



UNIVERSIDAD POLITÉCNICA DE JUVENTINO ROSAS

T E S I S

“Algoritmos desarrollados con las librerías de ROS para la
gestión de dispositivos externos”

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN REDES Y TELECOMUNICACIONES

Presenta:

Alejandro VILLAFRANCO DESIDERIO

Asesores:

M.I. Luis Rey Lara Gonzaléz

M.I. Juan Heriberto Gallegos Galindo

Juventino Rosas, Gto. 19 de Septiembre de 2020.

Agradecimientos

Agradezco a mis padres José Guadalupe Villafranco Alcantara y María Guadalupe Desiderio Silva que siempre me apoyaron incondicionalmente en la parte moral y económica para llegar a ser un profesional, eternamente agradecido por todo el amor que me han dado y por esta educación que ustedes me han podido brindar con todos sus esfuerzos.

A mis hermanos y familia en general por el apoyo y amor que siempre me brindaron día con día en el transcurso de cada año de mi carrera Universitaria.

A mis profesores que día con día dieron lo mejor de ellos por brindarme los mejores conocimientos que ellos podrían dar. En especial a mi asesor Luis Rey Lara González por su arduo trabajo de transmitirme sus diversos conocimientos así mismo por su guía, apoyo e ideas que me motivaron a seguir con mis estudios

Tú, quien ha sido un gran apoyo en el transcurso de esta etapa en mi vida, te agradezco por tu desinteresada ayuda, por tu gran paciencia al soportarme, por echarme una mano cuando siempre lo necesite, no solo te agradezco por la ayuda brindada, sino por todos los buenos momentos que tuvimos juntos. Encantado de tenerte a mi lado como una gran amiga, muchas gracias Aranzazú Mahelet Téllez Castellanos.

Dedicatoria

Dedico con todo mi corazón mi tesis a mis padres, pues sin ellos no lo habría logrado. Su amor y bendición a diario a lo largo de mi vida me protege y me lleva por el camino del bien, por eso les dedico mi trabajo por su amor y paciencia, los amo

Resumen

La presente tesis muestra una investigación sobre la gestión de sistemas embebidos haciendo uso de las librerías de ROS. Así mismo se demuestra el uso de estas librerías en un entorno de Ubuntu instalado en ordenador utilizando simulaciones 2D y 3D para observar el funcionamiento de este framework que nos ofrece ROS.

Se realizan las pruebas para observar el comportamiento de los nodos tanto en el entorno de Ubuntu y Raspberry individualmente, utilizando ejemplos de prueba que son proporcionados por las librerías.

De igual manera se realiza la configuración de ROS master node para que pueda ser compartido por la red, brindando la posibilidad de conectar varios clientes a un robot o viceversa según sea el caso, en esta investigación se utiliza una Raspberry Pi que será el sistema embebido a controlar, después de configurado el entorno de red, se muestran las pruebas realizadas para comprobar el funcionamiento de este, siendo así capaces de controlar y demostrar como se comunican los nodos a través de la red haciendo uso de simulaciones.

Abstract

The present thesis shows an investigation on the management of embedded systems using ROS libraries. Likewise, the use of these libraries is demonstrated in an Ubuntu environment installed on a computer using 2D and 3D simulations to observe the operation of this framework that ROS offers us.

Tests are performed to observe the behavior of the nodes both in the Ubuntu and Raspberry environment individually, using test examples that are provided by the libraries.

In the same way, the ROS master node configuration is carried out so that it can be shared by the network, offering the possibility of connecting several clients to a robot or vice versa as the case may be, in this investigation a Raspberry Pi is used that will be the embedded system to be controlled, after configuring the network environment, the tests carried out to check its operation are shown, thus being able to control and demonstrate how the nodes communicate through the network using simulations.

Índice general

Índice de figuras	VIII
1. Introducción	1
1.1. Problemática	2
1.2. Justificación	2
1.3. Objetivos	3
1.3.1. Objetivo General	3
1.3.2. Objetivos específicos	3
2. Estado del arte	4
2.1. Sistema de adquisición de datos de una unidad de navegación inercial y ROS como herramienta de visualización.	5
2.2. Modelado, simulación y control de pequeños vehículos submarinos no tri- pulados	6
2.3. Diseño de Algoritmos de Control de Nivel Bajo en Vehículo Inteligente . .	7
3. Marco Teórico	8
3.1. Sistemas embebidos	8

3.2. ROS	9
3.2.1. Herramientas de simulación Gazebo y RViz proporcionadas por ROS	9
3.2.2. Definición de algunas palabras utilizadas en ROS	10
3.2.3. Licenciamiento de ROS	11
3.2.4. Distribuciones de ROS	11
3.3. Algoritmo	14
3.3.1. Partes de un algoritmo	14
3.4. C++	15
3.5. Raspberry	16
3.6. Arduino	17
3.7. Sistema operativo	18
3.8. Software de código abierto	19
3.9. Rufus	20
4. Descripción de la Implementación	21
5. Pruebas y Resultados	23
5.1. Pruebas de la instalación de ROS en Ubuntu	23
5.2. Pruebas en instalación de ROS en tarjeta Raspberry Pi	26
5.3. Comunicación de Raspberry y Ubuntu utilizando el master roscore mediante la red	28
6. Conclusiones	32
7. Anexos	34
7.1. Anexo A: Instalación herramienta de simulación 3D de ROS	34

7.1.1.	Configuraciones necesarias antes de realizar las pruebas	35
7.1.2.	Pruebas de la herramienta de simulación 3D	37
7.2.	Anexo B: Instalación de sistema operativo en tarjeta Raspberry Pi	41
7.2.1.	Configuración para conectar con protocolo SSH	42
7.3.	Anexo C: Creación de paquetes de ROS	45
7.4.	Anexo D: Instalación de Ubuntu	48
7.5.	Anexo E: Instalación de ROS en ordenador	50

Índice de figuras

2.1.	Pruebas del sistema.	5
2.2.	Pruebas de la simulación UWSim	6
2.3.	Vehículo Mitsubishi Imiev utilizado para pruebas.	7
3.1.	Ejemplo de un sistema embebido. Raspberry Pi.	8
3.2.	Logo de ROS,	9
3.3.	Logo de Gazebo	10
3.4.	Logos de distribuciones aún en circulación.	12
3.5.	Distribuciones antiguas de ROS	13
3.6.	Diagrama de flujo ejemplo de un algoritmo.	14
3.7.	Logo de C++	15
3.8.	Raspberry Pi.	16
3.9.	Logo de Arduino.	17
3.10.	Placas Arduino.	18
3.11.	Logo de Open Source.	19
3.12.	Logo de Rufus.	20
5.1.	Ejecución de roscore.	24

5.2.	Simulación 2D de ROS "turtlesim"	25
5.3.	Ejecución de nodo para control por flechas de simulación 2D.	26
5.4.	Nodo talker, publicando mensajes cada 10 ms	27
5.5.	Nodo listener, se suscribe al nodo de talker para mostrar lo que escucha	28
5.6.	Configuraciones en Ubuntu para comunicar nodos a través de la red	29
5.7.	Configuraciones en Ubuntu para comunicar nodos a través de la red	30
5.8.	Ejecución de nodos en cada una de las maquinas.	31
7.1.	Simulación de casa 3D en Gazebo.	38
7.2.	Herramienta de visualización RViz en simulación de casa	39
7.3.	Programa de control para simulación 3D	40
7.4.	Instalación de Raspberry Pi OS utilizando Etcher	41
7.5.	Conexión de tarjeta Raspberry Pi, cable de red y fuente de energía	42
7.6.	Información de dispositivos en la red utilizando Fing	43
7.7.	Conexión de Raspberry utilizando el protocolo SSH en Putty	44
7.8.	Terminal SSH para Raspberry OS en Putty	45
7.9.	Ejemplo de creación de paquete	46
7.10.	Código para crear ejecutable	47
7.11.	Página de descarga de Ubuntu 20.04	48
7.12.	Creación de medio de instalación	49
7.13.	Repositorio de GitHub para la instalación de ROS Noetic Ninjemys	50

Capítulo 1

Introducción

El presente documento muestra una investigación e implementación de **ROS** (Sistema Operativo Robótico por sus siglas en inglés) para controlar sistemas embebidos, ROS es un conjunto de librerías de software y herramientas que ayudan a crear aplicaciones de robots. Haciendo posible desarrollar controladores hasta algoritmos de última generación utilizando estas potentes herramientas que se nos brindan a los desarrolladores, estas librerías siendo de código abierto brindan la posibilidad de modificar todo código proporcionado en las librerías, brindando una gran cantidad de ventajas en el desarrollo de nuevos proyectos. ROS puede ser utilizado en distintos sistemas operativos, esto debido a la llegada de ROS2, sin embargo en esta investigación se utiliza una versión anterior **ROS Noetic Ninjemys** (veáse en la sección 3.2.4), utilizando el sistema operativo de **Ubuntu** en su **versión 20.04**, el cual es de software libre y código abierto siendo una distribución de Linux basada en Debian este puede correr en computadores de escritorio y servidores el cual es orientado principalmente al usuario promedio, esta distribución es utilizada dado que la información y experimentación con la nueva distribución de ROS es

escasa.

1.1. Problemática

Hoy en día existen formas de controlar un sistema embebido, sin embargo estas maneras de controlarlos son muy complicadas, esto debido a que se utiliza una programación de bajo nivel teniendo que hacer que los programas interactúen directamente con los dispositivos electrónicos, haciendo de esto más complejo para los desarrolladores de software dado que deben adentrarse a los drivers del dispositivo en cuestión. Por otro lado las librerías de ROS trabajan a un nivel de programación alto, dado que dentro de las librerías de ROS ya están programados algunos drivers que nos solucionan este inconveniente de adentrarnos con el hardware, haciendo más sencillo el desarrollo de algoritmos para el control de los sistemas embebidos, sin embargo, existe muy poco aprovechamiento de estas librerías, dado que con ellas se pueden realizar un sin fin de proyectos de una manera más sencilla.

1.2. Justificación

Debido a la complejidad que existe en el desarrollo de software de gestión de sistemas embebidos, ROS resulta ser un gran espacio de trabajo debido a su simplicidad al manipular los componentes de los sistemas embebidos, evitando que el usuario tenga que realizar una programación de bajo nivel, teniendo en sus librerías una solución más sencilla e igual de eficiente, de esta manera ROS se vuelve una gran solución para los desarrolladores de software con poca experiencia en este ámbito al igual que para los desarrolladores más

habiles permitiendoles dejar esta programación más compleja de lado, es por ello que en esta investigación se demostrará el funcionamiento de los sistemas embebidos al utilizar esta solución.

1.3. Objetivos

1.3.1. Objetivo General

- Desarrollar e implementar algoritmos para el control de sistemas embebidos utilizando las librerías de ROS.

1.3.2. Objetivos específicos

- Desarrollar algoritmo para el control de dispositivos utilizando las librerías de ROS.
- Utilizar simulaciones proporcionadas por ROS para ejecutar los algoritmos desarrollados.
- Implementar los algoritmos desarrollados con las librerías de ROS en un sistema embebido para su gestión.
- Implementar librerías de ROS en un algoritmo para realizar la comunicación con el sistema embebido.

Capítulo 2

Estado del arte

En este capítulo se realiza una revisión profunda de los trabajos relacionados con el desarrollo del presente proyecto.

2.1. Sistema de adquisición de datos de una unidad de navegación inercial y ROS como herramienta de visualización.

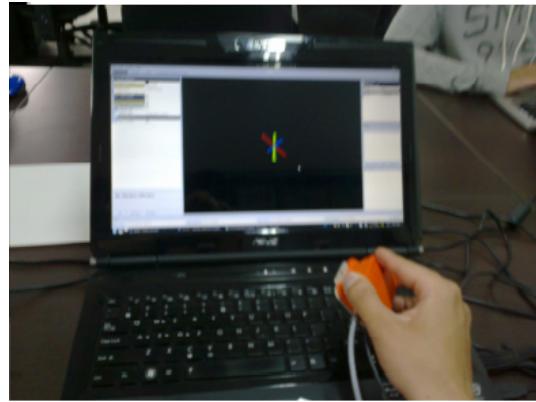


Figura 2.1: Pruebas del sistema.

