

# Práctica 1. Primeros pasos con ROS 2

## (Duración: 2 sesiones)

### Introducción

En esta práctica inicial adquirirá algo de experiencia en el uso de ROS 2 (del inglés, *Robot Operating System*), la revisión más reciente del sistema operativo —aunque es más bien un conjunto de librerías de *software* y herramientas— que se ha convertido en el estándar *de facto* para la programación de robots, no solo a nivel de investigación sino cada vez más también en entornos industriales. Las primeras secciones son una adaptación de algunos tutoriales oficiales de ROS<sup>1</sup>, que han sido seleccionados para destacar los conceptos principales. Siéntase libre de explorar el resto de tutoriales si tiene interés en aprender más.

Como primer paso hacia la construcción de un robot autónomo con código modular, aprenderá a crear dos aplicaciones (nodos) que se intercambien mensajes utilizando ROS 2 mediante un mecanismo de publicación/suscripción (*pub/sub*) basado en temas (*topics*). Aunque C++ es más común en la programación de robots por su mayor rendimiento, utilizaremos Python como lenguaje principal dado que es más fácil prototipar con él y porque ya debería ser un experto en su manejo a estas alturas del grado.


Para acabar, se conectará a un viejo conocido, el robot TurtleBot3 Burger de Robotis<sup>2</sup> con el que ya ha tenido ocasión de trabajar en Sistemas Dinámicos el cuatrimestre pasado. Una vez se haya familiarizado con la forma de manejarlo con ROS 2, utilizará sus recientemente adquiridos conocimientos para desarrollar un nodo de teledirección que mejore el existente. Si es hábil, quizá pueda ganar a sus compañeros en una carrera...

### Objetivos

Al concluir la práctica, el alumno deberá ser capaz de:

- Comprender los fundamentos de los principales componentes de ROS 2 y cómo interactúan entre sí.
- Crear y compilar paquetes (*packages*).
- Escribir nodos (*nodes*) simples en Python que publiquen y se suscriban a temas (*topics*).
- Desplegar y ejecutar nodos de ROS 2 en el robot TurtleBot3 Burger.

### Evaluación

Esta práctica carece de trabajo previo por lo que se evaluará exclusivamente por el informe en formato PDF que deberá entregar a través de Moodle una semana después del final de la segunda sesión. El informe debe incluir las respuestas a las preguntas que se van planteando en el enunciado (identificadas con el símbolo ) , así como el código desarrollado —que deberá entregar también comprimido en formato .zip o .7z—. **No es necesario que adjunte el código de los tutoriales de ROS 2**, solo el que desarrolle para el TurtleBot3. Tenga en cuenta que puede responder a la mayoría de las preguntas sobre la marcha.

<sup>1</sup><https://docs.ros.org/en/humble/Tutorials.html>

<sup>2</sup><https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>

## Preparación

En esta asignatura trabajará preferentemente en su **portátil personal** para que pueda desarrollar cómodamente fuera del laboratorio. Como la instalación de todas las dependencias no es tarea sencilla, usaremos un Dev Container<sup>3</sup> de Visual Studio Code, que no es más que un contenedor de Docker altamente integrado con el entorno de desarrollo. Presenta la ventaja de que mantiene los archivos en el sistema operativo anfitrión y permite incluso instalar extensiones dentro del contenedor con solo indicarlo en un archivo de configuración denominado `devcontainer.json`. Esto garantiza que todos los alumnos tengan exactamente la misma instalación base.

Antes de la **primera sesión**, instale Docker Desktop<sup>4</sup> —posiblemente ya disponga de él de otras asignaturas— y la extensión Dev Containers<sup>5</sup> de Visual Studio Code. A continuación, descargue el Dev Container que encontrará en la sección de laboratorio de Moodle y descomprímalo en el directorio de su elección. Tenga en cuenta que, una vez construido, **no debe cambiarlo de ubicación**. Aparecerá una carpeta denominada `sim_ws` que, según su sistema operativo y las opciones del explorador que tenga activas, puede que parezca vacía, ya que los directorios precedidos por un punto se consideran ocultos por defecto en sistemas basados en Unix. Abra la carpeta con el menú de Visual Studio Code —o arrástrela directamente sobre la ventana— y podrá ver su contenido. Si dispone de la extensión Dev Containers, le preguntará si quiere construir el contenedor. Responda afirmativamente y espere unos minutos hasta que se instale. Las siguientes veces que lance el entorno, el contenedor se cargará de forma prácticamente instantánea.

**▲ Observación:** Asegúrese de cerrar sesión en Docker Desktop antes de construir el contenedor. De lo contrario, el proceso falla porque no consigue localizar la imagen base en Docker Hub.

