UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Firmware para robot autónomo utilizando FreeRTOS

Autor:
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 diciembre de 2019.

Resumen

En el presente trabajo se describe el desarrollo de un robot móvil destinado a utilizarse por estudiantes universitarios en el aprendizaje de tópicos de robótica, tales como odometría, localización y navegación.

Esta propuesta se ajusta a dos importantes restricciones: disponibilidad de componentes en el mercado local y precio competitivo respecto a productos comerciales de similares características.

Se incluye además, un repositorio público en Github desde el que se puede acceder al software de ejemplo y documentación necesaria para comenzar a utilizar la plataforma.

Agradecimientos

Índice general

Índice de figuras

Índice de Tablas

Dedicado a... [OPCIONAL]

Capítulo 1

Introducción general

En este capítulo se introduce el campo de estudio de la robótica, contraste entre robótica de manipuladores y móvil, así como la importancia de una planta de pruebas física como motivación para la realización de este trabajo. Asimismo, se presentan los objetivos y el alcance del presente proyecto.

1.1. Robótica móvil

La robótica de manipuladores, también llamados brazos robóticos, se han ganado su puesto como ciudadanos de primera clase en la industria de la manufactura. Difícilmente podríamos al día de hoy, imaginarnos una planta de fabricación en serie que no disponga de estos dispositivos para la realización de tareas repetitivas y de alta precisión.

En la industria electrónica, por citar un ejemplo, los manipuladores son capaces de colocar componentes de montaje superfical con una precisión y velocidad por lejos sobre-humana, haciendo posible la elaboración de teléfonos celulares, computadoras portátiles, etc. Sin embargo, y a pesar de su innegable éxito, estos robots sufren de una desventaja particular: la falta de movilidad. En contraste con un manipulador fijo posee un rango de movimiento limitado que depende del sitio en que el mismo se encuentre instalado, un robot móvil es capaz de moverse a través de la planta, permitiendo el aprovechamiento de sus facultades donde sea que estas sean precisadas. Ante estas limitaciones, en los años noventa surgen los robots móviles.

Una definición correcta de robot móvil plantea la capacidad de movimiento sobre entornos no estructurados, de los que se posee un conocimiento incierto, mediante la interpretación de la información suministrada a través de sus sensores y del estado actual del vehículo.

El principal problema a resolver en un robot móvil es el de generar trayectorias y guiar su movimiento según éstas, en base a la información proveniente de los sensores externos, permitiendo al vehículo desplazarse entre dos puntos cualesquiera del ambiente de trabajo de manera segura, sin colisiones. Esto exige diseñar sistemas de control de trayectorias (posición, dirección, velocidad) en diversos niveles jerárquicos, de manera que el procesamiento de la información proveniente de los sensores externos asegure la mayor autonomía posible.

Los robots móviles operando en grandes ambientes no estructurados deben enfrentarse a significativas incertidumbres en la posición e identificación de objetos. En efecto, la incertidumbre es tal que, trasladarse desde un punto A hasta un punto B es una actividad arriesgada para un robot móvil, una actividad relativamente trivial para un manipulador industrial. En compensación por tener que enfrentarse con más incertidumbres del entorno, no se espera que un robot móvil siga trayectorias o alcance su destino final con el mismo nivel de precisión que se espera de un manipulador industrial (en el orden de las centésimas de milímetro).

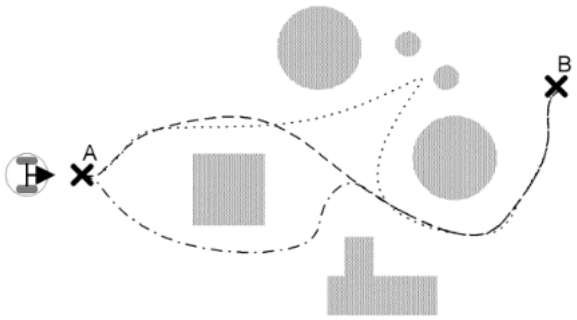

FIGURA 1.1: Algunas de las posibles trayectorias que podría adoptar un robot móvil.

1.2. Motivación

El estudio de la robótica en las universidades de la región se encuentra mayormente avocada a la robótica de manipuladores. Esto tiene sentido desde un punto de vista de oferta y demanda en la industria local, sin embargo, el mercado internacional esta viviendo una fuerte demanda de profesionales capaces de entender y aplicar técnicas de robótica móvil.

Teniendo en cuenta que hasta hace solo un puñado de años atrás, los dispositivos requeridos para llevar a cabo estas tareas hacían que el estudio, desarrollo y aplicación de tareas de robótica móvil sean extremadamente costosos, aún a pequeña escala, la realidad actual es diferente. Hoy, y gracias a la masificación de ciertas tecnologías, es posible armarse de toda una gama de sensores, actuadores y sistemas capaces de procesar el flujo de datos requerido para estas aplicaciones con muy poco dinero, en muchos casos reutilizando componentes originalmente destinados a otros propósitos.

Aún con la importante reducción de precios vivida en los últimos años, las plantas de prueba ofrecidas por empresas internacionales para fines didácticos continúan siendo prohibitivas para muchas instituciones educativas, así como también para la mayoría de los estudiantes. Es por esto que el presente trabajo se desarrolla en torno a la propuesta de una planta de pruebas para el estudio y aplicación de técnicas de robótica móvil. Esta propuesta se centra en dos restricciones importantes, bajo costo y disponibilidad en el mercado local.

1.3. Objetivos

Los objetivos de este trabajo fueron:

- Seleccionar los componentes necesarios para la implementación de un robot móvil de bajo costo con componentes disponibles en el mercado local argentino.
- Generar el código de acceso público y la documentación necesaria para que el robot pueda ser aprovechado, modificado y mejorado por alumnos universitarios de la materia robótica.