La investigación de David Albeiro Taborda Alvarez y Andrés Guillermo Velásquez Gómez presenta el diseño, implementación y desarrollo de un prototipo de un vehículo aéreo no manejado (UAV por sus siglas en inglés), basado en un helicóptero aeromodelo, el cual tiene como función posicionarse en cualquier lugar que le sea indicado a partir de un punto inicial, en donde **ROS** se utiliza como una herramienta de visualización. [2]

2.2. Modelado, simulación y control de pequeños vehículos submarinos no tripulados



Figura 2.2: Pruebas de la simulación UWSim

Esta investigación se utiliza UWSim el cual es un simulador de vehículos submarinos pensado para el desarrollo e investigación de robots marinos. Este funciona sobre ROS es un sustituto para el simulador ROS por defecto Gazebo lo cual les permite visualizar e integrar la herramienta de visualización con las arquitecturas de control. [1]

2.3. Diseño de Algoritmos de Control de Nivel Bajo en Vehículo Inteligente



Figura 2.3: Vehículo Mitsubishi Imiev utilizado para pruebas.

En la investigación de Sergio Muñoz Ruiz se utilizan las librerías de ROS para programar la estructura del sistema, en donde utiliza nodos y mensajes para lograr regular el PID a los valores deseados y monitorizar el comportamiento del coche mediante gráficas integradas en las librerías de ROS.[3]

Capítulo 3

Marco Teórico

3.1. Sistemas embebidos

Un sistema embebido es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas frecuentemente en un sistema de computación en tiempo real.



Figura 3.1: Ejemplo de un sistema embebido. Raspberry Pi.

En un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base (la tarjeta de vídeo, audio, módem, etc.) aunque muchas veces los dispositivos no lucen como computadoras, por ejemplo relojes de taxi, registradores, controles de acceso entre otras múltiples aplicaciones. Por lo general los sistemas embebidos se pueden programar directamente en el lenguaje ensamblador del microcontrolador incorporado sobre el mismo o bien, utilizando algún compilador específico, suelen utilizarse lenguajes como: C, C++ y hasta en algunos casos BASIC.

3.2. ROS



Figura 3.2: Logo de ROS,

El sistema operativo para robots por sus siglas en inglés (ROS) es un framework flexible para escribir software de robots. Es una colección de herramientas y librerías que tienen como objetivo simplificar la tarea de crear un comportamiento de robot complejo y robusto en una amplia variedad de plataformas robóticas.

3.2.1. Herramientas de simulación Gazebo y RViz proporcionadas por ROS

RViz

Rviz es una herramienta de visualización en 3D para aplicaciones de ROS. Proporciona una vista del modelo de robot, capture la información de los sensores del robot y reproduce

los datos capturados. Puede mostrar datos de cámara, láseres y dispositivos 3D y 2D, como imágenes y nubes de puntos.

Gazebo



Figura 3.3: Logo de Gazebo

Gazebo es un simulador 3D multi-robot con dinámica. Ofrece la posibilidad de simular con precisión y eficiencia, diversidad de robots, objetos y sensores en ambientes complejos interiores y exteriores. Gazebo genera, tanto la realimentación realista de sensores, como las interacciones entre los objetos físicamente plausibles, incluida una simulación precisa de la física de cuerpo rígido.

3.2.2. Definición de algunas palabras utilizadas en ROS

- **Topic:** Son canales de información entre los nodos. Un nodo puede emitir o suscribirse a un tópico. Un tópico no es más que un mensaje que se envía. Podemos usar distintos tipos de clases de mensajes std_msgs, geometry_msgs, sensors_msgs, etc.
- **Package:** El software en ROS está organizado en paquetes. Un paquete puede tener un nodo, una librería, conjunto de datos, o cualquier cosa que pueda constituir un módulo.

- **Node:** Un nodo es un proceso que realiza algún tipo de computación en el sistema. Los nodos se combinan dentro de un grafo, compartiendo información entre ellos, para crear ejecuciones complejas. Un nodo puede controlar un sensor láser, los motores de un robot y la construcción de mapas.
- **Master Node:** Este es el nodo principal de ROS en donde todos los Nodos son ejecutados y administrados por este nodo principal, este nodo enlaza a todos los nodos permitiendo o negando acceso dependiendo del topic al que el nodo desee comunicar.

3.2.3. Licenciamiento de ROS

El núcleo de ROS está licenciado bajo la licencia BSD estándar de tres cláusulas. Se trata de una licencia abierta muy permisiva que permite su reutilización en productos comerciales y de código cerrado. Mientras que las partes principales de ROS se licencian bajo la licencia BSD, otras licencias se utilizan comúnmente en los paquetes de la comunidad, como la licencia Apache 2.0, la licencia GPL, la licencia MIT e incluso licencias propietarias. Cada paquete en el ecosistema ROS es necesario para especificar una licencia, de modo que sea fácil identificar rápidamente si un paquete satisface las necesidades de licencia.

3.2.4. Distribuciones de ROS

Una distribución ROS es un conjunto versionado de paquetes ROS. Estos son similares a las distribuciones de Linux (por ejemplo, Ubuntu). El propósito de las distribuciones ROS es permitir que los desarrolladores trabajen contra una base de código relativamente

estable hasta que estén listos para avanzar todo. Hasta el día de hoy ROS tiene en su repertorio 12 distribuciones de estas librerías en donde sólo 3 de ellas pueden ser utilizadas dado que las más recientes son actualizaciones de las anteriores mejorando varios aspectos.

Distribuciones de ROS aún en uso

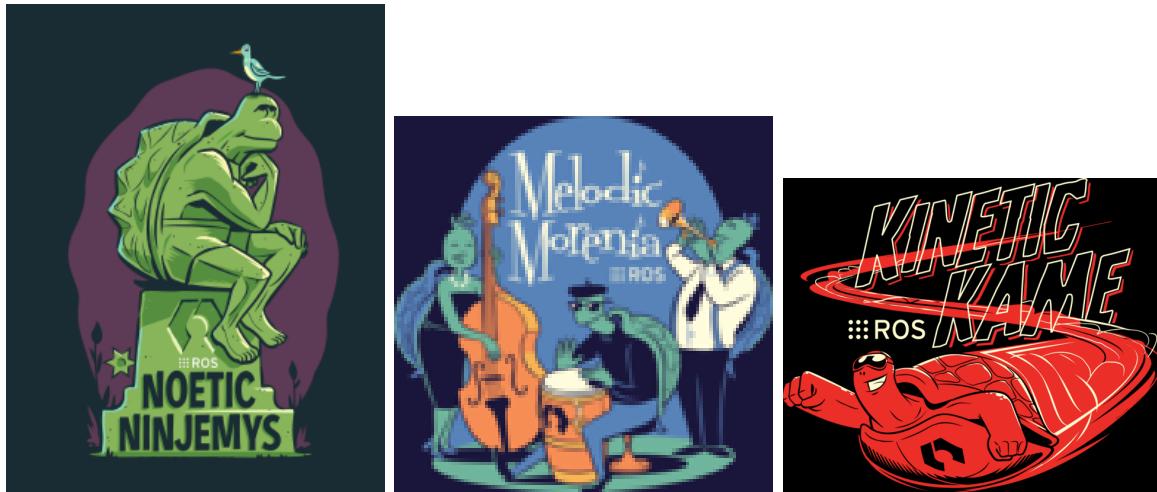


Figura 3.4: Logos de distribuciones aún en circulación.

Actualmente estas versiones (Kinetic Kame, Melodic Morenia y Noetic Ninjemys) son las que aún siguen siendo utilizadas, aunque se nos recomienda utilizar la distribución Noetic Ninjemys. Esta versión de ROS es la más reciente siendo lanzada en mayo de 2020 y con fecha de expiración de mayo de 2025, dado que esta es la ultima versión esta es la más recomendada de utilizar. Esta versión cuenta con las siguientes características:

Soporte requerido para:

- Ubuntu Focal Fossa (20.04)

Soporte recomendado para:

- Debian Buster

- Fedora 32

Arquitecturas admitidas:

- amd64
- arm32
- arm64

Distribuciones antiguas de ROS

Estas son las distribuciones de ROS que ya alcanzaron su fecha de expiración, cada una de ellas ya no es recomendable utilizar debido a los problemas que no se solucionaron y porque no reciben más actualizaciones.

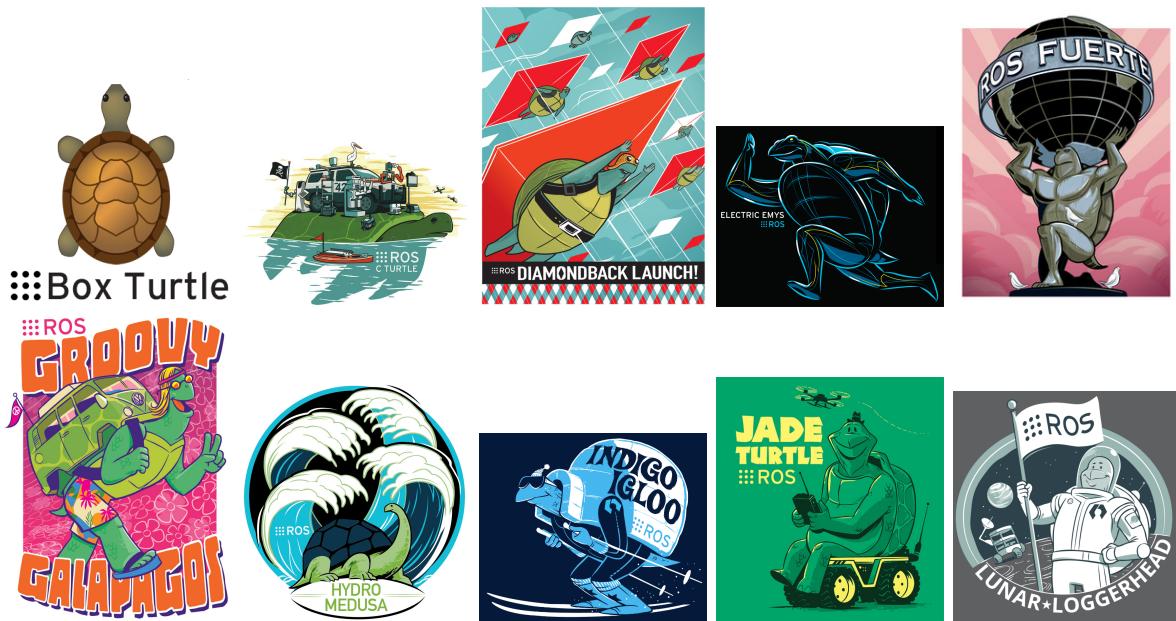


Figura 3.5: Distribuciones antiguas de ROS

3.3. Algoritmo

Un algoritmo es una sucesión de instrucciones secuenciales, en el que se llevan a cabo algunos procesos con la finalidad de dar respuestas a determinadas decisiones o necesidades. De la misma manera, los algoritmos son usados frecuentemente en lógica y matemáticas, además que son el fundamento de la elaboración de manuales de usuario, panfletos ilustrativos, entre otros. Unos de los más distinguidos en las matemáticas, es el atribuido al geómetra Euclides, para alcanzar el máximo común divisor de dos enteros que sean positivos y el conocido "método de Gauss" para determinar los sistemas de ecuaciones lineales. [8]

En relación con las ciencias de la computación, este cálculo puede ser conocido como la secuencia de pautas a seguir para la determinación de un problema a través del uso de un computador.

