FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN
SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Firmware para robot de navegación autónoma

Autor:
Ing. Alexis Martin Pojomovsky

Director:
Dr. Pablo De Cristóforis (FCEyN-UBA)

Jurados:
Esp. Ing. Diego Fernández (FI-UBA)
Esp. Ing. Edgardo Comas (CITEDEF/UTN-FRBA)
Esp. Ing. Gerardo Puga (UNLP)

Este trabajo fue realizado en las Ciudad Autónoma de Buenos Aires, entre enero de 2018 y agosto de 2020.

Resumen

La presente memoria describe el desarrollo del robot móvil “Lubobot“, destinado a entornos educativos de grado universitario a través de las distintas etapas de estudio de la materia “Robótica Móvil“, dictada como materia de la carrera de Ingeniería Mecatrónica en la Universidad Nacional de Asunción, Paraguay. Como parte del trabajo se realizó la implementación básica requerida para utilizar este robot con el Sistema Operativo Robótico (ROS).

Se aplicaron diversos conocimientos adquiridos durante la carrera, entre los que se destacan los referidos a sistemas operativos, control de versiones y protocolos de comunicación. Así también, se hicieron uso de patrones de diseño y técnicas de modularización de software.

Agradecimientos

Agradezco el apoyo provisto por mi esposa Laura así como el de mis padres Evelyn y Ernesto, presentes durante todo el proceso de mi formación, siempre alentándome a ir un paso mas allá.
Al equipo docente y colaboradores de la Carrera de Especialización de Sistemas Embebidos, por la dedicación y entrega manifestados durante cada etapa de la cursada.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción general

En este capítulo se introduce al campo de estudio de la robótica móvil. Se aborda una comparación entre diferentes plataformas didácticas comerciales y se exponen el alcance y motivaciones que llevaron al desarrollo del presente proyecto.

1.1. Robótica móvil

La robótica móvil se encarga del estudio de los robots móviles y hace especial hincapié en el desarrollo de capacidades les permitan decidir de manera autónoma cómo, cuándo y a dónde moverse.

En contraste con los robots manipuladores cuya base se encuentra fija con respecto a un sistema de referencia, los robots móviles son aquellos capaces de moverse a sí mismos de un lugar a otro. Esta particularidad les obliga a ser capaces de interactuar con entornos no determinísticos, es decir, propensos a situaciones impredecibles como por ejemplo, una puerta entreabierta, un objeto o persona que obstaculiza el camino, etc.

1.1.1. Tipos de robots móviles

Dependiendo de cómo realizan su locomoción, es posible caracterizar a los robots móviles en los siguientes tipos:

- Robots con patas
- Robots aéreos
- Robots con ruedas

Cada uno de estos tipos plantea su propio conjunto de ventajas y desventajas, así como dificultades para su implementación. En el presente trabajo se hizo énfasis solo en el último tipo de la lista, es decir en robots con ruedas.

1.2. Estado del arte

Existe una amplia gama de robots móviles con ruedas ofrecidos específicamente para el sector académico. A continuación se ofrece un breve sumario de opciones que se encuentran actualmente en el mercado.

1.2.1. TurtleBot

TurtleBot constituye al día de hoy una familia de robots móviles para uso personal de bajo costo. Su uso se extiende tanto a la academia como a roboticistas aficionados en todo el mundo.

Aunque su primera iteración vió la luz en 2010 con un conjunto de características bastante modestas, el lanzamiento de nuevas versiones le aseguró su lugar como dispositivo de referencia dentro de la plataforma ROS.

Muchas de las consideraciones de diseño del robot propuesto en este trabajo fueron tomados de este modelo de robot por lo que se espera que se encuentren similitudes entre ambos al observar la figura 1.1.

FIGURA 1.1. Vista frontal del Turtlebot 2 con un sensor Kinect acoplado.[1]

1.2.2. Clearpath Jackal

El Jackal es un robot móvil 4x4 apto para uso en exteriores. Su robustez lo hace la elección preferida de muchas universidades a la hora de implementar soluciones "de campo", principalmente debido a su resistencia total al polvo y al agua de lluvia.

[1] https://static.generation-robots.com/6739-large_default/turtlebot-2-assembled-clearpathrobotics.jpg

Posee una capacidad de carga de hasta 20 kg, lo que lo hace apto para cargar una importante cantidad de sensores, actuadores y manipuladores, tal como el ejemplo mostrado en la figura 1.2.

FIGURA 1.2. Vista frontal de la plataforma robótica Jackal con una batería de sensores instalados por el usuario.[2]

1.2.3. Fetch Freight 100 Base

Fetch Robotics ofrece con el Freight 100 Base una plataforma robótica para uso en interiores [1]. Esta fue diseñada específicamente para moverse en edificios adaptados a personas en sillas de ruedas que cumplen con la normativa ADA o *Americans with Disabilities Act*[2].

El Freight 100 incluye un sensor del tipo LIDaR 2D, que se puede apreciar en la figura 1.3, lo que lo hace adecuado para tareas de navegación. Además, al estar basado en un robot industrial de carga, la plataforma Freight 100 está diseñada para soportar hasta 100 kg de peso, característica que no solo se impone por un amplio margen respecto a sus competidores del segmento, sino que le posibilita a cargar con un manipulador industrial en su parte superior.

FIGURA 1.3. Vista frontal de la plataforma robótica Fetch Freight donde se puede apreciar su sensor integrado del tipo LIDaR.[3]

[2] https://img.directindustry.es/images_di/photo-g/177123-10073681.jpg

[3] https://www.dymesich.com/wp-content/uploads/2019/06/freight-100-low.jpg

1.2.4. FESTO Robotino

Robotino es un robot móvil comercializado por la compañía alemana FESTO Didactic, el cual está diseñado para entornos educativos, de entrenamiento y de investigación. Esta plataforma dispone de un sistema de tracción omni que como su nombre sugiere, le otorga libertad de movimiento omnidireccional en dos dimensiones.

