

UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD DE INGENIERÍA



ROBÓTICA 2

Proyecto Final

Profesor:
Dr. Fernando BRUNETTI

Alumno:
Hans MERSCH
Carlos GAONA

Índice

1. Estado del Arte	2
2. Objetivos	2
2.1. Objetivo General	2
2.2. Objetivos Específicos	2
3. Marco Teórico	3
3.1. Hardware a Utilizar	3
3.1.1. Raspberry Pi	3
3.2. Software a Utilizar	3
3.2.1. ROS Noetic	3
3.2.2. Instalación del ROS en el Raspberry Pi	3
3.2.3. Instalación del ROS en la Maquina Virtual	7
3.2.4. Lenguaje de Programacion	9
4. Diagramas de Bloques	10
5. Diseño del Proyecto	12
5.1. Hardware	12
5.1.1. Raspberry PI	12
5.1.2. Sensores Ultrasónicos	13
5.1.3. SparkFun LSM9DS1	13
5.1.4. L298N	14
6. Implementación	15
6.1. Conexionados	15
6.1.1. Sensores Ultrasónicos	16
6.1.2. Motor con L298N	16
6.2. Conexión entre el Raspberry y la Computadora	17
6.3. Paradigmas de Comunicación	17
6.3.1. Cliente-Servidor	17
6.3.2. Publicador-Suscriptor	18
6.3.3. Paradigma Seleccionado	18
6.4. Sistemas de Navegación	18
6.4.1. Navegación Aleatoria	18
6.4.2. Navegación por Barrido Paralelo	20
6.5. Desafíos en la Implementación	21
7. Anexos	22
8. Conclusiones	27
Bibliografía	28

1. Estado del Arte

El sistema operativo de robot o *Robot Operating System*(ROS) es un conjunto de bibliotecas de software y herramientas que lo ayudan a crear aplicaciones de robot. Desde controladores hasta algoritmos de última generación y potentes herramientas de desarrollo, ROS está listo para usarse en una amplia gama de aplicaciones robóticas, desde interiores hasta exteriores, del hogar a la automoción, del agua al espacio y del consumidor a la industria[7].

Esta plataforma es de código abierto, es decir, tiene la flexibilidad de decidir dónde y cómo usar ROS, así como la libertad de personalizarlo según sus necesidades. Además, ROS no es exclusivo, no necesita elegir entre ROS o alguna otra pila de software; ROS se integra fácilmente con software existente para llevar las herramientas a problemas particulares.

Por este motivo, ROS se utiliza en toda la industria de la robótica como también en su enseñanza. Es la base para la mayoría de las investigaciones en robótica, desde proyectos de un solo estudiante hasta colaboraciones de múltiples instituciones y competencias a gran escala. Y son los robots internos los que se están ejecutando en producción en todo el mundo hoy en día.

Además, ROS es una plataforma madura, pues ya existe hace más de 10 años. El proyecto ROS ha producido un vasto ecosistema de software para robótica al nutrir una comunidad global de millones de desarrolladores y usuarios que contribuyen y mejoran ese software. ROS está desarrollado por y para esa comunidad, quienes serán sus administradores en el futuro.

En este trabajo emplearemos esta plataforma para el control de un robot móvil que sea capaz de realizar tareas varias, tales como el movimiento aleatorio con evasión de obstáculos, el barrido en paralelo de una zona específica y el movimiento de un punto específico A a otro punto B.

2. Objetivos

2.1. Objetivo General

- Desarrollar un robot móvil que funcione a bajo una arquitecturas ROS (Robot Operating System) y que cumpla ciertas funciones de navegación

2.2. Objetivos Específicos

- Diseñar de una estructura robótica móvil.
- Construir e implementar del robot móvil.
- Diseñar e implementar de un sistema de navegación en ROS.
- Controlar de actuadores en lazo cerrado.
- Diseñar e implementar una tarea asignada para el robot.

3. Marco Teórico

3.1. Hardware a Utilizar

3.1.1. Raspberry Pi

Como el software a utilizar depende plenamente del hardware a utilizar, la selección del hardware adecuado nos permitirá definir el software a emplear y la metodología que se implementara. El ROS es compatible con la familia del Raspberry Pi y, de acuerdo a la versión del hardware disponible, se emplearía una versión diferente del ROS:

Raspberry Pi 3 Modelo B en adelante: Te permite trabajar con *ROS Noetic*, la cual es la ultima versión del ROS en la actualidad. Emplear este modelo nos permitiría trabajar con los drivers mas actualizados, es compatible con Python3, posee una comunidad activa, y esta soportada por el proyecto ROS.

Raspberry Pi 3 Modelo A y versiones anteriores: Sera necesario emplear el *ROS Kinetic*, la cual es la versión anterior al Noetic. Esta versión es compatible con Python2 ya ademas ya no posee el soporte del proyecto ROS, por lo que la comunidad ya casi no la utiliza.

Los miembros de este grupo cuentan con un Raspberry Pi 3 Modelo B(RPi3B) y un Raspberry Pi 3 Modelo A(RPi3A), el Laboratorio de Automatización de Robótica(LAR) cuenta solo con Raspberry Pi Modelo 2 o versiones similares. Como el precio del Raspberry se incremento significativamente por la pandemia del Covid-19, comprar uno nuevo no es una opción viable, pues la inversión en el dispositivo rondaría los Gs. 1.200.000. Teniendo en cuenta los equipos que tienen disponible los integrantes del grupo, se empleara el RPi3B para el robot móvil, dejando el RPi3A como equipo de respaldo en caso de que sea necesario.

3.2. Software a Utilizar

3.2.1. ROS Noetic

Teniendo en cuenta el hardware a utilizar, se empleara el ROS Noetic Ninjemys, también conocido como ROS Noetic, para el control del robot, tanto en el Raspberry como en la computadora. Esta es la versión 13 del ROS, y fue lanzada el 23 de Mayo del 2020. Esta version esta optimizada para operar primordialmente en las versiones de Ubuntu y Debian en Linux, pero también existen versiones experimentales para Windows y Arch Linux.

Como buscamos la mayor simplicidad y evitar tanto como sea posible cualquier problema de compatibilidad, para el control por computadora optamos por emplear una maquina virtual para correr Ubuntu dentro de Windows. Como ROS Noetic es la versión actual de ROS, es necesario instalar la versión mas actualizada de Ubuntu, la version *20.04 LTS*. Como trabajaremos con Python3, emplearemos *Visual Studio Code* como plataforma para desarrollar nuestros scripts de Python. A continuación, se explicara con mas detalles la antelación del ROS Noetic en cada hardware a utilizar.

3.2.2. Instalación del ROS en el Raspberry Pi

La instalación del ROS en el Raspberry presento ciertas dificultades. Nosotros contamos con el *Raspberry Pi Model B*, sin embargo, todas las guías disponibles en internet están enfocadas en el *Raspberry Pi 4*. Pese a que no contábamos con el modelo recomendado, siguiendo la guías disponibles en foros de internet[2], logramos instalar el programa en nuestro Raspberry. A continuación, se presentaran los pasos a tener en cuenta para instalar el ROS en el Raspberry Pi.

3.2.2.1 Instalación del Sistema Operativo

ROS Noetic, la ultima versión del ROS, solo es compatible con el *Raspberry Pi OS Buster*, el cual es una versión legacy del sistema operativo del Raspberry. la imagen de dicha versión en el siguiente link. Para descargar la imagen correcta, seleccionar la opción visualizada en la **figura 1**

Para poder cargar la imagen en la memoria SD del Raspberry, se utiliza *balenaEtcher*. balenaEtcher es un software utilitario libre y de código abierto que se utiliza para escribir archivos de imagen, como archivos .iso y .img, así como carpetas comprimidas en medios de almacenamiento para crear tarjetas de memoria SD Live y unidades flash USB. Está desarrollado por balena y con licencia Apache License 2.0. Para descargar este software, es necesario acceder al siguiente link.

Una vez instalado el software, se selecciona la imagen a instalar, se selecciona el objetivo, en este caso la tarjeta SD, y se carga la imagen en el dispositivo.