3.3.1. Partes de un algoritmo

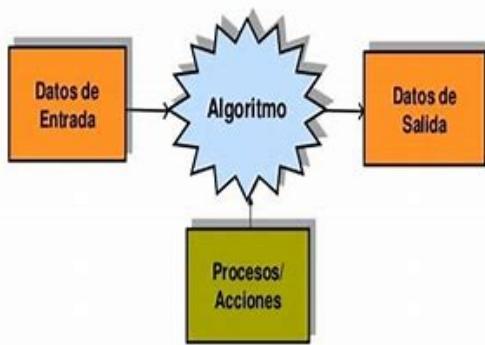


Figura 3.6: Diagrama de flujo ejemplo de un algoritmo.

- Entrada: también llamada cabecera o punto de partida, es la instrucción inicial que representa el génesis del algoritmo y la que motiva su lectura.
- Proceso: también llamado declaración, es la elaboración precisa que ofrece el algoritmo y se trata básicamente del tronco de sus claves para la formulación de instrucciones.
- Salida: en esta ultima fase se encuentran las instrucciones puntuales determinadas por el algoritmo, ejemplo, sus comandos o resoluciones.

3.4. C++



Figura 3.7: Logo de C++.

C++ es un lenguaje imperativo orientado a objetos derivado del C. En realidad un superconjunto de C, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. La definición "oficial" del lenguaje nos dice que C++ es un lenguaje de propósito

general basado en el C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones inline, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería (en realidad la librería Estándar C es un subconjunto de la librería C++). [8]

3.5. Raspberry

La Raspberry Pi es un sistema embebido de bajo costo y con un tamaño compacto, puede ser conectada a un monitor de computador o un TV, y usarse con un mouse y teclado estándar. Es un pequeño computador que corre un sistema operativo linux capaz de permitirle a las personas de todas las edades explorar la computación y aprender a programar lenguajes como Scratch, Python, C++, Java, entre otros. Es capaz de hacer la mayoría de las tareas típicas de un computador de escritorio, desde navegar en internet, reproducir videos en alta resolución, manipular documentos de ofimática, hasta reproducir juegos. [11]



Figura 3.8: Raspberry Pi.

Además la Raspberry Pi tiene la habilidad de interactuar con el mundo exterior, puede

ser usada en una amplia variedad de proyectos digitales, desde reproductores de música y video, detectores de padres, estaciones meteorológicas hasta cajas de aves con cámaras infrarrojas. Siendo raspberry una computadora de bajo costo, con dimensiones pequeñas y un buen funcionamiento, esta también tiene la ventaja de ser de software libre.

3.6. Arduino



Figura 3.9: Logo de Arduino.

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso. En la **figura 3.9** se pueden ver algunas de las placas desarrolladas, que pueden utilizarse, cada una de ellas tiene diferentes especificaciones, esto porque se utilizan en distintos ámbitos.

[10]



Figura 3.10: Placas Arduino.

3.7. Sistema operativo

Un sistema operativo es el software principal o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software, ejecutándose en modo privilegiado respecto de los restantes dado que este debe tener todo el control del computador. [15] Algunos ejemplos de sistemas operativos son los siguientes:

- Microsoft Windows
- MS-DOS
- UNIX
- MacOS
- Ubuntu
- Android

3.8. Software de código abierto

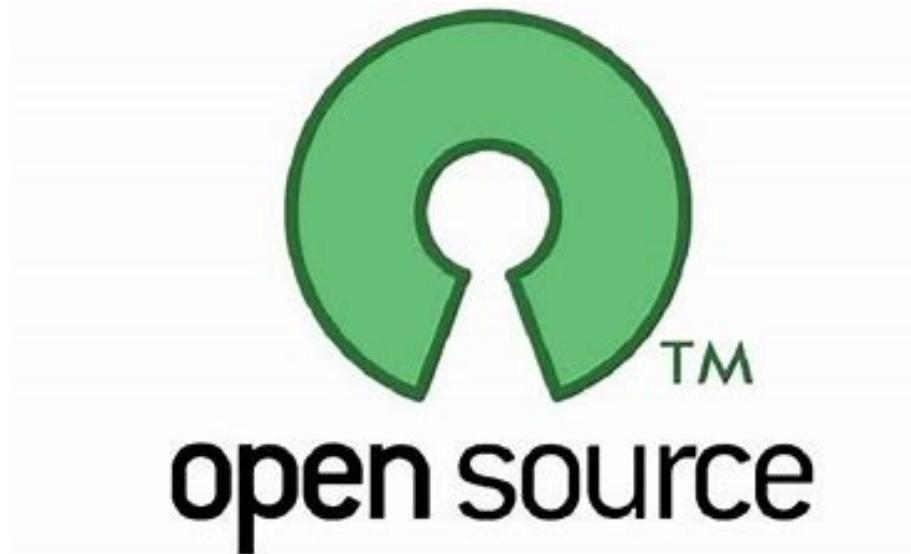


Figura 3.11: Logo de Open Source.

El Software de código abierto o libre se basa en los programas cuyo código no es secreto ni está sujeto a ningún tipo de licencia. Esto implica que se puede usar, cambiar y distribuir

del modo que uno desee, sin ningún tipo de trabas, dado que no está sujeto a ningún tipo de licencia. Los programadores de una aplicación entregan el programa a los usuarios, pero además enseñan, por así decirlo, su código, para que otros programadores puedan ver como funciona, como han trabajado e incluso para que puedan cambiarlo, mejorarlo o distribuirlo entre más gente. [9]

3.9. Rufus



Figura 3.12: Logo de Rufus.

Rufus es una herramienta para Windows que nos permitirá crear dispositivos de arranque de diversa índole a partir de unidades de disco externas como pendrives USB o tarjetas SD. Su versatilidad permite tanto formatear una unidad como instalar imágenes de disco de Linux, Windows e incluso FreeDOS, integrado en la herramienta.

Capítulo 4

Descripción de la Implementación

Para dar comienzo las librerías de ROS han de ser instaladas sobre el sistema operativo de Ubuntu 20.04 como se muestra en el **Anexo 7.5** el cual deberá esta instalado en otra partición de disco para tener un buen control del sistema, esta instalación de Ubuntu se muestra en el **Anexo D 7.4**.

Teniendo en cuenta todas las herramientas a utilizar, se desarrollaran algoritmos haciendo uso de las librerías proporcionadas por ROS, así como también este código será escrito en lenguaje C++ dado que es el lenguaje que más se domina entre los recomendados por la distribución de ROS Noetic Ninjemys.

Para hacer uso de la tarjeta Raspberry Pi, como se muestra en el **Anexo B 7.2** se realiza la instalación del sistema operativo de Ubuntu 16.04, imagen ISO proporcionada por ubiquityrobotics [17] dado que en la instalación de Raspbian surgían errores al instalar las librerías de ROS

Se emplearán simulaciones utilizando las herramientas proporcionadas por las librerías para realizar pruebas del código desarrollado, estas simulaciones servirán para probar los

algoritmos en estos entornos simulados, herramientas proporcionadas por ROS.

Tras haber realizado cada una de las simulaciones y probado cada uno de los algoritmos desarrollados, se comenzará a utilizar una Raspberry Pi v3 en la cuál se ejecutarán los algoritmos, gestionando el sistema embebido.

Capítulo 5

Pruebas y Resultados

En este capítulo se muestran todas las pruebas realizadas, al igual que los resultados obtenidos en el uso de las librerías de ROS.

5.1. Pruebas de la instalación de ROS en Ubuntu

Una vez que ROS este instalado en nuestro sistema podemos realizar algunas pruebas para saber si la instalación terminó con éxito. La primer prueba que se puede realizar es abrir una terminal de Ubuntu y en esta escribir el comando ***roscore*** y darle enter para que se ejecute el comando, una vez que este comando se debe mostrar que se inicio el nodo maestro de ROS como se aprecia en la **figura 5.1** en caso de que surja algún error en la página de instalación se presentan varias soluciones de errores.

```

/home/alejandro/catkin_ws/src:/opt/ros/noetic/share
alejandro@alejandro-Nitro-AN515-52:~$ roscore
... logging to /home/alejandro/.ros/log/3186333a-13fd-11eb-a7ed-87d047b9702d/roslaunch-alejandro-Nitro-AN515-52-5320.log
Checking log directory for dtsk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://alejandro-Nitro-AN515-52:37105/
ros_comm version 1.15.8

SUMMARY
========
PARAMETERS
  * /rosdistro: noetic
  * /rosversion: 1.15.8

NODES
auto-starting new master
process[master]: started with pid [5332]
ROS_MASTER_URI=http://alejandro-Nitro-AN515-52:11311/

setting /run_id to 3186333a-13fd-11eb-a7ed-87d047b9702d
process[rosout-1]: started with pid [5342]
started core service [/rosout]

```

Figura 5.1: Ejecución de roscore.

Una vez probado esto se considera que la instalación de ROS ha sido correcta, pero para estar seguros se pueden ejecutar algunos nodos que vienen de ejemplo dentro de las librerías.

Utilizando el comando ***rosrun turtlesim turtlesim_node*** en una nueva terminal, al ejecutarlo este desplegará otra ventana en la que dentro estará una tortuga como se muestra en la **figura 5.2**, esta es una simulación 2D que proporciona ROS, esta puede ser movida en rotación de la imagen al igual que su movimiento dentro de la ventana desarrollando algún código, en este caso las librerías nos proporcionan un nodo ejemplo en donde se pueden utilizar las flechas del teclado para controlar el movimiento de la tortuga, para utilizar este nodo se debe de ejecutar el comando ***rosrun turtlesim turtlesim_teleop_key*** en una nueva terminal, ejecutando el comando se nos mostrará la información como aparece

en la **figura 5.3** y seremos capaces de mover la tortuga.

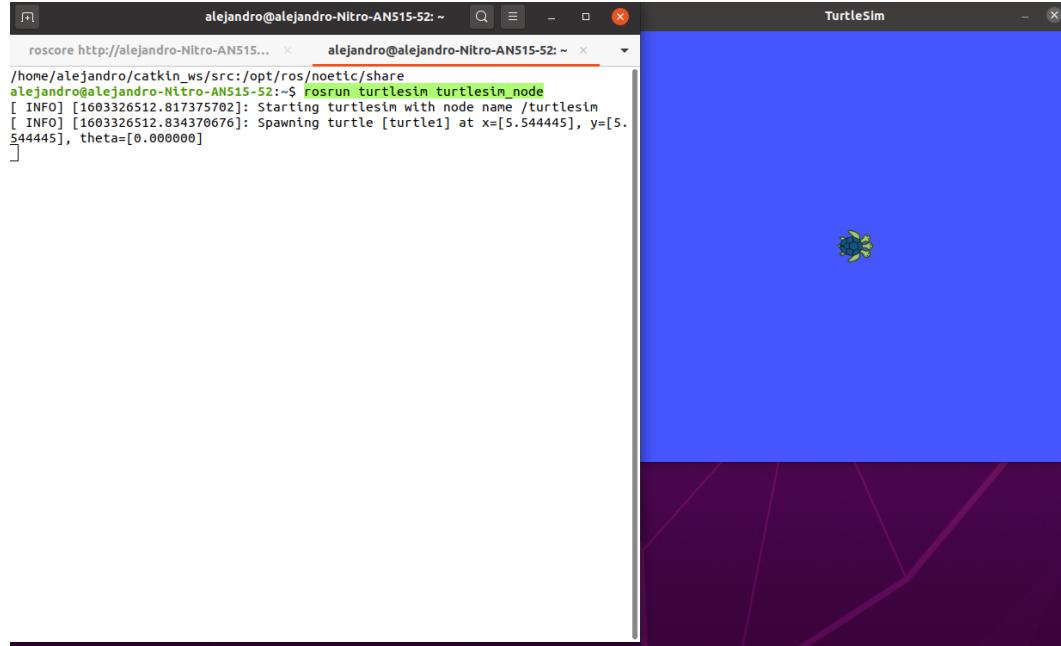


Figura 5.2: Simulación 2D de ROS "turtlesim".

