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

“Robot autónomo con comunicación inalámbrica de búsqueda de personas implementado en ROS”

TRABAJO RECEPCIONAL
QUE PARA OBTENER EL TÍTULO DE LICENCIADO EN
INGENIERÍA EN SISTEMAS ELECTRÓNICOS
Y DE TELECOMUNICACIONES

PRESENTA

ADOLFO CORTÉS MURILLO

DIRECTOR

M. en C. Magali Cortez Vázquez

Ciudad de México, febrero de 2019

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

En este proyecto se describe el desarrollo para realizar una aplicación ejecutada por un robot para explorar un área, evadir obstáculos, encontrar personas y regresar al punto de partida de manera autónoma. Se cuenta con un robot Lego, una cámara GoPro, un router y una computadora.

La aplicación es programada en Python e integra un algoritmo de reconocimiento facial de la biblioteca de código abierto OpenCV y el sistema operativo EV3DEV con base en Debian Linux. La aplicación se ejecuta en ROS (Robot Operating System) y proporciona además una interface visual para ver en tres dimensiones el funcionamiento de la aplicación. La principal característica de este proyecto es que se desarrolla de forma modular e integra hardware y software de distintas marcas realizando la comunicación entre los equipos a través de redes inalámbricas.

Palabras clave: ROS, Python, Robot Lego, 802.11, OpenCV.

Dedicatoria

A mi familia y amigos.

Agradecimientos

Agradezco profundamente a todos mis profesores y profesoras de la carrera quienes cambiaron de forma drástica mi manera de ver el mundo, a los que representaron un reto; esos de quienes todos huían, son de los que más aprendí.

A mis profesores y profesoras de ciencias sociales quienes enriquecieron mi visión.

En especial a los que, en momentos complicados, me ayudaron a continuar con mi formación.

Contenido

Capítulo 3

Capítulo 4

Capítulo 5

01

Capítulo 1
Introducción

En este capítulo se pone en contexto el proyecto a realizar, se mencionan algunos robots y las aplicaciones en las que se usan, también se describen brevemente las herramientas disponibles para la elaboración de este proyecto que resuelve un problema en específico: la búsqueda de personas para su posterior rescate.

Motivación del proyecto

La norma ISO8373:2012 define los términos utilizados en relación con los robots y dispositivos roboticos que operan en entornos industriales y no industriales.

Cuando ocurren accidentes como derrumbes o incendios es necesario explorar la zona del siniestro en búsqueda de personas y no siempre es posible hacerlo sin poner en riesgo una vida. A pesar de que se usan equipos sofisticados como cámaras térmicas y detectores de gas, requieren ser operados manualmente. Por otro lado, acceder a las zonas afectadas siempre implica un riesgo, por lo cual es de interés presentar opciones autónomas y eficientes, siendo los robots una alternativa viable. De acuerdo a la norma ISO8373:2012 un robot es un mecanismo accionado programable en dos o más ejes con un grado de autonomía moviéndose dentro de su entorno para realizar tareas.

De acuerdo a la norma la autonomía es la capacidad para realizar tareas previstas en función del estado y la detección actuales, sin intervención humana.

En este proyecto se unen: las comunicaciones inalámbricas, un robot Lego, una computadora y una cámara de video para desarrollar una aplicación de búsqueda de personas.

Antecedentes

Las redes inalámbricas de sensores permiten supervisar un área determinada empleando un consumo bajo de energía. Estas redes no requieren de mucha potencia para transmitir ya que se trata de redes de área personal, sin embargo, no manejan altas tasas de transmisión de datos dado que se limita a unos cientos de kbps [1]. Por otro lado, en las aplicaciones en donde se requiere transmitir contenido multimedia, es necesario tener un medio de transmisión con mayor ancho de banda, para ello las versiones del protocolo IEEE 802.11 [2] brindan una mejor opción además de contar con mayor cobertura, aunque tienen la desventaja de consumir más energía en comparación con las redes que emplean el protocolo IEEE 802.15 [2]. En este proyecto se usa el protocolo IEEE 802.11, como se describe más adelante. Por su parte, en el ámbito de los robots, los más conocidos son los usados en la industria para la fabricación en serie, por ejemplo, de automóviles, en donde los robots se encargan de las tareas de soldar, ensamblar, pintar, etc. Existen también, robots para transportar carga, para tareas de limpieza y algunos otros que están siendo utilizados en la exploración espacial como el Curiosity [3] o la exploración marina como el BlueROV2 [4], con cámaras de video. En la figura 1.1 a) y b) se muestran estos robots.

a) b)

Figura 1.1. a) Robot Curiosity [3] b) Robot Bluerov2 [4]

Algunos de los puntos fuertes en el desarrollo de robots para aplicaciones de exploración en distintos tipos de terreno son: capacidad de desplazamiento, tipo de control, adquisición de datos y resistencia [5], [6], [7]. En este proyecto se utiliza un robot educativo llamado EV3 de la marca Lego.

Respecto a las cámaras de video asequibles al público están en constante desarrollo, cada vez son más compactas, resistentes, capaces de grabar con gran resolución, además de contar con protocolos de comunicación inalámbricos. Las principales marcas son: GoPro, Garmin y TomTom. En este proyecto se utiliza la cámara GoPro Hero 3+ black.

Actualmente la mayoría de las cámaras de video líderes en el mercado alcanzan resoluciones de imagen de hasta 4K y llegan a grabar a 240 cuadros por segundo dependiendo del tamaño de la imagen.

Redes Inalámbricas

Las redes inalámbricas son de bajo costo, utilizan pocos

recursos para la creación de estas redes y brindan libertad de movimiento dentro del área de cobertura. Existen dos modos para conectar los equipos de manera inalámbrica: el más conocido es el modo infraestructura, en el cual los equipos se asocian con un AP (Access Point, punto de acceso). El otro modo de conexión es ad-hoc donde los equipos se conectan directamente entre sí. En figura 1.2 se aprecian los dos modos de conexión: a) infraestructura y b) ad-hoc.

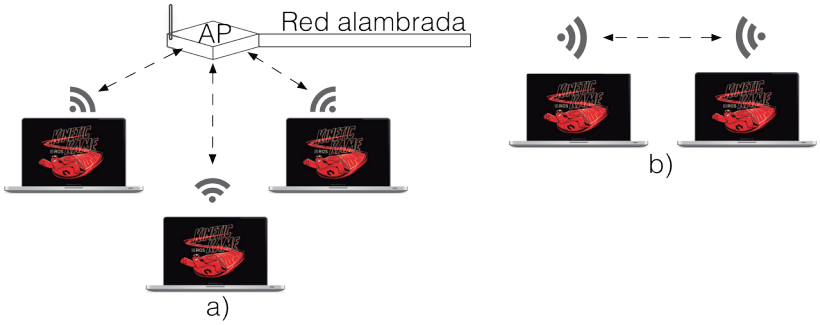

Figura 1.2. Modos de conexión de una red inalámbrica: a) infraestructura y b) ad-hoc.

Las redes inalámbricas presentan ventajas como la movilidad y el bajo costo para conectar varios equipos, sin embargo, también tienen ciertas desventajas como la interferencia inherente a transmisiones inalámbricas o la seguridad, ya que es posible interceptar las transmisiones colocándose dentro del área de cobertura.

Hardware y software utilizados en robots

La norma ISO8373:2012 define dos tipos de robot; robot industrial y robot de servicio. El robot de servicio realiza tareas útiles para humanos o equipos, excluidas las aplicaciones de automatización industrial.

Existen distintos tipos de robots, todos tienen diferentes características y las aplicaciones que se pueden hacer con ellos son variadas: robots verticales articulados para producción industrial como el VS-050/060, robots para la exploración marina como el BlueROV2 o robots capaces de volar como el Erle-HexaCopter [8]. En la figura 1.3 a), b) y c) se muestran los robots antes mencionados.

a) b) c)

Figura 1.3. Robots: a) VS-050/060, b) BlueROV2 y c) Erle-HexaCopter.

Para la aplicación de búsqueda de personas se utiliza el robot Lego EV3. Este robot puede recorrer superficies firmes con ligeras variaciones. Por otro lado, el software disponible para realizar las aplicaciones con robots se distribuye con el propio harware, sin embargo en este proyecto se utilizan recursos de

diferentes marcas y ROS (Robot Operating System) tiene como principio el desarrollo de sofware para robots en colaboración, siendo esta la principal herramienta para desarrollar este proyecto.

Descripción del problema

Explorar una zona en busca de personas, adquirir información del entorno e indicar la ubicación de una posible víctima son las principales necesidades cuando se trata de ver si hay alguien atrapado en accidentes como derrumbes, temblores o incendios. Para no arriesgar una vida humana en este tipo de siniestros se han empleado perros entrenados por su tamaño, agilidad y por contar con algunos sentidos más desarrollados que en el humano, como el olfato y el oído. Sin embargo sigue siendo un ser vivo susceptible a los factores ambientales. Para esta tarea también se han empleado robots equipados con cámaras y diferentes sensores que son operados a distancia. Actualmente se cuenta con tecnologías más compactas, sistemas de localización del tamaño de una moneda, cámaras cada vez más pequeñas y equipos electrónicos resistentes a altas temperaturas, así como robots con un mejor desplazamiento. También se ha desarrollado software capaz de consolidar las nuevas tecnologías. El reto es entonces reunir el mayor número de herramientas y tecnología

disponible en un solo sistema que brinde una opción viable para la búsqueda de personas en estos tipos de percances, tal es el objetivo del presente proyecto.

Objetivos

Implementar un robot autónomo capaz de explorar un área, evadir obstáculos, identificar personas, así como también guardar la ubicación en el caso de localizar a una persona y regresar al punto de partida.
Los objetivos específicos son los siguientes:

- Implementar un robot con la capacidad de evadir obstáculos y capaz de ubicarse en el espacio.
- Integrar una cámara de video al robot.
- Integrar un algoritmo de reconocimiento facial a la aplicación.
- Comunicar los equipos que integran la aplicación de manera inalámbrica.
- Proporcionar una interface visual para ver la ejecución de la aplicación.

Metodología

- Sustituir el sistema operativo original del robot EV3 por el sistema operativo EV3DEV para poder utilizar el lenguaje de programación Python.
- Realizar la comunicación inalámbrica necesaria para la transmisión de imágenes a una estación base y establecer la comunicación con todos los equipos de la aplicación.
- Agregar una cámara GoPro al robot EV3 y programar un algoritmo para controlarla y recibir las imágenes de manera inalámbrica.
- Implementar en el lenguaje de programación Python algoritmos que permitan controlar y obtener información de cada elemento del robot como: batería, sensor y motores para que el robot explore un área, evada obstáculos y regrese al punto de partida de la exploración de manera autónoma.
- Agregar un algoritmo de reconocimiento facial para reconocer personas.
- Programar los elementos necesarios para tener una representación gráfica del robot y los elementos que lo conforman al momento de realizar la ejecución de la aplicación.

02

Capítulo 2
Marco teórico

En este capítulo se describen los principios teóricos de las tecnologías usadas para llevar a cabo este proyecto, comenzando por los enlaces utilizados para comunicar al robot, la computadora y la cámara. Después se aborda de manera breve cómo se forma una imagen digital y en qué consiste el procesamiento de imagen, así como también la explicación básica de cómo funciona el algoritmo para la detección de rostros. Finalmente, se aborda el tema del software empleado para unir todas estas tecnologías para crear la aplicación de búsqueda de personas.

Comunicación entre dispositivos electrónicos

La realización de este proyecto contempla el uso de tres equipos diferentes: una computadora MacBookPro, el robot EV3 y una cámara GoPro, por ello es importante entender cómo se lleva a cabo la comunicación entre dispositivos de diferentes marcas.

El intercambio de información entre equipos electrónicos se realiza a través de redes de comunicación. Las redes de comunicación se clasifican de acuerdo a su cobertura en: redes de área personal (PAN, *Personal Area Networks*), redes de área local (LAN, *Local Area Networks*) y redes de área amplia (WLAN, *Wide Area Networks*). Las redes de área personal tienen

una cobertura de pocos metros y los dispositivos conectados a estas redes son de uso personal, las redes de área local cubren una pequeña área donde los dispositivos se encuentran en el mismo lugar geográfico, como puede ser un centro de trabajo, universidades, etc. mientras que las redes de área amplia implican grandes distancias y distintos sitios geográficos.

La mayoría de las redes se encuentran organizadas a partir de niveles o capas en donde cada capa depende de la inferior. El propósito de cada capa es ofrecer servicios a la capa superior sin dar detalles de cómo implementa estos servicios.

Un modelo simple de comunicaciones involucra los siguientes elementos para el intercambio de información: el emisor, el transmisor, el medio de transmisión, el receptor y el destino. Para que la comunicación se lleve a cabo de manera exitosa se tiene en cuenta que hay distintos fabricantes de equipos y que es necesario establecer reglas y convenciones para que todos los equipos se puedan comunicar entre sí. A estas reglas se les conoce como protocolos de comunicación y al conjunto de capas y protocolos se le conoce como arquitectura de red. La lista de protocolos utilizados por un sistema se le conoce como pila de protocolos.

La comunicación entre equipos se lleva acabo en capas equivalentes en cada uno, pasando la información a través de las capas inferiores hasta llegar al medio físico. Es importante

tener en cuenta que los servicios y los protocolos son conceptos diferentes. Un servicio es un conjunto de operaciones mientras que un protocolo es un conjunto de reglas que rigen el formato de los mensajes que se intercambian en una capa.

Existen dos arquitecturas de redes importantes: el modelo de referencia OSI (*Open Systems Interconnection)* y el modelo de referencia TCP/IP (lleva el nombre TCP/IP debido a los principales protocolos que emplea). En el caso del modelo de referencia OSI los protocolos que emplea casi no son usados, sin embargo, el modelo es general. Por otro lado el modelo de referencia TCP/IP no es del todo general, sin embargo, son los protocolos que más se emplean actualmente.

El modelo OSI surge como una propuesta de la Organización Internacional de Normas ISO (*International Standarization Organization*), cuenta con siete capas, este modelo no especifica los servicios y protocolos que se emplean en cada capa. Tiene como base los siguientes principios:

- Se debe de crear una capa en donde se requiera un nivel diferente de abstracción.
- Cada capa debe de realizar una función.
- La función se debe de elegir teniendo en cuenta la definición de protocolos estandarizados.
- Es necesario definir los limites de las capas para que se

minimice el flujo de información a través de las interfaces.
- La cantidad de capas debe ser suficiente para no agrupar funciones distintas en la misma capa.

El otro modelo de referencia importante TCP/IP tiene como objetivos principales: conectar varias redes, además de que la red pueda seguir operando aunque exista una pérdida de hardware en alguna parte de la subred sin que se interrumpan las comunicaciones existentes. Este modelo se emplea desde la red de computadoras más antigua ARPANET y en la sucesora Internet. En la tabla 2.1 se describen brevemente los dos modelos.

OSI	TCP/IP
Capa de aplicación Proporciona a las aplicaciones, como correo electrónico, los medios para acceder al entorno OSI.	Capa de aplicación Proporciona la forma de interacción entre el equipo y el usuario. Algunos de los protocolos usados en esta capa son: HTTP, TELNET, FTP, DNS, entre otros.
Capa de presentación Ofrece los servicios de transformación de datos como por ejemplo, la de compresión y cifrado.	

OSI	TCP/IP
Capa de sesión En esta capa se ofrecen varios servicios como: el control de dialogo, el manejo de tokens y la sincronización.	
Capa de transporte Su función básica es aceptar datos de la capa superior, dividirlos en unidades más pequeñas en caso de ser necesario, pasar los datos a la capa de red y asegurar que lleguen al otro extremo.	Capa de transporte Esta capa permite que las entidades pares en el emisor y receptor logren tener una conversación. En esta capa se definen dos protocolos, TCP (*Transmission Control Protocol,* Protocolo de Control de Transmisión) orientado a conexión que permite la entrega fiable de datos. Y el UDP (*User Datagram Protocol,* Protocolo de Datagrama de Usuario) no orientado a conexión.

OSI	TCP/IP
Capa de red Determina la manera en que se transportan los paquetes desde el origen hasta el destino, además del manejo de la congestión.	Capa de interred Esta capa es la que mantiene unida a la arquitectura TCP/IP, básicamente permite que los host manden paquetes en cualquier red para que viajen de manera independiente hasta el destino. En esta capa se define un formato de paquete y el protocolo oficial IP (*Internet Protocol*, Protocolo de Internet), además de un protocolo complementario ICMP (*Internet Control Message Protocol*, Protocolo de Mensajes de Control de Internet).
Capa de enlace de datos Su tarea principal es transformar un medio de transmisión puro en una línea que esté libre de errores de transmisión, activar, mantener y desactivar el enlace con el medio de transmisión; el emisor divide los datos de entrada en trama de datos, las transmite de forma secuencial.	Capa de enlace Esta capa describe qué enlaces se deben llevar a cabo (líneas seriales y Ethernet) para cumplir con las necesidades de esta capa sin conexión. Funciona más como una interfaz entre los host y los enlaces de transmisión.

Se le denomina hosts a las computadoras que ejecutan programas de usuario (aplicaciones).

OSI	TCP/IP
Capa física En esta capa se realiza la transmisión de bits, también todo lo referente a las propiedades físicas de los dispositivos como el tipo de conector, funciones de los circuitos, nivel de voltaje y secuencias de bits.	

Tabla 2.1. Comparación entre arquitecturas de protocolos OSI y TCP/IP.

En el modelo OSI se definen tres conceptos importantes:

- Servicios. Definen lo qué hace cada capa sin especificar cómo lo hace.
- Interfaces. Indican a las capas superiores cómo acceder a ellas.
- Protocolos. Cada capa decide cuáles protocolos usar a fin de llevar a cabo los servicios de cada una.

Por su parte, al principio del modelo TCP/IP no existía una clara diferencia entre servicios, interfaces y protocolos, esto se ha tratado de reajustar con el fin de que se parezca más al modelo OSI.

El modelo OSI surge antes de los protocolos por lo que no está

orientado a un conjunto específico de protocolos por lo que es muy útil al momento de hablar sobre redes de computadoras. Por otro lado el modelo TCP/IP tiene su fortaleza en sus protocolos los cuales surgen antes del modelo, por lo que el modelo en sí no sirve para describir otras redes que no sean TCP/IP. En este proyecto se utilizan los protocolos TCP/IP por lo que se explicará a continuación su funcionamiento a grandes rasgos.

En la figura 2.1 se muestra un ejemplo de intercambio de datos entre dos dispositivos utilizando TCP/IP, en el cual es posible que pasen a través de una o varias redes; al conjunto de estas redes se les denomina subredes [9].

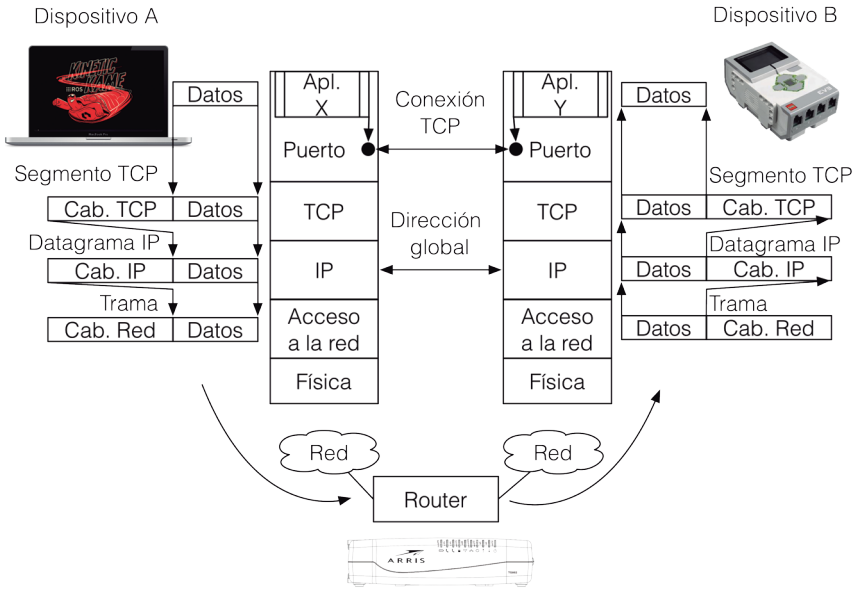

Figura 2.1. Intercambio de datos TCP/IP.

De acuerdo a la figura 2.1, en primer lugar el protocolo IP se implementa en todos los dispositivos: la estación emisora, la estación receptora y los dispositivos de enrutamiento, cuya función es la de transportar los datos. El protocolo TCP se implementa en los sistemas finales para asegurar que los datos se entreguen de manera confiable. Cada entidad en el sistema global debe de tener una dirección única, cada dispositivo en una subred debe de tener una dirección IP y por último cada proceso dentro del dispositivo debe de tener un identificador llamado puerto. De esta manera, el protocolo TCP puede entregar los datos al proceso correspondiente. Continuando con el ejemplo, un proceso asociado al puerto 1 en el dispositivo A envía un mensaje a otro proceso asociado al puerto 2 del dispositivo B. TCP pasa el mensaje al protocolo IP para enviarlo al dispositivo B, no es necesario que el protocolo IP conozca la identidad del puerto destino. El protocolo IP pasa el mensaje a la capa de acceso a la red con la instrucción de enviarlo al dispositivo de enrutamiento. Cuando el proceso emisor genera un bloque de datos y lo pasa a TCP, éste lo puede dividir en fragmentos más pequeños, a cada fragmento se le añade información de control llamada cabecera TCP que incluye los siguientes datos:

- Puerto destino. Cuando TCP en B recibe el segmento se debe conocer a qué proceso se le entregan los datos.

- Número de secuencia. TCP numera los fragmentos enviados para que TCP en B pueda reordenarlos.
- Suma de comprobación. TCP en el emisor incluye un código calculado en función del resto de los fragmentos, TCP en el receptor realiza el mismo cálculo y lo compara para asegurar que la transmisión se ha llevado a cabo con éxito.

Después TCP pasa los fragmentos al protocolo IP para que se transmitan hacia el dispositivo B a través de uno o más dispositivos de enrutamiento. En esta nueva etapa también se requiere de información de control llamada cabecera IP, a este nuevo conjunto de datos se le conoce como datagrama IP, dentro de la cabecera IP se incluye la dirección del dispositivo B. Finalmente los datagramas IP se pasan a la capa de acceso a la cual a su vez añade su propia cabecera, al resultado final se le conoce como paquete o trama. La trama se transmite a través del dispositivo de enrutamiento. La cabecera de la trama tiene entre otros campos los siguientes campos de interés para este ejemplo:

- Direccion de la subred destino. La subred requiere conocer a cuál dispositivo entregar la trama.
- Funciones solicitadas. Un ejemplo de estas funciones puede ser la utilización de prioridades.

En el dispositivo de enrutamiento la cabecera de la trama se quita y se examina la cabecera del paquete IP. El protocolo IP en el dispositivo de enrutamiento direcciona la trama a través de la red hacia B con base en la dirección destino de la cabecera del paquete IP. Al recibir los datos en el dispositivo B se realiza el proceso a la inversa.

IEEE 802.11

La red a través de la cual se realiza la comunicación entre los dispositivos empleados en este proyecto es una LAN inalámbrica empleando el estándar 802.11 versión n. A continuación una breve explicación de su funcionamiento.

En la jerga computacional al estándar 802.11 se le conoce también como WiFi.

A mediados de la decada de 1990 surge IEEE 802.11, con el objetivo de desarrollar un protocolo de acceso al medio y la especificación del medio para LAN inalámbricas. Este estándar opera en bandas sin licencias como las bandas ISM (*Industrial Scientific and Medical*, Industriales, Científicas y Medicas) definidas por el ITU-R (*International Telecommunication Union - Radiocommunication Sector*, Sector de Radiocomunicaciones de la Unión Internacional de Telecomunicaciones), por ejemplo, 902-929 MHz, 2.4-2.5 GHz, 5.725-5.825 GHz.

Las LAN 802.11 se componen de estaciones (ST) como: laptops, celulares, cámaras o cualquier dispositivo equipado con 802.11

y puntos de acceso (AP) inalámbricos que están conectados a la red alambrada. La comunicación entre los dispositivos en una LAN inalámbrica se realiza utilizando los puntos de acceso, sin embargo dos dispositivos que se encuentren dentro de su rango de cobertura se pueden comunicar directamente, este tipo de conexión se conoce como ad-hoc [10].

A los puntos de acceso se les conoce también como estaciones base.

En este proyecto la cámara, el robot y computadora se conectan entre sí de manera inalámbrica a través de dos puntos de acceso. La cámara cuenta con opción de conexión de manera inalámbrica, pero sólo como punto de acceso, admitiendo un usuario solamente. El robot se conecta a un punto de acceso utilizando un adaptador USB 802.11n. Entonces, los equipos se conectan de la siguiente manera: la computadora se conecta al router utilizando un puerto ethernet, el robot se conecta al router de manera inalámbrica empleando el adaptador USB 802.11n y la computadora se conecta a la cámara por medio de su interface inalámbrica; de esta manera todos los dispositivos se logran comunicar. En la figura 2.2 se muestra la topología empleada y en la tabla 2.2 se mencionan los términos clave empleados en el estándar IEEE 802.11 [11].

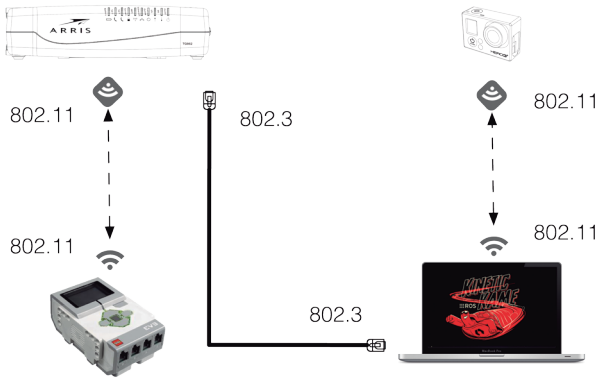

Figura 2.2. Topología utilizada en este proyecto.

AP (*Access Point, Punto de Acceso*)	Cualquier entidad que tenga la función lógica de estación y que proporcione acceso al sistema de distribución a través del medio inalámbrico a las estaciones asociadas.
BSS (*Bassic Set Service*, Conjunto Básico de Servicios)	Conjunto de estaciones controladas por una sola función de coordinación.
Función de coordinación	Función lógica que determina cuándo una estación funcionando dentro de un BSS tiene permiso para transmitir y puede recibir una unidad de datos de protocolo PDU (*Protocol Data Unit*).
DS (*Distribution System*, Sistema de distribución)	Un sistema para interconectar un conjunto de BSS y LAN integradas para crear un ESS.

ESS (*Extended Service Set*, Conjunto Extendido de Servicios)	Conjunto de uno o más BSS interconectados y LAN integradas que aparece como un único BSS en la capa de control de enlace lógico (LLC, *Logic Link Control*) de cualquier estación asociada con uno de tales BSS.
MPDU (*MAC Protocolo Data Unit*, Unidad de datos del protocolo MAC)	Unidad de datos intercambiada entre entidades MAC (*Medium Acces Control*) paritarias usando los servicios de la capa física.
MSDU (*MAC Service Data Units*, Unidades de datos de servicio MAC)	Información entregada como una unidad entre usuarios MAC.
Estación	Cualquier dispositivo con capas físicas y MAC compatibles con 802.11.

Tabla 2.2. Terminos clave en el estándar IEEE 802.11.

El elemento principal es el BSS que consiste en un número de estaciones ejecutando el mismo protocolo MAC, compitiendo por el acceso al medio inalámbrico compartido. El BSS puede funcionar de manera aislada o estar conectado a un sistema troncal de distribución mediante un punto de acceso. Por otro lado el DS puede ser la red alambrada u otra red inalámbrica. En la figura 2.3 se ilustra la interacción de los elementos importantes del estándar.

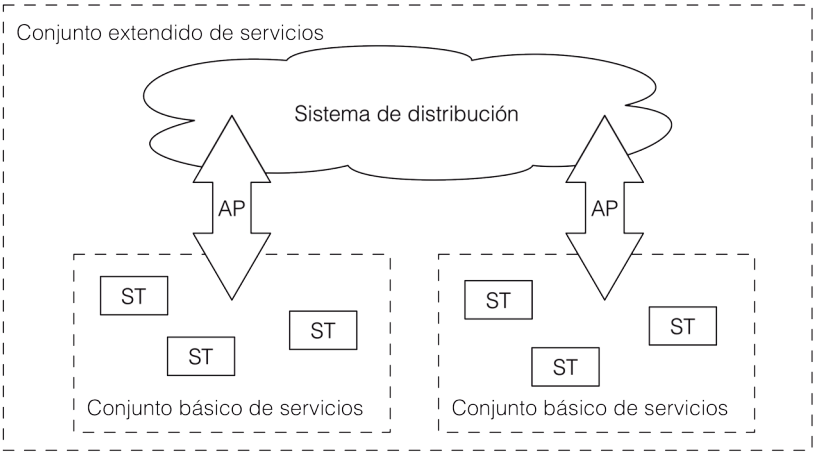

Figura 2.3. IEEE 802.11, figura propia con base en la referencia [12].