Una vez dentro del contenedor, basado en **Ubuntu 22.04 LTS** (Jammy Jellyfish), tendrá acceso a todas las utilidades de ROS 2 a través del terminal integrado en Visual Studio. Desafortunadamente, hay **herramientas de visualización** útiles para depurar que no siempre están accesibles mediante el terminal. En versiones recientes de Windows Subsystem for Linux (WSL), la posibilidad de ejecutar aplicaciones gráficas está soportada por defecto. En caso de que no pueda actualizar WSL o tenga macOS, el contenedor lanza al arrancar un servidor de escritorio remoto que le permite utilizar la interfaz gráfica. La información para acceder se indica en la Tabla 1. Descubrirá que el escritorio no es el estándar de Ubuntu, sino una versión más ligera, LXQt, que es la predeterminada de Lubuntu<sup>6</sup>.

**Tabla 1.** Credenciales para conectarse por escritorio remoto al Dev Container.

Propiedad	Valor
Nombre del equipo ( <i>hostname</i> )	localhost
Usuario	ros
Contraseña	humble

Para la **segunda sesión**, compruebe que tiene instalada la extensión Remote-SSH<sup>7</sup>, que ya utilizó el curso pasado en Sistemas Electrónicos. La empleará para conectarse al TurtleBot3.

<sup>3</sup><https://code.visualstudio.com/docs/devcontainers/containers>

<sup>4</sup><https://www.docker.com/products/docker-desktop/>

<sup>5</sup><https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers>

<sup>6</sup><https://lubuntu.me/>

<sup>7</sup><https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh>

## 1. Conoce tus armas (0,5 puntos)

Dado que se libera una nueva distribución de ROS 2 con mejoras significativas cada 23 de mayo —para conmemorar el Día Mundial de las Tortugas—, conviene comprobar siempre que se empieza a trabajar con un robot nuevo cuál es exactamente la distribución que tenemos entre manos. Esto permite consultar posteriormente la documentación adecuada y asegurarse de que las características que se van a necesitar están disponibles. Escriba lo siguiente en una pestaña del terminal integrado de Visual Studio Code:

```
printenv ROS_VERSION
printenv ROS_DISTRO
```

**Código 1.** Comandos para obtener la versión y el nombre de la distribución de ROS, respectivamente.

🔗 **Pregunta #1:** ¿Cuál es la versión y el nombre de la distribución de ROS instalada en el contenedor que se le ha proporcionado? ¿Cuándo finaliza su soporte? ¿Por qué?

## 2. ¿Dónde están las islas Galápagos? (0,5 puntos)

### 2.1. ros2 run

El comando `ros2 run` permite ejecutar un nodo sin tener que proporcionar explícitamente la ruta del paquete. Para probarlo, lanzaremos una instancia de TurtleSim, un simulador 2D ligero pensado para ayudar a los principiantes a iniciarse en los conceptos de ROS. En el mismo terminal que antes, escriba la última línea del Código 2.

```
ros2 run <nombre_del_paquete> <nombre_del_nodo>
ros2 run turtlesim turtlesim_node
```

**Código 2.** Estructura de `ros2 run` con los argumentos encerrados entre `<>` (arriba) y comando para lanzar TurtleSim (abajo).

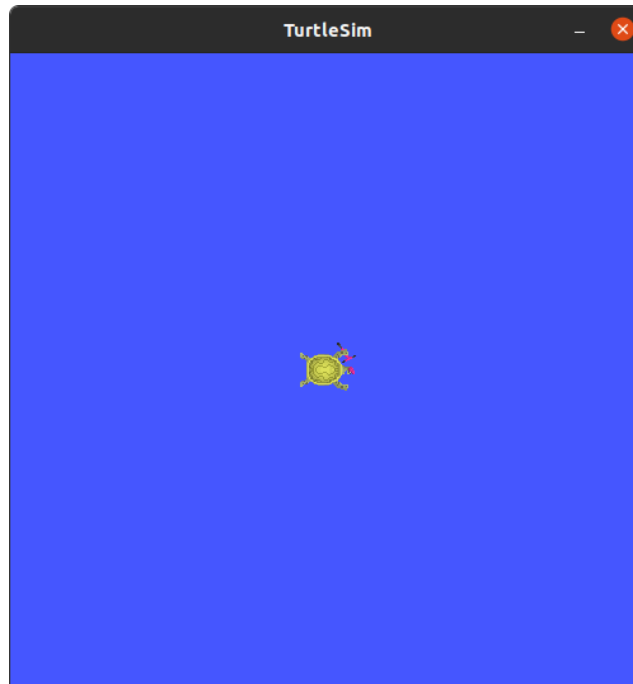
Se creará una nueva ventana con una tortuga centrada sobre un fondo azul —nadando en busca del paraíso de las tortugas— como la que se muestra en la Figura 1. No se preocupe si su tortuga tiene un aspecto diferente, siempre es una sorpresa qué especie de tortuga va a aparecer. Por descontado, deberá acceder al contenedor mediante escritorio remoto para verla.