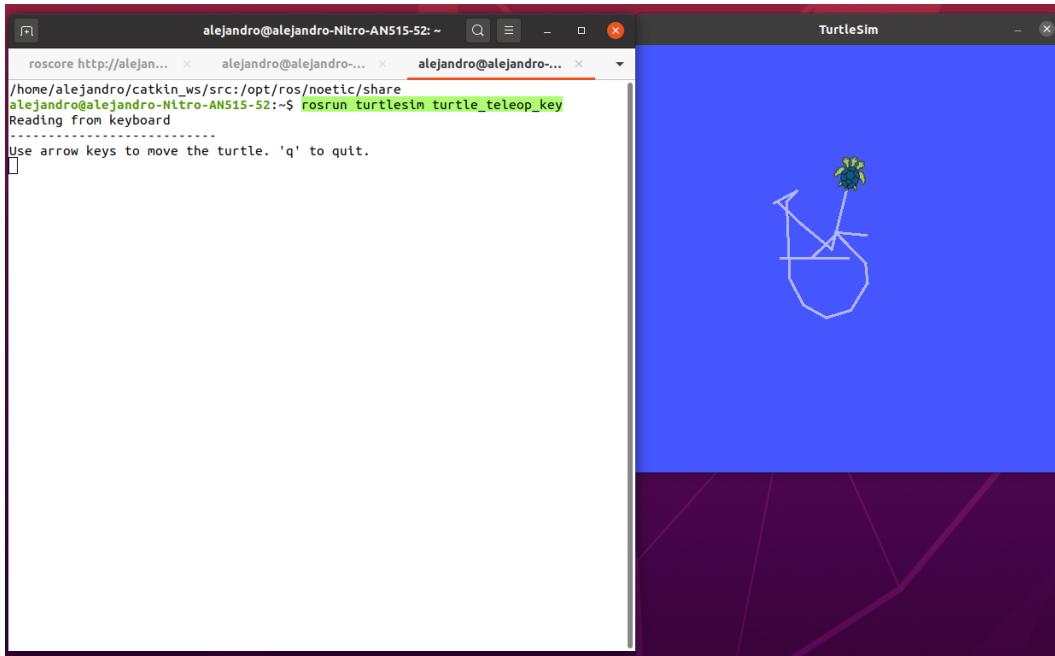


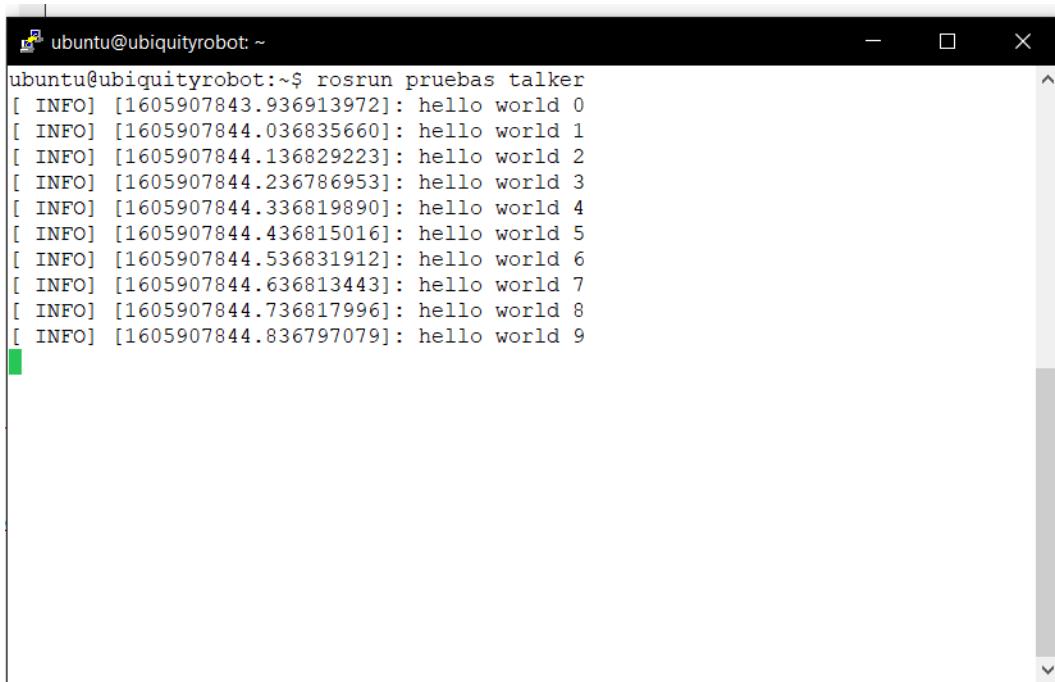
Figura 5.3: Ejecución de nodo para control por flechas de simulación 2D.

Después de realizar estas pruebas se puede estar seguro que la instalación de ROS fue exitosa, en caso de aún querer estar más seguros se pueden probar los nodos ejemplos que nos proporcionan las librerías.

5.2. Pruebas en instalación de ROS en tarjeta Raspberry Pi

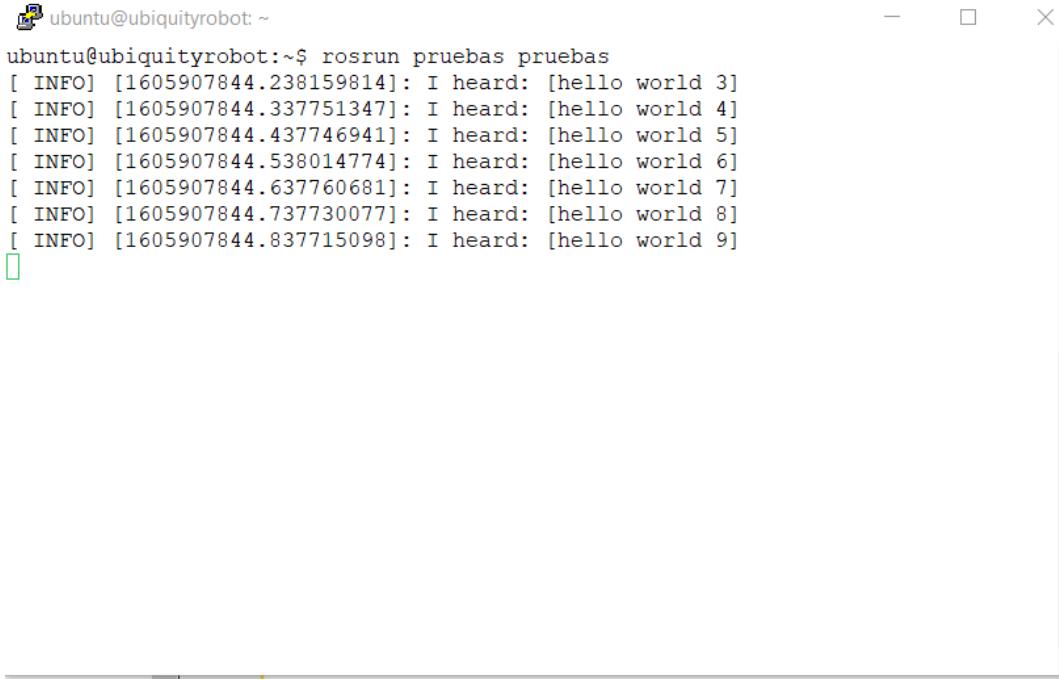
En esta sección se comprobará el correcto funcionamiento de ROS instalado en la Raspberry Pi, la instalación del sistema operativo de Raspberry Pi junto con Ros se puede ver en el **Anexo B 7.2**. Una vez que tenemos nuestra tarjeta funcionando junto con ROS se realizan las pruebas pertinentes para observar el funcionamiento de este en el sistema

operativo de Raspberry. Como se puede observar en la **figura 5.4** al iniciar el nodo talker este comienza a publicar mensajes a un topic, permitiendo a todos los nodos que requieran de esta información acceder a el mensaje suscribiéndose al topic publicado por el nodo, para esto como se puede observar en la **figura 5.5** este se suscribe al topic publicado y nos muestra en pantalla lo que esta escuchando cada 10 milisegundos, desmostrando que las librerías de ROS funcionan correctamente en el entorno de Raspberry.

A screenshot of a terminal window titled "ubuntu@ubiquityrobot:~\$". The window contains a command-line interface where the user has run "rosrun pruebas talker". The output shows a series of [INFO] messages, each containing the text "hello world" followed by a number from 0 to 9, indicating that the node is publishing messages at regular intervals.

```
ubuntu@ubiquityrobot:~$ rosrun pruebas talker
[ INFO] [1605907843.936913972]: hello world 0
[ INFO] [1605907844.036835660]: hello world 1
[ INFO] [1605907844.136829223]: hello world 2
[ INFO] [1605907844.236786953]: hello world 3
[ INFO] [1605907844.336819890]: hello world 4
[ INFO] [1605907844.436815016]: hello world 5
[ INFO] [1605907844.536831912]: hello world 6
[ INFO] [1605907844.636813443]: hello world 7
[ INFO] [1605907844.736817996]: hello world 8
[ INFO] [1605907844.836797079]: hello world 9
```

Figura 5.4: Nodo talker, publicando mensajes cada 10 ms

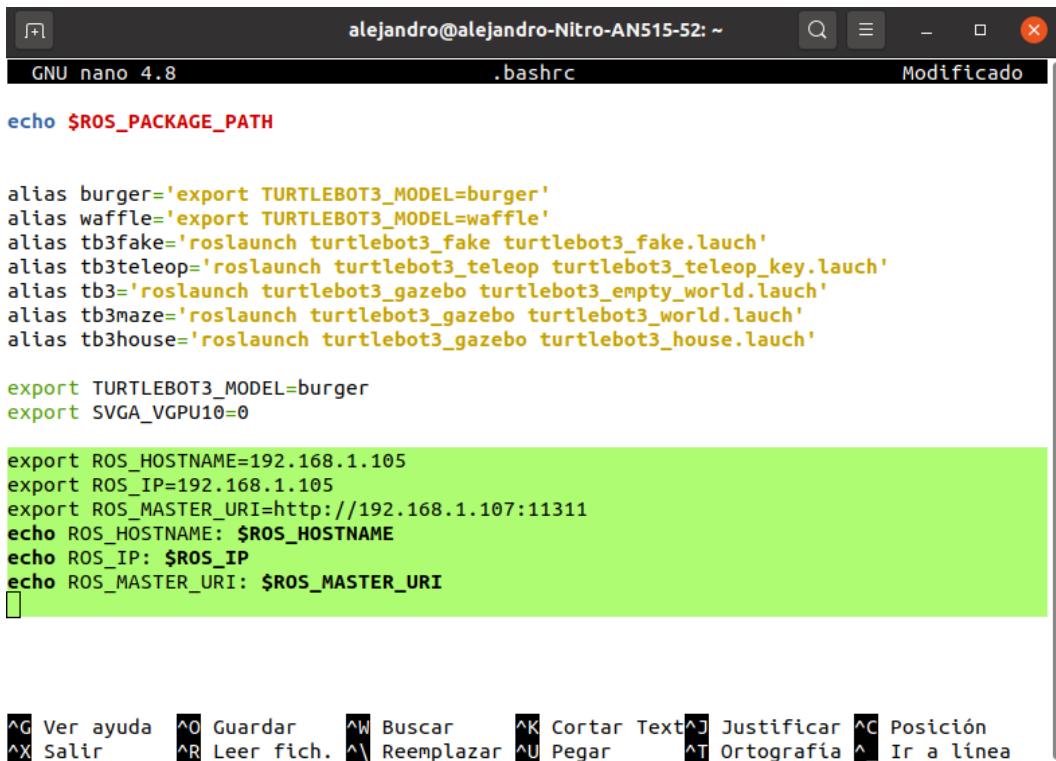


```
ubuntu@ubiquityrobot:~$ rosrun pruebas pruebas
[ INFO] [1605907844.238159814]: I heard: [hello world 3]
[ INFO] [1605907844.337751347]: I heard: [hello world 4]
[ INFO] [1605907844.437746941]: I heard: [hello world 5]
[ INFO] [1605907844.538014774]: I heard: [hello world 6]
[ INFO] [1605907844.637760681]: I heard: [hello world 7]
[ INFO] [1605907844.737730077]: I heard: [hello world 8]
[ INFO] [1605907844.837715098]: I heard: [hello world 9]
```

