Guía práctica para implementar un robot autónomo usando software libre

Martín Torres Galicia

COLEGIO DE CIENCIA Y TECNOLOGÍA

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO

2023

Introducción

La presente obra es un material educativo que tiene el propósito de guiar y encaminar a los estudiantes interesados en desarrollar un proyecto tecnológico enfocado a la robótica y las comunicaciones, haciendo uso de módulos de desarrollo. Para ello, es necesario contar con diversos conocimientos que se desarrollan y se obtienen cursando la licenciatura en Ingeniería Sistemas Electrónicos y de Telecomunicaciones del Colegio de Ciencia y tecnología, de la Universidad Autónoma de la Ciudad de México.

A partir de la realización de diversas prácticas y del uso de software libre, la obra busca fomentar en los estudiantes la creatividad para poner en práctica sus conocimientos técnicos y tecnológicos, e implementar un robot autónomo. La construcción de un robot es sólo un ejemplo de sistema autónomo, pero la obra da la posibilidad de imaginar sistemas con otros alcances.

Así mismo, el objetivo de este trabajo es reunir información, que se puede encontrar en la Internet y fuentes literarias, para poner en práctica lo necesario y adentrarse a proyectos de robótica de bajo costo e iniciar con proyectos de IoT (Internet de las cosas), entre otros temas. Se usa como tarjeta de desarrollo la tarjeta Raspberry y módulos de comunicación Xbee y ESP8266.

El documento se encuentra formado por pequeños tutoriales sobre instalación y configuración de software, módulos de comunicación y librerías. También se

encuentran ejemplos para la compresión de conceptos sobre ROS (Sistema Operativo de Robot, Robot Operating System), ejemplos de programación para hacer uso de los módulos de comunicación y, por último, se da un ejemplo para controlar un robot EV3 de la marca Lego por medio de scripts en lenguaje Python. Todo lo anterior se encuentra con ilustraciones, capturas de pantalla y diagramas para que se tenga un mayor entendimiento.

El contenido de esta obra hace uso de software y bibliotecas actualizadas al año 2023, con la programación, uso de bibliotecas y la configuración más reciente. Por eso es importante recalcar que después de un cierto tiempo es posible que las configuraciones, la programación, el uso de bibliotecas y de software puedan sufrir algunos pequeños cambios.

CONTENIDO

Índice de imágenes

Mapa de ruta de cada capítulo

Esta obra está compuesta de 11 secciones enfocadas en guiar al lector desde la instalación de las herramientas necesarias para crear un robot autónomo, hasta la configuración necesaria para que este mismo envié los datos obtenidos del robot a un servidor web. Todo esto se explica de manera explícita para que el lector no se pierda en el proceso. Es necesario tener conocimientos básicos de programación y uso de comandos de Linux para la configuración. En algunas partes del documento se mencionan pequeños consejos para solucionar o evitar cometer errores en la configuración o en la programación.

En la **Sección A** se explican los pasos a seguir para instalar el Sistema Operativo (SO) Debian en su versión Buster en la Raspberry Pi 4 modelo B. Así mismo, si no se cuenta con la tarjeta de desarrollo Raspberry, el SO Debian se puede instalar en una máquina virtual creada con la herramienta de software llamada VirtualBox.

En la **Sección B** se explica la solución a un posible error que puede encontrarse después de realizar la instalación del SO Debian en la Raspberry. El error consiste en la no visualización del escritorio del SO Debian, que solo ocurre cuando se hace uso del software VNC Viewer. Dicha herramienta ayuda a tener acceso remoto a la Raspberry sin necesidad de conectar a esta misma un teclado, mouse y un monitor.

En la **Sección C** se describe de manera detallada la instalación de ROS en su versión Noetic en la tarjeta de desarrollo Raspberry Pi 4 modelo B. Aquí se muestran los comandos necesarios para realizar la instalación, así como capturas de pantalla con los resultados que se van obteniendo al ejecutar dichos comandos. Este mismo proceso de instalación se puede realizar en la máquina virtual que previamente tiene cargado el SO Debian en su versión Buster.

En la **Sección D** se comprueba la instalación correcta y buen funcionamiento de ROS Noetic en la Raspberry o en la máquina virtual. Para comprobar todo lo anterior, aquí se desarrollan dos ejemplos básicos para comprender el uso de ROS. El primer ejemplo es crear un espacio de trabajo, crear un paquete del proyecto y crear dos nodos, los cuales se definen como publicador y suscriptor. El segundo ejemplo es descargar un paquete llamado turtlesim, el cual también ayuda a la comprensión de conceptos básicos de ROS.

En la **Sección E** se continua con la compresión básica de ROS, para este caso se ocupa una cámara web para hacer uso de la librería OpenCV. Aquí se crea el paquete que corresponde a este ejemplo, se crean dos nodos, publicador y suscriptor, que sirven para el envío y recepción de la información de la imagen que captura la cámara. Se explica de manera resumida la programación de cada nodo.

En la **Sección F** se explica la configuración para que exista la comunicación entre el Brick (cerebro del robot) del robot EV3 con la Raspberry Pi, en este caso la comunicación es cableada. Aquí también se explican las partes importantes del Brick del robot. Las configuraciones que se realizan son tanto en el Brick como en la Raspberry.

En la **Sección G** se corrobora que la configuración de la comunicación entre la Raspberry Pi y el robot EV3 haya sido correcta. Para comprobar el buen funcionamiento de la comunicación se realiza un ejemplo en el que se controla el robot desde la Raspberry Pi. Antes de eso es necesario instalar librerías, tanto en la Raspberry como en el Brick del robot. Después en la tarjeta de desarrollo se crea un script que está en lenguaje Python, que al ejecutarse dará la orden al robot de que mueva algún motor o realice una función específica.

En la **Sección H** se explica a detalle la configuración y uso de dos módulos Xbee Serie 1 Pro, para ello se hace uso de una herramienta de software llamada XCTU de la empresa Digi. Una vez que se configuran se realiza un ejemplo de comunicación entre los dos módulos Xbee, de los cuales uno se encuentra conectado en una computadora de escritorio y el otro se encuentra en una shield, la cual está conectada en la Raspberry Pi.

En la **Sección I** se realiza la configuración para el módulo NodeMCU, el cual tiene un SoC ESP8266. Aquí se muestra cómo realizar la instalación de bibliotecas y cómo programar al NodeMCU con ayuda del software Arduino IDE. Para corroborar que el módulo ha sido reconocido por el equipo de cómputo y que las bibliotecas se instalaron correctamente se realiza un pequeño ejemplo. Después se realiza la instalación de nuevas bibliotecas para que ahora el NodeMCU trabaje con ROS, igual se realiza un pequeño ejemplo para verificar su correcta instalación de bibliotecas y programación del NodeMCU.

En la **Sección J** se continúa trabajando con el NodeMCU. Una vez que se sabe cómo programar la tarjeta y la comunicación con ROS funciona de manera correcta, es momento de programar al NodeMCU como un nodo suscriptor. Para este caso se ocupa el nodo publicador del primer ejemplo de publicador y suscriptor que se vio en la **Sección D**. Se explica de manera resumida el código de programación que hace que el NodeMCU sea un nodo suscriptor y, al final, se corrobora su funcionamiento correcto.

Por último, en la **Sección K** se explica la programación requerida para enviar los datos almacenados en el NodeMCU a un servidor web, y además se combina la programación que se realizó en la **Sección J**, para que los datos obtenidos se guarden en la NodeMCU y después sean enviados el servidor web. Así mismo, se menciona un nuevo método para que el NodeMCU se conecte a una red

inalámbrica, no se explica a detalle el método, pero se proporciona la referencia para conocer más de este tema. Al final de esta sección se corrobora y se comprueba que se realicen las funciones que se desean.

Sección A

Instalación de SO Debian versión Buster

En la microSD para iniciar la Raspberry Pi

En esta sección se explica como cargar el SO a la tarjeta Raspberry Pi [1]. El software Raspberry Pi Imager, se descarga del sitio oficial de Raspberry Pi en el apartado de software[1]. Se debe de conectar la microSD a la computadora ya sea con su adaptador SD o con un adaptador USB. Una vez descargado e instalado el software en la computadora (en este caso con SO Linux en su distribución Ubuntu versión 20.04), buscamos en nuestro cajón de aplicaciones el programa que tiene el nombre **Imager** o simplemente se identifica por el logo de Raspberry. Al ejecutar el software se abre una ventana como la siguiente figura.

[1] https://www.raspberrypi.com/software/

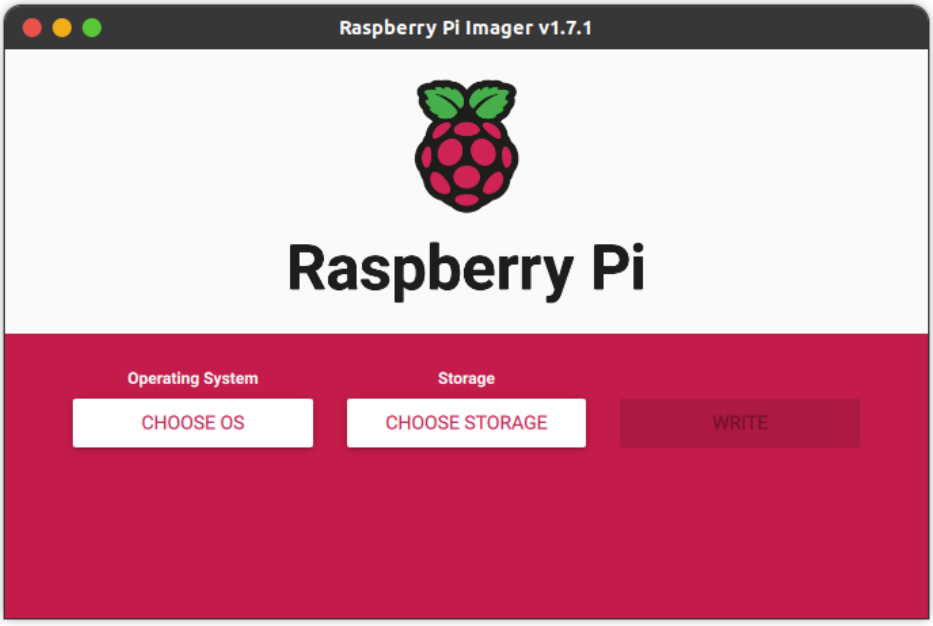


Figura 1. Ventana principal de Raspberry Pi Imager.

El primer paso es elegir el SO que instalaremos, así que damos clic en la opción **CHOOSE OS**, se mostrara una pestaña con una lista de opciones. Como primera opción se encuentra el SO más reciente para la Raspberry, que es la versión Bullseye. De acuerdo a las características que se requieren para instalar ROS Noetic en la Raspberry Pi, debemos tener el SO de versión Buster. Para este caso en particular se elige la opción de **Raspberry Pi OS (other)**, seguidamente buscamos y elegimos el SO que dice **Raspberry Pi OS (Legacy)** y en su descripción se nombra la versión Buster.

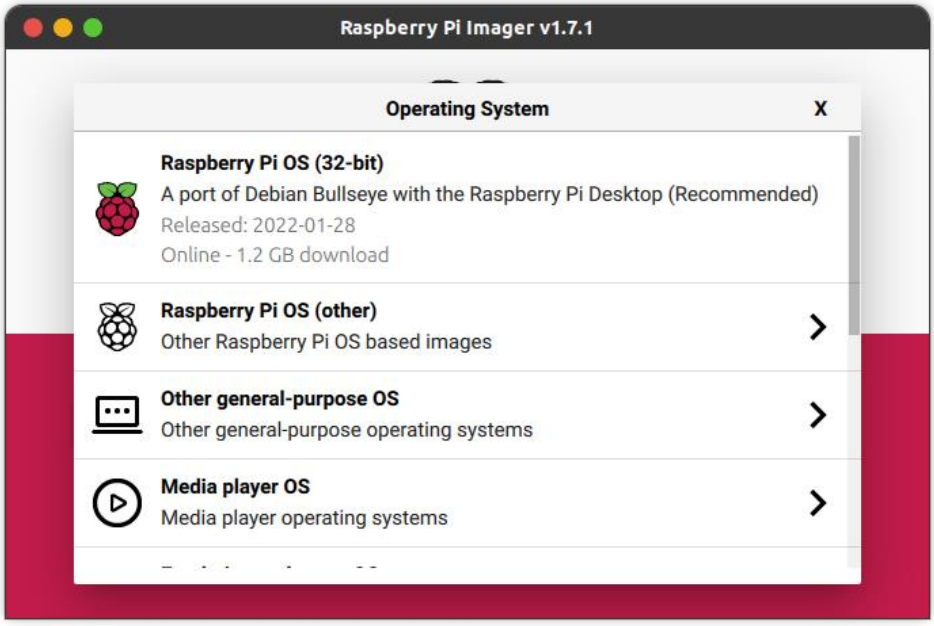


Figura 2. Lista de opciones para seleccionar sistema operativo.

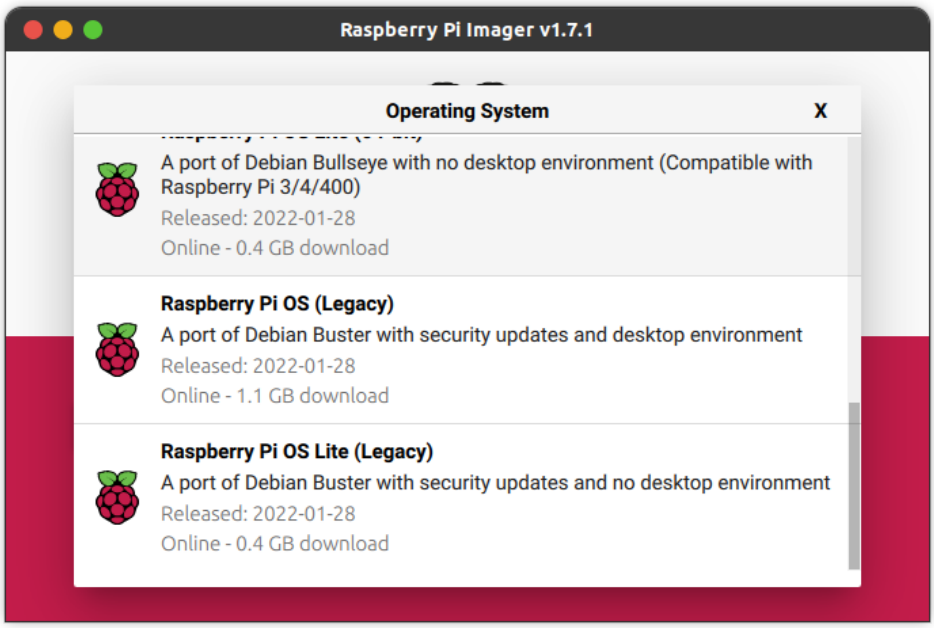


Figura 3. Elección de sistema operativo Raspberry Pi OS (Legacy) versión Buster.

Después elegimos el almacenamiento en donde se almacenará el SO, damos clic en **CHOOSE STORAGE**, buscamos y elegimos la microSD que insertamos previamente en la computadora. En seguida se da clic en la opción **WRITE** y a continuación se muestra una ventana en la que advierte que todo el contenido de la memoria microSD se borrara y pregunta si está seguro, se elige la opción **YES**, esperamos a que termine el proceso.

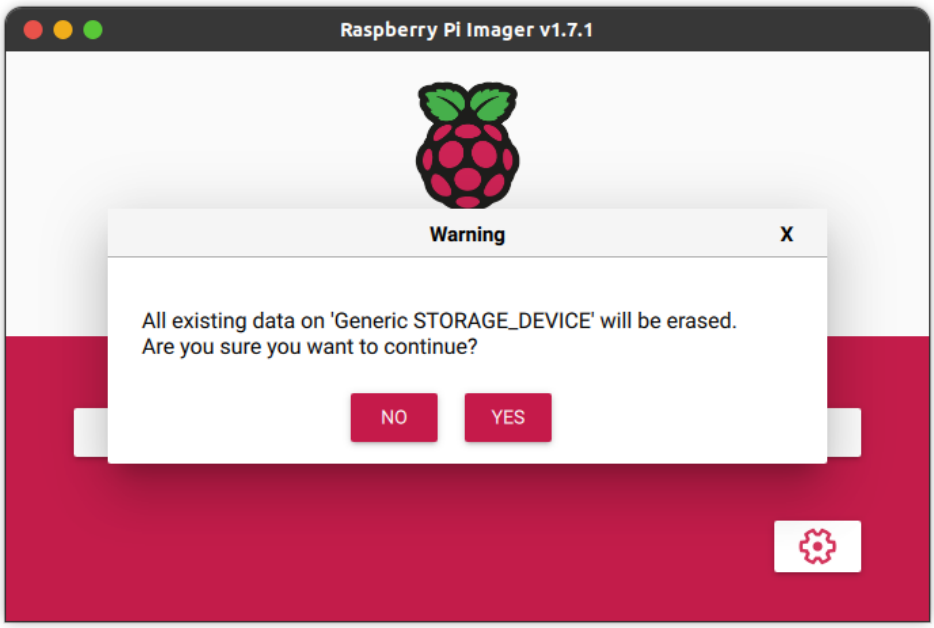


Figura 4. Advertencia de Raspberry Pi Imager, sobre el formateo de la microSD.

Una vez que finalice el proceso, se muestra una ventana en la que el SO ha sido escrito en la memoria microSD y por lo tanto se puede remover del equipo de cómputo.

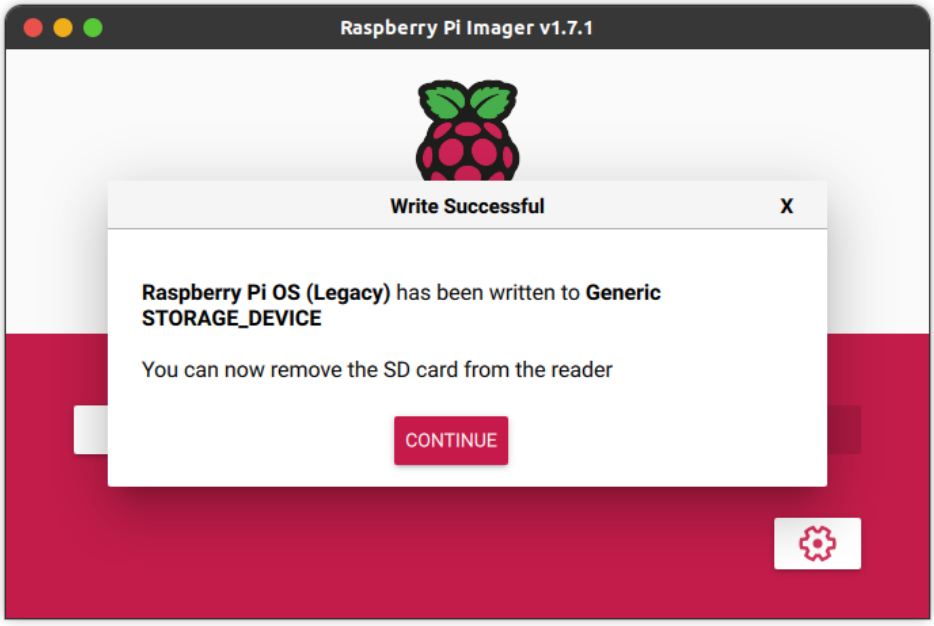


Figura 5. Se concluye la escritura del sistema operativo Raspberry Pi OS versión Buster en microSD.

Con lo anterior ya se puede insertar la memoria microSD a la ranura de la Raspberry Pi encender la misma y disfrutar del SO Debian en versión Buster. Cabe mencionar que antes de encenderla conectar teclado, mouse y un monitor a la Raspberry Pi.

Instalación de SO en una máquina virtual de VirtualBox para la PC

Esta sección se explica paso a paso la instalación de Raspberry Pi OS [2]. Para este caso en particular se instala Raspberry Pi OS en su versión Buster, esto se va a realizar en una máquina virtual creada con ayuda del software VirtualBox, está debe estar instalada en el equipo de cómputo que se va a ocupar. Este proceso se realiza con la finalidad de hacer pruebas de instalación de ROS Noetic sin contar con la tarjeta Raspberry Pi 4 Modelo B, ya que en un primer instante no se contaba con la tarjeta física Raspberry para realizar este proyecto.

Como se menciona anteriormente, el primer requisito es tener instalado en la computadora el software VirtualBox, una vez hecho eso se procede a descargar la imagen o ISO del SO desde la página oficial de Raspberry Pi, en este caso la imagen de Raspberry Pi OS versión Buster.

Se ingresa a la página web principal de Raspberry, se dirige a la barra superior de menú de la página y se busca la opción de **So**ftw**are**, se ingresa en esa sección y una vez que haya cargado se busca el apartado que dice **Raspberry Pi Desktop for PC and Mac**, se da clic en el botón **Download Raspberry Pi Desktop** y por último en el botón **Download**. Una vez descargado toca el turno de crear la máquina virtual en la computadora.

Como primer paso se ejecuta el software VirtualBox y una vez abierto en el menú de herramientas se elige la opción **Maquina**, seguidamente se despliega una lista de configuraciones, la cual se da clic en la opción **Nueva...** A continuación, se abre una ventana para dar inicio a la creación de la máquina virtual. Un ejemplo de ello es la siguiente figura 6.

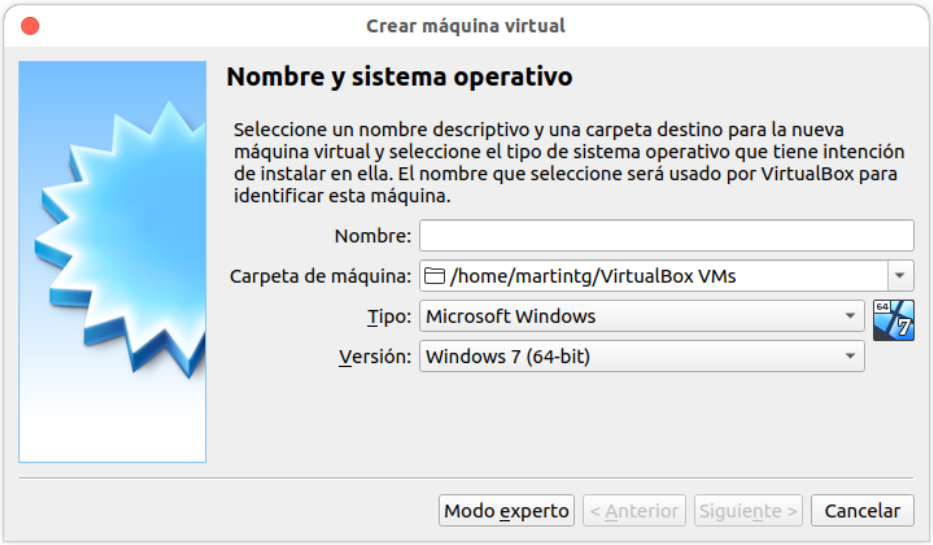


Figura 6. Ventana inicial para la configuración de una nueva máquina virtual en VirtualBox.

Observando la figura el primer dato importante es el nombre que se le da a la máquina virtual, en este caso se le nombra **Raspberry Pi 4**. La siguiente opción a modificar es donde dice **Tipo**, damos clic y elegimos la opción **Linux**. Para el tipo de versión se elige **Debian (32-bit)**, versión que es compatible con ROS. Se elige la arquitectura de 32 bit por las características que tiene la tarjeta de trabajo que es una Raspberry Pi 4 modelo B de 4 GB de memoria RAM.

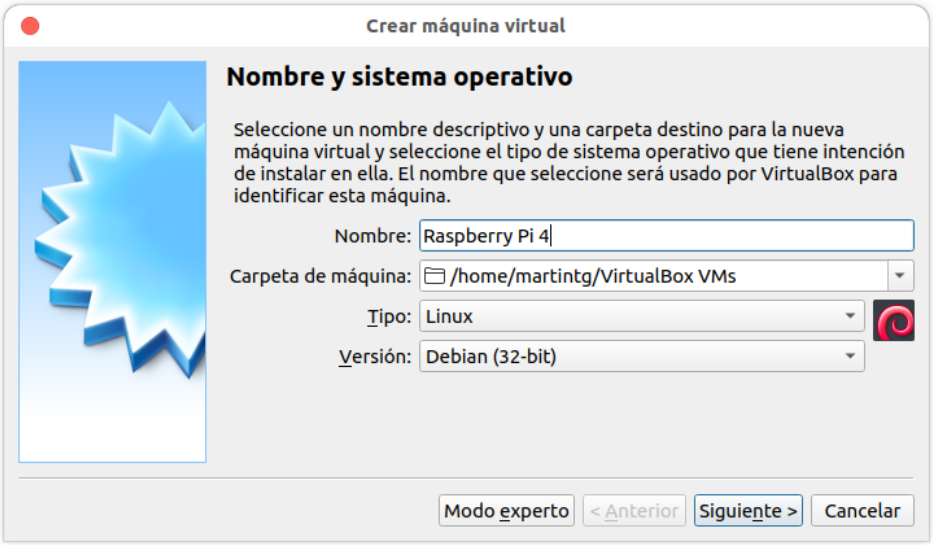


Figura 7. Nombrando máquina virtual, eligiendo sistema operativo y distribución.

La figura 7 muestra cómo debe de quedar configurado. Después se da clic en **Siguiente>**, ahora la ventana muestra la configuración sobre el tamaño de memoria que se le asignara a la máquina virtual, por default da un tamaño de 1 GB que en la parte derecha de la ventana está definido como **1024 MB**, para este caso pondremos un valor de **4096 MB**, para ello simplemente borramos el valor de 1024 y escribimos lo que deseamos, figura 8.

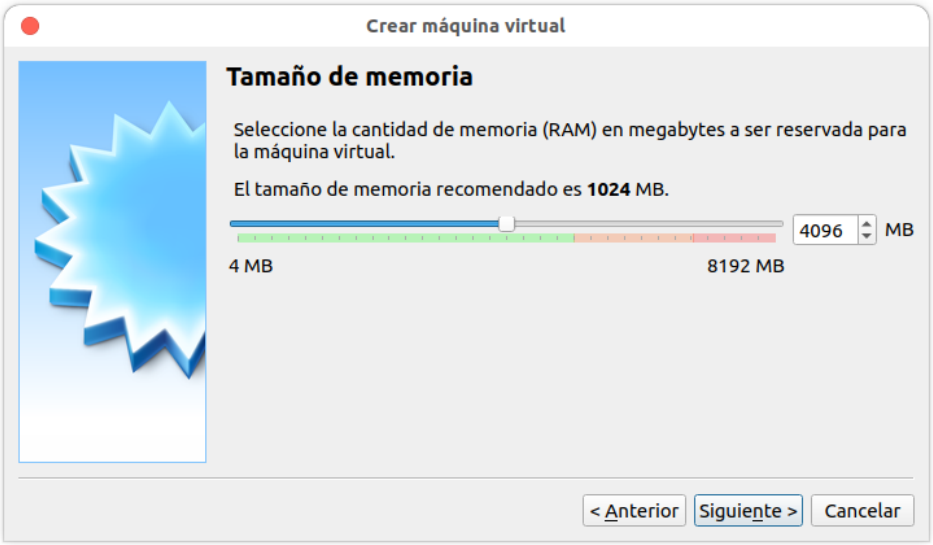


Figura 8. Definiendo tamaño de memoria (RAM) para la máquina virtual.

Clic en **Siguiente>**, en seguida aparece la configuración del disco duro y solo da en la opción **Crear**. La siguiente configuración es sobre el tipo de archivos del disco duro y se da clic en **Siguiente>**. De igual manera la siguiente configuración es del almacenamiento en unidad de disco duro física y se da en **Siguiente>**. Ahora viene la configuración sobre **la ubicación del archivo y tamaño del almacenamiento de nuestra máquina virtual**, este caso será de **32 GB**, para ello donde se encuentra el valor de 8.00 GB se va a remplazar por el valor de 32 GB, elegimos la opción de **Crear**, figura 9.

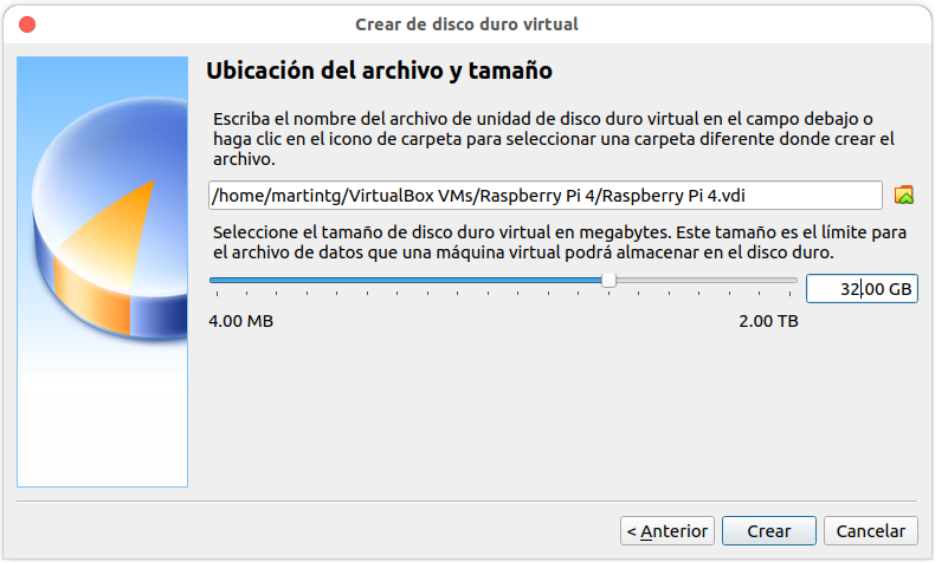


Figura 9. Escribiendo valor del tamaño del disco duro virtual.

Y listo se ha creado la máquina virtual que simulara la tarjeta de Raspberry Pi 4 modelo B, figura 10.

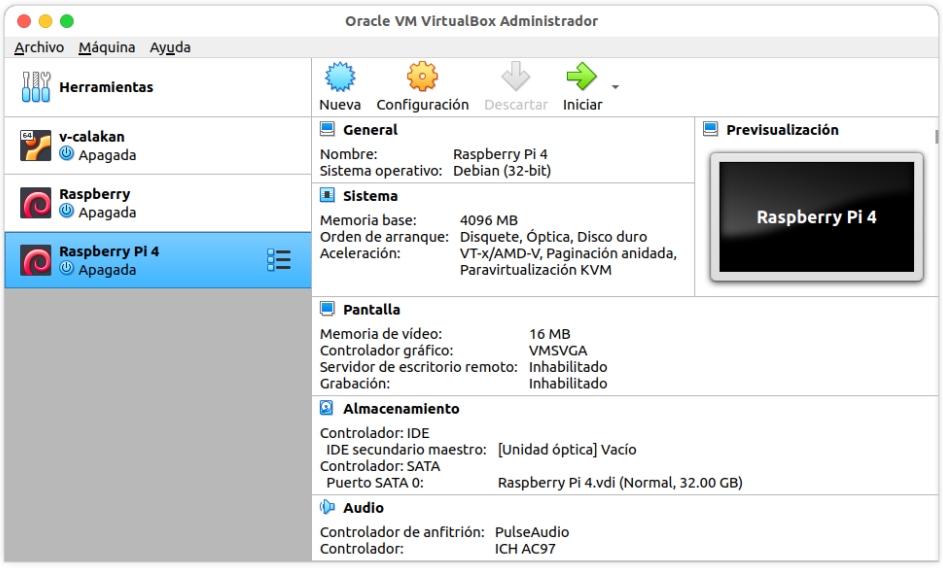


Figura 10. Detalles e información de máquina virtual creada con éxito.

Como siguiente paso es cargar la imagen ISO del sistema operativo que usa la Raspberry con versión Debian Buster. Para ello, seleccionamos la máquina virtual que se creó, **Raspberry Pi 4**, y en el panel derecho en la parte superior se elige la opción de **Configuración**. Se abre una ventana nueva. En el panel izquierdo se selecciona la opción de **Almacenamiento**, figura 11.

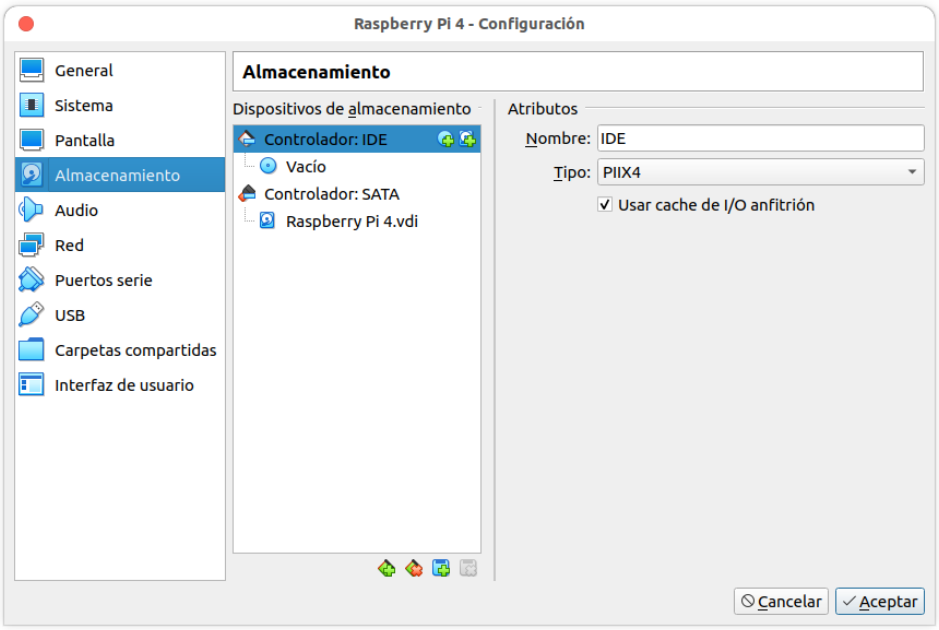


Figura 11. Ventana de configuración de almacenamiento de la máquina virtual.

En esta sección, se busca en el panel derecho la opción que dice **Controlador: IDE**, aquí se da clic en el icono “Añadir unidad óptica” que se muestra en la figura 12.

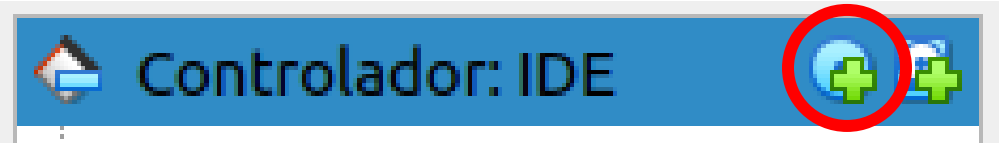


Figura 12. Opción encerrada en círculo rojo para agregar una unidad óptica a máquina virtual.

A continuación, se abre una ventana nueva de configuración donde se debe de agregar la imagen del SO a instalar, figura 13. Por lo tanto, se elige la opción de **Añadir**, se abre una ventana del explorador de archivos en el que se debe de buscar el archivo del SO que se descargó anteriormente, una vez ubicado, se da clic en **Abrir**.