Asímismo, Robotino viene equipado de fábrica con una computadora tipo PC de grado industrial, lo que facilita su integración con elementos de hardware y software externos tales como cámaras, sensores y actuadores que utilicen el protocolo RS-232, RS-485 o USB. En la figura 1.4 se puede apreciar al robot con una *webcam* acoplada.

FIGURA 1.4. Vista frontal del Robot FESTO Robotino con un sensor externo acoplado.[4]

1.2.5. Pioneer 3-DX

El Pioneer 3-DX es un pequeño robot de tracción diferencial ideal para utilizarse en entornos académicos ya sea de laboratorio o en salón de clases. Este incorpora de fábrica cinco sensores del tipo SONAR al frente del robot como los que se pueden apreciar en la figura 1.5, encoders para las ruedas y un microcontrolador con firmware específico.

FIGURA 1.5. Vista frontal del Robot Pioneer donde se aprecian sus cinco sensores SONAR.[5]

[4] https://www.festo-didactic.com/ov3/media/customers/1100/robotinohome.png

[5] https://static.generation-robots.com/6645-large_default/robot-mobile-pioneer-3-at.jpg

Su amplia superficie de carga le permite mover cargas de hasta 8 Kg, por lo que es un buen candidato para añadir sensores y actuadores extra, tales como cámaras o manipuladores de tamaño adecuado.

1.3. Motivación

Los robots móviles expuestos en la sección anterior representan propuestas comerciales listas para usar que permiten a profesores, investigadores y alumnos concentrarse en el estudio o desarrollo de aplicaciones de la robótica móvil sin la necesidad de diseñar un robot desde cero para cada caso de uso específico. Gracias a esto, las instituciones o individuos que adquieren dichos equipos pueden concentrarse de inmediato en las tareas de interés para el estudio de la materia sin perder el tiempo.

Resulta tentador entonces preguntarse ¿por qué no todas las instituciones adquieren un robot comercial para el laboratorio de robótica?. La respuesta está en los costos asociados a la adquisición de estos equipos. Los robots destinados a investigación poseen precios prohibitivos para la mayoría de las instituciones educativas y por ende, son muy pocas las que se encuentran en condiciones de invertir en ellos.

1.4. Objetivos

Mediante este trabajo se propone una plataforma destinada a la enseñanza de robótica móvil en instituciones educativas de nivel universitario de código abierto y con componentes disponibles en el mercado local paraguayo.

1.5. Alcance

El alcance del presente trabajo involucra el desarrollo del firmware para el microcontrolador encargado de:

- intermediar la comunicación entre la plataforma robótica “Roomba 500” y el framework de robótica ROS (*Robot Operating System*).
- realizar la interfaz entre la unidad de medición inercial MPU6050 con ROS.
- *port* de la librería rosserial para ser utilizada con FreeRTOS y STM32Cube HAL en el lenguaje de programación C++.

Se incluye el desarrollo de software requerido para:

- el paquete para ROS que brinda soporte básico para el robot propuesto "Lubobot", con sus sensores y actuadores.
- calcular la odometría del robot utilizando las lecturas de los encoders.
- la descripción del robot en formato URDF (*Unified Robot Description Format*), requerido para representar correctamente el robot en la herramienta RViz.
- una imagen de Docker con todas las dependencias necesarias para utilizar el robot de manera encapsulada y sin la necesidad de modificar la configuración del sistema operativo del usuario.

- una seccion extra en el repositorio oficial en GitHub con una Wiki con la documentación básica necesaria para iniciarse en el uso de la plataforma

No se incluye con el presente trabajo:

- ningún algoritmo de navegación local ni global
- el código requerido para la conexión al microcontrolador mediante Ethernet
- el diseño de una placa electrónica dedicada

Capítulo 2

Introducción específica

En este capítulo se desglosan las diferentes herramientas tanto de hardware como software, elegidas para el desarrollo del robot propuesto.

2.1. Robot Operating System

Típicamente denominado ROS, es un framework de robótica de código abierto, el cual fue diseñado originalmente para robots de uso académico. Sin embargo, al día de hoy su uso se ha extendido tanto a la industria como al público aficionado.

ROS ofrece un variado set de herramientas que facilitan las tareas del roboticista en tareas tales como paso de mensajes, computación distribuida e implementación de algoritmos para aplicaciones robóticas.

2.1.1. Organización de archivos

Es adecuado considerar a ROS como algo más que un framework de desarrollo y referirnos a el como un meta-sistema-operativo, ya que ofrece no solo herramientas y librerías sino también funciones similares a las de un sistema operativo. Entre ellas, podemos citar su abstracción del hardware, manejo de paquetes y un completo *toolchain* de compilación. Así también, tal como en un sistema operativo "real", los archivos que componen ROS se encuentran organizados en el disco duro de una manera particular, la cual se expresa en la figura 2.1.