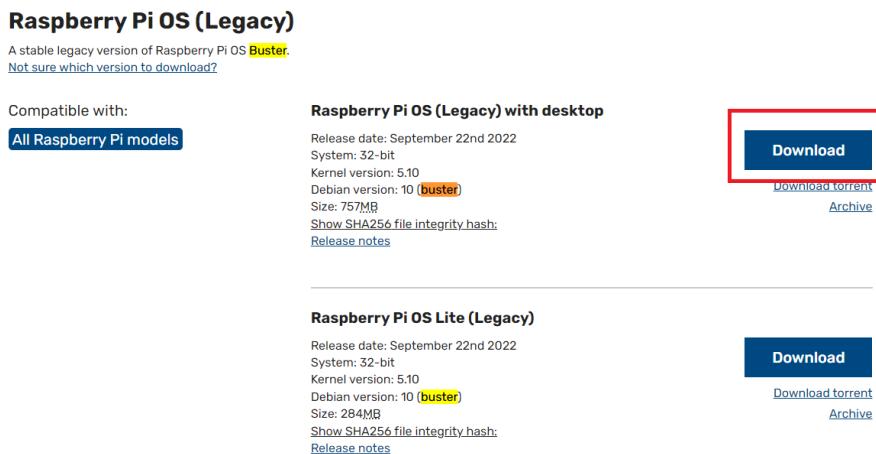


Figura 1: Selección de la imagen a descargar

3.2.2.2 Primeros Pasos con el Raspberry Pi OS

Luego de bootear por primera vez el Raspberry, y realizar las configuraciones iniciales, es necesario ejecutar los siguientes comandos en el terminal:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

El primer comando verifica si están instalados las ultimas versiones de los drivers y, de no ser así, los instala. El segundo comando verifica si están instalados las ultimas versiones de los paquetes y, de no ser así, los instala. Finalmente, el ultimo comando reinicia el OS para aplicar los cambios. También es necesario corroborar que se haya instalado la versión correcta del Raspberry Pi OS, la versión *buster*. Esto puede verificarse cargando el comando de la **figura 2**.

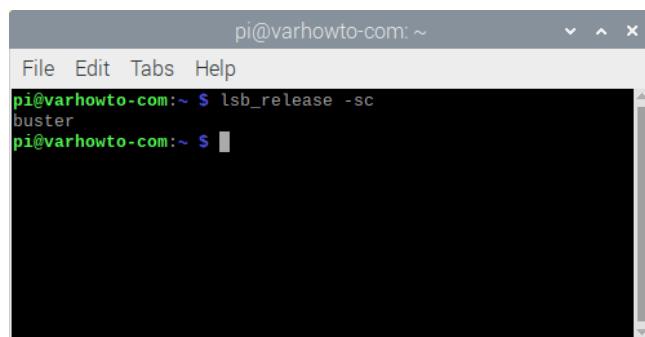


Figura 2: Verificación de la versión del Raspberry Pi OS

3.2.2.3 Incrementar el Swap

El espacio de Swap en Linux se usa cuando la cantidad de memoria física (RAM) está llena. Si el sistema necesita más recursos de memoria y la RAM está llena, las páginas inactivas en la memoria se mueven al espacio de Swap. Si bien el espacio de Swap puede ayudar a las máquinas con una pequeña cantidad de RAM, no debe considerarse un reemplazo para más RAM. El espacio de Swap se encuentra en los discos duros, que tienen un tiempo de acceso más lento que la memoria física[1]. Para modificar el Swap, es necesario cargar los siguientes comandos en el terminal:

```
sudo dphys-swapfile swapoff
sudoedit /etc/dphys-swapfile
```

Estos comandos nos permiten acceder al swapfile, donde podremos modificar la cantidad de memoria asignada al espacio Swap. Dentro del swapfile, es necesario localizar *CONF_SWAPSIZE = 100* (el 100 era el valor por defecto en nuestro Raspberry, este valor puede variar para otros), reemplazar el 100 por 2048, y guardar las modificaciones realizadas al archivo. A continuación, se ejecutan los siguientes comandos:

```
sudo dphys-swapfile setup
sudo dphys-swapfile swapon
```

Esta modificación permite que el espacio Swap aumente a 2GB. En otros dispositivos, es posible aumentar aun mas el tamaño del espacio del Swap. Sin embargo, el Raspberry Pi permite tener hasta 2GB de espacio Swap. Podemos corroborar que la modificación del espacio Swap fue exitosa con el comando de la **figura 3**.

```
pi@rpi:~ $ free -m
total        used       free     shared   buff/cache   available
Mem:      1626         711        255        24       658        803
Swap:    1023         286        737
pi@rpi:~ $
```

Figura 3: Verificación del espacio Swap

Nota: En el tutorial de referencia se utilizo un Raspberry pi 4 y solo era necesario añadir 1 GB al espacio Swap, como nosotros trabajamos con un Raspberry mas antiguo, se asigno 2GB al espacio Swap en vez de 1GB. Por lo tanto, el en terminal debe figurar un numero alrededor de 2048 en vez de 1023.

3.2.2.4 Configuración del repositorio de ROS Noetic en el Raspberry

Para instalar Noetic en Raspberry Pi 4, ahora agregaremos el repositorio oficial de ROS Debian al sistema operativo. Esto se logra a partir del siguiente comando:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
buster main" > /etc/apt/sources.list.d/ros-noetic.list'
```

Si la instalación fue exitosa, no habrá ninguna entrada. Sin embargo, podemos comprobar que se el repositorio se instaló correctamente empleando el comando `cat /etc/apt/sources.list.d/ros-noetic.list`, y se vera en el terminal lo que se muestra en la **figura 4**.

```
pi@varhowto-com:~ $ sudo sh -c 'echo "deb http://packages.ros.org/
ros/ubuntu buster main" > /etc/apt/sources.list.d/ros-noetic.list'
pi@varhowto-com:~ $ cat /etc/apt/sources.list.d/ros-noetic.list
deb http://packages.ros.org/ros/ubuntu buster main
pi@varhowto-com:~ $
```

Figura 4: Verificación del repositorio instalado

3.2.2.5 Agregar la clave oficial de ROS

Antes de ejecutar el comando `install Noetic`, primero agregaremos la clave ROS, que no es específica de Noetic sino para todas las distribuciones ROS, para asegurarnos de que instalaremos paquetes ROS autenticados para instalar en el Raspberry y evitar que los piratas informáticos intercepten nuestro tráfico de red.

Al ejecutar el siguiente comando, descargaremos la clave del servidor de claves de Ubuntu (`keyserver.ubuntu.com`) en el conjunto de claves de confianza:

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Una vez ejecutado dicho comando, el resultado en el terminal debe ser similar al de la **figura 5**.

```

pi@varhowto-com:~ $ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE886B172B4F42ED6FBAB17C654
Executing: /tmp/apt-key-gpghome.SG8pNIqy0T/gpg.1.sh --keyserver hkp://keyserver.ubuntu.com:80 --recv-key C1CF6E31E6BADE886B172B4F42ED6FBAB17C654
gpg: key F42ED6FBAB17C654: public key "Open Robotics <info@osrfoundation.org>" imported
gpg: Total number processed: 1
gpg: imported: 1
pi@varhowto-com:~ $ 

```

Figura 5: Verificación del repositorio instalado

3.2.2.6 Extracción de Metadatos de los paquetes del ROS Noetic

Antes de la instalación del ROS Noetic, es necesario actualizar el índice del paquete ROS para obtener toda la información del paquete Noetic del repositorio que agregamos. Para realizar esto, se ejecuta el siguiente comando:

```
sudo apt update
```

3.2.2.7 Instalación de Dependencias de Compilación en el Raspberry

Procedemos a la instalación del Noetic en el Raspberry. Crearemos un archivo *rosinstall* y obtendremos los paquetes individuales del repositorio de GitHub y los compilaremos. Para instalar todas las dependencias, es necesario ejecutar el siguiente código:

```
sudo apt-get install -y python-rosdep python-rosinstall-generator python-wstool python-rosinstall build-essential cmake
```

3.2.2.8 Configuración de Fuentes/Repositorios de Dependencia de ROS Noetic

Para instalar el escritorio de ROS Noetic en el Raspberry, es necesario inicializar *rosdep* para luego saber en donde encontrar las dependencias de Noetic. Posteriormente, es necesario obtener información del paquete de los repositorios que se acaban de instalar. Para realizar estas acciones, es necesario ejecutar las siguientes líneas de código:

```
sudo rosdep init
rosdep update
```

3.2.2.9 Instalación de las Dependencias del ROS Noetic

Primeramente, es necesario crear un espacio de trabajo *catkin*, y dirigirnos a dicho espacio de trabajo. Para ello, ejecutamos los siguientes comandos:

```
mkdir ~/ros_catkin_ws
cd ~/ros_catkin_ws
```

Dentro del espacio de trabajo se utiliza la función *rosinstall_generator* para generar una lista de dependencias de Noetic para diferentes variantes de Noetic, las cuales son:

desktop-full: Es el paquete tradicional, cuenta con simuladores y percepción en 2D/3D.

desktop: Cuenta con todo lo que trae *ROS-Base*, y las herramientas *rqt* y *rviz*.

ros_comm: Conocido también como el Bare Bones(solo los huesos), incluye el paquete de ROS, y las librerías de creación y comunicación. No incluye herramientas del GUI.

Si bien la instalación de la versión *desktop-full* sería la instalación ideal, el Raspberry cuenta con la potencia para correr exclusivamente la versión *ros_comm*. Al tratar de instalar las otras versiones, se generan errores en el terminal. Entonces, el comando para instalar la versión *ros_comm* es:

```
rosinstall_generator ros_comm --rosdistro noetic --deps --
wet-only --tar > noetic-ros_comm-wet.rosinstall
```

Es necesario obtener todos los repositorios remotos especificados para correr esta versión del ROS Noetic. Esto se logra empleando la herramienta *wstool*, la cual es llamada empleando la siguiente instrucción:

```
wstool init src noetic-ros_comm-wet.rosinstall
```

También es necesario compilar los paquetes adquiridos, e instalar todas las dependencias del sistema. esto se logra con la siguiente instrucción:

```
rosdep install -y --from-paths src --ignore-src --rosdistro
noetic -r --os=debian:buster
```

3.2.2.10 Compilación de Paquetes de Noetic en el Raspberry

Para compilar los paquetes, directamente se ejecuta el siguiente comando:

```
sudo src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/noetic -
j1 -DPYTHON_EXECUTABLE=/usr/bin/python3
```

Para comprender lo que realiza este comando, es conveniente segmentarlo:

- **-DCMAKE\$_BUILD\$_TYPE=Release:** Usar Release para que no genere símbolos de depuración.
- **--install-space /opt/ros/noetic:** Todo se instalará en /opt/ros/noetic, igual que si instalamos Noetic en Ubuntu.
- **-j1:** Por defecto, catkin usará todos los núcleos para los trabajos de compilación, pero se convierte en un problema en Raspberry porque tiene memoria limitada y cada trabajo de compilación consume memoria. Limitar el número de trabajos a 1 reduce la posibilidad de que se produzca un problema de poca memoria.
- **-DPYTHON\$_EXECUTABLE=/usr/bin/python3:** Aquí especificamos Python3 como el ejecutable de Python. Esto es muy importante porque Noetic solo admite 3, a diferencia de las distribuciones ROS anteriores como Melodic.

Tras correr el comando, se tardaran aproximadamente 40 mins en compilar todos los paquetes. Una vez compilados los paquetes, la instalación del ROS ha finalizado.

3.2.2.11 Configuraciones Adicionales y Prueba de Funcionamiento

Si bien el ROS ya se instaló, es conveniente añadir algunas líneas de comando al archivo *bashrc* del Raspberry para que la ejecución del ROS sea más simplificada. El archivo *bashrc* debería encontrarse en el directorio *HOME* del Raspberry. Una vez que hayamos accedido al bash, se debe pegar el siguiente código:

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

La primera linea añade el directorio del ROS Noetic a la lista de comandos que se pueden ejecutar al iniciar el terminal. La segunda linea nos permite acceder directamente al directorio de los entornos de trabajo desde el terminal.

Una vez guardados los cambios en el *bashrc*, abrimos dos terminales, una en una terminal ejecutamos el comando *roscore*, y nos debe inicializar el ROS. En la otra terminal, se debe ejecutar el comando *roscl*, y debe dirigirnos a la carpeta de entornos de trabajo *catkin*

3.2.3. Instalación del ROS en la Maquina Virtual

Para la instalación del ROS Noetic en la Maquina Virtual con Ubuntu 20.04 LTS se emplea la guía disponible en la ROS Wiki, la cual será resumida a continuación.

3.2.3.1 Configurar los Repositorios de Ubuntu

Es necesario configurar los repositorios de Ubuntu para permitir los repositorios *Restricted*, *Universe* y *Multiverse*. Para ello, se ingresa a la pestaña de software de Ubuntu, y se chequean dichos repositorios, como se puede apreciar en la **figura 6**. Posteriormente, se selecciona “Cerrar” para guardar los cambios, lo cual provocará la aparición un cuadro de diálogo que le pregunta si se desea actualizar la lista de repositorios. Se selecciona “Recargar” para actualizar la lista.

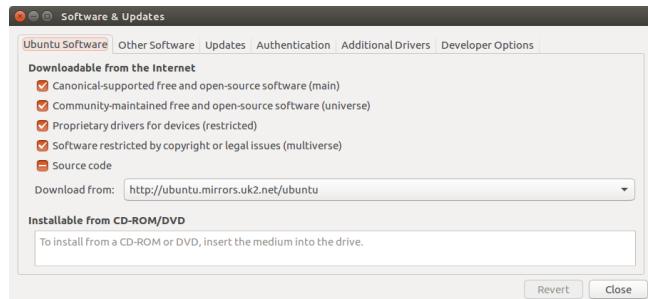


Figura 6: Configuración de Repositorios a utilizar en Ubuntu

3.2.3.2 Configurar la Lista de Fuentes y las Claves

Configure su computadora para aceptar software de packages.ros.org. Es necesario configurar el sistema operativo para que este pueda aceptar paquetes de software de ros.org. Esto se logra con el siguiente comando:

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

Para las claves, ejecutar directamente el siguiente comando:

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3.2.3.3 Instalación

Existen 3 versiones de ROS Noetic que son instalables

Ros-Base: También conocida como *Bare-Bones*, contiene solo los paquetes básicos de ROS, el build y las librerías de comunicación. No posee herramientas de GUI. Utilizado en hardware de potencia limitada

Desktop: Contiene todo lo que esta disponible en la verssion ROS-Base, y tambien incluye herramientas tales como *rqt* y *rviz*.

Desktop-Full: La version recomendada, contiene todo lo que esta disponible en la version Desktop, y tambien incluye simuladores 2D/3D y paquetes de percepción 2D/3D.

Como la computadora posee todos los recursos necesarios, se instalara la versión *Desktop-Full*. Para ello, se ejecutan los siguientes comandos:

```
sudo apt update
sudo apt install ros-noetic-desktop-full
```

También es posible instalar otros paquetes que no están instalados por defecto empleando la siguiente sintaxis:

```
sudo apt install ros-noetic-PACKAGE
```

Donde *PACKAGE* es el nombre del paquete a instalar

3.2.3.4 Configuración del Entorno

Primeramente, es necesario anadir unas lineas de comando al bash del SO de linux para poder ejecutar comandos de ROS en la terminal.

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

3.2.3.5 Dependencias para la Contrucción de Paquetes

Una vez instalado los paquetes necesarios para ejecutar los paquetes principales de ROS, para crear y administrar sus propios espacios de trabajo de ROS, existen varias herramientas y requisitos que se distribuyen por separado. Por ejemplo, rosinstall es una herramienta de línea de comandos de uso frecuente que le permite descargar fácilmente muchos árboles fuente para paquetes ROS con un solo comando. Para instalar esta herramienta y otras dependencias para crear paquetes ROS, se ejecuta el siguiente comando:

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

3.2.3.6 Inicializar el Rosdep

Antes de que poder utilizar muchas herramientas de ROS, es necesario inicializar *rosdep* . *rosdep* permite la instalación sencilla de las dependencias del sistema para la fuente que se desee compilar y es necesario para ejecutar algunos componentes principales en ROS. Su instalación se realiza con el siguiente comando:

```
sudo apt install python3-rosdep
```

Para su inicialización, se ejecuta el siguiente comando:

```
sudo rosdep init  
rosdep update
```

Una vez inicializado el *rosdep*, el ROS esta listo para su utilización.

3.2.4. Lenguaje de Programacion

El ROS Noetic nos permite emplear los siguientes lenguajes de programacion:

C++: es un lenguaje de programación diseñado en 1979. La intención de su creación fue extender al lenguaje de programación C y añadir mecanismos que permiten la manipulación de objetos. Si bien es bastante rápido y, por su longevidad, es un lenguaje de programacion bastante maduro, su sintaxis es poco amigable con programadores nuevos.

Python3: Es la versión mas actualizada de Python. Cuenta con un soporte y comunidad activos. Es amigable con los nuevos usuarios y hoy en dia, hay librerías para hacer de todo un poco.

Como los integrantes del grupo cuentan con mas experiencia en Python que en C++, se optó por utilizar Python como lenguaje de programación para este proyecto.

4. Diagramas de Bloques

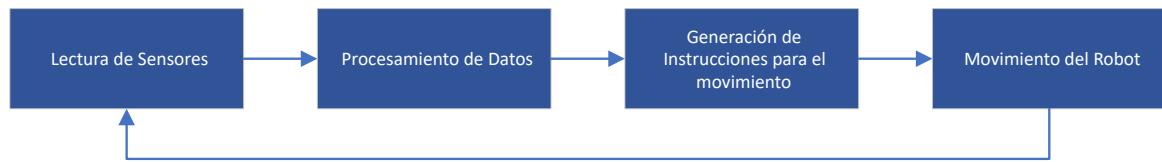


Figura 7: Esquema de control por lazo cerrado del proyecto

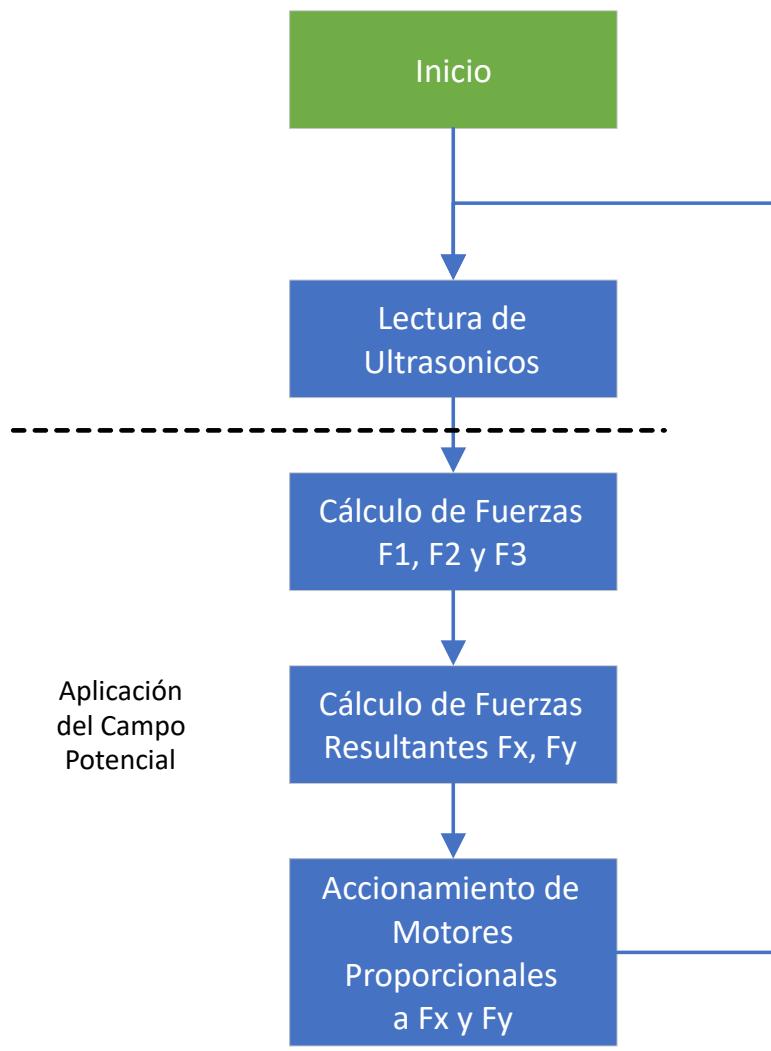


Figura 8: Esquema de funcionamiento de la navegación aleatoria

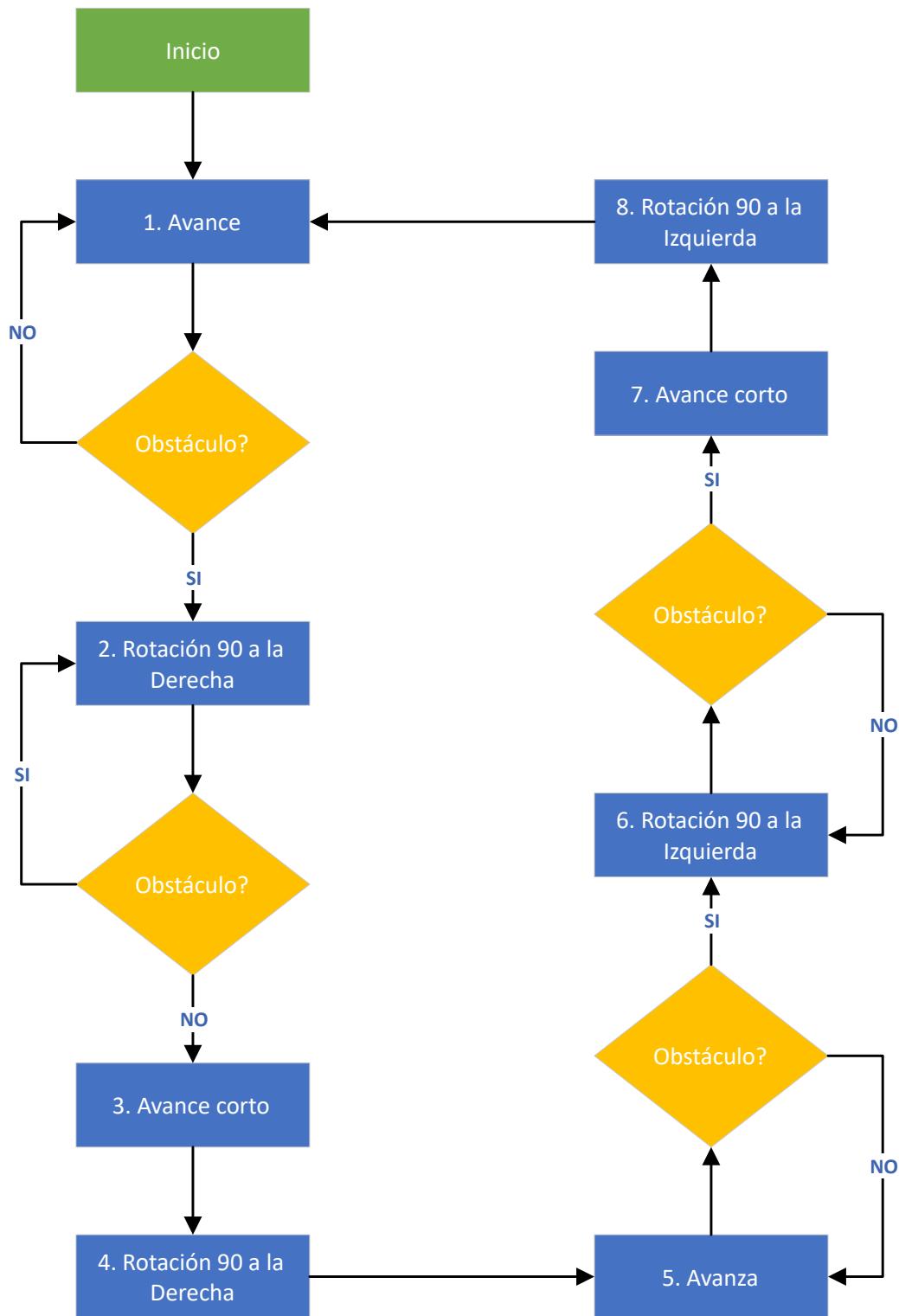


Figura 9: Esquema de funcionamiento de la navegación por barrido en paralelo

5. Diseño del Proyecto

5.1. Hardware

5.1.1. Raspberry PI

El Raspberry PI es un microprocesador muy versatil que posee una comunidad activa, el sistema operativo es Raspberry PI OS, el cual esta basado en una distribucion de Linux. Nuestro grupo contaba con un Raspberry PI 3 modelo B, el cual tiene las siguientes especificaciones:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 LAN inalambrico y Bluetooth Low Energy (BLE) en la placa
- 100 Ethernet de base
- 40-pin extendidos de GPIO
- 4 puertos USB
- 4 salida unipolar de stereo y puerto de video compuesto
- HDMI
- Puerto para camara CSI
- Puerto para display DSI
- Puerto Micro SD para cargarle el sistema operativo y los archivos
- Fuente de alimentación por Micro USB hasta 2.5A

Las principales ventajas del Raspberry sobre el Arduino es que el Raspberry ya trae incorporado un receptor WiFi, y el uso de una distribución de Linux como sistema operativo supondría ciertas facilidades mas adelante cuando sea necesario conectarse al ROS en Ubuntu.

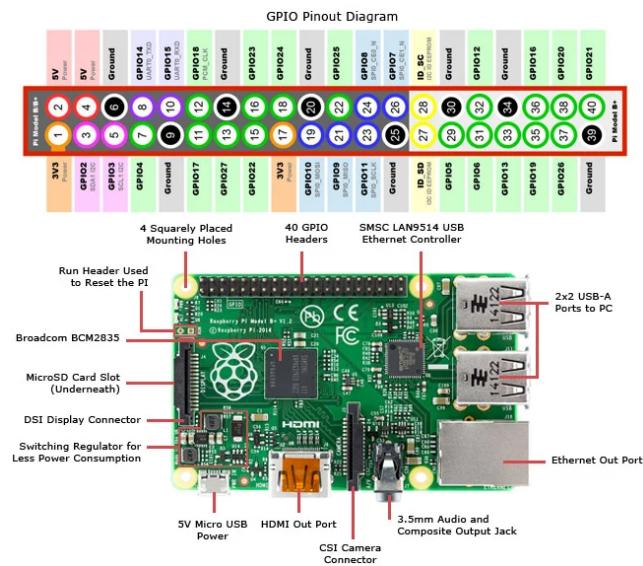


Figura 10: Representación física del Raspberry PI 3 modelo B y sus terminales

5.1.2. Sensores Ultrasónicos

El sensor HC-SR04 es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio. El sensor HC-SR04 es el más utilizado dentro de los sensores de tipo ultrasonido, principalmente por la cantidad de información y proyectos disponibles en la web. De igual forma es el más empleado en proyectos de robótica como robots laberinto o sumo, y en proyectos de automatización como sistemas de medición de nivel o distancia.

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoelectricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoelectrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG, las ondas de sonido viajan en el aire y rebotan al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoelectrico, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar)[6].

La distancia se puede calcular utilizando la siguiente formula:

$$Distancia[m] = \frac{\text{Tiempo del pulso ECHO}[s] * \text{Velocidad del sonido}[\frac{m}{s}]}{2}$$

5.1.3. SparkFun LSM9DS1