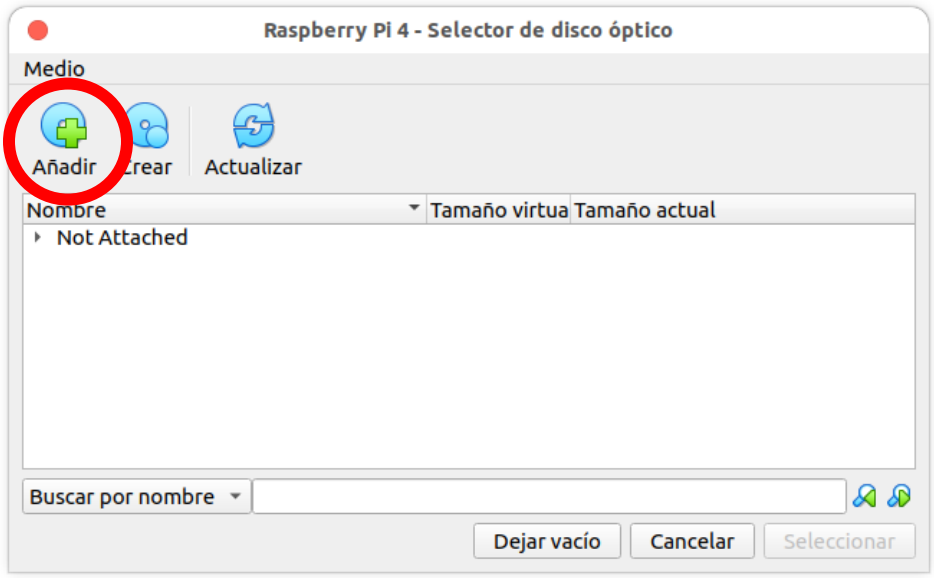


Figura 13. Ventana de configuración para elegir el archivo o imagen que se agregará a la unidad óptica.

Ahora se observa que ya se encuentra cargada la imagen del SO, figura 14, se da clic en **Seleccionar**, se cierra la ventana y ahora se puede apreciar que debajo de la opción de **Controlador: IDE** ya se encuentra el SO listo para ser instalado a la máquina virtual, figura 15. Por último, se da en **Aceptar**.

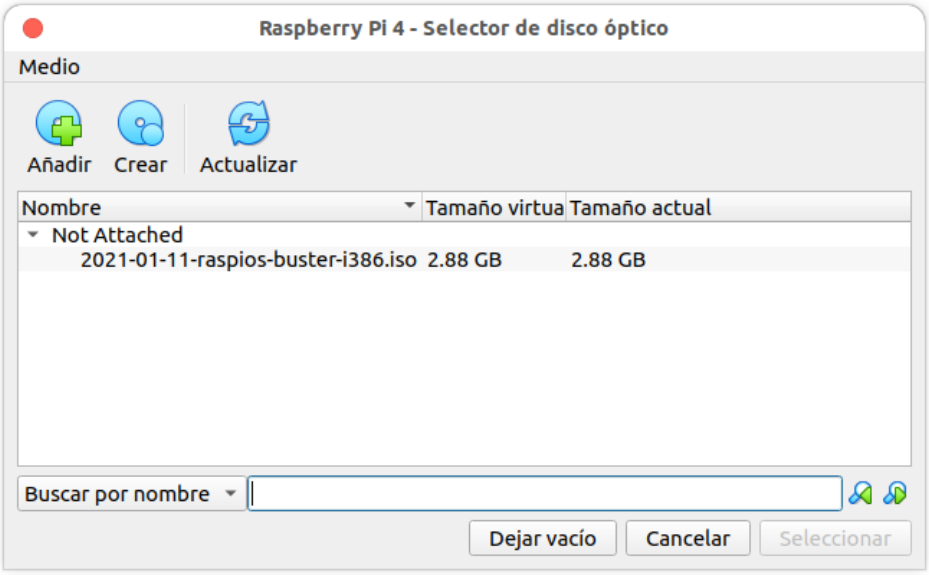


Figura 14. Seleccionando el SO RaspiOS-Buster para agregar a la unidad óptica de la máquina virtual.

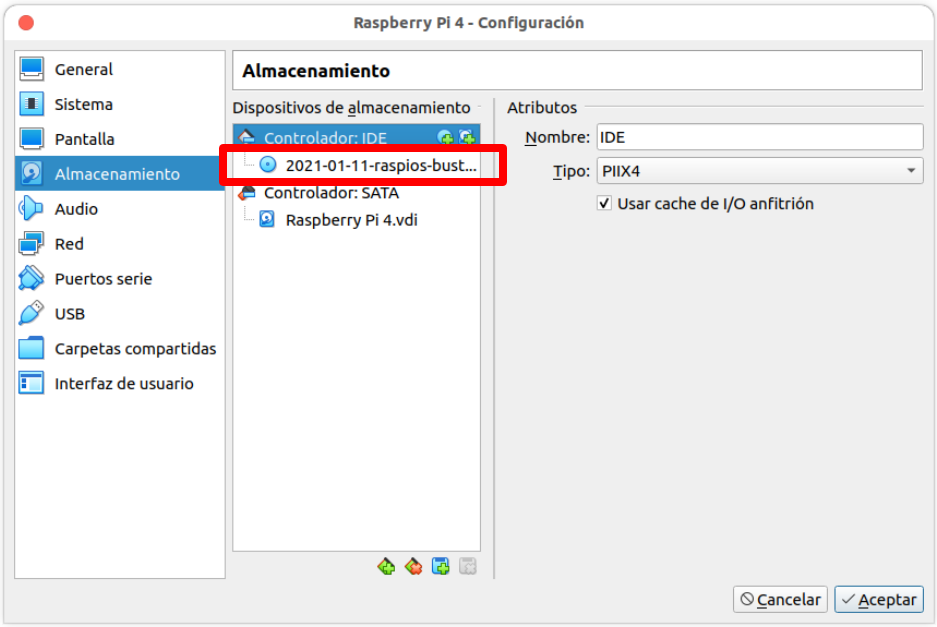


Figura 15. Imagen de SO agregado a la unidad óptica de la máquina virtual.

Aquí aún no se tiene instalado el SO, para comenzar con la instalación primero debemos de encender la máquina virtual que se creó. Así entonces, teniendo elegida la máquina **Raspberry Pi 4**, en el panel derecho, en la parte superior seleccionamos la opción de **Iniciar**. Tras iniciar la máquina virtual se abre una ventana que es la interfaz gráfica de la máquina que se creó, se espera a que cargue hasta que aparezca el menú de la siguiente figura, este menú te da 10 segundos para elegir una opción de lo contrario se ejecuta la primera opción, figura 16.

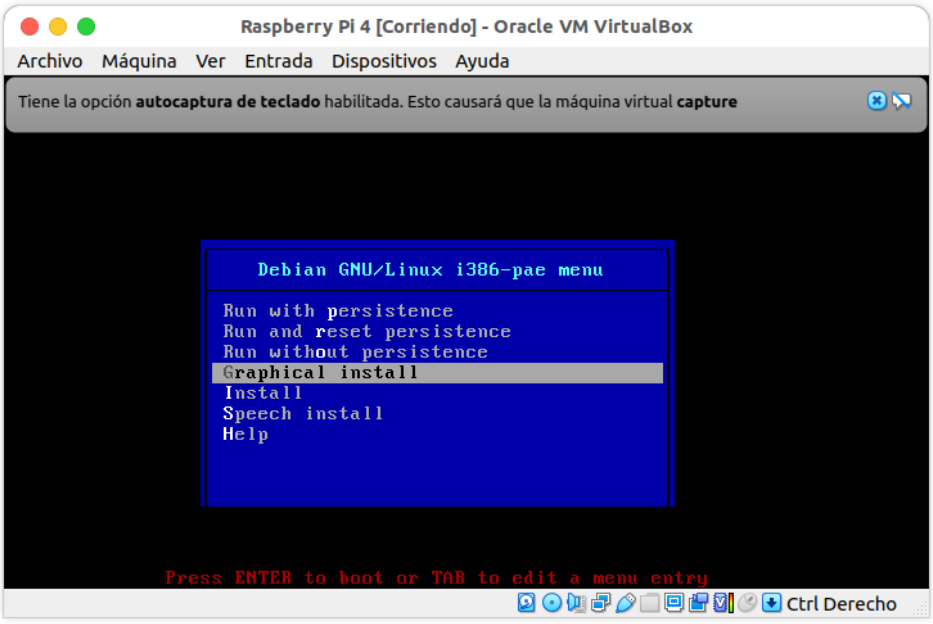


Figura 16. Menú de configuración para la instalación de SO Debian.

Se elige con ayuda de los cursores del teclado la opción de **Graphical install** y después se pulsa la tecla enter. A continuación, se muestra la configuración de idioma, en este caso se opta por el idioma **Spanish** y seleccionamos en **Continue**.

La siguiente ventana de configuración simplemente se marca la opción de **Guided – use entire disk** y se da clic en **Continue**, figura 17.

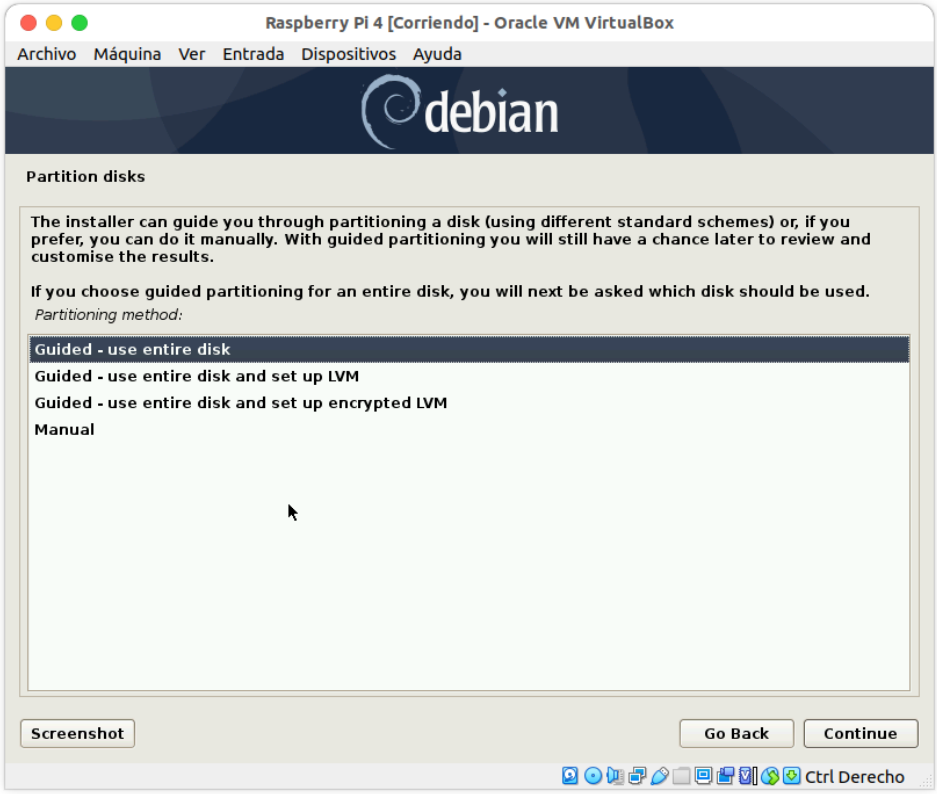


Figura 17. Configuración de la partición del disco duro.

La ventana de configuración que sigue es para seleccionar la partición del disco, no se mueve nada y solo se da clic en **Continue**, figura 18.

Figura 18. Seleccionando el disco a particionar.

El siguiente paso de la configuración es para determinar la partición de las carpetas **/home, /var y /temp**. En este caso simplemente se elige la primera opción en la cual todas las carpetas se van a encontrar en una sola partición, se selecciona **Continue**, figura 19.

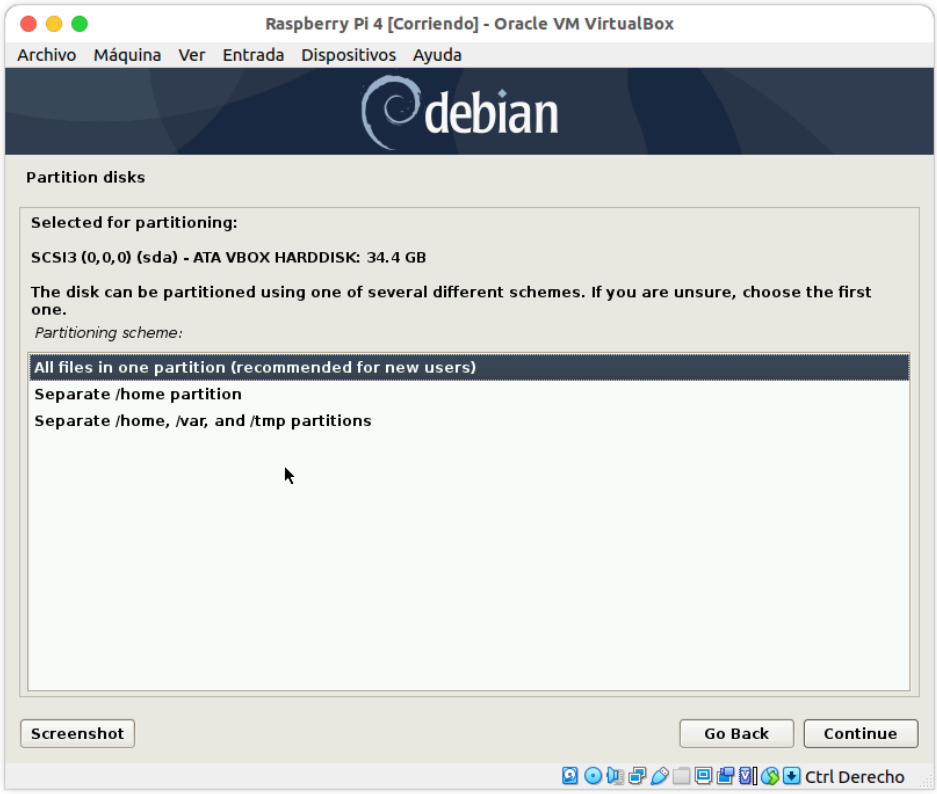


Figura 19. Seleccionando el esquema de partición.

A continuación, se muestra un resumen de las configuraciones que se realizaron, se da clic en **Continue**, figura 20.

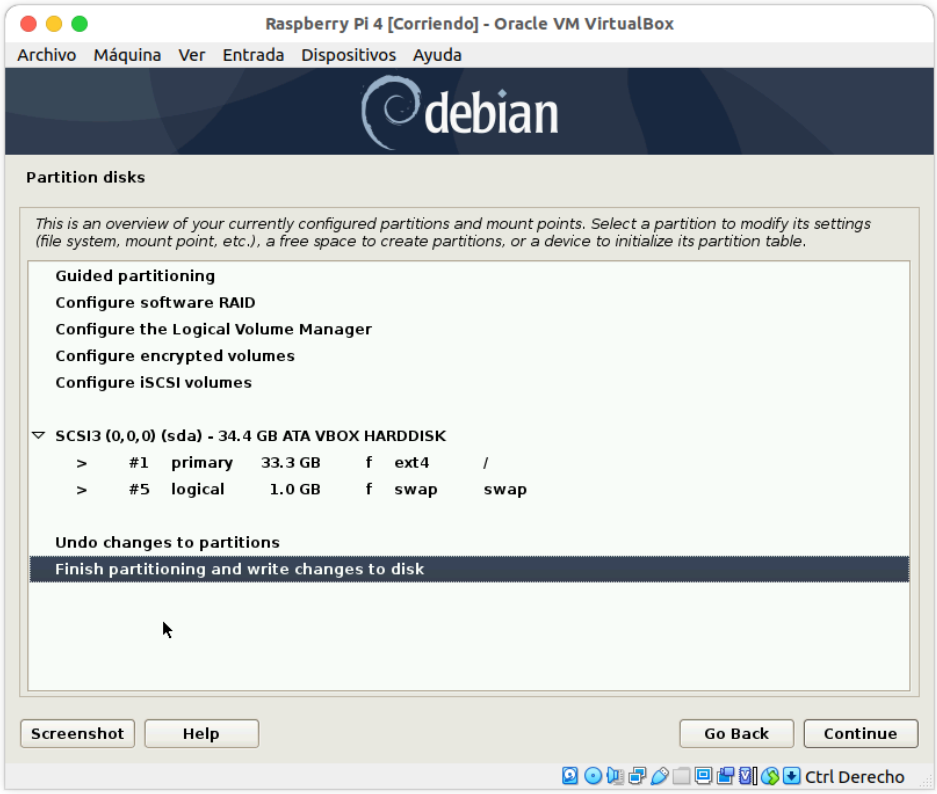


Figura 20. Ventana que muestra toda la información de la partición elegida.

La siguiente ventana, es para confirmar que se hagan los cambios en el disco duro, por lo tanto, se marca la casilla de **Yes** y se selecciona en **Continue**, figura 21.

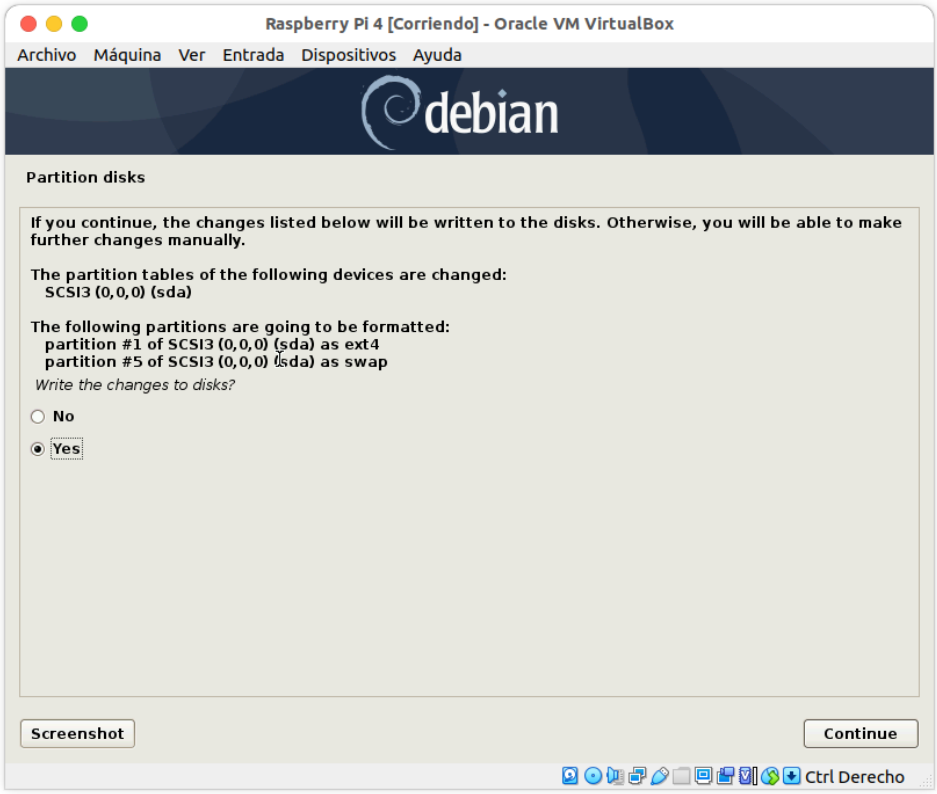


Figura 21. Confirmando las configuraciones de la partición.

El paso siguiente de la instalación pregunta si se desea instalar el Grub de arranque. Para este caso se marca la opción de **Yes** y seguidamente clic en **Continue**, figura 22.

Figura 22. Permitiendo la instalación del GRUB de arranque.

Inmediatamente se pide elegir la partición en la que se instalará el Grub, aquí se opta por la segunda opción que es la partición donde se instaló el SO, se da clic en **Continue**, figura 23.

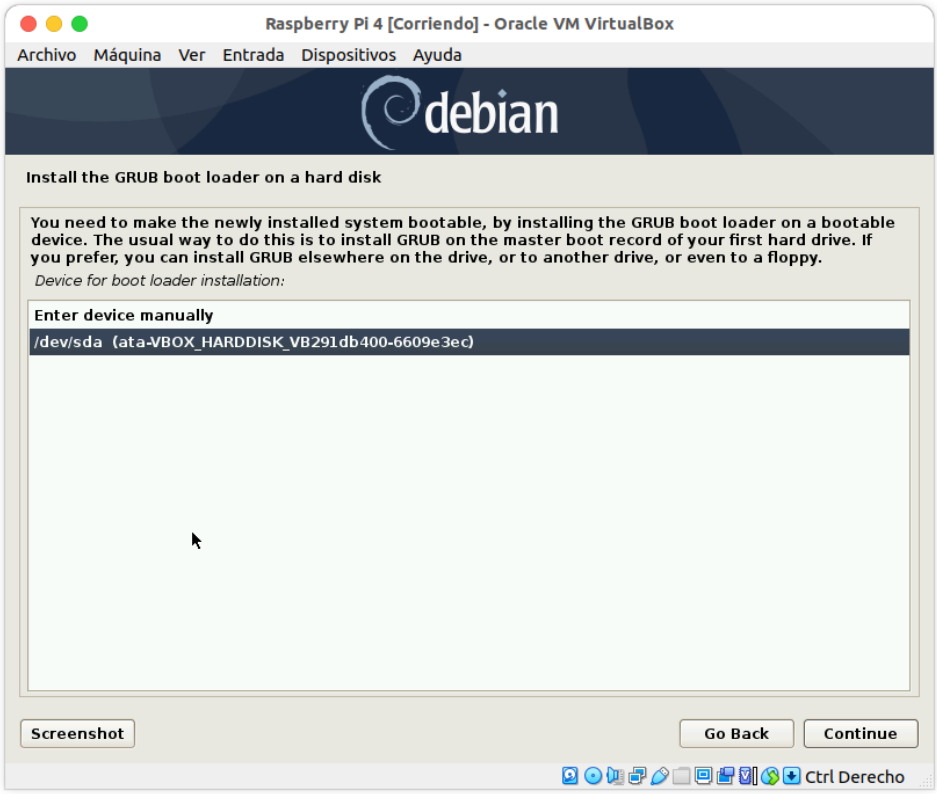


Figura 23. Seleccionado la ubicación donde se instalará el GRUB.

Tras unos minutos se mostrará el siguiente mensaje en el que afirma que la instalación ha finalizado, así mismo indica que se reiniciará el sistema. Se selecciona **Continue**, figura 24.

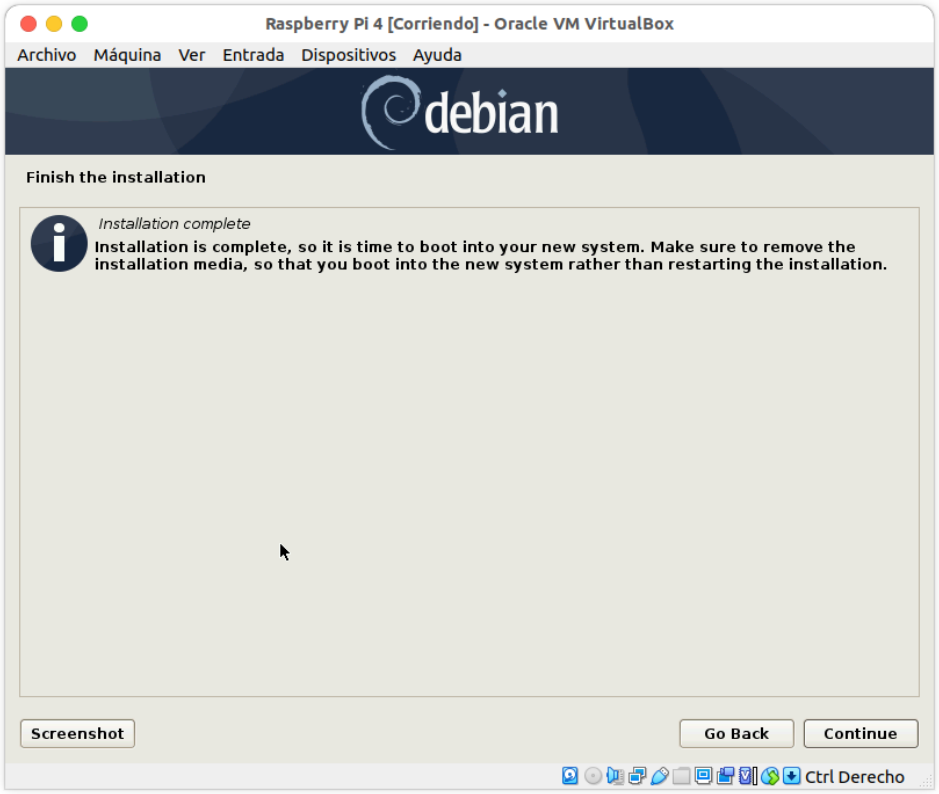


Figura 24. Ventana que confirma que la instalación finalizo.

Tras unos minutos, carga el escritorio del sistema operativo y sale una ventana dando la bienvenida a Raspberry Pi Desktop. Se presiona el botón de **Next**. Ahora se muestra la configuración para elegir el país, el idioma y la zona horaria. La configuración para este caso queda como la figura 25 y una vez terminado se da clic en **Next**.

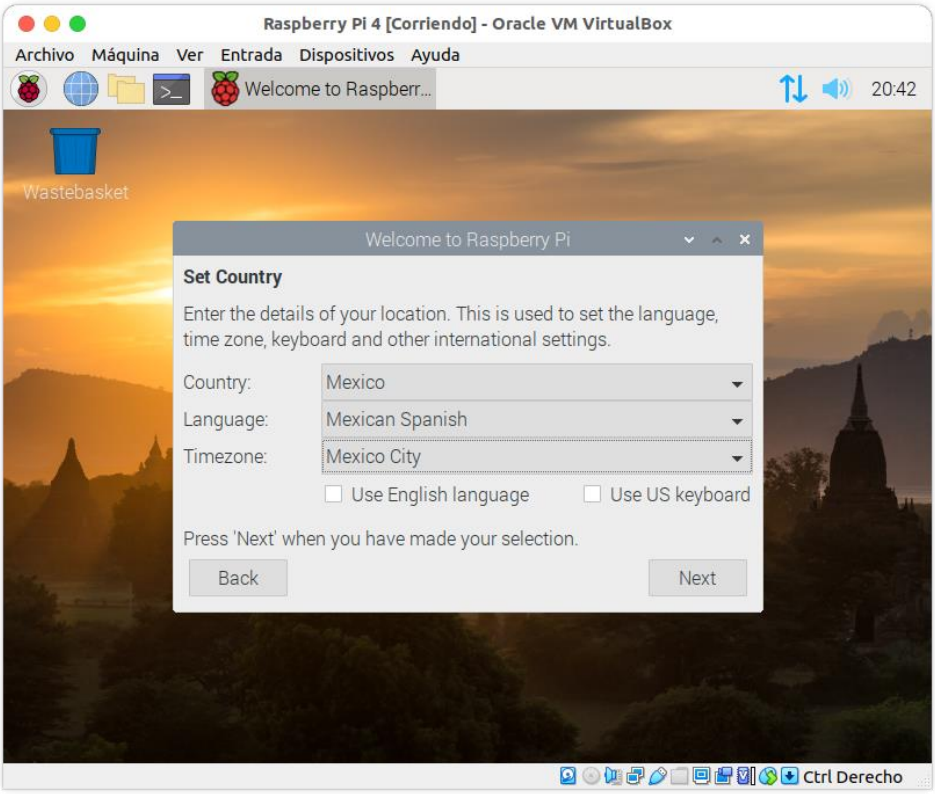


Figura 25. Captura de pantalla de la máquina virtual encendida con SO Debian y configurando el país, idioma y zona horaria.

Como siguiente configuración, se pide agregar una contraseña, eso ya queda a gusto del usuario, para este caso no se pondrá nada y solo se dará en la opción **Next**, figura 26.

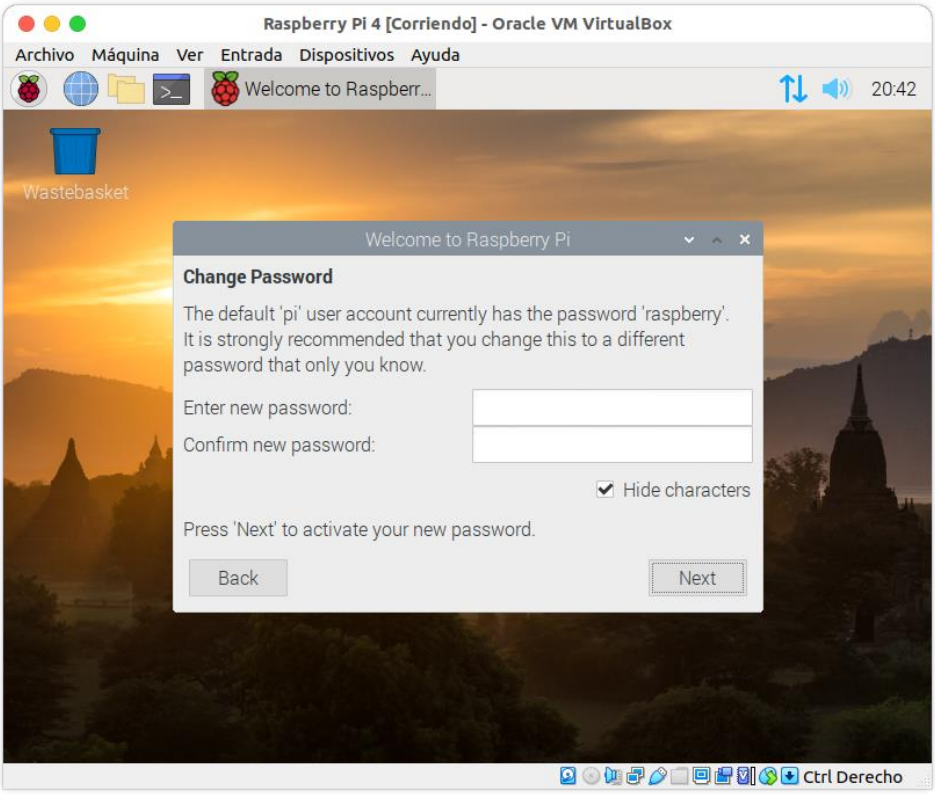


Figura 26. Configurando si se asigna una contraseña del modo super usuario.

Ya por último nos indica que buscará en internet si existen actualizaciones para el SO, por esta ocasión se omite esta opción y se selecciona en **Skip**, figura 27. Se omite este paso por que cabe la posibilidad que actualicé el sistema operativo a la versión **Bullseye** la cual no es compatible con ROS Noetic.

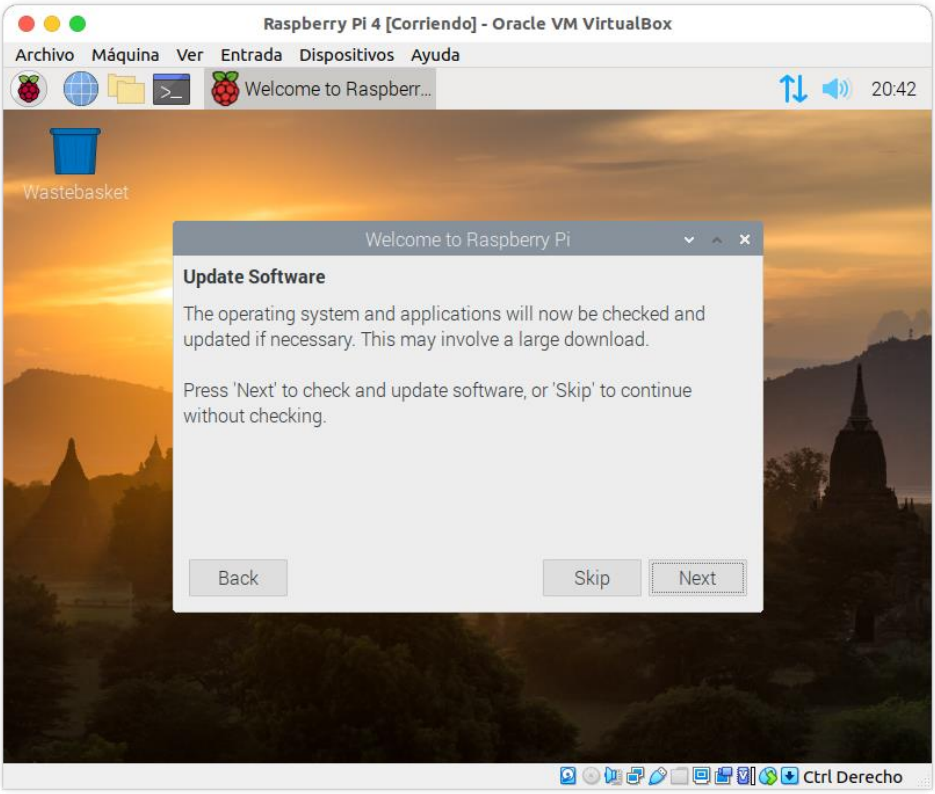


Figura 27. Ventana de configuración que ayuda a revisar si ay actualizaciones disponibles al SO.

Finalmente se tiene Raspberry Pi Desktop instalado en la máquina virtual como si se tuviera físicamente la tarjeta Raspberry Pi.

Sección B

Solución de error en Raspberry Pi OS Debian

Ya que se tiene conexión vía ssh de la Raspsberry Pi con la computadora y además conexión con el software VNC Viewer, es probable que no se aprecie el escritorio del SO instalado en la Raspberry y que solo se muestre algo parecido a la figura 28.

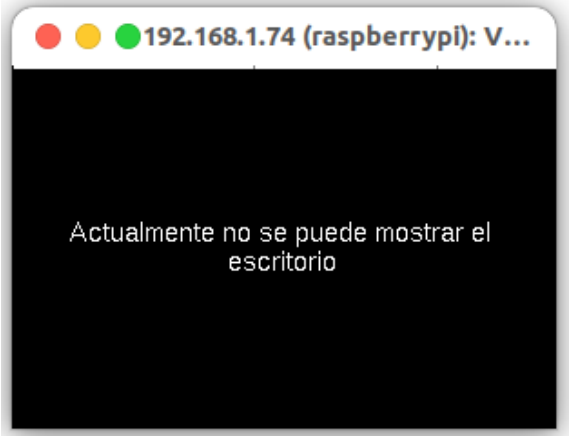


Figura 28. Ventana programa VNC Viewer no muestra escritorio de SO de la Raspberry Pi.

Para resolver este problema, simplemente se abre una la terminal y se ejecuta el código siguiente:

```
pi@raspberrypi:~$ sudo raspi-config
```

Después de ejecutar el comando se muestra una ventana que es la herramienta de configuración del software de Raspberry Pi, figura 29. Se elige la opción que dice **Display options**, seguidamente nos muestra una ventana con un listado de resoluciones, se selecciona la más alta resolución o simplemente un valor intermedio del listado, después damos en Aceptar, figura 30.

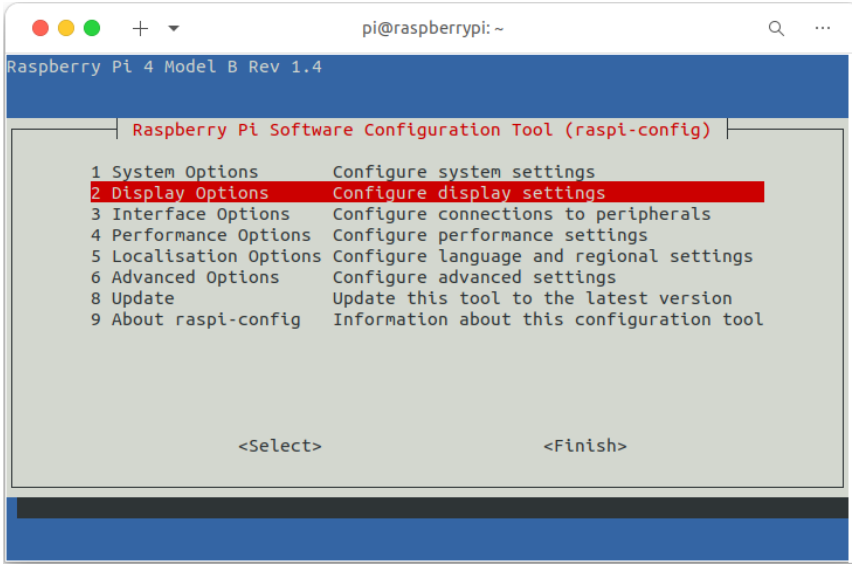


Figura 29. Menú de herramientas de configuración de Raspberry Pi.