1.4. Alcance

Este trabajo se basó en los siguientes lineamientos:

- Los componentes que conforman el robot deben estar disponibles en el mercado local o en su defecto ser fácilmente adquiridos en tiendas online con envío internacional.
- Los componentes de software utilizados, así como los recursos generados a lo largo de este trabajo son gratuitos y de código abierto.
- La plataforma es escalable tanto en términos de componentes físicos como de *software*.

En este trabajo se incluyó:

- Selección de la base móvil, y placa de desarrollo.
- Selección de la cámara de profundidad, IMU y computadora de a bordo.
- Diseño y armado de una superficie de tres niveles para el montaje de los distintos componentes sobre la base móvil.
- Implementación de la librería rosserial y desarrollo del *firmware* de interfaz entre la base móvil iRobot Roomba 500 y ROS, sobre la plataforma STM32F746-NUCLEO.
- Implementación de un nodo ROS para el cálculo de odometría a partir de las lecturas de los encoders.
- Generación de la estructura de archivos de ROS necesarios para lanzar los diferentes módulos de software en simultáneo.

En este trabajo no se incluyó:

- El desarrollo de una base móvil desde cero.
- El desarrollo de un software de navegación para el robot.
- El desarrollo de una placa de circuito impresa dedicada.

Capítulo 2

Introducción específica

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

2.1. Robot Operating System

ROS, el Sistema Operativo Robótico o *Robot Operating System*, es un *framework* de código abierto. ROS esta pensado para funcionar como una plataforma de software común para personas que diseñan y usan robots. La misma permite compartir código y y paquetes de una manera fácil y práctica lo cual permite minimizar el tiempo de desarrollo necesario para lograr que un robot comience a moverse.

2.1.1. Descripción de la plataforma

El ecosistema ROS esta compuesto de los siguientes elementos principales:

1. Un set de drivers que permiten leer informacion desde sensores y comandar actuadores de manera abstraída del hardware y con un formato bien definido. Una gran variedad de hardware popular se encuentra actualmente soportada, incluyendo un enorme número de robots disponibles comercialmente.
2. Una gran y creciente colección de algoritmos de robótica básicos o fundamentales para casi cualquier tipo de robot. Esto permite generar mapas del "mundo", navegarlo, representar e interpretar información generada por sensores, planear trayectorias, manipular objetos entre muhos otros. Gracias a la adopción de la plataforma en la comunidad acadpemica, una gran cantidad de algoritmos experimentales de última generación se encuentran disponibles para ROS.
3. Una infrastructura computacional basada en grafos que permite "mover" información, interconectar los componentes de un sistema robótico complejo, además de incorporar piezas de software personalizadas, tales implementaciones de algoritmos propiios. ROS es inherentemente distribuido, lo que significa que es posible separar las cargas de trabajo de los distintos componentes que conforman el robot entre diferentes computadoras de manera totalmente transparente.
4. Una gran cantidad de herramientas que facilitan la visualización del estado del robot y sus algoritmos, la depuración y el almacenamiento de datos

recolectados o generados. La depuración del software de un robot es una tarea notablemente difícil, por lo que este set de herramientas constituye uno de los diferenciadores que hacen de ROS una plataforma tan potente como la es.

5. Por último, el gran ecosistema de ROS incluye un set extensivo de recursos, tales como una wiki donde se documentan varios de los aspectos del framework, además de una plataforma de preguntas y respuestas muy activa entre los miembros de la comunidad.

2.1.2. Decisiones de diseño

El diseño original de ROS se guió por las siguientes restricciones:

- El *stack* de la aplicación puede descomponerse en varios sub-sistemas independientes, como navegación, visión artificial, locomoción, etc.
- Los sub-sistemas pueden re-utilizarse en distintas tareas, tales como patrullas de seguridad, limpieza, sistemas de *delivery*, etc.
- Mediante la correcta implementación de las capas abstracción de hardware y geometría del dispositivo, la mayor parte del software de aplicación puede reutilizarse en *cualquier* robot.

En la siguiente sección se describen los principales componentes que hacen a un sistema ROS,

2.1.3. El ROS Graph

Estos requerimientos pueden ilustrarse mediante el sistema de representación fundamental de un sistema ROS: El ROS Graph. Un sistema ROS esta conformado por muchos programas distintos corriendo simultáneamente, los cuales se comunican entre sí mediante el paso de mensajes. Resulta conveniente el uso de un *grafo* matemático para representar esta colección de programas y mensajes: los programas se representan mediante *nodos* y los mismos se conectan entre sí mediante *edges* o aristas. Típicamente, los nodos corresponden a procesos POSIX mientras que las aristas a conexiones TCP. Estas aristas representan el flujo de mensajes entre dos nodos, tal como puede apreciarse en la figura 2.1.

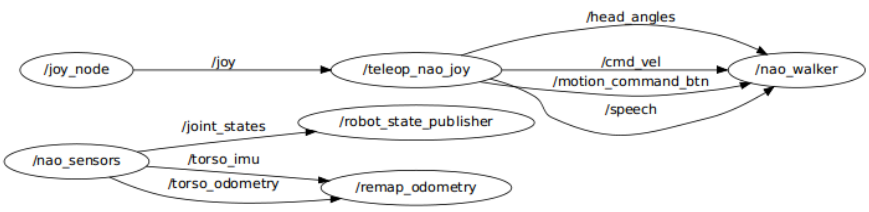

FIGURA 2.1: Ejemplo de un ROS Node Graph. Los nodos en el grafo representan los programas individuales. Las aristas representan el flujo de información entre los diferentes nodos.