El SparkFun LSM9DS1 Breakout es un sistema versátil de detección de movimiento en un chip. Alberga un acelerómetro de 3 ejes, un giroscopio de 3 ejes y un magnetómetro de 3 ejes, es decir, posee nueve grados de libertad (9DOF). El LSM9DS1 es compatible con I2C y SPI, dos protocolos de comunicación muy utilizados que se pueden implementar en el Raspberry.



Figura 11: HC-SR04



Figura 12: LSM9DS1

5.1.4. L298N

El módulo controlador de motores L298N H-bridge nos permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla, gracias a los 2 los dos H-bridge que monta. El puente-H o H-bridge es un componente formado por 4 transistores que nos permite invertir el sentido de la corriente, y de esta forma podemos invertir el sentido de giro del motor.

El rango de tensiones en el que trabaja este módulo va desde 3V hasta 35V, y una intensidad de hasta 2A. A la hora de alimentarlo hay que tener en cuenta que la electrónica del módulo consume unos 3V, así que los motores reciben 3V menos que la tensión con la que alimentemos el módulo.

Además el L298N incluye un regulador de tensión que nos permite obtener del módulo una tensión de 5V, perfecta para alimentar nuestro Arduino. Eso sí, este regulador sólo funciona si alimentamos el módulo con una tensión máxima de 12V. Es un módulo que se utiliza mucho en proyectos de robótica, por su facilidad de uso y su reducido precio[5].

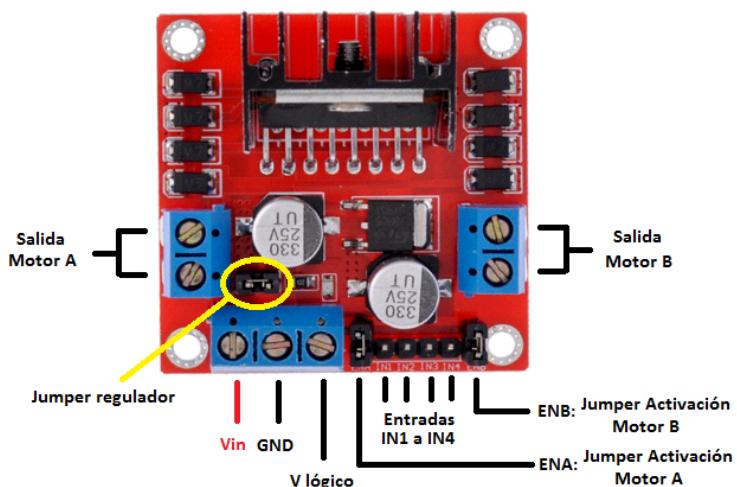


Figura 13: Descripción de los pines del L298N

6. Implementación

6.1. Conexionados

Para nuestra fortuna, el Raspberry Pi 3 Modelo B cuenta con las entradas y salidas necesarias para esta implementación. En la **figura 14** se muestra los pines empleados y los usos para cada pin. En las siguientes secciones, se detallara el conexionado para cada elemento del robot individualmente.

Pin	Funcion	Se conecta a...
1	3.3VDC	MPU(VCC)
2	5VDC	Ultrasonicos (Vcc)
3	SDA1(I1C)	MPU(SDA)
4		
5	SCL1(I2C)	MPU(SCL)
6	GND	MPU(GND)
7		
8		
9		
10		
11	GPIO 0	Ultrasonico 1 (Trig)
12		LM298(ENA)
13	GPIO 2	ultrasonico 1 echo
14		
15	GPIO 3	Ultrasonico 3 (Trig)
16		LM298(IN1)
17		
18		LM298(IN2)
19		
20		

(a) Conexión de los primeros 20 pines

Pin	Funcion	Se conecta a...
21		Encoder 2
22	GPIO 6	Conversor (a definir)
23		Encoder 2
24		
25		
26		
27		
28		
29	GPIO 21	motor
30		
31	GPIO 22	motor
32		
33	GPIO 23	motor
34		
35		
36	GPIO 27	Ultrasonico 2 (Trig)
37	GPIO 25	conversor (a definir)
38		Encoder 1
39	GND	Ultrasonicos (Gnd)
40		Encoder 1

(b) Conexión de los ultimos 20 pines

Figura 14: Pines empleados en el Raspberry Pi

6.1.1. Sensores Ultrasónicos

Para la conexión del sensor ultrasónico al Raspberry se empleo el conexionado de la **figura 15** para cada sensor ultrasónico. Se emplea un convertidor lógico de tensión entre el Raspberry y el sensor ultrasónico para realizar la conversión de 5 a 3.3 [V].

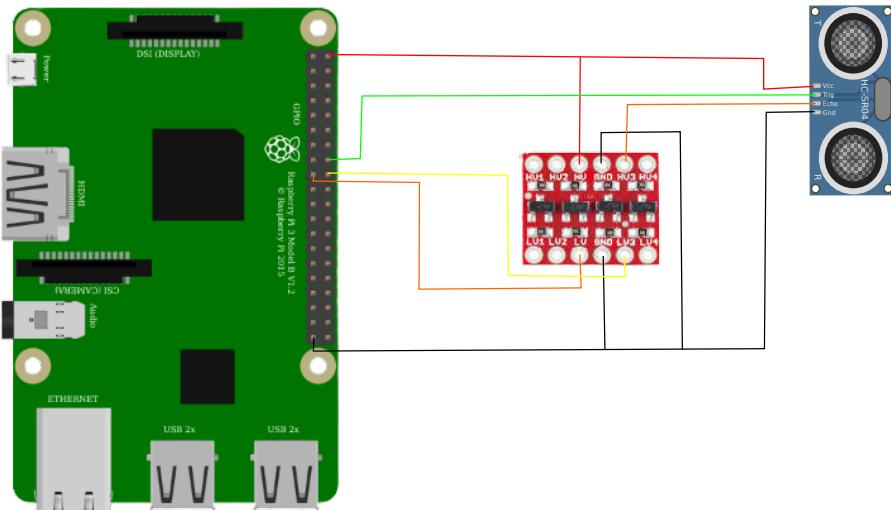


Figura 15: Conexionado del sensor ultrasónico

6.1.2. Motor con L298N

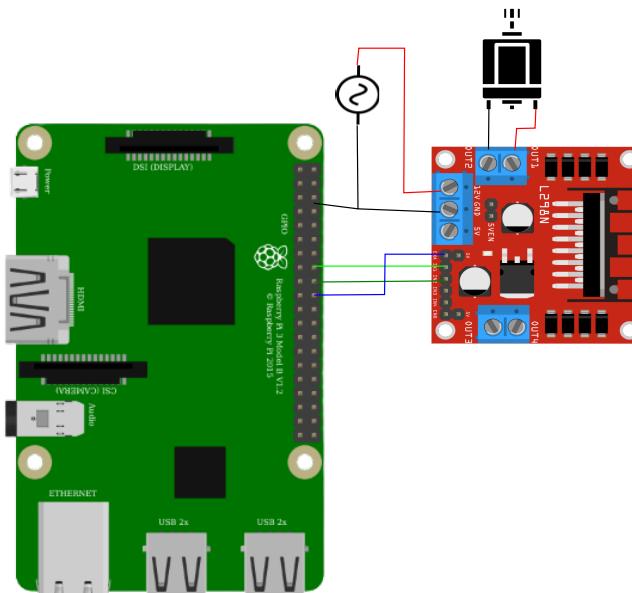


Figura 16: Conexionado del motor con el puente H

6.2. Conexión entre el Raspberry y la Computadora

Para realizar la comunicación entre el Raspberry y la computadora, es necesario obtener las direcciones de IP de ambos dispositivos, los cuales van conectados a la misma red. Esto se puede lograr ejecutando el comando *ifconfig* en el terminal. Luego, es necesario realizar modificaciones en el *bashrc* de ambos dispositivos. Para el esclavo, es necesario copiar los siguientes comandos:

```
export ROS_IP=192.168.205.168
export ROS_HOSTNAME=192.168.205.168
export ROS_MASTER_URI=http://192.168.205.127:11311

echo "ROS_HOSTNAME: \"$ROS_HOSTNAME"
echo "ROS_IP: \"$ROS_IP"
echo "ROS_MASTER_URI: \"$ROS_MASTER_URI"
```

La *ROS_IP* y el *ROS_HOSTNAME* contienen la dirección de IP del esclavo, mientras que el *ROS_MASTER_URI* contiene la dirección de IP del maestro. En el maestro, es necesario copiar los siguientes comandos:

```
export ROS_IP=192.168.205.127
export ROS_HOSTNAME=192.168.205.127
export ROS_MASTER_URI=http://hostname:11311

echo "ROS_HOSTNAME: \"$ROS_HOSTNAME"
echo "ROS_IP: \"$ROS_IP"
echo "ROS_MASTER_URI: \"$ROS_MASTER_URI"
```

La *ROS_IP* y el *ROS_HOSTNAME* contienen la dirección de IP del maestro. Una vez configurados ambos dispositivos, se ejecuta *rosrun* en ambos y el sistema debería estar listo para realizar comunicaciones entre dispositivos. Esto nos permite mandar información sobre los sensores del Raspberry a la computadora, y los comandos de control de la computadora al Raspberry.

6.3. Paradigmas de Comunicación

Para la navegación es necesario implementar el ROS tanto en la computadora maestra como también en el Dispositivo esclavo, en este caso, el Raspberry Pi. La arquitectura de ROS está compuesta por nodos interconectados entre sí y, según el paradigma a utilizar, la comunicación entre nodos puede variar. Existen dos paradigmas de comunicación: el publicador-suscriptor y el cliente-servidor.

6.3.1. Cliente-Servidor

La relación cliente-servidor, también conocida como solicitud-respuesta, es cuando un sistema que necesita datos los solicita a un servidor que los tiene. Dado que las entidades que solicitan datos están separadas de los proveedores de datos, la forma más fácil de pensar en este paradigma es imaginar a dos personas separadas por la distancia hablando por una línea telefónica.

Cuando la primera persona que llama decide que necesita cierta información, por ejemplo, los horarios de las salas de cine, debe ponerse en contacto con la parte que tiene esta información mediante un método estandarizado. El sistema telefónico es ese medio, y la tecnología que lo sustenta está estandarizada e implementada universalmente.

Las API funcionan de manera similar. Cuando un cliente necesita datos, se comunica a través de un estándar establecido según lo acordado por el host de datos. Esta modalidad de comunicación lleva la solicitud (y, por lo general, cualquier metadato asociado a la solicitud) al servidor, que recibe la solicitud y analiza la respuesta. Una vez que se forma la respuesta, se devuelve utilizando el modo de comunicación acordado.

La comunicación cliente-servidor tiene varias características esenciales. En primer lugar, es un paradigma de Solicitud-Respuesta. No se comparte información a menos que el usuario lo solicite específicamente. El intercambio es principalmente bidireccional y, aunque es tentador pensar en ello como una conversación, nuestro ejemplo anterior es un poco más sincrónico de lo que realmente está sucediendo. Estas solicitudes generalmente ocurren en un orden específico e incremental: solicitud, recibir la solicitud, analizar la respuesta, responder.



Figura 17: Cliente-Servidor

6.3.2. Publicador-Suscriptor

En comparación con la relación cliente-servidor, la relación Editor-Suscriptor es más como una delegación de responsabilidad. Mientras que la relación cliente-servidor es la presentación sincrónica de datos, el publicador-suscriptor es una intención expresa del usuario de una representación futura de esos datos que luego se les entrega en un conjunto de circunstancias.

Pensemos en nuestro ejemplo de la persona que llama en el cine que quiere saber cuándo se proyecta una película específica. Supongamos que el usuario no quiere saber acerca de una hora o fecha específica. En cambio, quieren que el teatro les envíe una lista de todas sus presentaciones diarias todos los días. Esta es la base de la relación editor-suscriptor.

En términos de API, en lugar de esperar a que un cliente solicite datos y que un servidor responda, el modelo publicador-suscriptor hace que el cliente indique que, dependiendo de ciertas circunstancias, les gustaría recibir los datos a medida que se modifican. Estas alteraciones pueden ser algo tan simple como “cada vez que cambien estos datos, actualízame”, pero también pueden ser algo más complejo, como “cuando esta variable incremental alcance un número superior a 100, avísame enviando una notificación automática”.