### 2.2. ros2 node

Analicemos lo que ha ocurrido internamente. Sin cerrar el terminal anterior —de lo contrario cerrará el simulador—, abra una **nueva** pestaña del terminal e introduzca el Código 3 para obtener una lista de los nodos activos. Dado que el único nodo en ejecución es el simulador que acaba de lanzar, solo debería ver `/turtlesim` en pantalla.

```
ros2 node list
```

**Código 3.** Listar los nodos activos.



**Figura 1.** Ventana de TurtleSim.

**Pregunta #2:** ¿Qué otro comando se puede ejecutar con `ros2 node`? Escriba `ros2 node -h` para obtener ayuda.

Una característica interesante de ROS —y divertida, por qué no— es que se pueden reasignar los nombres de los nodos desde la línea de comandos para mejorar su identificación. Cierre la ventana de TurtleSim para detener el nodo `/turtlesim` —o vuelva al terminal desde el que ejecutó `ros2 run turtlesim` y pulse `Ctrl`+`C`— y vuelva a ejecutarlo, pero esta vez utilizando un nombre diferente (Código 4). `ros2 node list` debería reflejar el cambio.

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=michelangelo
```

**Código 4.** Lanzar un nodo de ROS 2 con un nombre diferente.

### 3. El Maestro Astilla entrena a Michelangelo (1,5 puntos)



TurtleSim es bastante inútil por sí mismo. ¿De qué sirve una tortuga que no se puede controlar? Por esta razón vamos a ejecutar un nodo para guiar a la tortuga usando el teclado (Código 5). Asegúrese de que la ventana del terminal está activa o no se registrarán las pulsaciones.

```
ros2 run turtlesim turtle_teleop_key
```

**Código 5.** Ejecutar el nodo de ROS que permite teleoperar la tortuga de TurtleSim usando el teclado.

¿Qué está pasando bajo el capó? Los nodos `/turtlesim` —también conocido como `/michelangelo`— y `/teleop_turtle` se comunican a través de un tema (*topic*) de ROS. `/teleop_turtle` **publica** las pulsaciones de las teclas en un tema con un nombre concreto, mientras que `/michelangelo` está **suscrito** al mismo tema para recibir notificaciones cada vez que se presione una tecla.

### 3.1. rqt\_graph

Aunque en teoría se puede realizar un seguimiento de las relaciones entre los nodos utilizando `ros2 node info`, a medida que el programa se hace más complejo, esto se vuelve prácticamente imposible. Afortunadamente, ROS tiene una herramienta de visualización (Figura 2) que se puede iniciar con el Código 6. Asegúrese de marcar todas las casillas en la barra superior antes de pulsar  para refrescar la pantalla y ocultar la información que es irrelevante en este momento. También puede hacer clic en  para que el gráfico se ajuste automáticamente al espacio disponible.

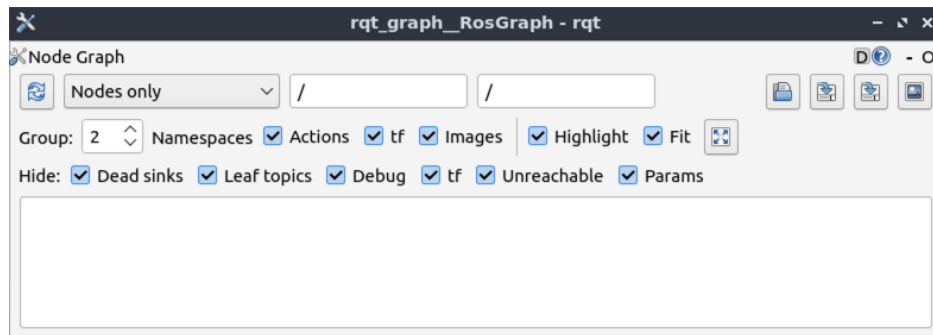


Figura 2. Interfaz de rqt\_graph.