Figura 5.5: Nodo listener, se suscribe al nodo de talker para mostrar lo que escucha

5.3. Comunicación de Raspberry y Ubuntu utilizando el master roscore mediante la red

Para tener una comunicación entre nuestro roscore Raspberry y nuestro roscore instalado en el sistema operativo de Ubuntu de nuestro ordenador, se deben realizar algunas configuraciones dentro del archivo `.bashrc`, en Ubuntu las configuraciones necesarias son las que se muestran en la **figura 5.6** donde se especifica la IP de la maquina, el nombre de host y la IP del master roscore que en este caso es la IP de nuestra Raspberry.



```
alejandro@alejandro-Nitro-AN515-52: ~
GNU nano 4.8          .bashrc          Modificado
echo $ROS_PACKAGE_PATH

alias burger='export TURTLEBOT3_MODEL=burger'
alias waffle='export TURTLEBOT3_MODEL=waffle'
alias tb3fake='roslaunch turtlebot3_fake turtlebot3_fake.launch'
alias tb3teleop='roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch'
alias tb3='roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch'
alias tb3maze='roslaunch turtlebot3_gazebo turtlebot3_world.launch'
alias tb3house='roslaunch turtlebot3_gazebo turtlebot3_house.launch'

export TURTLEBOT3_MODEL=burger
export SVGA_VGPU10=0

export ROS_HOSTNAME=192.168.1.105
export ROS_IP=192.168.1.105
export ROS_MASTER_URI=http://192.168.1.107:11311
echo ROS_HOSTNAME: $ROS_HOSTNAME
echo ROS_IP: $ROS_IP
echo ROS_MASTER_URI: $ROS_MASTER_URI
```

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar ^T Ortografía ^ I Ir a linea

Figura 5.6: Configuraciones en Ubuntu para comunicar nodos a través de la red

En cuanto a la configuración en la Raspberry, esta es un tanto parecida tal y como se muestra en la **figura 5.7** en donde se puede observar que se exportará como localhost el master roscore, dado que el nodo maestro estará situado en nuestra Raspberry, configurando también la IP y el nombre de host en esta misma.

```

ubuntu@ubiquityrobot: ~
GNU nano 2.5.3          File: .bashrc          Modified

if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
source /opt/ros/kinetic/setup.bash
source /home/ubuntu/catkin_ws/devel/setup.bash
source /etc/ubiquity/env.sh

export ROS_IP=192.168.1.107
export ROS_HOSTNAME=192.168.1.107
export ROS_MASTER_URI=http://localhost:11311
echo "ROS HOSTNAME: \"$ROS_HOSTNAME"
echo "ROS IP: \"$ROS_IP"
echo "ROS MASTER URI: \"$ROS_MASTER_URI"

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Spell ^_ Go To Line

```

Figura 5.7: Configuraciones en Ubuntu para comunicar nodos a través de la red

Teniendo estas configuraciones en ambas maquinas se tiene que reiniciar cada una de ellas para que los cambios hagan efecto, una vez iniciadas las maquinas estas ya estarán comunicandose a través de la red dado que en la Raspberry el nodo maestro se inicia una vez encendida, para verificar que la comunicación este funcionando adecuadamente podemos hacer uso de los ejemplos “listener” y “talker” anteriormente utilizados para comprobar que Ubuntu puede acceder a los nodos publicados por la Raspberry y viceversa. Para esto crearemos un paquete con los códigos ejemplos como se muestra en el **Anexo C 7.3.**

Después de crear un nodo en la Raspberry y en nuestra maquina Ubuntu podremos ejecutarlos para observar el funcionamiento de ambos como se puede ver en la **figura 5.8.**

```

[ alejandro@alejandro-Nitro-AN515-52: ~/catkin_ws ] % rosrun pruebas listener_node
[ 7%] Built target turtlebot3_msgs_generate_messages_cpp
[ 12%] Built target turtlebot3_msgs_generate_messages_nodejs
[ 15%] Built target flat_world_lm_node
[ 21%] Built target turtlebot3_msgs_generate_messages_py
[ 32%] Built target turtlebot3_msgs_generate_messages_lisp
[ 32%] Built target turtlebot3_msgs_generate_messages_eus
[ 43%] Built target turtlebot3_example_generate_messages_cpp
[ 54%] Built target turtlebot3_example_generate_messages_nodejs
[ 67%] Built target turtlebot3_example_generate_messages_eus
[ 79%] Built target turtlebot3_example_generate_messages_py
[ 90%] Built target turtlebot3_example_generate_messages_lisp
[ 90%] Built target turtlebot3_msgs_generate_messages
[ 93%] Built target turtlebot3_diagnostics
[ 93%] Built target turtlebot3_example_generate_messages
[ 96%] Built target turtlebot3_driver
[ 100%] Built target turtlebot3_fake_node
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws$ rosrun pruebas listener_node
[ INFO] [1605898012.651550591]: I heard: [hello world 4]
[ INFO] [1605898012.7512239171]: I heard: [hello world 5]
[ INFO] [1605898012.7518044474]: I heard: [hello world 6]
[ INFO] [1605898012.851704399]: I heard: [hello world 7]
[ INFO] [1605898012.9515238113]: I heard: [hello world 8]
[ INFO] [1605898013.051482536]: I heard: [hello world 9]
[ INFO] [1605898013.151343364]: I heard: [hello world 10]
[ INFO] [1605898013.251360441]: I heard: [hello world 11]
[ INFO] [1605898013.351385384]: I heard: [hello world 12]
[ INFO] [1605898013.451338987]: I heard: [hello world 13]
[ INFO] [1605898013.551286155]: I heard: [hello world 14]

ubuntu@ubiquityrobot:~/catkin_ws %
[ 50%] Built target docking_generate_messages_lisp
[ 62%] Built target docking_generate_messages_py
[ 68%] Built target docking_generate_messages_cpp
[ 81%] Built target docking_generate_messages_eus
[ 87%] Built target docking_generate_messages_nodejs
[ 87%] Built target docking_generate_messages
[ 87%] Built target docking_generate_messages
[100%] Built target pruebas
ubuntu@ubiquityrobot:~/catkin_ws$ rosrun pruebas talker
[ INFO] [1605898012.186160021]: hello world 0
[ INFO] [1605898012.286063994]: hello world 1
[ INFO] [1605898012.386000311]: hello world 2
[ INFO] [1605898012.486047356]: hello world 3
[ INFO] [1605898012.586036224]: hello world 4
[ INFO] [1605898012.686044311]: hello world 5
[ INFO] [1605898012.786107919]: hello world 6
[ INFO] [1605898012.886084548]: hello world 7
[ INFO] [1605898012.986054198]: hello world 8
[ INFO] [1605898013.086109629]: hello world 9
[ INFO] [1605898013.186099268]: hello world 10
[ INFO] [1605898013.286069449]: hello world 11
[ INFO] [1605898013.386066963]: hello world 12
[ INFO] [1605898013.486074842]: hello world 13
[ INFO] [1605898013.586070012]: hello world 14

```

Figura 5.8: Ejecución de nodos en cada una de las maquinas.

Capítulo 6

Conclusiones

La presente tesis tuvo como objetivo desarrollar e implementar algoritmos para el control de sistemas embebidos utilizando las librerías de ROS, para hacer esto posible se realizaron instalaciones de los sistemas operativos de Ubuntu 20.04, Raspbian y Ubuntu 16.04 sobre Raspberry Pi.

Surgieron problemas al intentar instalar las librerías de ROS en el sistema operativo de Raspbian, es por ello que se optó por utilizar un sistema operativo de Ubuntu 16.04 con Ros preinstalado, esto debido a que no existe aún una solución al problema que se genera al intentar instalar ROS en Raspbian.

A pesar de los percances de la investigación se logró seguir con el desarrollo de esta, siendo así que se consiguió realizar la comunicación entre el entorno de ROS instalado en Ubuntu 20.04 y Raspberry a través de la red, brindando la posibilidad de manipular completamente el sistema embebido como Raspberry Pi desde un ordenador. Para demostrar de otra manera más visual se utilizan los simuladores proporcionados por ROS para controlarlos utilizando esta comunicación de ROS a través de la red.

Dejando como trabajo a futuro una gran posibilidad de proyectos, a fin de poder ensamblar robots de diferentes modelos y poder gestionar una gran cantidad de ellos dado que se pueden tener varios robots en una misma red y pueden ser esamblados utilizando estos sistemas embebidos.

Capítulo 7

Anexos

7.1. Anexo A: Instalación herramienta de simulación 3D de ROS

La instalación de esta herramienta resulta ser algo sencilla, aunque se debe tener cuidado al escribir los comandos de instalación, especificando la distribución de ROS que tenemos instalada, en caso de escribir otra versión ocasionará un sin fin de problemas o simplemente no lo instalará. Para realizar la instalación primero se deben descargar los paquetes más recientes, después de haber actualizado la lista de paquetes se instala. Para poder realizar esto se necesita de los comandos:

- apt-update: actualiza la lista de paquetes disponibles y sus versiones, pero no instala o actualiza ningún paquete.
- apt-upgrade: Instalará las nuevas versiones descargadas respetando la configuración del software cuando sea posible.

El paso anterior no debe ser omitido, dado que al omitir este paso puede que la instalación de la herramienta no se realice satisfactoriamente. Como paso siguiente se clonarán los repositorios de GitHub que contienen las librerías de esta herramienta en el espacio de trabajo y se compilan utilizando los siguientes comandos:

- `cd /catkin_ws/src`
- `git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git -b noetic-devel`
- `git clone https://github.com/ROBOTIS-GIT/turtlebot3.git -b noetic-devel`
- `git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git -b noetic-devel`
- `cd /catkin_ws/`
- `catkin_make`

Para realizar una instalación correcta de esta herramienta se debe tener conocimiento de la distribución de ROS que estamos utilizando, en este caso **Noetic Ninjemys**, para tener compatibilidad de la herramienta y la distribución.

7.1.1. Configuraciones necesarias antes de realizar las pruebas

Una vez que hemos compilado todos los repositorios en nuestro entorno de trabajo debemos escribir las siguientes líneas, en nuestro archivo `.bashrc`, este archivo es el que se ejecuta justo al momento de abrir una nueva terminal, y para que la herramienta de simulación este funcionando se deben de ejecutar algunas líneas antes de ser ejecutada cada simulación, para realizar esto se utiliza primero el comando `gedit .bashrc` en una

terminal, esta nos abrirá un archivo de texto en el que se deben colocar las siguientes líneas:

- alias burger='export TURTLEBOT3_MODEL=burger'
- alias waffle='export TURTLEBOT3_MODEL=waffle'
- alias tb3fake='roslaunch turtlebot3_fake turtlebot3_fake.launch'
- alias tb3teleop='roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch'
- alias tb3='roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch'
- alias tb3maze='roslaunch turtlebot3_gazebo turtlebot3_world.launch'
- alias tb3house='roslaunch turtlebot3_gazebo turtlebot3_house.launch'

Las líneas anteriores sirven para identificar facilmente los ejecutables dentro del bashrc, esto porque se les asigna un alias corto, para que en el momento de ser requeridos no se escriba todo el comando completo sino su alias asignado.