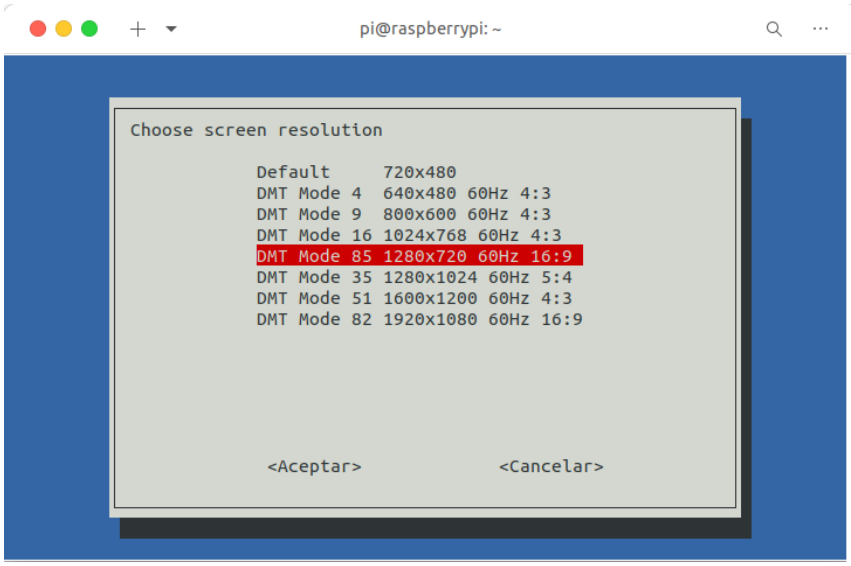


Figura 30. Lista de resoluciones disponibles.

A continuación, se reinicia la tarjeta Raspberry se desconecta del software y de la computadora vía ssh, por lo que nuevamente realizamos la conexiones pertinentes y listo ya se podrá observar el escritorio de la Raspberry [3].

Sección C

Instalación de ROS Noetic en Rasperry Pi OS

La presente sección se explica de manera detallada los pasos para instalar ROS en Raspberry Pi OS [4]. Como se mencionó anteriormente, se verifica el sistema operativo requerido para proceder con la instalación de ROS, para ello abrimos una terminal y escribimos el siguiente comando:

```
lsb_release –sc
```

Seguidamente como respuesta debemos obtener “`buster`” a continuación, una captura de pantalla.

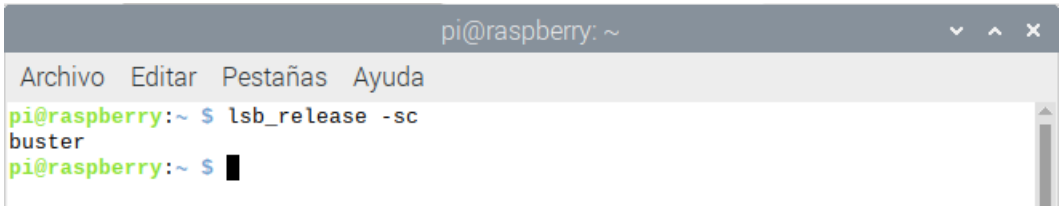


Figura 31. Ventana de terminal que tras ejecutar comando muestra la versión de SO en la Raspberry Pi.

Lo que se observa en la figura 31 confirma que es la distribución Debian en su versión Buster. Es importante comprobar este requisito para proceder con la instalación.

El siguiente paso es agregar los repositorios oficiales de ROS para las distribuciones de Debian. El repositorio: es la fuente donde se descarga el software o el SO en cualquiera de sus versiones, en este caso de ROS en su versión Noetic. Para ello agregamos la siguiente línea de comandos en la terminal:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu buster
  main" > /etc/apt/sources.list.d/ros-noetic.list' BAB17C6
```

Es importante mencionar que la mayor parte de las líneas de comando se ejecutan bajo los permisos de administrador, por ello es que se usa al principio `sudo`, este nos ayuda a ser super usuarios y realizar cambios en archivos protegidos del SO. Por otra parte, cuando se va a acceder como super usuario inmediatamente saldrá en mensaje en el que se solicita la contraseña del usuario para acceder. En este caso no se asignó una contraseña de usuario por lo que esta solicitud no aparecerá. Si se quiere corroborar que se ha agregado el repositorio, simplemente ejecutamos en terminal la siguiente instrucción:

```
cat /etc/apt/sources.list.d/ros-noetic.list
```

Como resultado el comando `cat` muestra el contenido del archivo, por lo que debemos de encontrar la siguiente línea:

```
deb http://packages.ros.org/ros/ubuntu buster main
```

El siguiente paso es agregar la clave oficial de ROS, esto es para asegurarnos que instalaremos los paquetes oficiales de ROS, cabe mencionar que esta clave no es específica para cada versión, es para todas las versiones de ROS existentes. Así entonces agregamos el siguiente comando a nuestra terminal y lo ejecutamos:

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

Una vez ejecutado, obtendremos el resultado que se muestra en la figura 32.

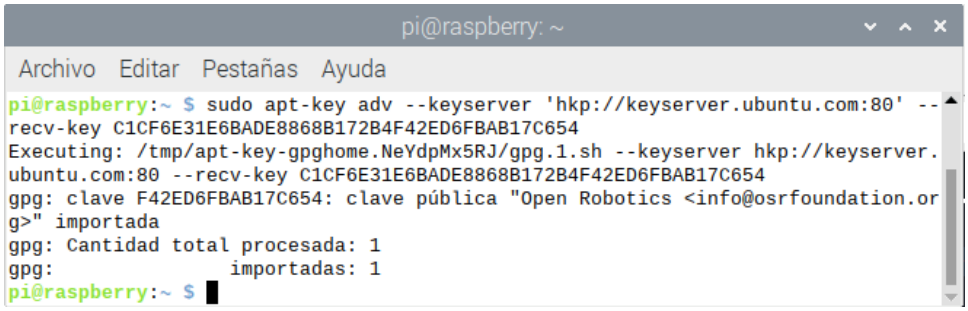


Figura 32. Resultado que obtiene tras ejecutar el comando mencionado.

A continuación, debemos de actualizar el índice de los repositorios de ROS Noetic que fueron agregados desde la fuente oficial de ROS. Para ello, ejecutamos el siguiente comando:

```
sudo apt update
```

Se selecciona la opción *Yes* para descargar los paquetes encontrados. En nuestra terminal obtendremos algo similar como la figura 33, en donde se indica el enlace http://packages.ros.org/ros/ubuntu buster, el cual está relacionado con los paquetes encontrados para descarga de ros.

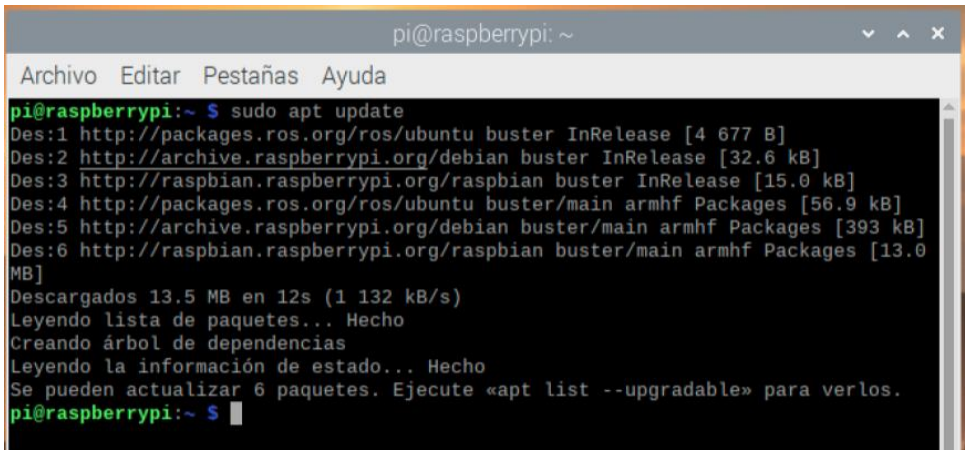


Figura 33. Resultado en venta de terminal tras ejecutar el comando sudo apt update.

Ahora instalaremos las dependencias de compilación, para ello ejecutamos la siguiente línea de comandos en nuestra terminal:

```
sudo apt-get install -y python3-rosdep python3-rosinstall-
generator python-wstool python3-rosinstall build-essential cmake
```

Cabe mencionar que para este caso estamos trabajando con la versión ROS Noetic, esta misma trabaja con Python 3. Por lo que, en el comando anterior, se tiene que modificar y agregar un 3 al final de cada palabra python, a excepción de `pyhton-wstool`. De otra manera se mostrará un error y no se instalarán los paquetes.

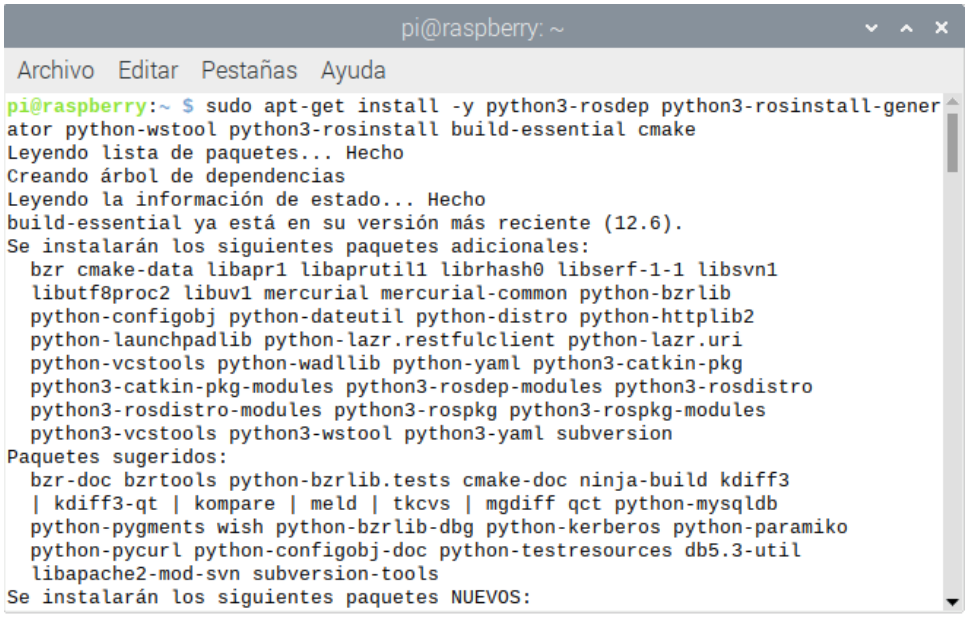


Figura 34. Resultado obtenido en terminal al ejecutar el comando mencionado.

Estas dependencias, contienen herramientas de ROS, por ejemplo: `rosdep`, `rosinstall_generator` y `ws_tool`, así como otras de compilación, como lo son: `make` y `cmake`.

Antes de instalar ROS Noetic, se deben de iniciar las dependencias de ROS, para ello, debemos de ejecutar la siguiente línea de comando:

```
sudo rosdep init
```

Esta instrucción inicia `rosdep`, la cual es una herramienta de ROS. Una vez ejecutado el comando, obtendremos en terminal el siguiente resultado que se muestra en la figura 35.

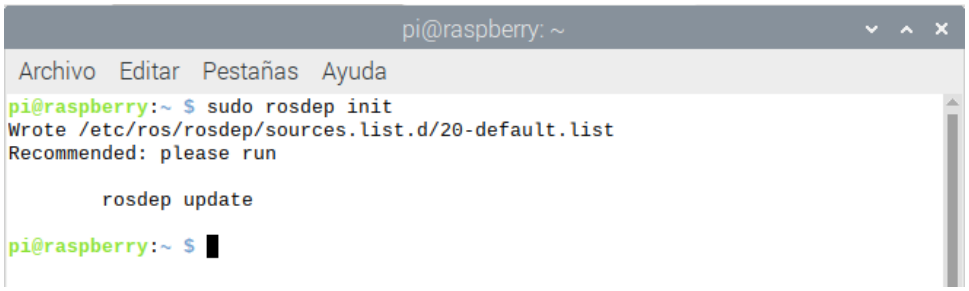


Figura 35. Resultado en terminal al ejecutar la herramienta rosdep.

Con el resultado anterior se recomienda ejecutar el siguiente comando en nuestra terminal:

```
rosdep update
```

Con esta instrucción actualizamos las dependencias de ROS, esto es para obtener información de los paquetes de los repositorios que acabamos de iniciar. Después de haber ejecutado el comando, obtendremos el siguiente resultado que se observa en la figura 36. En el despliegue de resultados se observa que la distribución Noetic se agregó, figura 36.

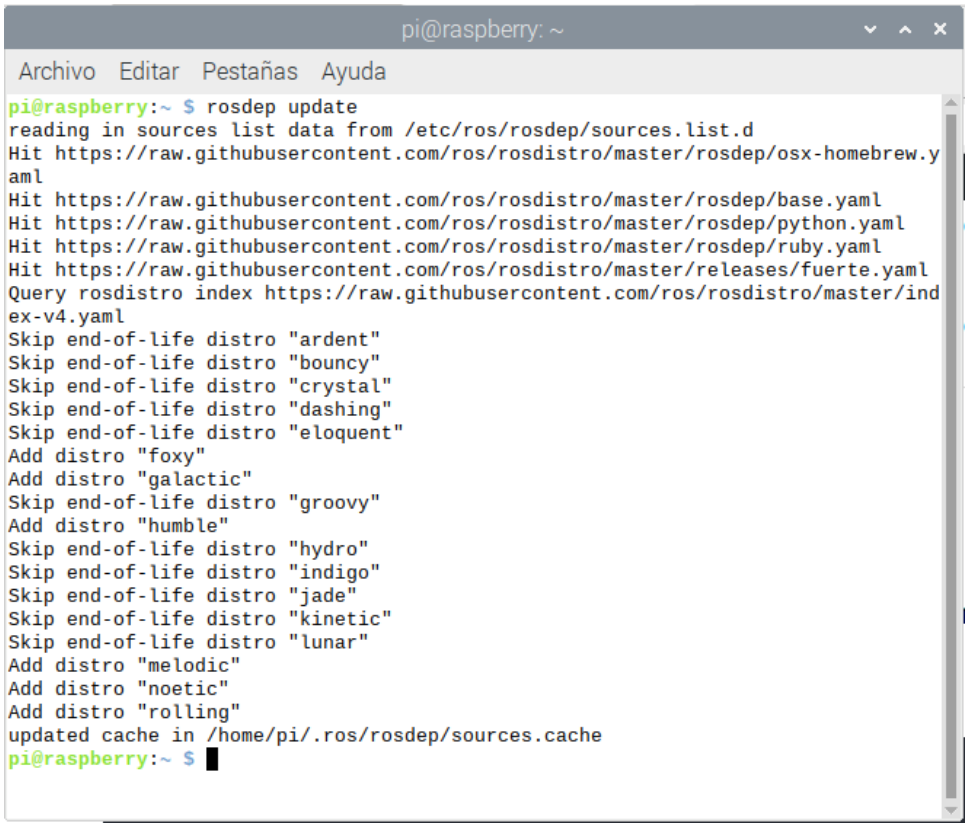


Figura 36. Despliegue de resultados en terminal al actualizar rosdep.

Otro requisito antes de instalar ROS es primero crear un directorio donde se descargarán los paquetes de ROS y a su vez se compilarán, este directorio se denomina espacio de trabajo, en este caso se nombrará `ros_catkin_ws`.

Para construir nuestro espacio de trabajo, ejecutamos las siguientes instrucciones:

```
mkdir –p ~/ros_catkin_ws
   cd ~/ros_catkin_ws
```

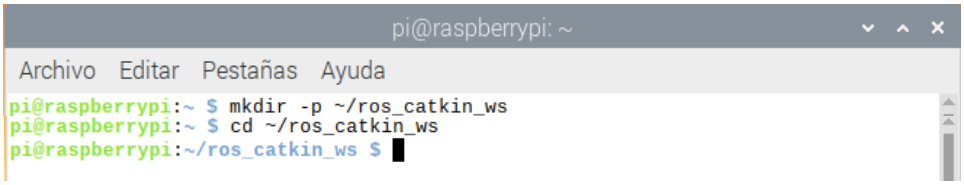


Figura 37. Ventana de terminal que muestra el resultado al ejecutar los comandos.

La primera instrucción crea una carpeta nombrada `ros_catkin_ws` ubicada en la carpeta personal (lugar donde se encuentran archivos del usuario, como: Documentos, Imágenes, Descargas, etc). La segunda instrucción nos permite dirigirnos al interior de una carpeta, en este caso `~/ros_catkin_ws`.

En seguida ejecutamos la siguiente línea de comandos [5]

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

En nuestra terminal no obtendremos ningún resultado, figura 38.

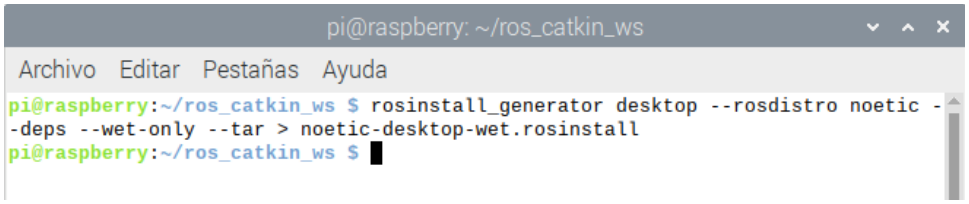


Figura 38. Ningún resultado en terminal al ejecutar comando antes mencionado.

Este comando consta de varias instrucciones a destacar. Por ejemplo, primero se encuentra el comando `rosinstall_generator`, este comando tiene la función de generar una lista de dependencias de Noetic en todas sus variantes, llámese:

desktop-full, **desktop** y **ros_comm.** Como se mencionó anteriormente, instalaremos la versión desktop.

Escribimos el siguiente comando en nuestra terminal y lo ejecutamos:

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

Para este paso, se requerirá de varios minutos para terminar la descarga e instalar los paquetes. Terminará cuando observemos en nuestra terminal el mensaje `update complete`, figura 39.

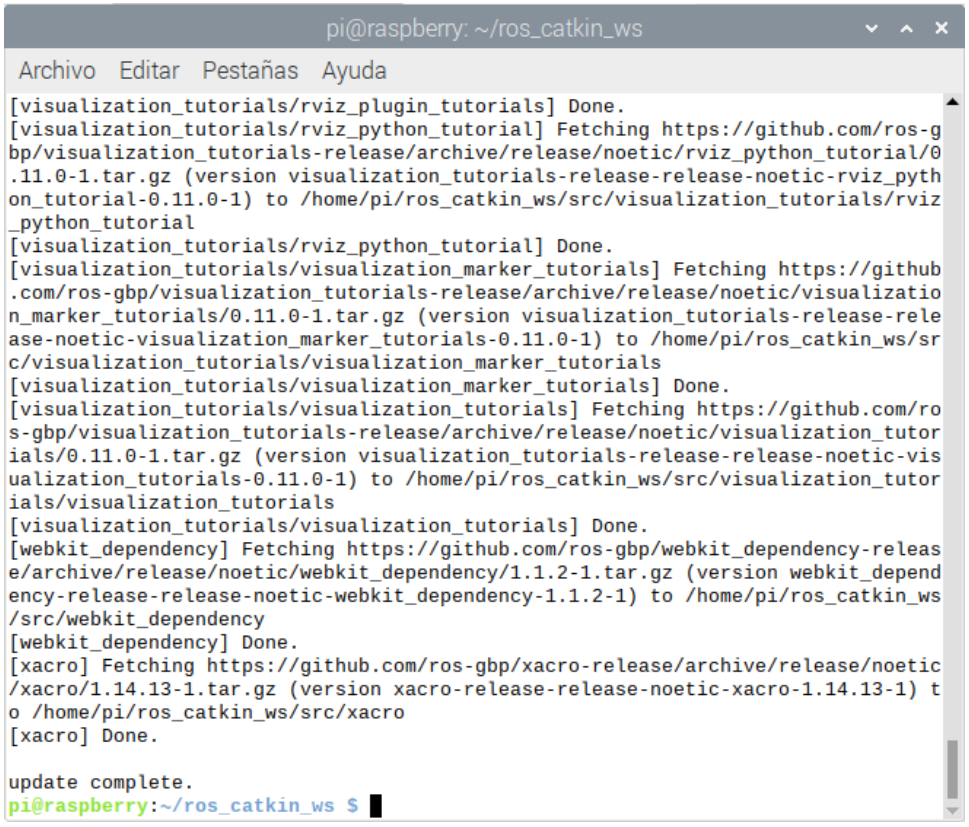


Figura 39. Resultado final mostrado en terminal al finalizar el proceso de descarga e instalación.

Antes de compilar todos los paquetes de la carpeta src, lugar donde se encuentran todos los paquetes de **ROS Desktop**, debemos de instalar todas las dependencias que se requieran. Para ello volvemos a utilizar la herramienta `rosdep` de ROS, así entonces ejecutamos el siguiente comando:

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

Tomará algunos minutos para que termine la instalación y al finalizar obtendremos el mensaje en terminal de `#All required rosdeps installed successfully`, observe la figura 40.

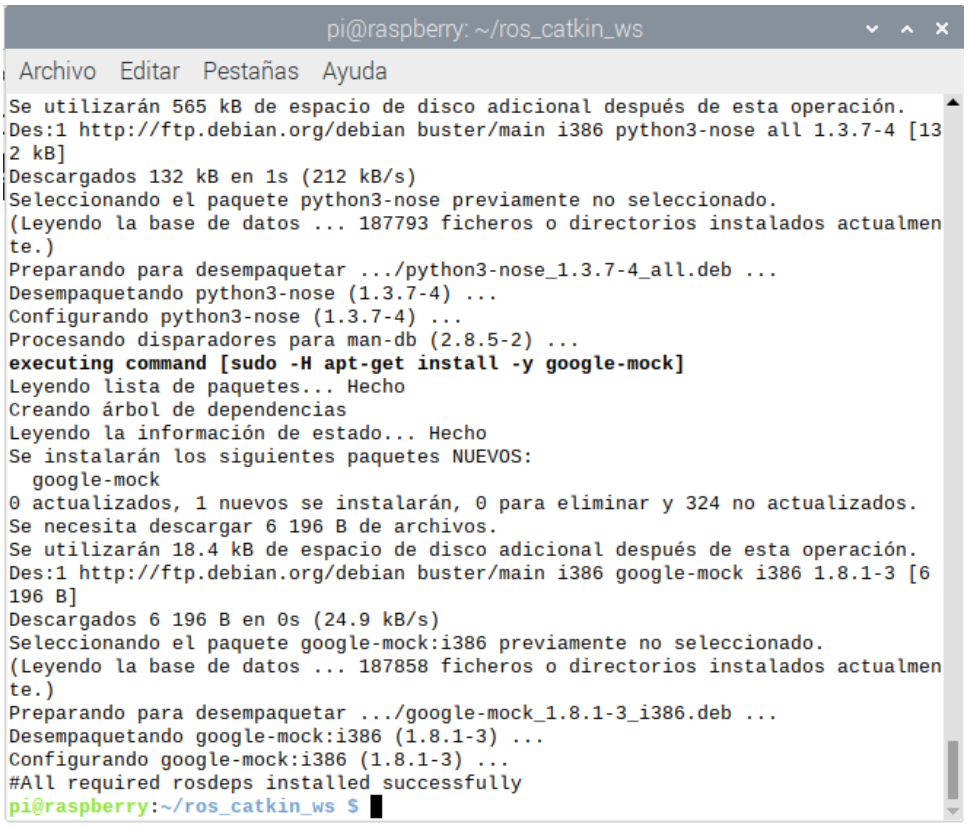


Figura 40. Resultado obtenido en terminal al terminar la instalación de las dependencias.

Como último paso debemos de compilar los paquetes de ROS Noetic. Pero antes de eso debemos de realizar una configuración para modificar la memoria swap o memoria de intercambio.

Como primer paso se desactiva la memoria swap, para eso ejecutamos en la terminal el comando bajo los permisos de super usuario escribiendo primero el comando sudo:

```
sudo dphys-swapfile swapoff
```

A continuación, se procede aumentar el tamaño de la memoria swap, para hacerlo se edita un archivo con el comando que sigue:

```
sudoedit /etc/dphys-swapfile
```

En terminal se abrirá el archivo a editar, seguidamente ubicaremos la línea que dice `CONF_SWAPSIZE`, se observa que esa línea tiene asignado un valor de `100`, esto significa que tiene 100 MB de memoria swap, este valor se modifica con el valor de 1 GB (1024 MB). Con los cursores del teclado nos dirigimos a esa línea y ponemos el valor de 1024. Para salir del editor, simplemente se presionan las teclas **Crtl** y **X** al mismo tiempo, enseguida en la parte de abajo de la terminal pregunta si deseamos guardar, para confirmar que sí, solo se presiona la tecla **s**, después saldrá un mensaje del nombre del archivo a escribir, solo presionamos la tecla enter y se finaliza la edición del archivo. Como ejemplo la figura 41 muestra el antes y la figura 42 muestra el después de los valores de swapfile asignados.

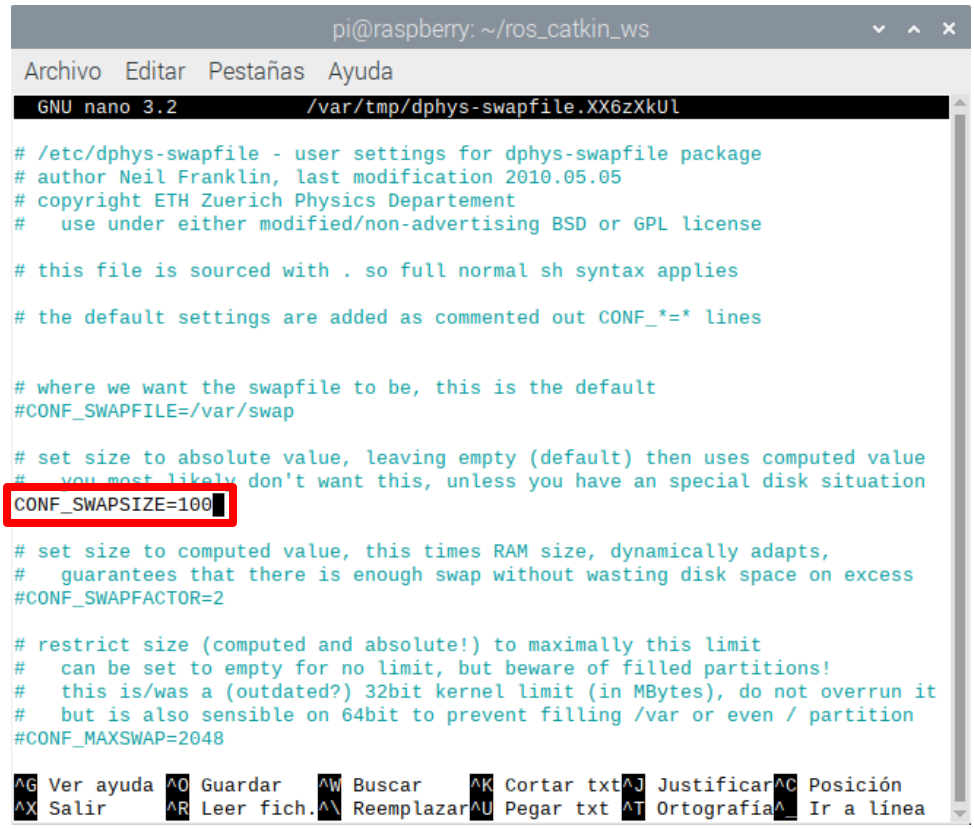


Figura 41. Contenido del archivo dphys-swapfile.

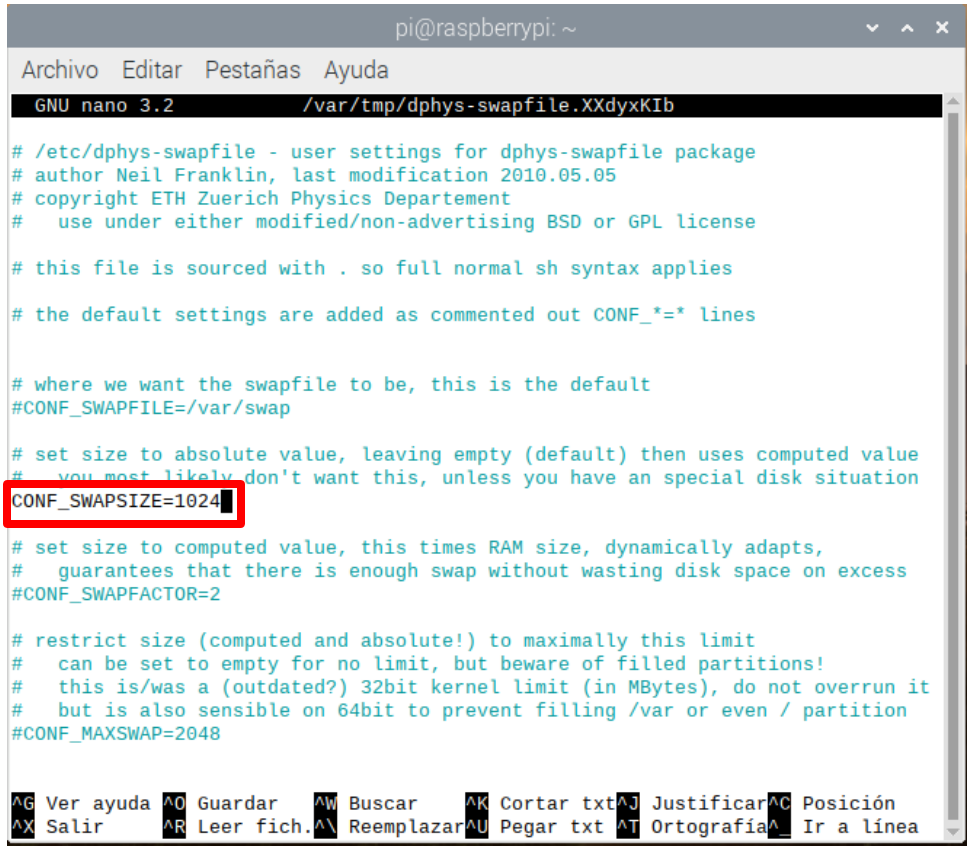


Figura 42. Modificación del tamaño de la memoria a un valor de 1024 en el archivo.

Para que se realicen las configuraciones que hicimos en el archivo se ejecuta el comando:

```
sudo dphys-swapfile setup
```

Como resultado, obtendremos lo que se muestra en la figura 43, el cual nos indica que se quiere un valor de 1024 en swap, revisa el valor existente y lo elimina y después se genera un swapfile de 1024 MB.

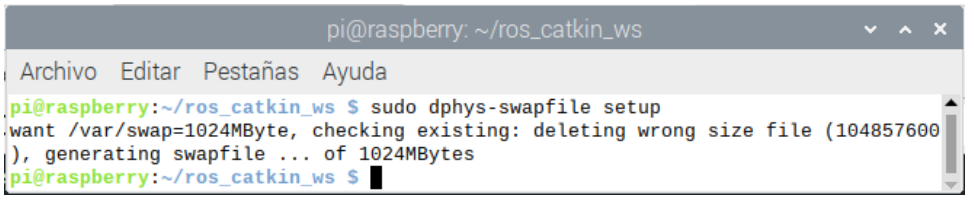


Figura 43. Resultado en terminal tras ejecutar comando para verificar que se modificó el tamaño de memoria.

Finalmente, ejecutamos el comando que sigue para activar nuevamente el swap:

```
sudo dphys-swapfile swapon
```

Para corroborar que lo que editamos fue correcto, ejecutamos el comando en terminal:

```
free -m
```

Y como resultado veremos lo que se ve en la figura 44. La línea que corresponde swap tiene el valor de 1023, lo que indica que la configuración es correcta.

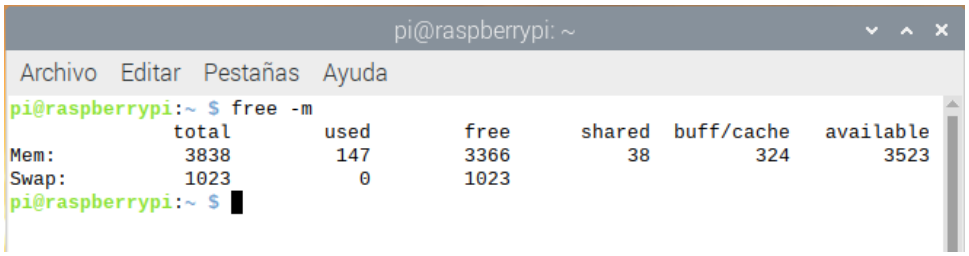


Figura 44. Ejecutando comando free -m para ver tamaño de memoria swap.

Se aclara que la anterior figura se obtendrá los mismos resultados solo cuando se está trabajando con la tarjeta física Raspberry Pi ya que, si se está trabajando en una máquina virtual con el SO Debian en su versión Buster, el valor de la memoria swap será de **1998**. Se realizaron las pruebas pertinentes y hasta el momento este

valor en la memoria swap no afecta el funcionamiento de ROS, por lo tanto, se puede proseguir con la instalación de ROS Noetic.

Por último, se procede a compilar todos los paquetes de Noetic, para ello primero se dirige a la carpeta `ros_catkin_ws`. Luego escribimos y ejecutamos el siguiente comando el cual tomará unos minutos en finalizar el proceso:

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

Cuando finalice el proceso de compilación debemos de observar que la terminal se muestre como la figura 45.

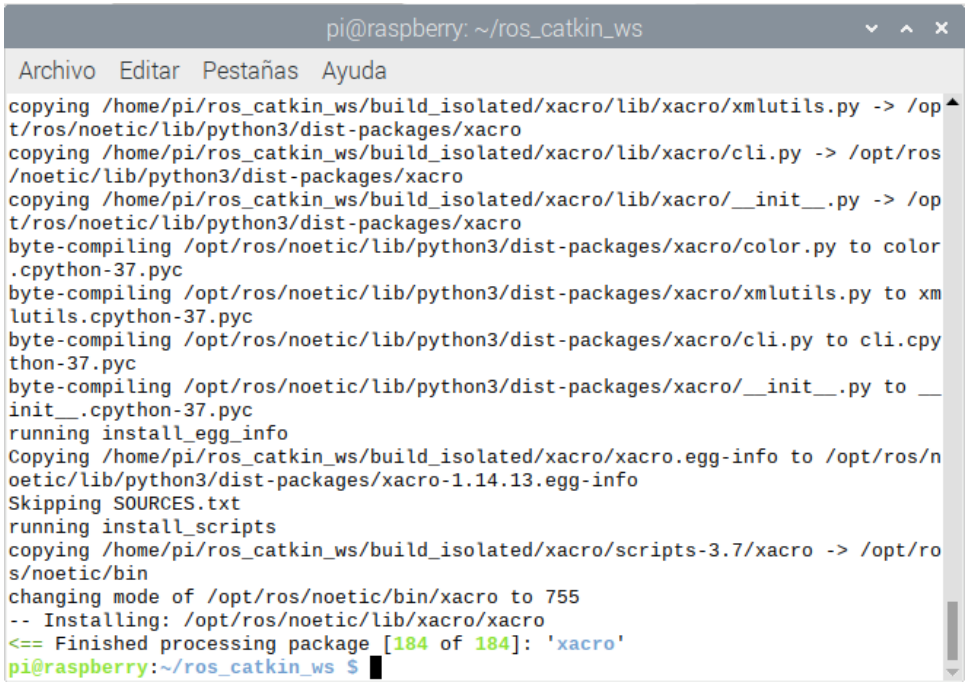


Figura 45. Captura de pantalla de la ventana de terminal al finalizar el proceso de compilación.

Con esto último termina el proceso de instalación de ROS Noetic en la tarjeta Raspberry Pi 4 modelo B como en una máquina virtual.

Sección D

Tutoriales para comprobar el funcionamiento de ROS Noetic

Publicador y Suscriptor

En este anexo se explica paso a paso cómo realizar un ejemplo sencillo para entender los conceptos de publicador y suscriptor, así como los temas de cómo crear un espacio de trabajo, crear un paquete, crear los nodos y por último verificar el buen funcionamiento de ROS Noetic en su versión **Desktop** instalada en la tarjeta Raspberry Pi 4 modelo B o en una máquina virtual.

Este ejemplo fue elaborado con base en ejemplos de [6], [7]. El objetivo de este ejemplo es crear un nodo publicador que imprima cada vez un número cuyo valor se va incrementando consecutivamente, mientras el nodo suscriptor escucha e imprime la información que le es enviada del publicador.

Antes de empezar se necesita instalar una herramienta, la cual es un editor de texto llamado “gedit”. Este editor se instala ejecutando el siguiente comando en la terminal, por lo que abrimos una terminal nueva y se escribe lo siguiente:

```
sudo apt-get install gedit
```

Tras ejecutar el comando nos pregunta si se deseamos continuar, a lo que decimos que sí y oprimimos la tecla s y enseguida la tecla enter, figura 46.

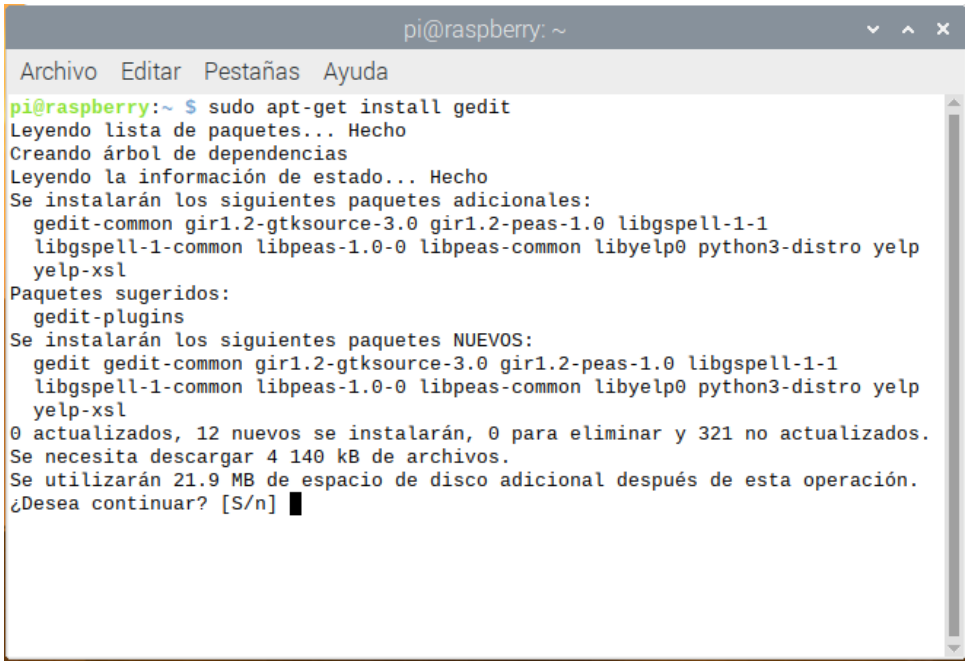


Figura 46. Realizando la instalación de editor de textos "gedit" desde la terminal.

Tomará unos minutos para terminar la descarga e instalación de los archivos que se requieren para tener instalado el editor “gedit”.

Figura 47. Finalización de instalación de "gedit".

Para confirmar que termino el proceso de instalación, al final de la ventana de la terminal, tiene que aparecer la siguiente línea **`pi@raspberry:~$`**. Después se cierra la terminal.

Una vez concluida la instalación, se abre una terminal nueva y ejecutamos el siguiente código:

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

Después creamos el espacio de trabajo, pero primero se designa y se crea el directorio de la ubicación del espacio de trabajo, esto se hace escribiendo y ejecutando la siguiente línea en la terminal:

```
mkdir –p ~/catkin_et/src
```

El siguiente paso es dirigirnos al interior de la carpeta **src** ejecutando el comando en terminal como sigue:

```
cd ~/catkin_et/src
```

En seguida se inicializa el espacio de trabajo, por lo que se escribe y se ejecuta el comando siguiente en terminal:

```
catkin_init_workspace
```

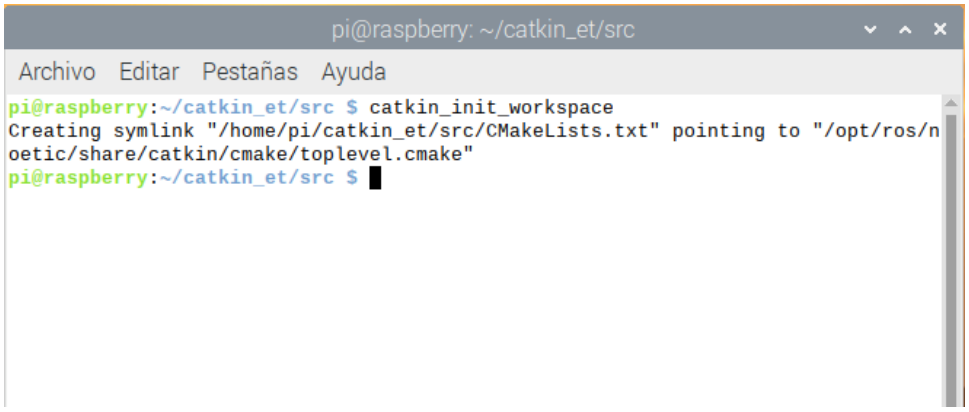


Figura 48. Tras ejecutar comando para inicializar el espacio de trabajo.

Una vez ejecutado el comando se obtendrán los resultados que se ven en la figura 48. Los resultados confirman que el espacio de trabajo se ha iniciado correctamente. Seguidamente escribimos el comando:

```
cd ..
```

El comando anterior solo nos regresa un nivel atrás del directorio, por lo que ahora nos encontramos en la ruta `~/catkin_et`. Continuando, se procederá a la construcción del espacio de trabajo, esto se hace ejecutando el siguiente comando en terminal:

```
catkin_make
```

Una vez ejecutado el comando se obtendrán en terminal los resultados que se observan en la figura 49, esto nos comprueba que se creó satisfactoriamente el espacio de trabajo.

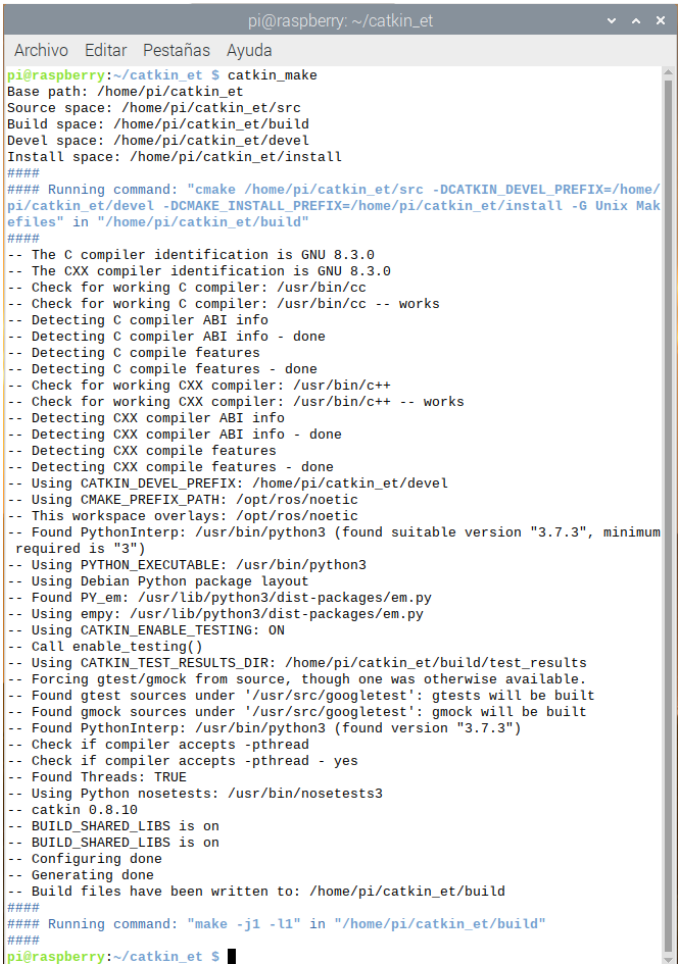


Figura 49. Resultado después de ejecutar el comando catkin_make.

Nos volvemos a dirigir dentro de la carpeta **src** utilizando el siguiente comando en terminal:

```
cd src
```

Como siguiente paso, debemos de crear el paquete del proyecto, el cual nombraremos como: “numeros”, esto lo haremos ejecutando el siguiente comando en terminal:

```
catkin_create_pkg  numeros std_msgs rospy
```

Después de ejecutar la instrucción obtendremos como resultado lo que se observa en la figura 50.

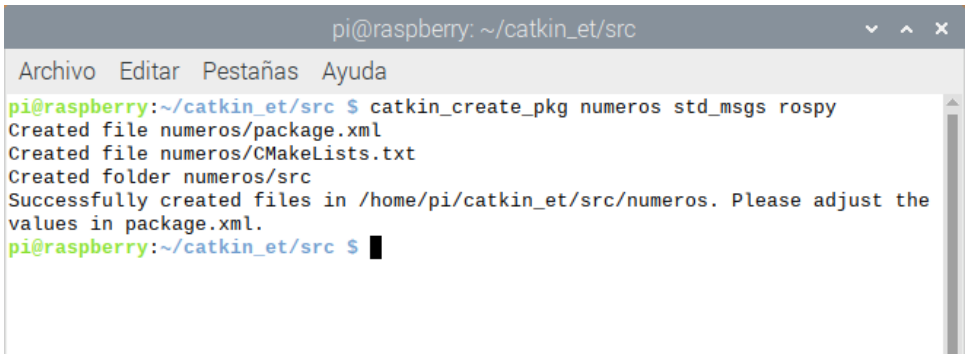


Figura 50. Resultados mostrados en terminal después de crear el paquete "numeros".

Lo que se desplegó en la terminal es para verificar que se creó con éxito nuestro paquete, así como hace mención de los archivos y directorios que contine un paquete. Una vez hecho lo anterior, ahora nos dirigimos a nuestra carpeta de nuestro paquete “números”, para ello ejecutamos en terminal el siguiente comando:

```
cd  numeros
```

Como siguiente paso, ya estando dentro de nuestra carpeta, creamos una carpeta nueva donde estarán nuestros dos nodos, publicador y suscriptor, escribimos el comando en terminal:

```
mkdir scripts
```

Nos ubicamos dentro de la carpeta **scripts** para crear nuestros nodos. Esto lo hacemos con el siguiente comando en terminal:

```
cd scripts
```

Los nodos los elaboraremos por separado, entonces primero se crea el nodo publicador, para ello simplemente ejecutamos el comando en la terminal:

```
gedit numeros_publisher.py
```

Recordemos que para nuestro caso los scripts estarán codificados en lenguaje python, es por eso que cada script tiene extensión .py.

Después de haber ejecutado el comando, como resultado se abrirá un editor de textos llamado gedit, el cual abre un archivo vacío con el nombre **numeros_publisher.py.** Se procede a escribir el siguiente contenido[2] en el script y se guardan los cambios:

```
#!/usr/bin/env python3
# license removed for brevity
import rospy
from std_msgs.msg import UInt32

def talker():
    pub = rospy.Publisher('numero_pub', UInt32, queue_size=10)
    rospy.init_node('numeros_publisher', anonymous=True)
    r = rospy.Rate(1)
    num_int = 0
    while not rospy.is_shutdown():
        rospy.loginfo(num_int)
        pub.publish(num_int)
        r.sleep()
        num_int = num_int + 1

if __name__ == '__main__':
    try:
        talker()
```

[2] Script con referencia a [8]

```
    except rospy.ROSInterruptException: pass
```

Cerramos el editor de textos. Ahora toca el turno de crear el script para el nodo suscriptor. De igual manera que el otro script, también lo crearemos desde la terminal. En la terminal abierta que tenemos ejecutamos el siguiente comando:

```
gedit numeros_subscriber.py
```

Para el caso de **numeros_subscriber.py** agregamos la siguiente información[3] y guardamos los cambios:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import UInt32

def callback(data):
    rospy.loginfo(" Recibo numero: [%d]",data.data)

def listener():

    rospy.init_node('numeros_subscriber', anonymous=True)

    rospy.Subscriber("numero_pub", UInt32, callback)

    rospy.spin()

if __name__ == '__main__':
    listener()
```

Una vez terminada la edición se cierra y se guardan los cambios. Ya que se tienen los dos scripts con la información correspondiente a la funcionalidad de cada nodo que se creó, en este caso el nodo publicador y el subscriptor, estos necesitan configurarse para que sean archivos ejecutables. Para ello, ejecutamos las

[3] Script con referencia a [8]

siguientes líneas en la terminal que ya se tiene abierta. Se tienen que ejecutar por separado (una por una).

```
chmod +x numeros_publisher.py
chmod +x numeros_subscriber.py
```

Después nos dirigimos a la ruta **`cd ~/catkin_et`** y en seguida ejecutamos el comando:

```
catkin_make
```

En esta misma terminal abierta ejecutamos el siguiente comando que iniciará el nodo maestro de ROS:

```
roscore
```

Seguidamente abrimos dos nuevas terminales y en cada una de ellas ejecutamos los siguientes comandos, línea por línea y por separado:

```
source /opt/ros/noetic/setup.bash
source ~/catkin_et/devel/setup.bash
```

La figura 51 muestra de manera grafica las terminales que tenemos abiertas. Para mayor entendimiento y visualización, se recomienda acomodar de esta misma manera las ventanas.

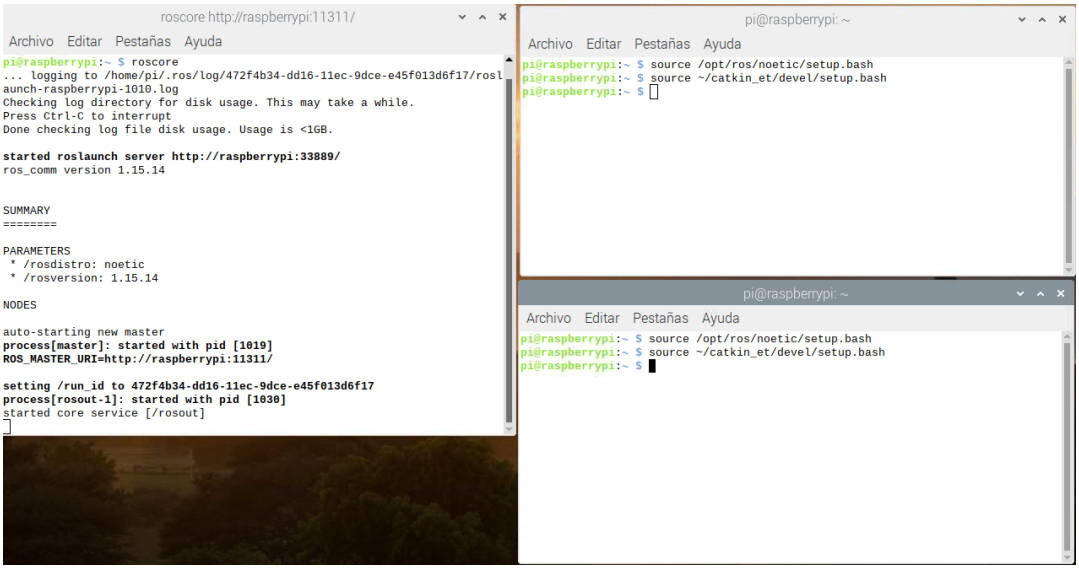


Figura 51. Captura de pantalla. Ventana izquierda: terminal que ejecuta roscore. Ventana superior e inferior derecha: Se van a ejecutar los nodos.

Ahora bien, en la terminal inferior derecha, se ejecuta el nodo publicador con ayuda del siguiente comando:

```
rosrun numeros numeros_subscriber.py
```

Mientras que en la terminal superior derecha ejecutamos la siguiente línea, que inicia el nodo publicador:

```
rosrun numeros numeros_publisher.py
```

Tras ejecutar los dos nodos se obtendrán resultados en las terminales donde se ejecutaron los respectivos nodos.

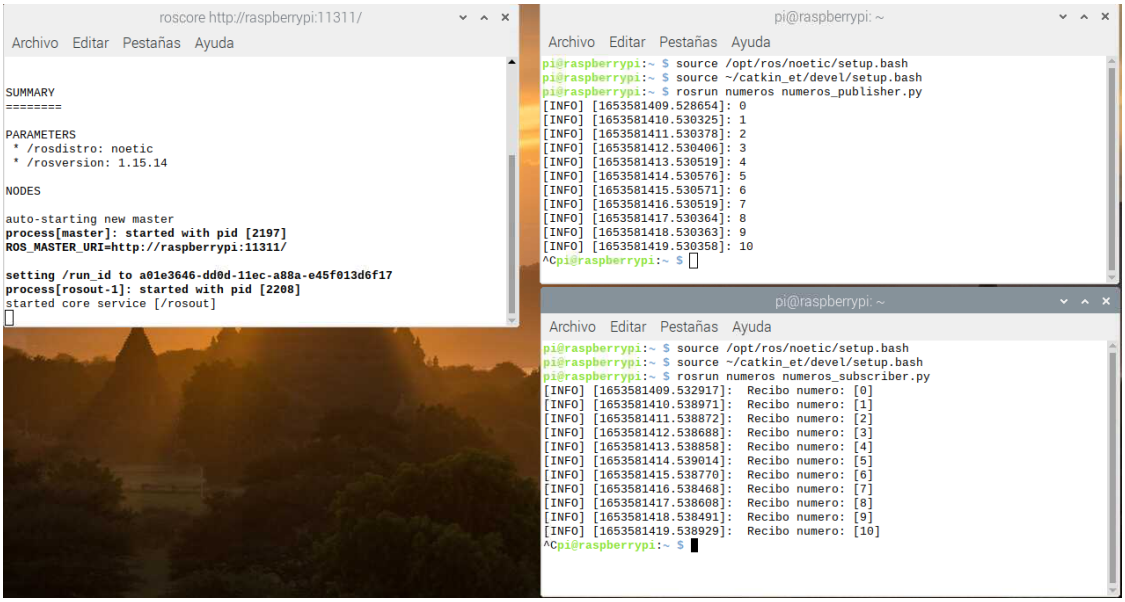


Figura 52. Captura de pantalla. Ventana izquierda: terminal que ejecuta roscore. Ventana superior derecha: Nodo publicador. Ventana inferior derecha: Nodo subscriptor.

Observamos en la figura 52 que el nodo publicador imprime cada vez un número cuyo valor se va incrementando consecutivamente (ventana de terminal ubicada en el lado superior derecho), mientras el nodo suscriptor escucha e imprime la información que le es enviada del publicador (ventana de terminal ubicada en el lado derecho inferior).

Para terminar con la ejecución o parar el proceso del nodo publicador y subscriptor, simplemente ejecutamos en cada terminal la combinación de teclas de **Ctrl** y **C** al mismo tiempo.

Turtlesim

Este es otro ejemplo que ayuda a la comprensión y manejo del software ROS, porque explica desde cómo ejecutar un proyecto, así como familiarizarse con algunos comandos y ver qué información nos proporcionan [5]. El paquete Turtlesim es la simulación de una tortuga en la cual podremos modificar su nombre y moverla con ayuda de las teclas del teclado. Esta tortuga la veremos de manera gráfica en una ventana para ver su comportamiento. De igual manera este ejemplo nos ayuda a comprobar que efectivamente se ha instalado correctamente ROS Noetic en la tarjeta Raspberry Pi o también para comprobar el buen funcionamiento de todo el proceso que se realizó en la máquina virtual para simular la tarjeta Raspberry Pi, esto último cuando no se cuenta con la tarjeta física.

Ahora bien, para trabajar con el paquete Turtlesim, primero se comienza instalando los tutoriales de ROS, estos paquetes serán instalados en el directorio donde se encuentra compilado ROS Noetic, estamos hablando de la carpeta `ros_catkin_ws`. Por lo tanto, abrimos una terminal nueva y ejecutamos la siguiente línea:

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

Y después se ejecuta la siguiente línea para ubicarnos en el directorio `ros_catkin_ws.`

```
cd ~/ros_catkin_ws
```

El siguiente paso es crear un paquete de instalación el cual estará compuesto por el paquete ROS en su versión Desktop y el paquete **ros_tutorials**. Para hacer lo anterior, en la terminal ejecutamos la siguiente línea:

```
rosinstall_generator desktop ros_tutorials --rosdistro noetic --
      deps --wet-only --tar > noetic-custom_ros.rosinstall
```

No se muestra ningún resultado en la terminal. A continuación, se debe de agregar el paquete de instalación que se creó con el nombre de `noetic-custom_ros.rosinstall` al directorio fuente, ósea la carpeta **src**, todo esto se hace ejecutando el comando:

```
wstool merge -t src noetic-custom_ros.rosinstall
```

De igual manera no se ve reflejado en la terminal algún resultado. Como se realizaron cambios, agregando nuevos paquetes en el espacio de trabajo donde se encuentran los paquetes de ROS Noetic y ahora también **ros_tutorials**, este espacio de trabajo se debe de acuatizar con los cambios hechos. Escribimos en terminal la siguiente línea:

```
wstool update -t src
```

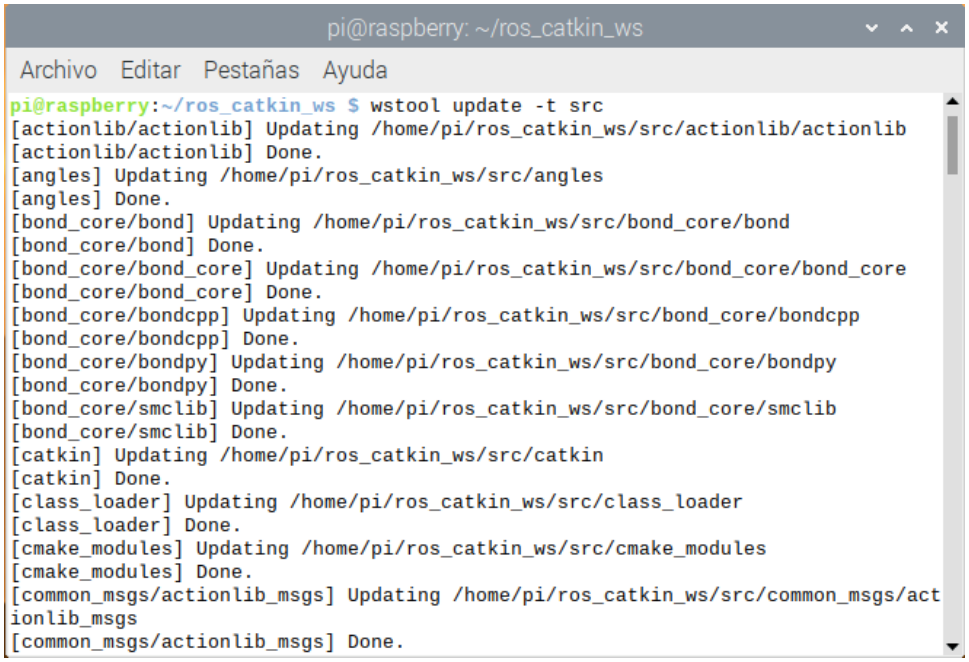


Figura 53. Proceso de actualización del espacio de trabajo.

En la figura 53 se observa que al ejecutar el comando comienza un proceso de actualización que puede demorar unos minutos. Una vez terminado este proceso toca el turno de actualizar e instalar las dependencias requeridas para el nuevo paquete de **ros_tutorials**. Por lo tanto, se ejecuta en terminal el comando que sigue:

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

Como único resultado se va a mostrar el mensaje confirmando que todas las dependencias requeridas han sido instaladas satisfactoriamente. Como paso final es realizar la compilación de todos los paquetes que se encuentran en el espacio de trabajo. Para proceder con la compilación, se ejecuta en terminal la siguiente línea:

```
sudo ./src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/noetic
```

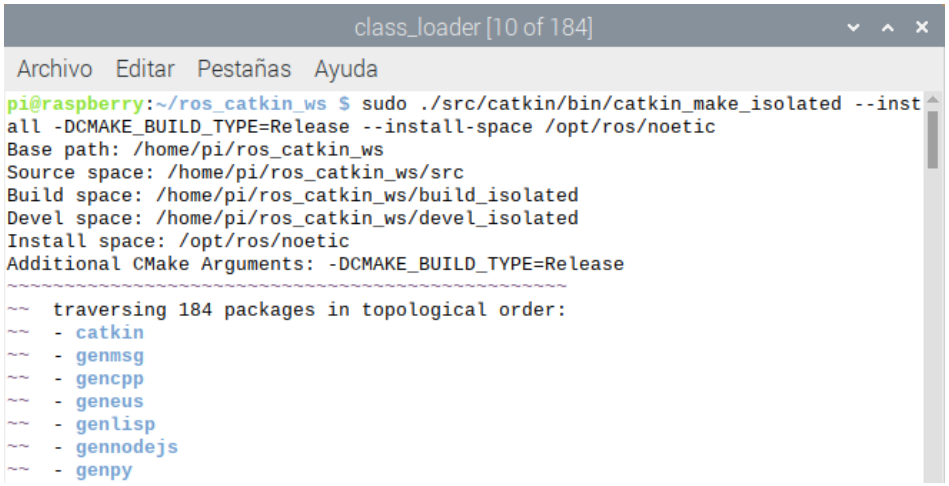


Figura 54. Realizando la compilación del espacio de trabajo.

Para este caso al ejecutar el comando se deben de obtener resultados como los que se muestran en la figura 54. Se espera hasta que finalice el proceso y cerramos la terminal, con esto se da por terminada la instalación de los tutoriales de ROS, la cual contine el ejemplo del proyecto del Turtlesim.

Para comprobar que efectivamente tenemos este proyecto, simplemente abrimos una nueva terminal y primero ejecutamos el código:

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

Después escribimos en la terminal el código que sigue:

```
rospack find turtlesim
```

Esta línea de comandos permite obtener información acerca de un paquete o proyecto en específico, en este caso Turtlesim. Así mismo, este comando nos arroja

una ruta de directorio en el que se encuentra ubicado el proyecto Turtlesim. Eso se puede apreciar en la siguiente figura.

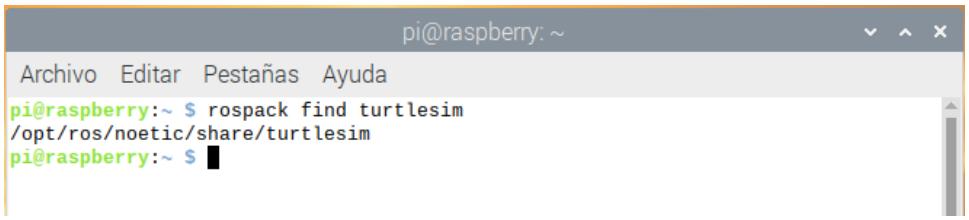


Figura 55. Directorio en que se encuentra el paquete Turtlesim.

Una vez confirmada la instalación correcta del Turtlesim en la tarjeta Rapsberry Pi o en la máquina virtual, se procede a ejecutar el paquete Turtlesim. Primero en la terminal que tenemos abierta ejecutamos el nodo maestro, para ello escribimos:

```
roscore
```

Después abrimos una terminal nueva y ejecutamos el siguiente código:

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

Seguidamente escribimos la siguiente línea de comandos para dar comienzo a la ejecución del paquete Turtlesim:

```
rosrun turtlesim turtlesim_node
```

Tras ejecutar la línea anterior se abrirá una ventana nueva como la figura que se muestra a continuación.

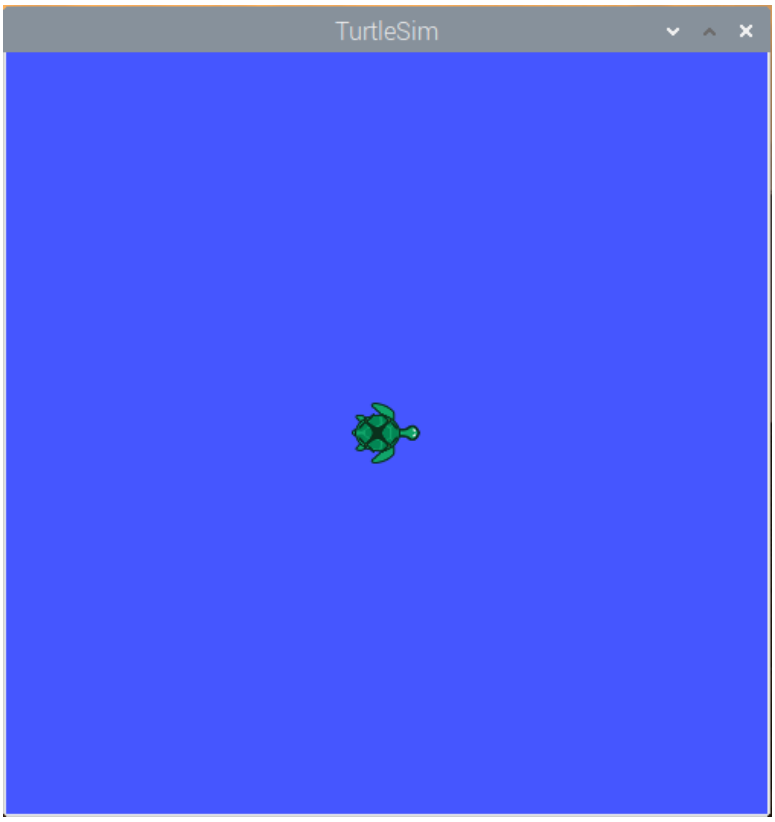


Figura 56. Ventana que muestra gráficamente una tortuga del paquete Turtlesim.

Se observa que la tortuga se encuentra en el centro de la ventana. Cabe destacar que este paquete tiene la característica de que cada vez que se ejecute cambia de forma gráfica, teniendo así varios diseños de la tortuga.

Sección E

OpenCV en ROS Noetic usando una cámara web

Esta sección explica de manera detallada un ejemplo sobre cómo configurar y crear un proyecto en ROS para hacer uso de las librerías de OpenCV, que se encuentran disponibles para ROS Noetic [9].

Para continuar con este ejemplo se debe contar con una cámara web con puerto USB y tener instalado el editor de textos “gedit”. En este caso se cuenta con la cámara web modelo Tmcam 8305.

Figura 57. Cámara web Tmcam 8305.

Como primer paso, por medio de la terminal, se dirige al interior del espacio de

trabajo con el que ya se cuenta, en este caso llamado **catkin_et**, después se ingresa a la carpeta **src** y dentro de ella se crea el paquete **cv_basics**, esto se hace ejecutando el comando en el terminal:

```
cd ~/catkin_et/src
```

```
catkin_create_pkg cv_basics image_transport cv_bridge sensor_msgs
rospy roscpp std_msgs
```

Ahora se dirige al interior de la carpeta **cv_basics** y estando dentro de ella se crea el directorio **scripts**, lugar donde se crearán los nodos publicador y suscriptor que se usarán para este ejemplo. El directorio se crea ingresando en terminal lo siguiente:

```
mkdir scripts
```

Ya creada la carpeta **scripts**, se ingresa a ésta y se crea el primer script que corresponde al publicador y que lleva por nombre **web_pub.py**. Cabe mencionar que estará codificado en python y para crearlo solo se ejecuta el siguiente comando en terminal:

```
gedit webcam_pub.py
```

se abre un editor de textos, en este caso gedit, y se pega el siguiente código[4]:

```
#!/usr/bin/env python3
# Basics ROS program to publish real-time streaming
# video from your built-in webcam
# Author:
# - Addison Sears-Collins
# - https://automaticaddison.com
```

[4] Código con referencia a [9]

```
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

def publish_message():
  pub = rospy.Publisher('video_frames', Image, queue_size=10)
  rospy.init_node('video_pub_py', anonymous=True)
  rate = rospy.Rate(10)
  cap = cv2.VideoCapture(0)
  br = CvBridge()

  while not rospy.is_shutdown():
      ret, frame = cap.read()
      if ret == True:
      rospy.loginfo('publicando fotograma de video')
      pub.publish(br.cv2_to_imgmsg(frame))
      rate.sleep()

if __name__ == '__main__':
  try:
      publish_message()
  except rospy.ROSInterruptException:
      pass
```

En las primeras líneas se importan las librerías que se van a ocupar, como: **rospy**, **Image**, **CvBridge;** que se encargan de convertir la imagen de ROS y OpenCV y por último la librería **cv2** que corresponde a OpenCV. Como siguiente función se define **publish_message**, aquí se declara el publicador que usará el topic **video_frames** y el tipo de mensaje **Image**. Más adelante se define el nombre del nodo, este debe de ser único, es por ello que se define **Anonimous=True**. A continuación, se define el bucle que será de 10 Hz. Ahora se declara **cap** que es una función compuesta de la librería de **cv2** y usa **VideoCapture**, este tiene un argumento de “0”, que significa que usará la cámara web. De haber más cámaras conectadas se elige un valor diferente de 0 de acuerdo al dispositivo que se desee. Seguidamente se define

br, el cual se usará para convertir las imágenes entre ROS y OpenCV. Ya por último se define una función **while,** un ciclo que indica que mientras se esté ejecutando ROS y si se captura frame por frame un video se imprimirá en terminal un mensaje de: **Publicando fotograma de video** y seguidamente se publicará la imagen, para ello se usa la función **cv2_to_imgmsg**, la cual convierte una imagen de OpenCV en un mensaje de imagen de ROS. Se guardan los cambios y se cierra el editor de textos. En la misma terminal abierta se escribe y ejecuta el siguiente comando, este es con la finalidad de que el script sea ejecutable:

```
chmod +x webcam_pub.py
```

El terminal no muestra salida, pero si tras ejecutar el comando anterior sale un mensaje de “chmod: no se puede acceder a ‘webcam_pub.py’: No existe el fichero o el directorio”, entonces ejecutar el siguiente comando:

```
sudo chmod +x webcam_pub.py
```

En esa misma terminal se ejecuta el comando para crear ahora el suscriptor:

```
gedit webcam_sub.py
```

De igual manera se abre el editor de textos y se pega el siguiente código[5]:

```
#!/usr/bin/env python3
# Description:
# Subscribes to real-time streaming video from your built-in webcam.
#
# Author:
# Addison Sears-Collins
# https://automaticaddison.com

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
```

[5] Código con referencia a [9]

```
import cv2

def callback(data):
  br = CvBridge()
  rospy.loginfo("recibiendo fotograma del video")
  current_frame = br.imgmsg_to_cv2(data)
  cv2.imshow("camera", current_frame)
  cv2.waitKey(1)

def receive_message():
  rospy.init_node('video_sub_py', anonymous=True)
  rospy.Subscriber('video_frames', Image, callback)
  rospy.spin()
  cv2.destroyAllWindows()

if __name__ == '__main__':
  receive_message()
```

Las primeras líneas importan las librerías que se van a necesitar, en este caso del suscriptor, mismas que se usan en el publicador. Ahora se definen dos funciones: **callback** y **receive_message**. La función **callback** se encarga de convertir la imagen entre ROS y OpenCV, después imprimirá en terminal el mensaje: **Recibiendo fotogramas de vídeo**, también se usa la función **imgmsg_to_cv2**, pero ahora será para convertir los mensajes de imagen de ROS a imagen de OpenCV, por último, se declara una función que abre una ventana para observar lo que se recibe de la cámara web. Por otra parte, en la definición de la función **receive_message** se inicializa el nodo suscriptor, se le asigna un nombre único y también dentro de esta función se declara el nodo **Subscriber** el cuál se estará suscribiendo al topic **video_frames**, por medio del tipo de mensaje **Image** que recibirá datos de la función **callback**. Después se coloca la función **spin ()**, esto es para que el código no se detenga sino hasta que lo dicte este nodo. Al final se declara una función que

cerrará la ventana donde se observa lo que capta la cámara, esto se hará cuando se detenga la transmisión de video. Se guardan los cambios y se cierra el editor y en la misma terminal se ejecuta el siguiente comando para hacer que el script sea ejecutable:

```
chmod +x webcam_sub.py
```

Ahora se necesita crear el archivo **launch**, este es el encargado de iniciar los dos nodos, para ello se abre una nueva terminal y primero se dirige al interior de la carpeta **catkin_et** y luego se ejecuta el comando:

```
catkin_make
```

Este comando es para compilar los scripts de los nodos que se crearon anteriormente. Una vez que termine el proceso de compilación, en esta misma terminal se va al directorio **cv_basics** y dentro se crea la carpeta **launch** y después se va al interior de esta misma. Para hacerlo simplemente se ejecuta las siguientes líneas:

```
cd cv_basics

mkdir launch
```

Se crea el archivo **.launch** que lleva el nombre de **cv_basics_py.launch**, para ello se ejecuta en terminal el siguiente comando:

```
gedit cv_basics_py.launch
```

Nuevamente se abre una ventana nueva del editor de textos, se agrega el siguiente código[6]:

```
<launch>
  <node
     pkg="cv_basics"
     type="webcam_pub.py"
     name="webcam_pub"
     output="screen"
  />
  <node
     pkg="cv_basics"
     type="webcam_sub.py"
     name="webcam_sub"
     output="screen"
  />
</launch>
```

En el código anterior se declaran los nodos que se van a iniciar, en este caso **webcam_pub.py** y **webcam_sub.py**, se especifica el paquete al que pertenecen, su nombre, su tipo y la salida. Se guardan los cambios y se cierra el editor. Se abre una nueva terminal donde se inicia el core de ROS, para ello se ejecuta el comando **roscore**. Después, en la terminal que se tenía abierta se ejecuta el comando:

```
source ~/catkin_et/devel/setup.bash
```

Este comando es para indicar la fuente de donde se leerán los paquetes. Luego, en esa misma terminal, se ejecuta el siguiente comando para iniciar los nodos publicador y suscriptor:

```
roslaunch cv_basics cv_basics_py.launch
```

Como resultado en la terminal se mostrarán mensajes como en la figura 58.