```
rqt_graph
```

Código 6. Comando para abrir rqt\_graph.

**Pregunta #3 (🔗):** ¿Cómo se llama el tema (*topic*) donde `/teleop_turtle` publica los comandos de movimiento? Adjunte una captura de pantalla de rqt\_graph donde se vea el nombre.

🔗 ¡Enhorabuena! Ha obtenido información clave que le ayudará a ganar la carrera de las tortugas durante la segunda sesión de la práctica.

### 3.2. ros2 topic

La herramienta `ros2 topic` dispone de varios subcomandos que permiten recopilar información sobre los temas de ROS. Examinemos algunos de ellos con más detalle.

- `ros2 topic echo` muestra los datos publicados en un tema. En un nuevo terminal, escriba este comando seguido del nombre del tema que el nodo de teleoperación utiliza para controlar el simulador. Si desmarca la opción “Debug” en rqt\_graph, debería ver un nuevo nodo suscrito al tema. ¿Por qué no ve nada en el terminal entonces? ¡Porque aún no se ha publicado nada! Vuelva al terminal del nodo de teleoperación, pulse las teclas de flecha un par de veces y luego regrese de nuevo a la de `ros2 topic echo`.

**Pregunta #4 (🔗):** ¿Qué recibe `/michelangelo` de `/teleop_turtle`?

- `ros2 topic hz` proporciona estadísticas sobre la tasa de publicación de un tema determinado. Como en este caso no se publica nada de forma periódica, la información no resulta demasiado útil.
- `ros2 topic list` devuelve una lista con todos los temas que se están publicando actualmente o a los que hay alguien suscrito. Si utiliza este comando en combinación con `-t` o `--show-types`, se mostrará adicionalmente el tipo de mensaje (*message*).

- También puede comprobar el tipo de mensaje de un tema determinado mediante `ros2 topic info`. Además, mostrará el número de publicadores y suscriptores.

```
ros2 topic info <nombre_del_tema>
```

**Código 7.** Comprobar el tipo de mensaje publicado por un tema.

Una vez que conozca el tipo de mensaje, puede obtener su estructura de datos interna con:

```
ros2 interface show <tipo_de_mensaje>
```

**Código 8.** Mostrar los detalles de un mensaje.

🔗 **Pregunta #5 (🔗):** ¿Qué tipos de datos se intercambian entre el nodo de teleoperación y TurtleSim?

## 4. ¡Ey! ¿Qué pasa Raphael? Aquí Mikey (2,5 puntos)

### 4.1. Crear un espacio de trabajo

Un espacio de trabajo (*workspace*) es un directorio que contiene paquetes de ROS 2 y permite la compilación de múltiples de ellos a la vez. El Dev Container que ha descargado de Moodle ya contiene uno denominado `sim_ws`, en el que desarrollará todo el código de la asignatura destinado al entorno simulado. Cree un subdirectorio `src` en la raíz de su espacio de trabajo para almacenar los paquetes. El Código 9 muestra cómo hacer todo este proceso con el terminal, en lugar de utilizar la interfaz gráfica. Como ya dispone del espacio de trabajo, solo necesita ejecutar la última línea.

```
mkdir sim_ws
mkdir sim_ws/src
```

**Código 9.** Comandos para crear la estructura de carpetas de un espacio de trabajo de ROS 2 en Linux.

### 4.2. Crear un paquete

Los paquetes (*packages*) son la principal unidad de organización para el código de ROS 2, ya que permiten instalar y compartir su trabajo con la comunidad fácilmente. Puede crear un paquete tanto en C++ (usando CMake) como en Python. La primera línea del Código 10 muestra el comando para crear un paquete en Python. Ejecútelo **dentro de la carpeta** `src` y nombre su paquete `tutorial_pubsub_py`. Esto debería añadir una nueva carpeta homónima que contiene varios archivos y subdirectorios. Como nota al margen, si alguna vez quiere crear un paquete y la plantilla de un nodo sencillo simultáneamente, puede añadir el argumento opcional `--node-name` al comando anterior.