- source /opt/ros/noetic/setup.bash
- source /home/alejandro/catkin_ws/devel/setup.bash
- export TURTLEBOT_MODEL=waffle
- export SVGA_VGPU10=0

En el caso de estas líneas siguientes son mucho más importantes, debido a que con las primeras dos líneas se identifican los espacios de trabajo que se estan utilizando, es decir,

donde se encuentran todos nuestros códigos y libreías. Las siguientes dos líneas son para exportar dos variables cada que una nueva terminal es abierta, siendo el modelo del bot a utilizar el cual es el **waffle** y una variable que nos permite ejecutar Gazebo en una maquina virtual evitando que esta produzca un error.

7.1.2. Pruebas de la herramienta de simulación 3D

Después de haber realizado la configuración de la herramienta se pueden realizar las pruebas haciendo uso de las demostraciones que nos proporciona ROS en sus librerías.

Para utilizar estas demos, se abre una terminal en la que deberemos de ejecutar el comando dependiendo de cuál demo se utilizará, al utilizar el comando **roslaunch turtlebot3_gazebo turtlebot3_house.launch** ejecutará en la maquina virtual una simulación en la que se nos presenta una casa en 3D con un robot dentro de la misma, en este caso el robot **waffle** especificado en el archivo bashrc así como se puede observar en la **figura 7.1**, dentro de esta ventana desplegada nosotros podemos modificar aspectos de la simulación, pudiendo agregar objetos, paredes, físicas, entre otros aspectos.

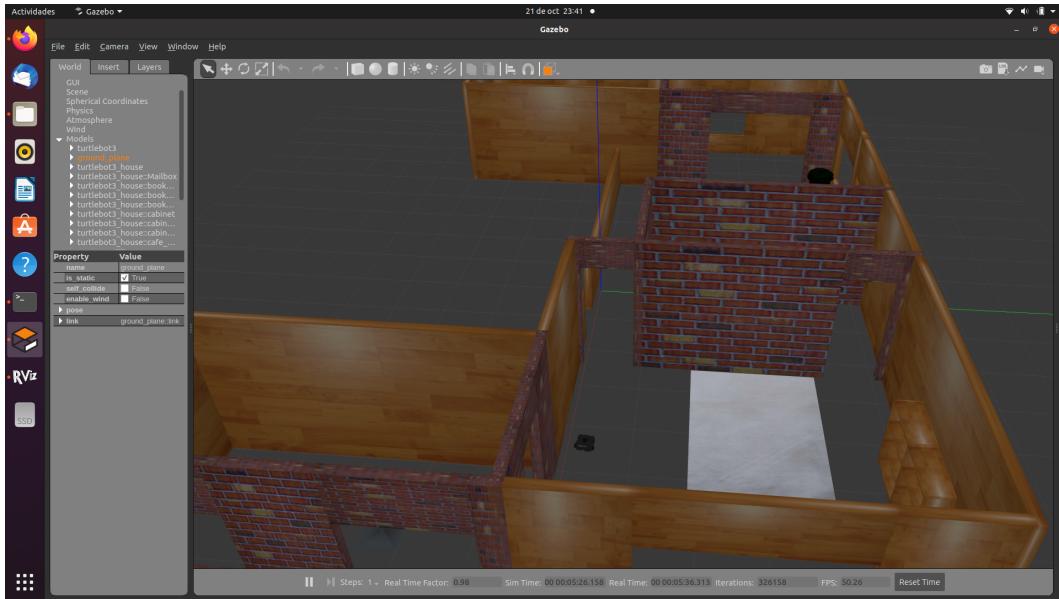


Figura 7.1: Simulación de casa 3D en Gazebo.

Para observar más información del robot dentro del entorno debemos ejecutar en otra terminal el comando `roslaunch turtlebot3_rviz turtlebot3_rviz.launch`, este comando ejecuta la herramienta **RViz** como se puede apreciar en la **figura 7.2**, la cual nos proporciona toda la información de los sensores del robot, cámaras, láseres y demás sensores en el robot, brindando de una gran cantidad de información en las simulaciones.

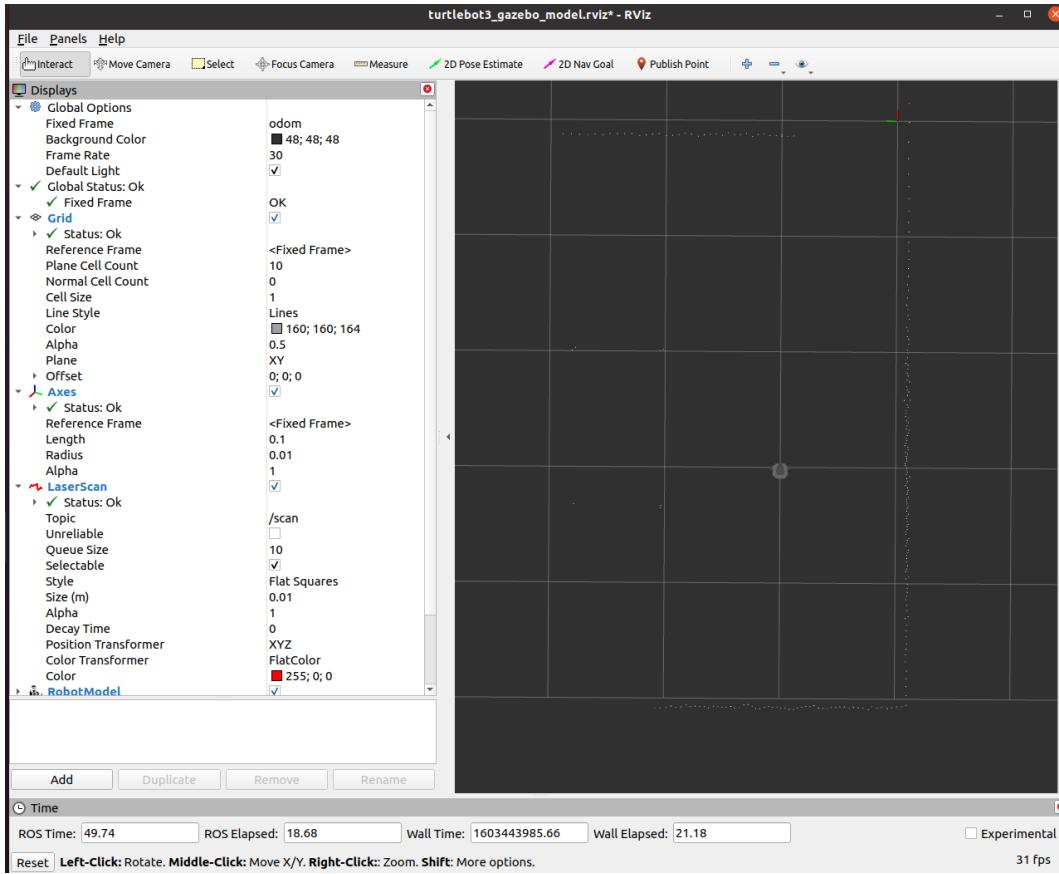


Figura 7.2: Herramienta de visualización RViz en simulación de casa

Después de haber ejecutado estas dos herramientas, las librerías nos proporciona un programa con el que podemos controlar el robot que esta dentro de la simulación logrando ver como es que el robot se mueve dentro de esta. Para realizar esto, en una nueva terminal se ejecuta el comando `roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`, una vez ejecutado este comando dentro de la terminal se nos mostrarán instrucciones de como se realiza el uso de este programa como se puede ver en la **figura 7.3**.

The screenshot shows a terminal window with three tabs open. The active tab is titled '/home/alejandro/catkin_ws/src/turtlebot3/turtlebot3_teleop/l...'. The window displays the 'Control Your TurtleBot3!' interface. It includes instructions for movement using keys w, s, d, a, and x. It also provides information about linear and angular velocity ranges for different models (Burger, Waffle, and Waffle Pi) and indicates that the space key stops the robot. A message at the bottom says 'CTRL-C to quit'. Below this, a series of 'currently:' log entries show the current linear and angular velocities.

```
Control Your TurtleBot3!
-----
Moving around:
      w
    a   s   d
      x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi :
      ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi
      : ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently: linear vel 0.21000000000000005 angular vel 0.0
currently: linear vel 0.22000000000000006 angular vel 0.0
currently: linear vel 0.23000000000000007 angular vel 0.0
currently: linear vel 0.24000000000000007 angular vel 0.0
currently: linear vel 0.25000000000000006 angular vel 0.0
currently: linear vel 0.26 angular vel 0.0
currently: linear vel 0.26 angular vel 0.0
```

Figura 7.3: Programa de control para simulación 3D

En este programa se nos indica primero cuales son las teclas que se utilizan para el control (a,w,s,d,x), en donde se nos indica que las teclas (w,x) son para incrementar y decrementar la velocidad lineal que es la que hace que el robot avance o retroceda, las teclas (a,d) para incrementar y decrementar la velocidad angular del robot la cual es la que hace que el robot gire en su eje para un lado o para otro y la tecla (s) la cual hace función de detener el robot sin importar la velocidad que este tenga. En este programa se nos indican las velocidades cada que estas tienen un cambio.

7.2. Anexo B: Instalación de sistema operativo en tarjeta Raspberry Pi

En esta sección se realizará la instalación del sistema operativo en la tarjeta Raspberry Pi, como primer paso tenemos que tener instalado **Raspberry Pi OS** en una memoria SD. Para realizar esto se utiliza el programa **Etcher** para instalar el sistema operativo en la memoria SD como se muestra en la **figura 7.4**

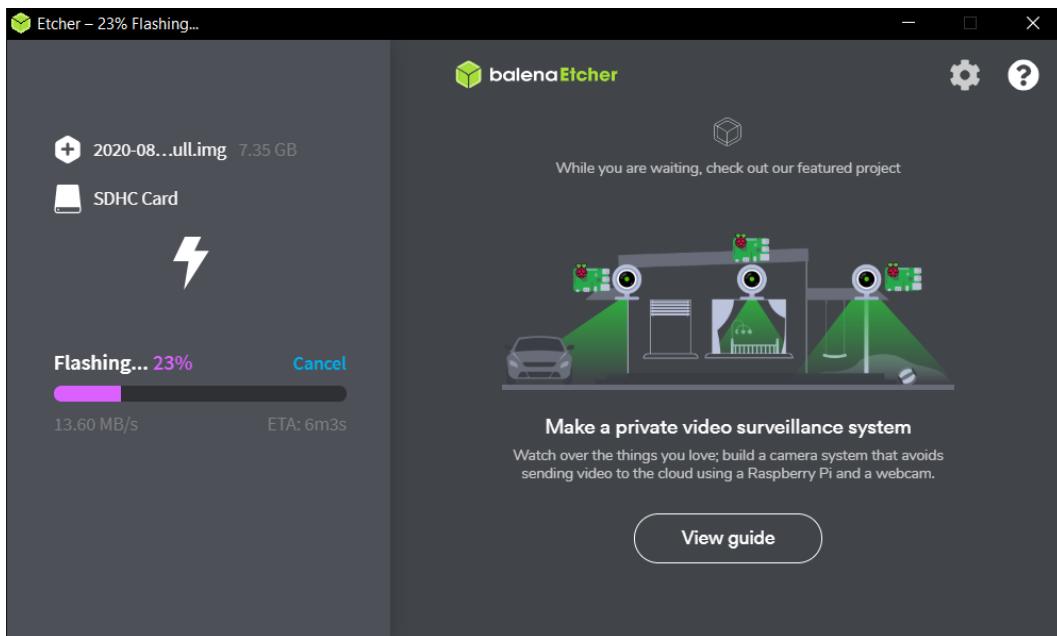


Figura 7.4: Instalación de Raspberry Pi OS utilizando Etcher

Cuando este proceso finalice, el sistema operativo para Raspberry puede ser utilizado, sin embargo, como no se cuenta con un monitor se creará un archivo con el nombre SSH para permitirnos acceder al sistema operativo utilizando este protocolo de comunicación.

7.2.1. Configuración para conectar con protocolo SSH

Para utilizar el protocolo SSH, la Raspberry Pi es conectada utilizando un cable de red como se muestra en la **figura 7.5** en donde la tarjeta Raspberry Pi esta conectada con su fuente de energía y el cable de red al router.