En la tabla 2.3 se aprecian los nueve servicios de IEEE 802.11 para que la red funcione como si fuera un red alambrada.

Servicio	Proveedor	uso
Asociación	Sistema de distribución	Entrega de MSDU
Autenticación	Estación	Acceso a LAN
Fin de la autenticación	Estación	Acceso a LAN
Disociación	Sistema de distribución	Entrega de MSDU
Distribución	Sistema de distribución	Entrega de MSDU
Integración	Sistema de distribución	Entrega de MSDU
Entrega MSDU	Estación	Entrega de MSDU
Privacidad	Estación	Acceso a LAN
Reasociación	Sistema de distribución	Entrega de MSDU

Tabla 2.3. Servicios IEEE 802.11.

De la tabla 2.3 se puede ver que el proveedor de servicios puede ser tanto la estación como el sistema de distribución DS. Los servicios de la estación son implementados en cada estación IEEE 802.11.
Tres de los servicios enumerados se emplean para controlar el acceso a una LAN IEEE 802.11 y también para proporcionar confidencialidad. Los servicios restantes dan soporte a la entrega de datos de servicio MAC (*MSDU, MAC Service Data Units*).

Distribución de mensajes dentro del DS

Para llevar a cabo el envío de mensajes se requieren los servicios de distribución e integración. El servicio de distribución es empleado para intercambiar tramas MAC de un BSS a otro. Si las estaciones están dentro del mismo BSS, el servicio de distribución pasa lógicamente mediante el AP. El servicio de integración permite la transferencia de las tramas cuando es necesario intercambiar datos desde una estación en una LAN 802.11 y una LAN 802.x que se encuentre conectada con la primera.

Servicios relacionados con la asociación

El objetivo de la capa MAC es el envío de MSDU entre entidades MAC. Esta tarea la desempeña el sistema de distribución, sin embargo, para que esto suceda, primero tiene que asociarse la estación destino, es decir, debe de estar en el conjunto extendido de servicios. Para esta tarea se hace uso de tres servicios:

- Asociación. Establece una asociación inicial entre un AP y una estación.
- Reasociación. Permite que una asociación previa se pueda transferir de un AP a otro.
- Disociación. Notifica que la asociación termina, puede ser en el AP o en la estación.

Servicios de acceso y privacidad

En el caso de las redes inalámbricas, cualquier estación en el rango de cobertura puede transmitir y recibir. Para limitar el acceso de dispositivos ajenos, el estándar IEEE 802.11 proporciona tres servicios:

- Autenticación. Este servicio es utilizado para que la estación envíe su identidad. El estándar no impone ningún esquema

de autenticación en concreto, por lo que puede tratarse de algún procedimiento relativamente inseguro hasta diferentes esquemas de cifrado.
- Fin de la autenticación. Este servicio indica el fin de la autenticación.
- Privacidad. Servicio que "asegura" que los datos transmitidos no sean leídos por una estación ajena a la que se enviaron originalmente.

En este proyecto estos tres servicios se están utilizando en el robot y la cámara, ya que para conectarse tanto el robot al router como la computadora a la cámara se requiere de una clave.

Control de acceso al medio en IEEE 802.11

La capa MAC de IEEE 802.11 cumple con tres funciones importantes: entrega fiable de datos, control de acceso y la seguridad, aunque el tema de la seguridad no se contempla en este proyecto, cabe mencionar que aunque el tipo de datos que se trasmiten no son de interés comercial, son valiosos por tratarse información que puede salvar una vida humana.

Entrega fiable de datos

Como la señal se transmite de forma inalámbrica, puede ser afectada por interferencia y ruido. En capas más altas, protocolos como TCP realizan mecanismos para que la entrega de datos sea confiable, sin embargo, esto representa tiempo de retardo. Por tanto, el estándar IEEE 802.11 proporciona una trama de confirmación ACK (*acknowledge*) a la estación origen como mecanismo de entrega fiable de datos.

Para el intercambio de datos se requieren los siguientes elementos al transmitir: en primer lugar la fuente solicita un permiso para enviar hacia el destino (RTS, *request to send*), la estación destino responde con el permiso para enviar (CTS, *clear to send*), una vez que se recibe el permiso para enviar se realiza el envío de datos. Por último, la estación destino responde con una trama de confirmación de recepción ACK, si esta trama no es recibida por la fuente, se vuelve a transmitir la trama. Una vez que el intercambio de datos se realiza, las estaciones fuente y destino avisan a las demás estaciones que se está llevando a cabo el envío de datos utilizando las tramas RTS y CTS con la finalidad de evitar transmisiones y colisiones de tramas.

Control de acceso

En el estándar IEEE 802.11 se consideran dos tipos de algoritmos MAC: protocolo de acceso distribuido y protocolo de acceso centralizado. El protocolo de acceso distribuido cobra sentido en una red ad-hoc. La decisión de transmitir se distribuye sobre los nodos empleando un mecanismo de detección de portadora, mientras que el protocolo de acceso centralizado es empleado cuando varias estaciones se encuentran interconectadas a través de un punto de acceso.

Capa física IEEE 802.11

En el estándar original se definen tres medios físicos: espectro expandido de secuencia directa, espectro expandido con salto de frecuencia trabajando en la frecuencia de 2.4 GHz, e infrarrojos. Todos a velocidad de 1 y 2 Mbps [11].
El estándar IEEE 802.11 se sigue desarrollando y se han agregado diferentes técnicas de transmisión a la capa física de este estándar, estas técnicas operan en las bandas de 2.4 GHz y 5 GHz. A continuación se describen las diferentes versiones hasta la utilizada en este proyecto.

IEEE 802.11b

Esta versión del estándar fue aprobada antes que la versión a, por esa razón se pone en primer lugar. Utiliza la técnica de espectro disperso con velocidades que van de 1 a 11 Mbps. Para enviar a diferentes velocidades se emplean distintos tipos de modulaciones.

IEEE 802.11a

La versión del estándar 802.11b tiene siete veces mayor alcance que la versión 802.11a.

Una de las razones por las cuales este estándar salió al mercado después de la versión b, es por que opera en una banda más alta de los 5 GHz. Tiene como base la técnica de modulación OFDM y trabaja a velocidades de transmisión que van de los 6 Mbps a los 54 Mbps. Aunque la velocidad de transmisión es mayor, la versión b del estándar tiene mayor alcance.

IEEE 802.11g

En el año 2003 la IEEE aprueba esta versión del estándar, la cual utiliza la misma modulación que la versión 802.11a, pero trabajando en la banda de los 2.4 GHz logrando ofrecer las mismas tasas de transmisión de datos (de 6 Mbps a 54 Mbps).

IEEE 802.11n

Esta versión del estándar se ratifica en el año 2009 con el propósito de aumentar de manera considerable la tasa de transmisión de datos (por lo menos 100 Mbps). Para ello, duplicó los canales de 20 MHz a 40 MHz. Una peculiaridad de este estándar es que utiliza hasta cuatro antenas para transmitir diferentes flujos de información al mismo tiempo. Las señales de estos flujos llegan al receptor, en el cual se separan utilizando las técnicas de comunicaciones MIMO (*Multiple Input Multiple Ouput*, Múltiples Entradas Múltiples Salidas). Al aumentar el número de antenas se aumenta además de la velocidad, el alcance y la confiabilidad.

Para una breve introducción a las antenas múltiples en el 802.11 se puede consultar el artículo "802.11 with Multiple Antennas for Dummies"[13].

Procesamiento digital de imagen

Para poder reconocer a una persona a través de una cámara es necesario llevar a cabo análisis y procesamiento digital de imagen. Para lo cual es necesario dar una breve explicación que una imagen es producto de la reflexión de algún tipo de radiación capturada en algún medio físico, esta radiación puede ser visible al ojo humano o no. Por otro lado, una secuencia de imágenes reproducida a un determinado tiempo es percibida por el cerebro como una imagen en movimiento (video). Actualmente rara vez se observa una imagen que no sea en un formato

digital. Para poder capturar una imagen digital se requiere de una cámara cuyos componentes principales son: el lente, el sensor y la memoria. El lente dirige la luz hacia el sensor, en el sensor se lleva acabo la digitalización de la imagen tomando muestras de la onda de luz, cuantificando y codificando los valores de estas muestras, para después almacenar estos datos en la memoria. Por ejemplo una imagen en blanco y negro se representa por una matriz de dos dimensiones, en donde la dimensión de la matriz representa el número de muestras y los valores numéricos de estas muestras representan la escala de grises con las que se muestra la imagen. En la figura 2.4 se muestra el acercamiento a una imagen en donde se aprecia la sección de la matriz anteriormente descrita.

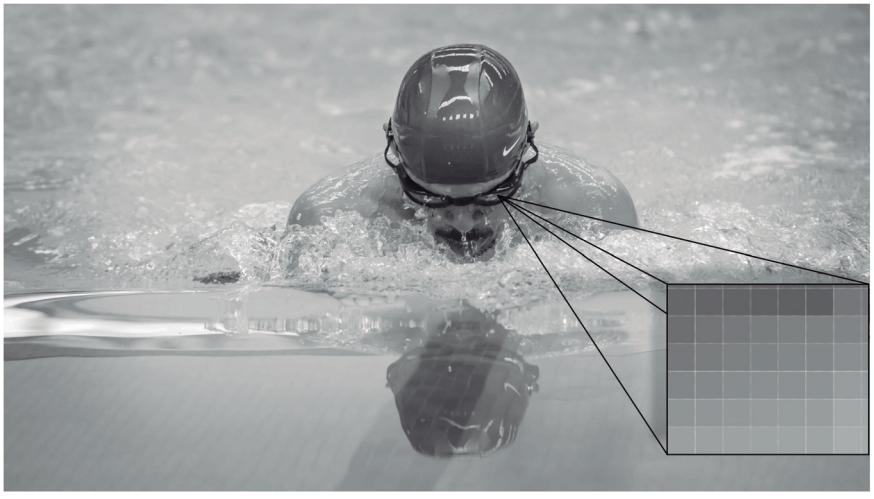

Figura 2.4. Acercamiento a una imagen.

La intersección de un renglón y una columna de la matriz de la imagen se le conoce mejor con el nombre de pixel (*picture x element*). Entre mayor sea el número de muestras que se tienen se pueden apreciar más detalles de la imagen. En la figura 2.5 se muestra la misma imagen pero con diferente número de muestras, comenzando con 128 x 72 pixeles, 256 x 144 pixeles y por último 512 x 288 pixeles.

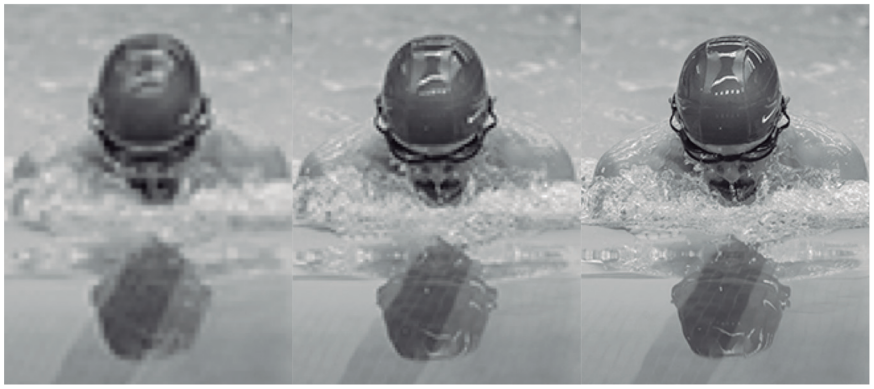

Figura 2.5. Misma imagen, pero con diferente resolución.

El valor numérico de los pixeles se le conoce como cuantificación, entre mayor sea este valor, más niveles de tonos se pueden representar, por ejemplo, si se tuvieran dos valores en cada pixel, sólo se podría formar la imagen con un tono de blanco y un tono de negro. Para conocer con cuántos niveles de grises (en una imagen en blanco y negro) se puede representar una

imagen, se emplea la siguiente formula:

$$2^b = k$$

En donde b es el número de bits, y k es el número de niveles de grises que se pueden representar, por ejemplo, una imagen con 8 bits se puede representar con 256 valores diferentes ya que $2^8 = 256$. En la figura 2.6 se muestra la misma imagen con diferente cuantificación comenzando por 6, 7 y 8 número de bits por pixel, respectivamente.

Figura 2.6. Fotografía con la misma resolución pero con diferente cuantificación.

Por otra parte, las imágenes a color están representadas por la suma de tres matrices R, G, B, (*Red, Green, Blue*, rojos, verdes y azules). En el grupo de imágenes de la figura 2.7 se muestran las matrices R, G y B cuya suma da como resultado una imagen a color [14].

Figura 2.7. Componentes R G B de una imagen a color.

El procesamiento de imagenes se reduce a operaciones con matrices. Para el procesamiento de imagen, en este proyecto se emplea la biblioteca de software OpenCV.

OpenCV

Open Source Computer Vision es una biblioteca abierta que contiene algoritmos para el procesamiento de imagenes. Su estructura es modular en donde cada módulo tiene una biblioteca para una función específica, los principales módulos son los siguientes:

- Core. Es un módulo compacto que define las estructuras básicas de datos, incluyendo arreglos multidimensionales

matemáticos y funciones básicas que son empleadas por otros módulos.

- Imgproc. Este módulo cuenta con filtrado de imagen lineal y no lineal, transformación de imagen geométrica, cambio de espacio de color, histogramas, etc.
- Video. Módulo de análisis de video que incluye varios algoritmos como: estimación de movimiento, seguimiento de objetos, eliminación de fondos, entre otros.
- Calib3d. Algoritmos básicos de geometría de vista simple, calibración de cámara única y en estéreo (dos cámaras), estimación de posición de objeto, algoritmos de correspondencia estéreo y de elementos de reconstrucción 3D.
- Objdetec. Módulo para la detección de objetos predefinidos como autos, rostros, ojos etc.
- Highgui. Interface para capturar imágenes y video en diferentes formatos (H.264, MPEG-4, JPG, PNG, etc.).
- GPU. Algoritmos acelerados de unidad de procesamiento gráfica (GPU, Grafic Procesor Unit) para otros módulos de OpenCV.

Existe una distribución de OpenCV para Windows, Linux/Mac, android e iOS. Trabaja con Java, Python, C++ y Matlab [15].

Detección de rostros

Para realizar la detección de rostros en este proyecto se emplea un algoritmo de la biblioteca OpenCV el cual tiene tres elementos principales: una nueva representación de la imagen original llamada imagen integral, un algoritmo clasificador que funciona con base en el algoritmo AdaBoost (*Adaptive Boosting*) [16], el cual selecciona un grupo reducido de características de un conjunto más grande para producir un clasificador más eficiente y por último combina clasificadores en cascada cada vez más complejos para aumentar la velocidad del detector enfocando la atención en regiones prometedoras de la imagen.
Este algoritmo trabaja por subventanas y clasifica las imágenes en función del valor de las características. Las características son figuras rectangulares dentro de una ventana de detección. Tiene como base tres tipos de características: dos rectángulos, tres rectángulos y cuatro rectángulos. En el figura 2.8 se muestran estas características.

El algoritmo empleado es descrito en el artículo: *"Rapid Object Detection using a Boosted Cascade of Simple Features"*
Cuyos autores son:
Paul Viola y Michael Jones.

El *Boosting* se refiere a un método general y probadamente efectivo para producir una regla de predicción muy precisa combinando reglas generales y moderadamente inexactas.

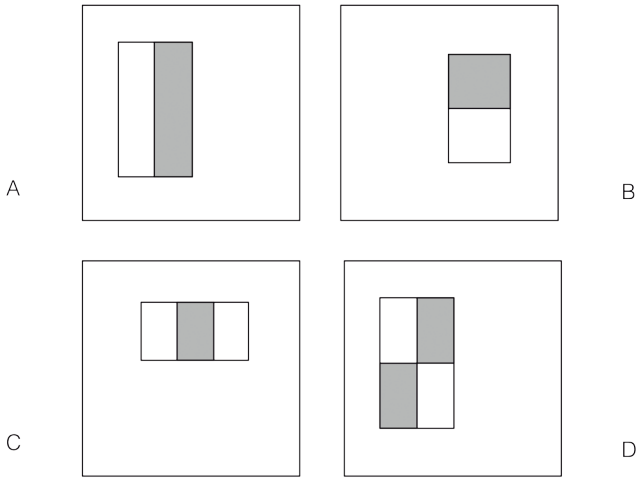

Figura 2.8. Características rectangulares mostradas en relación con la ventana de detección [17, Fig. 1].

- El valor de una característica de dos rectángulos está dada por la diferencia entre la suma de los píxeles dentro de dos regiones rectangulares. Las regiones tienen el mismo tamaño y forma y son horizontal o verticalmente adyacentes (figura 2.8 A y B).
- Una característica de tres rectángulos calcula la suma dentro de dos rectángulos exteriores restados de la suma en un rectángulo central (figura 2.8 C).
- Finalmente, una característica de cuatro rectángulos calcula la diferencia entre los pares diagonales de rectángulos (figura 2.8 D).

El valor de las características de un rectángulo puede ser calculado mediante una imagen intermedia llamada imagen integral en donde el valor de (x,y) de la imagen integral contiene la suma de los pixeles de arriba a la izquierda de (x,y) de la imagen original.

La ecuación 1 modela el cálculo de la imagen integral [18]:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y'), \quad (1)$$

De la ecuación 1 se puede observar que para realizar el cálculo de la imagen integral, la ventana de detección recorre toda la imagen original.

En este algoritmo los autores utilizan una variante del algoritmo *AdaBoost* para seleccionar un pequeño conjunto de características y entrenar al clasificador. En su forma original, el algoritmo de aprendizaje de *AdaBoost* se utiliza para impulsar el rendimiento de clasificación de un algoritmo de aprendizaje simple. El algoritmo de aprendizaje está diseñado para seleccionar la característica de rectángulo único que mejor separe los ejemplos positivos y negativos. Un ejemplo positivo es una imagen con rostro y un ejemplo negativo es una imagen que no contiene rostro.

En los experimentos que realizaron al algoritmo, los autores reportan que un clasificador de cara frontal construido a partir de 200 características produce una tasa de detección del 95% con una tasa de falsos positivos de 1 en 14,084.

Para la tarea de detección de rostros, las características rectangulares iniciales seleccionadas por *AdaBoost* son significativas y fáciles de interpretar, en la figura 2.9 se muestra un ejemplo con la primera y la segunda característica seleccionada por *AdaBoost*.

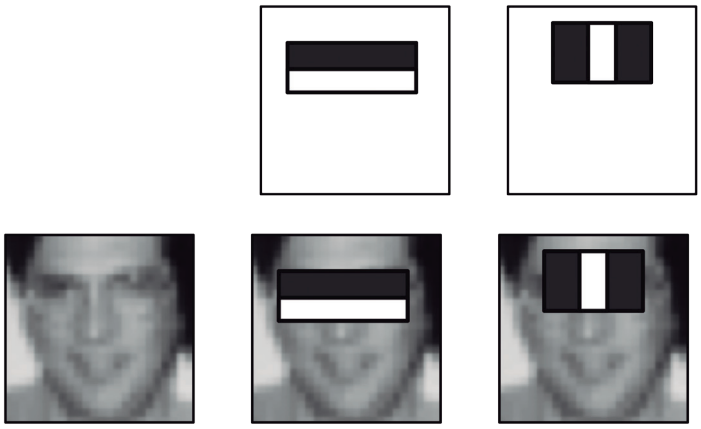

Figura 2.9 Ejemplo de características iniciales *AdaBoost* [17, Fig. 3].

Al utilizar clasificadores en cascada los autores logran un mayor rendimiento de detección a la vez que se reduce radicalmente el tiempo de cálculo. Así mismo, al utilizar clasificadores más simples se rechaza a la mayoría de las ventanas secundarias antes de recurrir a clasificadores más complejos para lograr tasas bajas de falsos positivos. Un resultado positivo del primer clasificador desencadena la evaluación de un segundo clasificador que

también se ha ajustado para lograr tasas de detección muy altas. Un resultado positivo del segundo clasificador desencadena un tercer clasificador, y así sucesivamente. Un resultado negativo en cualquier punto lleva al rechazo inmediato de la ventana secundaria. La cascada completa de detección de rostros tiene 38 etapas con más de 6,000 características. Sin embargo, la estructura en cascada da como resultado tiempos rápidos de detección promedio. En un conjunto de datos difícil, que contiene 507 caras y 75 millones de ventanas secundarias, las caras se detectan utilizando un promedio de 10 evaluaciones de características por cada ventana secundaria.

El clasificador en cascada de 38 etapas lo diseñaron para detectar caras verticales frontales. Para entrenar al detector, los autores utilizaron un conjunto de imágenes de entrenamiento de cara y no cara. El conjunto de entrenamiento facial reportan que constó de 4,916 caras que etiquetaron a mano, escaladas y alineadas a una resolución base de 24 por 24 píxeles. Las imágenes con rostro las bajaron de internet de manera aleatoria. En la Figura 2.10 se muestran algunos ejemplos de estas imágenes. Las subventanas no faciales utilizadas para entrenar el detector provienen de 9,544 imágenes que examinaron manualmente [17].

Figura 2.10. Imágenes de entrenamiento para el algoritmo de reconocimiento facial [17, Fig. 5].

ROS

ROS (*Robot Operating System*) es un sistema que permite crear software para robots, en el que se conjuntan herramientas y bibliotecas para una amplia gama de robots. La licencia para poder utilizar ROS es libre y se puede emplear en productos comerciales y de código cerrado. Ofrece comunicación entre dos o más sistemas utilizando mensajes, un sistema los publica y otro u otros sistemas se suscriben a ellos. Provee bibliotecas y herramientas específicas para que el sistema del robot pueda ejecutarse de manera rápida y eficiente. Tiene una base de

mensajes estándar utilizados por robots como pueden ser del tipo geométrico, involucrando conceptos como posición, vectores y transformación. Resuelve temas importantes como proveer sistemas de referencia de ubicación entre las partes del robot. También tiene herramientas para describir y modelar el robot de tal manera que se pueda identificar de manera visual. En la figura 2.11 se muestra el logotipo de ROS.

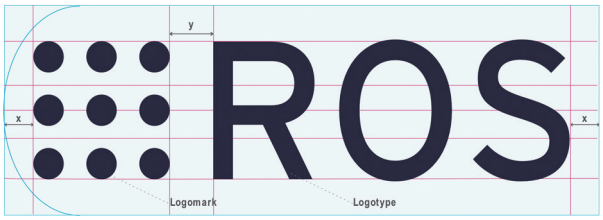

Figura 2.11. Logotipo ROS [19].

Los principales elementos de ROS son:

- Roscore. Es el núcleo, el corazón del sistema, el cual ejecuta inicialmente varios nodos para el funcionamiento de ROS, tiene como función principal asignar nombres o identificadores a todos los elementos en el sistema para poder enviar y recibir mensajes o servicios entre ellos.
- Nodos. Es un archivo ejecutable dentro de un paquete de ROS, este archivo ejecutable puede ser un programa escrito en diferentes lenguajes de programación, por ejemplo Python.

Los nodos son procesos de computación.

- Mensajes. Son los datos que se generan en los nodos que son transportados mediante los buses de comunicación del sistema llamados topics, los mensajes pueden ser de varios tipos, pueden ser una cadena de caracteres, datos de posición del robot, velocidad, etc.
- Topics. Es el bus de datos mediante el cual se manda la información, para emplearlos se requiere definir el nodo que va a publicar los mensajes y el nodo que se va a suscribir para recibir los mensajes, se pueden obtener las características de este bus, como por ejemplo el ancho de banda, velocidad con la que se envían los mensajes y el tipo de datos que transporta, ya que es necesario especificar si va a transportar caracteres, datos numéricos enteros o flotantes, etc.

Los topics son flujos de datos de computación.

- URDF. (*Unified Robot Description Format*), es un archivo xml que contiene las propiedades físicas y de apariencia del robot, en este formato se describen las partes que conforman al robot y cómo están unidas.
- Rviz. Proporciona una interface visual de tres dimensiones que lee e interpreta los datos de archivos URDF.
- Rqt. Permite modificar y configurar plugins para desarrollar interfaces gráficas para los robots, por ejemplo, uno de los plugins de *rqt* es *rqt_graph* en el cual se muestra de manera gráfica la interacción entre los nodos que se están ejecutando en un sistema ROS.

Un plugin es una aplicación que se ejecuta dentro del software principal que provee funciones adicionales.

El software que se desarrolla en ROS está organizado en paquetes, estos paquetes contienen los elementos necesarios para su funcionamiento. Por ejemplo, el paquete que se desarrolla en este proyecto para la aplicación de búsqueda de personas se llama robot. Los elementos que lo conforman de manera general son: la carpeta src y dos archivos con terminación .txt y .xml. En la carpeta src se crean las carpetas: scripts que contienen los programas desarrollados en Python, la carpeta de nombre rviz contiene la configuración de la interface visual, la carpeta urdf que contiene el archivo .xml con la descripción del robot y la carpeta launch con el archivo que inicia todos los nodos necesarios para la aplicación. En la imagen 2.12 se muestran las carpetas antes mencionadas del paquete *robot*, más adelante se analiza a detalle el contenido de este paquete.

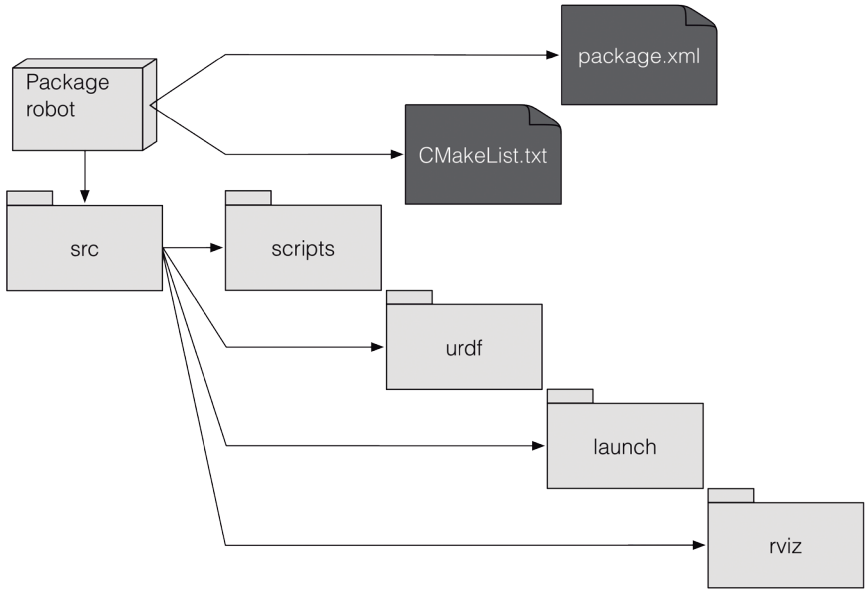

Figura 2.12. Paquete robot.

El archivo CMakeList.txt es un script de un sistema de compilación multiplataforma, este archivo contiene las herramientas de compilación de CMake, su estructura se especifica en los siguientes párrafos, en donde los términos entre paréntesis se refieren a las funciones que se utilizan en el archivo:

- *Required CMake Version* (*cmake_minimum_required*). Especifica la versión de CMake requerida para el paquete.
- *Package Name* (*project*()). Nombre del paquete que se está desarrollando.

- *Find* other *CMake/Catkin packages needed for build (find_package*()). En esta parte es donde se especifican los elementos requeridos para compilar el proyecto, si depende de otros paquetes estos se convierten en componentes. Cuando un paquete es encontrado por *find_package* se crean las variables de entorno CMake que contienen: la ubicación de los archivos, de qué tipo de bibliotecas depende y la ruta hacia esas bibliotecas.
- *Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files()).* Generan archivos de lenguaje de programación específicos para que se puedan utilizar los mensajes, ya que cada mensaje requiere una compilación especial, por ejemplo, no es lo mismo un mensaje de texto a mensajes con coordenadas espacio temporales.
- *Invoke message/service/action generation (generate_messages()).* Se encarga de llamar a los mensajes en el orden establecido.
- *Specify build targets*. Los objetivos de compilación se representan generalmente de dos formas: Objetivo ejecutable, el cual es el programa que se puede ejecutar, y el objetivo de las bibliotecas, las cuales son utilizadas en la compilación o mientras se lleva a cabo la ejecución.
- *Libraries/Executables to build (add_library()/add_executable*()/*target_link_libraries*()). Si el paquete que se desarrolla genera mensajes, hay que crear una dependencia

explícita en el destino del mensaje para que se compile en el orden correcto.