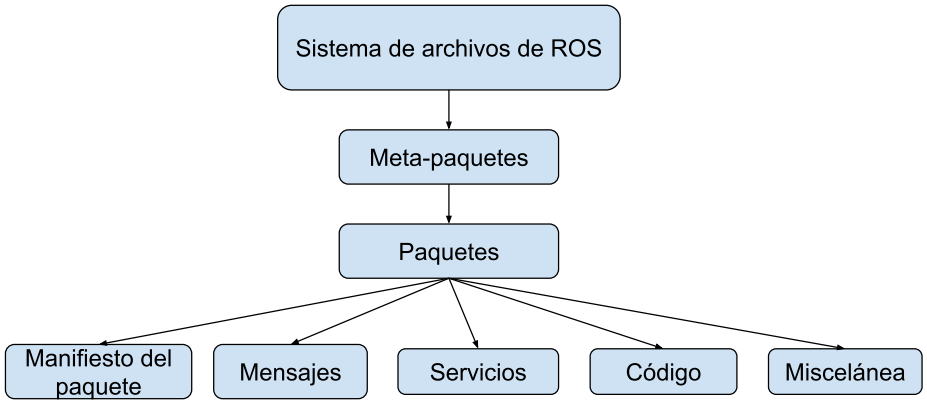

FIGURA 2.1. Organización de archivos en ROS

A continuación se detalla en qué consiste cada uno de los bloques que la componen:

- **Paquetes**: Los paquetes de ROS representan la unidad básica de sofware en la plataforma. Estos contienen uno o mas programas de ROS (nodos), librerías, archivos de configuración, etc, los cuales son organizados como una unidad coherente.
- **Manifiesto del paquete**: Esta representado por un archivo único dentro del paquete y el mismo contiene información sobre el mismo tal como el nombre, autor, tipo de licencia, dependencias, banderas de compilación, etc. El archivo **`package.xml`** encontrado en la raíz del paquete es el manifiesto del mismo.
- **Metapaquete**: Es un paquete que hace referencia a uno o mas paquetes usualmente relacionados entre sí, pero sin estar necesariamente acoplados fuertemente unos con otros. El software proveído con este proyecto es un metapaquete.
- **Manifiesto del metapaquete**: Similar al manifiesto del paquete, con la diferencia que el mismo puede incluir dependencias a de tiempo de ejecución hacia otros paquetes.
- **Mensajes**: Representados con la extensión **`.msg`**, son los tipos de datos utilizados por ROS internamente para comunicar los distintos procesos entre sí. El usuario puede definir tipos de mensajes personalizados con campos adaptados a sus necesidades en del directorio **`msg`** dentro del paquete.
- **Servicios**: Representados con la extensión **`.srv`**, representan interacciones del tipo solicitud/respuesta entre distintos procesos. Los formatos tanto para solicitud como para respuesta pueden definirse en el directorio **`srv`** dentro del paquete.
- **Repositorios**: La gran mayoría de los paquetes de ROS son mantenidos utilizando un sistema de control de versiones (SCV) tales como Git, Mercurial o SVN. Es común encontrar metapaquetes definidos dentro de un mismo único repositorio, el cual es el caso para el repositorio proveído en este trabajo.

2.1.2. Arquitectura interna

ROS esta construido sobre una arquitectura basada en grafos, esto significa que las tareas de cómputo son realizadas a través de una red de procesos llamados nodos. Esta red es denominada *Computation Graph* o grafo de cómputo.

Los principales componentes de este grafo son los nodos, el *master* o maestro, el *parameter server* o servidor de parámetros, además de los mensajes, tópicos, servicios y *bags* o bolsas. Cada uno de estos elementos contribuye al funcionamiento del Computation Graph con una funcionalidad específica. Los elementos que lo componen, mostrados en la figura 2.2, se describen a continuación:

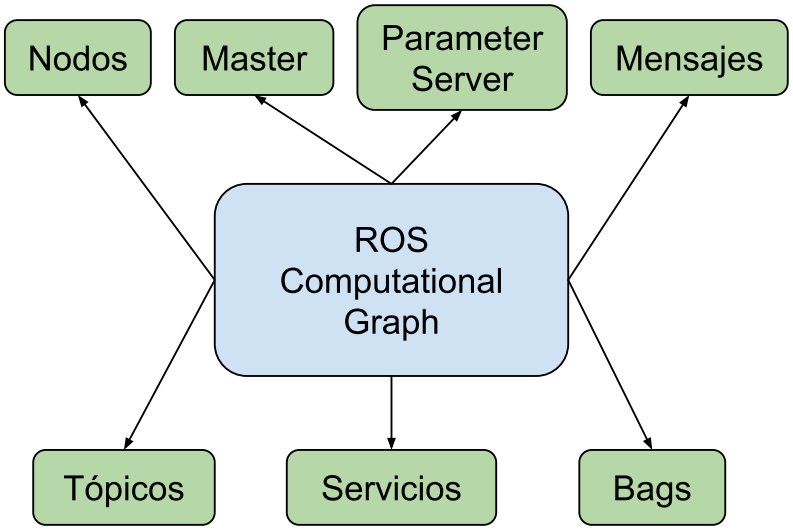

FIGURA 2.2. Estructura del *Computation Graph* de ROS

- **Nodos**: Son los procesos que realizan las tareas de cómputo dentro del robot, los cuales pueden comunicarse unos con otros a través de la API de ROS. Esto resulta particularmente útil cuando distintos nodos necesitan compartir información entre si. En ROS se fomenta el uso de múltiples nodos que realicen procesos sencillos por sobre procesos grandes y complejos que abarquen toda la funcionalidad.
- **Master**: El ROS Master se encarga de buscar y registrar los diferentes componentes que interactúan en el sisteam. Esto posibilita que diferentes nodos sean capaces de "encontrarse" mutuamente, intercambiar mensajes o invocar servicios. En un sistema distribuido, el master debe ejecutarse solamente en una de las computadoras.
- **Servidor de parámetros**: El servidor de parámetros o *parameter server*, permite mantener la información utilizada en configuración de los nodos almacenada en una ubicación central. Esto permite a cada uno de los nodos a acceder y modificar dichos valores.
- **Mensajes**: Los nodos se comunican entre si mediante mensajes, estructuras de datos cuyos campos pueden editarse y permiten ser enviados entre si. Existen tipos de mensajes estandares (enteros, flotantes, booleanos, etc.) así como también es posible definir mensajes propios, adaptados a las necesidades de la aplicación.
- **Tópicos**: Cada mensaje en ROS es transportado utilizando buses llamados tópicos. Cuando un nodo envía un mensaje a través de un tópico, se puede decir que el nodo esta "publicando un tópico". Así mismo cuando un nodo recibe un mensaje a través de un tópico, se puede decir que el nodo esta "subscripto al tópico". El nodo publicante y el suscriptor no tienen información sobre su mutua existencia, por lo que es posible que existan nodos publicando a tópicos sin suscriptores o viceversa, nodos suscriptos a tópicos sin publicantes.
- **Servicios**: En determinadas aplicaciones, el mecanismo de publicador/suscriptor definido en el ítem anterior podría no ser adecuado. Por ejemplo, en