[6] Código con referencia a [9]

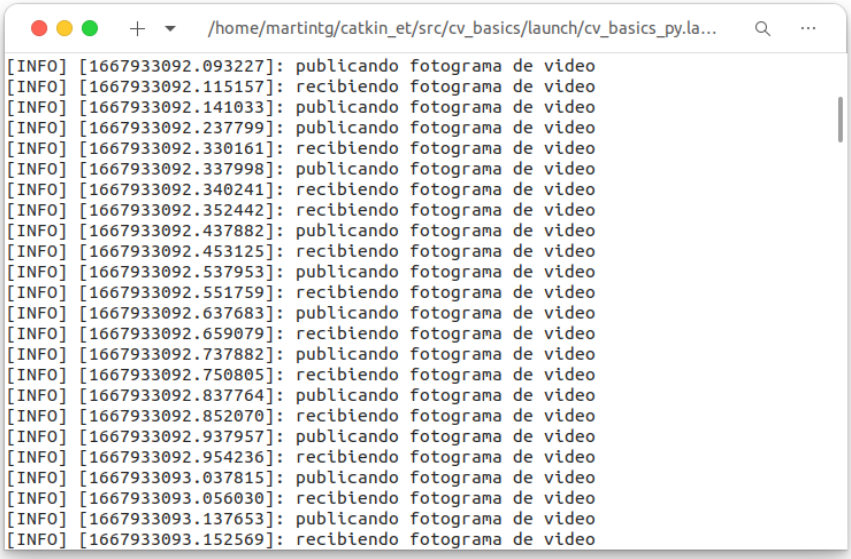


Figura 58. Resultados mostrados tras iniciar los nodos.

En la figura 58 se observa que se imprime en terminal el mensaje de “publicando fotograma de video” y “recibiendo fotograma de video”. Por otra parte, se abre una ventana en donde se estará observando lo que capta la cámara web, solo que no tiene el efecto espejo, figura 59.

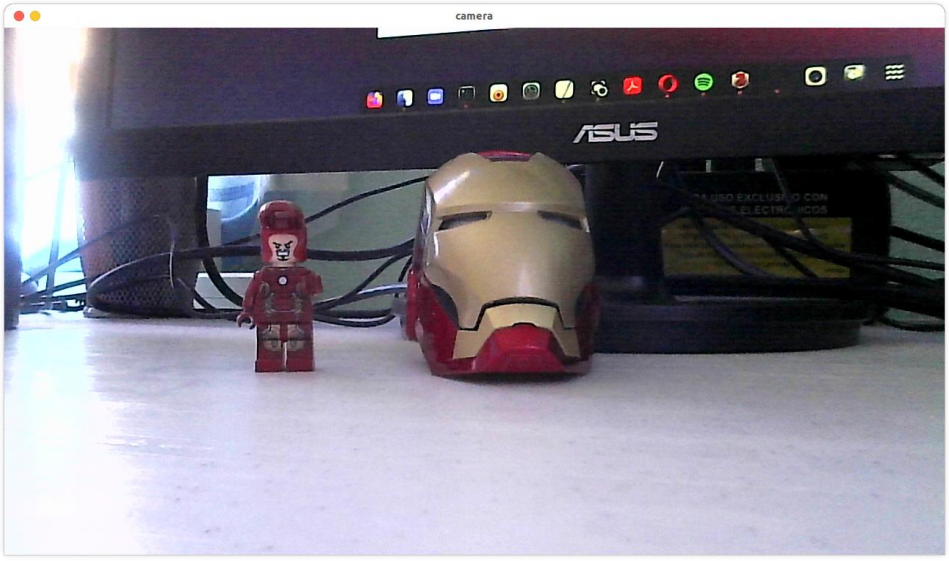


Figura 59. Ventana que se crea por el nodo suscriptor que muestra lo que captura la cámara web.

Para crear este efecto espejo de la imagen simplemente usamos una función de la librería **cv2** [10]. La función **flip** permite orientar la imagen de video como nosotros lo deseemos, en este caso solo queremos el reflejo, es por ello que se modificará el script de suscriptor, ya que es el nodo que genera la ventana para observar lo que se capta en la cámara. Desde la terminal se dirige al directorio **catkin_et/src/cv_basics/scritps** y con ayuda del editor de textos se abre el script. Una vez abierto se agregan las siguientes líneas:

```
mirror= cv2.flip(current_frame, 1)

cv2.imshow("mirror", mirror)
```

En este caso se define la variable **mirror,** que ocupa la función **flip** de la librería **cv2**. **flip** tiene en este caso dos atributos: el primero es **current_frame,** que son los datos que se obtienen al suscribirse al nodo que publica, el segundo atributo corresponde a la orientación que se le dará a la imagen, para el caso de solo hacer el reflejo se le coloca el valor de **1**. La siguiente línea que se agrega es solo para crear una ventana que muestre la imagen ya reflejada. La figura 60 muestra dónde se agregan las líneas en el script.

```
15
16 def callback(data):
17
18   br = CvBridge()
19
20   rospy.loginfo("recibiendo fotograma de video")
21
22   current_frame = br.imgmsg_to_cv2(data)
23
24   mirror= cv2.flip(current_frame, 1)
25
26   cv2.imshow("camera", current_frame)
27
28   cv2.imshow("mirror", mirror)
29
30   cv2.waitKey(1)
31
```

Figura 60. Parte del código del script del nodo suscriptor donde se agregan las líneas con la función flip.

Se guardan los cambios y se cierra el editor de textos, se inician los nodos y ahora se abren dos ventanas, una con la imagen reflejada, figura 61 y la otra no, figura 62.

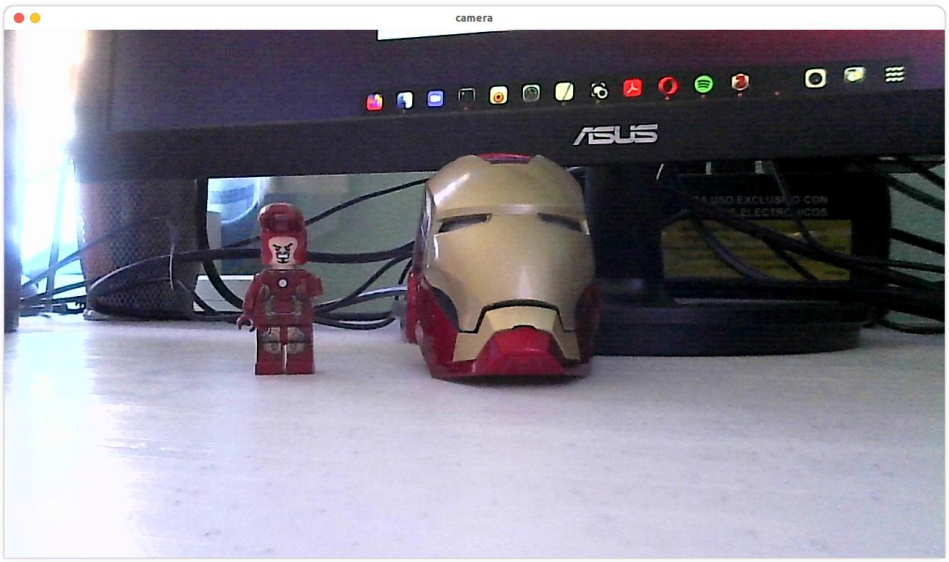


Figura 61. Ventana que no tiene función de reflejo.

Figura 62. Ventana que muestra la imagen con reflejo.

Con todo lo anterior se da por terminado el ejemplo de uso de la librería OpenCV para ROS.

Sección F

Comunicación entre Rapsberry Pi y el robot Lego Mindstorm EV3

El robot EV3 puede comunicarse de diferentes formas con algún equipo de cómputo, ya sea inalámbricamente o por cableado. Para este caso será por cable. Por lo tanto, la comunicación será desde el puerto miniUSB del robot que se encuentra ubicado en el Brick a un puerto USB de la Raspberry Pi 4 por medio de un cable con las entradas correspondientes.

Las siguientes figuras muestran los botones y puertos del Brick, esto es con el objetivo de familiarizarse con él dispositivo.

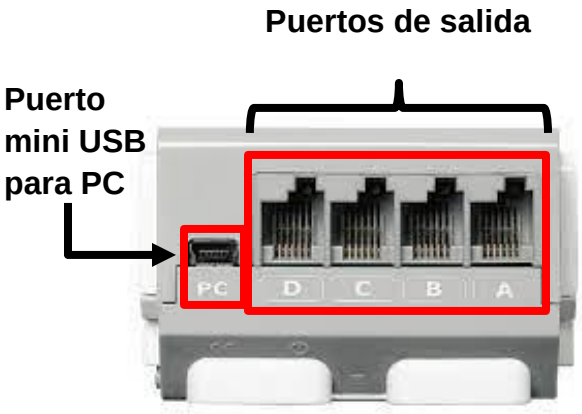


Figura 63. Parte superior del Brick.

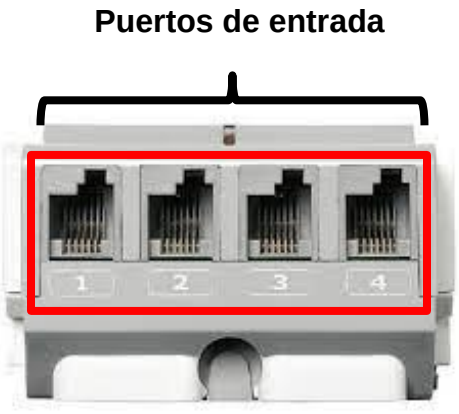


Figura 64. Parte inferior del Brick.

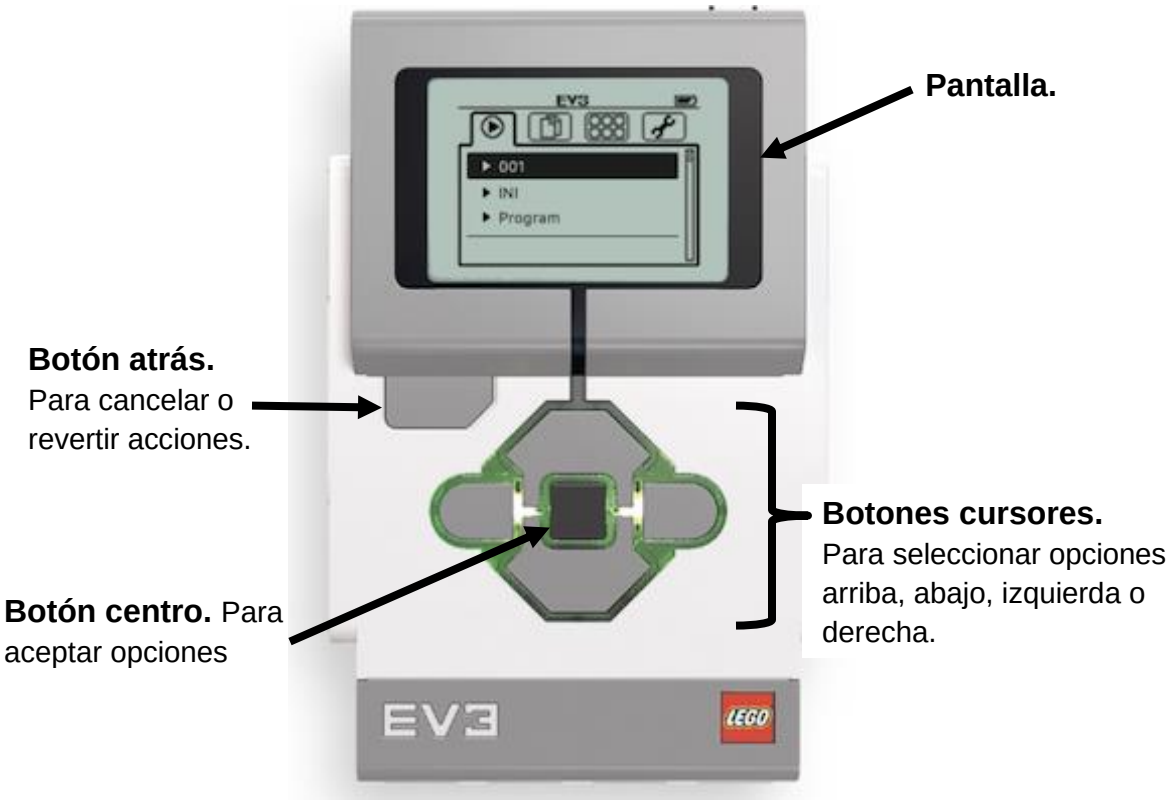


Figura 65. Parte frontal del Brick.

Para que la comunicación funcione se deben de realizar ciertas configuraciones [11]. El primer paso es encender el Brick, para ello se deja presionado el botón central. También se procede a encender la tarjeta Raspberry Pi. El Brick debe de tener cargado el SO llamado ev3dev. Una vez encendido el Brick se obtendrá el menú que se muestra en la figura 66.

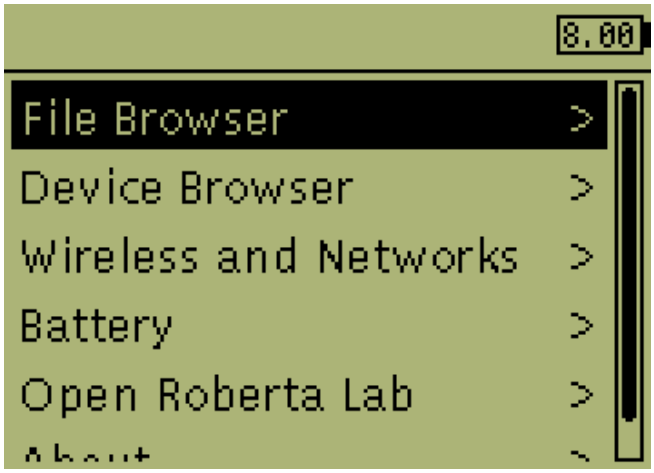


Figura 66. Menú inicial del SO ev3dev en el Brick.

Con los botones de navegación se dirige a la opción que dice **Wireless and Networks** y se presiona el botón centro para entrar a ese submenú, aparecen otras opciones se elige y se entra en la opción **All Network Connections**. Tal vez en este submenú no aparezca ninguna opción, por ello conectamos el Brick a un puerto de la Raspberry Pi, esto se hace con ayuda del cable. Enseguida aparece una sola opción, se selecciona y se elige la opción de **Wired**, seguidamente aparece otro listado de opciones, se elige la que dice **IPv4**. Al momento de entrar aparece información sobre la conexión de wired, donde se especifica una dirección IP, una máscara de red y un Gateway, elegimos la única opción que dice **Changes**. Ahora aparece una mini ventana con tres opciones visibles, para este caso se elige la que dice **Load Linux defaults** (con esto asigna una dirección IP default al Brick que es la 10.42.0.3). Después se regresa al menú **Wired**, para ello se presiona el botón atrás. Se verifica que indique que está en el **Status** Online; si no es así, se selecciona la opción de **Connect**. En ese mismo menú seleccionamos la opción de

Connect Automatically y marcamos esa opción presionando el botón centro. Por último, se regresa al menú principal, esto se hace presionando el botón atrás hasta estar en el menú principal.

Ahora se pasa a la tarjeta Raspberry Pi, se abre una nueva terminal y se ejecuta el comando:

```
sudo ip ad add 10.42.0.1/24 dev usb0
```

Este comando lo que hace es asignar una dirección IP a el puerto usb0, en este caso la dirección es **10.42.0.1**, aquí se puede asignar la dirección que desee el usuario. Una vez que se realizó esta configuración ya se puede realizar comunicación ssh del EV3 con el equipo de cómputo, para hacerlo simplemente se coloca en una terminal el siguiente comando:

```
ssh robot@10.42.0.3
```

Tras ejecutar el comando se solicitará el password, en este caso es **maker**. Este tipo de conexión es útil para crear los scripts para programar el comportamiento del EV3.

Como última configuración es dejar la dirección IP asignada al puerto **usb0** como estática, para ello se abre una terminal y se ejecuta el siguiente comando:

```
sudo vi /etc/dhcpcd.conf
```

Se abre el archivo **conf** por medio de un editor de textos y se verifica que dentro de la información contenida estén las siguientes líneas, si no están se agregan:

```
interface usb0
static ip_address=10.42.0.1/24
```

Seguidamente se guardan los cambios y se cierra el editor de textos, la terminal y se reinicia la tarjeta Raspberry Pi, eso es para que se guarden correctamente los cambios que se realizaron. Listo se tiene comunicación de la tarjeta Raspberry Pi con el robot Lego EV3.

Sección G

Configuraciones iniciales para controlar el robot EV3 de manera remota

Para controlar el robot EV3 de manera remota es necesario contar con una biblioteca de Python que se llama RPyC. Esta biblioteca ayuda a controlar el robot EV3 de manera remota, en este caso será por medio de la Raspberry Pi. Como requisito principal, para realizar lo anteriormente mencionado, la biblioteca debe de estar instalada tanto en el Brick del EV3 como en la Raspberry Pi [12]. En esta sección describiremos de manera explícita todo lo que se tiene que realizar para controlar de manera remota al robot EV3 y para comprobar el funcionamiento de la biblioteca, realizando un ejemplo sencillo.

Antes de comenzar con la instalación es necesario contar con lo siguiente: Brick EV3 con SO ev3dev, cable USB a mini USB, un adaptador WiFi USB, Raspberry Pi, un equipo de cómputo. Tanto la Raspberry como el Brick deben de contar con

conexión a Internet para hacer la descarga de la biblioteca RPyC, y el equipo de cómputo debe de estar en la misma red que el Brick.

Como primer paso, se conecta en el puerto USB del Brick el adaptador WiFi, se enciende el Brick, se espera a que cargue el SO y se configura para que el Brick tenga acceso a Internet. Seguidamente, en el equipo de cómputo, se abre una terminal nueva para realizar conexión SSH con el Brick, para ello, en la terminal se ejecuta el siguiente comando:

```
ssh robot@192.168.1.83
```

Recordamos que la dirección IP que se coloca es la que se ve en la parte superior izquierda de la pantalla del Brick, figura 67.

Figura 67. Menú de inicio de ev3dev en el brick del robot lego EV3.

Tras ejecutar el comando se pide un password, que en este caso es **maker**. Después, se muestra el mensaje de ev3dev y luego la línea **robot@ev3dev**; con

ello se comprueba que se ha realizado correctamente la conexión SSH con el Brick. Como paso siguiente se realiza la instalación de la biblioteca RPyC en el Brick, para ello se ejecuta el comando que sigue:

```
sudo easy_install3 rpyc
```

Tras ejecutar el comando puede que tarde unos minutos en desplegarse algún resultado. La siguiente figura 68 es un ejemplo de los resultados que se muestran tras instalar la biblioteca.

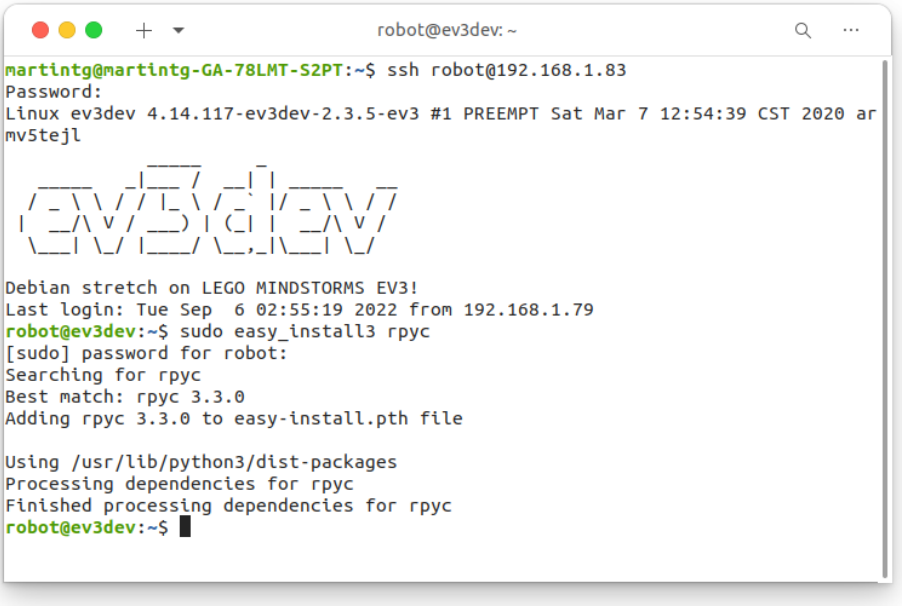


Figura 68. Conexión ssh al Brick del robot EV3 realizado desde la terminal de una PC y realizando instalación de RPyC.

Una vez que se termina la instalación es necesario verificar la versión de RPyC instalada en el Brick. Esto es porque tanto el Brick como el PC o equipo de cómputo, en este caso la Raspberry Pi, deben de contar con la misma versión de RPyC, de lo contrario, al momento de realizar o ejecutar un script desde la Raspberry para controlar al robot EV3 se generarán errores.

Continuando con la terminal que se tiene abierta en el equipo de cómputo con la conexión SSH con el Brick, se ejecuta el siguiente comando:

```
python3
```

El comando anterior ejecuta Python en su versión 3, esto es porque RPyC es una biblioteca de Python. Seguidamente se escribe y ejecuta la línea:

```
import rpyc
```

Esta línea se encarga de cargar la librería RPyC, ahora se ejecuta el comando que sigue:

```
rpyc__version__
```

Como resultado de escribir lo anterior se obtiene lo que se muestra en la figura 69.

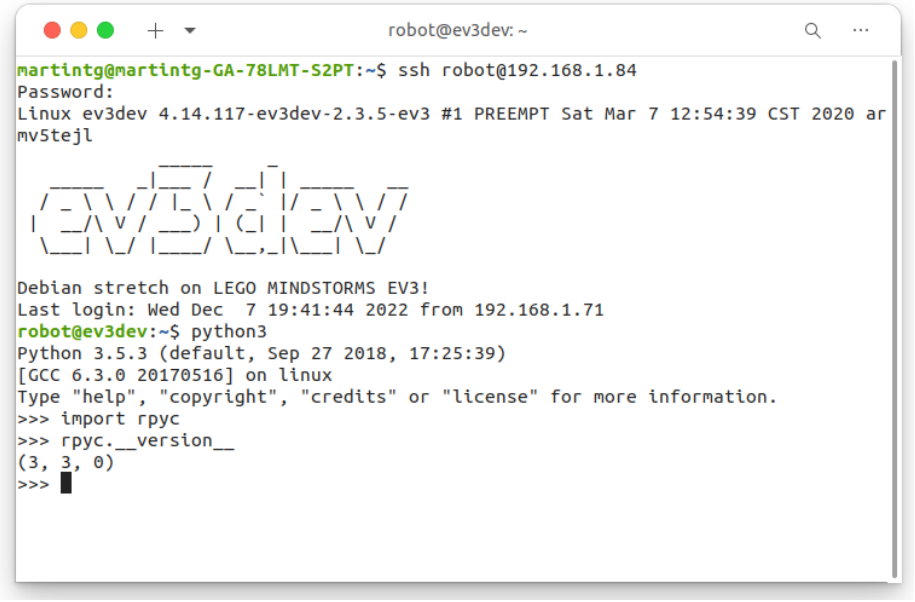


Figura 69. Verificando la versión de RPyC instalado en el Brick del robot EV3.

Como resultado se muestra que el Brick cuenta la versión 3.3.0 de la biblioteca RPyC. Con esta información es que se procede a realizar la instalación de RPyC en su versión 3.3.0 en la Raspberry Pi.

Se enciende la Raspberry Pi y se verifica que tenga conexión a Internet, después se abre una terminal nueva y se ejecuta el siguiente comando para comenzar con la instalación de RPyC:

```
sudo pip3 install rpyc==3.3.0
```

Para este caso en concreto se especifica en el comando la versión de RPyC que se desea instalar. Luego de ejecutar el comando se observan los resultados en terminal como la siguiente figura 70.

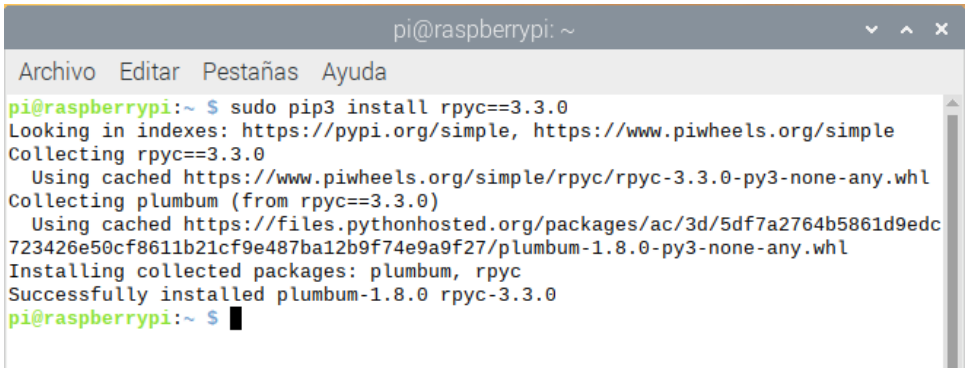


Figura 70. Resultados obtenidos en la terminal de la Raspberry Pi des pues de instalar RPyC.

Con lo anterior se da por terminado el proceso de instalación de la biblioteca tanto en la Raspberry Pi como en el Brick del robot EV3.

Se cierra la conexión SSH que existe entre el Brick y el equipo de cómputo, para ello simplemente se escribe **exit** en la terminal que se tiene abierta en el equipo de cómputo y por último se cierra la terminal.

Ahora se hace uso del cable USB a mini USB para conectar el Brick a un puerto USB de la Raspberry Pi, esto se observa en la siguiente figura 71.

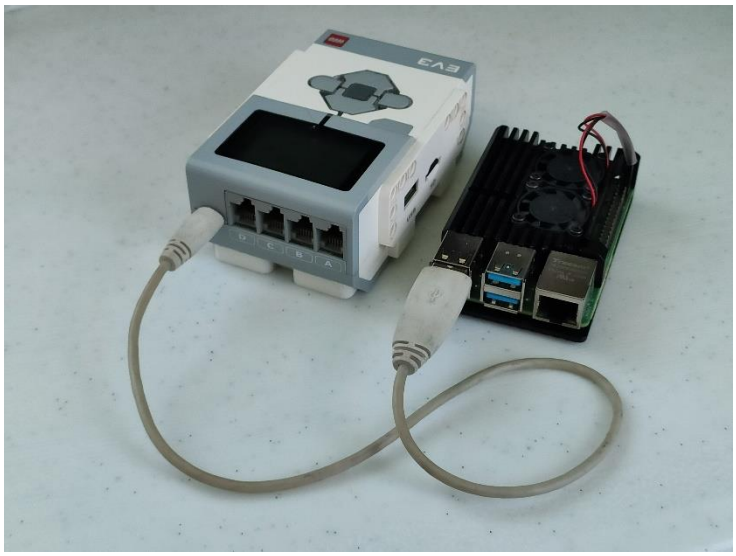


Figura 71. Conectando el Brick del EV3 con la Raspberry Pi.

Al momento de conectar el Brick con el puerto USB de la Raspberry Pi, en la pantalla superior izquierda del Brick ahora muestra la IP de `10.42.0.`3, recordar que ahora esa será la IP para conectarse vía SSH de la Raspberry Pi al Brick.

A continuación, realizaremos un pequeño script como ejemplo para verificar el buen funcionamiento de la biblioteca RPyC al realizar con éxito el control desde la Raspberry Pi al Brick por medio del cable USB. Para este caso, se usa un motor grande que va conectado a la salida A del Brick y éste va a girar por un segundo. También se agrega una instrucción más, en la que se usa la bocina integrada del Brick, para ello se usa una función para que se pronuncie en la bocina un texto se va a escribir en el script.

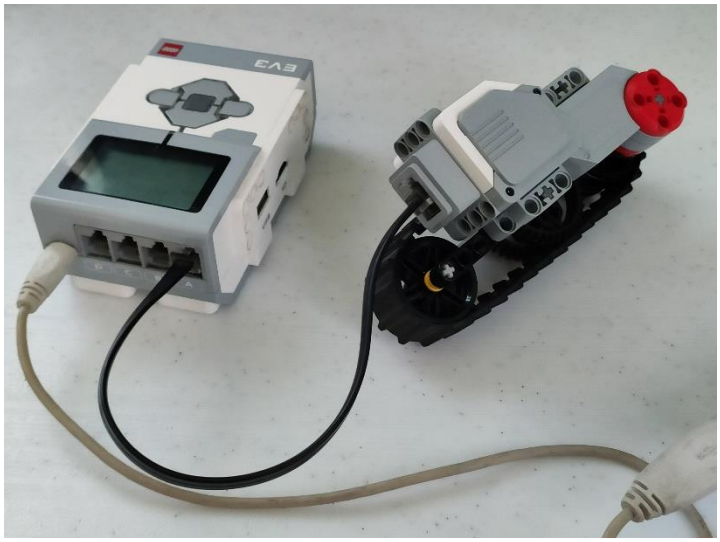


Figura 72. Conectando un motor grande al Brick del EV3 en los puertos de salida.

Estando en el escritorio de la Raspberry Pi se abre una terminal nueva y se escribe el siguiente código: **`mkdir rpyc_lego`**.

Este comando es para crear el directorio donde se ubicará el script de este ejemplo.

Después se dirige al interior de la carpeta rpyc_lego, y se ejecuta el comando: **`gedit mov_motor.py`**, se abre el editor de textos con un documento nuevo y en él se ingresa el siguiente código del script:

```
import rpyc

conn = rpyc.classic.connect('10.42.0.3') # host name or IP address of the
EV3

ev3dev2_sound = conn.modules['ev3dev2.sound']
ev3dev2_motor = conn.modules['ev3dev2.motor']

m = ev3dev2_motor.LargeMotor('outA')
m.run_timed(time_sp=1000, speed_sp=600)

sound = ev3dev2_sound.Sound()
sound.speak('Hola soy el robot Lego EV3!')
```

En la primera línea se declara el uso de la biblioteca RPyC. A continuación, se declara una variable **conn**, en la que se define la función **rpyc.classic.connect**, la cual ayuda a realizar la conexión remota de la Raspberry Pi con el Brick del robot. Para hacer posible esta conexión remota en la función anteriormente mencionada se escribe la dirección IP del Brick (**10.42.0.3**).

Las líneas **ev3dev2_sound** y **ev3dev2_motor** se declaran los módulos a ocupar en este caso el motor y el sonido que se reproducirá en la bocina del Brick.

Las siguientes instrucciones son para hacer que gire el motor grande por un segundo. Primero se define una variable m, a la que se le indica que del módulo **ev3dev2_motor** solo elija el **LargeMotor** (motor grande) y que éste se encuentra en **outA** (el motor grande está conectado en la salida A del Brick). Por último, se llama la variable **m** y se agrega la función de **run_timed**, la cual especifica cuánto tiempo va a girar el motor y a qué velocidad.

Las últimas líneas del código son para que se active la bocina del Brick para reproducir el texto que se desee. Primero se declara una variable sound, a la que se le asigna el módulo **ev3dev2_motor** y de ahí solo elijé la función **Sound()**. Después la instrucción siguiente se llama la variable **sound** y se agrega la función **speak**, que activa la bocina del Brick para reproducir una frase, la cual debe de escribirse entre los paréntesis () y entre comillas simples ‘’, en este caso la frase es: **Hola soy el robot Lego EV3!**[7]

[7] Consultar el siguiente link para conocer la biblioteca de Pyhton3 que ayuda a controlar todas las interfaces del robot EV3. Esta biblioteca se ejecuta únicamente teniendo el SO ev3dev en el Brick. https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/index.html

Terminado de escribir el script, se guardan los cambios y se cierra el editor de textos.