- *Tests to build (catkin_add_gtest()).* Provee una forma específica de hacer pruebas en catkin, es decir en el espacio de trabajo.
- *Install rules (install())* En está parte se específica en dónde se va a instalar el código generado.

El segundo archivo, el .xml, es un manifiesto donde se coloca el nombre del paquete, la versión, el autor y quién es el que da mantenimiento al programa junto con las dependencias que emplea, por ejemplo, es aquí donde se coloca el nombre del programador, el tipo de licencia con la que cuenta el paquete, la descripción del mismo y las dependencias que se están usando. Si ya se creó el paquete y se requieren más dependencias se pueden agregar en este archivo.

En este proyecto se utiliza la versión de ROS Kinetic Kame. La figura 2.13 muestra el logotipo de la versión de ROS utilizada en este proyecto.

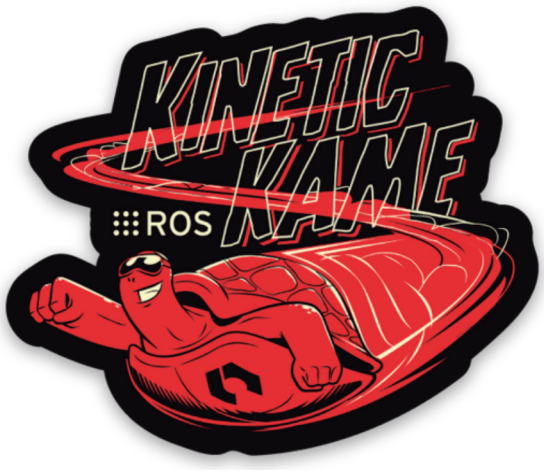

Figura 2.13. Logotipo Kinetic Kame [20].

03

Capítulo 3
Implementación

En este capítulo se describe el hardware disponible: el robot EV3 de Lego y la cámara GoPro Hero 3+ Black, la instalación del software necesario tanto en la computadora como en el robot, la configuración de los enlaces inalámbricos entre los dispositivos utilizados y la programación para realizar la aplicación de búsqueda de personas.

"Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño"
Primera ley de la robotica de Isaac Asimov.

Robot EV3

El EV3 cuenta con cuatro puertos de entrada y cuatro puertos de salida RJ12, puerto USB 2.0 y USB 1.1, ranura para memoria Micro SD y comunicación inalámbrica a través de BlueTooth. El bloque EV3 tiene un procesador ARM9 a 300 MHz con 64 MB de memoria RAM. Viene con un sistema operativo con base LINUX y funciona con seis baterías AA. En la figura 3.1 se muestra el bloque EV3.

Figura 3.1. Bloque EV3 de Lego [21].

Se cuenta con dos tamaños de motores para el EV3: grande y mediano a 160 rpm (revolución por minuto) y 250 rpm con torques de 20 Ncm (Newton centímetro) y 8 Ncm respectivamente, se pueden ver algunas diferencias entre los motores en la figura 3.2.

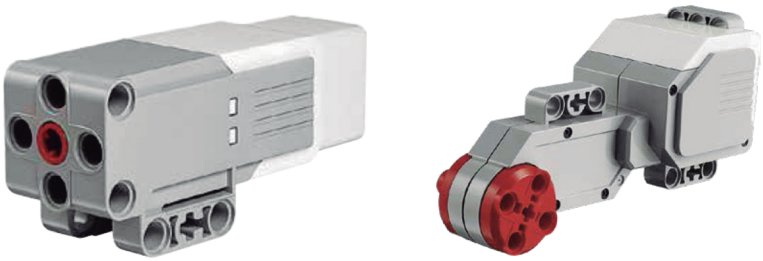

Figura 3.2. Motores EV3 grande y mediano [21].

Se tienen además sensores de color, infrarrojo y de contacto. El sensor de color puede ser utilizado en tres modos distintos: detección de color en el cual puede reconocer hasta siete colores, modo de luz reflejada y modo de luz ambiental, el sensor infrarrojo para detectar obstáculos y un sensor táctil con tres modos: presionado, lanzado y en contacto. En la figura 3.3 se muestran los sensores disponibles para este proyecto.

Figura 3.3. Sensores EV3: infrarrojo, touch, color [21].

GoPro Hero 3+ Black

Las cámaras GoPro son empleadas para grabar en situaciones dinámicas, son de tamaño compacto y con accesorios que les permiten sumergirse en el agua, soportar impactos fuertes, además de tener una buena resolución de imagen. La cámara usada en este proyecto es la GoPro Hero 3+ Black la cual puede grabar video con una resolución de hasta 4,093 pixeles horizontales a quince cuadros por segundo, tomar fotografías de 12 MP, cuenta con puerto IEEE 802.11n y ranura para una tarjeta Micro SD de hasta 64 GB de capacidad. Tiene un lente gran angular que permite tener un amplio campo de visión, en la figura 3.4 se muestra el modelo de la cámara utilizada [22].

La distancia focal para el modo de visión amplio mediano y reducido es de: 17.2 mm, 21.9 mm y 34.4 mm. La distancia focal indica el ángulo de visión, a distancias más cortas el campo de visión se reduce.

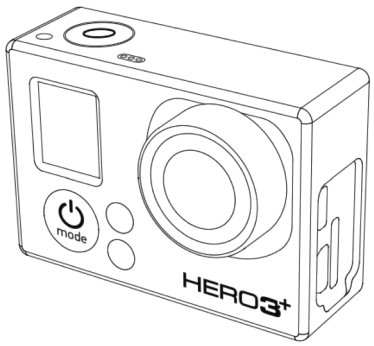

Figura 3.4. Cámara GoPro [22].

Instalación de software

A continuación se describe la instalación de ROS, la biblioteca goprohero, y el sistema operativo EV3DEV. La biblioteca OpenCV está incluida en la versión de ROS Kinetic. Si es de interés del lector conocer los comandos empleados durante la instalación puede consultar el apéndice más adelante.

Instalación ROS Kinetic, creación del espacio de trabajo catkin y la creación del paquete robot

La versión de ROS utilizada en este proyecto es la versión Kinetic para el sistema operativo Ubuntu 16.04. Kinetic tiene la opción para instalar de manera completa o por paquetes específicos, para este proyecto se realiza la instalación de manera completa. Después se deben iniciar las dependencias que el software requiere para funcionar. Existen además herramientas como rosinstall, para instalar de manera rápida paquetes adicionales a ROS, por lo que es conveniente tenerla disponible.

Una vez que se tienen instaladas todas las herramientas y dependencias se crea un espacio de trabajo en el cual se desarrollan los paquetes, se compilan y se obtienen archivos ejecutables. El espacio de trabajo que se emplea en este proyecto

es catkin, en primer lugar se crean las carpetas catkin_ws y dentro de ella la carpeta src.

Una vez que se tiene el espacio de trabajo se cuenta con todas las herramientas para desarrollar la aplicación de búsqueda de personas.

"Un robot debe cumplir las órdenes dadas por los seres humanos, a excepción de aquellas que entrasen en conflicto con la primera ley"
Segunda ley de la robotica de Isaac Asimov.

En primer lugar en la carpeta src se crea el paquete de nombre robot con las dependencias necesarias para la aplicación, en este proyecto por ejemplo, una dependencia es rospy la cual es necesaria para poder ejecutar Python. Después de crear el paquete es necesario compilar.

Una vez que se realizan los pasos anteriores, se tiene el paquete robot con dos carpetas; include y src más los archivos, CMakeList.txt y Package.xml. Ahora dentro de la carpeta src del paquete robot se crean cuatro carpetas para separar los elementos del paquete de acuerdo a su clase, las carpetas son: urdf, scripts, launch y rviz. En la figura 3.5 se muestra la imagen del paquete robot completo.

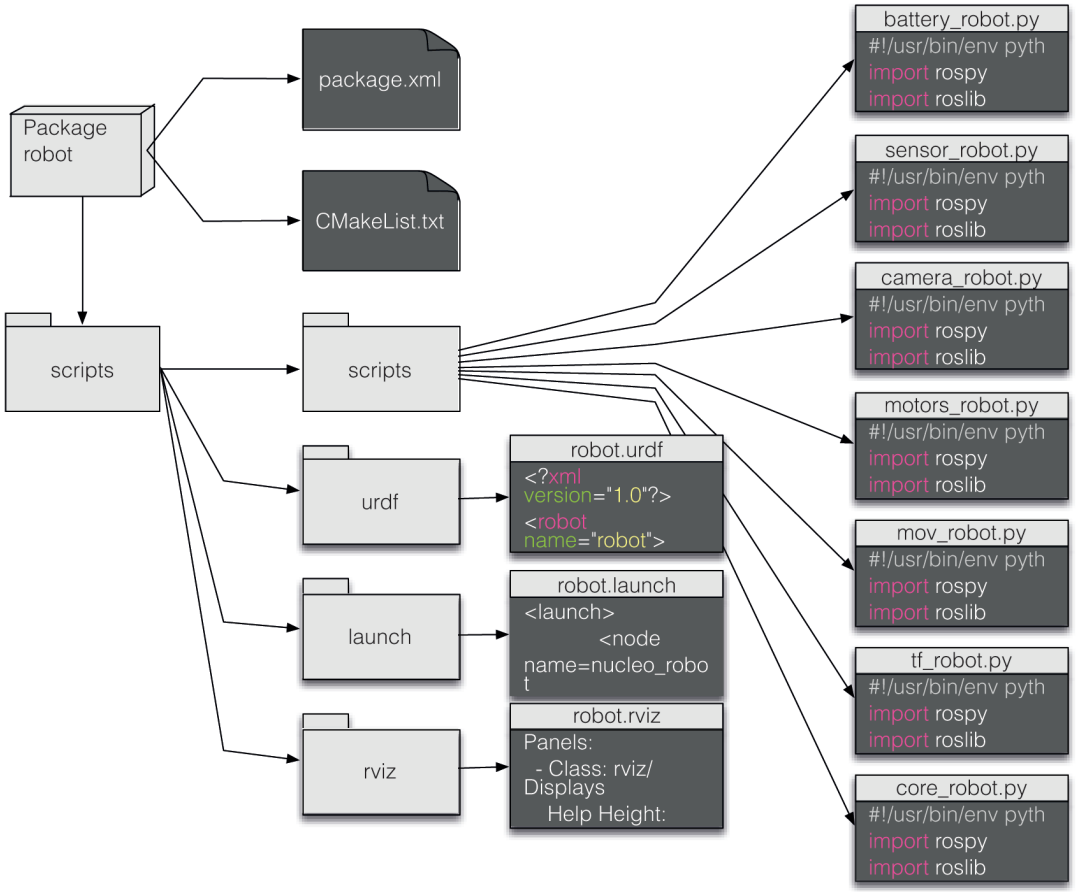

Figura 3.5 Paquete robot.

Los paquetes en ROS se conforman de varios nodos, los cuales realizan una función especifica, generan o reciben datos, pero antes es necesario ejecutar primero el núcleo de ROS, el cual

asigna números identificadores a cada nodo y topic.
Como se mencionó anteriormente, este es un proyecto que puede crecer cambiando o agregando componentes, como por ejemplo, diferentes sensores o motores con más potencia, para que sea práctico hacerlo se desarrolla un nodo para cada función, de esta manera si cambia algún componente del robot, sólo se actualizaría el nodo de interés.
Los nodos pueden ser programas escritos en Python, pero para que un programa escrito en Python pueda ser reconocido como un nodo dentro del entorno ROS, debe de contener las líneas que se describen a continuación, utilizando como ejemplo, el nodo *sensor_robot*, con el archivo *sensor_robot.py*.

```
1  #!/usr/bin/env python
```

La línea 1 indica que el código escrito se debe de ejecutar como un programa desarrollado en Python.
Después hay que importar la biblioteca Python para ROS llamada rospy, línea número 2.

```
2  import rospy
```

Además de importar también las bibliotecas definidas en ROS para los mensajes que se vayan a publicar o recibir, por ejemplo para recibir y enviar datos de sensores, línea número 7.

```
7   from sensor_msgs import Range
```

En la línea de código 31 se define el nombre del nodo y se inicia para que se identifique en el sistema ROS:

```
31  rospy.init_node("sensor_robot")
```

En la línea 13 se define la interfaz del nodo, en este caso su función es la de publicar. Los parámetros a definir según sea la necesidad son: el nombre del topic por el cual se envía o recibe la información, el tipo de mensaje y la cantidad de mensajes en cola que se envían o reciben:

```
13  pub_sensor=rospy.Publisher('robot_sensor_
    topic', Range, queue_size=1)
```

En el siguiente capítulo se describe ampliamente el funcionamiento de cada nodo.

Instalación de software ev3dev

El robot EV3 tiene un software propio para programar al robot, que funciona por medio de iconos, es un software básico, por lo que es necesario instalar en el EV3 un sistema operativo que soporte lenguajes de programación como Python y C++.

"Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o con la segunda ley"
Tercera ley de la robotica de Isaac Asimov.

Este sistema operativo es el ev3dev el cual se descarga de la pagina ev3dev.org, se instala en una tarjeta micro SD y se ejecuta desde la misma tarjeta por lo que no es necesario borrar el sistema operativo original. El sistema ev3dev tiene como base Debian Linux, por lo que existen varios paquetes de software libre que se pueden emplear, además se pueden usar una gran cantidad de dispositivos como: puertos USB, adaptadores IEEE 802.11, cámaras, teclados, joysticks, entre otros.
La computadora, el robot y la cámara se conectan entre sí de la siguiente manera: el robot se conecta al ruteador Arris TG862 de manera inalámbrica en modo infraestructura, la computadora también se conecta al ruteador a través del puerto de ethernet, para que la computadora con su interfaz inalámbrica se conecte a la cámara, teniendo así a todos los dispositivos conectados. La topología utilizada se muestra en la figura 3.6.

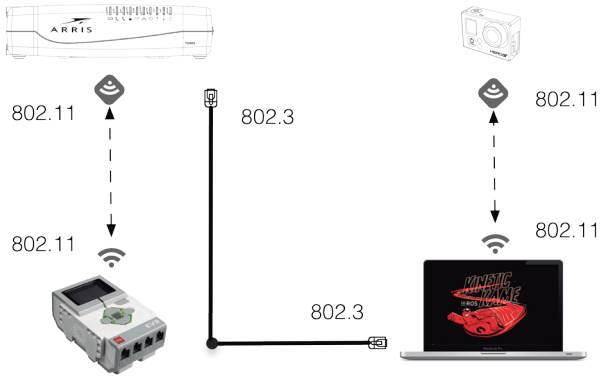

Figura 3.6. Topología utilizada para conectar los dispositivos en red.

Con todos los dispositivos en red y la dirección IP del robot, utilizando el interprete de instrucciones ssh, desde la computadora se accede al robot para configurar un usuario y contraseña en el robot.
La instalación del sistema operativo ev3dev incluye las bibliotecas de Python compatibles con el robot, sin embargo, es necesario instalar RPyC, la cual es una biblioteca para realizar procedimientos remotos, agrupamiento y computación distribuida. Mediante RPyC se ejecutan los programas como si estuvieran instalados en el robot, de esta manera el robot se convierte en servidor y la computadora en cliente. Es importante conectarse al robot antes de ejecutar ROS e iniciar RPyC

Instalación de software para el procesamiento de imagen

Para realizar el procesamiento de la imagen para la detección de rostros, se requieren dos elementos importantes: la biblioteca de Python goprohero y OpenCV. La biblioteca goprohero permite controlar la cámara de manera inalámbrica desde la computadora. Utilizando las bibliotecas de OpenCV se reciben las imágenes de la cámara para realizar el procesamiento de imagen para la detección de rostros. Es necesario que el archivo xml clasificador de rostros *haarcascade_frontalface_alt* se

encuentre en la carpeta scripts al llamar a este archivo en el nodo camera_robot.

04

Capítulo 4
Programación

La aplicación de búsqueda de personas está formada por un conjunto de nodos, los cuales realizan una función especifica como: adquisición de datos, cálculo de desplazamiento, toma autónoma de decisiones, sistema de referencias de ubicación y desplazamiento. Además de proporcionar una interfaz visual para poder ver el funcionamiento de la aplicación que permite ver las imágenes recibidas por la cámara y también modelar gráficamente al robot y su desplazamiento en tiempo real.

Se explica en primer lugar el funcionamiento de los nodos de adquisición de datos: *battery_robot*, *camera_robot* y *sensor_robot*; después los nodos generadores de datos y salidas: *motors_robot*, *mov_robot* y *core_robot*; el nodo que proporciona el sistema de referencias para la aplicación: *tf_robot;* el archivo que modela gráficamente los elementos de hardware de la aplicación robot: robot.urdf; el archivo que guarda las configuraciones y preferencias del nodo para la visualización: *robot.rviz*; por último, el archivo *robot.launch*, el cual ejecuta todos los nodos mencionados. El conjunto de los nodos del paquete *robot* y su interacción se muestran en la figura 4.1.

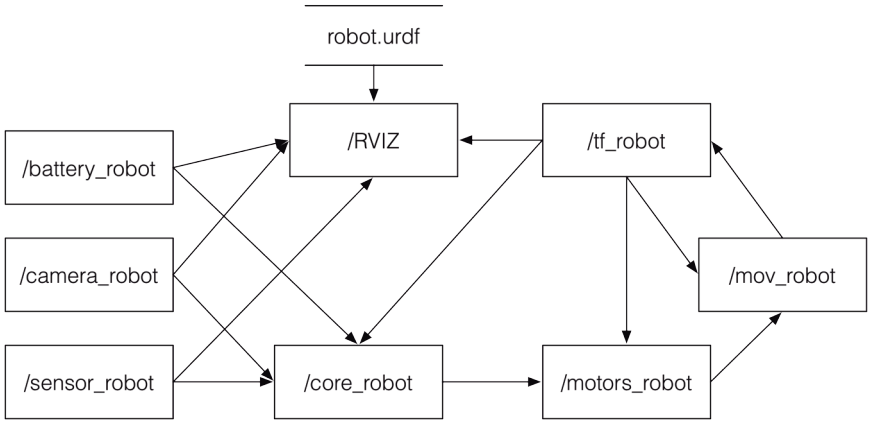

Figura 4.1. Nodos del paquete *robot*.

Nodo battery_robot

La función de este nodo es la de informar al nodo *core_robot* el estado actual de la batería del robot. Al conocer el estado de la batería, el nodo *core_robot* toma la decisión de iniciar la aplicación de búsqueda de personas o no. Cuando la aplicación de búsqueda de personas ya se está ejecutando y la carga de la batería disminuye, el nodo *core_robot* toma la decisión de regresar al punto de partida para no quedar en algún punto durante la exploración.

El nodo *battery_robot* envía la información de la batería utilizando los mensajes *BatteryState* de la categoría *sensor_msgs*

a través del topic *battery_topic*. La tabla 3.1 muestra los campos del mensaje *BatteryState*.

sensor_msgs/BatteryState.msg	
voltage	Voltaje expresado en volts
current	Corriente, se expresa en amperes y en valores negativos cuando se está descargando
charge	Carga actual
capacity	Capacidad de carga
percentage	Porcentaje de carga expresado de 0 - 1
power_supply_status	Desconocido=0 Cargando=1 Descargando=2 Sin carga=3 Cargada=4
power_supply_healt	Desconocido = 0 Buen estado de la batería= 1 Calentamiento = 2 Inservible = 3 Sobrecarga = 4 Falla = 5 Fría = 6 Expirada = 7 Próxima a expirar = 8
power_supply_technology	Desconocido = 0 NIMH = 1 LION = 2 LIPO = 3 LIFE = 4 NICD = 5 LIMN = 6
bool_present	Indica si está presente la batería
string_location	Indica en qué slot está ubicada la batería si existe más de un slot
string_serial_number	Número de serie de la batería

Tabla 4.1. Campos del mensaje *BatteryState*.

El funcionamiento del nodo es el siguiente: una vez que se establece la conexión con el robot se configura el intervalo de tiempo al cual va a estar informando el estado de la batería, para este proyecto el intervalo de tiempo elegido es de un segundo, se leen los datos del estado de la batería y se envían al nodo *core_robot* mientras el nodo esté activo. En la figura 4.2 se muestra el diagrama de flujo del funcionamiento que se acaba de describir del nodo *battery_robot*.

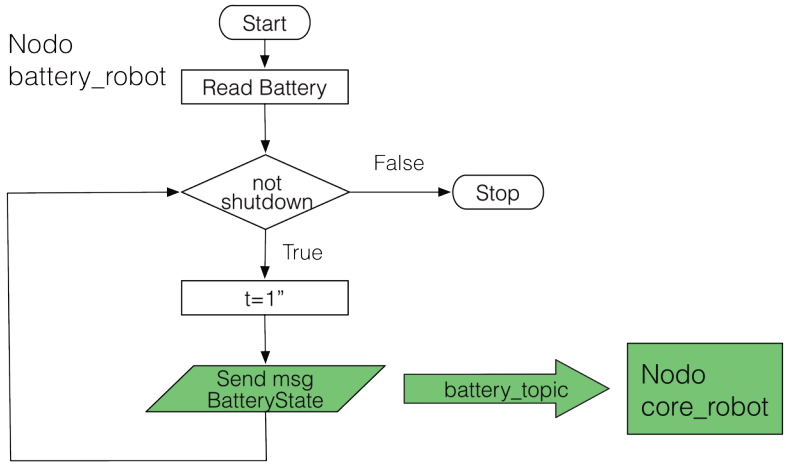

Figura 4.2. Diagrama de flujo del nodo *battery_robot*.

Nodo camera_robot

Este nodo realiza la tarea de la detección de rostros, para ello establece una conexión inalámbrica entre la cámara y la computadora, una vez que se establece la comunicación los siguientes pasos son:

- Se realiza una visualización previa de lo que capta la cámara
- Y se inicia la captura de imágenes en el nodo de manera inalámbrica.

Mientras la captura de imágenes en el nodo se encuentre activa, se realiza el procesamiento de imágenes, este procesamiento consiste en las siguientes etapas.

- Dado que la cámara que se utiliza en este proyecto genera imágenes a color, se convierte la imagen a color en blanco y negro, para que el proceso de detección de rostros se haga sobre una sola matriz y no en tres matrices que conforman una imagen a color.
- Se llama al clasificador de rostros. En el caso de detectar un rostro se dibuja un recuadro y se añade a la captura de imágenes. En esta etapa se manda un mensaje estándar del tipo *Bool* al nodo *core_robot*, para indicar si se detecta o no

a una persona, lo anterior a través del topic *human_topic*. El formato de los mensajes estándar del tipo *Bool* se muestran en la tabla 4.2.

std_msgs/Bool Message	
bool.data	True ó False

Tabla 4.2. Campo mensaje del tipo *Bool*.

Recordando que en el proyecto se está utilizando la biblioteca OpenCV para recibir las imágenes y hacer el procesamiento para la detección de rostros, es necesario convertir estas imágenes al formato que utiliza ROS en el mensaje del tipo *Image*. En la tabla 3.3 se muestran los campos del mensaje *Image* de la categoría *sensor_msgs*.

sensor_msgs/Image.msg	
height	Número de renglones que forma la imagen
width	Número de columnas que forman la imagen
string encoding	Codificación de los pixeles
is_bigendian	Tipo de ordenamiento de acuerdo al byte más o menos significativo
step	Longitud de fila completa en bytes
data	Tamaño de la matriz en step y renglones

Tabla 4.3. Campos del mensaje *Image* de la categoría *sensor_msgs*.

Para convertir las imágenes recibidas por OpenCV al formato que utiliza ROS, se emplea la biblioteca *cv_bridge*. En la figura 4.3 se ilustra la función de *cv_bridge*.

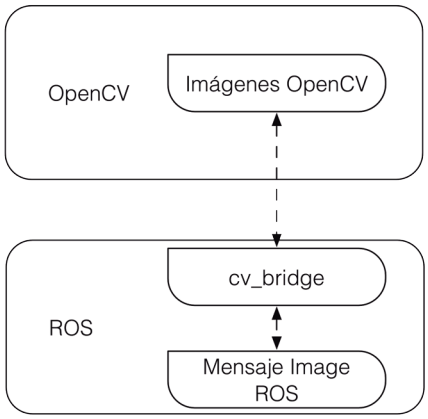

Figura 4.3. Conversión de imágenes del formato OpenCV a formato ROS utilizando *cv_bridge*.

Independientemente de que se detecte o no a una persona, las imágenes se envían al nodo RVIZ a través del topic *image_topic*. Este proceso continua mientras la captura de imágenes esté activa. El diagrama de flujo del funcionamiento descrito de este nodo se muestra en la figura 4.4.

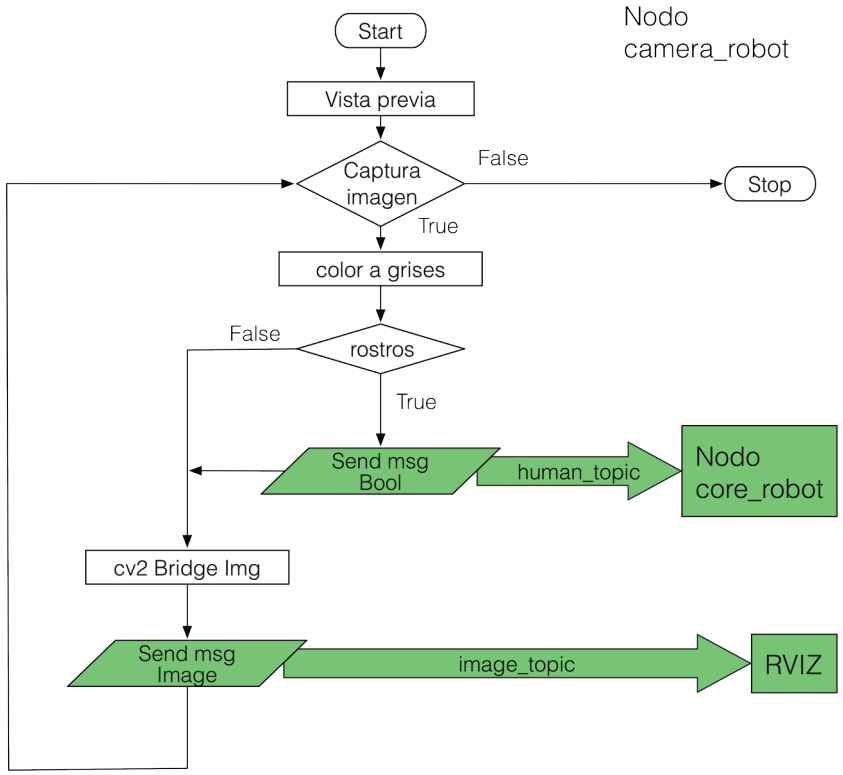

Figura 4.4. Diagrama de flujo del nodo *camera_robot*.

Nodo sensor_robot

Este nodo realiza la función de detectar obstáculos con el sensor infrarrojo del robot. Informa al nodo *core_robot* el estado del sensor para que este nodo pueda decidir si debe de avanzar o debe de girar, para evadir el obstáculo. El mensaje que envía

es del tipo *Range* de la categoría *sensor_msgs* a través del topic *sensor_topic*. En la tabla 4.4 se muestran los campos que contiene este mensaje.

sensor_msgs/Range.msg	
radiation_type	Se marca con 0 ó 1, para sensor ultrasónico y para sensor infrarrojo respectivamente
field_of_view	Define la apertura del cono de radiación
min_range	La distancia mínima de funcionamiento del sensor
max_range	La distancia máxima a la que detecta un objeto
range	Distancia a la que detecta un objeto el sensor infrarrojo
frame_id	Indica en dónde se encuentra representado el sensor infrarrojo en el archivo URDF

Tabla 4.4. Campos del mensaje *Range*.

El nodo *sensor_robot* funciona de la siguiente manera: realiza la conexión con el robot, verifica que el sensor se encuentre conectado al puerto de entrada del robot y después lee los datos que genera el sensor infrarrojo. Se debe de configurar el tiempo al cual el nodo *sensor_robot* va a estar informando el estado del sensor infrarrojo, para este propósito se establece un intervalo de tiempo de cuatro veces por segundo para enviar el mensaje al nodo *core_robot*. Esta tarea se realiza mientras el nodo se encuentre activo. En la figura 4.5 se muestra el diagrama de flujo del funcionamiento descrito de este nodo.