ciertos casos es necesaria una interacción del tipo solicitud/respuesta. En dichas situaciones un nodo podría solicitar la ejecución de un procedimiento rápido por parte de otro nodo y el envío de una respesta con el resultado de dicho cálculo.

- **Logging**: ROS provee un sistema de registro o *logging* denominada Rosbag (bolsa), la cual se utiliza para almacenar información publicada en los tópicos activos, como por ejemplo la proveniente de un sensor que podría ser difícil de generar una y otra vez, pero que a su vez resulta necesaria para depurar determinados algoritmos. Un rosbag permitiría en este caso, generar la información una única vez para luego reproducirla como si fuese una grabación las veces que resulte necesaria.

2.1.3. Herramienta RViz

La herramienta RViz (o ROS Visualization tool), es la herramienta oficial de visualización ROS, la cual permite representar de manera gráfica la información transmitida a través de los distintos tópicos. Posee soporte nativo para la mayoría de los mensajes estándar y permite además, expandir su funcionamiento mediante *plugins* los que posiblitan visualizar mensajes personalizados, entre otros.

RViz permite visualizar la representación física del robot, de modo a analizar su configuración de articulaciones (o *joints*) y enlaces (o *links*), como se muestra en la figura 2.3.

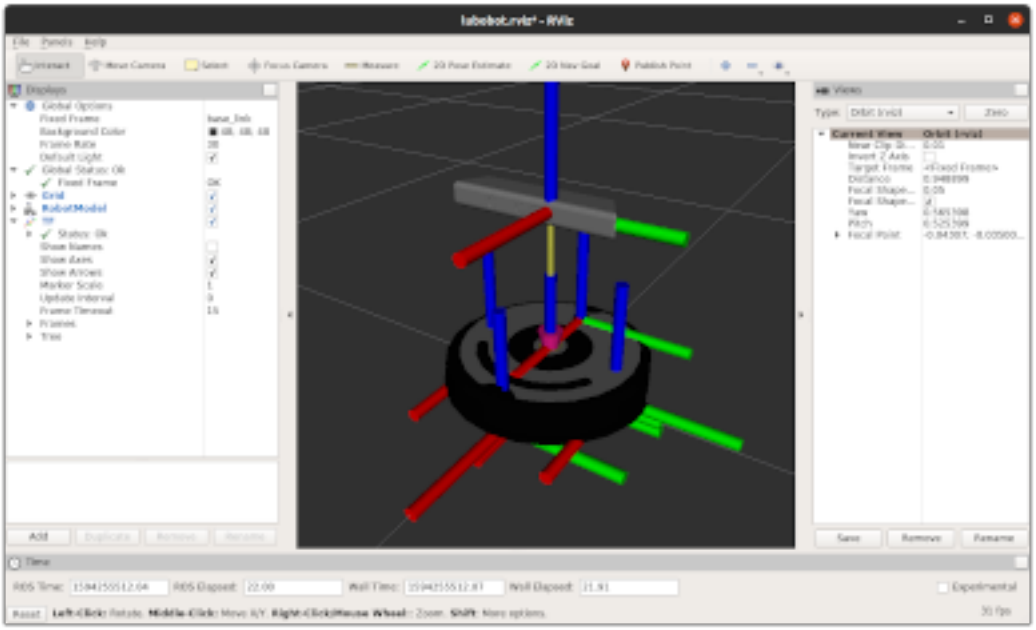

FIGURA 2.3. Interfaz gráfica de RViz mostrando los links y joints del robot Lubobot.

2.1.4. Formato universal de descripción de robots URDF

El formato universal de descripción de robots o *Universal Robot Description System*, comúnmente referido como URDF es un formato estándar para representación de modelos conformados por múltiples piezas conectadas entre sí, como es el caso de brazos robóticos o líneas de ensamblaje. El mismo es ampliamente utilizado

dentro del ecosistema de ROS, plataforma donde vio su origen aunque al día de hoy su uso se ha extendido a herramientas fuera del mismo como MATLAB, la cual permite importar archivos URDF directamente a su *toolbox* de robótica.

2.1.5. Librería rosserial

Es un protocolo para la transmisión serial de mensajes y multiplexación de múltiples tópicos y servicios de ROS sobre un *character device* tal como un puerto UART o un sócket de red. Además de la definición del protocolo de serialización en si mismo, rosserial se compone de otros dos elementos principales:

- **Librerías cliente**: permiten la integración de nodos ROS en diferentes plataformas, apuntando principalmente a sistemas embebidos. Dichas librerias son especializaciones de una clase base, escrita en ANSI C++ para mayor compatiblidad y denominada rosserial_client. Para este trabajo se realizó un *port* de dicha librería a la plataforma STM32CubeHAL.
- **Interfaz con ROS**: Las librerías cliente requieren de un nodo corriendo en la computadora *host* que funcione como puente entre el serie y la red de ROS, encargándose de des-serializar y serializar los mensajes que llegan y se despachan, respectivamente. La participación de los distintos actores involucrados en el uso de rosserial se muestran en la figura 2.4.