Estando en el escritorio de la Raspberry Pi se abre una nueva terminal para realizar la conexión SSH con el Brick. Recordar que ahora es la siguiente línea: `ssh robot@10.42.0.3`

Una vez que se realizó con éxito la conexión SSH, se debe de crear un servicio que inicie el rpyc_classic y funcione el control remoto desde la Raspberry Pi [13]. Estando en la terminal de la conexión SSH con el Brick se copia y ejecuta lo siguiente[8]:

```
echo "[Unit]
Description=RPyC Classic Service
After=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/rpyc_classic.py

[Install]
WantedBy=multi-user.target" > rpyc-classic.service
```

Después se copian las siguientes líneas, una por una. Dado que en el inicio se usa el comando **sudo**, se pedirá un **password**, que es **maker**.

```
sudo cp rpyc-classic.service /lib/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable rpyc-classic.service
sudo systemctl start rpyc-classic.service
```

[8] Código con referencia a [13]

Por lo general después de ejecutar las líneas no se muestra algún resultado, pero puede demorarse unos segundos en ejecutar cada comando. La siguiente figura 73 muestra el proceso.

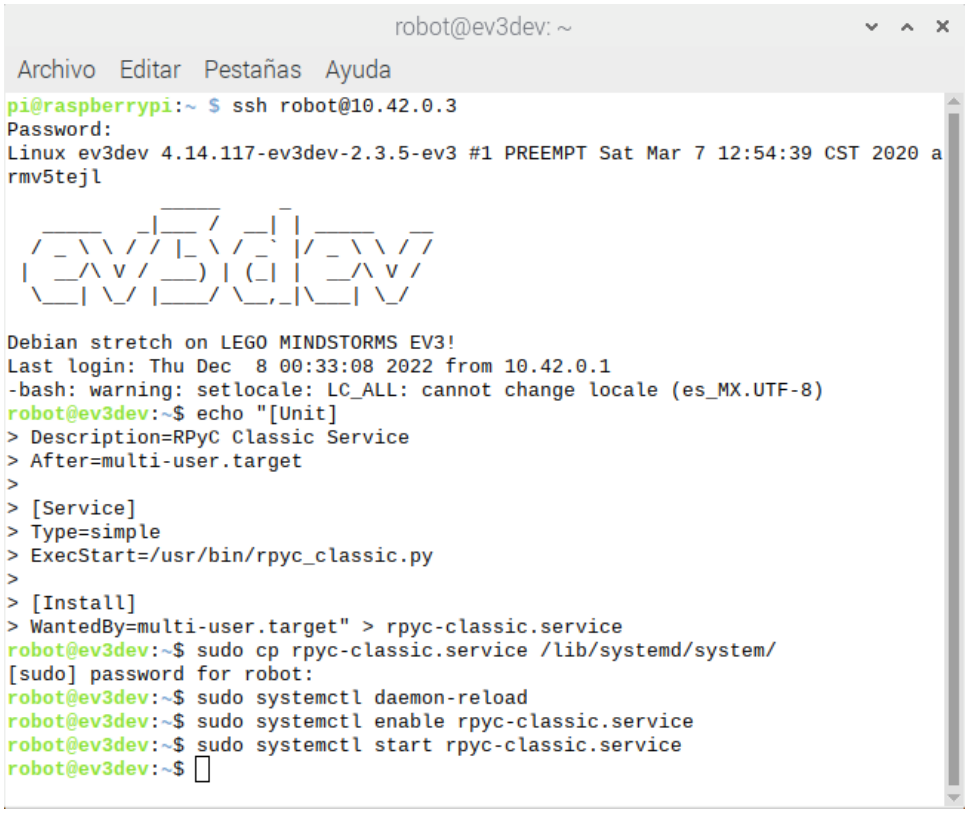


Figura 73. Resultados obtenidos en terminal de Raspberry Pi al iniciar el servicio de RPyC en el Brick a través de la conexión ssh.

Después de eso, se dirige a la terminal donde se creó el script y ejecutamos el siguiente comando:

```
python mov_motor.py
```

Se espera un momento y lo primero que pasara es que el motor largo dará un giro de 1 segundo y después, por la bocina, se escucha la frase que se programó.

Figura 74. Resultado tras ejecutar el script desde la Raspberry Pi para dar órdenes al Brick del EV3.

Con esto se da por terminado lo necesario para realizar el control remoto del robot EV3 desde la Raspberry Pi a través de la ejecución de la programación de un script.

Sección H

Configuración y pruebas con los módulos Xbee Serie 1

En este anexo se realiza un ejemplo básico sobre el uso y configuración de los módulos Xbee. Se realiza la comunicación entre dos computadoras, una de ellas es la Raspberry Pi 4 modelo B (SO Debian versión Buster) y la otra es una PC de escritorio (SO Ubuntu 20.04). La comunicación será por medio de módulos Xbee que estarán conectados a los equipos con ayuda de un shield o adaptadores [14].

Los materiales que se necesitan son: un cable de comunicación de USB a USB mini, tener instalado en la PC de escritorio o Laptop el software XCTU de Digi y VNC Viewer, dos módulos Xbee Serie 1, un shield para Raspberry y un adaptador USB, ambos para conectar los módulos. Estos dos últimos se muestran en las siguientes figuras.

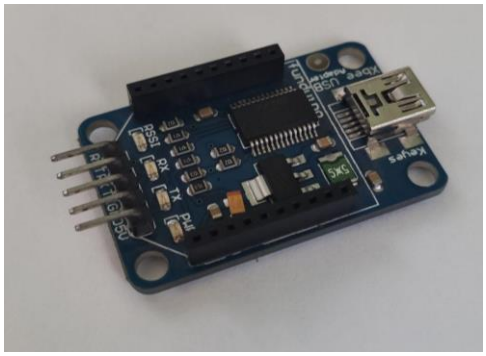


Figura 75. Adaptador USB para Xbee.

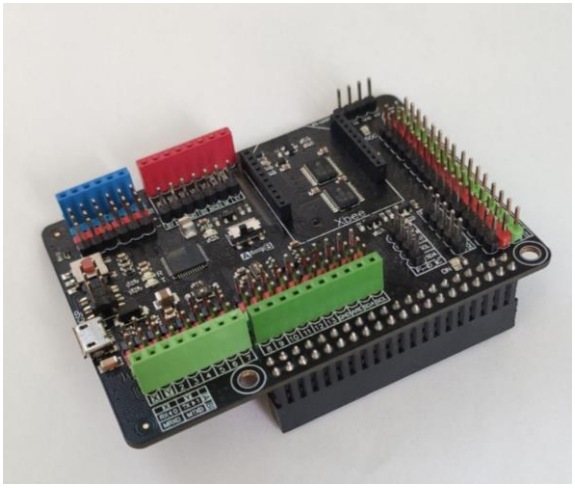


Figura 76. Shield para Raspberry Pi 4 con adaptador para Xbee.

Dado que la shield para la Raspberry se conectará por la GPIO[9], se tiene que configurar el puerto serie de la Raspberry, así como realizar otras configuraciones [15], [16].

[9] GPIO de las siglas General Purpose Input/Output, son entradas y salidas de propósito general. Por lo regular son pines que se configuran para que realicen ciertas funciones y el mismo usuario los configura, los habilita o deshabilita.

Se enciende la Raspberry Pi 4 y se conecta en sus periféricos correspondientes una pantalla, un teclado y un mouse para realizar las configuraciones pertinentes. Otra forma de realizar las configuraciones es por vía remota, ya sea por ssh o con ayuda del software VNC Viewer, para ambas opciones se requiere de una PC o Laptop. En este caso se va a configurar la Raspberry vía remota por medio de ssh, por este método lo único que se requiere es que la Raspberry esté conectada a internet, ya sea por medio de cable Ethernet o por wifi. Es importante recordar que la Raspberry y el equipo de cómputo estén en la misma red.

Conectamos la alimentación y esperamos que encienda, previamente la Raspberry se encuentra configurada para conectarse a la red por wifi. Por otro lado, en la PC de escritorio se abre una terminal nueva y se ejecuta el código siguiente.

```
ssh pi@192.168.1.80
```

Tras ejecutarlo, se pide escribir la contraseña, por default es "**raspberry**", enseguida se despliegan textos de resultados para realizar la conexión vía ssh, la conexión remota estará lista cuando en la terminal veamos `pi@raspberrypi:~$.`

Ahora se procede a configurar el puerto serie de la Raspberry, para ello en la terminal que se tiene abierta se ejecuta la siguiente línea

```
sudo raspi-config
```

Como resultado se muestra una ventana de menú de configuraciones de la Raspberry, figura 77. Con los cursores se elige la opción de "**Interface Options**" y se presiona la tecla enter.

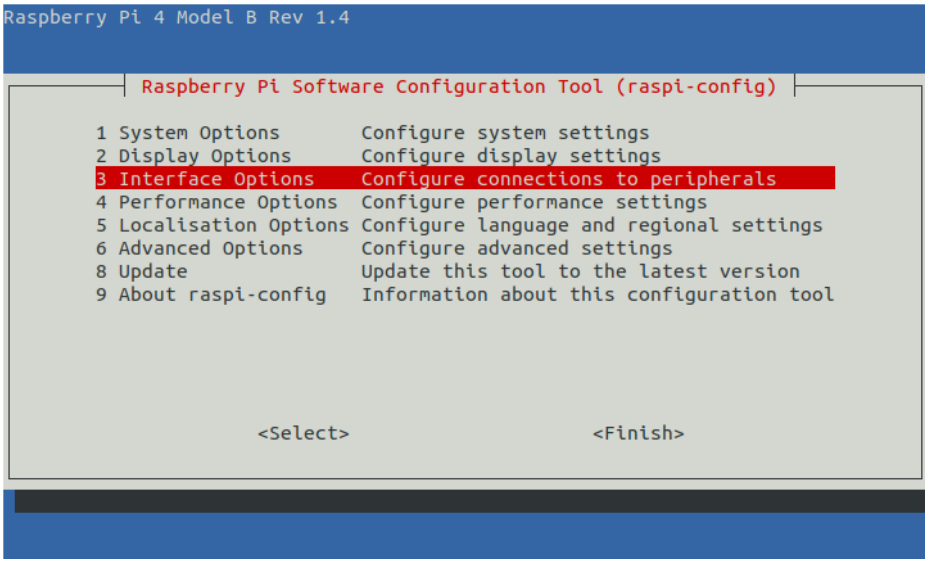


Figura 77. Menú de herramientas de configuración de Raspberry Pi.

A continuación, se muestra otro menú en que se selecciona la opción de "**Serial Port**", seguidamente se pregunta si se desea iniciar sesión a través de serial, se selecciona la opción que "**NO**". La siguiente ventana pregunta si se desea habilitar el hardware del puerto serial, se elige la opción "**SI**". Ya como último paso se muestra una ventana confirmando las configuraciones que se realizaron y se da la opción aceptar, figura 78.

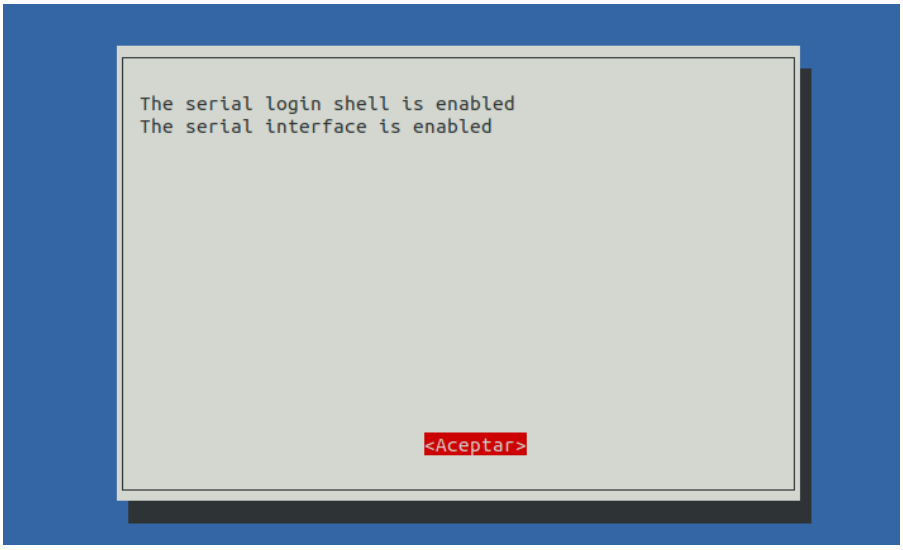


Figura 78. Confirmación de la interfaz serial habilitada en la Raspberry Pi.

Después se regresa al menú principal, se elige la opción **Finish**. Puede que salga una ventana en la que se pregunte si se desea reiniciar ahora, se selecciona la opción de "**SI**". Es recomendable que se reinicie la Raspberry para que se complete la configuración. En dado caso que no salga esa ventana y solo se muestre la terminal que se tiene abierta, se hace lo siguiente para realizar el reinicio en la terminal que se tiene abierta en la PC se escribe la siguiente línea.

```
sudo reboot
```

En la terminal se mostrará un mensaje que dirá que se cerró la conexión remota ssh.

Se cierra la terminal se esperan unos dos minutos para que encienda las Raspberry y ahora se ejecuta el programa VNC Viewer en el equipo de cómputo, se escribe la dirección IP de la Raspberry (192.168.1.80) y en seguida aparece una ventana para iniciar sesión, la contraseña es Raspberry. Se espera a que se abra una ventana

que muestra el escritorio de las Raspberry. Ahora se realizará una configuración con el módulo de bluetooth módem y se habilitará el puerto PL011 UART de la Raspberry. Para ello, primero se abre una terminal nueva en la Raspberry y se ejecuta la siguiente línea.

```
sudo gedit /boot/config.txt
```

Tras ser ejecutado se abre un editor de texto con las configuraciones de arranque, donde se habilita el puerto **PL011UART**. Esto se hace agregando la siguiente línea hasta la sección final que dice `[all]` y antes de la línea `enable_uart=1`

```
dtoverlay=disable-bt
```

Se guardan los cambios y se cierra el editor de textos. Por último, en la terminal que se encuentra abierta se ejecuta el comando

```
sudo systemctl disable hciuart
```

Esto es para asegurarse que efectivamente se ha deshabilitado el módulo de Bluetooth. Para este caso se apaga la Raspberry se desconecta de la alimentación y se conecta la shield.

Una vez realizado las configuraciones es momento de configurar los módulos Xbee. Para ello se usa el Xbee USB adapter, en el que se coloca el módulo Xbee y después se conecta el cable USB micro a USB al adaptador y después a un puerto USB de la PC de escritorio, figura 79. Seguidamente en el equipo de cómputo se abre el software XCTU el cual debe de estar previamente instalado, se abre una ventana como la que sigue, figura 80.

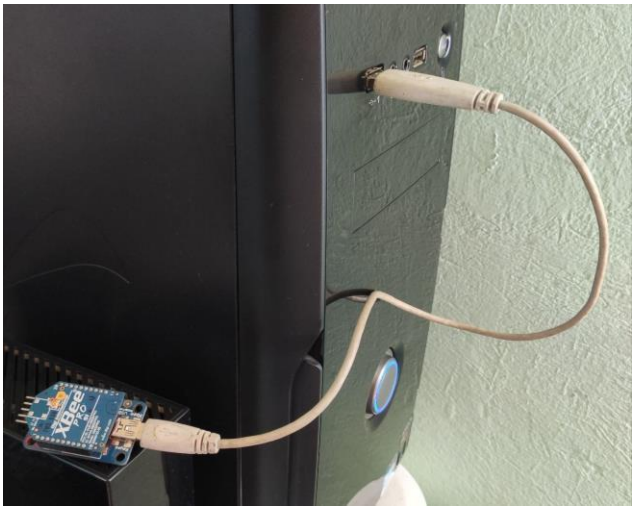


Figura 79. Conexión del adaptador USB para Xbee a un puerto USB de la PC.

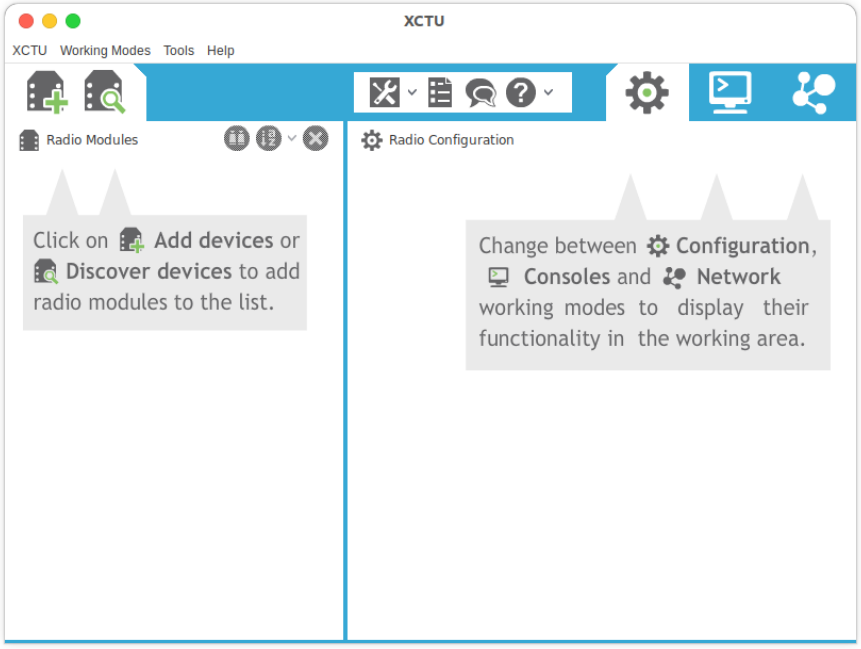


Figura 80. Ventana principal del software XCTU.

Después se presiona el icono  , este es para buscar módulos Xbee que se encuentren conectados a la PC, en seguida sale una ventana que indica el puerto

serial al que se encuentra conectado el módulo con la shield o adaptador a la PC, observar la figura 81.

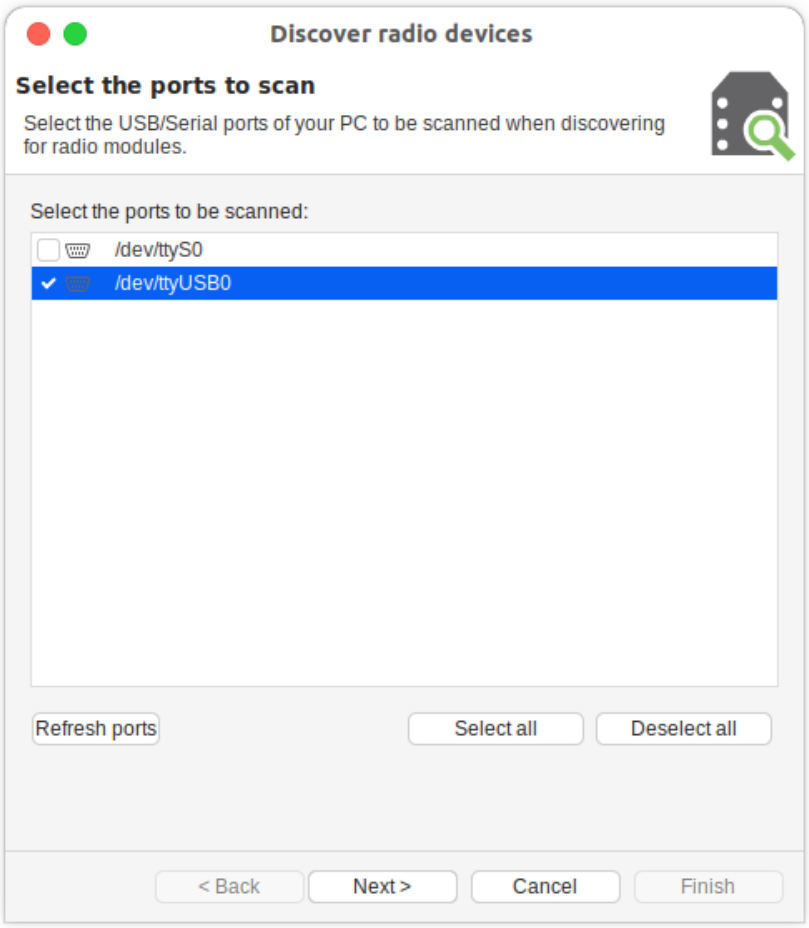


Figura 81. Venta emergente después de elegir la opción buscar módulos.

Para este caso se elige la opción de **/dev/ttyUSB0** y después la opción **Next**. A continuación, salen opciones de parámetros, no se mueve nada y se selecciona en **finish** que está en la parte inferior de la ventana. Seguidamente sale una ventana que muestra los módulos encontrados y se da clic en la opción de **Add select devices**, figura 82.

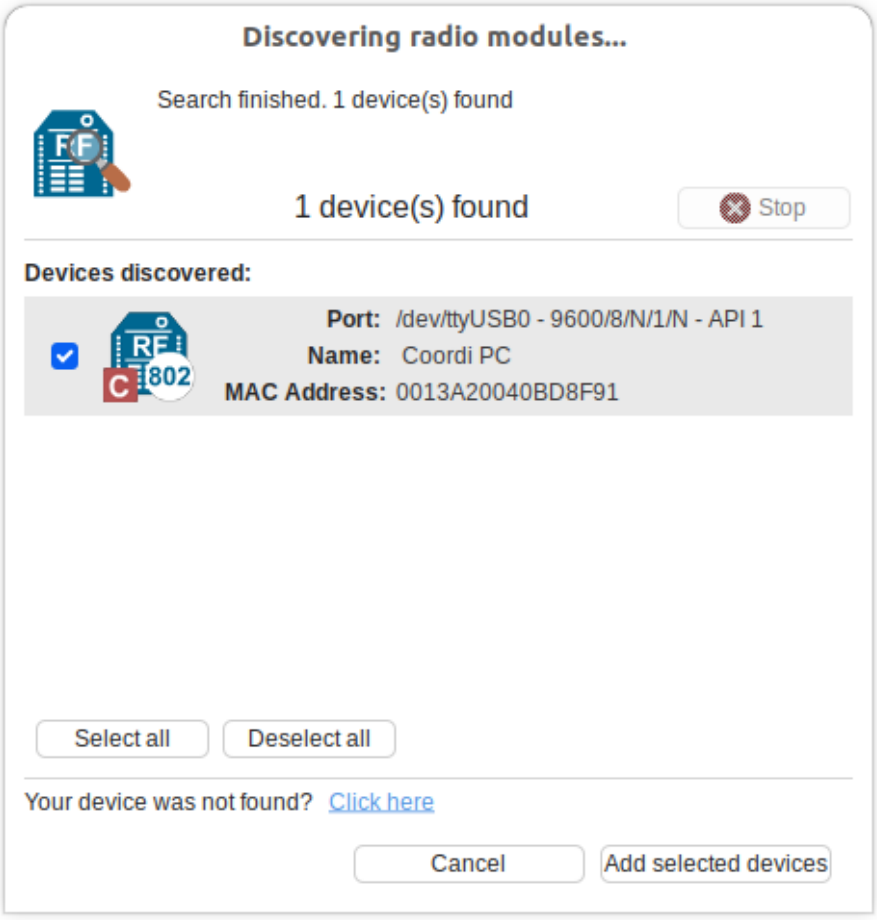


Figura 82. Ventana que muestra los módulos encontrados y conectados a al PC.

Ahora se observa en el lado izquierdo de la ventana en el panel de Radio Modules el módulo Xbee que tenemos conectado a la PC. Cuando se selecciona el módulo que está conectado, en el panel derecho de la ventana aparecen las configuraciones disponibles para el módulo Xbee.

Antes de comenzar a modificar las configuraciones, es necesario tomar los datos que se encuentran debajo de cada módulo Xbee, estos se van a requerir para este ejemplo, figuras 83 y 84.

Figura 83. Módulo A Xbee Serie 1 Pro.

Figura 84. Módulo B Xbee Serie 1 Pro.

Es importante saber que se va a crear una pequeña red, que estará conformada por dos módulos Xbees, en el que uno será un **coordinador** y él otro un **end device**. Continuando con las configuraciones, este primer módulo que ya se encuentra conectado a la PC y está identificado en XCTU será el coordinador, por lo tanto, los valores de configuraciones que se modificarán serán los siguientes.

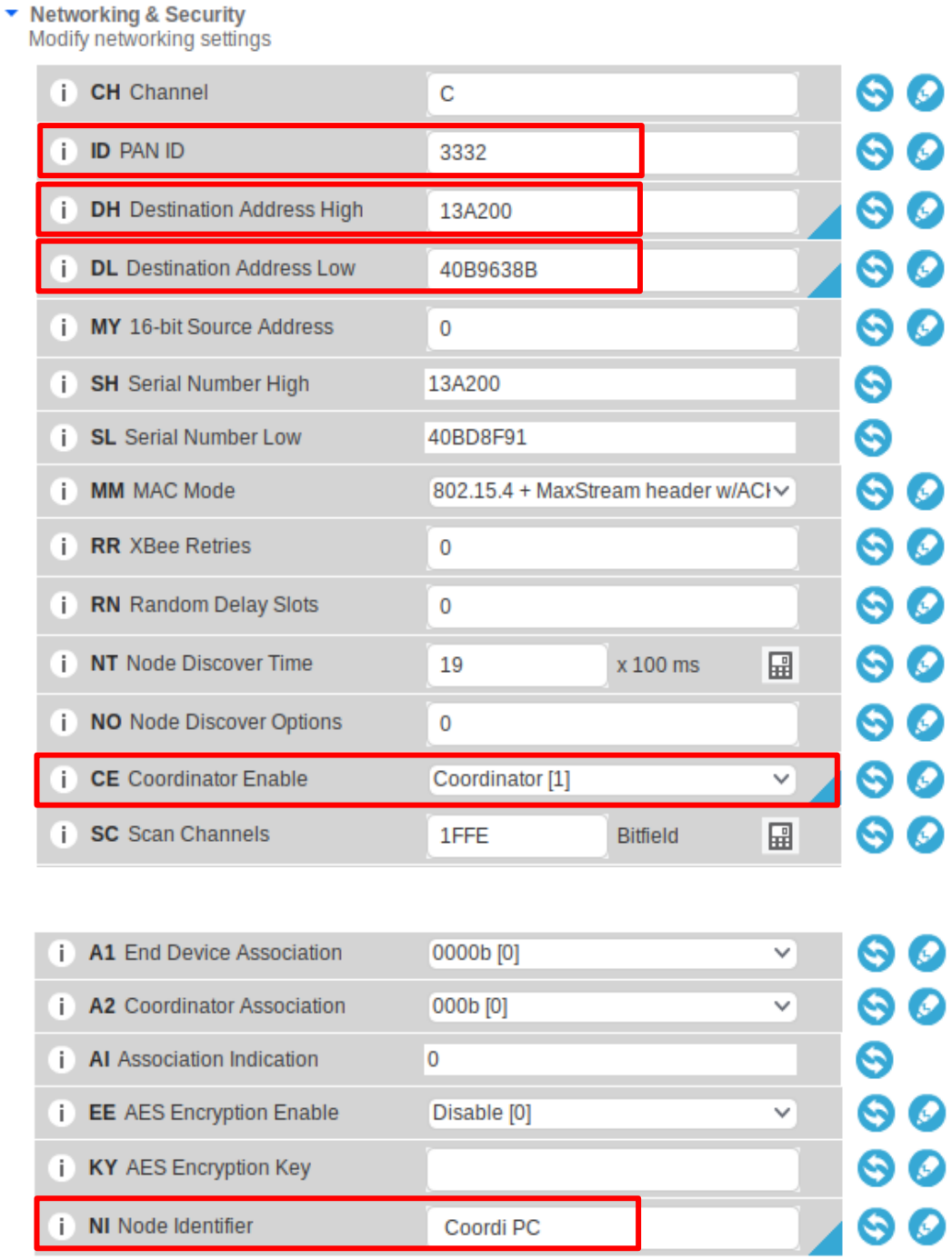


Figura 85. Panel de configuraciones Networking & Security para el módulo Xbee Coordinador.

Después se busca la sección de Serial Interfacing y se realizan las siguientes modificaciones.

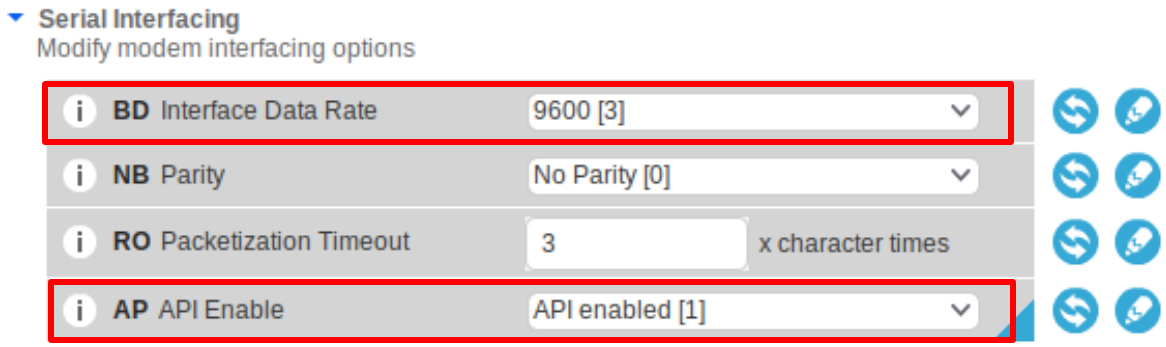


Figura 86. Panel de configuraciones Serial Interfacing para el módulo Xbee.

Una vez realizadas las configuraciones buscamos el icono de write  , esperamos a que se guarden los cambios. Después cerramos el programa, desconectamos el cable USB que está conectado con el adaptador del módulo Xbee. Retiramos el módulo y colocamos el otro módulo en el adaptador para configurarlo. Nuevamente se conecta el cable, se abre el software XCTU y se busca el módulo conectado y después se agrega. Para este caso, este módulo será el End Device y se realizan las siguientes configuraciones:

- ✓ **ID**: Se asigna el valor por default 3332
- ✓ **DH**: Se asigna el valor de SH del módulo que es 13A200
- ✓ **DL**: Se asigna el valor de SL del otro módulo que es 40BD8F91
- ✓ **CE**: Se elige la opción de **End Device**.
- ✓ **NI**: Se denomina el nombre del módulo en este caso Rasp EndDev

Después se busca la sección de Serial Interfacing y se realizan las siguientes modificaciones.

- ✓ **BD**: Se asigna el valor de 9600
- ✓ **AP**: Se elige la opción de API **disable**

Se guardan los cambios como se hizo con el primer módulo, se espera a que termine el proceso y se cierra la ventana de XCTU. Nuevamente se desconecta el cable USB de la PC que va al adaptador. Se coloca el módulo Coordi PC en el adaptador y el módulo Rasp EndDev en la shield que a su vez estará conectada en la Raspberry, figura 87.

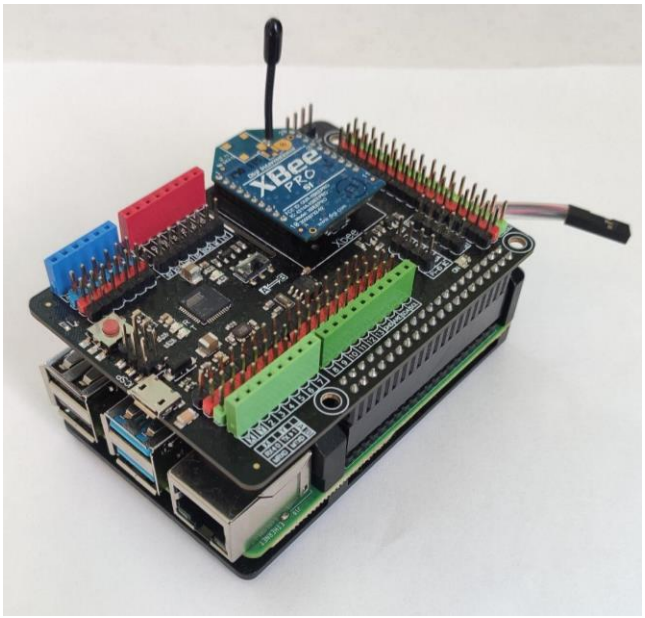


Figura 87. Módulo Xbee montado en la shield y a su vez conectado en las Raspberry Pi 4 modelo B.

Se enciende la Raspberry, y se procede a descargar e instalar la herramienta minicom[10]. Este programa ayuda a enviar y recibir información por el puerto serie.

[10] Programa que se encarga de la comunicación con otros dispositivos por medio del puerto serie. Esta herramienta se encuentra disponible de manera gratuita y se ejecuta bajo Linux.

Para comenzar con la descarga e instalación, se coloca la siguiente línea en terminal.

```
sudo apt-get install minicom
```

Después de ejecutar la anterior línea, sale una línea que pregunta si se desea continuar, se presiona la tecla **S** para continuar con la instalación. Una vez instalado el programa ahora se pasa a la PC de escritorio, se conecta el Xbee Coordi PC con el adaptador por medio del cable USB al equipo de cómputo. Se ejecuta el software XCTU y se buscan los módulos conectados, se elige el módulo Coordi PC, después elige el módulo que parece en el panel izquierdo y en la parte superior se oprime la opción con el icono  , esta opción ayuda a trabajar en modo consola y así enviar información con otros módulos. Una vez seleccionada esta opción ahora en el panel derecho aparece algo como lo que se muestra en la figura 88.

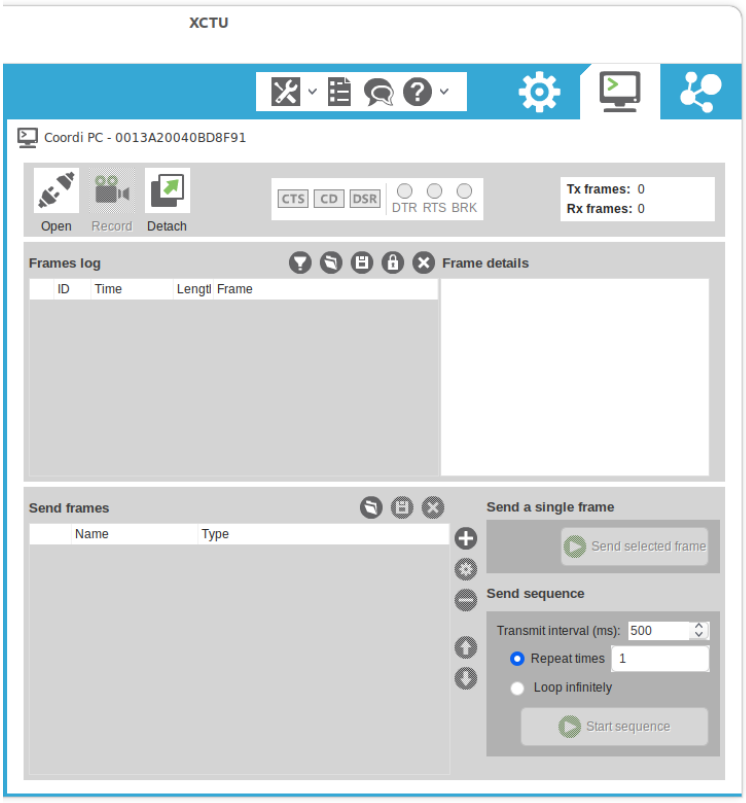


Figura 88. Panel de modo consola de XCTU.

Como siguiente paso se oprime el icono de open  para comenzar con la comunicación.

Después, en la Raspberry se abre una terminal nueva y se ejecuta el siguiente código.

```
minicom -b 9600 – o - D /dev/ttyAMA0
```

Como resultado se despliega un mensaje de bienvenida al programa minicom e informando el puerto de comunicación. Ahora se regresa nuevamente a la PC de escritorio en el programa XCTU en el apartado de terminal y en la zona de Console log se escribe el mensaje de “Saludos desde la PC”, figura 89. Si ahora se observa

la terminal de la Raspberry se puede apreciar que se recibió el mensaje, observar la figura 90.