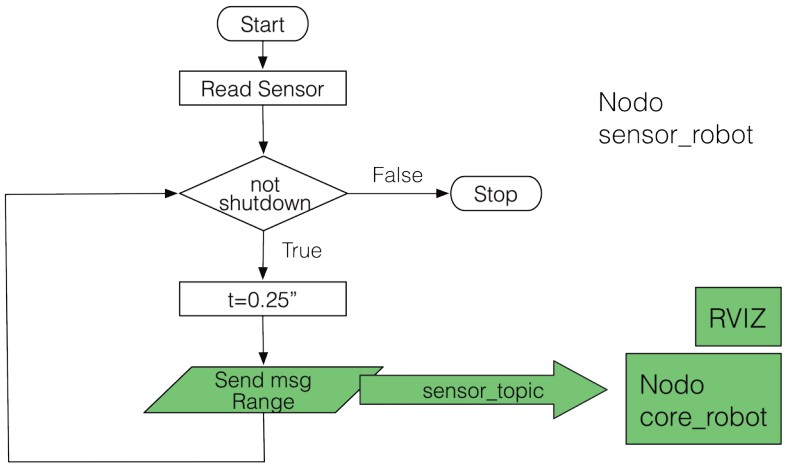

Figura 4.5. Diagrama de flujo del nodo *sensor_robot*.

Nodo motors_robot

El movimiento de los motores y el cálculo para regresar al origen se realiza en este nodo. Al iniciar el nodo se realiza la conexión con el robot y revisa que los motores estén conectados a los puertos de salida del robot. Este nodo recibe mensajes de dos nodos, el nodo *tf_robot* y el nodo *core_robot*. Del nodo *tf_robot* recibe la posición del robot para que con este dato pueda calcular el regreso al punto de partida. Por otro lado, del nodo *core_robot* recibe las instrucciones de: avanzar, retroceder, girar o alinear hacia el punto de partida. Los mensajes que recibe son del tipo *Pose* de la categoría *geometry_msgs* y también mensajes

estándar del tipo *Int16*. Estos mensajes se reciben a través de los topics *pose_topic* y *motors_msg_topic*, respectivamente. En las tablas 3.5 y 3.6 se muestran los campos de estos mensajes.

geometry_msgs/Pose Message	
Campo	Descripción del campo
Point position	Representación de la posición en el espacio compuesta por posición y orientación.
Quaternion orientation	

Tabla 4.5. Campos del mensaje del tipo *Pose*.

std_msgs/Int16 Message	
Campo	Descripción del campo
Int16 data	Entero 16 bits

Tabla 4.6. Campos del mensaje del tipo *Int16*.

Este nodo además de recibir datos de los nodos *tf_robot* y *core_robot* envía mensajes al nodo *mov_robot* y al nodo *core_robot*, que corresponden a la rotación de los motores y al aviso de que los motores se detuvieron, respectivamente. Para ello, se utilizan las componentes angulares X y Y para enviar la rotación de los motores y la componente angular Z para indicar que los motores se detuvieron con la bandera *stop*. Los mensajes se envían a través del topic *motors_topic*, utilizando mensajes del tipo *Twist* de la categoría *geometry_msgs*. En la tabla 4.7 se muestran los campos del mensaje *Twist*.

geometry_msgs/Twist Message	
Campo	Descripción del campo
Vector 3 lineal	Expresa la velocidad en el espacio en sus partes lineales y angulares
Vector 3 Angular	

Tabla 4.7. Campos del mensaje del tipo *Twist*.

El nodo recibe la posición que en un principio corresponde a las coordenadas (0, 0) y después se actualizan de acuerdo al desplazamiento del robot. Las coordenadas de la posición del robot se reciben del nodo *tf_robot*. Del nodo *core_robot* se reciben las siguientes instrucciones: avanzar, retroceder, girar y alinear. Al recibir la instrucción de avanzar, se arrancan los dos motores con el mismo sentido de rotación positivo, con cada grado de rotación de los motores se genera un mensaje que es enviado al nodo *mov_motors*, para que calcule el desplazamiento y lo almacene en el nodo *tf_robot* al recibir la bandera *stop*. Cuando se recibe la instrucción retroceder, los motores giran con el mismo sentido de rotación, pero con signo negativo. Con la instrucción girar, un motor rota en sentido positivo mientras que el otro motor gira en sentido contrario. Por último, cuando se recibe la instrucción alinear, se calcula la dirección al punto de partida y se mueven los motores hasta que el robot apunte en esa dirección. En la figura 4.6 se muestra el diagrama de flujo del nodo *motors_robot* correspondiente a lo antes descrito.

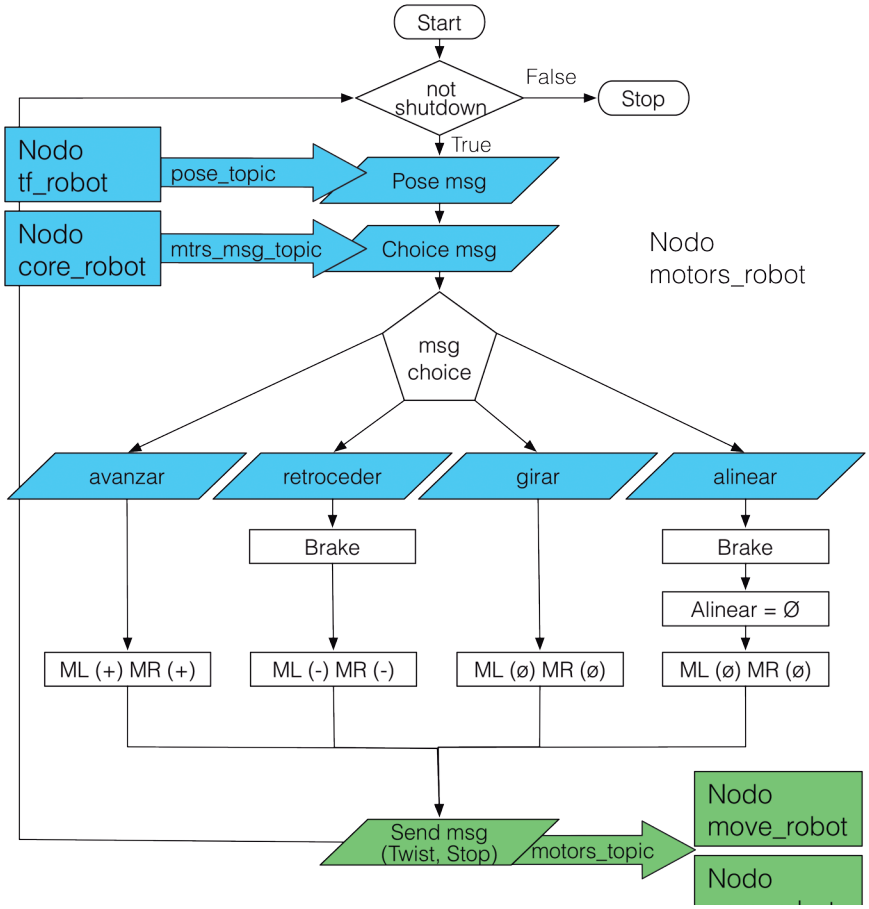

Figura 4.6. Diagrama de flujo del nodo *motors_robot*.

Nodo mov_robot

Este nodo tiene como entrada los mensajes del nodo *motors_robot* con el movimiento de rotación de los motores a través del topic *motors_topic* y los mensajes de posición del nodo *tf_robot* que son recibidos por el topic *pose_topic*. Los campos de estos mensajes ya se explicaron en las tablas 4.7 y 4.5. La función del nodo *mov_robot* es la de calcular el movimiento lineal del robot y su dirección con base en el número de giros de los motores, e ir actualizando la posición del robot conforme se va desplazando. Para ello, recibe la posición inicial del robot del nodo *tf_robot* y suma el desplazamiento actual del robot, para después enviarlo al mismo nodo *tf_robot* utilizando mensajes del tipo *Twist*, explicados en la tabla 4.7 y enviados a través del topic *move_topic*. Se recalca que aunque envía y recibe datos del mismo nodo *tf_robot* la información es diferente: recibe la posición actual del robot y envía el desplazamiento actual del robot. En la figura 4.7 se muestra el diagrama de flujo del nodo *mov_robot* descrito.

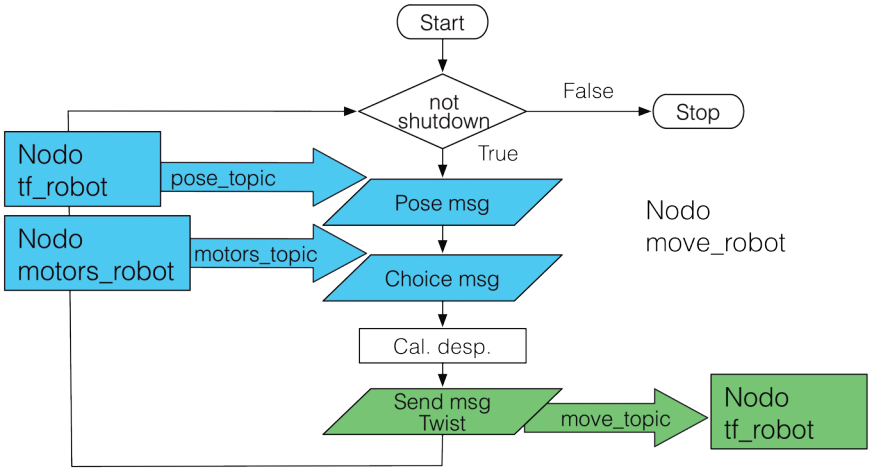

Figura 4.7. Diagrama de flujo del nodo *mov_robot*.

Nodo tf_robot

La función principal de este nodo es proveer un sistema de referencias, para proveer al robot la capacidad de conocer su ubicación en el espacio. Estas referencias servirán también para representar al robot de manera gráfica en un plano de tres dimensiones en conjunto con el archivo URDF, utilizando la herramienta de visualización RVIZ. Por ejemplo, el robot físicamente se desplaza en el área que se va a explorar, marcando como punto de partida la posición en la que inicia, este punto tendrá las coordenadas (0,0,0). Para modelar el movimiento que tiene el robot físicamente se requieren tener referencias

de desplazamiento, en este proyecto la primer referencia se llama *Start* y los cálculos de desplazamiento del robot se hacen con base en esta referencia, es decir, el robot se mueve a una distancia X de la referencia *Start*. De la misma manera, las piezas que conforman al robot están ubicadas a una distancia X de la referencia de estas partes. Por ejemplo, los motores están ubicados lateralmente a 8.5 cm de distancia del centro geométrico del Bloque EV3, en este caso la referencia de los motores es el Bloque EV3. De manera similar, las ruedas están unidas a los motores del robot, siendo la referencia para las ruedas el centro geométrico de los motores del robot. Entonces el nodo *tf_robot* se encarga de transmitir el desplazamiento de todas las piezas con sus respectivas referencias para modelar el movimiento del robot en conjunto con el archivo URDF y RVIZ. La transmisión de las piezas y sus referencias se realiza a través del topic TF.

Además de la transmisión de las piezas y las respectivas referencias este nodo también transmite la posición actualizada del robot al nodo *mov_robot* utilizando los mensajes del tipo *Pose* descritos en la tabla 4.5 utilizando el topic *pose_topic*. Este nodo en conjunto con el nodo *mov_robot* permiten mantener actualizada la posición del robot. En la figura 4.8 se muestra el diagrama de flujo del funcionamiento de este nodo.

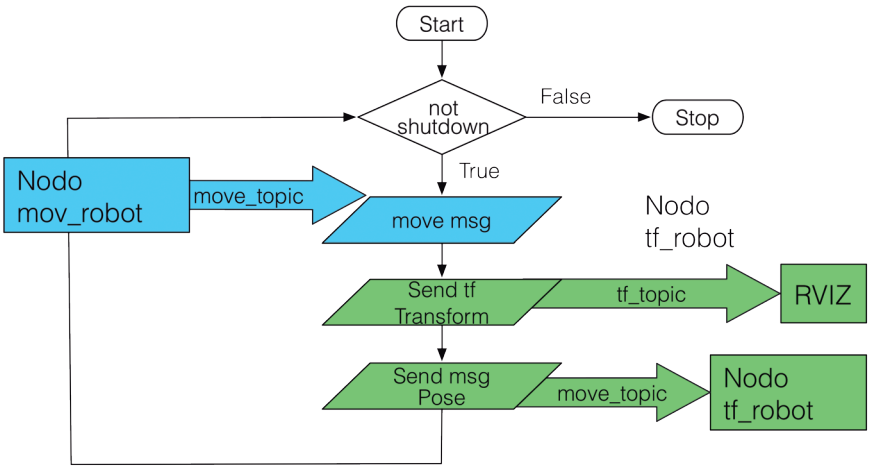

Figura 4.8. Diagrama de flujo nodo *tf_robot*.

Nodo core_robot

Por último, el nodo *core_robot* es el nodo que toma las decisiones con base en la informacion que recibe de los nodos: *battery_robot, camera_robot, sensor_robot* y *tf_robot*, después envía las siguientes opciones de decisiones al nodo *motors_robot*:

- Avanzar. Indica que mueva los motores hacia adelante.
- Retroceder. Instrucción para que el robot retroceda.
- Girar. Con esta instrucción el robot gira de forma aleatoria.
- Alinear. Instrucción para que el robot se coloque en dirección al punto de partida.

Las características tanto de los mensajes como de los topics utilizados en este nodo ya se describieron anteriormente. El funcionamiento de este nodo es el siguiente:

- Recibe los mensajes de los nodos de adquisición de datos y con base en esa información realiza diferentes acciones.
- Mensaje del nodo *batterry_robot*: cuando el mensaje recibido indica que no hay voltaje suficiente para iniciar o continuar con la aplicación de búsqueda de personas, coloca la bandera *Home=True*.
- Mensajes del nodo *camera_robot*: si el mensaje indica que encuentra a una persona, coloca la bandera *Home=True* e informa en qué coordenadas se localizó a la persona.
- Se coloca la bandera *Home* inicialmente en *False*. La bandera *Home=False* indica que el robot puede iniciar o continuar la exploración, si la bandera es *Home=True* el robot se dirige al punto de partida.
- Mensaje del nodo *motors_robot*: cuando se envía una instrucción al nodo *motors_robot* los motores se ponen en movimiento, razón por la cual no se debe enviar otra instrucción hasta que los motores se detienen informando de ello a través del mensaje *stop*.
- Mensajes del nodo *sensor_robot*: indican si existe un obstáculo o no.

- Las coordenadas de referencia y ubicación durante la exploración las recibe en los mensajes del nodo *tf_robot*.
- Después de que se inició la recepción de los mensajes con la bandera *Home=False,* revisa el estado del sensor infrarrojo, cuando no hay obstáculo envía la instrucción avanzar al nodo *motors_robot* y si hay un obstáculo manda el mensaje retroceder, para después enviar el mensaje girar. En todos los casos después de mandar una instrucción a los motores espera el mensaje *stop* para poder enviar la instrucción correspondiente.
- Cuando la bandera *Home=True* se calcula la distancia y dirección al punto de partida, en esta etapa se coloca un valor de tolerancia, ésta indica a qué distancia se puede recuperar el robot después de que realiza la exploración. Si la distancia calculada al punto de partida es menor a la tolerancia, termina la aplicación. En caso contrario se envía el mensaje de alinear para dirigir el robot hacia el punto de partida. Después, se realiza el mismo procedimiento descrito en el punto anterior. El proceso se repite hasta que termina la aplicación.

En la figura 4.9 se muestra el funcionamiento descrito de este nodo usando un diagrama de estados, en el cual los estados considerados son: home, distancia, alinear, obstáculo, avanzar, retroceder, girar.

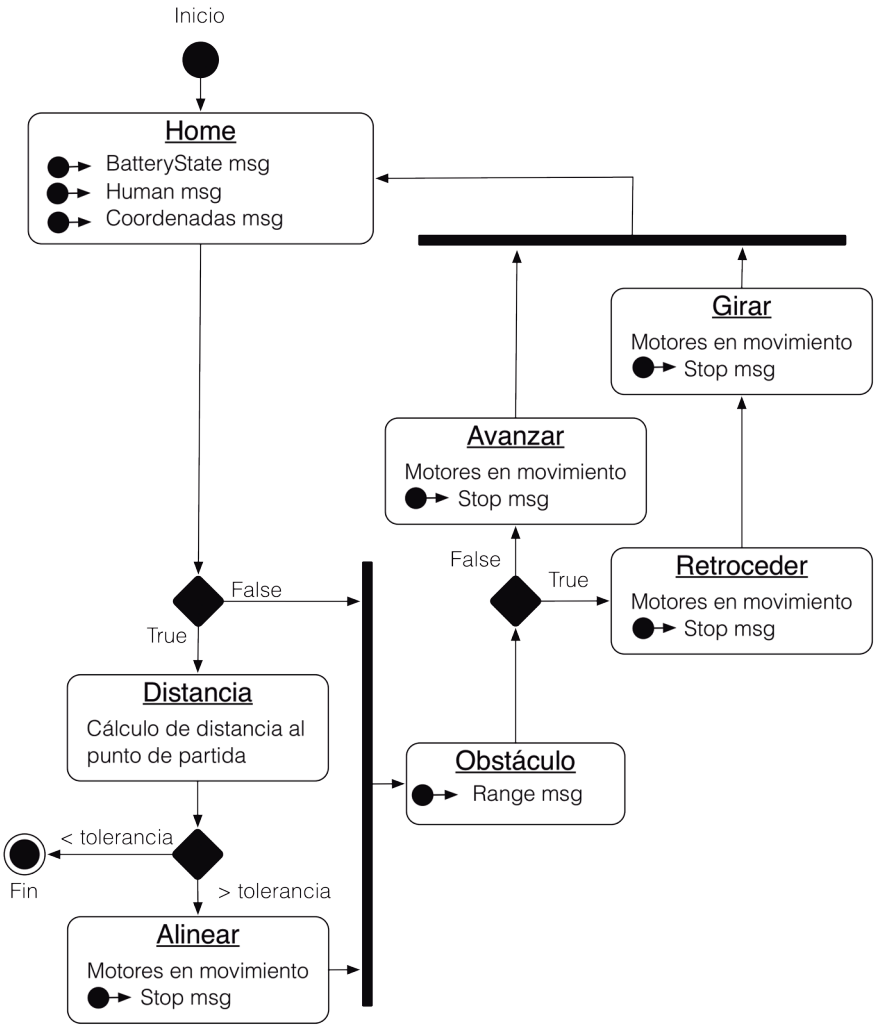

Figura 4.9. Diagrama de estados del nodo *core_robot*.

Aquí termina la parte correspondiente al funcionamiento de los nodos. Utilizando la herramienta de ROS *rqt*, se puede obtener un diagrama con todos los nodos activos y los topics a través de los cuales se comunican, ingresando el comando *rosrun rqt_graph.* En la figura 4.10 se muestran los nodos activos obtenidos con *rqt*.

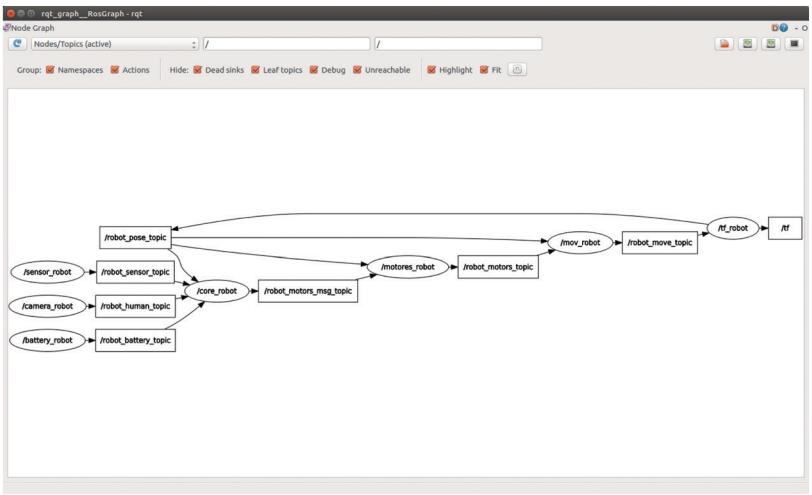

Figura 4.10. Nodos activos del paquete robot.

Archivo robot.urdf

No es un nodo, es un archivo que emplea la estructura XML con la descripción del robot en el formato de descripción de robot unificado (URDF, *Unified Robot Description Format*). La

descripción consiste en un conjunto de *links* y un conjunto de *joints*. En el entorno ROS los *links* describen las partes rígidas del modelo del robot y los *joints* describen las características de las uniones de estos elementos. En la figura 4.11 se muestran algunas líneas correspondientes al archivo robot.urdf.

```
1   <?xml version="1.0"?>
2   <robot name="robot">
.
.
.
10  <material name="robot_color_red">
11      <color rgba="1 0 0 1"/>
12  </material>
.
.
.
42  <link name="Robot">
43    <visual>
44      <geometry>
45          <box size="0.45 0.35 0.2"/>
46      < geometry>
47      <origin xyz="0 0 0"/>
48      <material name="robot_color_white"/>
49    </visual>
50  </link>
```

El color del robot está descrito en el formato RGBA (*Red, Green, Blue, Alpha*), la componente alpha representa la opacidad de los colores RGB, todas las componentes se representan con un número entre 0 y 1.

Las dimensiones del robot se encuentran a escala 1:5 tomando como referencia la unidad de las celdas del grid de RVIZ mostrado en la figura 3.23.

Figura 4.11. Líneas del archivo URDF escrito en XML.

Como se mencionó en el nodo *tf_robot*, para representar al

robot en un plano de tres dimensiones es necesario el trabajo en conjunto del nodo *tf_robot*, el cual proporciona el sistema de referencias, el archivo URDF que describe las características del robot y la herramienta de visualización RVIZ, que permite observar el funcionamiento de la aplicación. Por lo tanto, a continuación se explican algunas líneas del archivo robot.urdf (figura 4.11).

En la línea 1 se especifica la versión XML que se usa, la línea número 2 define el nombre del robot. Los colores se representan en el formato RGBA. Por conveniencia se define la paleta de colores usada para describir al robot al inicio del archivo, líneas 4-12. Las líneas que van de la 42 a la 50 describen el elemento link con el nombre Robot que corresponde al Bloque EV3, el cual es representado por una caja con medidas: largo, ancho y alto. Las medidas del robot se representan en metros con una escala 1:5. Y se utiliza el color definido con el nombre *robot_color_white*. Ubicado en las coordenadas X, Y, Z (0, 0, 0), que indica que se encuentra en esa posición respecto a las coordenadas indicadas en el nodo *tf_robot*. Por otro lado, es importante nombrar las piezas del robot con el mismo nombre que se usa en el sistema de referencias del nodo *tf_robot*. Dicho de otro modo, se le está dando forma a cada referencia del nodo *tf_robot*. En la figura 4.12 se muestran las líneas del código del nodo *tf_robot* que corresponden a la referencia principal del robot y las líneas

del archivo *robot.urdf* que describen la pieza que representa al Bloque EV3. Es muy importante usar los mismos nombres tanto en las referencias *tf* como en el archivo URDF.

Transmisión tf (nodo tf_robot)		Descripción URDF (archivo robot.urdf)	
48	br = tf.TransformBroadcaster()	42	<link name="Robot">
49	br.sendTransform((pose_robot.position.x, pose_robot.position.y, 0.03*s),	43	<visual>
50	tf.transformations.quaternion_from_euler(0, 0, pose_robot.orientatio.z),	44	<geometry>
51	rospy.Time.now(),	45	<box size="0.45 0.35 0.2"/>
52	'Robot',	56	</geometry>
53	'Start')	47	<origin xyz="0 0 0"/>
		48	<material name= robot_color_white"/>
		49	</visual>
		50	</link>

Figura 4.12. Transmisión de la referencia Robot (Python) y su descripción en el archivo URDF (XML).

El ejemplo anterior consiste en un elemento *link*, por ser la parte principal del robot, el cual tiene como referencia a *Start*, que representa el punto de partida de la exploración. Las partes del robot están unidas a éste, por lo que su referencia es el robot. En la figura 4.13 se muestran las líneas del código del nodo *tf_robot*, que corresponden a la transmisión de la pieza que representa al cilindro del motor del robot y su referencia, en este caso su

referencia es el brazo del motor izquierdo; también se muestran las líneas que describen esta pieza y la descripción de la unión con el brazo del motor izquierdo en el elemento *joint*.

Transmisión tf (nodo tf_robot)		Descripción URDF (archivo robot.urdf)	
257	br = tf.TransformBroadcaster()	188	<link name="Cilindro_motor_L">
258	br.sendTransform((0.0225*s, 0, -0.0025*s),	189	<visual>
259	tf.transformations.quaternion_from_euler(0, pose_robot.orientation.x, 0),	190	<geometry>
260	rospy.Time.now(),	191	<cylinder length="0.125" radius="0.05"/>
261	'Cilindro_motor_L',	192	</geometry>
262	'Brazo_motor_L')	193	<origin rpy="1.57 0 0" xyz="0 0 0"/>
		194	<material name="color_robot_rojo"/>
		195	</visual>
		196	</link>
		197	
		198	<joint name="Cilindro_motor_L_u_Brazo_robot_L" type="fixed">
		199	<child link="Cilindro_motor_L"/>
		200	<parent link="Brazo_motor_L"/>
		201	<origin xyz="0 0 0"/>
		202	</joint>

Figura 4.13. Transmisión "Cilindro_motor_L" (Python) y su descripción URDF (XML).

La descripción URDF del cilindro del motor izquierdo corresponde a un cilindro de color rojo de 0.125 de largo con un radio de 0.05, las medidas están expresadas en metros en escala 1:5. En la línea 193 de la descripción URDF se muestra además de las coordenadas en las cuales se encuentra colocada la pieza, aparece el elemento rpy que indica que el cilindro presenta una inclinación perpendicular sobre su eje X expresada en radianes.

El elemento *joint* de la descripción, URDF que va de la línea 198 a la 202 describe la unión del cilindro del motor izquierdo con el brazo del motor. La unión debe de tener nombre único en el archivo. Esta unión en particular se llama "*Cilindro_motor_L_u_Brazo_motor_L*" e indica que es una unión fija y que el *cilindro_motor_L* está unido al *Brazo_motor_L*, esto último aclarado con las etiquetas *child* y *parent* respectivamente. El archivo completo es representado con una estructura de árbol con los *links* y *joints* que describen al robot. En la figura 4.14 se muestra parte de la estructura de árbol de este archivo.

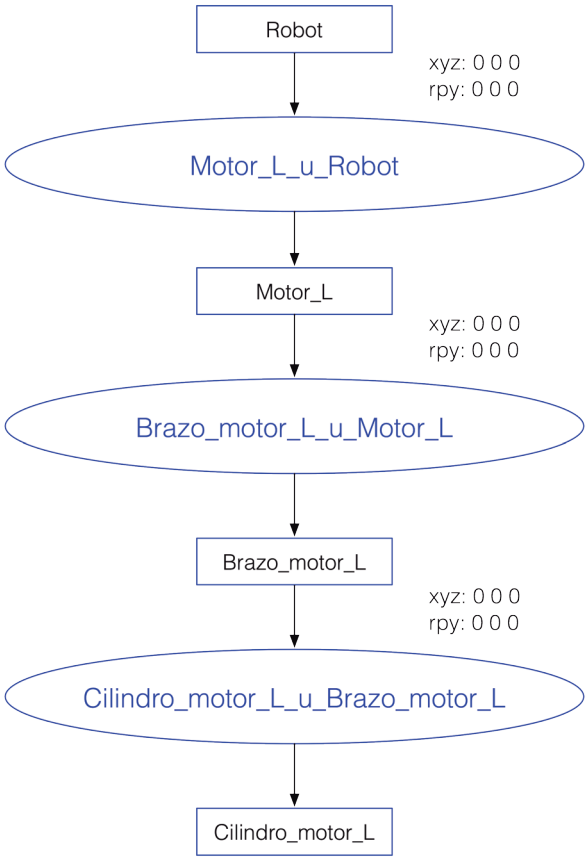

Figura 4.14. Parte de la estructura de árbol del archivo *robot.urdf*.

Se muestra únicamente la unión antes descrita dado que el árbol completo es muy grande y no se alcanza a ver con claridad en una imagen en una página. El número total de elementos que describen al robot en este proyecto es de 57

elementos descritos en 1,119 líneas.

Al final, en conjunto con las referencias *tf* y la descripción del robot URDF se tiene el modelo gráfico del robot. En la figura 4.15 se muestran a) las referencias principales del robot, b) el modelo gráfico del robot contenido en el archivo URDF y c) se muestra la fotografía del robot.

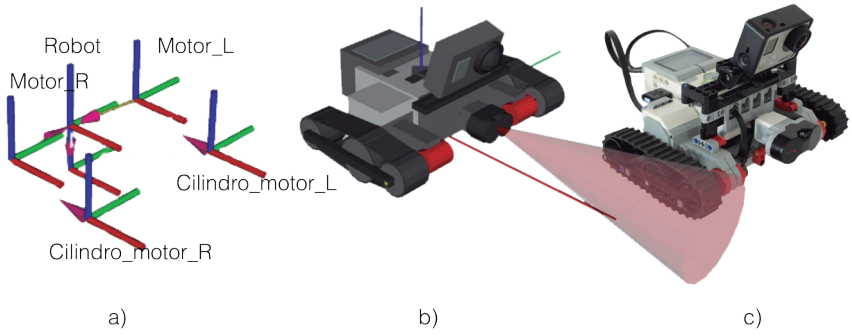

Figura 4.15. a) referencias *tf* del robot, b) modelo gráfico del robot y c) fotografía del robot.