Entre otras ventajas, esta arquitectura ofrece un nivel de tolerancia a fallos adicional: un crash en una de las piezas de software típicamente solo echará abajo a su

proceso asociado, mientras que el resto del grafo seguirá funcionando de manera normal. Las circunstancias del fallo podrán luego ser recreadas de manera sencilla grabando los mensajes que entran y salen de ese nodo en particular en un log y luego reproduciéndolo como si se tratase de una pista de sonido.

Hasta este punto se ha asumido que los nodos son capaces de encontrarse unos con otros pero todavía no se ha expuesto como funciona este sistema. La respuesta esta en un programa llamado `roscore`.

Roscore

El roscore es un servicio que provee de la información necesaria a los nodos, de modo a que estos puedan transmitir mensajes unos a otros. Cada nodo se conecta individualmente al roscore al momento de iniciarse para registrar los detalles sobre los mensajes que este publica y los que desea recibir. Cuando un nuevo nodo aparece, el roscore lo abastece con la información que este necesita para conformar una comunicación directa o *peer-to-peer* con otros nodos publicando y suscribiéndose al mismo tópico de mensaje. Cada sistema ROS necesita de un roscore corriendo ya que sin el, los nodos no serían capaces de encontrarse entre sí, aún así, la interacción real entre el roscore y el nodo se realiza solo al momento en que el nodo es lanzado, tal como se demuestra en la figura 2.2.

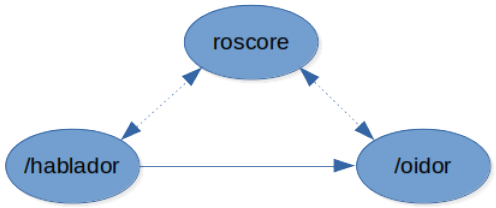

FIGURA 2.2: El roscore se conecta solo de manera efímera con otros nodos en el sistema.

Tópicos

Como ya se vió en el apartado anterior, un sistema ROS consiste en un número de nodos independientes que se relacionan en un grafo. Un nodo por si solo no resulta especialmente útil, pero el conjunto se vuelve realmente potente una vez que los pares empiezan a comunicarse unos con otros e intercambiar información. El canal más típico para este intercambio de mensajes se lleva a cabo a través de tópicos, una de las interpretaciones de las aristas de un grafo. Un tópico es el nombre dado a un canal de comunicaciones capaz de pasar mensajes de un tipo definido. Por ejemplo, la información de un escáner láser 2D podría transmitirse en un tópico llamado `/scan`, con un mensaje del tipo `LaserScan`, mientras que la información de una cámara modría enviarse a través de un tópico llamado `/image`, con un mensaje del tipo `Image`.

2.1.4. Espacios de trabajo o workspaces

Un *workspace* ROS es simplemente un conjunto de directorios dentro de los cuales reside un conjunto de código ROS. En general, un workspace representa un conjunto de paquetes que implementan funcionalidades distintas y que son integrados juntos para una finalidad particular. Es posible tener múltiples workspaces ROS, pero solo es posible trabajar en uno de ellos a la vez, ya que para un paquete solo es posible "ver" código que reside dentro del mismo workspace que él. Por ejemplo, para el desarrollo de este trabajo final se hizo uso de un solo workspace. La estructura típica de un workspace ROS podría verse así:

```
cese_ws/
`-- src
    `-- cese-bot
        |-- CMakeLists.txt
        |-- package.xml
        |-- include
        |   `-- robot_node.h
        `-- src
            `-- robot_node.cpp
```

2.1.5. Paquetes

El software desarrollado utilizando ROS se encuentra organizado en paquetes o *rospackages*, cada uno de los cuales contiene alguna combinación de código y documentación. Estos paquetes funcionan de manera similar a los paquetes que pueden instalarse en Ubuntu vía el comando `apt`. El ecosistema ROS contiene miles de paquetes disponibles en repositorios abiertos, los cuales pueden ser fácilmente instalados para ser aprovechado por cualquier usuario.

Los paquetes se ubican dentro del directorio `src`. Cada uno de estos paquetes debe incluir un archivo *CMakeLists.txt* y un *packages.xml* que describe el contenido de los mismos así como indicar a catkin cómo interactuar con los mismos.

2.1.6. Sistema de compilación catkin

`catkin` es el sistema de compilación defacto de ROS. Este consiste en un set de herramientas que ROS utiliza para generar ejecutables, librerías, scripts e interfaces que pueden ser utilizadas por otras plataformas. Catkin resulta indispensable para trabajar en lenguaje C++ con ROS. Catkin esta formado por un conjunto de macros de `CMake` y scripts de Python que proveen funcionalidad añadida al CMake tradicional de modo a facilitar su uso por parte del roboticista.

2.1.7. RViz

RViz son las siglas de ROS Visualization. Es un entorno de visualización en 3D de propósito general para robots, sensores y algoritmos. Como la mayoría de las herramientas de ROS, puede ser usado para cualquier robot y rápidamente adaptado a una aplicación en particular. RViz es capaz de mostrar una variedad de

tipos de dato transmitidos a través de un sistema ROS con énfasis especial, pero no limitado a, tipos de datos representables en tres dimensiones.

Así como otros sistemas complejos con interfaz gráfica, RViz posee un número de paneles y plugins que pueden ser configurados según necesidad para una tarea en particular, como el mostrado en la figura 2.3. Configurar RViz correctamente puede tomar cierto tiempo, por lo que es posible guardar la configuración del mismo en un archivo para usarse en la posteridad. En este trabajo se incluyen dos de estos archivos de configuración, uno para la generación del mapa y otro para la navegación del robot.