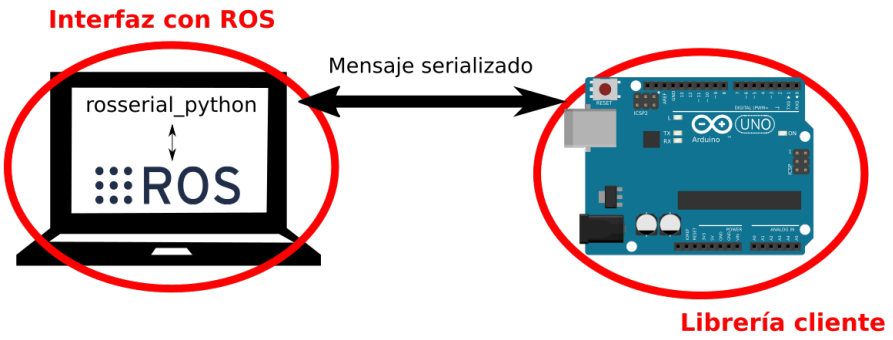

FIGURA 2.4. Interacción entre ROS y rosserial corriendo en un microcontrolador

2.1.6. Paquete de navegación ros_navigation_stack

El paquete de navegación de ROS, usualmente referido como *navigation stack*, es un set de algoritmos que hacen uso de los sensores del robot y la odometría para permitir controlar al robot utilizando un mensaje estándar mientras el mismo se encarga de evitar choques o atascos sin perder su ubicación en el mapa, previamente generado y proveido al mismo.

Para que un robot pueda hacer uso de este paquete correctamente, es necesario que se satisfaga una serie de requisitos:

- Solo puede utilizarse en robots con ruedas en configuración de tracción diferencial u holonómica. Cabe mencionar que el robot presentado en este trabajo es de configuración diferencial.
- Es necesario que el robot publique la información sobre las relaciones entre las posiciones de todas las articulaciones y sensores que lo componen.

- El robot deberá reportar sus velocidades lineal y angular utilizando un mensaje estándar de ROS.
- Un sensor del tipo LIDaR 2D deberá estar presente en el robot para generar el mapa y para el proceso de localización. Alternativamente, es posible utilizar otros tipos de sensores como cámaras de profundidad o *depth cameras* o SONAR, siempre y cuando la información recolectada se publique con el tipo de mensaje adecuado.

En la figura 2.5 se puede apreciar como se encuentra organizado el paquete de navegación, el cual utiliza dos mapas de obstáculos, uno local y otro global, que son construidos a partir de la información generada por el escaner laser en conjunto con el sistema de navegación. El mapa global es un mapa más extenso en dimiensiones y está diseñado para el cálculo de trayectorias global, por otro lado, el mapa local es usualmente un mapa mas acotado pero con mayor detalle y su objetivo es el de evadir obstáculos cercanos.

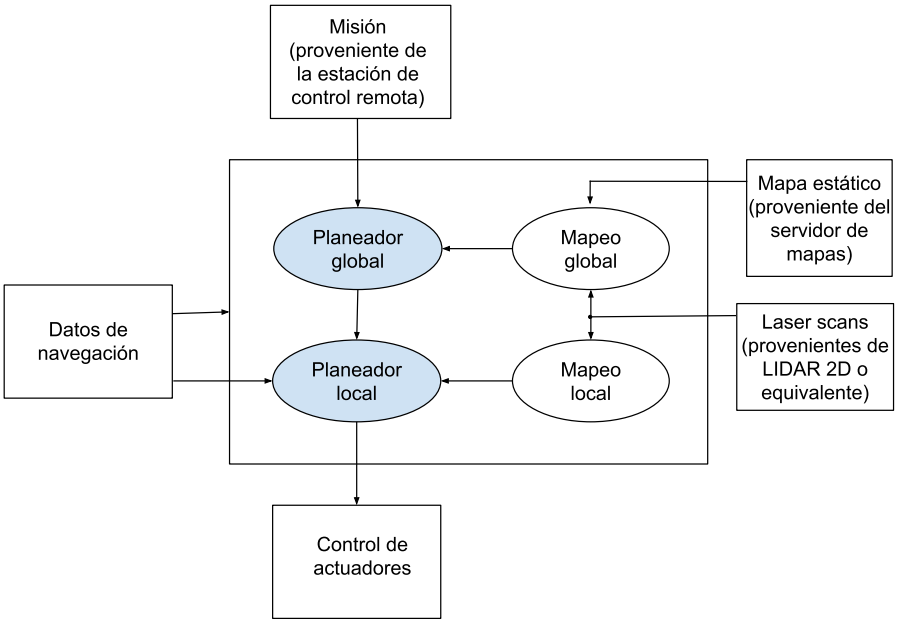

FIGURA 2.5. Configuración típica del paquete de navegación en ROS

Los mapas global y local son utilizados por los planeadores global y local, respectivamente. El planeador global se encarga de generar un plan para ir de un punto a otro del mapa homónimo. Por otro lado, el local se encarga de controlar los distintos actuadores del robot, por ejemplo las ruedas, para llevar a cabo el plan global pero a la vez que esquiva los obstaculos cercanos que puedan aparecer, incluso si los mismos no se encontrasen registrados en el mapa global. Esto resulta especialmente útil a la hora de esquivar obstáculos nuevos, hasta el momento desconocidos.

2.2. iRobot Roomba 500

El Roomba es un robot de limpieza fabricado y comercializado por la compañía iRobot. El primer modelo salió al mercado en el año 2002 y ha recibido siete actualizaciones desde entonces. En cada iteración, se han mejorado aspectos tanto de diseño como funcionalidad y esto le ha permitido mantenerse como el robot de limpieza con mas unidades vendidas en el mundo.

