos. Así mismo, para una mejor detección se sugiere usar un módulo de cámara de mayor resolución.

Y como último punto para un mejor desempeño en cuanto a la evasión de obstáculos, se necesita agregar un sensor ultrasónico, ya que el sensor IR se ve afectado por la iluminación.

Este trabajo ambiciona que en un futuro el robot pueda moverse en un escenario con muchos obstáculos y que, de acuerdo con sus recorridos, realice un mapeo del lugar y que a su vez el robot determine por qué camino ir e identifique por dónde ya pasó. Así mismo se propone que se tome una foto y se envié a la base de datos al detectar un rostro humano, además de enviar la ubicación del humano encontrado. Para ello se propone agregar un sistema de localización. Así mismo se ambiciona agregar una cámara con grados de libertad, esto es con el fin de tener una mejor visibilidad del entorno y así mejorar búsqueda de un rostro.

Referencias

[1] A. Cortés Murillo, «Robot autónomo con comunicación inalámbrica de búsqueda de personas implementado en ROS», Thesis, Universidad Autónoma de la Ciudad de México : Colegio de Ciencia y Tecnología : Licenciatura en Ingeniería en Sistemas Electrónicos y de Telecomunicaciones, 2019. Accedido: 13 de mayo de 2023. [En línea]. Disponible en: http://repositorioinstitucionaluacm.mx/jspui/handle/123456789/1529

[2] E. B. School, «Clasificación de los robots | Euroinnova», Euroinnova Business School. Accedido: 2 de abril de 2023. [En línea]. Disponible en: https://www.euroinnova.mx/blog/clasificacion-de-los-robots

[3] S. E. Martínez Rozas, «Modelado y simulación de robots terrestres para la inspección del alcantarillado», Universidad de Sevilla, Sevilla, 2018. Accedido: 26 de junio de 2023. [En línea]. Disponible en: https://idus.us.es/handle/11441/81107

[4] «Distributions - ROS Wiki». Accedido: 3 de abril de 2023. [En línea]. Disponible en: http://wiki.ros.org/Distributions

[5] O. Ramos, «Fundamentos de Robotica 2019-1», presentado en Introducción a ROS, Lima, Perú, 2019. [En línea]. Disponible en: https://profesores.utec.edu.pe/oramos/teaching/19-1/robotica/tmp/lab1.pdf

[6] «Explicando BSD», Portal de documentación de FreeBSD. Accedido: 25 de abril de 2023. [En línea]. Disponible en: https://docs.freebsd.org/es/articles/explaining-bsd/

[7] J. C. Martínez Romo, M. Romo, F. Javier, L. Rosas, M. Hernández, y V. Mora Romo, «Virtual Simulation Environment for Mobile Robots with ROS», presentado en CONiiN, may 2019.

[8] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, y S. Vicente-Diaz, «Robotics software frameworks for multi-agent robotic systems development», *Robotics and Autonomous Systems*, vol. 60, n.º 6, pp. 803-821, jun. 2012, doi: 10.1016/j.robot.2012.02.004.

[9] J. Lentin, *Mastering ROS for robotics programming: design, build, and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities*. en Community experience distilled. Birmingham Mumbai: Packt Publishing, 2015.

[10] «Gazebo». Accedido: 22 de octubre de 2022. [En línea]. Disponible en: https://classic.gazebosim.org/#status

[11] «Gazebo : Tutorial : ROS overview». Accedido: 28 de septiembre de 2022. [En línea]. Disponible en: https://classic.gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros

[12] V. Mazzari, «Robotic simulation scenarios with Gazebo and ROS», Génération Robots - Blog. Accedido: 4 de abril de 2023. [En línea]. Disponible en: https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/

[13] «List of moments of inertia», *Wikipedia*. 15 de septiembre de 2022. Accedido: 18 de octubre de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=List_of_moments_of_inertia&oldid=1110514563#List_of_3D_inertia_tensors

[14] R. Merino López, «Creación de modelo URDF del robot manfred», PROYECTO FIN DE CARRERA INGENIERÍA TÉCNICA INDUSTRIAL: ELECTRÓNICA INDUSTRIAL, Universidad Carlos III de Madrid, 2014. Accedido: 30 de septiembre de 2022. [En línea]. Disponible en: https://e-archivo.uc3m.es/handle/10016/22550