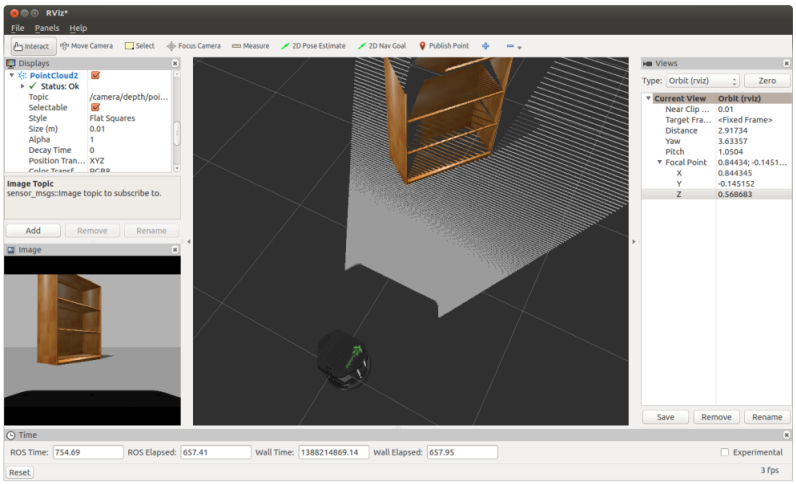

FIGURA 2.3: El panel de RViz cargado con un panel adicional para visualización de cámara durante una simulación.

2.2. iRobot Roomba 500/600

La serie Roomba es una gama de robots domésticos de limpieza, producidos por la empresa estadounidense iRobot. Estos robots se caracterizan por disponer de un puerto serie como el que se aprecia en la figura 2.4, destinado originalmente a procesos de depuración, mediante el cual es posible comunicarse bi-direccionalmente con el robot permitiendo, entre otras cosas, la lectura de sus sensores y el comando de sus actuadores. Esta comunicación se realiza mediante un protocolo propio de la empresa denominado Open Interface[1]que ha sido liberado al público para utilizarse en proyectos de *STEM*, por sus siglas en inglés: ciencia, tecnología, ingeniería y matemáticas.

Existe una variedad importante de robots Roomba, principalmente gracias a que la plataforma posee ya 17 años en el mercado. Por este motivo, resulta importante describir los modelos compatibles con este trabajo: todos los modelos de la serie 500 y 600. Estos, además de incluir el puerto de la Open Interface, disponen de sensores de mayor definición y precisión que la serie 400. Las series posteriores como la 700 en adelante, ya no disponen del puerto Open Interface por lo que debieron quedar descartadas.

[1]https://www.irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf

FIGURA 2.4: Puerto serie de depuración en robot Roomba 550.

2.2.1. Protocolo Open Interface

El protocolo Roomba Open Interface (OI), es una interfaz para controlar y manipular el comportamiento del robot Roomba. Esta interface permite manipular el comportamiento del robot, así como leer el estado de sus sensores a través de comandos serie que uno envía al robot conectando el puerto a una PC o microcontrolador mediante un conector Mini-DIN.

2.2.2. Conexión física

Para utilizar el OI; un procesador capaz de manipular un puerto serie, tal como una PC o microcontrolador, debe conectarse al conector externo Mini-DIN del que dispone la Roomba. Este conector ofrece una comunicación serie bi-direccional con niveles de tensión TTL (0 - 5 VCC). El mismo provee además de un pin de tensión no-regulada, conectada directamente a la batería de la Roomba. Para acceder a este puerto es necesario remover primeramente la careta plástica de la parte superior del robot.

Distribución de pines del conector Mini-DIN

En la figura 2.2 se muestra la distribucion de pines del conector hembra de la Roomba y en la tabla 3.1 se describen cada uno de los mismos.

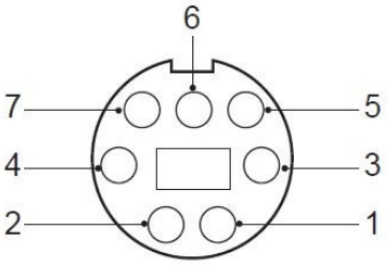

FIGURA 2.5: Distribución de pines en el conector hembra Mini-DIN.

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

Pin	Nombre	Descripción
1	Vpwr	Directo de batería + (no regulado)
2	Vpwr	Directo de 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 desarollo NUCLEO-F746ZG

La placa NUCLEO-F746ZG es un producto ofrecido por el fabricante ST como una plataforma de prototipado rápido basado en el microcontrolador STM32F746ZG. La misma se caracteriza por contar con un controlador Ethernet integrado, un programador-debugger que además puede funcionar como interfaz RS-232/USB, especialmente útil para tareas de desarrollo y depuración.

Este microcontrolador dispone de un núcleo ARM Cortex-M7 que corre a frecuencias de hasta 216 MHz. Posee además, 1 Mbyte de memoria tipo flash, y 320 KB de memoria RAM. Destaca por su gran cantidad de periféricos, los cuales son fácilmente configurables a través de la plataforma STM32CubeMX, una interfaz gráfica de configuración para microcontroladores ST que permite, además de inicializar periféricos, integrar middlewares de terceros tales como FreeRTOS, LwIP, etc., con una simplicidad sin precedentes. En la figura 2.6 se puede apreciar una de las pantallas de configuración de la aplicación.