Todos los Roomba incluyen una serie de sensores táctiles, ópticos y acústicos que le permiten detectar obstáculos, residuos, así como escalones o desniveles en el piso.

A nivel de locomoción, se catalogan como robots móviles con ruedas y utilizan el tipo de tracción diferencial, el cual consiste en dos ruedas motrices independientes que le permiten ejecutar giros de 360 grados. Esto es posible sin que el robot deba incurrir en desplazamiento lineal alguno como en el caso de los automóviles, por citar un ejemplo.

La base móvil utilizada para este trabajo consiste en un Roomba de la serie 500 como el mostrado en la figura 2.6, el cual fue introducido al mercado en el año 2007 y se mantuvo en el mercado hasta el 2017. Actualmente es posible conseguir estos equipos en condicion de usado o remanufacturado a precios muy accesibles comparado al precio de un equipo mas actual.

FIGURA 2.6. iRobot Roomba 500, utilizado como base móvil para este trabajo.[1]

2.2.1. Roomba Open Interface

Todos los Roomba lanzados a partir del año 2005 son compatibles con una interfaz de comunicación serial denominada Roomba Open Interface, a la cual es posible acceder mediante la conexión a un puerto físico disponible en la placa madre del robot. Esto habilita al usuario a todo tipo de interacción con el hardware del robot, tales como consultar la lectura de cada uno de sus sensores, así como comandar sus actuadores. Este protocolo ha sido actualizado a la par de de las sucesivas actualizaciones del Roomba para ofrecer nuevas funcionalidades o mejoras y en cada caso, se encuentra acompañado de un documento oficial por parte de iRobot en formato PDF[2].

[1] https://uncrate.com/assets_c/2009/04/roomba-560-stretched-thumb-960x640-3177.jpg

[2] https://cdn-shop.adafruit.com/datasheets/create_2_Open_Interface_Spec.pdf

2.2.2. Conexionado del robot por puerto serie

En la figura 2.7 y la tabla 2.1 se exponen respectivamente, la distribución de pines en el conector y su conexión correspondiente para entablar comunicación con el robot mediante el puerto Mini-DIN.

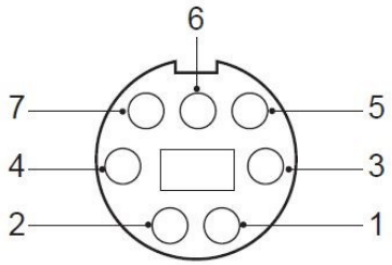

FIGURA 2.7. Distribución de pines en el conector hembra Mini-DIN.

TABLA 2.1. Referencia de pines en conector Mini-DIN hembra

Pin	Nombre	Descripción
1	Vpwr	Positivo directo de la batería (no regulado)
2	Vpwr	Positivo directo de la batería (no regulado)
3	RXD	0 - 5 VCC Entrada serie
4	TXD	0 - 5 VCC Salida serie
5	BRC	Cambio de baud-rate
6	GND	Tierra del robot
7	GND	Tierra del robot

2.3. Placa de desarrollo STM32-NUCLEO

Es una familia de placas de desarrollo elaboradas por la compañía ST. Entre sus características principales se destacan:

- Bajo costo
- Compatibilidad con *shields* de Arduino
- Programador/debugger ST-Link integrado
- Librería del tipo HAL con variados ejemplos de código

De modo a garantizar la escalabilidad del sistema, para este proyecto se optó por una de las placas mas avanzadas de esta familia, denominada NUCLEO-F746ZG y mostrada en la figura 2.8. La misma provee un microcontrolador ARM Cortex-M7 con FPU de doble precisión, 320 kB de RAM, así como 1 MB de Flash. Ofrece además conexión ethernet y una amplia cantidad de GPIOs.

[3] https://www.carminenoviello.com/wp-content/uploads/2015/12/nucleo_144_large-2-660x330.jpg

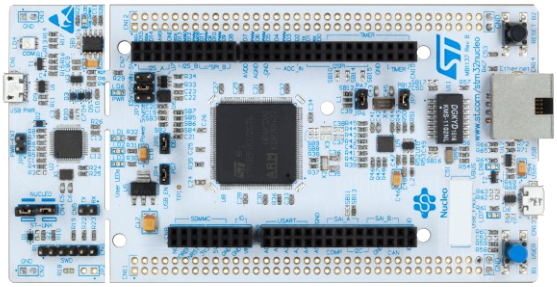

FIGURA 2.8. Placa de desarrollo STM32 NUCLEO-F746ZG elegida para comandar el robot.[3]

2.4. Sensor Kinect 360

El Kinect para Xbox 360 o simplemente Kinect, es un dispositivo de captura de movimiento diseñado por la empresa Microsoft que utiliza tecnología desarrollada por la empresa israelita PrimeSense.

Cuenta con una cámara que captura imágenes en formato RGB, un *array* de micrófonos, así como un sensor de profundidad. Si bien existen otras características que hacen a este dispositivo aún mas interesante, vale la pena detenerse en este último y explicar en qué consiste, puesto que se hace uso extensivo del mismo en la generación de mapas del robot propuesto.

2.4.1. Imagen de profundidad

El sensor de profundidad del Kinect consiste en un proyector de nube de puntos infrarrojos combinado con un sensor CMOS monocromático, similar al de una cámara fotográfica. Funcionando en combinación, estos dos elementos son capaces de captar la información necesaria para que un ASIC genere con ella una imagen de profundidad o *Depth map*, la cual consiste en un mapa de bits con información sobre la distancia relativa entre el sensor y los objetos capturados por el mismo. En la figura 2.10 se puede apreciar una imagen de profundidad generado mediante el sensor Kinect en el que se distingue claramente la silueta de una persona rodeada de diferentes objetos. Las distintas tonalidades de grises visualizadas son una representación de la distancia del objeto al sensor, por lo que para objetos cercanos veremos píxeles de color gris mas claro mientras que para objetos lejanos se verán mas oscuros.