Archivo robot.rviz

La herramienta de visualización de ROS es RVIZ. En las secciones anteriores se explicó la forma en la que se provee el sistema de referencias en el nodo *tf_robot* y la forma en la que se describe al robot en el archivo URDF. Entonces, para poder ver los elementos del robot en RVIZ, se agregan *displays*. Los *displays* son elementos

que dibujan algo en tres dimensiones, por ejemplo, el plano de desplazamiento del robot. En RVIZ el *display* que proporciona el plano tiene el nombre *Grid*, la cual tendrá en este proyecto la referencia *Start*. En la figura 4.16 se muestra el *display Grid*.

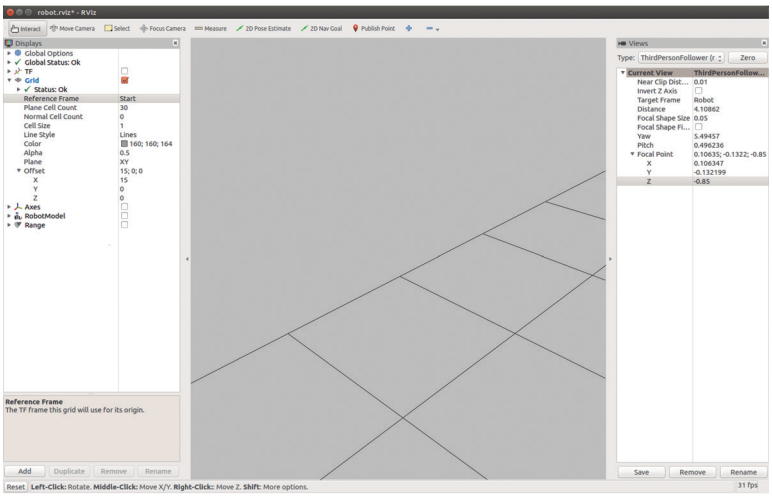

Figura 4.16. Plano generado por el *display Grid* con origen en la referencia *Start*.

En el panel izquierdo de la figura 4.16 se muestra: la información del display *Grid* en donde se muestra la referencia *Start*; el número de celdas que tiene, un offset (esto último debido a que el robot al inicio de la exploración no se coloca al centro del área por explorar), color, entre otros campos.

RVIZ cuenta con distintos tipos de *displays* para diferentes

propósitos. Los *displays* que se requieren para la aplicación de búsqueda de personas son las siguientes: para representar al sensor infrarrojo se requiere el *display* del tipo *Range*, para visualizar la imagen de la cámara GoPro se requiere un *display* del tipo *Image*, para ver las referencias del robot *tf*, se requiere el *display TF* y para ver el modelo del robot con base en las referencias *TF* y descrito en el archivo *robot.urdf*, se emplea el *display RobotModel*. Como se puede notar, todos los *displays* tienen nombres similares a los mensajes enviados por los nodos descritos a lo largo de este proyecto. Como se explicó antes, RVIZ también se suscribe a los nodos que generan la información que se acaba de listar. En la figura 4.17 se muestra la ventana RVIZ con todos los displays necesarios para poder visualizar la ejecución de la aplicación de búsqueda de personas. En la figura 4.18 se muestra la fotografía del robot realizando una prueba de ejecución de la aplicación.

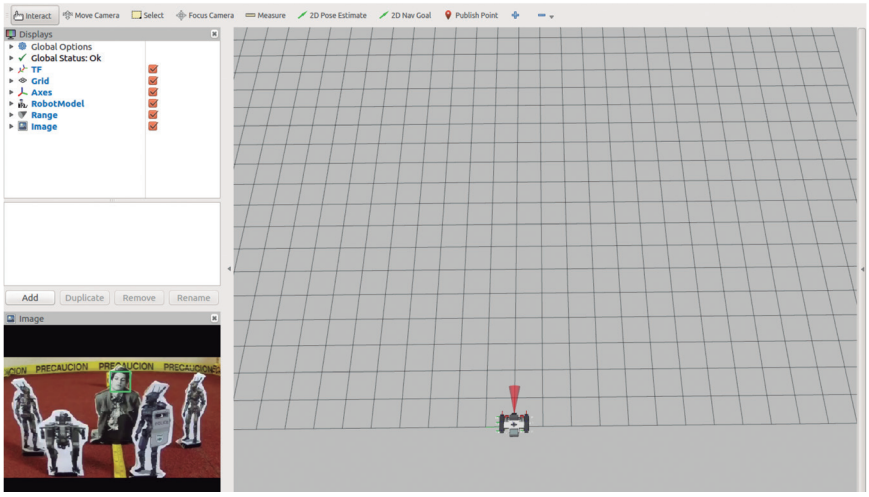

Figura 4.17. Aplicación de búsqueda de personas visualizada con RVIZ.

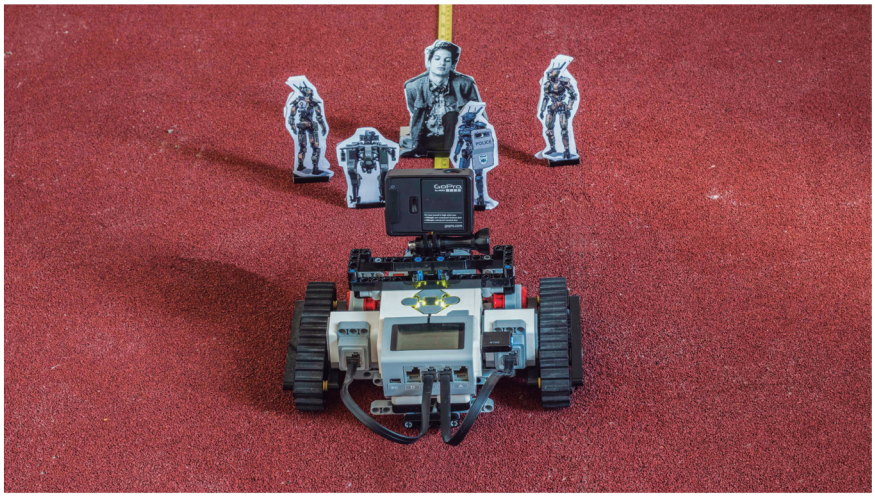

Figura 4.18. Fotografía del robot realizando una prueba de la aplicación de búsqueda de personas.

Como se acaba de explicar, al ejecutarse RVIZ se tienen que ir agregando los *displays* necesarios para poder ver la aplicación, en el archivo *robot.rviz* se guarda toda la configuración realizada en RVIZ. En la figura 4.19 se muestran parte de las líneas que contiene este archivo.

```
35  Visualization Manager:
36    Class: “”
37    Displays:
.
.
.
56      - Class: rviz/TF
57        Enabled: true
58        Frame Timeout: 15
59        Frames:
60          All Enabled: true
61          Start:
62            Value: true
63          robot:
64            Value: true
.
.
.
97    Views:
98      Current:
99        Class: rviz/ThirdPersonFollower
100       Distance: 5.05010176
101       Enable Stereo Rendering:
102         Stereo Eye Separation: 0.0599999987
```

Figura 4.19. Archivo robot.rviz.

Archivo robot.launch

Por último, el archivo *robot.launch* contiene la configuración de los nodos necesarios para ejecutar la aplicación de búsqueda de personas, el cual es utilizado con la herramienta de ROS llamada roslaunch. Este archivo inicia todos los nodos y archivos descritos anteriormente. Parte del código de este archivo se muestra en la figura 4.20.

```
3   <node
4     name="core_robot"
5     pkg="robot"
6     type="core_robot.py"
7     output="screen"
8     required="true"
9     launch-prefix="xterm -e"
10  />
.
.
.
68  <param name="robot"
69  textfile="/home/calakan/catkin_ws/src/robot/src/urdf/robot.urdf" />
```

Figura 4.20. Parte del archivo *robot.launch*.

Por ejemplo, en las líneas que van de la 3 a la 10, se indica que se iniciará el nodo core_robot perteneciente al paquete robot que está escrito en Python en el archivo *core_robot.py* y que se

requiere para la ejecución con los demás nodos. En las líneas 68 y 69 se indica la ruta para leer el archivo *robot.urdf*.

Como se puede observar cada parte de la aplicación funciona para una parte especifica utilizando las herramientas de ROS adecuadas para cada caso, se utilizan distintos tipos de mensajes de acuerdo a la información que se requiere enviar. En el siguiente capítulo se realizan las pruebas para poder ver el funcionamiento por etapas y después el funcionamiento de la aplicación completa.

05

Capítulo 5
Pruebas y resultados

Para conocer el desempeño de la aplicación de búsqueda de personas es necesario realizar pruebas a los nodos que la integran. Los datos de interés son: información constante del voltaje de la batería del robot, el alcance del sensor infrarrojo, saber si logra o no detectar a una persona, la capacidad del robot para evadir obstáculos, explorar un área y regresar al punto de partida de manera autónoma. Las pruebas consisten en medir la función de cada nodo y la interacción entre ellos, comenzando por el nodo *battery_robot*, ya que es el nodo que proporciona la información del estado de la batería, después, los nodos de adquisición de datos: *sensor_robot* y *camera_robot*, el nodo de ejecución: *motors_robot* y por último el nodo *core_robot* que integra a todos los nodos.

Las pruebas consisten en: verificar el enlace inalámbrico entre la computadora, el robot y la cámara; que los nodos funcionen de manera adecuada realizando pruebas generales para corroborar que los nodos se encuentran activos e intercambiando información entre ellos. Después, se analiza la lectura y ejecución del hardware que utilizan. Las pruebas de lectura y ejecución se realizan de forma controlada de acuerdo a la función que realiza cada nodo con fuente de alimentación constante. Después se integran los nodos por etapas y por último, se prueba el funcionamiento de la aplicación bajo las siguientes condiciones:

- Superficie lisa con obstáculo
- Superficie lisa con obstáculo y una fotografía

Pruebas de enlace de los nodos

Los nodos: *battery_robot*, *sensor_robot* y *camera_robot* realizan el enlace inalámbrico con el robot y la cámara para obtener información, así como también el nodo *motors_robot* realiza el enlace para realizar la tarea de mover los motores. Para que los equipos puedan comunicarse deben estar conectados entre sí, lo cual se realiza de la siguiente manera: la cámara y el *router* realizan la función de punto de acceso, la computadora se conecta a estos dos puntos de acceso a través de sus puertos 802.11 y Ethernet, respectivamente. El robot por su parte, se conecta al router utilizando el adaptador 802.11n. El *router* asigna direcciones IP al robot y a la computadora, por su parte, la cámara también asigna una dirección IP a la computadora. Para verificar cuáles equipos se encuentran conectados a los puertos de la computadora, además de conocer las direcciones IP, se utiliza el comando *arp* (*Addres Resolution Protocol*). Después de ejecutar el comando se obtienen las direcciones IP de los equipos conectados a la computadora y las interfaces a las cuales se encuentran conectados. En la figura 5.1 se muestran los resultados.

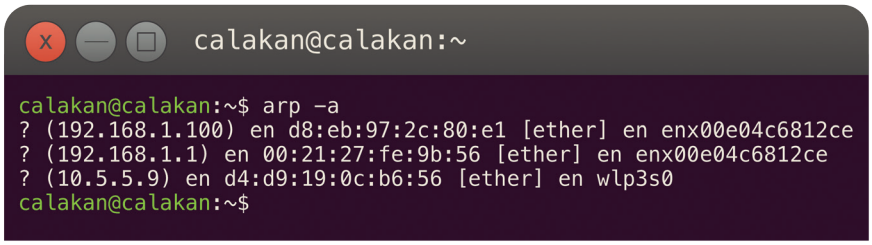

Figura 5.1. Lista de direcciones IP conectadas a la computadora, después de ejecutar el comando arp -a.

De la información mostrada en la figura 5.1, se puede observar que se encuentra un equipo conectado al puerto identificado con el código wlp3s0 y dos equipos conectados al puerto identificado con el código enx00e04c6812ce. La cámara es el único equipo que se encuentra conectado a la interfaz inalámbrica por lo que la dirección IP 10.5.5.9 le corresponde, por otro lado, se sabe que el robot se conecta de manera inalámbrica al router y la computadora al router a través del puerto Ethernet por que se encuentran a un salto de distancia. Para verificar que la dirección IP 192.168.1.100 le corresponde al robot, se ejecuta el comando traceroute el cual muestra a cuántos saltos se encuentran conectados los equipos, además de ejecutar el comando ifconfig para conocer las direcciones IP de las interfaces de la computadora, ya que tanto el *router* como la cámara le asignan una dirección IP, en la figura 5.2 se puede ver que el equipo conectado con esta dirección se encuentra a

un salto de distancia. También se observan las direcciones IP de las interfaces de la computadora, lo que demuestra que la computadora se encuentra conectada a dos redes.

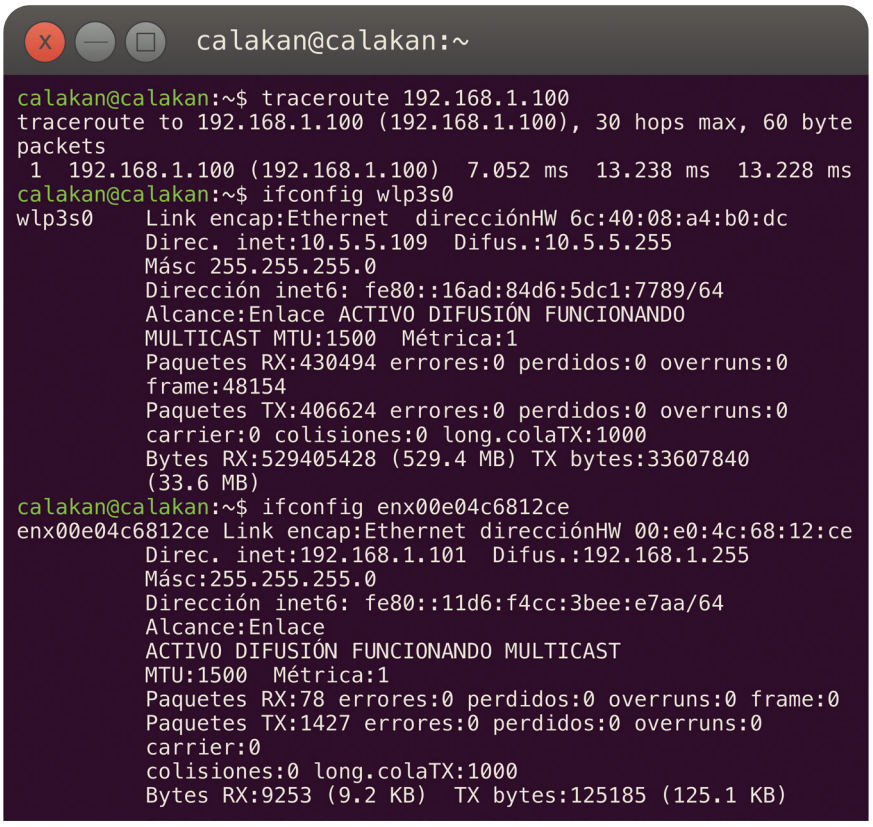

Figura 5.2. Ejecución del comando traceroute e ifconfig.

En la figura 5.3 se muestran los equipos con la dirección IP que les corresponden.

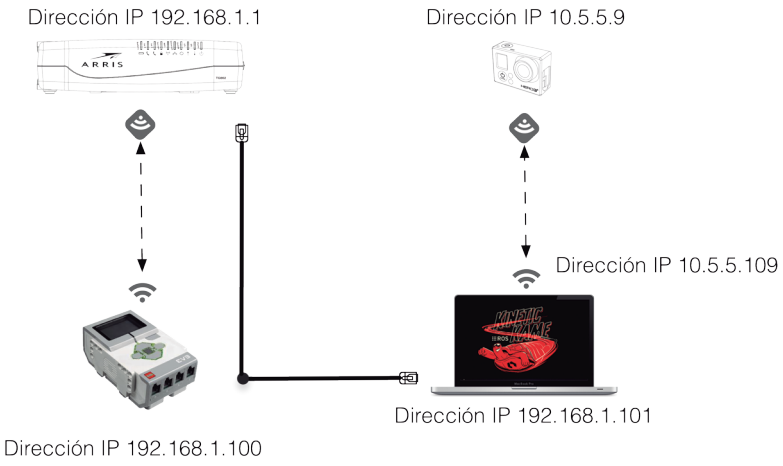

Figura 5.3. Direcciones IP de los equipos utilizados en la aplicación de búsqueda de personas.

Conociendo las direcciones IP del robot y la cámara se pueden realizar los enlaces. En el caso del nodo *camera_robot* no es necesario realizar ningún paso adicional, ya que para iniciar la captura de imagen de la cámara se requiere solamente de su dirección IP. Por otro lado, para realizar el enlace con el robot se utiliza el comando ssh y su dirección IP, de esta manera la computadora realiza la conexión al servidor (en este caso el robot), para que ejecute las tareas de los nodos en el robot. Después se inicia como servidor al robot ejecutando *rpyc_classic.py*. En la figura 5.4 se muestra el enlace con el robot.

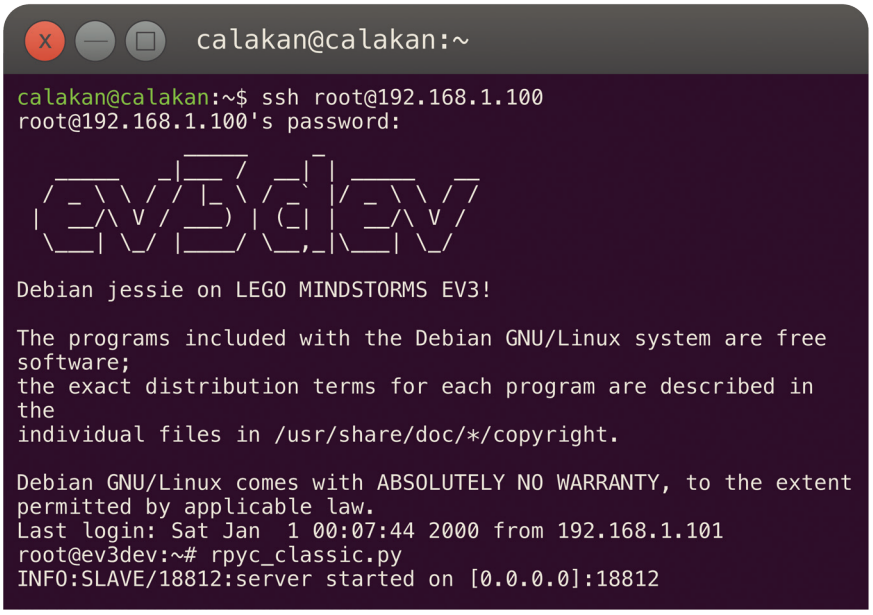

Figura 5.4. Enlace con el robot utilizando su dirección IP.

Una vez que se realizan los enlaces, de la cámara y el robot, se ejecutan los nodos para ver la información que generan, en qué formato la envían, a cuál nodo la envían y a través de cuáles topics la envían.

Prueba del nodo battery_robot

En primer lugar se analiza al nodo *battery_robot*, al ejecutar este nodo se muestra la información de la batería del robot.

En la figura 5.5 se observan los datos que lee el nodo, siendo de interés especial la variación del voltaje. En esta prueba se cambia el voltaje en la fuente para simular la disminución del voltaje en la batería.

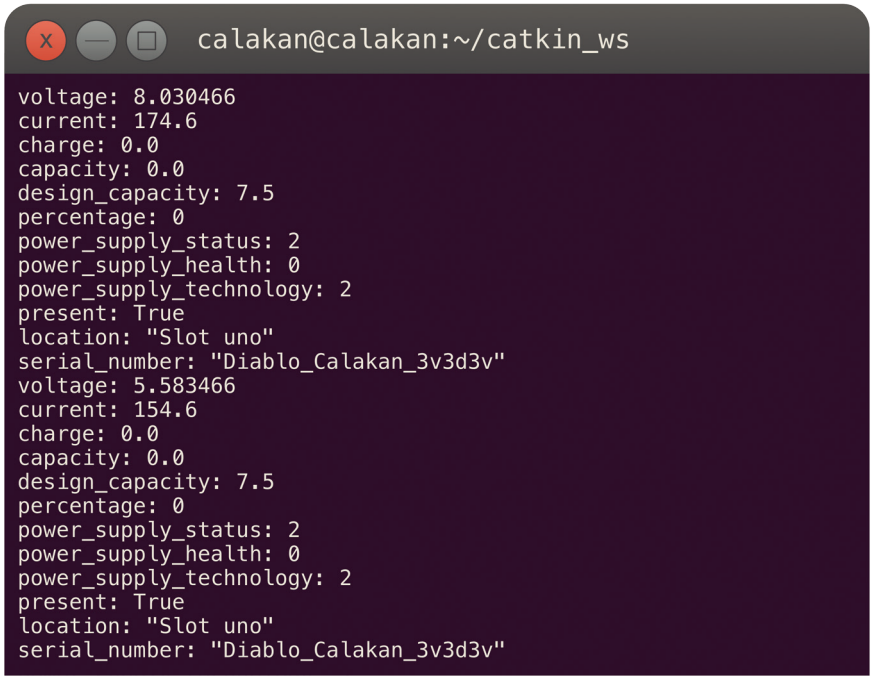

Figura 5.5. Ejecución del nodo *battery_robot* con información del estado de la batería.

Mientras el nodo está activo se puede comprobar que el nodo está mandando los mensajes correctamente al nodo *core_robot*, revisando el topic a través del cual se envían estos mensajes.

Para ello, se emplea el comando rostopic list para obtener una lista de los topics activos. Después se analiza la información del topic de interés utilizando el comando rostopic info y el nombre del topic de interés. La información que se obtiene después de ejecutar los comandos anteriores se muestra en la figura 5.6.

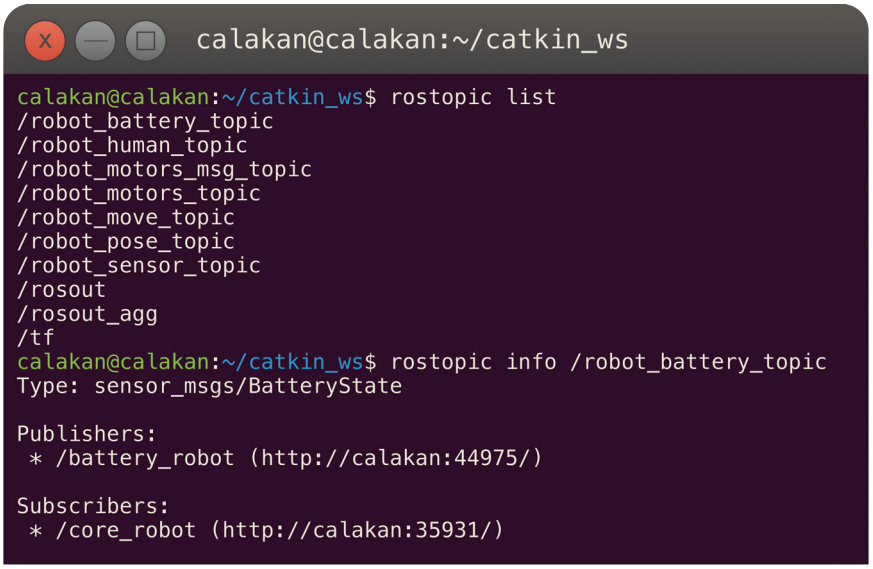

Figura 5.6. Información del topic *battery_topic*.

La información mostrada la figura 5.6 indica que transporta mensajes tipo *sensor_msgs/BatteryState*, los cuales son generados por el nodo *battery_robot* y recibidos por el nodo *core_robot*.

Prueba del nodo sensor_robot

De manera similar, el nodo *sensor_robot* lee la información referente al estado del sensor infrarrojo. En la figura 5.7 se muestra la información que se obtiene al ejecutar el nodo y al colocar un objeto a 20 centímetros de distancia.

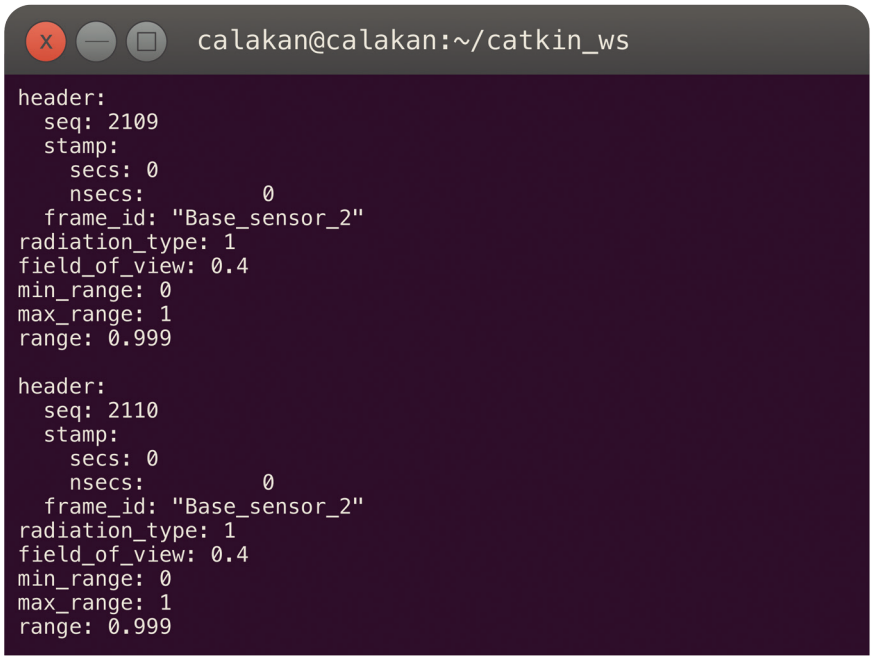

Figura 5.7. Ejecución del nodo *sensor_robot* con un obstáculo a 20 cm.

En la figura 5.8 se muestra la fotografía del robot con la prueba realizada.

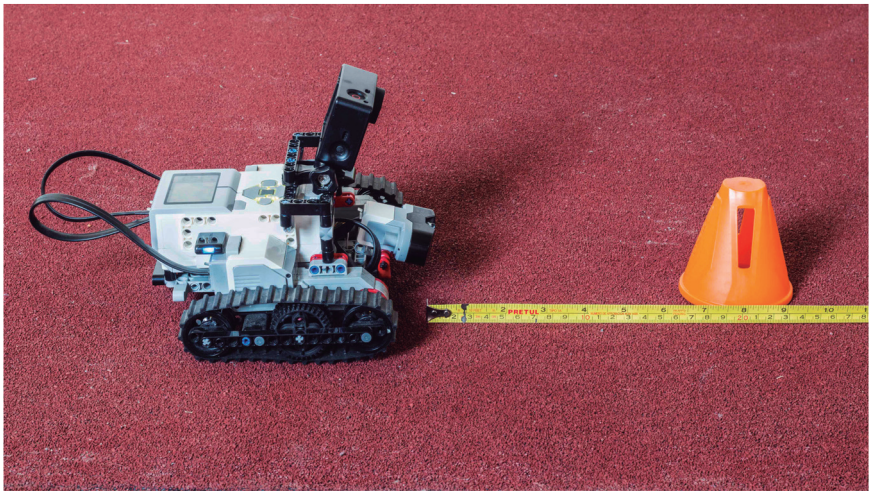

Figura 5.8. Pruebas del sensor infrarrojo.

Se ejecutan los mismos comandos usados en la prueba del nodo *battery_robot*, para conocer las características de este nodo, además se incluye el comando *rostpic hz* para conocer la velocidad a la que se envían los mensajes. Los resultados se muestran en la figura 5.9.