A diferencia de la comunicación cliente-servidor, editor-suscriptor no se centra en un paradigma de solicitud-respuesta inmediata. La información puede ser compartida incluso si no se solicita expresamente. El intercambio es unidireccional porque no se envía ninguna solicitud después de la suscripción, lo que establece la relación.

6.3.3. Paradigma Seleccionado

Para la implementación de este proyecto optamos por la aplicación del paradigma de publicador-suscriptor, pues queremos que los comandos que reciba el Raspberry sea secuencialmente de acuerdo a los últimos datos enviados al maestro.

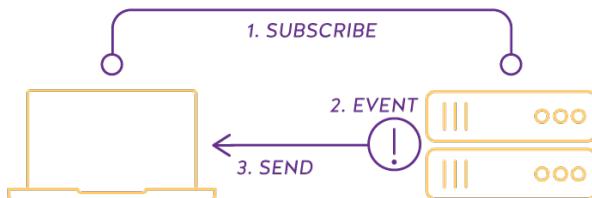


Figura 18: Publicador-Suscriptor

6.4. Sistemas de Navegación

La cátedra solicitó la implementación de varios sistemas de navegación. A continuación, se describirá detalladamente los sistemas implementados.

6.4.1. Navegación Aleatoria

Para la navegación aleatoria empleamos el concepto de Campo Potencial, de esta forma el Robot percibe los obstáculos y varia su dirección y velocidad en base a este concepto de navegación. Para el algoritmo los sensores ubicados en la parte frontal del robot miden la distancia con los objetos en su campo de visión, los tres sensores ultrasónicos generan tres fuerzas F1, F2 y F3 que son proyectados en un marco de referencia ortogonal xy del robot, además que generamos una fuerza F que siempre estira el robot hacia adelante. Se realizan los cálculos para determinar las fuerzas resultantes que están actuando sobre el robot. Por último las fuerzas resultantes determinan la proporción en que las ruedas varían su velocidad para así controlar los giros.

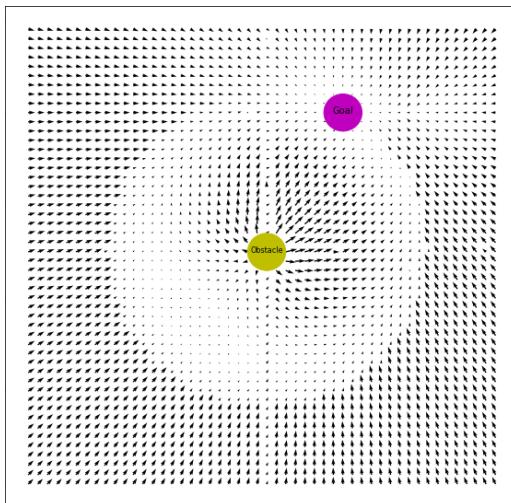
6.4.1.1 Campo Potencial

El campo potencial se puede utilizar para obtener una representación global del espacio para que la planificación general se pueda realizar a nivel global. Un campo de potencial continuo da una buena indicación de las distancias y las formas de los obstáculos para que los cambios necesarios en la posición y orientación del robot se puedan realizar de manera suave y continua. Al detectar colisiones, se evita la complejidad combinatoria de la detección de intersecciones realizada con representaciones geométricas.

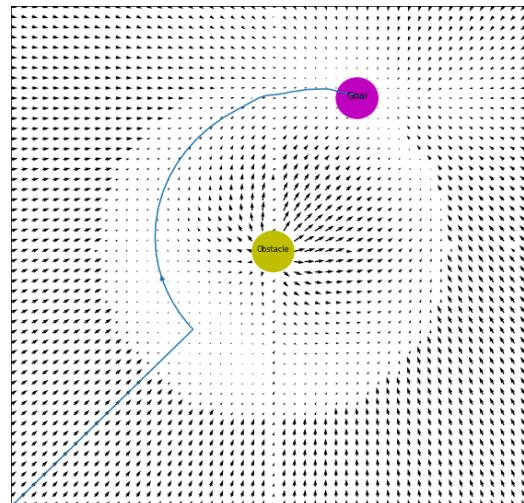
Esto se logra al eliminar la necesidad de realizar explícitamente la detección de intersecciones mediante el uso de un campo potencial que proporciona información sobre la distancia del objeto. Al usar una definición apropiada del potencial en un punto, se elimina la influencia de los obstáculos que no están en las proximidades del punto[3].

En robótica, podemos simular el mismo efecto creando un campo potencial artificial que atraerá al robot hacia la meta. Al diseñar un campo potencial adecuado, podemos hacer que el robot muestre comportamientos simples. Por ejemplo, supongamos que no hay ningún obstáculo en el entorno y que el robot debe buscar este objetivo. En la planificación convencional, se debe calcular la posición relativa del robot con respecto a la meta y luego aplicar las fuerzas adecuadas que conducirán al robot a la meta.

En el enfoque de campo potencial, creamos un campo atractivo que va dentro del objetivo. El campo potencial se define en todo el espacio libre y, en cada paso de tiempo, calculamos el campo potencial en la posición del robot y luego calculamos la fuerza inducida por este campo. Entonces, el robot debería moverse de acuerdo con esta fuerza[4].



(a) Potencial combinado cuando el obstáculo y el objetivo son diferentes



(b) Trayectoria definida por el robot para llegar al objetivo

Figura 19: Descripción de la implementación del campo potencial

6.4.2. Navegación por Barrido Paralelo

Para el barrido paralelo se realizo un programa que al detectar un obstáculo, realiza una rotación de 90 grados en un sentido por ejemplo el derecho, que realice un pequeño avance y luego vuelve a realizar un giro de 90 grados, con esto el robot queda paralelo a su trayectoria anterior. Luego realiza el avance hasta que el robot vuelva a detectar otro obstáculo, por lo que realiza un giro a la izquierda esta vez, un pequeño avance, otro giro a la izquierda y vuelve a retornar paralelo a su trayectoria. De esta manera se realiza el barrido alternando los giros a medida que se recorre el lugar