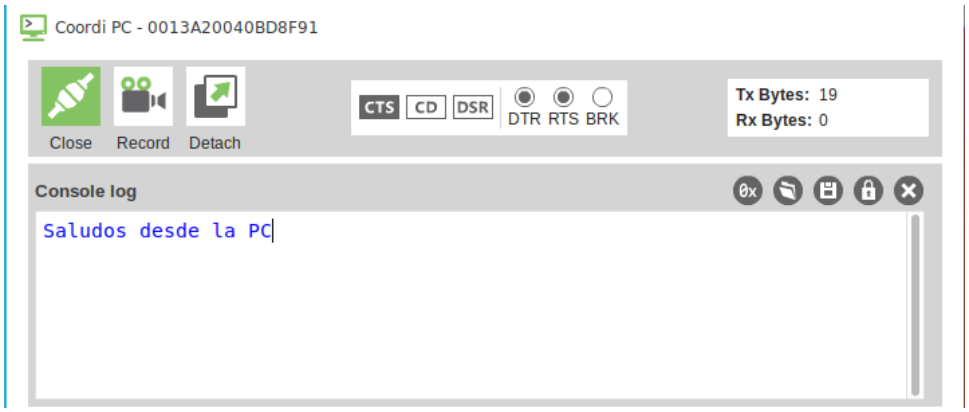


Figura 89. Panel de consola de XCTU del módulo Coordinador. Mandando mensaje a la Raspberry Pi.

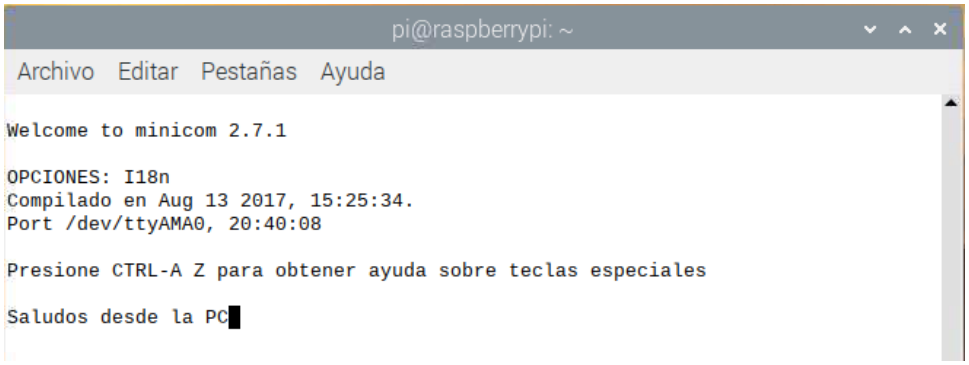


Figura 90. Terminal de programa minicom ejecutada desde la Raspberry Pi. Recibiendo mensaje de la PC.

Por otra parte, en la terminal de la Raspberry comenzamos a escribir el mensaje de “Saludos desde la raspberry”, cabe destacar que en esta terminal no se verá lo que se escribe, pero si se regresa al programa XCTU en la PC de escritorio se podrá observar que se recibe el mensaje de la Raspberry, figuras 91 y 92.

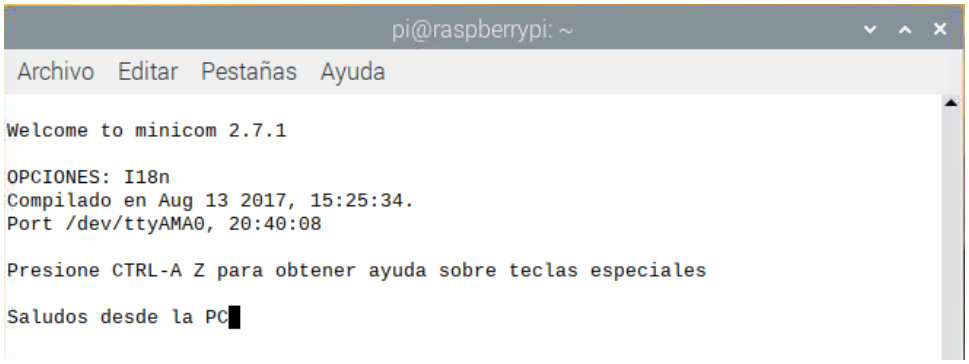


Figura 91. Terminal de programa minicom ejecutada desde la Raspberry Pi. Escribiendo mensaje para la PC.

Figura 92. Panel de consola de XCTU del módulo Coordinador. Recibiendo mensaje de la Raspberry Pi.

Se observa que las configuraciones fueron correctas dado que existe comunicación entre los dos módulos Xbee y, por lo tanto, se pueden realizar más ejercicios de aplicaciones con la Raspberry Pi y el módulo Xbee.

Sección I

Configuración de NodeMCU con ESP8266 para trabajar con ROS

En la presente sección se describen a detalle las configuraciones pertinentes, así como las librerías necesarias para comenzar a usar y programar el NodeMCU de manera sencilla, así como para trabajar con ROS en su versión Noetic [17].

Los materiales que se necesitan son: un módulo NodeMCU, cable de USB a USB micro y el software de Arduino IDE instalado en la computadora. Antes de comenzar a programar, se recomienda descargar los controladores de la tarjeta NodeMCU para que lo identifique la computadora, en el caso de Linux en su distribución Ubuntu en su versión 20.04 no fue necesario, la NodeMCU fue detectada sin problema.

Por la parte de Arduino IDE, se tienen que descargar librerías que ayudan a que el software reconozca el NodeMCU. Para ello, se abre Arduino IDE luego se copia la siguiente dirección en la sección para actualización de librerías:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

En esta dirección se encuentran las librerías para que reconozca Arduino IDE al

microcontrolador ESP8266 ya sea en su versión NodeMCU u otras. Continuando con el software de Arduino, se dirige a la pestaña **Archivo**, en la lista que se despliega se da clic en **Preferencias**, se abre una ventana nueva y se dirige a la opción que dice **Gestor de URLs Adicionales de Tarjetas**, figura 93. Junto a esa opción se encuentra un espacio que puede estar en blanco, ahí se pega la dirección URL que se copió. Si ya existe una dirección URL en ese espacio, simplemente se coloca una coma al final de la URL existente, se da un espacio y se pega la URL que se copió. Al final se da clic en la opción **OK** para guardar los cambios.

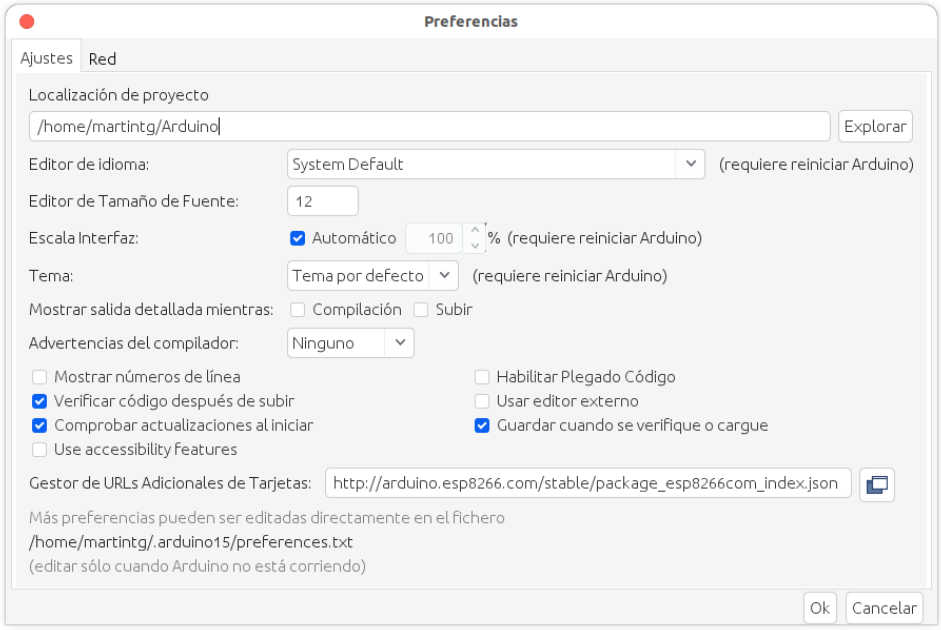


Figura 93. Ventana de preferencias del software Arduino IDE.

Estando en Arduino IDE ahora se elige la pestaña de **Herramientas**, sale un menú de opciones, se da clic en la que dice **Placa** y después se da clic a la opción de **Gestor de tarjetas**. En seguida se abre una ventana nueva, este es el gestor para instalar librerías de dispositivos para que se pueda programar con Arduino IDE. En

la ventana se debe de ubicar la sección de **búsqueda**, ahí se escribe **ESP8266** y en la parte de resultados se elige y se instala la librería que lleva por nombre **esp8266 by ESP8266 Community**, figura 94. Una vez que termine la instalación se cierra la ventana.

Figura 94. Gestor de tarjetas donde muestra la búsqueda para ESP8266.

Para comprobar que efectivamente se instalaron las librerías de los modelos de tarjetas ESP8266, se realiza un ejemplo. Primero se va a la pestaña de **Herramientas** de Arduino IDE, se elige la opción de **Placa**, sale un pequeño menú del cual se elige la que dice **ESP8266 Boards Arduino**, inmediatamente sale un listado, se opta por **NodeMCU 1.0 (ESP-12E Module)**. Habiendo hecho todo lo anterior ya es momento de conectar el NodeMCU con el cable USB a un puerto USB de la computadora. De nueva cuenta se elige la pestaña de **Herramientas** y ahora se dirige a la opción de **Puerto**, se selecciona el puerto COM que le fue asignado por la computadora a la placa.

Se copia en el espacio de trabajo de Arduino IDE el siguiente código[11], que hace uso de un led interno (LED_BUILTIN), este led se encuentra definido como una función en Arduino IDE como pinMode, para este caso el led se encuentra en el pin 2 de la placa y se define como OUTPUT. Dicho programa se encarga de encender por un segundo el led y en el siguiente segundo se apaga. Este mismo ejemplo se puede realizar haciendo uso de un led externo, es importante definir en que salida se encontrará conectado este led.

```
void setup() {
// inicializa el pin digital 2 como salida
pinMode(2, OUTPUT);
}
void loop() {
digitalWrite(2, HIGH); // Enciende el LED
delay(1000); // Espera un segundo
digitalWrite(2, LOW); // Apaga el LED
delay(1000); // Espera un segundo
}
```

Antes de subir el programa a la placa, el programa se debe de comprobar, para ello se oprime el botón de verificar  y si no existe error ahora sí se sube el programa a la tarjeta, se presiona el botón subir  . Cabe mencionar que si el programa no se puede cargar a la tarjeta esto se puede solucionar presionando el botón **FLASH** de la tarjeta. Este botón se oprime en el momento en que se esté subiendo el programa y se deja de presionar justo cuando se muestre en el Arduino IDE que se está cargando el programa. Se cierra Arduino IDE y se observa cómo parpadea el

[11] Código con referencia a [17]

led integrado de la tarjeta.

Una vez que se tiene configurado el Arduino IDE y se sabe programar la tarjeta NodeMCU, es momento de comenzar a trabajar con la librería de ROS que es compatible con esta tarjeta [18], [19].

Para este ejemplo se necesita instalar **rosserial**, lo cual se hará vía **git**, toda la instalación se realiza por medio de terminal. Como primer paso abrir una terminal nueva y ejecutar el siguiente comando para instalar **rosserial** para ROS Noetic:

```
sudo apt-get install ros-noetic-rosserial
```

Después, en la misma terminal se dirige al directorio del espacio de trabajo que, en este caso, es la siguiente dirección: **catkin_et/src**. Ya estando en el interior de la carpeta **src**, se ejecuta la siguiente línea de comando:

```
git clone https://github.com/ros-drivers/rosserial.git
```

La anterior línea lo que hace es clonar de un repositorio todos los archivos y descargarlos en el interior del directorio en el que se encuentra. Una vez que se termine el proceso de clonación y descarga se procede a ejecutar los siguientes comandos:

```
cd ..
```

```
catkin_make
```

El primer comando es para regresar a la carpeta **catkin_et** que es el espacio de trabajo, el segundo comando es para compilar nuevamente el contenido del espacio de trabajo, ya que en este caso se ha agregado **rosserial**, se espera a que termine el proceso de compilación.

Otro elemento importante, es tener instalado en Arduino IDE las librerías de ROS,

este proceso se va a realizar por medio de la terminal. Se cierra la terminal que se tiene abierta y se abre una nueva. Ahora se dirige a la ruta **~/Arduino/libraries/** y se ejecuta el siguiente comando para borrar librerías antiguas de **ros_lib**:

```
rm -rf ros_lib
```

Después se ejecuta en terminal el comando: **cd**, este comando nos manda a la carpeta raíz. Ahora se ejecuta la siguiente línea para instalar la librería de ROS en Arduino IDE:

```
rosrun rosserial_arduino make_libraries.py Arduino/libraries/
```

Se espera a que se termine el proceso. Para corroborar que las librerías se instalaron bien se abre Arduino IDE y se dirige a la pestaña de **Archivo**, se despliega un menú y de ahí se opta por la opción de **Ejemplos**, enseguida se despliega otro menú y se busca la sección de **Ejemplos de Librerías Personalizadas**, aquí se debe de encontrar la librería que se instaló de ROS, llamada **ros_lib**. Ya estando ahí, se elige esa librería y seguidamente aparece otro menú de los ejemplos que tiene **ros_lib**, para este caso se elige la que se llama **Esp8266HelloWorld**, se espera unos segundos y se abrirá una ventana nueva con el código del programa que se cargará al NodeMCU. Este programa define al NodeMCU como un nodo de ROS que publicará un mensaje de “hello world!”, este nodo se comunica con una computadora en la cual se está ejecutando roscore. Cabe destacar que tanto el módulo NodeMCU como la computadora deben estar en la misma red de datos.

El programa consta de 4 secciones. En la primera se declaran las librerías que se ocuparan, en este caso: **ESP8266.h**, **ros.h** y **std_msgs/String.h**. Para la segunda sección se pide un identificador de red (**ssid)** y una contraseña (**password)**, para este caso tendremos que poner los valores de nuestro modem, esto es para que el NodeMCU se encuentre en la misma red y se comunique con la computadora que ejecuta ROS. Así mismo, en esta sección se define una **IP Adress,** que para este ejemplo se debe de colocar la IP de la computadora que tiene ROS. A continuación, se crea un objeto **Nodehandle** que es para realizar la comunicación con ROS, después se define el tipo de mensaje ROS a publicar que será de tipo cadena de caracteres (**String)**, seguidamente se crea el tema para el publicador, el cual tendrá el nombre de **"chatter"** y el tipo de mensaje será **str_msg**. Como última definición se declara un arreglo de tipo carácter (**char)** para guardar el mensaje de **"hello world!"**.

En la tercera sección se encuentra una función vacía **void setup,** la cual hará uso del **monitor serie de Arduino IDE**, por lo tanto, acá se configura la velocidad de comunicación con el monitor serie que es de **115200 baudios**, el proceso de conexión de la tarjeta con el modem, y los mensajes que se desplegaran conforme el NodeMCU se conecta al modem. Después se declaran funciones para que se realice la conexión con el **servidor rosserial** y aquí mismo se inicializa el objeto **Nodehandle**.

Como última sección está la función de ciclo **loop** que hará el proceso de publicación del mensaje por periodos de tiempo. Primero se define una función condicional

en el que mientras el objeto **Nodelhandle** se encuentre **conectado** se imprimirá en el monitor serie **“Connected”** y se publicará el mensaje de **“hello world!”**, de lo contrario no se publicará nada y en el monitor serie se imprimirá el mensaje de **“Not Connected”**. Después, se define la frecuencia en la que se estará publicando el mensaje que será de **1 Hz**.

Una vez que se explicó a grandes rasgos el código y se colocaron los valores de **your-ssid**, **your-password** e **IPAddress server** se procede a cargar el programa a la tarjeta, se hará de la misma forma que en el ejemplo anterior. Tras concluir la carga del programa en la tarjeta, en la computadora donde se tiene instalado ROS se abre una terminal nueva y se ejecuta el **roscore** y después se abre otra terminal nueva y se ejecuta la siguiente línea:

```
rosrun rosserial_python serial_node.py tcp
```

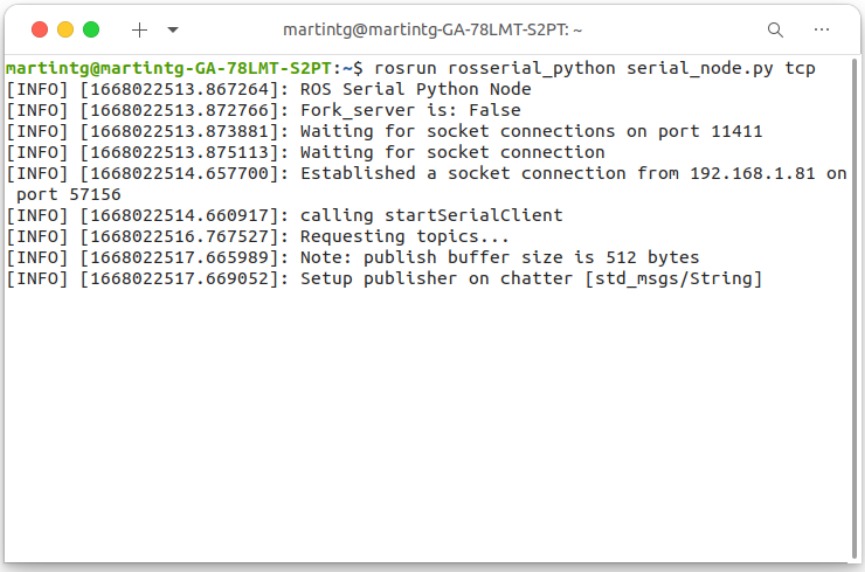


Figura 95. Iniciando desde la terminal el nodo serial.

La anterior línea es para iniciar el nodo **rosserial** y exista la comunicación con el NodeMCU. Después se dirige al software de Arduino IDE y se presiona el botón de  para abrir el monitor serie, una vez que se presiona se abre una ventana como la que se muestra en la figura 96.

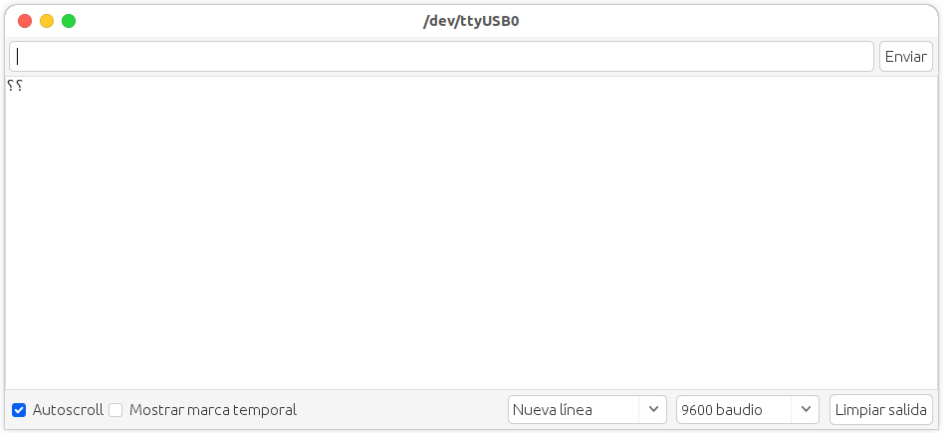


Figura 96. Visualización de monitor serie de Arduino IDE.

Se observa que en la parte inferior derecha se encuentra el valor de **9600 baudios**, se selecciona la pestaña y se elige el valor de **115200 baudios**, este valor es el que se especifica en el código para la comunicación serie. Después de hacer ese cambio inmediatamente se muestra lo siguiente en el monitor serie, figura 97.

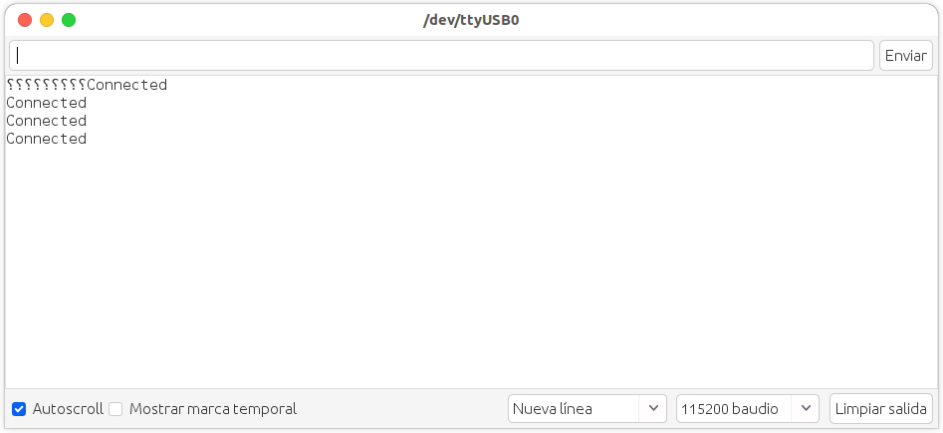


Figura 97. Ventana monitor serie donde se muestra el mensaje de conectado.

Se observa que efectivamente se conectó el módulo NodeMCU a la computadora que está ejecutando el nodo maestro roscore. Por otro lado, para corroborar que se está publicando el mensaje, se abre una nueva terminal y se ejecuta el comando:

```
rostopic echo chatter
```

Este comando es una función de ROS que ayuda a leer el contenido del topic que en este caso es “chatter” y que dentro lleva el mensaje de “hello world!”, esto se aprecia en la siguiente figura 98.

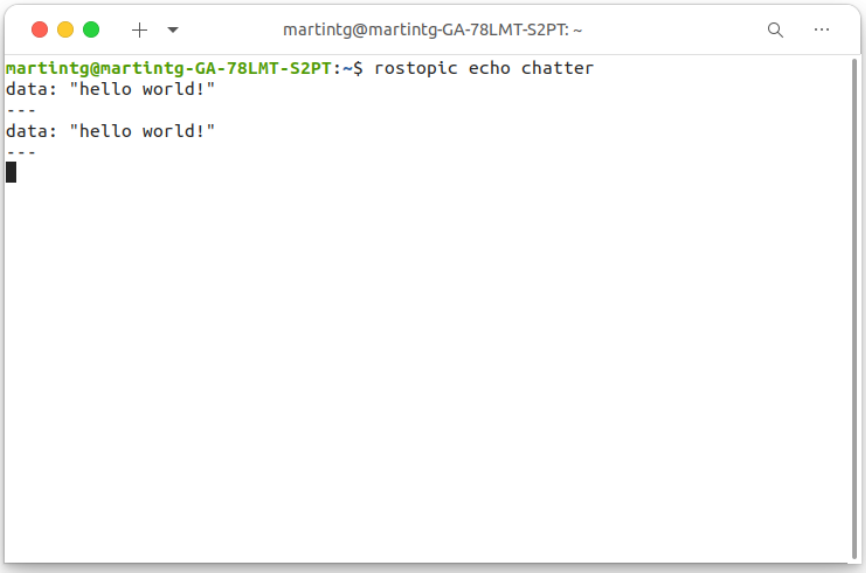


Figura 98. Se muestra el contenido del topic chatter con ayuda de la terminal.

Se observa que tras ejecutar el comando efectivamente se está publicando el mensaje que se programó en el módulo NodeMCU. Hasta aquí termina la configuración y uso de NodeMCU con ROS.

Sección J

Programando el NodeMCU como nodo subscriptor en ROS

Una vez que se corroboró la programación del NodeMCU como nodo publicador y su correcto funcionamiento, además de que a su vez éste es reconocido por ROS como un nodo más, es momento de programar al NodeMCU como un nodo suscriptor. Para ello, se usa el ejemplo del proyecto de contar números, en donde se creaban dos nodos, uno publicador y uno suscriptor; esto se vio en el **Anexo D**. Para este caso se pretende usar la Raspberry Pi que ejecutará ROS y al mismo tiempo se ejecuta el nodo publicador de números, mientras que el NodeMCU será el nodo suscriptor (quien estará recibiendo los mensajes). Como requisito importante es que tanto la Raspberry Pi como el NodeMCU deben de estar dentro de una misma red para que se logren comunicar, también se requiere un modem o un access point (AP) y contar con una PC que tenga instalado Arduino IDE, figura 99.

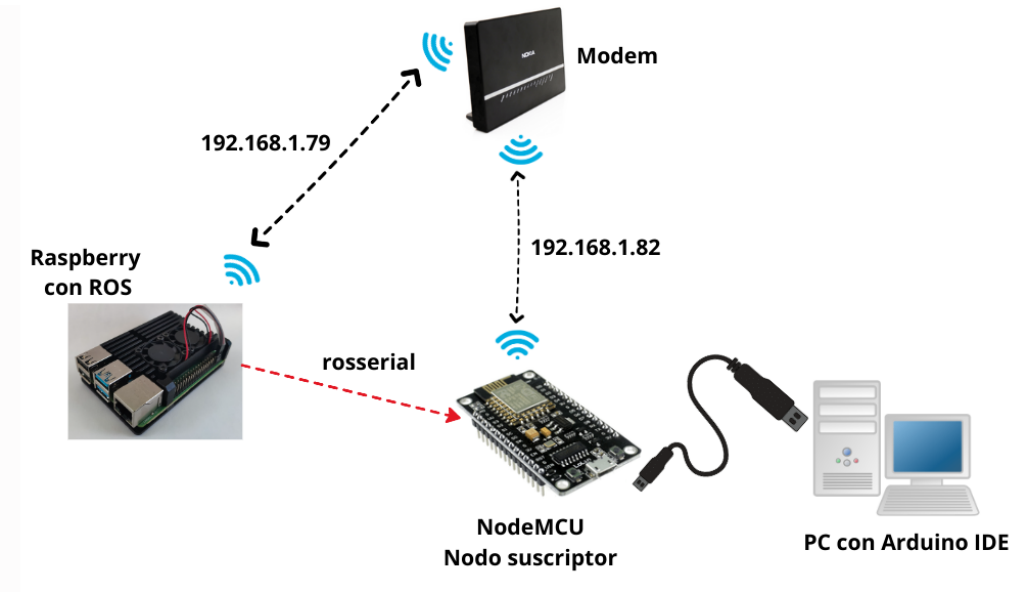


Figura 99. Esquema de conexión para realizar el ejemplo de NodeMCU como nodo suscriptor.

Como primer paso se realiza el programa para que el NodeMCU sea un nodo subscriptor. Se abre el software Arduino IDE para codificar el programa, compilarlo y después programarlo en el NodeMCU. A continuación, se muestra el cuerpo del programa, este está basado en la estructura del ejemplo de publicar el mensaje hello world![12]:

```
#include <ESP8266WiFi.h>

#define ROSSERIAL_ARDUINO_TCP
#include <ros.h>
#include "std_msgs/UInt32.h"

int numero;
const char* ssid     = "Nombre de la red WiFi";
const char* password = "contraseña de la red WiFi";

// Set the rosserial socket server IP address
IPAddress server(192,168,1,79);
```

[12] Programa con referencia a ejemplo “Esp8266HelloWorld” de Arduino IDE, ubicado en los ejemplos de la librería ros_lib.

```
// Set the rosserial socket server port
const uint16_t serverPort = 11411;

void callback(const std_msgs::UInt32& msg)
{
  numero = msg.data;
}

ros::NodeHandle nh;

ros::Subscriber<std_msgs::UInt32> sub("numero_pub", &callback);

void setup()
{
  // Use ESP8266 serial to monitor the process
  Serial.begin(115200);
  Serial.println();
  Serial.print("Conectando a ");
  Serial.println(ssid);

  // Connect the ESP8266 the the wifi AP
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi conectado");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Set the connection to rosserial socket server
  nh.getHardware()->setConnection(server, serverPort);
  nh.initNode();

  // Another way to get IP
  Serial.print("IP = ");
  Serial.println(nh.getHardware()->getLocalIP());

  // The subscriber node is started
  nh.subscribe(sub);
}

void loop()
{
```

```
  if (nh.connected()) {
    Serial.println("Conectado");
    //
    Serial.println("Escucho el número:");
    Serial.println(numero);
  } else {
    Serial.println("No Conectado");
  }
  nh.spinOnce();
  delay(1000);
}
```

Observando el código, se identifican cuatro secciones. En la primera sección se declaran las librerías a usar, para este caso se define el tipo de mensaje (std_msgs) de ROS de las cuales solo se van a usar los mensajes del tipo UInt32 (datos enteros de 32 bits sin signo).

En la segunda sección se declara una variable **numero** de tipo entero, en ésta se van a guardar los datos que se reciban del nodo publicador. También se definen los datos de la red a la que se va a conectar el nodeMCU, aquí se debe agregar el ssid y el password del modem o AP. Como se mostró en la figura 91, tanto la Raspberry Pi como el nodeMCU estarán conectados al modem o AP. De igual manera se especifica la IP del equipo de cómputo en el que se va a ejecutar el roscore, en este caso se coloca la dirección IP de la Raspberry Pi. Enseguida se comienza a definir el nodo suscriptor, para ello se escribe lo siguiente:

```
ros::Subscriber<std_msgs::UInt32> sub("numero_pub", &callback);
```

El suscriptor se estará suscribiendo a un topic llamado **numero_pub**, en el cual se envía un mensaje de tipo **UInt32**, el suscriptor hará uso de una función **callback**,

ésta se encarga de devolver el mensaje que obtuvo el nodo suscriptor por medio del topic **numero_pub**. Es importante mencionar que la función **callback** se declara antes de definir el nodo suscriptor y antes de la función **setup()**.

```
void callback(const std_msgs::UInt32& msg)
{
  numero = msg.data;
}
```

La función **callback** se define con la palabra reservada **void,** que indica que no devuelve ningún valor. Por otra parte, **callback** tiene como atributo la línea **const std_msgs::UInt32& msg** en el que va a recibir el mensaje del tipo **UInt32**. Dentro de la función **callback** se define que todo mensaje que se reciba se va a guardar en la variable **numero**.

En la tercera sección se realizan las configuraciones pertinentes para tener conexión con el monitor serie de Arduino IDE, así como las configuraciones para que el NodeMCU se conecte al modem y que a su vez mande mensaje al monitor serie de que se conectó correctamente. En seguida se realiza la conexión con el **servidor rosserial** y se inicializa el nodo suscriptor.

En la última sección se declara una función que dependerá del estado de conexión del NodeMCU con el **servidor rosserial**. Si está conectado, en el monitor serie se mostrarán mensajes de "**Conectado**, **Escucho el número"** y se imprime el contenido de la variable **numero**. Y si no se encuentra conectado el NodeMCU solo

se imprime el mensaje **No Conectado**. Todos estos mensajes se imprimen con una frecuencia de 1 segundo en el monitor serie del software Arduino IDE.

Una vez terminado de escribir el programa se verifica el código para que no exista algún error en la codificación. Ya que se corroboró que no hay errores, se conecta el NodeMCU y se procede a cargar el programa. Se abre el monitor serie de Arduino IDE, se verán algunos mensajes, pero para ver todo el proceso de conexión se oprime el botón físico RST de la NodeMCU.

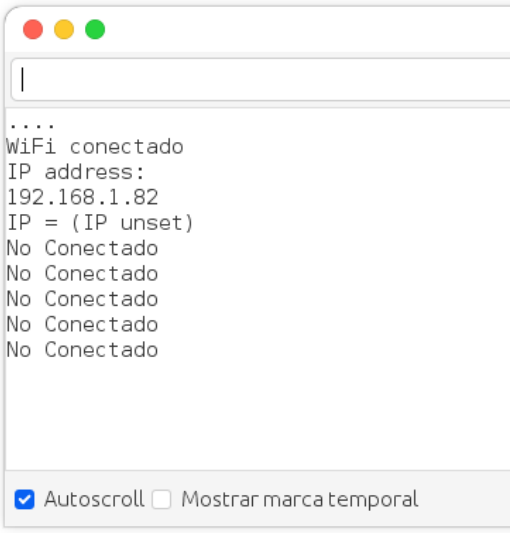


Figura 100. Ventana de monitor serie de Arduino IDE, mostrando resultado que el NodeMCU se conectó de manera exitosa a una red WiFi.

Como se observa en la figura 100, en el monitor serie se imprime el mensaje de **WiFi Conectado**, nos indica que se conectó correctamente al modem, los siguientes mensajes indican que el NodeMCU no se conectó con el **servidor rosserial**, por lo tanto, se imprime el mensaje de **No Conectado**.

Se enciende la Raspberry Pi, se abre una terminal nueva y se inicia el nodo maestro ejecutando el comando: **roscore**. Se abre una nueva terminal y se inicializa el **servidor rossserial**, para ello se ejecuta primero el comando: **`source ~/catkin_et/devel/setup.bash.`** Y luego se ejecuta el comando: **`rosrun rosserial_python serial_node.py tcp.`**

Ahora se abre nuevamente el monitor serie de Arduino IDE y se observa lo que se muestra en la figura 101.

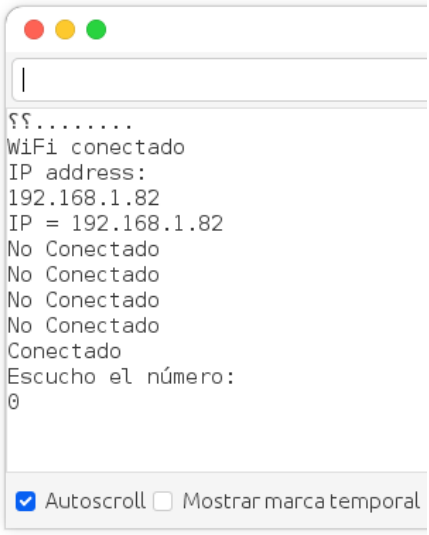
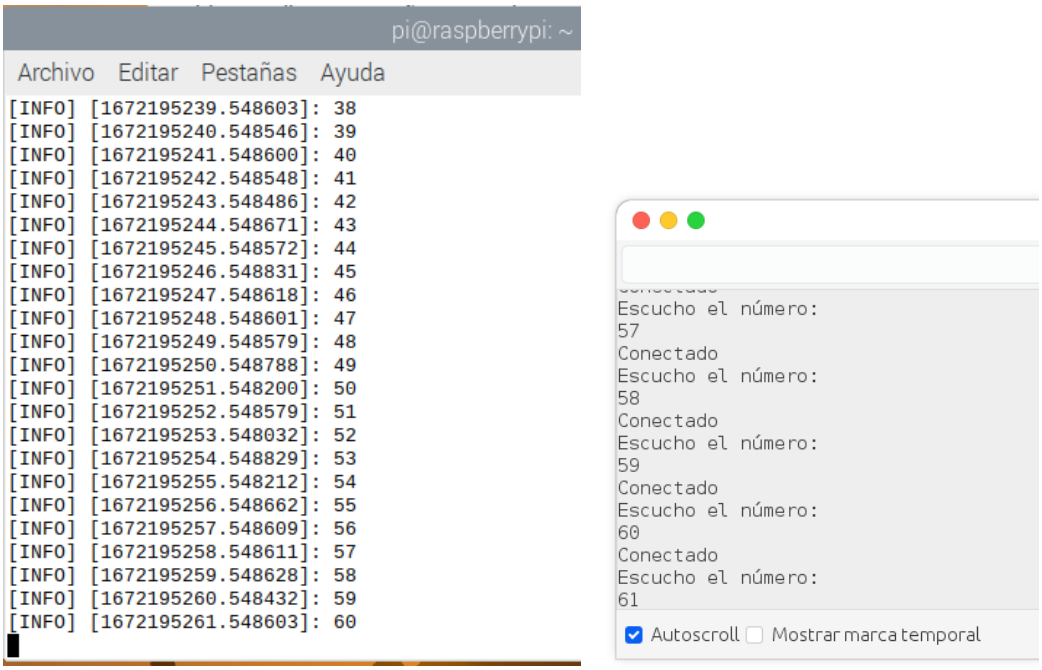


Figura 101. Monitor serie muestra resultado del NodeMCU tras iniciar el servidor rosserial.