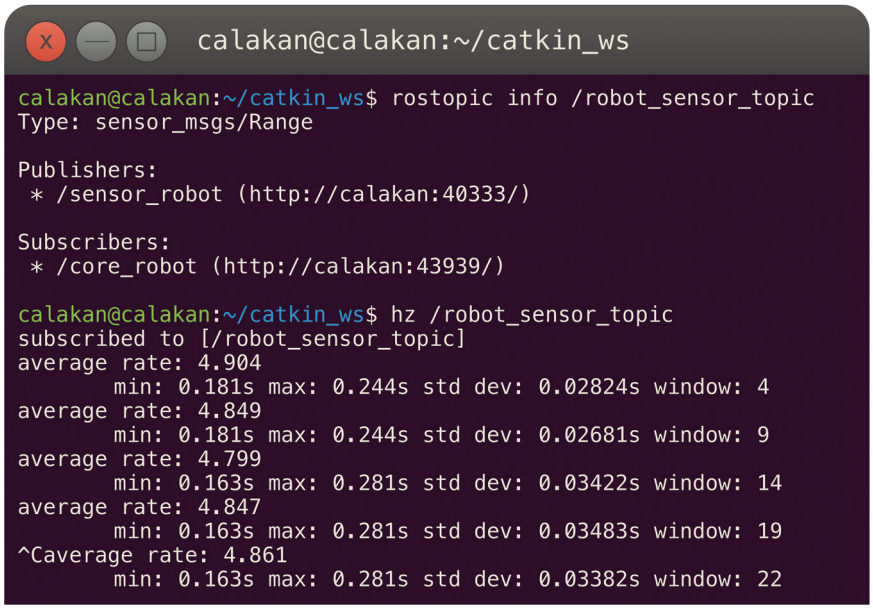

Figura 5.9. Información del topic *robot_sensor_topic*: tipo, nodo que envía, nodos que reciben y velocidad promedio de los mensajes enviados.

En la figura 5.9 se muestran los datos que se obtienen del topic *robot_sensor_topic*; mensajes *sensor_msgs/Range*, generados por el nodo *sensor_robot,* los cuales son recibidos por RVIZ y el nodo *core_robot,* con una velocidad promedio de 4 veces por segundo.

Pruebas del nodo camera_robot

El nodo *camera_robot* genera dos tipos de mensajes *Bool* e *Image*, de la misma forma en que se realizaron las pruebas a los nodos anteriores se obtiene la información de los topics utilizados por este nodo. En la figura 5.10 se muestra que el nodo *camera_robot* utiliza los topics: *human_topic* e *image_topic*, el tipo de mensajes que emplea así como también a qué nodos envía esta información.

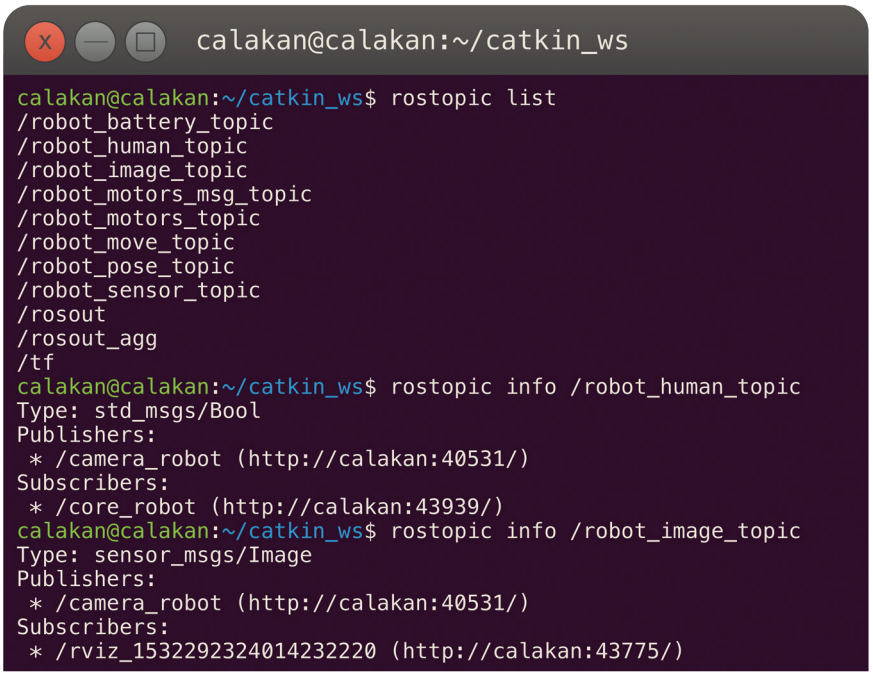

Figura 5.10. Topics utilizados por el nodo *camera_robot*.

Un dato importante de conocer del topic *image_topic*, además de la velocidad con la que envía los mensajes, es el ancho de banda que utiliza, ya que a diferencia de los topics anteriores, este topic envía imágenes. Se utiliza el comando *rostopic bw* para conocer el ancho de banda del topic *image_topic*. En la figura 5.11 se muestra a qué velocidad se envían las imágenes, así como también el ancho de banda que utiliza.

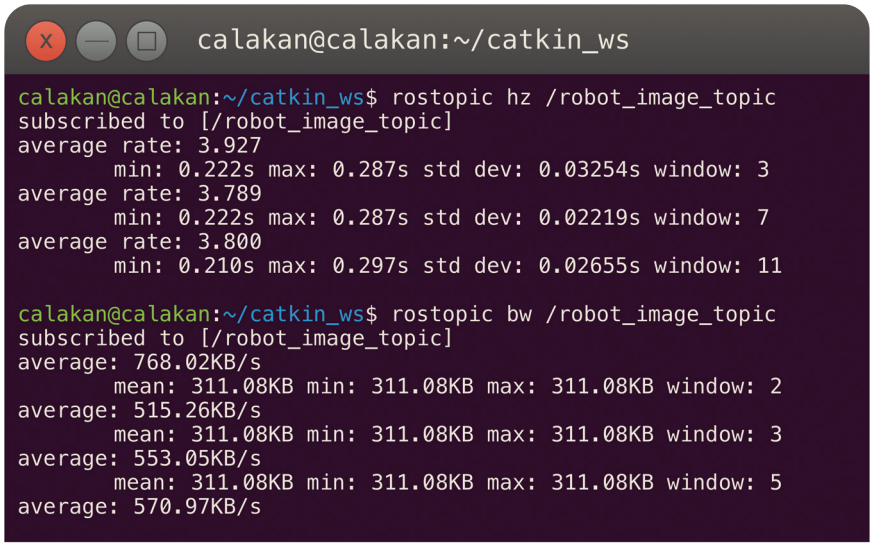

Figura 5.11. Información del topic *robot_image_topic*: velocidad promedio de los mensajes enviados y ancho de banda utilizado.

Estos resultados corresponden a una prueba realizada con la fotografía de una persona colocada a 25 cm de distancia de la cámara. En la figura 5.12 se muestra la fotografía de la prueba realizada.

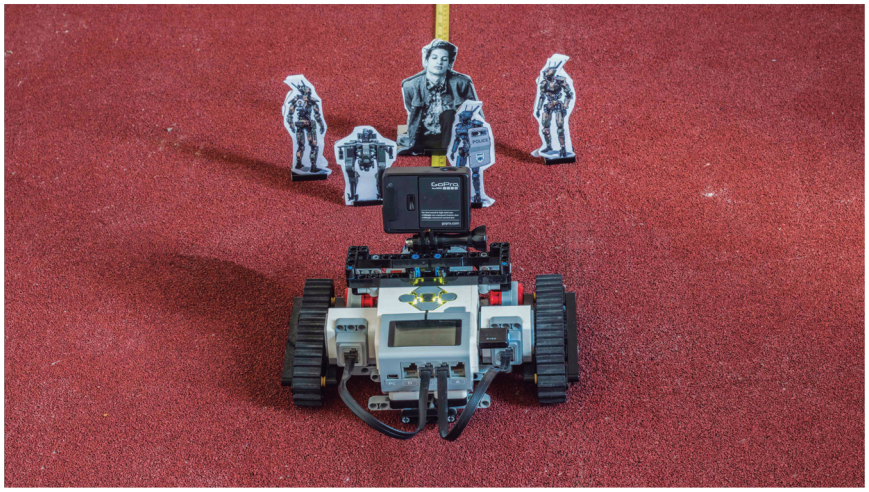

Figura 5.12. Prueba realizada al nodo *camera_robot*.

De la misma forma se analizan las características generales de los nodos restantes a través de los topics que utilizan.

Prueba de los nodos: motors_robot, mov_robot, tf_robot y core_robot

La función del nodo *motors_robot* consiste ejecutar las instrucciones de desplazamiento del nodo *core_robot* y enviarlas al nodo *mov_robot*. En la figura 5.13 se muestra que el topic a través del cual el nodo *motors_robot* recibe las instrucciones, se encuentra activo y que las instrucciones que recibe son a través de mensajes estándar del tipo *Int16*.

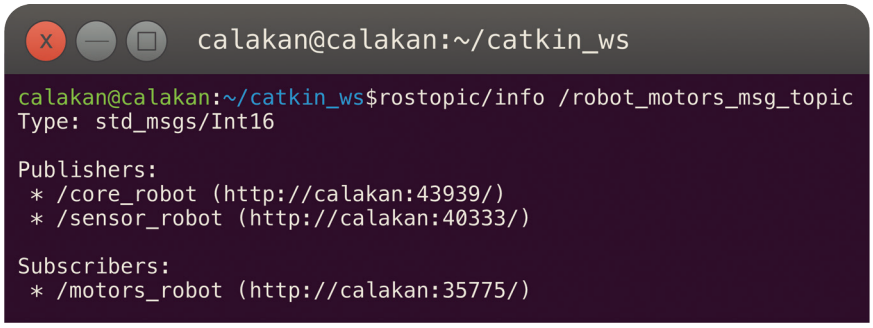

Figura 5.13. Características del topic *robot_motors_msg_topic*.

Y en la figura 5.14 se muestra que el topic *robot_motors_topic* con el cual se envía el movimiento de los motores al nodo *mov_robot*.

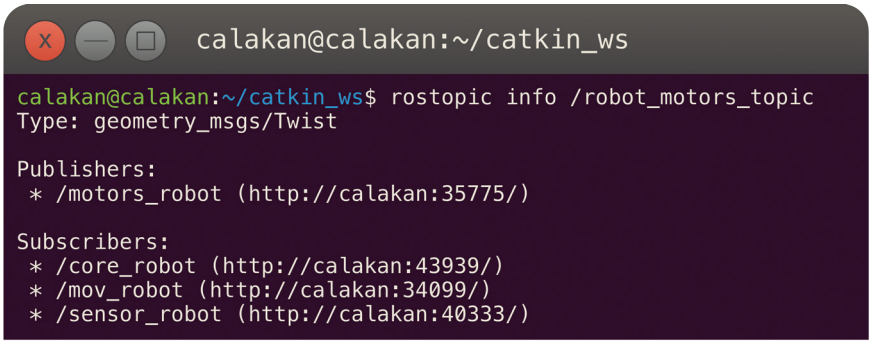

Figura 5.14. Características del nodo *robot_motors_topic*.

Por su parte, la tarea del nodo *mov_robot* es de realizar los cálculos del desplazamiento del robot con base en los mensajes recibidos de las rotaciones de los motores, para enviar la posición final al nodo *tf_robot*. En la figura 5.15 se muestra el topic a través del cual se envía el desplazamiento del robot.

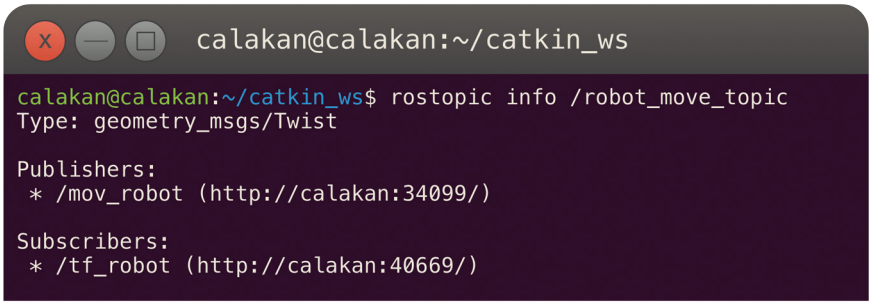

Figura 5.15. Características del topic *robot_move_topic*.

El nodo *tf_robot*, realiza la tarea de almacenar la posición después del último desplazamiento del robot y transmitir este desplazamiento a los nodos *core_robot* y *mov_robot*. En la figura 5.16 se puede ver que el nodo se encuentra activo y que envía la información con los mensajes apropiados.

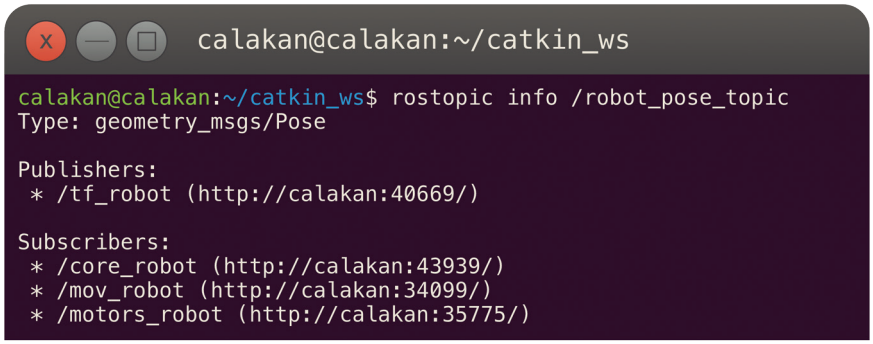

Figura 5.16. Características del topic *robot_pose_topic*.

Además el nodo *tf_robot* también realiza la función de transmitir las referencias para que se pueda visualizar la aplicación de búsqueda de personas en RVIZ, utilizando el topic *tf.* En la figura 5.17 se comprueba que el nodo se encuentra transmitiendo las referencias a RVIZ.

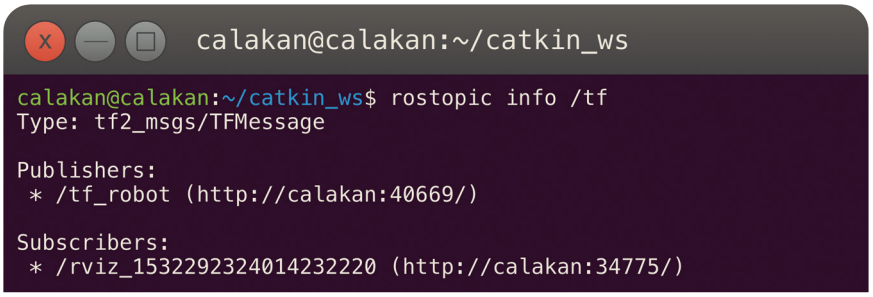

Figura 5.17. Características del topic *tf*.

Como se puede observar, las figuras anteriores muestran que los nodos se encuentran activos y trasmitiendo la información que generan a través de los topics y mensajes apropiados. De la misma manera, también se puede observar que el nodo *core_robot* por ser el nodo principal, está relacionado con todos los nodos recibiendo y transmitiendo información.

Pruebas de hardware

Las pruebas a realizar son: medir el alcance del sensor infrarrojo, comprobar que el algoritmo de reconocimiento facial funcione correctamente, evaluar que el desplazamiento del robot físicamente corresponda con los datos calculados y representados en RVIZ.

Pruebas de alcance del sensor infrarrojo

Para medir el rango de alcance del sensor infrarrojo se ejecuta únicamente el nodo *sensor_robot*, después se coloca un objeto a diferentes distancias del robot. En la figura 5.18 se muestran los cambios del campo range. La distancia a los cuales se obtiene un cambio de valor se muestran en la tabla 5.1.

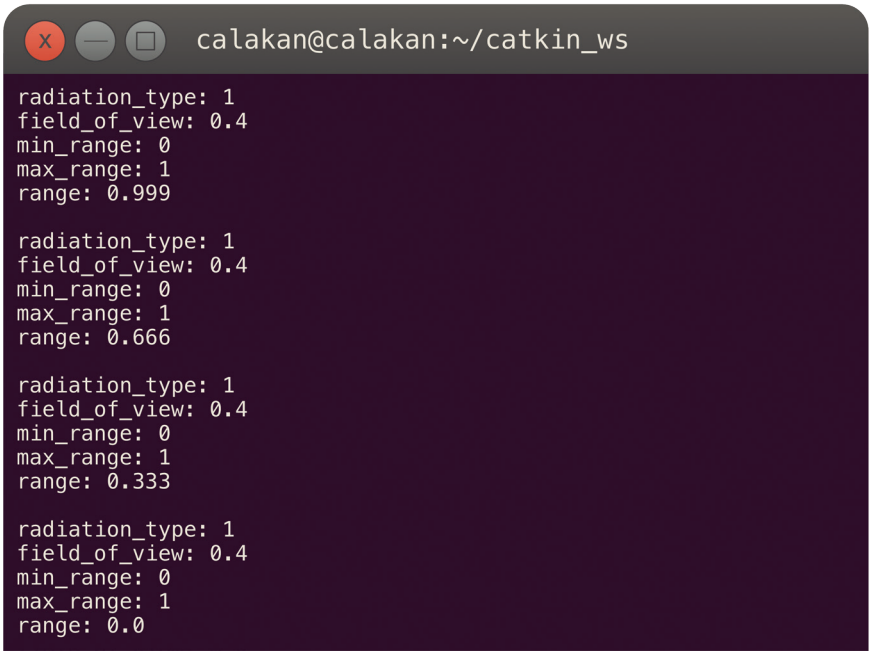

Figura 5.18. Resultados de la prueba realizada al sensor infrarrojo.

Distancia (D)	Valor del rango del sensor (Range)
D > 20 cm	0.999
D = 17 cm	0.66
D = 11 cm	0.33
D < 9 cm	0

Tabla 5.1. Valores obtenidos en las pruebas realizadas al sensor infrarrojo.

Los resultados obtenidos en la prueba anterior son utilizados para que el nodo *core_robot* cambie de dirección al detectar un obstáculo.

Prueba del algoritmo de reconocimiento facial

Esta prueba se realiza en el nodo *camera_robot*. Este nodo realiza el procesamiento de imagen para poder reconocer a una persona, con base en el reconocimiento facial. La prueba consiste en poner siluetas similares a las personas, pero en donde no se aprecia un rostro humano y una fotografía de una persona. Al detectar un rostro el nodo manda un mensaje del tipo *Bool* con el dato *True*. En la figura 5.19 se muestra la imagen de la cámara con la detección del rostro.

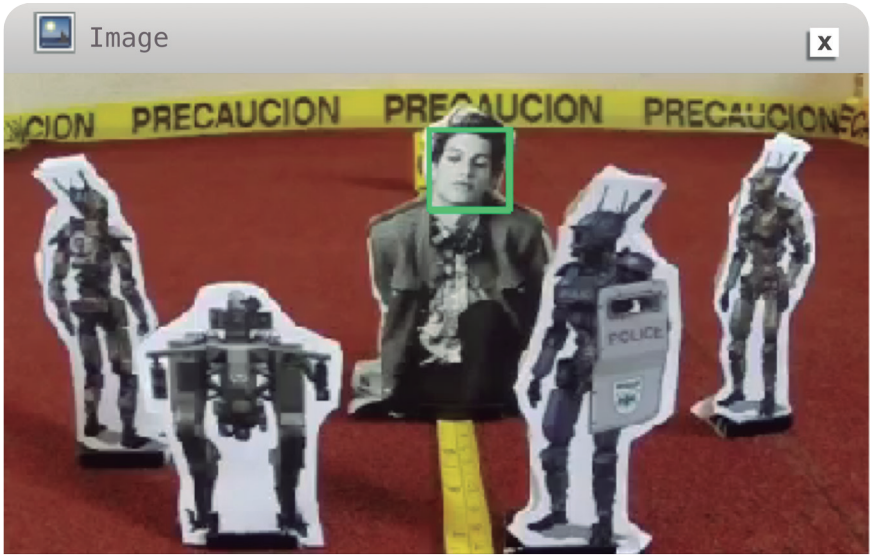

Figura 5.19. Resultado del reconocimiento facial del nodo *camera_robot*.

Pruebas de lectura y ejecución de los nodos motor_robot, mov_robot y tf_robot

Para que el robot se pueda desplazar se involucran tres nodos: *motors_robot*, el cual mueve los motores para que el robot avance, *mov_robot*, calcula el desplazamiento del robot con base en las rotaciones de los motores y el nodo *tf_robot*, que almacena la posición del robot conforme este se desplaza, además de mandar las referencias a RVIZ.

Para realizar esta prueba se utiliza un nodo adicional para

mandar las instrucciones de avanzar y girar al robot, para que cada movimiento del robot sea calculado y registrado en los nodos antes mencionados. En la figura 5.20 a) se muestra la fotografía del robot con una cinta métrica para medir su desplazamiento, en la figura 4.20 b) se muestra la posición inicial del robot en RVIZ, cada celda del Grid en RVIZ corresponde a un cuadrado de 20 cm.

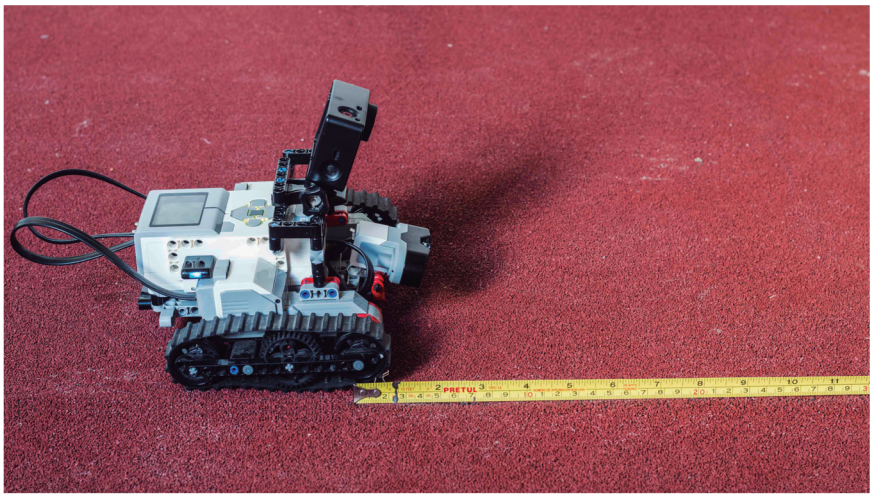

Figura 5.20. a) Fotografía del robot junto a una cinta métrica.

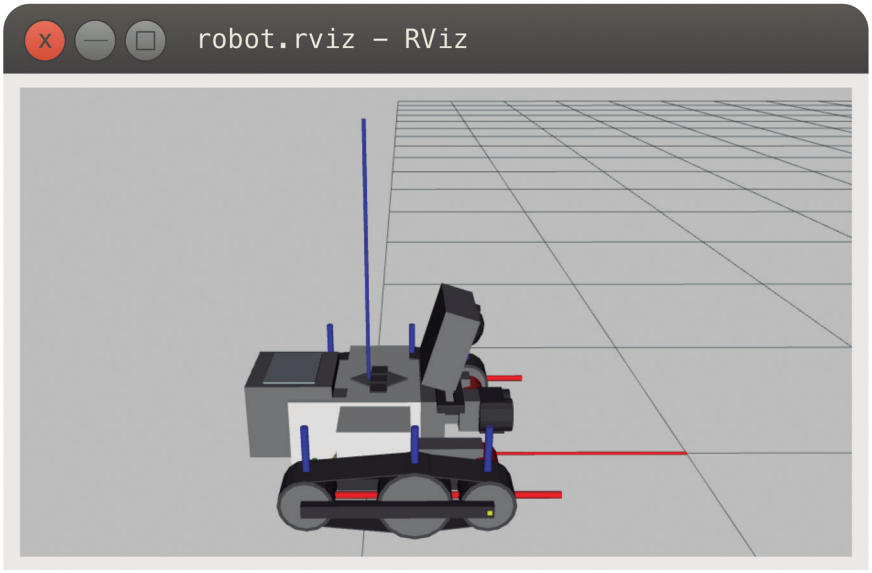

Figura 5.20. b) Posición inicial del robot en RVIZ.

En la figura 5.21 a) se muestra al robot después de ejecutar la instrucción de avanzar junto a la ventana de RVIZ. La distancia que recorrió el robot fue de 20 cm mientras que el nodo *mov_robot* calcula un desplazamiento de 0.94 de celda del grid en el eje X, que corresponde a 18.8 cm. En la figura 5.21 b se muestra el desplazamiento en RVIZ.

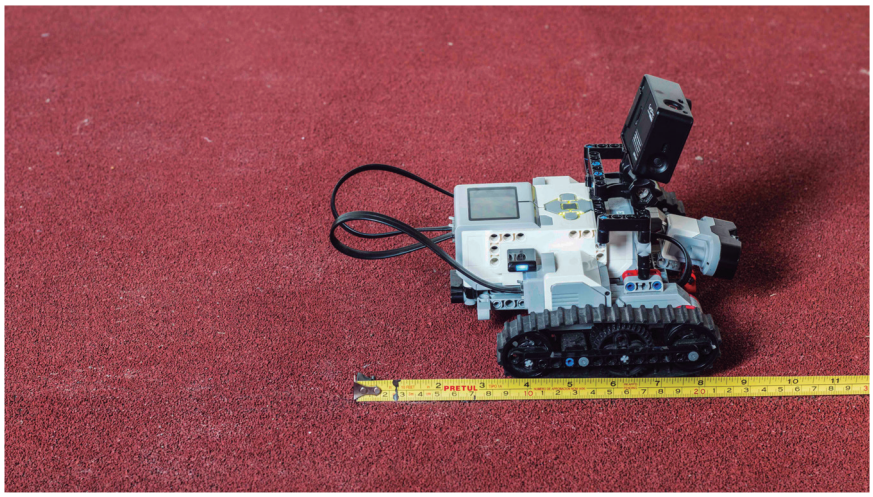

Figura 5.21. a) Robot después de desplazarse 20 cm.

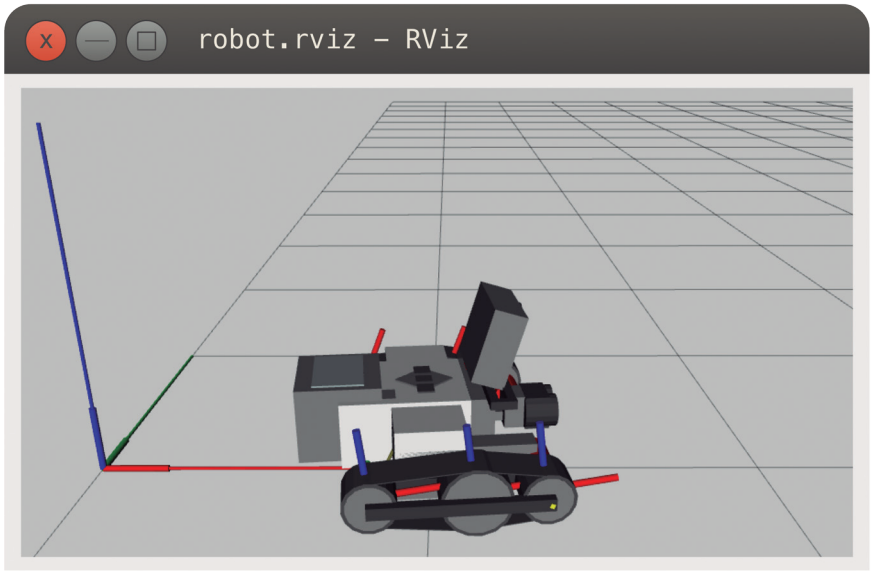

Figura 5.21 b) Posición del robot en RVIZ.

En la figura 5.22 a) se muestra la ejecución del nodo mov_robot, el cual calcula el desplazamiento del robot con cada grado de movimiento de los motores, cuando los motores terminan de moverse recibe la posición a la cual llegó el robot. En la figura 5.22 b) se muestra que el nodo *tf_robot* envía la posición del robot.

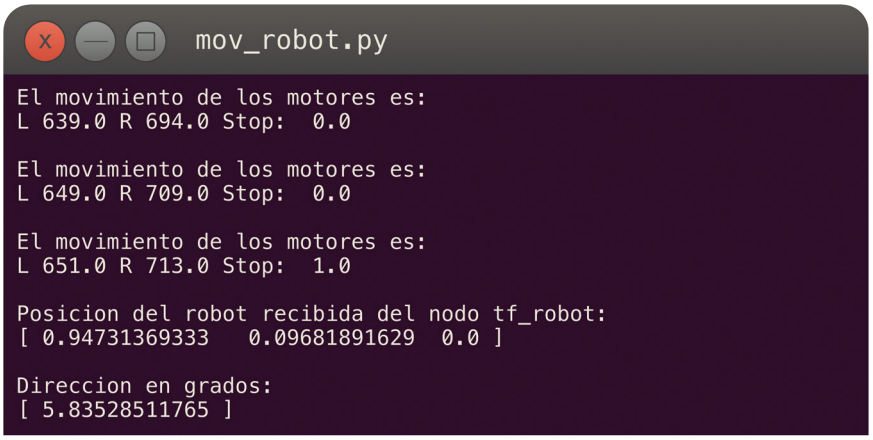

Figura 5.22. a) nodo *mov_robot* en ejecución.