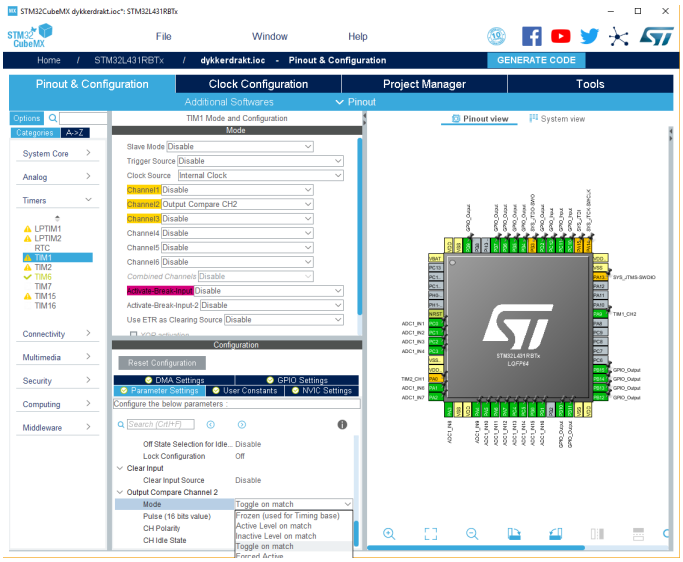

FIGURA 2.6: Configuración de un microcontrolador a través de la aplicación STM32CubeMX.

Resulta importante destacar que esta aplicación permite la exportación de proyectos pre-generados hacia una amplia variedad de IDEs comerciales y gratuitos, además de una opción basada en Makefiles, especialmente útil para su uso en

flujos de trabajo más "minimalistas", como los basados en editores de texto avanzados y línea de comandos.

2.4. Computadora OrangePI PC

La OrangePI PC es una *Single Board Computer* de hardware abierto, es decir que tanto sus esquemáticos como el firmware de la misma se encuentran disponibles de manera pública. Este dispositivo es, en factor de forma y características, similar a la ampliamente conocida Raspberry PI, pero con un precio de venta aún mas moderado, lo que la vuelve una alternativa interesante para este trabajo, en que se busca reducir costos.

Esta placa disponde de un microprocesador Allwinner de 32bit y 4 núcleos, y cuenta con 1 GB de memoria RAM DDR3. Es compatible con una amplia gama de sistemas operativos y distribuciones, y para este trabajo en particular, se decidió utilizarla con Ubuntu 14.04. Esta decisión viene ligada a que ROS Indigo, la versión de ROS compatible con este sistema, es la más ampliamente difundida y la que dispone de una mayor cantidad de paquetes.

FIGURA 2.7: Vista superior de la SBC OrangePI PC, alternativa económica al Raspberry PI.

En la figura 2.7 se muestra una vista superior de la placa OrangePI PC donde se pueden apreciar sus conectores y layout, muy similares a los de la Raspberry PI 2. Para este trabajo, se hizo uso del conector ethernet así como de 2 de sus 3 puertos USB 2.0.

2.5. Sensor Kinect para Xbox 360

El Kinect para Xbox 360 o simplemente Kinect, es un dispositivo de captura de movimiento diseñado por la empresa Microsoft, utilizando la tecnología desarrollada por la empresa israelita PrimeSense. El dispositivo cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado, que proporcionan, al utilizarse en una consola Xbox 360, captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y capacidades de reconocimiento de voz.

El sensor de profundidad es un proyector de infrarrojos combinado con un sensor CMOS monocromo que permite a Kinect "ver" la habitación en 3D incluso en condiciones de ausencia total de luz.

Gracias al desarrollo de un conjunto de controladores open source llamado freenect[2], el uso del sensor Kinect se ha extendido a sistemas operativos del tipo GNU/Linux y con este, su utilización en ROS.

Salvando distancias entre una cámara de profundidad y un sensor LIDAR, en determinados escenarios es posible remplazar un costoso sensor LIDAR con un Kinect, procesando el mapa de puntos en 3D y generando un LaserScan en 2D a partir de uno de los múltiples planos que lo componen. Esta información ya procesada, puede utilizarse para alimentar paquetes de ROS diseñados para consumir datos generados por un LIDAR.

[2] https://github.com/OpenKinect/libfreenec

Capítulo 3

Ensayos y Resultados

En este capítulo se detallan las pruebas realizadas tanto de manera individual, para comprobar que tanto los componentes de *hardware* como de *software* funcionen como se espera, así como las pruebas de integración realizadas entre los componentes individuales y el *framework* ROS.

3.1. Pruebas funcionales del hardware

Aquí se describen las pruebas que se realizaron individualmente a cada uno de los componentes del robot.

3.1.1. Sensor Microsoft Kinect 360

El Kinect es un dispositivo complejo que posee varios componentes internos, todos con diferente identificador USB, que se conectan mediante un cable único. Este cable posee un conector propietario especial que implementa, además de los hilos del estándar USB 2.0, uno de alimentación a 12 VCC y su respectiva tierra. Por cuestiones de disponibilidad y precio, se procedió a remover manualmente este conector y añadir manualmente un conector estándar USB macho y un plug de alimentación a 12 VCC.

Para testear adecuadamente el funcionamiento del sensor con la adaptación casera, se procedió a conectar el cable USB a un PC con Linux y la alimentación a una fuente externa de 12 VCC. Aquí, se utilizó el comando lsusb, disponible como parte del paquete usb-utils en Ubuntu, y se procedió a listar los dispositivos USB advertidos por el sistema operativo:

Una vez corroborado esto, se procedió a verificar que la nube de puntos se esté generando correctamente. Para esto se utilizaron las siguientes herramientas y paquetes de software:

- libfreenect: librería de código abierto para el sensor Kinect.
- rqt_image_viewer: herramienta integrada de ROS para visualizar imágenes RGB y nube de puntos.