[15] «Gazebo : Tutorial : URDF in Gazebo». Accedido: 4 de abril de 2023. [En línea]. Disponible en: http://classic.gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros
[16] «¿Qué son las telecomunicaciones?», Trends and Innovation. Accedido: 7 de junio de 2023. [En línea]. Disponible en: https://www.galileo.edu/trends-innovation/que-son-las-telecomunicaciones/
[17] A. S. Tanenbaum y D. J. Wetherall, *Redes de computadoras*, 5a ed. México: Pearson Educación, 2012.
[18] W. Stallings, R. V. Ramírez Velarde, y J. López Barrientos, *Comunicaciones y redes de computadores*, 7ª ed., Reimp. Madrid [etc]: Pearson Prentice Hall, 2010.
[19] J. M. Barceló Ordinas, J. Íñigo Griera, R. Martí Escalé, E. Peig Olivé, y X. Perramon Tornil, *Redes de computadores*, 1ra Edición. Barcelona: Eureca Media, SL, 2004.
[20] «¿Qué es una red de área extendida (WAN)? - Explicación sobre las redes de área extendida - AWS», Amazon Web Services, Inc. Accedido: 3 de abril de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/wan/
[21] M. C. Liberatori, *Redes de datos y sus protocolos*, 1ra edicion. Mar del Plata, Argentina: Editorial de la Universidad Nacional de Mar del Plata EUDEM, 2018.
[22] «¿Qué tipos de servidores hay?», Claranet. Accedido: 3 de abril de 2023. [En línea]. Disponible en: https://www.claranet.es/blog/que-tipos-de-servidores-hay
[23] «rosserial - ROS Wiki». Accedido: 29 de noviembre de 2022. [En línea]. Disponible en: http://wiki.ros.org/rosserial
[24] Y. Pyo, H. Cho, R. Jung, y T. Lim, *ROS Robot Programming*, First Edition. Seoul, Republic of Korea: ROBOTIS Co.,Ltd, 2017.
[25] L. Santos, T. Costa, J. M. L. P. Caldeira, y V. N. G. J. Soares, «Performance Assessment of ESP8266 Wireless Mesh Networks», *Information*, vol. 13, n.º 5, Art. n.º 5, may 2022, doi: 10.3390/info13050210.
[26] «RPyC - Transparent, Symmetric Distributed Computing — RPyC». Accedido: 10 de enero de 2023. [En línea]. Disponible en: https://rpyc.readthedocs.io/en/latest/
[27] «RPyC on ev3dev — python-ev3dev 2.1.0.post1 documentation». Accedido: 6 de marzo de 2023. [En línea]. Disponible en: https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/rpyc.html
[28] «About», OpenCV. Accedido: 4 de noviembre de 2022. [En línea]. Disponible en: https://opencv.org/about/
[29] Legoeducation.com, «Guia de usuario Mindstorm EV3». The Lego Group, 2016. Accedido: 25 de octubre de 2022. [En línea]. Disponible en: https://le-www-live-s.legocdn.com/sc/media/files/user-guides/ev3/ev3_user_guide_esmx-6ac740d3cdd578cc6a52d10d7d173da9.pdf
[30] «Getting Started with ev3dev». Accedido: 4 de abril de 2023. [En línea]. Disponible en: https://www.ev3dev.org/docs/getting-started/
[31] M. A.- melqui sierra@gmail com Sierra Arcos, «Simulación de un algoritmo de control de colonias de robots móviles ruteadores para ambientes de exploración utilizando Ros y Gazebo», Universidad del Bio-Bio, Chile, 2019. Accedido: 8 de junio de 2023. [En línea]. Disponible en: http://repobib.ubiobio.cl/jspui/handle/123456789/3473
[32] P. Mascaró Pons, «Modelado virtual y simulación de vehículos autónomos en Gazebo», bachelorThesis, 2018. Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://e-archivo.uc3m.es/handle/10016/29078
[33] «urdf/XML/link - ROS Wiki». Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://wiki.ros.org/urdf/XML/link

[34] «urdf/XML/joint - ROS Wiki». Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://wiki.ros.org/urdf/XML/joint
[35] R. P. Ltd, «Buy a Raspberry Pi 4 Model B», Raspberry Pi. Accedido: 4 de noviembre de 2022. [En línea]. Disponible en: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[36] «rosserial_xbee/Tutorials/Example Network - ROS Wiki». Accedido: 25 de octubre de 2022. [En línea]. Disponible en: http://wiki.ros.org/rosserial_xbee/Tutorials/Example%20Network
[37] «Cómo generar una red WiFi con el ESP8266 o ESP32 (modo AP)», Luis Llamas. Accedido: 10 de enero de 2023. [En línea]. Disponible en: https://www.luisllamas.es/como-generar-una-red-wifi-con-el-esp8266-modo-ap/
[38] «Cámara Raspberry Pi Rev 1.3 5MP», UNIT Electronics. Accedido: 10 de enero de 2023. [En línea]. Disponible en: https://uelectronics.com/producto/camara-raspberry-pi-rev-1-3-5mp/
[39] «Projects | Computer coding for kids and teens | Raspberry Pi». Accedido: 10 de enero de 2023. [En línea]. Disponible en: https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/0
[40] «Descargar e instalar XAMPP para Linux (Ubuntu) [2021]». Accedido: 4 de abril de 2023. [En línea]. Disponible en: https://todoxampp.com/descargar-e-instalar-xampp-para-linux-ubuntu/
[41] P. Viola y M. Jones, «Rapid object detection using a boosted cascade of simple features», en *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, dic. 2001, p. I-I. doi: 10.1109/CVPR.2001.990517.
[42] «HeidiSQL - MariaDB, MySQL, MSSQL, PostgreSQL and SQLite made easy». Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://www.heidisql.com/
[43] «Software de Arduino | Arduino.cl - Compra tu Arduino en Línea». Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://arduino.cl/programacion/
[44] «Activar un cliente VNC para acceder a la máquina virtual remotamente». Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://docs.vmware.com/es/VMware-Fusion/13/com.vmware.fusion.using.doc/GUID-97A2E489-4390-4B9B-BC2A-E97A5CD5F90E.html
[45] «¿Qué es VNC y para qué sirve? - Definición», GEEKNETIC. Accedido: 8 de junio de 2023. [En línea]. Disponible en: https://www.geeknetic.es/VNC/que-es-y-para-que-sirve