FIGURA 2.9. *Depth map* de la silueta de una persona.

2.5. Unidad de medición inercial

Una unidad de medición inercial o IMU por sus siglas en inglés, es un dispositivo electrónico capaz de medir y reportar la velocidad angular y en algunos casos, el campo magnético que rodea al cuerpo.

Las IMU funcionan detectando la aceleracion lineal mediante uno o más acelerómetros y la tasa de rotación usando uno o más giroscopios. Algunos de estos dispositivos incluyen también un magnetómetro que se utiliza comúnmente como una referencia de rumbo.

Las configuraciones típicas de una IMU continenen un acelerómetro, un giroscopio y un magnetómetro por eje, para cada uno de los tres ejes de giros y fuerzas del vehículo: cabeceo, alabeo y guiñada que se muestran en la figura reffig:imu. Para el caso particular del robot móvil propuesto en este trabajo, solo se hace uso de las aceleraciones: angular en el eje Z y lineal en el eje X, respectivamente.

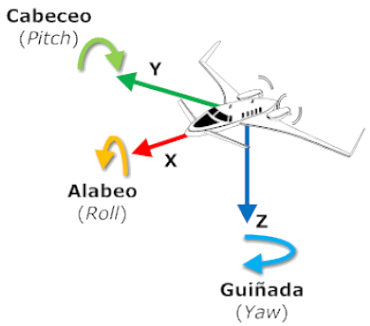

FIGURA 2.10. Ejes de giros y fuerzas de un vehículo.[4]

2.5.1. Sensor MPU6050

El dispositivo MPU-6050 mostrado en la figura 2.11, combina en el mismo chip un giroscopio de 3 ejes con un acelerómetro también de 3 ejes. Mediante lo que el fabricante denomina *Digital Motion Processor* o DMP, esta IMU es capaz de procesar algoritmos de fusión de datos de 6 ejes, usualmente ejecutados en un microcontrolador o DSP mediante un filtro de Kalman o similar. Si bien el chip no incluye un magnetómetro, sí ofrece una interfaz I2C que posibilita conectarla a un magnetómetro externo, de este modo el DMP puede aprovechar la información de campo magnético en en sus algoritmos de fusión para generar resultados más precisos.

[4] http://skiras.blogspot.com/2012/10/4copter-conceptos-i-ejes-giros-y-fuerzas.html

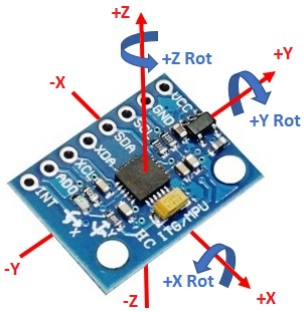

FIGURA 2.11. Sensor MPU-6050 con sus ejes de giros y fuerzas.

Capítulo 3

Diseño e implementación

En este capítulo se exponen las decisiones de diseño tomadas para la elaboración del robot propuesto. Se detallan las dimensiones de las piezas que constituyen el chasis, su esquema de conexiones eléctricas y finalmente la implementación del software y el firmware embebido.

3.1. Estructura mecánica

En esta sección se describe la estructura física del robot y se explican los criterios utilizados tanto para el dimensionamiento de piezas como para su disposición.

3.1.1. Disposición en niveles

Tal como se menciona en la sección 1.2.1, el diseño propuesto toma como referencia a la base móvil *open source* Turtlebot 2. Por este motivo, Lubobot también se compone de una base móvil cilindrica de tracción diferencial y la misma funciona como soporte para una estructura escalable en "niveles", gracias a a lo cual el usuario puede agregar nuevos componentes sin dificultad, según los requisitos particulares de su aplicación. En la figura 3.1 se muestra una representación en 3D de su estructura física en la configuración propuesta.

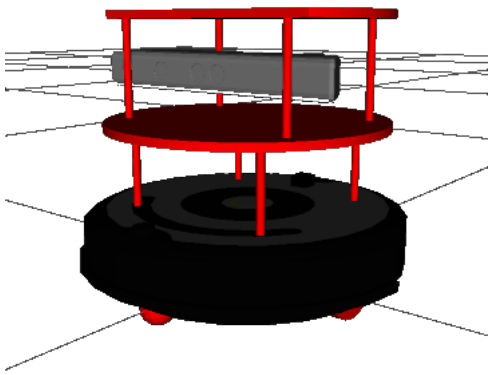

FIGURA 3.1. Vista frontal del modelo URDF de Lubobot representado en RViz.

Sobre la base móvil se montaron cuatro varillas roscadas de 5 mm de diámetro y 80 mm de largo, de las cuales las dos frontales fueron fijadas a la agarradera o *handler* del Roomba, mientras que las dos traseras se fijaron directamente al chasis. Dicha disposición fué calculada en base a las áreas del robot que el autor determinó como seguras para su modificación, es decir, que no representan un riesgo al buen funcionamiento de la base móvil.

3.1.2. Disposición de componentes

Los diferentes componentes funcionales del robot fueron dispuestos en la configuración que se aprecia en la figura 3.3, a excepción de la *laptop*, presente para fines de ejemplo ya que representa un elemento opcional. Para esta disposición se tuvieron en cuenta los siguientes objetivos:

- Mantener bajo el centro de gravedad del conjunto.
- Maximizar el espacio libre en el nivel superior para futuros *upgrades*.
- Posicionar la IMU lo mas cerca posible del centro de rotación de la base.
- Posicionar el sensor Kinect a alrededor de 30 cm del suelo.
- Mantener sin obstrucciones el acceso al botón central del Roomba.