3.1.2. Unidad de medición inercial InvenSense MPU6050

Para corroborar el correcto funcionamiento de la IMU se procedió a conectarlo a un Arduino MEGA2560 y se utilizó la librería Adafruit_MPU6050, la cual se encuentra activamente mantenida por la compañía Adafruit.

Se procedió así, al envío de información RAW via puerto serie y se procedió a graficar, individualmente, cada uno de los componentes de los vectores de aceleración y giro proveídos por el sensor.

3.2. Pruebas funcionales del software

Aquí se describen las pruebas que se realizaron individualmente a cada uno de los módulos de software que componen el sistema.

3.2.1. Envío y recepción de datos entre el robot y microcontrolador

Al tratarse esta de una base móvil comercial, y gracias a la existencia de un documento oficial publicado por el fabricante con información detallada sobre el protocolo de comunicación del robot, se dispone de la gama completa de comandos para manipular los actuadores, así como para leer los sensores disponibles.

Mediante las pruebas descriptas a continuación, se procedió a verificar el subconjunto de periféricos que fueron implementados en el microcontrolador que actúa como interfaz entre el robot y el sistema ROS, corroborando que los mismos funcionen tal y como se describen en el documento mencionado anteriormente.

Para el envío de órdenes desde el microcontrolador hacia el robot, se analizó la respuesta por parte del mismo ante una serie definida de comandos, ante los cuales se corroboró de manera manual que la respuesta sea la esperada.

3.2.2. Publicación y recepción de mensajes ROS en el microcontrolador

El proceso en el que se probó la correcta publicación de mensajes desde el microcontrolador hacia ROS, involucró la utilización de las siguientes herramientas integradas del mismo:

- rosserial_python serial_node
- rostopic list
- rostopic echo
- rostopic hz
- RViz

Para verificar la existencia de los tópicos anunciados por el microcontrolador fué necesario, primeramente, ejecutar el nodo serial_node, el cual se encarga de la serialización y des-serialización de los mensajes transmitidos mediante UART o Ethernet, actuando de puente entre el protocolo intermedio de rosserial y ROS.

TABLA 3.1: Sub-conjunto de comandos utilizados para testear el correcto funcionamiento del la base móvil

Comando	Descripción	Verificación
START	Se inicializa el protocolo Open Interface	El robot deja de ignorar comandos
STOP	Se finaliza el protocolo Open Interface	El robot empieza a ignorar comandos
SAFE	El robot pasa a modo seguro	El robot deja de moverse si es levantado del suelo
FULL	El robot pasa a modo full	El robot no deja de moverse a menos que reciba explícitamente un comando STOP
DRIVE DIRECT	El robot recibe comandos de velocidadpara cada rueda	El robot empieza a moverse inmediatamente después de recibir el comando
SENSORS	Se solicita la lectura del estado actualde un sensor	El robot responde con el mensaje correspondiente al sensor solicitado

Luego de haber ejecutado el nodo rosserial en la computadora host, fué posible verificar la existencia de los tópicos invocando el comando 'rostopic list', el cual despliega una lista de los tópicos que se encuentran anunciados.

Una vez conocida la lista de tópicos, es posible reproducirlos en la consola mediante el comando 'rostopic echo' así también, es posible verificar su frecuencia de publicación mediante el comando 'rostopic hz'.

Luego de verificar la existencia de los mensajes en cada tópico, es posible graficarlos utilizando la herramienta RViz. Esto resulta especialmente útil para mensajes compuestos por varios campos, como los del tipo IMU.

3.3. Pruebas de integración

Estas pruebas integran las tareas de alto nivel que se pretende puedan ser utilizadas por los usuarios que adopten la plataforma. Estas involucran la interacción entre todos los componentes del sistema en simultáneo, así como su interacción con el sistema ROS.

Las pruebas se encuentran ordenadas de manera tal que las primeras formen parte de los requisitos para las siguientes y así respectivamente, concluyendo con la prueba de integración

3.3.1. Estimador de pose a partir de IMU

Mediante el nodo imu_complementary_filter, integrado como parte del paquete imu_tools[1], se procede a procesar la información *raw* proveniente del la IMU. Este nodo provee una estimación de la orientación del robot a partir de lecturas de

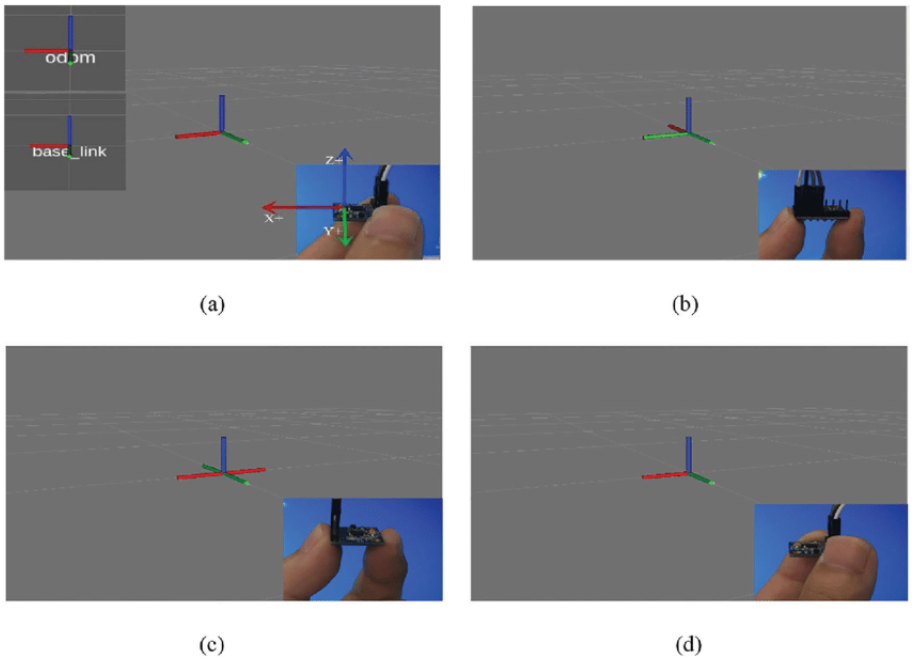

FIGURA 3.1: El ángulo *yaw* mostrado en RViz. (a) Posición inicial. (b) Posición rotada a 90 grados. (c) Posición rotada a 180 grados. (d) Posición rotada a 180 grados en sentido anti-horario.