Figura 7.5: Conexión de tarjeta Raspberry Pi, cable de red y fuente de energía

Al conectar la tarjeta tenemos la posibilidad de ver nuestra tarjeta Raspberry Pi en la red, para hacer esto podemos utilizar algún programa para computadora o alguna aplicación para celular que nos permita observar que dispositivos estan conectados a nuestra red. Para observar esto, se utilizó la aplicación **Fing**, esta nos brinda la información de los dispositivos que estan conectados a nuestra red como se muestra en la **figura 7.6**, entre ellos la tarjeta Raspberry Pi, de la cual necesitamos la **IP** de esta para tener acceso mediante el protocolo **SSH**.

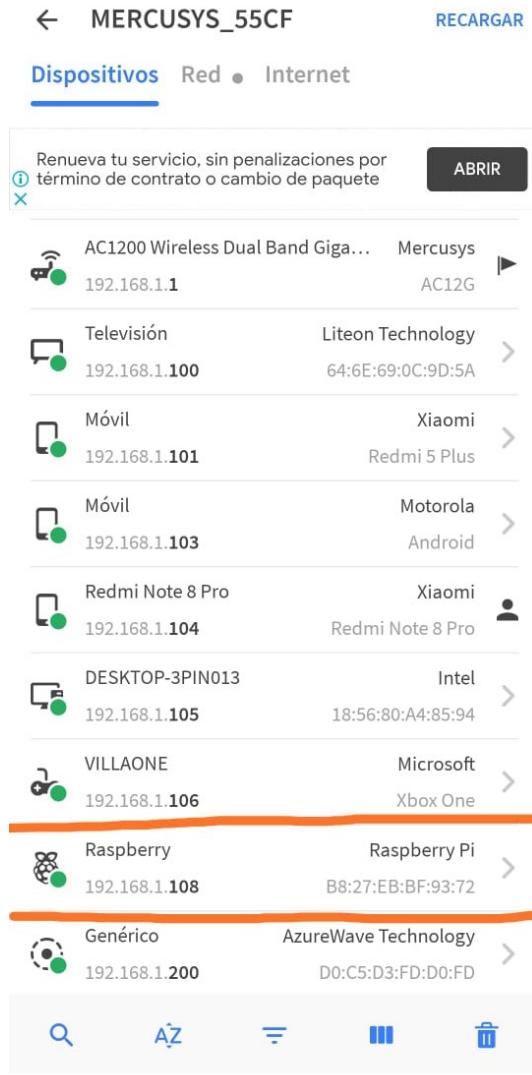


Figura 7.6: Información de dispositivos en la red utilizando Fing

Para realizar la conexión con la tarjeta, utilizamos el programa **Putty** el cual nos permite hacer uso del protocolo **SSH**, en donde proporcionamos el usuario y la IP para acceder a esta como se observa en la **figura 7.7**, utilizando **pi** como usuario y siendo la IP **192.168.1.108**.

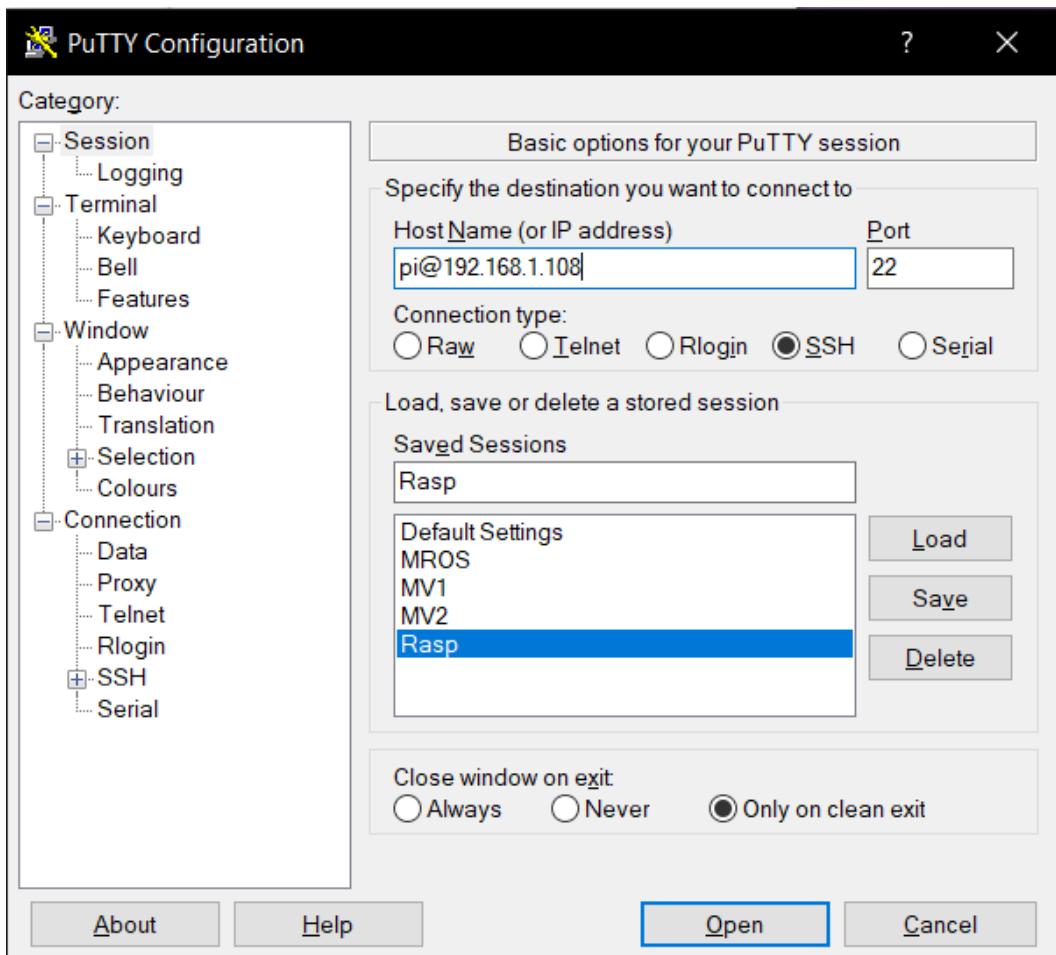


Figura 7.7: Conexión de Raspberry utilizando el protocolo SSH en Putty

Al dar en el botón **Open** se nos abrirá una ventana que nos mostraría la terminal de como se muestra en la **figura 7.8**, sin embargo, antes de ingresar para controlar la tarjeta esta requiere una contraseña que por defecto es **raspberry**, una vez ingresada tenemos acceso a todo el sistema operativo de Raspberry para poder hacer uso de este.

```
pi@raspberrypi: ~
Using username "pi".
pi@192.168.1.108's password:
Linux raspberrypi 5.4.51-v7+ #1333 SMP Mon Aug 10 16:45:19 BST 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Oct 30 22:32:23 2020 from 192.168.1.105

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi: ~ $
```

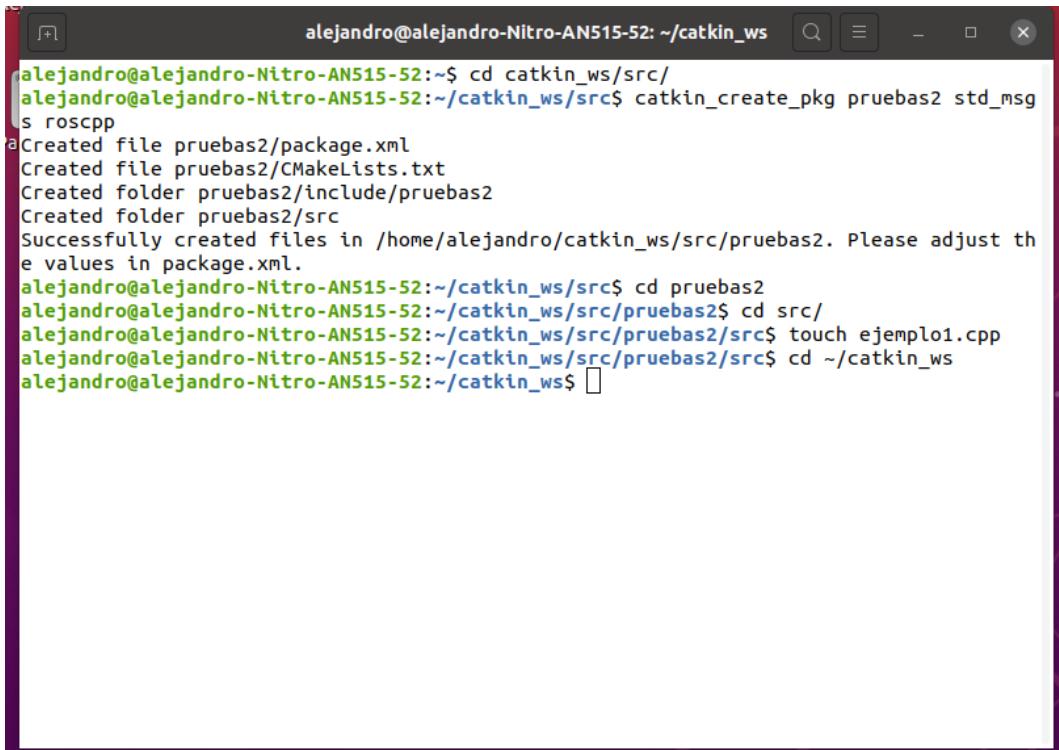
Figura 7.8: Terminal SSH para Raspberry OS en Putty

Hasta este punto la instalación del sistema operativo de Raspbian estará completa, sin embargo, al momento de querer instalar ROS en este sistema, no se nos concederán permisos, por este mismo motivo se utilizó otra imagen de sistema con Ubuntu 16.04 y Ros preinstalado para tener el mismo control de la tarjeta Raspberry Pi sin tener estos problemas de instalación de ROS. Para realizar la instalación de este otro sistema operativo son los mismos pasos que la anterior, solo se obtendrá otra imagen proporcionada por ubiquitirobotics [17].

7.3. Anexo C: Creación de paquetes de ROS

En este Anexo se mostrará la creación de paquetes en el entorno de ROS, para realizar esto como primer paso debemos abrir una nueva terminal y dirigirnos a nuestro espacio

de trabajo y situarnos en la carpeta src en este caso **/catkin_ws/src**, en esta ubicación se utilizará el comando **catkin_create_package nombre_paquete std_msgs roscpp**, en donde se especifica el nombre del paquete, el tipo de mensajes que este utilizará en este caso de tipo estándar y el lenguaje con el que se compilará siendo este C++, al ejecutar esta línea de comando nos mostrará un mensaje el cual nos informará los archivos creados. Después de crear este paquete, nos situaremos dentro del src del paquete **/catkin_ws/src/pruebas2/src** en este caso, dentro de esta carpeta crearemos el archivo de C++ el cual contendrá nuestro código utilizando el comando **touch ejemplo.cpp** teniendo que utilizar la extensión de archivo **cpp**, una vez creado este archivo seremos capaces de escribir nuestro código dentro de este para después compilarlo.



```
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws$ cd catkin_ws/src/
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws/src$ catkin_create_pkg pruebas2 std_msgs
s roscpp
aCreated file pruebas2/package.xml
Created file pruebas2/CMakeLists.txt
Created folder pruebas2/include/pruebas2
Created folder pruebas2/src
Successfully created files in /home/alejandro/catkin_ws/src/pruebas2. Please adjust the values in package.xml.
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws/src$ cd pruebas2
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws/src/pruebas2$ cd src/
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws/src/pruebas2/src$ touch ejemplo1.cpp
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws/src/pruebas2/src$ cd ~/catkin_ws
alejandro@alejandro-Nitro-AN515-52:~/catkin_ws$
```

Figura 7.9: Ejemplo de creación de paquete

Antes de compilar este archivo debemos crear el ejecutable al que se vinculará, dado que sin este no podremos hacer uso del mismo, para esto abriremos el archivo CMakeLists.txt creado dentro de nuestro paquete para modificarlo añadiendo el ejecutable como se muestra en la **figura 7.10**.

```

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#####
## Testing ##
#####

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_pruebas.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)

add_executable(listener_node src/listener.cpp)
target_link_libraries(listener_node ${catkin_LIBRARIES} )

```

Figura 7.10: Código para crear ejecutable

Creado este ejecutable seremos capaces de compilar nuestros códigos para esto debemos estar situados en la carpeta de trabajo desde la terminal, una vez aquí se ejecutará el comando **catkin_make** el cual compilará todos los códigos dentro de la carpeta de trabajo, al terminar el proceso de compilación podremos encontrar y ejecutar nuestro código utilizando el comando **rosrun pruebas2 listener_node** indicando la carpeta y el nombre del ejecutable.