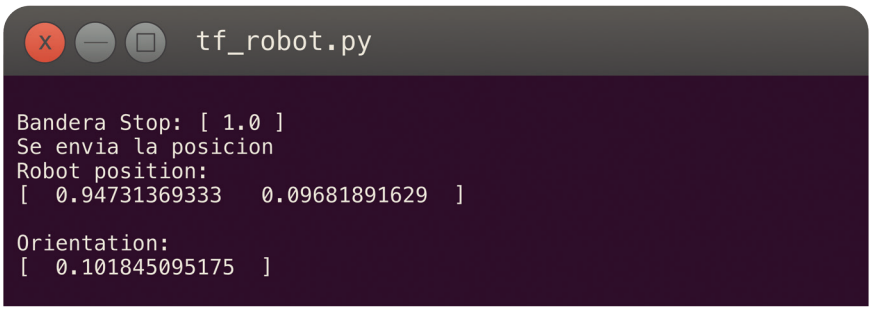

Figura 5.22. b) nodo *tf_robot* en ejecución.

Se manda al robot la instrucción de girar después de que había realizado un desplazamiento frontal, en la figura 5.23 a) se muestra al robot después de haber girado y en la figura 5.23 b) se muestra al robot en RVIZ.

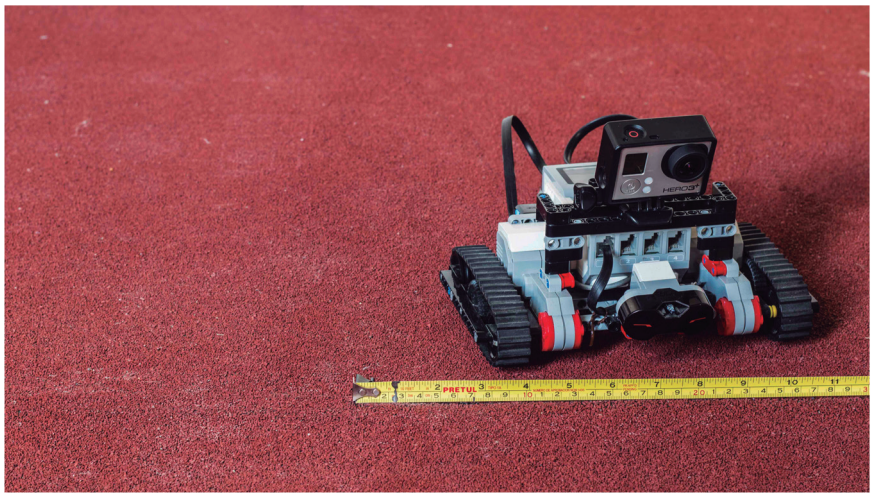

Figura 5.23. a) Giro del robot después de haber avanzado.

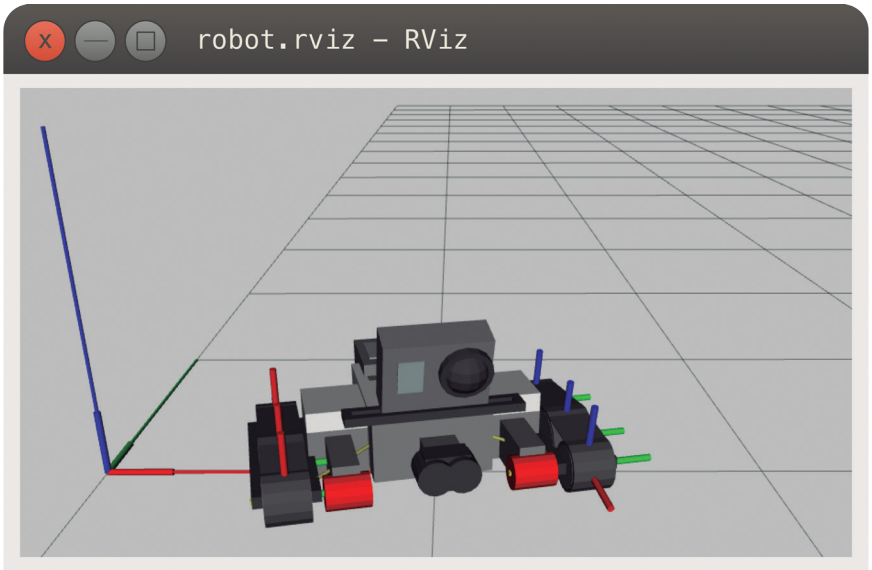

Figura 5.23. b) Giro del robot en RVIZ.

En la figura 5.24 a) se puede observar que el nodo *mov_robot* recibió el movimiento de los motores; el motor izquierdo con una rotación positiva mientras que el motor derecho tuvo una rotación negativa, dando como resultado un ángulo negativo de 76 grados. En la figura 5.24 b) se muestra el nodo *tf_robot* que guardó los desplazamientos del robot.

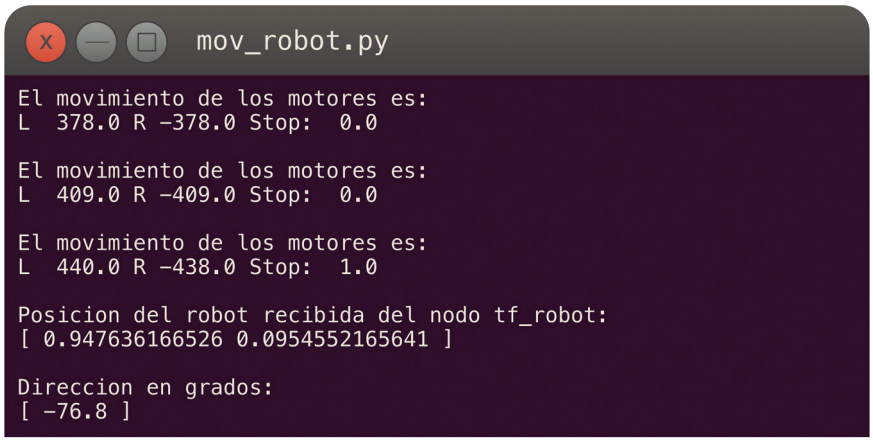

Figura 5.24. a) Nodo *mov_robot*.

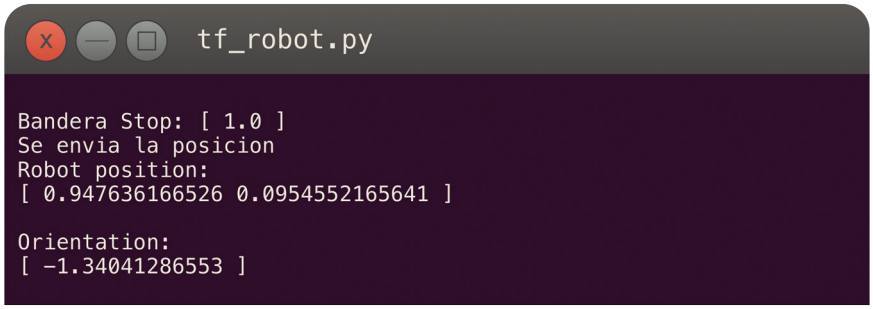

Figura 5.24. b) Nodo *tf_robot*.

Cabe mencionar que estas pruebas se realizaron en varias ocasiones obteniendo resultados similares, aunque la instrucción de avanzar y girar siempre fue la misma, hubo variaciones en la distancia recorrida o en el número de grados que giró el robot. Sin embargo, lo anterior no significa un problema ya que el nodo que calcula el desplazamiento funciona de manera independiente con base en la rotación de los motores.

Pruebas de la aplicación en diferentes escenarios

El nodo *core_robot* integra a todos los nodos para tener la aplicación de búsqueda de personas, por lo que el desempeño de su funcionamiento depende principalmente de la forma en la que utiliza la información que recibe de los demás nodos. La función del nodo *core_robot* es explorar un área y evadir obstáculos, hasta que la bandera home sea igual a 1, esto sucede cuando el voltaje es bajo o cuando se localiza una persona. El principal objetivo es encontrar a una persona por lo que se realizan a continuación dos pruebas en escenarios diferentes: en el primer escenario se tiene que evadir un obstáculo, en el segundo escenario se tiene que evadir un obstáculo, encontrar a una persona y regresar al punto de partida.

Escenario 1: evasión de obstáculo

En un área de 2.5 metros cuadrados se coloca un obstáculo a 80 cm de distancia y se pone en funcionamiento la aplicación. Se mantienen activas las terminales de ejecución de los nodos *core_robot* y *motors_robot*, para observar su funcionamiento. En la figura 5.25 a) se muestra el robot al inicio de la aplicación y en la figura 5.25 b) se muestra la ventana RVIZ, en la cual se observa la representación del robot, con el sensor infrarrojo activo y la imagen de la cámara.

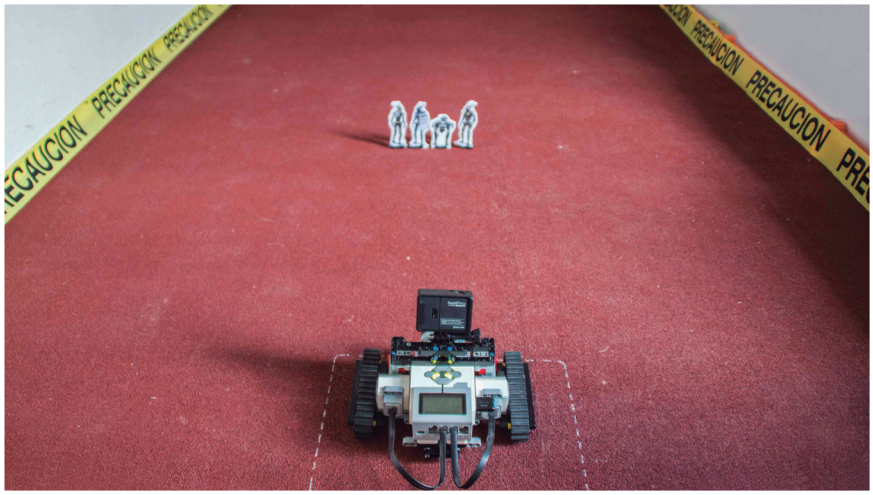

Figura 5.25 a) Inicio de la prueba evasión de obstáculo.

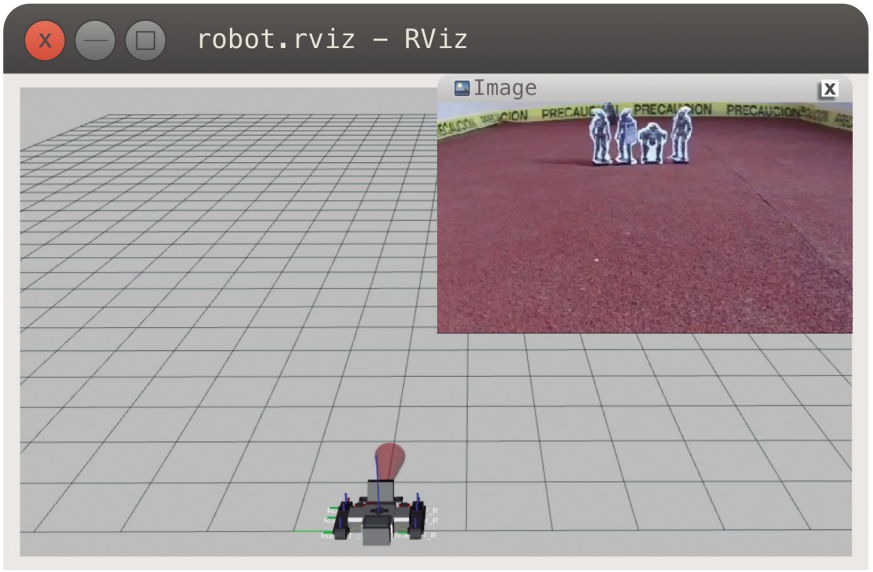

Figura 5.25. b) Inicio de la prueba evasión de obstáculo en RVIZ.

En los nodos *core_robot* y *motors_robot*, se muestra que al no detectar un obstáculo el nodo *core_robot* envía la instrucción avanzar, el nodo *motors_robot* la recibe y ejecuta, cuando termina de mover los motores envía la bandera *stop*. Cada envío y ejecución de instrucción de movimiento termina con la bandera *stop*. Este proceso continua hasta que se encuentra con un obstáculo. En la figura 5.26 a) y b) se muestra la interacción de los nodos *core_robot* y *motors_robot* durante el avance del robot.

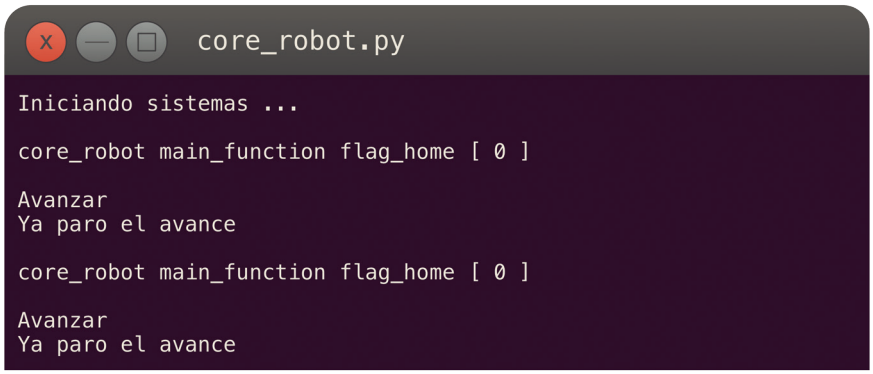

Figura 5.26. a) Nodos *core_robot* durante el avance del robot.

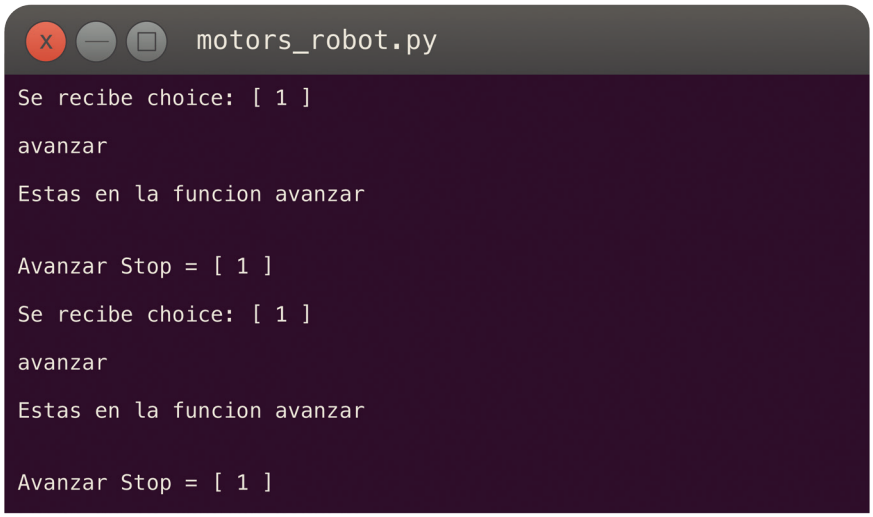

Figura 5.26. b) Nodo *motors_robot* durante el avance del robot.

Cuando el sensor infrarrojo detecta el obstáculo el nodo *core_robot* envía el mensaje de retroceder y espera a que el nodo *motors_robot* le envíe la bandera de *stop*, después de que recibe la bandera de *stop*, manda la instrucción de girar (gira de forma aleatoria), y cuando ya no detecta ningún obstáculo vuelve avanzar. En la figura 5.27 a) se muestra al robot cuando encuentra un obstáculo y en la figura 5.27 b) se puede ver en la ventana RVIZ, al detectar un obstáculo el cono de radiación que representa al sensor infrarrojo disminuye su tamaño.

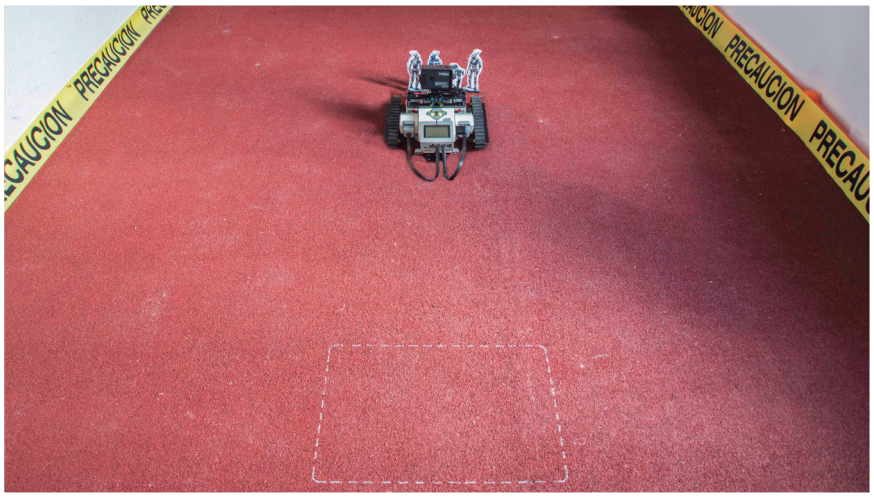

Figura 5.27. a) Prueba evasión de obstáculo: obstáculo detectado.

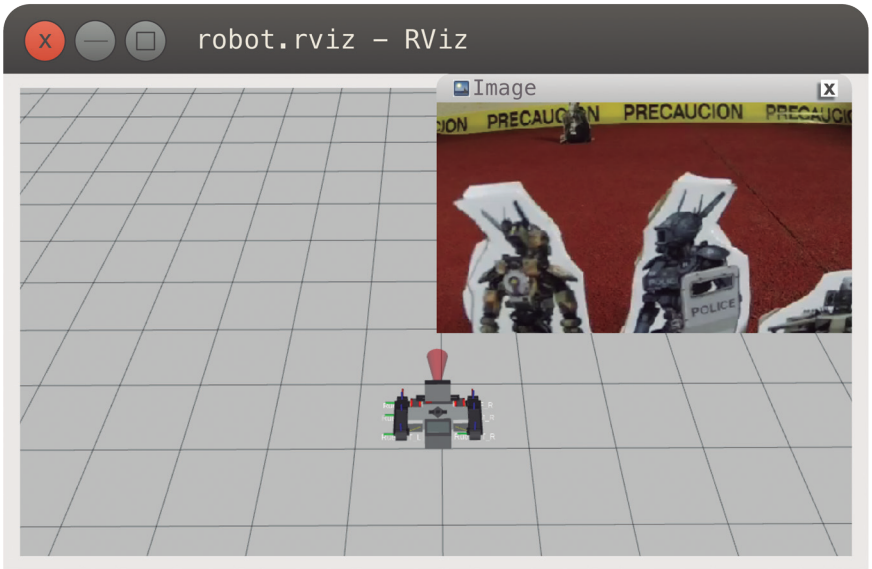

Figura 5.27. b) Prueba evasión de obstáculo: obstáculo detectado en RVIZ.

Por otro lado, en la figura 5.28 a) y B) se muestra la interacción de los nodos *core_robot* y *motors_robot* cuando se detecta un obstáculo.

core_robot.py

```
Avanzar
Ya paro el avance

core_robot main_function flag_home [ 0 ]

Reversa!
Ya paro la reversa
Gira
```

Figura 5.28. a) Nodos *core_robot* durante la detección de un obstáculo.

motors_robot.py

```
Se recibe choice: [ 1 ]

avanzar

Estas en la funcion avanzar


Avanzar Stop = [ 1 ]

Se recibe choice: [ 2 ]

Retroceder

Estas en la funcion Retroceder


Retroceder Stop = [ 1 ]

Se recibe choice: [ 3 ]
```

Figura 5.28. b) Nodo *motors_robot* durante la detección de un obstáculo.

En la figura 5.29 a) finalmente se muestra la posición del robot después de que retrocedió y giró, en la figura 5.29 b) se puede observar en la ventana RVIZ que el cono de radiación que representa al sensor infrarrojo vuelve a cambiar de tamaño.

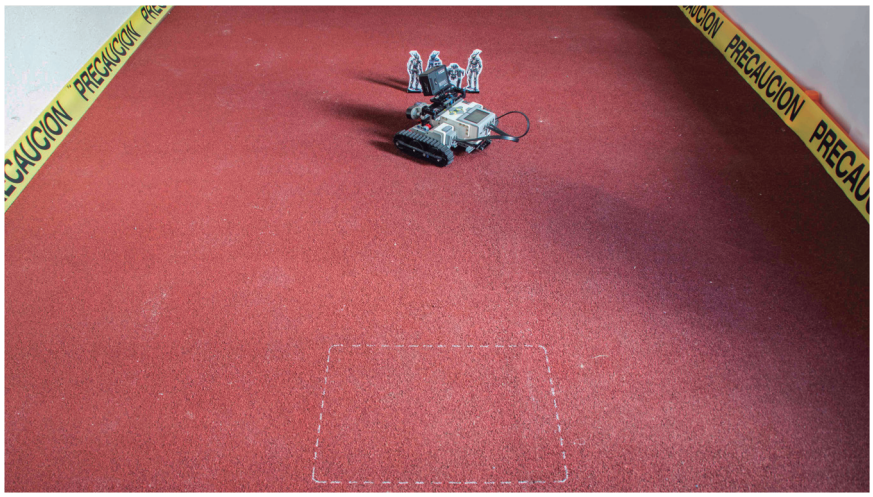

Figura 5.29. a) Robot después de girar para evadir un obstáculo.

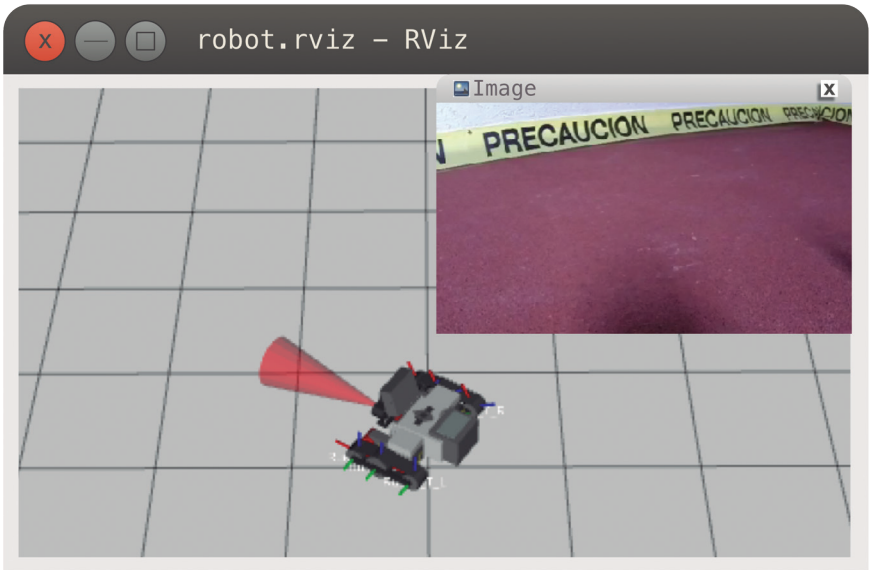

Figura 5.29. b) Robot en RVIZ después de girar para evadir un obstáculo.

Como se puede observar en la prueba realizada, el robot detecta y evade un obstáculo. El siguiente paso es probar qué sucede cuando detecta a una persona.

Escenario 2: evasión de obstáculo, detección de rostro y regreso al punto de partida

Esta prueba tiene las mismas características de la prueba realizada en el escenario 1, agregando a 1.20 metros de distancia del punto de partida del robot la fotografía que se utilizó en la prueba de detección de rostro. En la figura 5.30 a) se muestra el inicio de la prueba con el robot en el punto de partida, el obstáculo y la fotografía que representa a una persona, en la figura 5.30 b) se muestra la ventana de RVIZ al inicio de la prueba.

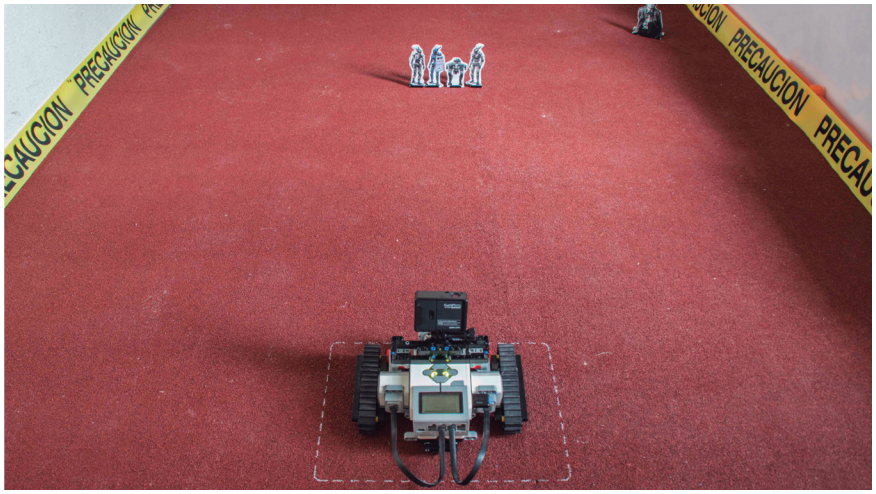

Figura 5.30. a) Inicio de la prueba evasión de obstáculo, detección de rostro y regreso al punto de partida.

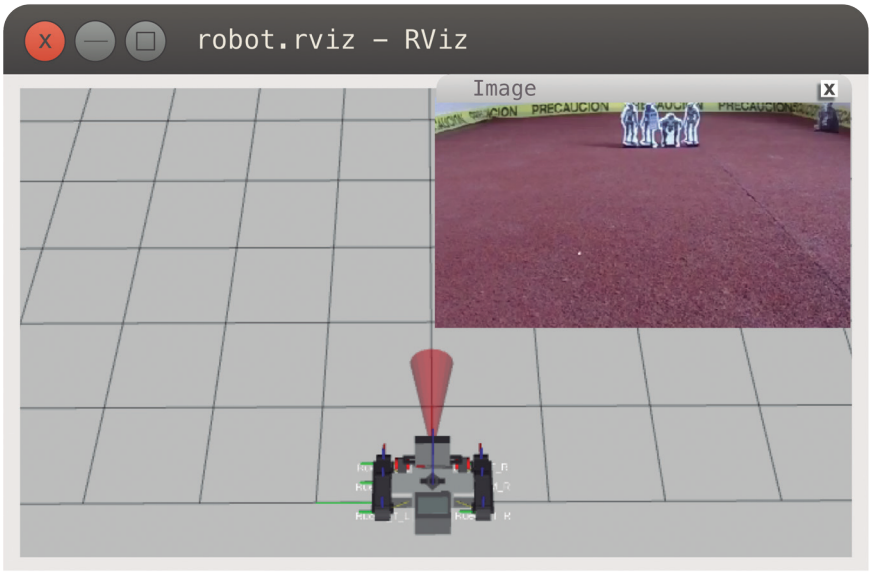

Figura 5.30. b) Ventana RVIZ al inicio de la prueba evasión de obstáculo, detección de rostro y regreso al punto de partida.

De la misma manera que en la prueba anterior el robot evade el obstáculo (el robot gira de manera aleatoria para evadir obstáculos). En la figura 5.31 a) se muestra el cambio de dirección del robot después de detectar el obstáculo y en la figura 5.31 b) se muestra la ventana de RVIZ en donde se observa el cambio de dirección.

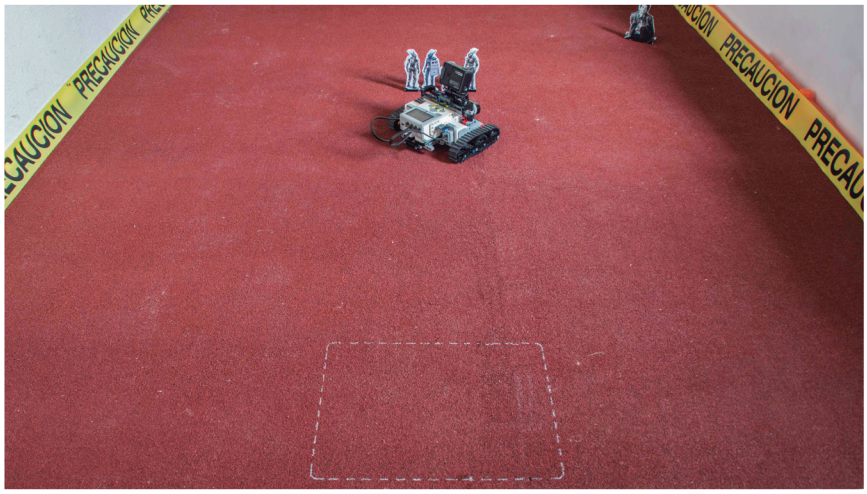

Figura 5.31. a) Giro del robot después de detectar el obstáculo.