velocidad angular y aceleraciones, sin la necesidad de un magnetómetro. A su salida, est nodo publica mensajes estándares del tipo sensor_msgs/Imu[2], requeridos por el nodo encargado de la localización del robot.

3.3.2. Tele-operación del robot

Esta prueba consistió en operar el robot mediante un nodo de ROS, el cual se encarga de publicar mensajes del tipo geometry_msgs/Twist[3]. Este mensaje es recibido por el robot y aplica los comandos de velocidad a cada una de las ruedas. Para esto se utilizó el nodo teleop_twist_keyboard:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Este nodo permitió comandar el movimiento del robot de forma manual utilizando el teclado de la computadora mediante el uso de los siguientes comandos:

```
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
```

[2]https://github.com/ccny-ros-pkg/imu_tools

[2]https://docs.ros.org/melodic/api/sensor_msgs/html/msg/Imu.html

```
e/c : increase/decrease only angular speed by 10%
anything else : stop

CTRL-C to quit
```

Esta prueba se consideró satisfactoria el verificar que el robot era efectivamente controlado mediante este nodo. Las velocidades deseadas o *set-points* se verificaron recién durante las pruebas de odometría.

3.3.3. Odometría

Esta prueba esta basada en el mismo procedimiento utilizado para tele-operación del robot, con la diferencia que se procedió, además, a comprobar el cálculo de velocidad basado en la lectura arrojada por los encoders. Para esto se utilizó el nodo calculador de odometría elaborado por el autor en lenguaje C++, incluido en el repositorio del presente trabajo.

Para realizar la prueba, se procede de la misma manera que para la de tele-operación, con la diferencia de que se abre, además, el nodo de odometría antes de enviar los comandos de velocidad.

Para lanzar el nodo de odometría utilizamos el siguiente comando:

```
rosrun odom_publisher odom_publisher
```

Una vez listo, se hace un eco del tópico de odometría, el cual arrojará los valores publicados en la consola. La consigna es comparar los valores de velocidad publicados en el tópico de odometría con los comandos enviados desde el nodo `teleop_twist_keyboard`, debiendo ser éstos similares.

3.3.4. Generación de mapa

Para generar un mapa se utilizó la herramienta gmapping[4], ofrecida como un paquete de ROS. Esta herramienta es capaz de generar un *occupancy grid* en 2D, muy similar a un plano civil, a partir de las lecturas de *laserscan* generadas por el sensor Kinect, mas la información de odometría colectada por el robot. En la figura 3.2 se puede apreciar como el mapa es generado a partir de las lecturas del *laserscan*. Por el contrario en la figura 3.3, se aprecia el mapa final generado satisfactoriamente.

3.3.5. Localización sobre mapa pre-existente

Esta prueba consistió en colocar el robot en un punto arbitrario del mapa sin proveerle información previa sobre su localización. Aquí el robot debe ser capaz de utilizar sus sensores para reconocer su entorno cercano y reconocer características conocidas entre este y el mapa, siendo capaz de estimar su ubicación. Para este procedimiento se utilizó el mapa generado en el apartado anterior en conjunto con la aplicación RViz para visualizar el estado del procedimiento.