```
ros2 pkg create --build-type ament_python <nombre_del_paquete>
ros2 pkg create --build-type ament_python --node-name <nombre_del_nodo> <nombre_del_paquete>
```

**Código 10.** Comandos para crear un paquete de Python.

### 4.3. Programando publicadores y suscriptores en Python

Para terminar esta sección, seguiremos el tutorial oficial de ROS 2 sobre cómo crear dos nodos en Python que se comuniquen a través de un tema. Puede acceder al tutorial directamente desde el siguiente enlace, pero tenga cuidado con algunos detalles antes de sumergirse en él:

- Puede saltarse el primer paso, dado que acaba de crear el paquete.
- El espacio de trabajo en el tutorial se llama `dev_ws` en lugar de `sim_ws` —obviamente, la persona que lo escribió no estaba pensando exclusivamente en esta asignatura— de modo que asegúrese de modificarlo donde sea necesario.
- Cuando vaya a ejecutar los nodos, recuerde que el terminal integrado en Visual Studio Code soporta múltiples pestañas y vistas divididas.

<https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html>

Ahora que ya tiene algo de experiencia práctica con los nodos, sigamos recopilando información para el reto final:

🔗 **Pregunta #6 (Q):** ¿Qué campos hay que completar en `package.xml`? ¿Qué es exactamente una licencia en este contexto? ¿Puede citar algunas licencias de *software*?<sup>8</sup>

🔗 **Pregunta #7 (Q):** ¿Qué hay que añadir a `setup.py`?

🔗 **Pregunta #8 (Q):** ¿Cuáles son los pasos para construir y ejecutar un nodo?

🔗 **Pregunta #9:** En sus propias palabras, explique brevemente cómo se comunica el nodo parlante (*talker*) con el oyente (*listener*).

## 5. Tortugas a la carrera (5 puntos)

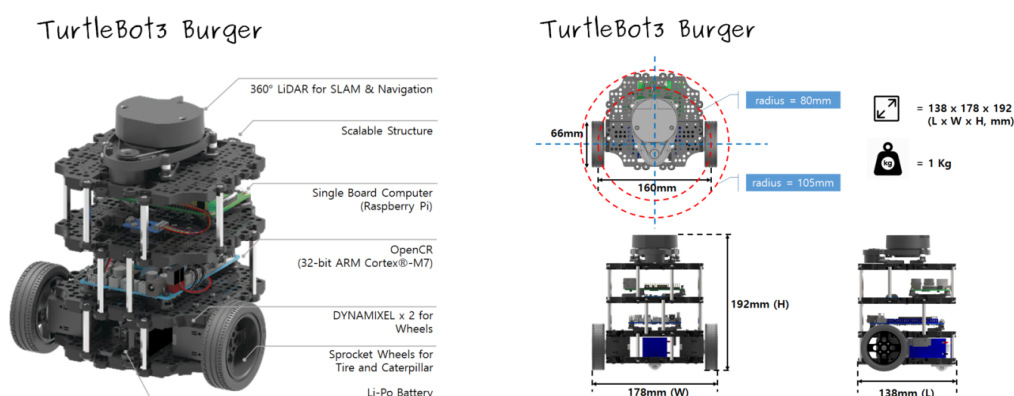
En esta segunda parte de la práctica, va a aprender a controlar el TurtleBot3 Burger (Figura 3) utilizando ROS 2. Primero ejecutará todos los nodos que se encargan de gestionar los sensores y actuadores con un archivo de lanzamiento (*launch file*) y se familiarizará con los temas que publican y a los que están suscritos utilizando el terminal y las herramientas de gráficas `rqt` y `rviz2`. Posteriormente ejecutará un nodo de teleoperación —bastante rudimentario, por cierto— que permite controlar el robot de forma remota con el teclado. Finalmente, tendrá que implementar su propio módulo de teleoperación supervisado —todavía es un ingeniero robótico en prácticas— que impida que el robot se choque.

Para ello, lo primero que necesita hacer es conectarse a la Raspberry Pi 4 Model B de 4 GB de RAM, que dispone de la misma distribución de Linux que el contenedor de desarrollo: Ubuntu 22.04.5 LTS (Jammy Jellyfish). Aquí es donde se ejecutarán todos los algoritmos de control del robot, los cuales se comunicarán entre sí utilizando temas. Primero se conectará a través de escritorio remoto para que pueda utilizar las herramientas de visualización de ROS 2 y, en adelante, usaremos casi en exclusiva la conexión por SSH. Las credenciales que necesita utilizar en ambos casos se muestran en la Tabla 2.

Por defecto el robot se une a la red **comillas** al arrancar, de modo que tendrá que conectarse a esa misma red con su portátil para poder interactuar con él. Los ordenadores de sobremesa del laboratorio deberían tener acceso sin tener que realizar ninguna configuración adicional.

<sup>8</sup>Quizá tenga que buscar en Internet o preguntar a algún modelo de lenguaje para responder esta última pregunta.