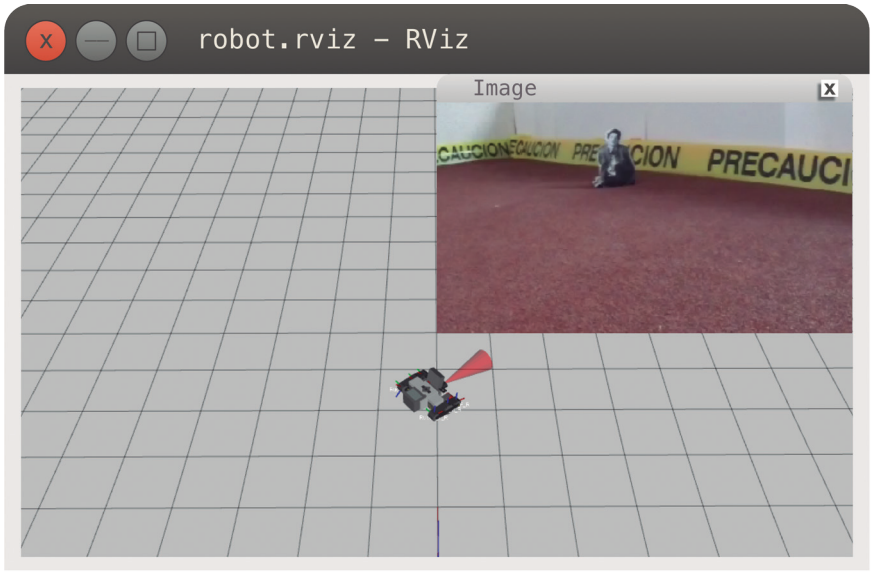

Figura 5.31. b) Giro del robot en RVIZ después de detectar el obstáculo.

Después de girar, el robot avanza como se explicó en la prueba anterior hasta encontrar a una persona, representada por una fotografía. Cuando detecta que es el rostro de una persona, la bandera *Home* cambia a *True*. En la figura 5.32 a) se muestra la imagen del robot en el momento en que se detecta un rostro y en la figura 5.32 b) se muestra a la ventana RVIZ que corresponde a este momento.

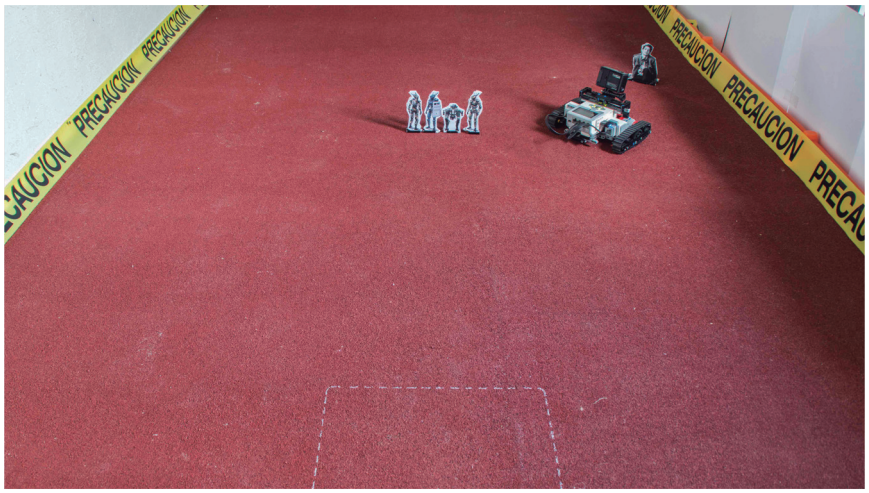

Figura 5.32. a) Detección de rostro durante la prueba, después de evadir un obstáculo.

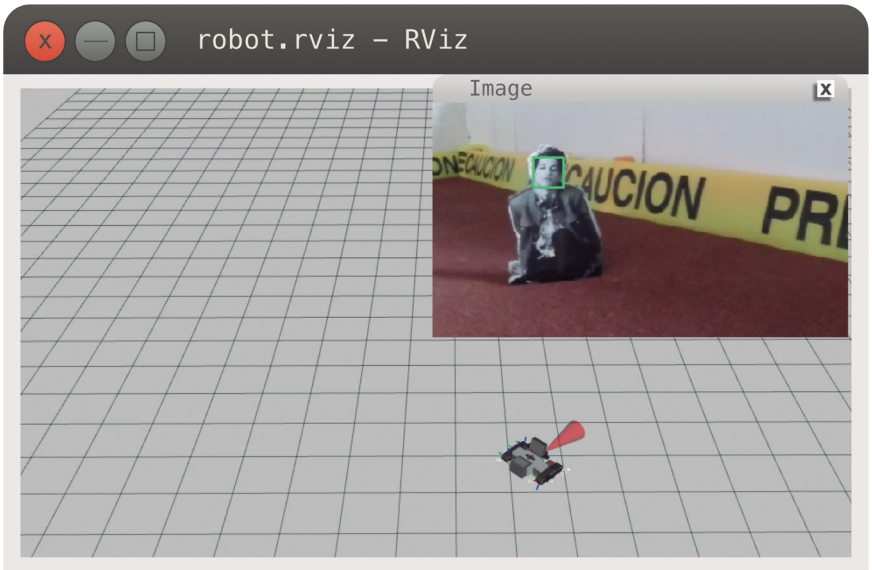

Figura 5.32. b) Detección de rostro en RVIZ durante la prueba, después de evadir un obstáculo.

Las coordenadas corresponden a las celdas del grid las cuales equivalen a 20 cm.

En la figura 5.33 se muestra que el nodo *core_robot*, detecta a una persona y cambia el estado de la bandera *Home* a *True*, indicando además las coordenadas en las que detectó a la persona. A partir de este momento el robot cambia su dirección hacia el punto de partida, y calcula con base en sus coordenadas de posición la distancia hacia el punto de partida. En esta prueba se establece una tolerancia de 1.5 de celda alrededor del punto de partida, que corresponden a 30 cm de distancia, la tolerancia representa la distancia a la cual se puede recuperar el robot después de realizar la exploración.

```
core_robot.py

core_robot
 Human [ True  ] flag_home: [  1 ]

Pose Human [ 4.63456 , 1.47835 ]


core_robot main_function flag_home [ 1 ]
```

Figura 5.33. Ejecución del nodo *core_robot* en el momento en que detecta a una persona.

Las coordenadas tiene como origen el punto de partida del robot. El eje X va hacia adelante y hacia atrás y el eje Y va de izquierda a derecha. Coordenadas positivas adelante y a la derecha y coordeanadas negativas hacia atrás y a la izquierda.

En la figura 5.34 a) se muestra la imagen del robot después de ejecutar la instrucción alinear para regresar al punto de partida así como en también en la figura 5.34 b) se muestra la ventana de RVIZ con este proceso.

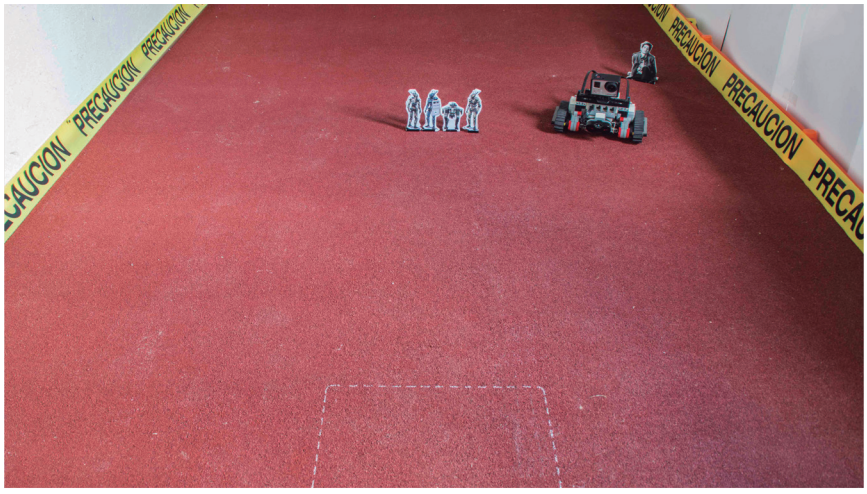

Figura 5.34. a) Alineación del robot hacia el punto de partida.

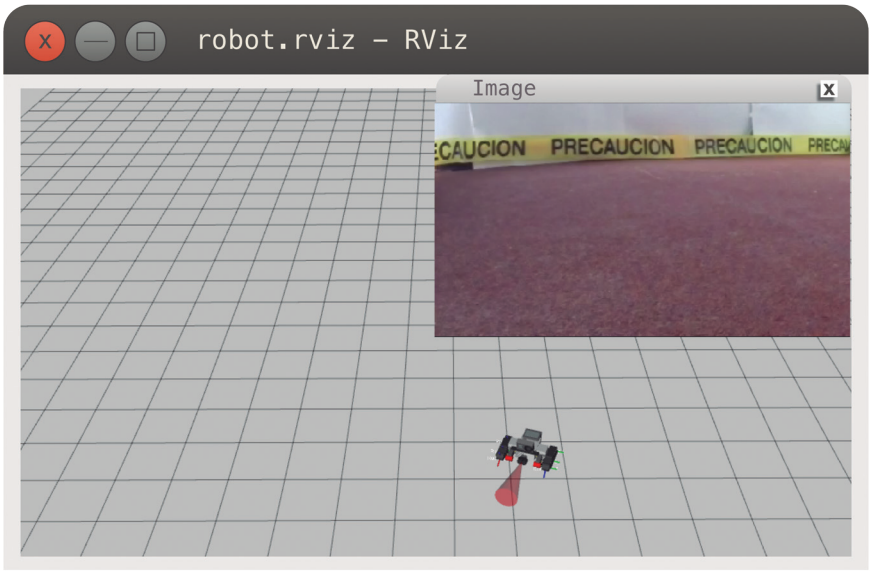

Figura 5.34. b) Alineación del robot en RVIZ hacia el punto de partida.

En la figura 5.35 se muestra el proceso anterior en el nodo *core_robot*.

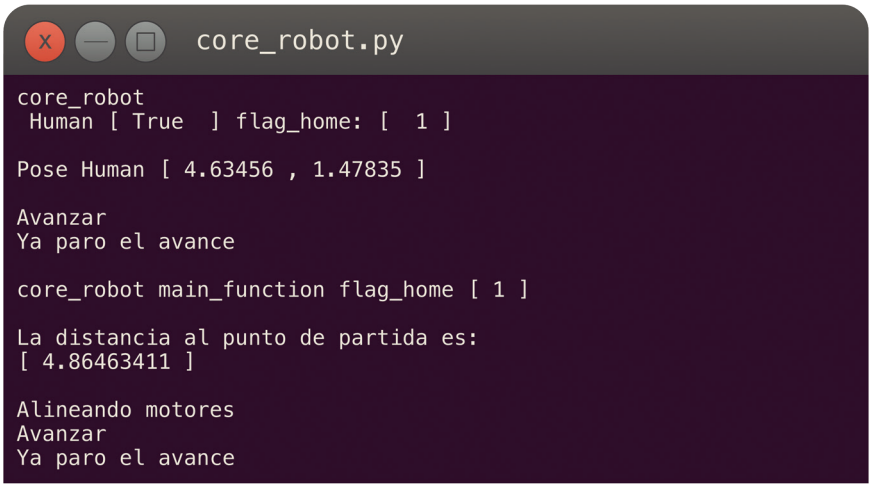

Figura 5.35. Nodo *core_robot* durante el proceso de alineación hacia el punto de partida.

El robot continua avanzando hasta llegar al punto de partida, momento en que termina la aplicación. En la figura 5.36 a) se muestra al robot de regreso al punto de partida y en la figura 5.36 b) se muestra la imagen correspondiente en RVIZ.

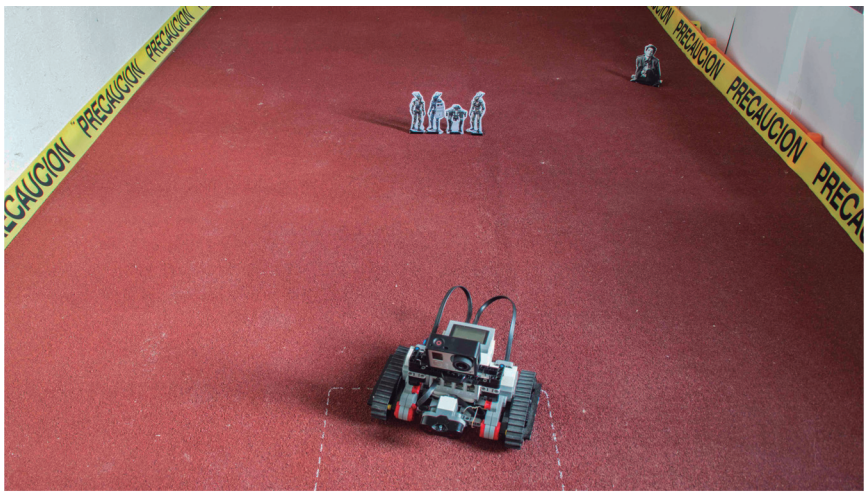

Figura 5.36. a) Regreso del robot al punto de partida.

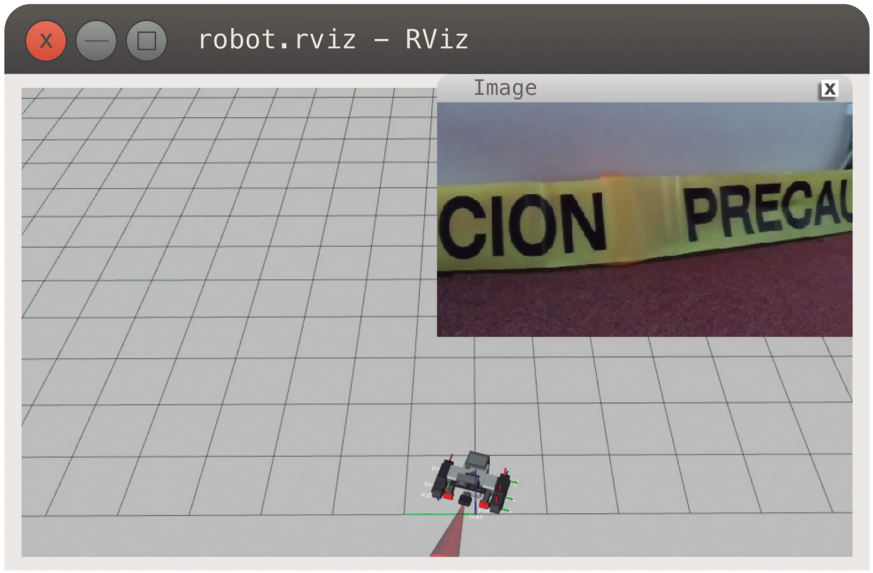

Figura 5.36. b) Regreso del robot al punto de partida en RVIZ.

En la figura 5.37 se muestra el proceso anterior en el nodo *core_robot*.

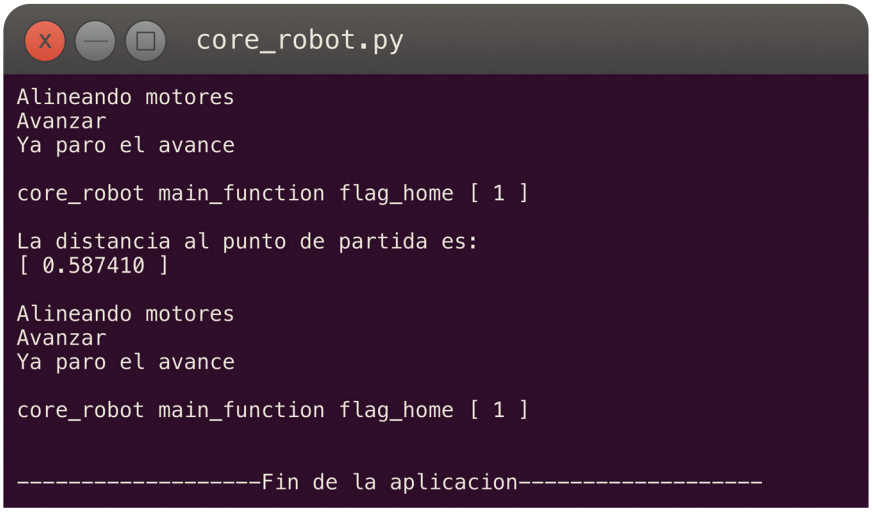

Figura 5.37 Fin de la aplicación en el nodo *core_robot.*

Cabe mencionar que aunque se realizaron varias pruebas para mostrar este objetivo, esta es una forma resumida de presentar las pruebas realizadas cubriendo las funciones principales de la aplicación.

Discusión de resultados

A lo largo del desarrollo de este proyecto, se implementó un robot capaz de localizar personas. Los recursos utilizados para este proyecto son: una computadora, un *router*, una cámara y un robot Lego. La plataforma en la cual se desarrolla la aplicación es ROS. Para que el robot pueda explorar un área en busca de personas es necesario que cuente con comunicación inalámbrica, ya que el sistema se ejecuta en la computadora y no en el robot porque el procesador del robot no tiene la capacidad para realizar el procesamiento de imagen.

Para que los equipos se comuniquen fue necesario implementar una LAN inalámbrica en modo infraestructura. Una LAN inalámbrica en modo infraestructura está formada por estaciones de trabajo y puntos de acceso que comparten un medio inalámbrico, las estaciones de trabajo en este caso son el robot y la computadora y la función de puntos de acceso la realizan el router y la cámara. En este caso, el robot con el *router* y la computadora con la cámara representan dos conjuntos básicos de servicios 802.11, que se encuentran unidos por la computadora ya que ésta además de conectarse al punto de acceso, función que realiza la cámara, se encuentra conectada a través del puerto ethernet al *router*, teniendo al final un conjunto extendido de servicios. Los equipos cuentan con direcciones

IP, las cuales se utilizan para poder comunicarse entre sí. En este proyecto se observa que se tienen dos redes: una es la red 192.168.1.0 que tiene tres elementos, computadora, el robot y router, este último realiza la función de punto de acceso; y la otra red, la 10.5.5.0 formada por la computadora y la cámara, esta última realiza la función de punto de acceso.

La aplicación se encuentra desarrollada en el sistema ROS que se organiza en paquetes y que está programada en Python. El sistema ROS provee bibliotecas y herramientas de comunicación, lo cual permite interactuar entre diferentes sistemas. Como se pudo observar, este proyecto está dividido en nodos que realizan una tarea en específico con diferentes características. Los nodos utilizados fueron: *battery_robot, sensor_robot, camera_robot, motors_robot, mov_robot, tf_robot* y *core_robot*, cada nodo genera información diferente. La aplicación se dividió por nodos para que el mantenimiento o ampliación de cada nodo sea posible aprovechando los mensajes que proporciona ROS de acuerdo al tipo de información que se requiera enviar. En los nodos *battery_robot*, *sensor_robot* y *camera_robot* se emplearon los mensajes *sensor_msgs* del tipo *BatteryState, Range* e *Image*, respectivamente. ROS también permite utilizar bibliotecas externas como OpenCV, del cual se utilizó un algoritmo de su biblioteca para realizar el reconocimiento facial e integrarlo a la aplicación en ROS utilizando la biblioteca *cv_bridge*. Como se

vio en la prueba del nodo *camera_robot*, cuando el algoritmo de reconocimiento facial detecta un rostro lo comunica empleando los mensajes *std_msgs* del tipo *Bool* y para elegir las opciones de qué hacer con la información obtenida en los nodos de adquisición de datos, el nodo *core_robot* envía la acción a realizar utilizando los mensajes *std_msgs* del tipo *Int16*. También se comprobó que los nodos *motors_robot* y *mov_robot* utilizan los mensajes *geometry_msgs* del tipo *Twist* para enviar la rotación de los motores y del tipo *Pose* para almacenar la información de posición y orientación del robot, así como también para almacenar las coordenadas de dónde se localizó a una persona. Por último, se pudo observar que se utilizó el sistema de referencia *tf* y el formato de descripción unificado para robots URDF en conjunto con la herramienta RVIZ, para poder ver de manera gráfica el desplazamiento del robot en la aplicación de búsqueda de personas.

A continuación se describen algunos resultados que se obtienen al ejecutar varias veces las pruebas descritas anteriormente:

- El sistema de ubicación del robot funciona con base en la rotación de los motores. En ocasiones los motores llegaban perder tracción por lo que existía rotación en los motores más no así un desplazamiento, lo que ocasionaba un cálculo de desplazamiento en RVIZ que no correspondía del todo

al movimiento real del robot. Para minimizar esté error se cambio la superficie de prueba por una más aspera.

- En cuanto al algoritmo de reconocimiento facial, no siempre reconoce a una persona a la misma distancia debido a que cuando el robot se encuentra en movimiento la imagen no es estable. Esta fue la razón por la cual el robot avanza una distancia corta y se detiene, permitiendo que la cámara no tenga trepidaciones y la imagen sea lo suficientemente estable para que el algoritmo de reconocimiento facial funcione correctamente.
- En cuanto a la detección de obstáculos, frenar a tiempo los motores fue un asunto que requirió varias pruebas, ya que hay tres modos de frenar los motores: una es retirando el voltaje en ellos y dejando que la inercia los detenga, otra es mandando una carga pasiva que absorbe la energía de los motores y la última es forzando el motor a mantener una posición fija. En el primer caso aunque se detectaba a tiempo el obstáculo no daba tiempo suficiente para frenar, en los otros casos el tiempo de frenado es menor, por lo que se eligió utilizar la última opción, forzar el motor a mantener su posición cuando detectaba un obstáculo.
- Además se observó que el cono de radiación del sensor infrarrojo se dirige hacia arriba y no frontalmente, por lo que se inclinó la posición, para que detectara de mejor manera.

Conclusiones

El objetivo principal de este proyecto es el de implementar un robot autónomo capaz de explorar un área, evadir obstáculos, identificar personas, así como también guardar la ubicación en el caso de localizar a una persona y regresar al punto de partida, para lo cual se utilizó ROS de la manera más amplia posible y se adaptó al robot que se tiene. Se integró al robot una cámara GoPro y un software de reconocimiento facial de OpenCV. Se demuestra que es posible integrar diferente hardware y software para resolver un problema utilizando ROS. La forma en la que se realizó la aplicación fue dividiendo las tareas en nodos, con el propósito de poder ampliar la aplicación, es decir, en caso de contar con sensores o motores u otro tipo de hardware adicional, la integración resulte sencilla.

El objetivo se logra ya que se demostró que el robot es capaz de evadir obstáculos, reconocer rostros y logra regresar al punto de partida de manera autónoma con las siguientes consideraciones:

- El robot puede explorar superficies que sean del todo planas.
- Puede evadir obstáculos siempre y cuando los obstáculos se encuentren dentro del rango del sensor infrarrojo.
- La identificación de personas se realiza con base en la detección de rostros por lo que es necesario que el rostro sea visible.
- La cámara funciona en lugares iluminados unicamente.

- Al evadir un obstáculo se cambia la dirección de manera aleatoria, sin embargo es posible integrar algoritmos más complejos.

Al final las limitaciones en el desarrollo de este proyecto son de hardware y no necesariamente de software.

En esté proyecto se crea un sistema base para poder probar otro tipo de hardware y algoritmos de busqueda más complejos. Como trabajo a futuro hay varios temas a desarrollar, por ejemplo:

- Utilizar un mayor número de sensores para la adquisición de datos y poder tener mayor información del entorno que se está explorando.
- Utilizar diferentes cámaras y algoritmos de procesamiento de imagen que pudieran implementarse también como sistemas de navegación para que el robot pueda trazar diferentes rutas y elegir mejores opciones de exploración.
- Agregar variables físicas como peso e inercia al modelo gráfico con el cual se visualiza la aplicación para tener un mejor sistema de monitoreo del robot.
- Se podrían implementar algoritmos de búsqueda que trabajen en conjunto con varios robots, lo que requiere que los robots cuenten con un de sistema comunicación.

Apéndice

Instalación ROS

Para la instalación es necesario configurar la computadora para que acepte los paquetes que provee ROS.

```
sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Después se instala la versión completa de kinetic.

```
apt-get install ros-kinetic-desktop-full
```

Una vez que se instala hay que iniciar las dependencias.

```
rosdep init
```

Para instalar paquetes adicionales a ROS se recomienda instalar la herramienta rosinstall.

```
apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Para trabajar con ROS es necesario contar con un espacio de trabajo, en este proyecto se emplea catkin. Para el espacio de trabajo es necesario crear las carpetas catkin_ws y dentro de ella la carpeta src; después en la carpeta src se compila.

```
catkin_make
```

Para crear el paquete robot es necesario ubicarse dentro de la carpeta src y ejecutar el siguiente comando:

```
catkin_create_pkg robot geometry_msgs rosconsole roscpp roslib rostime std_msgs
```

El cual indica que se crea el paquete robot con las dependencias que se requieren.

```
catkin_create_pkg <nombre_del_paquete> [dependencia 1] [dependencia 2]
```

Después hay que compilar.

```
catkin_make
```

Instalación EV3DEV

Este sistema operativo se instala en una tarjeta micro SD, no es necesario borrar el sistema operativo original.
De la pagina ev3dev.org se descarga este sistema operativo y se instala de la siguiente manera:
Primero se desmonta la tarjeta micro SD, en este caso dev/sdb1, como súper usuario con el siguiente comando:

```
umount /dev/sdb1
```

Después utilizando los comandos xzcat y dd se descomprime y copia el archivo con salida estándar en la memoria micro SD.

```
xzcat ~/Downloads/ev3dev-yyyy-mm-dd.img.xz | dd bs=4M of=/dev/sdb
```

Una vez que se terminó de realizar la copia se sincronizan los datos en la memoria.

```
sync
```

Por último se retira la tarjeta de la computadora y se coloca en el robot EV3.
Para acceder al robot se utiliza ssh y la dirección IP.

```
ssh root@192.168.1.101
```

La instalación de ev3dev incluye las bibliotecas de Python compatibles con el robot, sin embargo, es necesario instalar la biblioteca RPyC. Con RPyC se ejecutan los programas como si estuvieran instalados en el robot. La instalación se realiza con el siguiente comando:

```
pip install rpyc
```

Instalación biblioteca goprohero

La instalación de la biblioteca goprohero de Python se realiza con el siguiente comando:

```
pip install goprohero
```

Bibliografía

[1] L. Iacono, P. Godoy, O. Marianetti, and C. García Garino, “Estudio de plataformas de hardware empleadas en redes de sensores inalámbricas”, XVI Congreso Argentino de Ciencias de la Computación, 2010.

[2] W. Stallings, “Comunicaciones y redes de computadores”, Pearson Educación, 2004, pp. 580-581.

[3] NASA/JPL-Caltech, “Curiosity - The Next Mars Rover”, ed, 2018.

[4] B. Robotics, “Bluerov2”, 2017.

[5] R. González, F. Rodríguez, and J. L. Guzmán, “Robots Móviles con Orugas Historia, Modelado, Localización y Control”, Revista Iberoamericana de Automática e Informática Industrial RIAI, vol. 12, no. 1, 2015.

[6] M. A. Soler, H. A. Rodríguez, and C. A. Peña, “Desarrollo de un robot explorador operado mediante neuroseñales”, Revista Politécnica, vol. 10, no. 19, 2014.

[7] G. Bermudez, K. S. Novoa, and W. Infante, “La robótica en actividades de búsqueda y rescate urbano. Origen, actualidad y perspectivas”, Tecnura, vol. 8, no. 15, pp. 97-108, 2004.

[8] E. Robotics, “Erle-HexaCopter drone RTF”, ed, 2017.

[9] D. J. W. Andrew S. Tanenbaum, “Redes de Computadoras”, Pearson Educación, México, 2012, pp. 15-46.

[10] D. J. W. Andrew S. Tanenbaum, “Redes de Computadoras”, Pearson Educación, México, 2012, pp. 59-62.

[11] W. Stallings, “Comunicaciones y redes de computadores”, Pearson Educación, 2004, pp. 567-581.

[12] W. Stallings, “Comunicaciones y redes de computadores”, Pearson Educación, 2004, p. 568.
[13] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, “802.11 with multiple antennas for dummies”, ACM SIGCOMM Computer Communication Review, vol. 40, no. 1, 2010.
[14] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, “Digital image processing using MATLAB”, vol. 624: Pearson-Prentice-Hall Upper Saddle River, New Jersey, 2004, pp. 318-321.
[15] (2018). OpenCV. Available: https://opencv.org/
[16] R. E. Schapire and Y. Freund, “A short introduction to boosting”, Journal of Japanese Society for Artificial Intelligence, vol. 14, no. 5, pp. 771-780, 1999.
[17] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1: IEEE, 2001.
[18] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1: IEEE, 2001, p. 2.
[19] “ROS Logo”, ed, 2017.
[20] “Kinetic Kame Logo”, ed, 2017.
[21] “User Guide LEGO MINDSTORMS EV3”, ed: LEGO, 2013.
[22] “Manual de usuario Hero 3+ Black Edition”, ed: GoPro, 2013.

Lista de figuras

Lista de tablas