[3] https://docs.ros.org/melodic/api/geometry_msgs/html/msg/Twist.html

[4] https://wiki.ros.org/gmapping

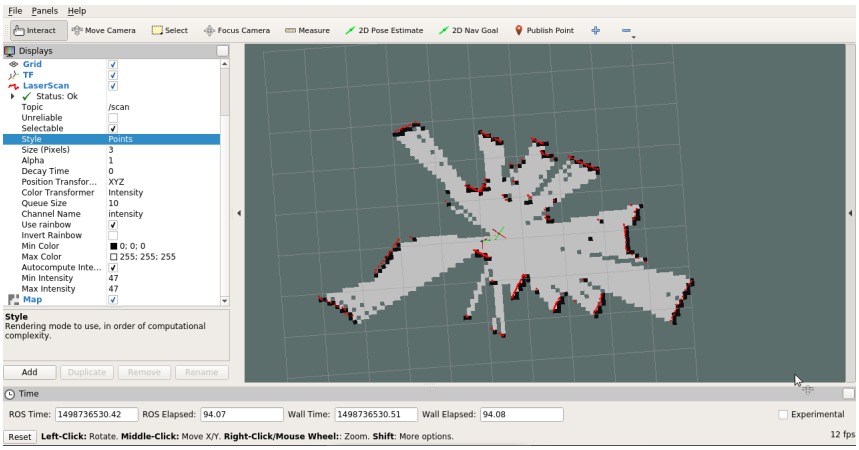

FIGURA 3.2: Visualización en RViz de un mapa en proceso de generación

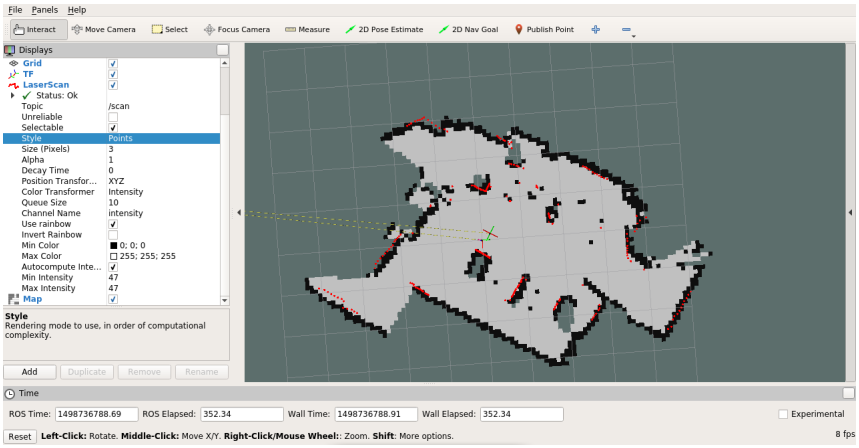

FIGURA 3.3: Visualización en RViz de un mapa terminado

3.3.6. Navegación sobre mapa pre-existente

La navegación sobre un mapa pre-existente esta basada en todas las pruebas de integración anteriores y representa el punto culminante de la serie. Aquí se utilizó la aplicación RViz no solo para visualizar el mapa y la localización del robot, sino también para el envío de puntos de consigna dentro del mapa a los que el robot debe ser capaz de llegar utilizando toda su batería de sensores y estimadores a la vez que evita colisionar con las paredes y objetos presentes.

Capítulo 4

Conclusiones

En este capítulo se describen los aportes generados con el trabajo realizado, detallando los logros obtenidos. Así también, se especifican las técnicas mediante las cuales se hizo posible la ejecución del proyecto. Por último, se deja constancia de las metas originales que no pudieron ser abordadas dentro del alcance final, identificando las propuestas de acción a futuro.

4.1. Conclusiones generales

En este trabajo se completó la primer iteración en el ciclo de diseño e implementación de una planta de pruebas destinada a utilizarse en el aprendizaje de técnicas de robótica móvil a nivel personal o universitario.

Se pudo desarrollar una planta de pruebas completa y funcional en base a componentes disponibles en el mercado local, integrándola con el framework de robótica ROS. Se pudo desarrollar un firmware capaz de funcionar como una interfaz entre una base móvil comercial y ROS, así como interfacear los sensores necesarios con dicho framework de modo a posibilitar su uso inmediato en aplicaciones de navegación autónoma. Este desarrollo sienta las bases y estructura para una segunda iteración, dejando abiertas una serie de propuestas con mejoras que pueden implementarse en base a las necesidades que se le vayan presentando al usuario.

A lo largo del desarrollo de este trabajo final, se aplicaron extensivamente los conocimientos adquiridos durante la carrera. Mas allá de que el conjunto de asignaturas posibilitaron la apreciación de un panorama muy completo en lo que refiere a diseño y ejecución de un proyecto de sistemas embebidos, se destacan a continuación aquellas materias cuyo impacto en el desarrollo de este trabajo:

- Protocolos de Comunicación: se aplicaron los conocimientos adquiridos en la utilización de protocolos UART e I2C. Así también, se implementó la librería rosserial, la cual permite ser utilizada de manera intercambiable tanto con comunicación UART como Ethernet, utilizadas durante la etapa inicial y final del proyecto, respectivamente.
- Sistemas Operativos de Tiempo Real (I y II): se utilizaron los conocimientos adquiridos aplicando FreeRTOS como sistema operativo para el sistema propuesto.

- Arquitectura de microprocesadores: resultó necesario tener conocimientos sobre la Arquitectura ARM Cortex M para la programación de la plataforma STM32 NUCLEO y el uso efectivo de sus periféricos.
- Programación de microprocesadores: se aplicaron las buenas prácticas de programación que fueron mostradas a lo largo de la materia. Se empleó un formato de código consistente a modo de obtener código más modular, legible y reutilizable.
- Gestión de Proyectos en Ingeniería: la elaboración de un Plan de Proyecto para organizar el Trabajo Final, el cual facilitó en gran manera la ejecución del mismo y evitó demoras innecesarias.

Por lo tanto, se concluye que los objetivos planteados al comienzo del trabajo han sido alcanzados satisfactoriamente, habiéndose cumplido con los criterios de aceptación del sistema final y obtenido conocimientos valiosos para la formación profesional del autor.

4.2. Trabajo a futuro

Se plantea a continuación una lista de mejoras que podrían implementarse a fin de aumentar las capacidades del robot, las cuales se espera puedan ser abordadas por los usuarios que hayan decidido adoptar la plataforma, a quienes se exhorta a hacer aportes públicos al repositorio de modo a que todos puedan verse beneficiados con los mismos:

- Agregar soporte para sensores y actuadores disponibles en el robot Roomba que aún no han sido implementados en el sistema.
- Agregar un magnetómetro a la IMU via SPI o bien, reemplazarla por una IMU que ofrezca 9 grados de libertad, como la MPU9250, de modo a que la estimación de la pose del robot sea mas precisa.
- Utilizar la salida de potencia del controlador del motor de la aspiradora para alimentar el sensor Kinect, eliminando la necesidad de una batería externa.
- Reemplazar el actual computador OrangePI PC con un Nvidia Jetson Nano o similar, de modo a poder procesar la información proveniente desde una mayor cantidad de sensores.
- Implementar un segundo sensor Kinect apuntando en dirección contraria al actual para duplicar el campo de visión.
- Agregar soporte para ROS 2.