**Figura 3.** Componentes (a) y dimensiones (b) del robot TurtleBot3 Burger.

**Tabla 2.** Información necesaria para conectarse de manera remota a la Raspberry Pi. <xy> debe reemplazarse por el número de dos dígitos que figura en la etiqueta del robot. Por ejemplo, la IP podría ser 10.120.107.142 y el nombre del equipo imat-tb3-042.

Propiedad	Valor
IP (comillas)	10.120.107.1<xy>
IP (iMATx)	192.168.0.<xy>
Nombre del equipo ( <i>hostname</i> )	En comillas: imat-tb3-0<xy>.rpi-deac.alumnos.upcont.es En cualquier otro sitio: imat-tb3-0<xy>
Usuario	turtlebot
Contraseña	imat@ICAI2025

Si existe algún problema con la red de la universidad, los profesores desplegarán los *routers* del laboratorio denominados **iMATx**, donde x es un número entre 1 y 3. Los TurtleBot3, que se identifican con un número de dos cifras, están configurados para conectarse con mayor prioridad que a comillas solo a uno de estos *routers*. Aquellos cuya cifra más significativa sea 1 o 4 (por ejemplo, el robot 42), se conectarán a iMAT1; los que empiecen por 2 o 5, a iMAT2; y los que comiencen por 3 o 6, a iMAT3. En esta situación solo podrá utilizar la IP para acceder a su robot. Si el robot estuviera encendido antes de que la red iMATx sea visible, tendrá que reiniciarlo.

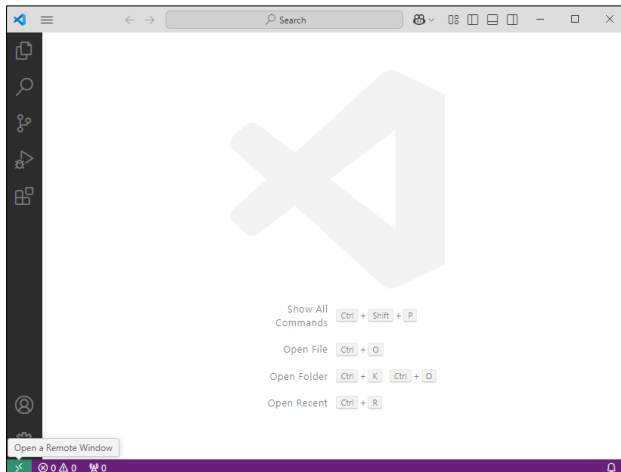
## 5.1. Conexión a través de SSH desde Visual Studio Code

Por si no recuerda cómo conectarse por SSH (de *Secure SHell*) desde Visual Studio Code, a continuación se reproducen las instrucciones. Asegúrese de tener la extensión Remote-SSH habilitada, haga clic en el icono verde de la esquina inferior izquierda (Figura 4(c)) y seleccione la opción **Connect Current Window to Host...** Alternativamente, abra la interfaz de comandos de Visual Studio Code con **Ctrl** + **⇧** + **P**<sup>9</sup> y busque la opción escribiendo, por ejemplo, **ssh connect** (Figura 4(c)). Finalmente, introduzca el usuario y la IP (Figura 4(d)) o el nombre (Figura 4(e)) de su Raspberry Pi separados por una arroba siguiendo las directrices de la Tabla 2. La primera vez que se conecte aparecerá ventana preguntando el tipo de sistema operativo remoto —seleccione **Linux**, obviamente— y, a continuación, otra para que acepte la huella digital (*fingerprint*) del servidor SSH.

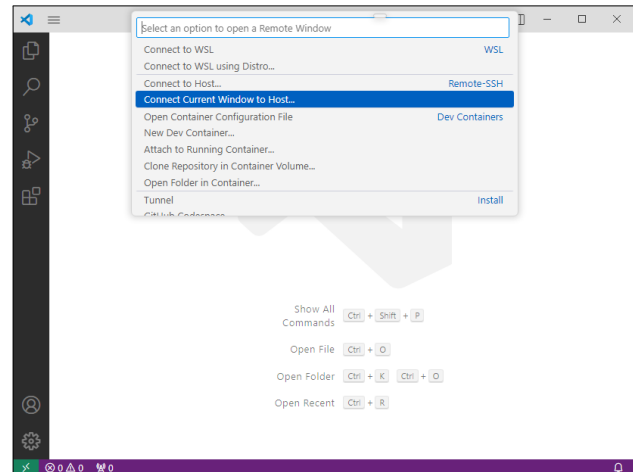
Una vez dentro debería ver una pantalla como la de la Figura 4(f). En la esquina inferior izquierda sobre fondo verde se indica el equipo al que está conectado y en la parte inferior hay un Terminal que le permite interactuar con el sistema operativo usando comandos estándar de Linux.

<sup>9</sup>En macOS es **⌘** + **⇧** + **P**.

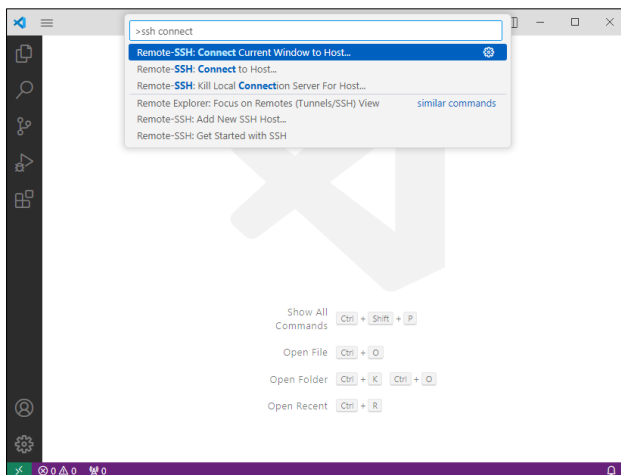




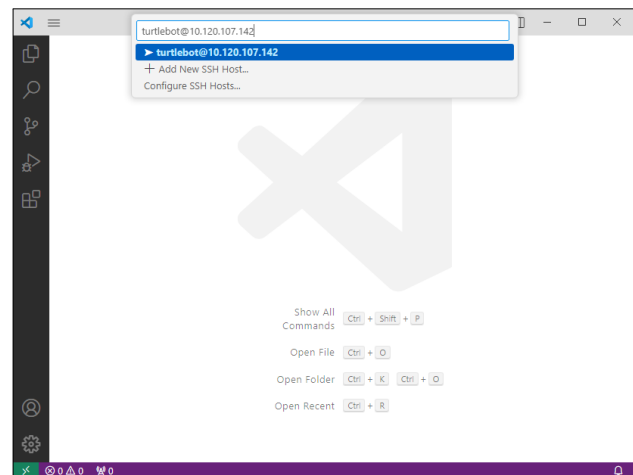
(a)



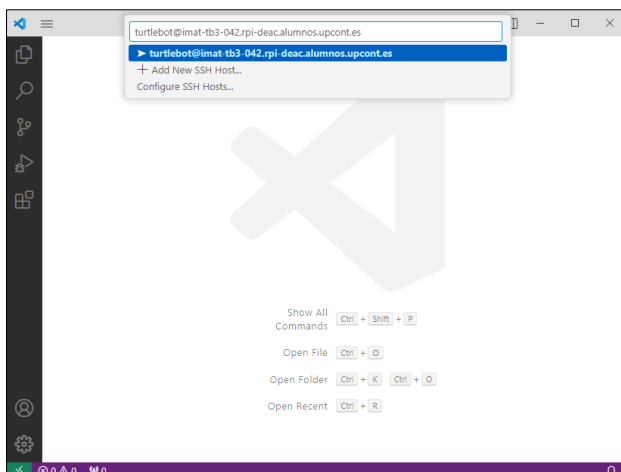
(b)



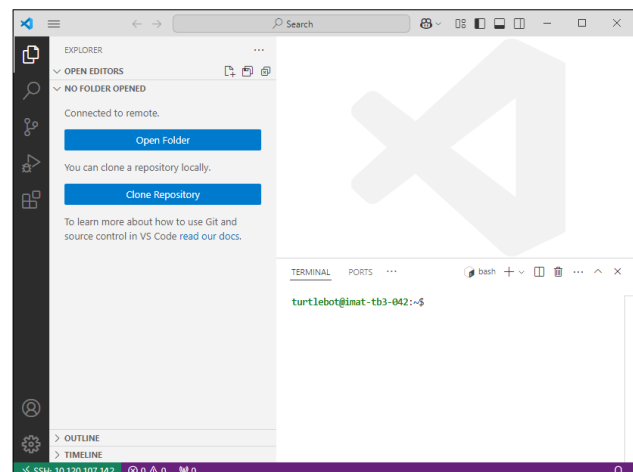
(c)



(d)



(e)



(f)

**Figura 4.** Pasos para conectarse por SSH desde Visual Studio Code. En la fila superior, botón para lanzar el menú de conexiones remotas (a) y opción que hay que seleccionar para iniciar una sesión de SSH (b). En (c) se muestra el diálogo equivalente a (b) que aparece si se usan atajos de teclado. (d) y (e) indican cómo introducir el usuario seguido de la dirección IP o el *hostname*, respectivamente. Finalmente, interfaz después de establecer la conexión (f).

## 5.2. El alumbramiento de una tortuga (1 punto)

Encienda su TurtleBot3 y conéctese por **escritorio remoto**. Desde un terminal —puede usar la aplicación Terminal o Tilix<sup>10</sup>, que es más cómodo por su habilidad para dividir pestañas en múltiples ventanas— ejecute el Código 11. En Sistemas Dinámicos se lanzaba automáticamente al encender el robot para que no tuviera que preocuparse por el funcionamiento de ROS 2 cuando aún era magia.