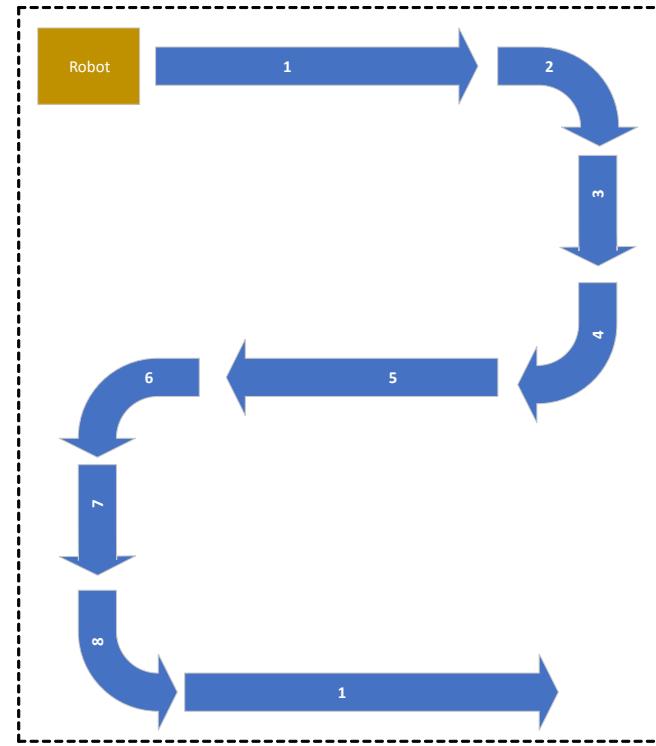


Figura 20: Funcionamiento de la navegación por barrido paralelo

6.5. Desafíos en la Implementación

Originalmente, planeamos utilizar un sistema de publicador/servidor como se aprecia en la **figura 21**. Todos los sensores se median en un nodo, los datos se procesaban y se mandaban en un tópico al maestro. El maestro procesaba los datos de los sensores, y mandaba las instrucciones de vuelta al Raspberry para el control de los motores.

Sin embargo, las configuraciones originales del Raspberry que controlaban los ultrasónicos y los encoders requieren que los pines del Raspberry se configuren con la librería *GPIO.BOARD* lo que pone en modo lectura/escritura analógica, en cambio para la IMU se requiere el envío y recepción de datos por I2C por lo que se debe implementar la librería *BUSIO*. Lamentablemente al dedicar demasiado tiempo para configurar la lectura de la IMU, no supimos solucionar la incompatibilidad de las librerías.

Para solucionar este problema, implementamos el sistema descrito en la **figura 22**. Un segundo Raspberry posee un nodo independiente encargado de recibir los datos del MPU para posteriormente enviarlos al nodo de los sensores. El nodo de sensores, aparte de medir los valores leídos por los ultrasónicos y encoders, también estaría suscrito al nodo del MPU para así obtener los datos de todos los sensores, y así mandarlos al maestro

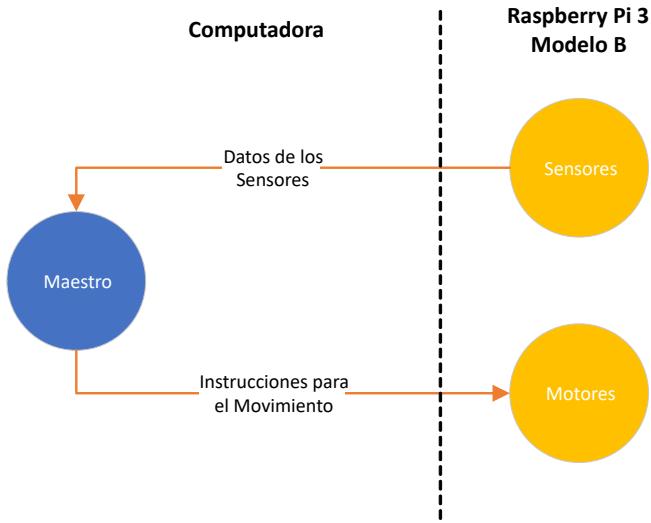


Figura 21: Metodología planificada para el control del proyecto

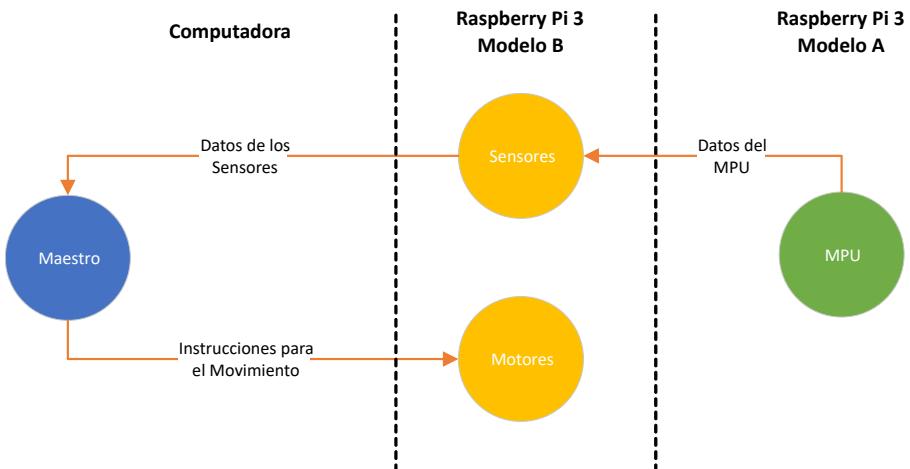


Figura 22: Metodología aplicada para el control del proyecto

7. Anexos

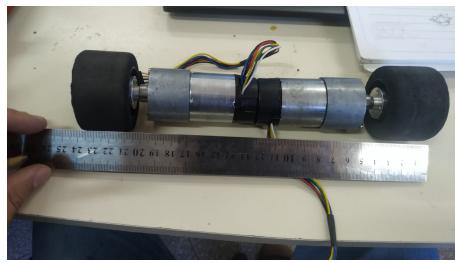


Figura 23: Medición de los motores



(a) Diseño de la base del chasis del robot



(b) Diseño de la rueda loca

Figura 24: Dibujos realizados en Fusion 360



Figura 25: Motores montados en el chasis diseñado

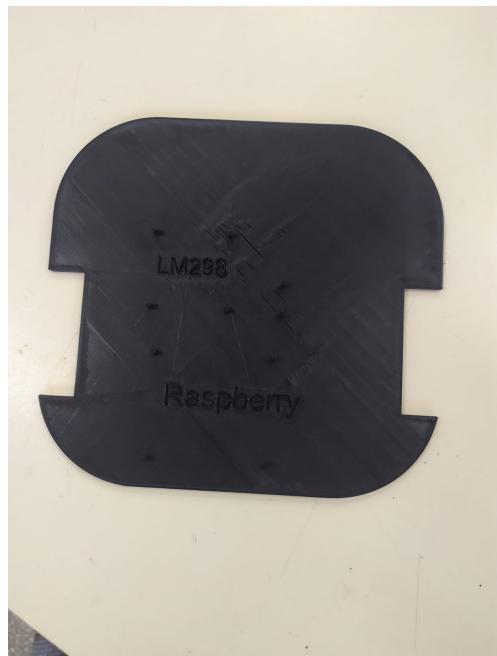


Figura 26: Tapa superior del robot



Figura 27: ubicación de los elementos de control dentro de la tapa superior



Figura 28: Ensamblado de la tapa del robot sobre el robot

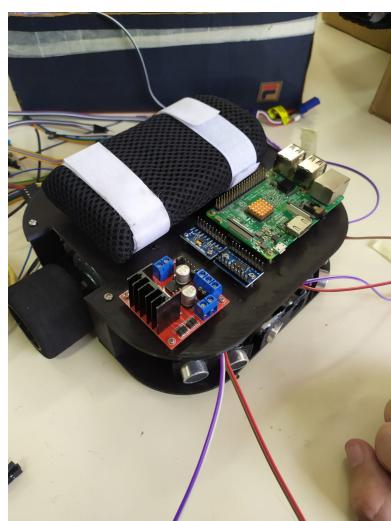


Figura 29: Fijado de la batería del Raspberry en el robot



Figura 30: Conexionados preliminares de prueba



Figura 31: Versión Final del robot

8. Conclusiones

Durante el desarrollo del proyecto se presentaron diversos obstáculos que fuimos sorteando a medida que avanzaba nuestro conocimiento. En lo que respecta al hardware y al armado no se presentaron muchas dificultades pues el conocimiento adquirido durante las cátedras de proyecto y de Robótica 1 nos brindaron una base sólida. El proceso de aprendizaje en lo que respecta al ROS, fue largo y estuvo lleno de altibajos pero al final fue gratificante obtener buenos resultados, que se lograron tras mucho ensayo y error. Queremos destacar que creemos en las potenciales aplicaciones de ROS para el control de varios robots en simultáneos, además que recomendariamos no emplear ultrasónicos pues el retraso en la lectura de esto provoca demoras en la ejecución del programa, por lo que los sensores ópticos brindarían mejores resultados.

Bibliografía

- [1] *Chapter 5. Swap Space.* URL: https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/ch-swapspace.html#:~:text=Swap%20space%20in%20Linux%20is,a%20replacement%20for%20more%20RAM..
- [2] *How to Install ROS Noetic on Raspberry Pi 4 - VarHowto.* URL: <https://varhowto.com/install-ros-noetic-raspberry-pi-4/>.
- [3] Yong Koo Hwang, Narendra Ahuja et al. «A potential field approach to path planning.» En: *IEEE transactions on robotics and automation* 8.1 (1992), págs. 23-32.
- [4] *Local Path Planning Using virtual Potential Field in Python — by ASLAN — Nerd For Tech — Medium.* URL: <https://medium.com/nerd-for-tech/local-path-planning-using-virtual-potential-field-in-python-ec0998f490af>.
- [5] *Módulo controlador de motores L298N — Tienda y Tutoriales Arduino.* URL: <https://www.prometec.net/l298n/>.
- [6] *Sensor Ultrasonido HC-SR04.* URL: <https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html>.
- [7] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System.* Ver. ROS Melodic Morenia. 23 de mayo de 2018. URL: <https://www.ros.org>.