7.4. Anexo D: Instalación de Ubuntu

Para realizar la instalación de Ubuntu 20.04, lo primero que se debe realizar es acceder a la página oficial de Ubuntu como se puede observar en la **figura 7.11** para descargar la imagen ISO del sistema operativo.

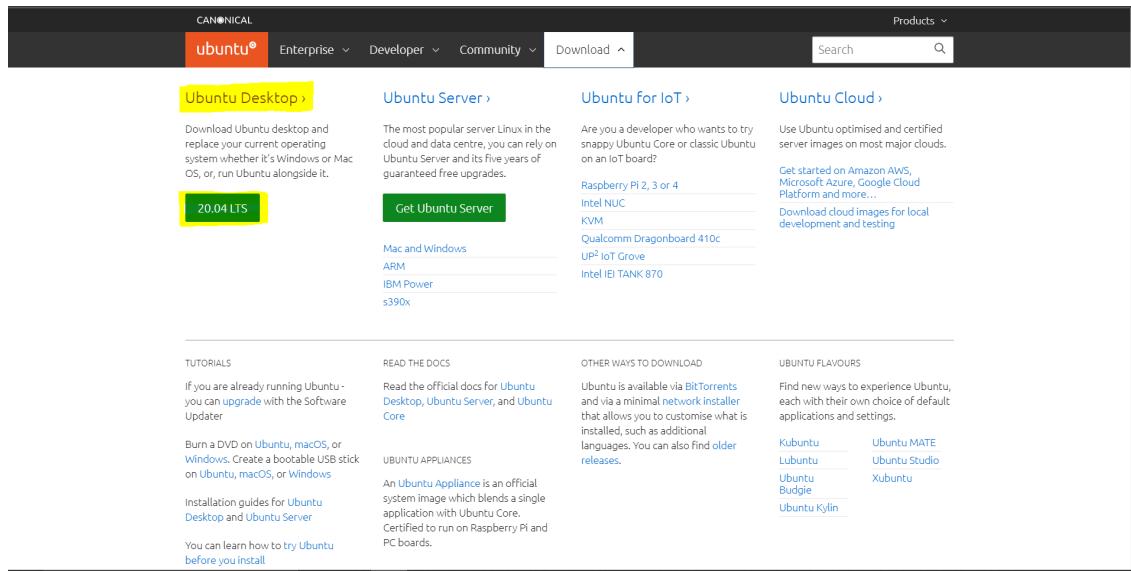


Figura 7.11: Página de descarga de Ubuntu 20.04

Una vez descargada la imagen ISO se utilizará el programa Rufus (veáse en la sección 3.9) para crear el medio de instalación en una memoria USB como se muestra en la **figura 7.12** en donde se puede observar las configuraciones realizadas para la creación.

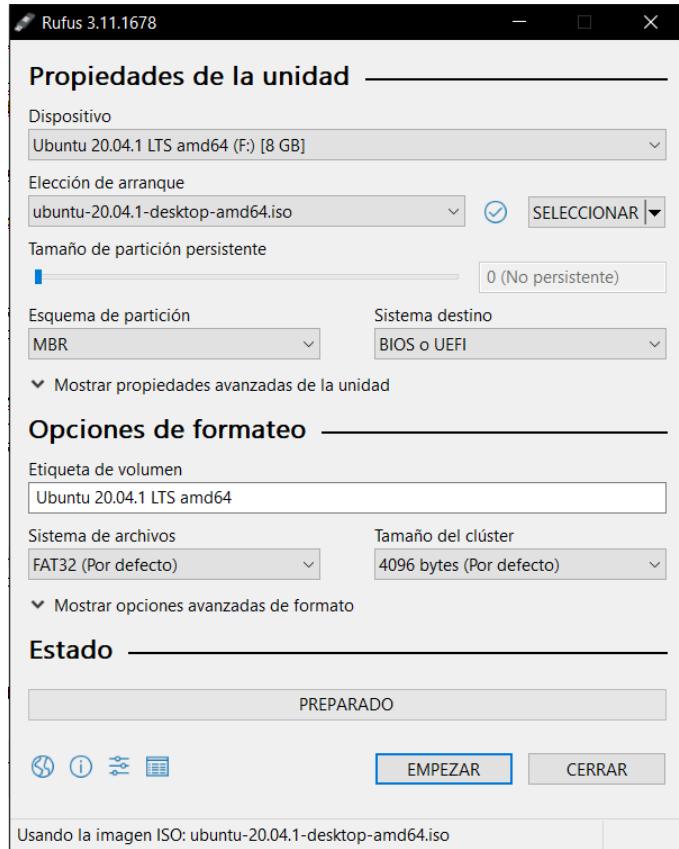


Figura 7.12: Creación de medio de instalación

Al haber creado el medio de instalación, se introducirá la memoria en la computadora, después de esto se reiniciará el equipo para entrar al modo de selección de dispositivo con el que inicia la computadora para seleccionar la memoria y de esta forma inicie el instalador de Ubuntu. Entrando a la interfaz de instalación nos aparecerán las pantallas para realizar las configuraciones, estas son:

- Selección del idioma del sistema.
- Selección de idioma de teclado.
- Descargar herramientas, drivers y aplicaciones (recomendado).

- Seleccionar como se instalará el sistema operativo, en este caso en un nuevo disco duro.
- Selección de país.
- Configuración de nombre de usuario y contraseña.

7.5. Anexo E: Instalación de ROS en ordenador

Una vez que Ubuntu esta instalado y funcionando en nuestro equipo se procede a realizar la instalación de las librerías de ROS, en este caso la instalación de ROS Noetic Ninjemys es sencilla, esto porque la Wiki de ROS nos proporciona un tutorial en GitHub incluso con video como se puede apreciar en la **figura 7.13**, aunque realmente solo es cuestión de ejecutar un comando en la terminal de Ubuntu, ya que, este comando descargará un script con todos los comandos necesarios para realizar la instalación.

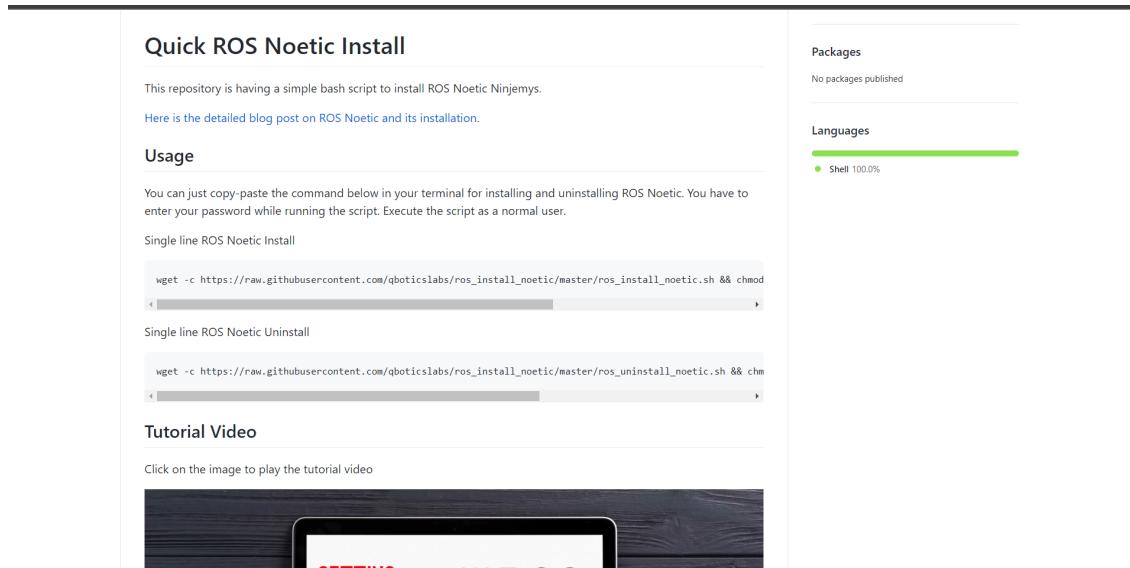


Figura 7.13: Repositorio de GitHub para la instalación de ROS Noetic Ninjemys

Bibliografía

- [1] Pablo Aguirre Cárcel. *Modelado, simulación y control de pequeños vehículos submarinos no tripulados*. PhD thesis, Universidad Politécnica de Cartagena, 2020.
- [2] Andrés Guillermo Velásquez Gómez David Albeiro Taborda Alvarez. Sistema de adquisición de datos de una unidad de navegación inercial y ros como herramienta de visualización. *Scientia et Technica Año XVIII*, 1:1–6, 2013.
- [3] Sergio Muñoz Ruiz. *Diseño de algoritmos de control de nivel bajo en un vehículo inteligente*. PhD thesis, Universidad Carlos III de Madrid, 2019.
- [4] Laboratorio de Inteligencia Artificial de Stanford (2007). ROS. Recuperado el 10 de Enero de 2020. Url: <https://www.ros.org>
- [5] Robotis. Turtlebot3 simulación en ROS. Recuperado el 10 de Septiembre de 2020. Url: <https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/>
- [6] Laboratorio de Inteligencia Artificial de Stanford. Documentación de ROS. Recuperado el 10 de Enero de 2020. Url: <http://wiki.ros.org>
- [7] Maxim Sokolov, Roman Lavrenov, Aidar Gabdullin, Ilya Afanasyev, and Evgeni Magid. 2016. 3D modelling and simulation of a crawler robot in ROS/Gazebo. In Proceedings of the 4th International Conference on Control, Mechatronics and Automation (ICCMA '16). Association for Computing Machinery, New York, NY, USA, 61–65. DOI:<https://doi.org/10.1145/3029610.3029641>
- [8] Joyanes Aguilar, Luis. Programación en C++: Algoritmos: Algoritmo, estructura de datos y objetos. *Madrid (España)*. McGraw-Hill. 2002.
- [9] Lawrence Rosen. (2005). *Open source licensing* (Vol. 692). Prentice Hall.
- [10] McRoberts, M. (2018). *Arduino básico*. Novatec Editora.
- [11] Richardson, M., & Wallace, S. (2012). *Getting started with raspberry PI*. “ O ’Reilly Media, Inc.”.

- [12] Koubaa, A. (Ed.). (2019). *Robot Operating System (ROS)* (Vol. 1, pp. 112-156). Springer.
- [13] Quigley, M., Gerkey, B., & Smart, W. D. (2015). *Programming Robots with ROS: a practical introduction to the Robot Operating System*. “ O ’Reilly Media, Inc.”.
- [14] Amsters, R., & Slaets, P. (2019, April). *Turtlebot 3 as a Robotics Education Platform*. In *International Conference on Robotics and Education RiE 2017* (pp. 170-181). Springer, Cham.
- [15] Flynn, I. M., McHoes, A. M., & Sánchez, G. (2001). Sistemas operativos (No. 001.61 F5.). Thomson.
- [16] Hiltunen, T. (2018). C++ robottikirjasto.
- [17] UbiquitiRobotics, (2019) Raspberry Pi Images. DOI:<https://www.ubiquityrobotics.com/>