```
ros2 launch turtlebot3_bringup robot.launch.py
```

**Código 11.** Comando para lanzar los nodos que permiten interactuar con el *hardware* del TurtleBot3.

🔗 **Pregunta #10 (🔗):** Utilice las herramientas de terminal de ROS 2 y `rqt` para descubrir los nodos que operan el TurtleBot3, así como los publicadores y suscriptores que tienen. Explore también la estructura y el contenido de los mensajes e intente explicar con sus palabras para qué sirve cada nodo a partir de esa información. Ignore todos los temas (*topics*) que empiecen con el prefijo `rcl_`.

## 5.3. ¡Mira, mamá! ¡Estoy dentro del ordenador! (0,5 puntos)

A veces el terminal y `rqt` no son suficientes para entender bien qué es lo que el robot está viendo y haciendo. En ROS existe una aplicación adicional denominada RViz que permite, entre otras muchas cosas, cargar un modelo 3D del robot, visualizar las detecciones de los sensores y su pose con respecto al mapa del entorno en caso de que se disponga de uno. Puede abrirla escribiendo `rviz2` en un terminal. Robotis proporciona un segundo archivo de lanzamiento que abre RViz con la configuración necesaria para ver el contexto del robot (Código 12). Tenga en cuenta de que si no ha arrancado el robot en otro terminal con el Código 11, no se mostrará nada.