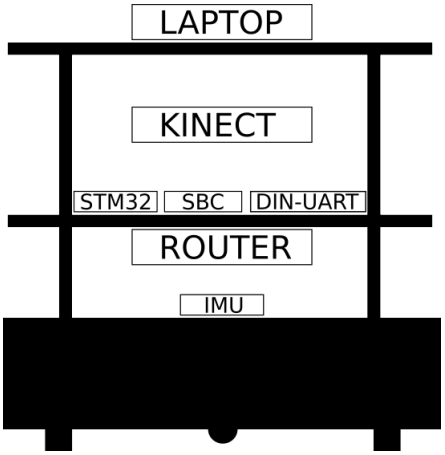

FIGURA 3.2. Distribución de componentes del robot.

3.2. Diagrama en bloques de conexiones

En la figura 3.3 se muestra el diagrama de conexiones entre los distintos componentes electronicos que componen el sistema y se indica además, el protocolo utilizado para cada una de las mismas. Se decidió dotar al robot de un *router* wifi a bordo de modo a facilitar su conexión con la computadora destinada a hacer de estación de control para el envío de misiones.

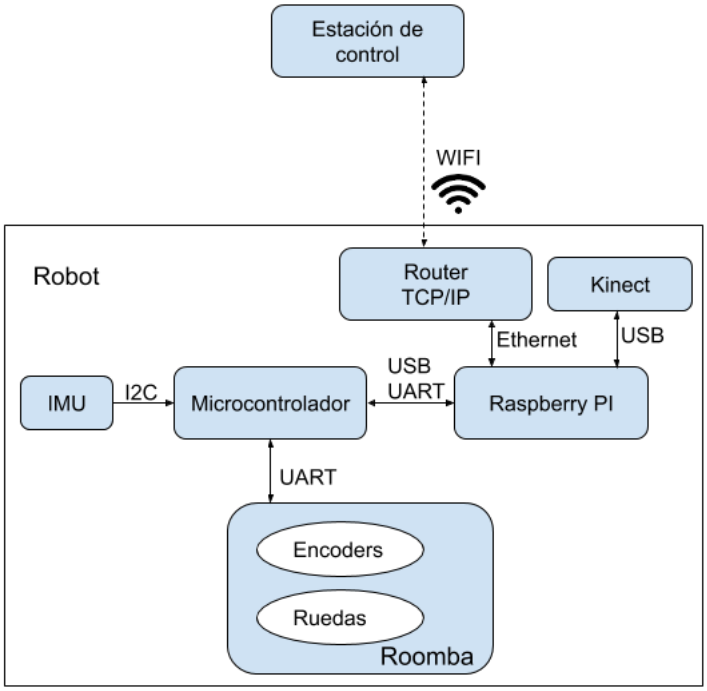

FIGURA 3.3. Diagrama de conexiones de los componentes constitutivos del robot.

3.3. Implementación de firmware

En esta sección se detallan las decisiones de diseño e implementación del firmware, encargado de la interconexión del robot Roomba y la Raspberry PI así como también de la lectura de sensores ajenos a la base móvil original como lo es la IMU.

3.3.1. Distribución de tareas en FreeRTOS

Debido a la necesidad de atender rutinas naturaleza tanto síncrona como asíncrona en el firmware, se decidió utilizar un RTOS para su implementación. Las tareas involucradas se describen a continuación, agrupadas en base a su responsabilidad dentro del sistema.

3.3.2. Interfaz Roomba-microcontrolador

Este conjunto de tareas cumplen la función de *proxy* entre la base móvil Roomba y el resto del sistema. El microcontrolador se ocupa de la serialición y deserialización de los paquetes intercambiados con el robot mediante el protocolo Open Interface, detallado en la sección 2.2.1. Las tareas responsables del manejo de esta función se describen en el apartado siguiente.

- **Solicitud de lectura de sensores**: realiza la lectura de cada uno de los dos encoders disponibles de manera periódica cada 100 ms, actualizando las variables internas de estado en cada iteración con el valor actualizado.
- **Envío de comandos a actuadores**: realiza el despacho de comandos a los distintos actuadores disponibles en base a las órdenes recibidas desde ROS.

Si bien esta tarea puede considerarse de carácter asíncrono, se implementó en la misma un mecanismo de *watchdog* para evitar que un posible fallo en la comunicación con ROS ponga en peligro la integridad del usuario y del robot. Para esto, la tarea verifica cada 100 ms la existencia de un comando de velocidad nuevo. En su ausencia, envía una orden para detener el robot.

3.3.3. Interfaz microcontrolador-ROS

Esta interfaz involucró la migración de la librería rosserial, mencionada en la sección 2.1.5, la cual posibilita entablar una comunicación serial asíncrona entre el microcontrolador y ROS utilizando el protocolo de mensajes estandar del *framework*.

3.4. Implementación de software

3.4.1. Paquete Lubobot para ROS

3.4.2. IMU y estimador de orientación

3.4.3. Odometría basada en encoders

3.4.4. Matriz de covarianza

3.4.5. Configuración del paquete ros_localization

3.4.6. Configuración del paquete de navegación de ROS

Bibliografía

[1] Melonee Wise y col. *Fetch & Freight : Standard Platforms for Service Robot Applications*. 2016.

[2] 101st United States Congress. *Americans with Disabilities Act of 1990*. 1.ª ed. United States Congress, 1990. URL: https://www.ada.gov/pubs/adastatute08.pdf (visitado 09-04-2020).