Después de un momento al final se imprime el mensaje de Conectado y el mensaje de Escucho el número y número cero. Esto comprueba que el NodeMCU ya se conectó con el **servidor rosserial** y que el valor de nuestra variable **numero** es cero, después de unos segundos seguirá mostrándose la misma información. Regresando con la Raspberry Pi se abre otra terminal nueva y se ejecuta el

comando: **`source ~/catkin_et/devel/setup.bash`** y en seguida el comando: **`rosrun numeros numeros_publisher.py`**, este último es para iniciar el nodo publicador que comenzara a imprimir el conteo de números en la terminal.



(a) *(b)*

Figura 102. (a) Terminal en Raspberry Pi que muestra los números publicados por el nodo. (b) Resultados en monitor serie obtenidos por el NodeMCU.

La figura 102(a) muestra el conteo de números desde el nodo publicador de la Raspberry Pi, mientras que la figura 102(b) muestra el monitor de terminal de Ardunio IDE, el cual corrobora que efectivamente el NodeMCU está funcionando como nodo suscriptor y que el mensaje que recibe del nodo publicador se guarda correctamente en la variable **numero.** El contenido de la variable es impreso en el monitor serie y va acorde con lo impreso en la terminal de la Raspberry Pi.

Por otra parte, para comprobar que el NodeMCU es un nodo suscriptor de ROS se hace lo siguiente. En la Raspberry Pi se abre una terminal nueva y se ejecuta el comando: **rqt_graph**.

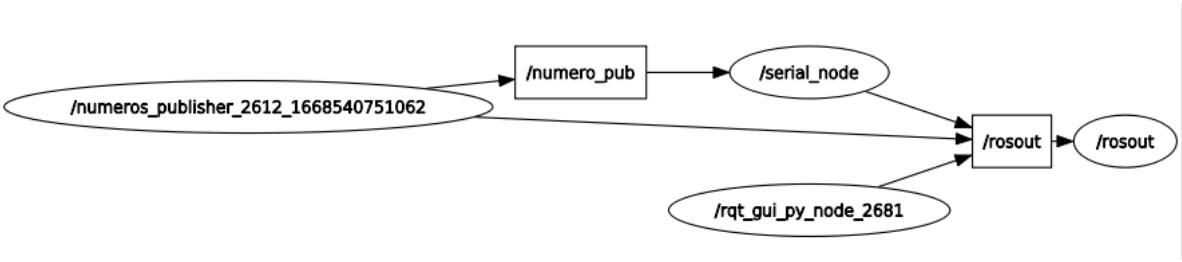


Figura 103. Diagrama de grafos de los nodos activos, usando la herramienta de ROS rqt_grpah.

Una vez que se ejecuta el comando se abre una nueva ventana, la cual es una herramienta de ROS que nos muestra a manera de grafos los nodos activos. En este caso se observa en la figura 103 que se encuentra activo el nodo **numeros_publisher** y el nodo **serial_node** es el que corresponde al NodeMCU, que es él nodo suscriptor. Se observa que ambos nodos se encuentran comunicados por el topic **numero_pub**.

Con el anterior ejemplo se comprueba que el NodeMCU puede también ser programado como un nodo suscriptor y que a su vez guarda el valor en una variable.

Sección K

Programando el NodeMCU para enviar información a una base de datos de un servidor web

Retomando lo que se realizó en la sección anterior, ahora se desea que la información que fue guardada en la variable **numeros** sea enviada a la base de datos de un servidor web local.

Antes de comenzar a realizar este ejemplo y corroborar que efectivamente realice lo que se desea, es necesario contar con lo siguiente: dos computadoras ya sea de escritorio o laptops (una de ellas será el servidor web local y la otra debe tener ROS Noetic instalado); y dos tarjetas NodeMCU con el módulo ESP8266 con sus respectivos cables para conectarlos a un puerto USB. Uno debe estar configurado en modo AP y el otro programado como nodo suscriptor, aunque más adelante se programa para enviar información a un servidor web local. Por otra parte, en el ámbito de software, se debe tener instalado en uno de los equipos de cómputo Arduino IDE, Xampp y en el otro equipo solo contar con ROS. Cabe recordar que

es necesario que los dos equipos de cómputo deben de encontrarse conectados en la misma red local y se tenga el servidor web funcionando, así como la base de datos del servidor web.

En este caso en particular se va a utilizar una PC de escritorio, que cuenta con un adaptador WiFi por USB, Arduino IDE, el servidor web local y la base datos del servidor ya creados. Como segundo equipo se va a usar una Raspberry Pi 4 modelo B que cuenta con ROS Noetic instalado. Previamente se tienen configuradas las direcciones IP acordes a la red local, misma que se crea con ayuda de uno de los NodeMCU configurado en modo AP, figura 104.

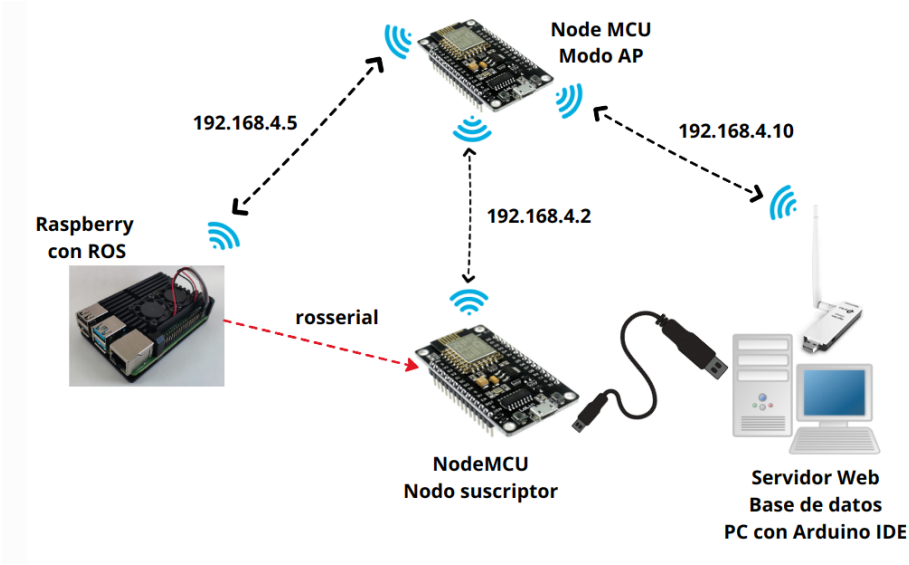


Figura 104. Esquema de conexión parta enviar datos de ROS a un servidor web.

Una vez que se tiene creado el servidor web, la red local y todas las configuraciones pertinentes, se realiza la programación al NodeMCU para enviar datos a un servidor web local. Se configuran y se crean archivos en el servidor web y la base de datos para que estos trabajen conjuntamente [20], [21].

Antes de comenzar es necesario saber lo que es una URL, qué es el protocolo HTTP, los métodos de envío de solicitudes a una página web y los estados de respuesta de HTTP. En este ejemplo se hace uso de la herramienta de software para administración y creación de base de datos llamada HeidiSQL. Cabe recalcar que el método de envió de solicitud a un servidor que se aplicará en esta programación es usando POST.

Los nuevos elementos que se van a ocupar en la programación son los siguientes:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPClient.h>

WiFiClient client;

HTTPClient http;

String postData;

http.begin(client, url);
http.addHeader("Content-Type", "application/x-www-form-urlencoded");

postData = "device_label=" + device + "&temperatura="+ String(temp) +
"&humedad=" + String(hume) + "&numros=" + String(numero);

int httpCode = http.POST(postData);
String respuesta = http.getString();

Serial.println(httpCode);
Serial.println(respuesta);

http.end();
```

Las primeras 4 líneas corresponden a las librerías que son necesarias para la programación, es por ello que las primeras tres líneas corresponden al método de conexión de WiFi con el NodeMCU.

Las siguientes dos líneas corresponden a la definición de dos objetos que se usan y son: **client** y **http**, estas forman parte de la librería de **ESP8266HTTPClient.h** y de las clases **WiFiClient** y **HTTPClient**. Cabe mencionar que en algunos ejemplos sobre este tema de envío de datos a un servidor web por medio de una NodeMCU no se coloca la clase **WiFiClient** ni se define el objeto **client**, de no ser así se muestra el siguiente error al momento de programar al NodeMCU:

```
error: call to 'HTTPClient::begin' declared with attribute error:
obsolete API, use ::begin(WiFiClient, url)
   25 |   http.begin(url); //inicializamos el objeto pasandole una url
      |   ~~~~~~~~~~^~~~
exit status 1
call to 'HTTPClient::begin' declared with attribute error: obsolete API,
use ::begin(WiFiClient, url)
```

Esto es porque en las siguientes líneas se hace uso del objeto **client** en una función que se declara más adelante.

Más adelante, la línea donde se encuentra la función **http.begin** realiza la inicialización de los objetos que fueron definidos o, en otras palabras, esta función realiza la conexión como cliente hacia un servidor y para ello se necesitan especificar los atributos **client** y **url**.

En seguida se encuentra la función **http.addheader,** aquí se especifica el tipo de contenido que ira en la cabecera de la cabecera de **http** para él envió de solicitud

a un servidor, en este caso **POST**. En resumen, la información ira codificada en la URL al enviar los datos.

Como siguiente línea se hace uso de la función **http.POST**, el cuál especifica la solicitud de envió de datos al servidor, esta función tiene como valor una variable de tipo String, que en este caso se define como **postData**. La variable **postData** es donde se define la información que se va a enviar y se envía en pares, o sea nombre de la variable y su valor. Es importante saber que, si los valores de las variables no son tipo String, se recomienda convertir el valor a tipo String. Para el caso de enviar un número finito de variables estos pares deberán ser unidos con el signo **&** y así todas éstas sean enviadas de una sola vez. Para este caso se van a enviar cuatro valores, de los cuales tres son variables: **temperatura**, **humedad** y **numeros**. El valor de la variable **device_label** para este caso siempre será constante. Es importante mencionar que los nombres de las variables deben de ser igual a como se definen en el archivo .php que recibe los datos de un formulario [22]. Continuando con la función **http.POST** se asigna a una variable tipo **int** llamada **httpCode**, ya que el método POST regresa un valor **int**, el cual es un valor que es referido como código de respuesta de envió **http**, si es **200** él envió de datos al servidor fue correcto y en caso contrario si el código es **–1**.

A continuación, se hace uso de la función **http.getstring**, esta se encarga de regresar una respuesta obtenida del servidor. Es por ello que aquí se va a guardar esa respuesta de tipo String en una variable llamada **respuesta,** igual de tipo String.

Las dos antepenúltimas líneas ayudan a imprimir en el monitor serie de Arduino IDE el código de respuesta obtenido y el contenido de la respuesta recibida por el servidor web. Y la última línea finaliza la conexión.

Una vez explicado lo anterior, se realiza la programación del NodeMCU, agregando las líneas anteriormente explicadas en el programa del ejemplo **Programando el NodeMCU como nodo subscriptor en ROS**, el programa queda de la siguiente forma:

```
#include "lib_wifi.h"

#include <ESP8266WiFi.h>

#define ROSSERIAL_ARDUINO_TCP

#include <ros.h>
#include "std_msgs/UInt32.h"

#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPClient.h>


String url = "http://192.168.4.10/servidorev3/prueba_recibe.php";
String device = "tarjeta1";
int temp;
int hume;
int numero;
String postData;

// Set the rosserial socket server IP address
IPAddress server(192,168,4,5);
// Set the rosserial socket server port
const uint16_t serverPort = 11411;

void callback(const std_msgs::UInt32& msg)
{
  numero = msg.data;
}
```

```
ros::NodeHandle nh;

ros::Subscriber<std_msgs::UInt32> sub("numero_pub", &callback);

void setup(){
  conectaWiFi_ESP();

  // Set the connection to rosserial socket server
  nh.getHardware()->setConnection(server, serverPort);
  nh.initNode();

  // Another way to get IP
  Serial.print("IP = ");
  Serial.println(nh.getHardware()->getLocalIP());

  // The subscriber node is started
  nh.subscribe(sub);
}

void loop () {

  if (nh.connected()) {
    Serial.println("Conectado");
    //
    Serial.println("Escucho el número:");
    Serial.println(numero);

    WiFiClient client;

    HTTPClient http; //Se declara el objeto de la clase HTTPClient

    http.begin(client, url); //inicializamos el objeto pasandole una url

    http.addHeader("Content-Type",  "application/x-www-form-urlencoded");
// Permite el tipo de contenido en el header

    //se asignan valores radom a las variables temp y hume

    temp = random(1,60);
    hume = random(10,99);

    //Método de envio por POST, usando POST DATA
    // se especifica los datos que se van a enviar
    // si son valores int se deben de convertir a string para concatenarlos
    postData = "device_label=" + device + "&temperatura="+ String(temp) +
"&humedad=" + String(hume) + "&numros=" + String(numero);
```

```
    //postData = "device_label=tarjeta1&temperatura=23&humedad=15";

    int httpCode = http.POST(postData); // aqui se enviara un codigo (int)
para decir si el envio fue correcto al servidor
    String respuesta = http.getString(); // Regresa respuesta del servidor

    Serial.println(httpCode);
    //Serial.println(postData);
    Serial.println(respuesta);


    http.end(); // Se cierra la conexion

    delay(5000);

  } else {
    Serial.println("No Conectado");
  }
    nh.spinOnce();
    delay(5000);
}
```

Recordando que los archivos creados en Arduino IDE están estructurados por 4 secciones, iremos agregando las nuevas líneas a cada una de ellas. Cabe destacar que para este ejemplo se usa el método de conexión llamado **“wifi manager”** para conectar el NodeMCU a una red Wifi [23]. Así entonces, en la primera sección se agregan las nuevas librerías a usar y se definen las variables: **temp**, **hume** y **postData**. Las variables de **temp** y **hume** se usan para mandar datos al servidor web, estos son usados de acuerdo al ejemplo en el que se está guiando para realizar la programación. Después en la sección de **setup** se agrega la línea **conectaWiFi_ESP()**, esto es para realizar la conexión del NodeMCU con la red wifi, usando el programa de **wifi manager**. En seguida, se va a la última sección donde se define la función **void**, aquí se observa que se define una función **if,** dentro de ésta se agregan las nuevas líneas que corresponden a la programación de enviar

datos a un servidor web y aquí mismo de definen los valores para las variables **temp** y **hume**, que en este caso se usa la función **random** que elige valores aleatorios dentro de un rango que se define aquí mismo.

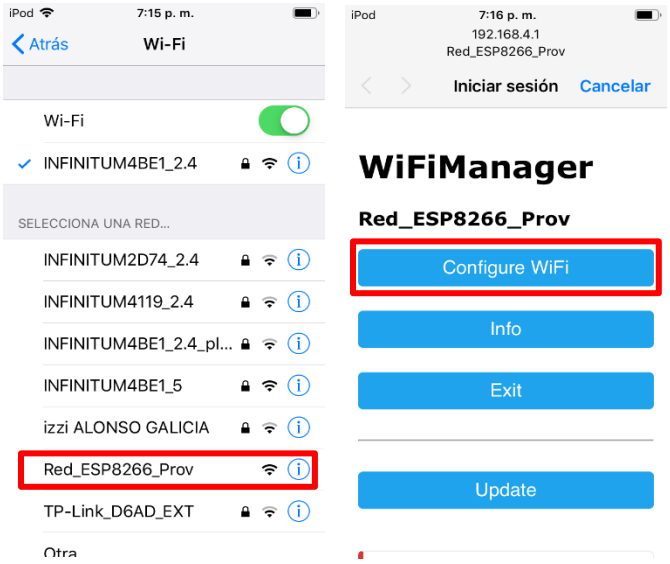
Una vez terminado el programa, se guarda el proyecto y en este caso lo nombramos como **“nodo_enviodatos”**. Después se oprime la opción de **Verificar** de Arduino IDE, esto es para que no exista un error en el programa. En seguida se conecta el NodeMCU a la computadora de escritorio y se carga el programa a la tarjeta.

Una vez programado el NodeMCU, se inicia la WLAN alimentando el NodeMCU que será el AP, después se enciende la tarjeta Raspberry Pi, se verifica que tanto la Raspberry como la computadora que tiene el servidor web estén conectados en la red y por último se inicia el servidor web. Por el momento se desconecta de la computadora el NodeMCU que se acaba de programar. Ahora estando en la Raspberry Pi se abre una terminal nueva para iniciar el Nodo maestro de ROS o el Core, simplemente se ejecuta la línea **roscore**. Nuevamente se abre otra terminal y se ejecuta la siguiente línea que especifica la fuente donde se iniciará el servidor rosserial: **`source ~/catkin_et/devel/setup.bash.`**

Seguidamente se conecta a la computadora el NodeMCU, que es el **nodo_enviodatos,** y teniendo abierto Arduino IDE en la computadora se abre el monitor serie. Se observa que se imprime un mensaje de no conectado, lo que significa el NodeMCU no se encuentra conectado a la red WLAN, así mismo parpadea el led integrado al NodeMCU indicando que no se encuentra conectado una red inalámbrica, figura 105.

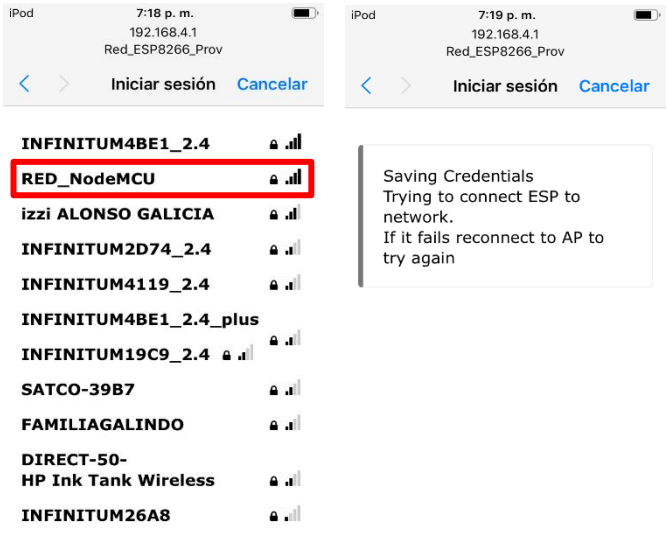
Figura 105. Aviso en monitor serie que el NodeMCU no está conectado a una red WiFi.

Recordando que se hizo uso de la librería **"wifi manager",** el modo de conectar el NodeMCU a la red WLAN se hace de la siguiente manera. Es necesario un smartphone o un dispositivo multimedia con conexión a wifi, en este caso se hace uso de un IPod. Se enciende la conexión de wifi del IPod y se dirige a las configuraciones de conexión wifi, dentro se observa una lista de redes disponibles, ahí se va a encontrar una con el nombre de **"Red_ESP8266_Prov"**, se elige esa conexión y se conecta a esa red. Esta función que proporciona **"wifi manager"** es que su programa verifica que, si el NodeMCU no se encuentra conectado a una red wifi, este configura al NodeMCU como un AP provisional para que al momento que te conectes puedas configurar al NodeMCU para que se conecte a una red wifi. Una vez que se conecta el IPod a la red aparece en pantalla algo como las siguientes figuras.



(a) *(b)*

Figura 106. (a) Lista de redes inalámbricas disponibles, vistas desde dispositivo IPod. (b) Página de inicio para configura el NodeMCU para conectarse a una red WiFi.



(a) *(b)*

Figura 107. (a) Listado de las redes inalámbricas disponibles para conectarse, vistas desde el NodeMCU. (b) Mensaje mostrado tras conectarse a la Red_NodeMCU.

De la figura 106(b) se muestra en la pantalla del IPod un menú, del cual se elige la opción **Configure WiFi**, seguidamente se observa en pantalla como la figura 107(a) que es un listado de las redes inalámbricas disponibles en este caso se elige nuestra red local que se llama **RED_NodeMCU**, después pide la contraseña de la red a la que se va a conectar, en este caso es **Is3tu4cm**, se da la opción **Save** y al final se muestra un mensaje de que las credenciales fueron guardadas, como se muestra en la figura 107(b). A continuación, se verifica la conexión abriendo el monitor serie de Arduino IDE. Se observan los siguientes mensajes, figura 108.

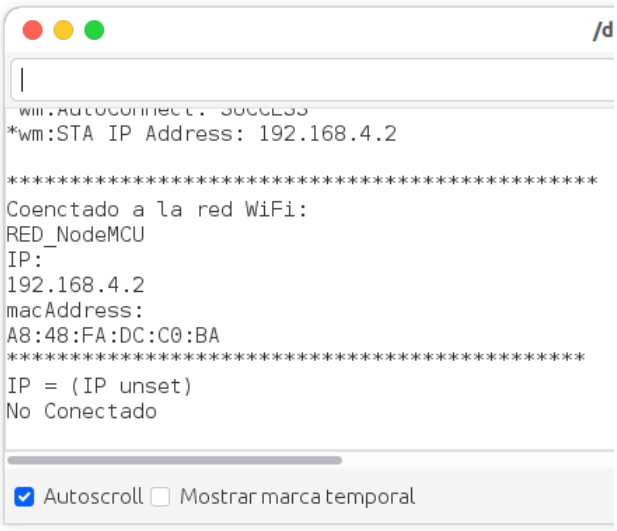


Figura 108. Despliegue de resultados en monitor serie tras conectar el NodeMCU a una red WLAN.

Se verifica que el NodeMCU se encuentra conectado a una red inalámbrica y se muestra la dirección IP que se le asigna, los últimos mensajes corresponden a que el NodeMCU como nodo no se encuentra conectado con el servidor rossserial.

Regresamos a la Raspberry Pi y en la terminal que se va a iniciar el servidor rosserial se ejecuta la siguiente línea: **`rosrun rosserial_python serial_node.py tcp`.**

Tras ejecutar la instrucción, veremos en la terminal lo que se observa en la figura 109.

Figura 109. Resultados obtenidos en terminal de Raspberry Pi, tras iniciar el servidor rosserial.

Los mensajes que se despliegan significan que se ha encontrado el nodo suscriptor del NodeMCU. Por otro lado, en la computadora de escritorio en el monitor serie de Arduino IDE se observan los mensajes mostrados en la figura 110.

Figura 110. Resultados obtenidos en monitor serie tras iniciar el servidor rosserial.

Se comienzan a desplegar los mensajes de: **Conectado**, se imprime el valor de la variable número que por ahorita es **0**, se imprime el valor de **200** que es el estado de recepción del envió de datos al servidor web y por último se imprime la respuesta del servidor web ya que los valores de las variables **numero**, **temp** y **hume** fueron enviados al servidor web y guardados en la base de datos del mismo.

Regresando a la Raspberry Pi, se abre una nueva terminal y se ejecuta la siguiente línea: **`source ~/catkin_et/devel/setup.bash`** y después el siguiente enunciado de comandos para iniciar el nodo publicador: **`rosrun numeros numeros_publisher.py`**

Tras iniciar el nodo publicador en la terminal de la Raspberry Pi se imprime el conteo de los números, ahora si se dirige a la computadora al monitor serie de Arduino IDE, se observa que el NodeMCU que es nodo suscriptor si recibe la información sobre

el conteo de números, los imprime y a su vez los envía al servidor web y el este mismo nos envía la respuesta imprimiendo los valores que recibió. Cabe mencionar que existe un retraso de aproximadamente milisegundos en cuanto a la impresión de la información en el monitor serie.

Por último, con ayuda del software HeidiSQL se corrobora que los datos sí llegan a la base de datos del servidor web, se actualizan y que exista un historial de estos mismos.

Se toma como ejemplo que el nodo publicador de ROS publica el numero 38, por lo tanto, se guarda en la variable **numero,** que se definió en el NodeMCU, y seguidamente éste se envía al servidor web y se guarda en la base de datos en la variable **numros**, por otro lado, las variables de **temp** y **hume,** generadas en el NodeMCU, también son enviadas al servidor y guardadas en la base de datos. **temp** tiene un valor de 4 y **hume** un valor de 88. En las figuras siguientes se corrobora todo lo anteriormente mencionado.

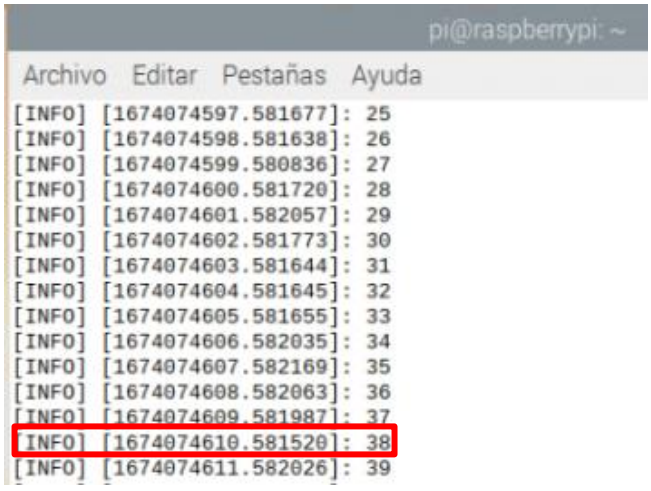


Figura 111. Despliegue de números en terminal, publicado por el nodo.

Figura 112. Resultados mostrados en el monitor serie del NodeMCU que es nodo suscriptor.

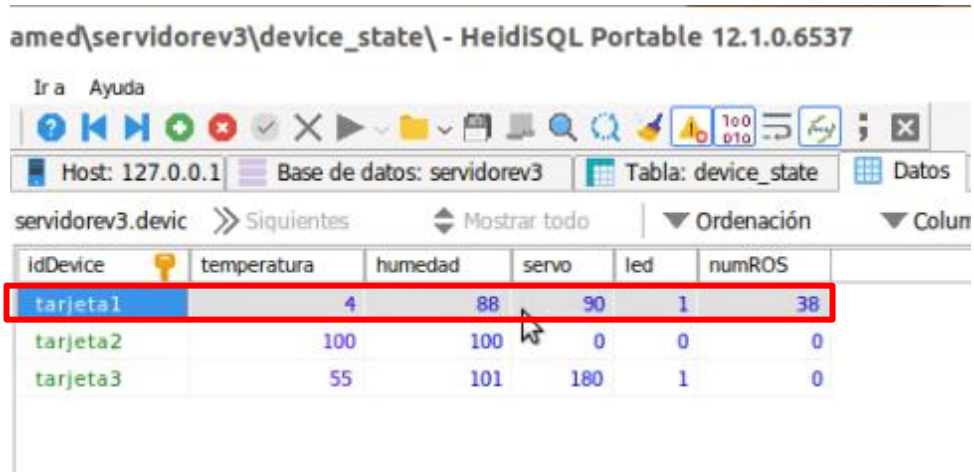
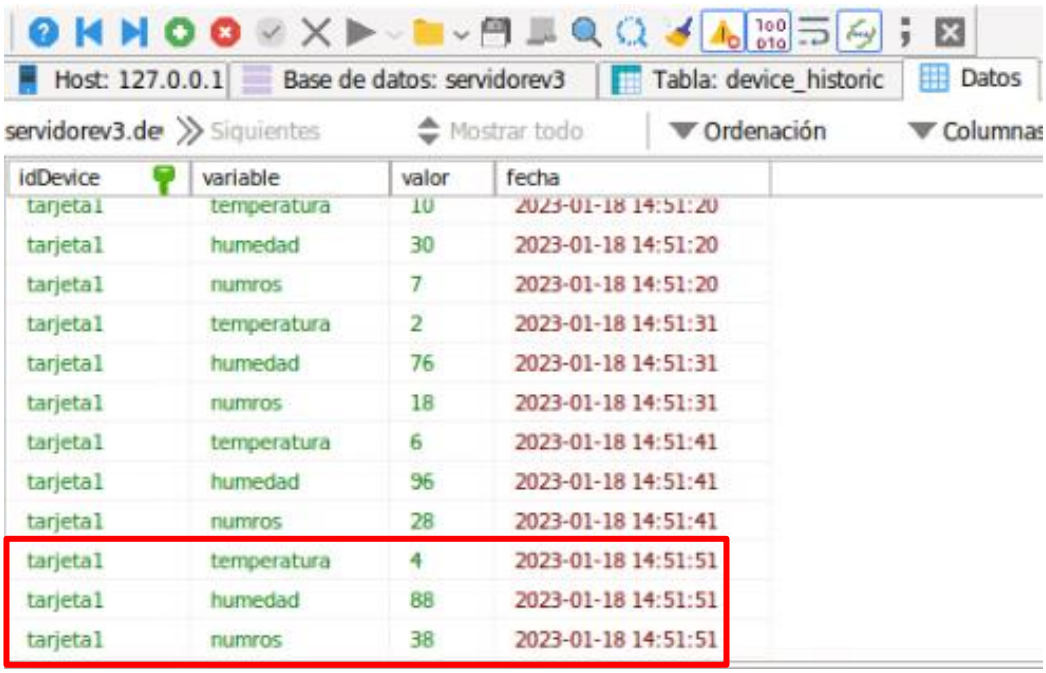


Figura 113. Base de datos del servidor confirmando que llegaron los datos enviados desde NodeMCU.



idDevice	variable	valor	fecha
tarjeta1	temperatura	10	2023-01-18 14:51:20
tarjeta1	humedad	30	2023-01-18 14:51:20
tarjeta1	numros	7	2023-01-18 14:51:20
tarjeta1	temperatura	2	2023-01-18 14:51:31
tarjeta1	humedad	76	2023-01-18 14:51:31
tarjeta1	numros	18	2023-01-18 14:51:31
tarjeta1	temperatura	6	2023-01-18 14:51:41
tarjeta1	humedad	96	2023-01-18 14:51:41
tarjeta1	numros	28	2023-01-18 14:51:41
tarjeta1	temperatura	4	2023-01-18 14:51:51
tarjeta1	humedad	88	2023-01-18 14:51:51
tarjeta1	numros	38	2023-01-18 14:51:51

Figura 114. Muestra de historial de los datos recibidos desde el NodeMCU a la base de datos del servidor web.

En la última figura se observa el historial de todos los valores que han tenido las variables **temp**, **hume** y **numros** con su respectiva fecha y hora.

Se observa que sí es posible enviar información de ROS a un servidor web, guardarla en su base de datos y contar con un historial de la información. Esto tiene muchas aplicaciones futuras.

Referencias

[1] *Como instalar RASPBERRY PI OS (Raspbian) sin monitor, mouse ni teclado*, (17 de junio de 2020). Accedido: 8 de noviembre de 2022. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=fFj3a4qtTkA

[2] J. A. Castillo, «▷ Cómo instalar Raspbian en VirtualBox paso a paso», *Profesional Review*, 29 de diciembre de 2018. https://www.profesionalreview.com/2018/12/29/instalar-raspbian-virtualbox/ (accedido 6 de marzo de 2023).

[3] *Raspberry Pi 4, solucion al Cannot currently show the desktop al conectarse por VNC*, (12 de abril de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=gL5U36Ul_d4

[4] VarHowto Editor, «How to Install ROS Noetic on Ubuntu 20.04 - VarHowto», 19 de agosto de 2020. https://varhowto.com/install-ros-noetic-ubuntu-20-04/ (accedido 6 de marzo de 2023).

[5] G. Perez, «Introducción a ROS en Raspberry Pi», Tesis Máster Universitario Ingeniería de Telecomunicación, Oberta de Catalunya, España, 2017. [En línea]. Disponible en: https://openaccess.uoc.edu/bitstream/10609/63705/3/jgutierrezperTFM0617memoria.pdf

[6] L. Joseph y J. Cacace, *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System, 2nd Edition*. Packt Publishing Ltd, 2018.

[7] L. Joseph, *Learning Robotics using Python: Design, simulate, program, and prototype an autonomous mobile robot using ROS, OpenCV, PCL, and Python, 2nd Edition*. Packt Publishing Ltd, 2018.

[8] L. Joseph, *Learning robotics using Python: design, simulate, program, and prototype an autonomous mobile robot using ROS, OpenCV, PCL, and Python*, Second edition. Birmingham Mumbai: Packt, 2018.

[9] automaticaddison, «Working With ROS and OpenCV in ROS Noetic – Automatic Addison», 27 de junio de 2020. https://automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/ (accedido 6 de marzo de 2023).

[10] *Efecto ESPEJO en Python-OpenCV*, (30 de julio de 2019). Accedido: 8 de noviembre de 2022. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=aDK-PaQACIU

[11] W. Auer, «Connecting EV3 Brick (EV3dev) and Raspberry Pi via USB (November 11, 2017)», *Noscere Vivere Est*, 11 de noviembre de 2017. https://noscerevivereest.wordpress.com/2017/11/11/connecting-ev3-brick-ev3dev-and-raspberry-pi-via-usb-november-11-2017/ (accedido 6 de marzo de 2023).

[12] «Working with ev3dev remotely using RPyC — python-ev3dev 1.0.0.post5 documentation». https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/ev3dev-jessie/rpyc.html (accedido 6 de marzo de 2023).

[13] «RPyC on ev3dev — python-ev3dev 2.1.0.post1 documentation». https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/rpyc.html (accedido 6 de marzo de 2023).
[14] «Xbee con Raspberry Pi». Accedido: 6 de marzo de 2023. [En línea]. Disponible en: https://www.electronicaembajadores.com/datos/pdf1/lc/lcrb/lcrbssb.pdf
[15] *Configure Serial Raspberry Pi 4B - UART - GPIO 14 and 15*, (13 de agosto de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=oevxqPk78sM
[16] «Instrucciones de WHADDA WPSH456 NEO-6M Shield para Raspberry PI - Manuales+». https://manuals.plus/es/whadda/wpsh456-neo-6m-shield-for-raspberry-pi-manual#axzz7Xj0riYo4 (accedido 6 de marzo de 2023).
[17] «NodeMCU Rectangular / Base Shield NodeMCU UNIT Electronics», *UNIT Electronics*. https://uelectronics.com/producto/nodemcu-rectangular-base-shield-nodemcu/ (accedido 29 de noviembre de 2022).
[18] «rosserial_arduino/Tutorials/Arduino IDE Setup - ROS Wiki». http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup (accedido 25 de octubre de 2022).
[19] R. G, «ROS en ESP8266», *Notas sobre robótica, domótica, sistemas operativos y programación*, 25 de junio de 2018. https://minibots.wordpress.com/2018/06/25/ros-en-esp8266/ (accedido 6 de marzo de 2023).
[20] ✔ *Enviar datos a un servidor web a través de un formulario* 👈 - 😉 *Curso IoT con ESP8266 #21*, (2 de marzo de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=gSqyScSLUCQ
[21] ✔ *Almacena la información en una Base de Datos de MySQL Server* 👈 - 😉 *Curso IoT con ESP8266 #22*, (4 de marzo de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=U3lMjhQ64TI
[22] ✔ *Envía datos de tu tarjeta NodeMCU ESP8266 a tu servidor web local* 👈 - 😉 *Curso IoT con ESP8266 #24*, (16 de marzo de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=5bX_wHDf9Ts
[23] ✔ *WiFiManager con NodeMCU ESP8266 -* 👈 *Cambia SSID y PWD desde página -* 😉 *Curso IoT con ESP8266 #18*, (15 de febrero de 2021). Accedido: 6 de marzo de 2023. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=phKFYccGO2A