```
ros2 launch turtlebot3_bringup rviz2.launch.py
```

**Código 12.** Comando para abrir RViz y mostrar el modelo del TurtleBot3 Burger.

Si el robot se mueve, el modelo virtual lo hará de la misma forma pero... ¿dónde están los mandos? Al igual que sucedía con TurtleSim, podemos ejecutar un nodo que se encargue de la teleoperación utilizando el teclado. En un nuevo terminal escriba el Código 13 y a continuación mueva el robot. Comprobará como el modelo virtual responde de la misma manera y le indica dónde están los obstáculos en el entorno a partir de las medidas del LiDAR.

```
ros2 run turtlebot3_teleop teleop_keyboard
```

**Código 13.** Comando para ejecutar el nodo de teleoperación del TurtleBot3.

🔗 **Pregunta #11 (🔗):** ¿Qué criterio de signos utiliza el TurtleBot3 para la velocidad angular  $\omega$  por defecto? ¿Coincide con el habitual?

<sup>10</sup><https://gnunn1.github.io/tilix-web/>

## 5.4. Las tortugas bebé corren hacia el mar (3,5 puntos)

Para concluir, veamos si ha asimilado todo lo visto en la práctica. Cierre la sesión remota, conéctese por **SSH** desde Visual Studio Code y cree un espacio de trabajo llamado `tb3_ws` en la raíz del usuario `turtlebot` —`~` en el terminal, la carpeta `Home` en el explorador de archivos—. A continuación, añada dentro un paquete de Python con el nombre `amr_teleoperation` que contenga un nodo `teleoperation_node.py`, el cual será el encargado de comandar el robot publicando en el tema correspondiente. ¿Tiene claro ya cuál es?

En Python existen varios paquetes que facilitan la captura de las pulsaciones de teclado. Entre los más famosos están `pynput`<sup>11</sup> y `keyboard`<sup>12</sup>, aunque este último ya no se mantiene. Desafortunadamente, ninguno de los dos funciona a través de SSH, por lo que no son una alternativa viable. Utilizaremos `sshkeyboard`<sup>13</sup>, que solventa este problema pero tiene una pega: bloquea el hilo principal y, por tanto, impide la ejecución correcta de los publicadores y suscriptores de ROS 2. Vamos a convertir este inconveniente en una oportunidad para aprender aislando la lectura del teclado en un nodo independiente `keyboard_node.py` y creando nuestro propio mensaje personalizado. Los pasos que debe dar para conseguir operar remotamente el robot con su nodo de teleoperación son los siguientes:

1. **(0,5 puntos)** Siga el tutorial del siguiente enlace hasta el punto 2 inclusive para crear un mensaje personalizado, pero utilice `amr_msgs` como nombre del paquete en lugar del que sugiere el ejemplo. Para la estructura interna del mensaje, quizá lo más sencillo sea utilizar un `string`, dado que eso es lo que devuelve la librería `sshkeyboard`, pero siéntase libre de codificar los datos de otra manera.

<https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Single-Package-Define-And-Use-Interface.html>

2. **(0,75 puntos)** Programe un archivo `keyboard_node.py` que contenga un nodo encargado de publicar cada pulsación detectada por `sshkeyboard` —tendrá que consultar la documentación de la librería— utilizando el mensaje personalizado anterior.
3. **(1,5 puntos)** En `teleoperation_node.py`, codifique un segundo nodo que se suscriba a las pulsaciones del teclado, modifique las consignas de velocidad lineal y angular, y las publique en el tema por el que el TurtleBot3 espera recibirlas. Puede utilizar tantas teclas como quiera para tener mayor manejo del robot según las circunstancias. Por seguridad, convendría que al menos hubiera una tecla (¿la barra espaciadora por ser la más grande?) que permitiera detener el robot de inmediato.
4. **(0,75 puntos)** Como aún está aprendiendo a conducir el TurtleBot3, vamos a ponerle unos ruedines a su nodo de teleoperación para prevenir accidentes. Suscríbase al tema que publica el LiDAR y utilice los haces que estime oportunos para supervisar que los comandos que se generan desde el teclado no van a provocar una colisión. No olvide comprobar el perfil de calidad de servicio (QoS).

Aunque seguro que ha ido comprobando los nodos poco a poco, ha llegado el momento de divertirse y probar el sistema en conjunto. Lance todos los nodos y eche unas carreras por el laboratorio para verificar que el sistema de teleoperación supervisada funciona como debe. Preparados, listos, ¡ya!

🎥 **Video #1:** Grabe un video que demuestre el correcto funcionamiento del código implementado en este apartado y adjúntelo en el archivo comprimido con el espacio de trabajo.

<sup>11</sup><https://pynput.readthedocs.io/>

<sup>12</sup><https://pypi.org/project/keyboard/>

<sup>13</sup><https://sshkeyboard.readthedocs.io/>