



UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTIMENTO DI INGEGNERIA

Corso di Laurea in Computer Science Engineering

**Replacing USB cable with RFCOMM wireless
communication in MicroROS implementations**

Alessandro Giuffrè

ANNO ACCADEMICO 2024/2025

Indice

1	Abstract	1
2	Online documentation	1
3	Bluetooth + microROS implementation	3
3.1	Bluetooth/BLE protocol overview	3
3.1.1	Online documentation	3
3.2	General workflow	3
3.3	Setting up ROS2 communication	3
3.3.1	ROS 2 installation	4
3.3.2	Agent setup	4
3.3.3	Cube IDE project setup	5
3.3.4	Pins setup on Cube MX	6
3.3.5	Missing snippets	9
3.3.6	Running the communication	10
3.4	Setting up Bluetooth communication	12
3.4.1	Working with Bluetooth Module HC-05	12
3.4.2	Connecting to HC-05 on Windows 11	13
3.4.3	Connecting to HC-05 on Ubuntu 22.04.2	14
3.5	Implementing ROS communication through Bluetooth	17

List of Figures

1	MicroROS communication architecture	1
2	Visualization of working micro-ros agent	10
3	Active ros topics	11
4	Visualization of communication from subscriber's side	11
5	STM32F401RE connected to HC-05 through breadboard	12
6	Devices found by bluetooth	14
7	Error when launching "PuTTY" terminal	15
8	Terminal output when running ROS2 agent using bluetooth serial port	17
9	Setting USART1 to handle the communication	18

1 Abstract

The final goal is to achieve microROS communication through a non-traditional transport protocol, replacing so the USB cable with Bluetooth.

MicroROS communication works through a middleware called XRCE-DDS. DDS stands for "Data Distribution Service" which thanks to its way of communicating (publish-subscribe) it ensures that datas are exchanged with low latency, high reliability and high scalability.

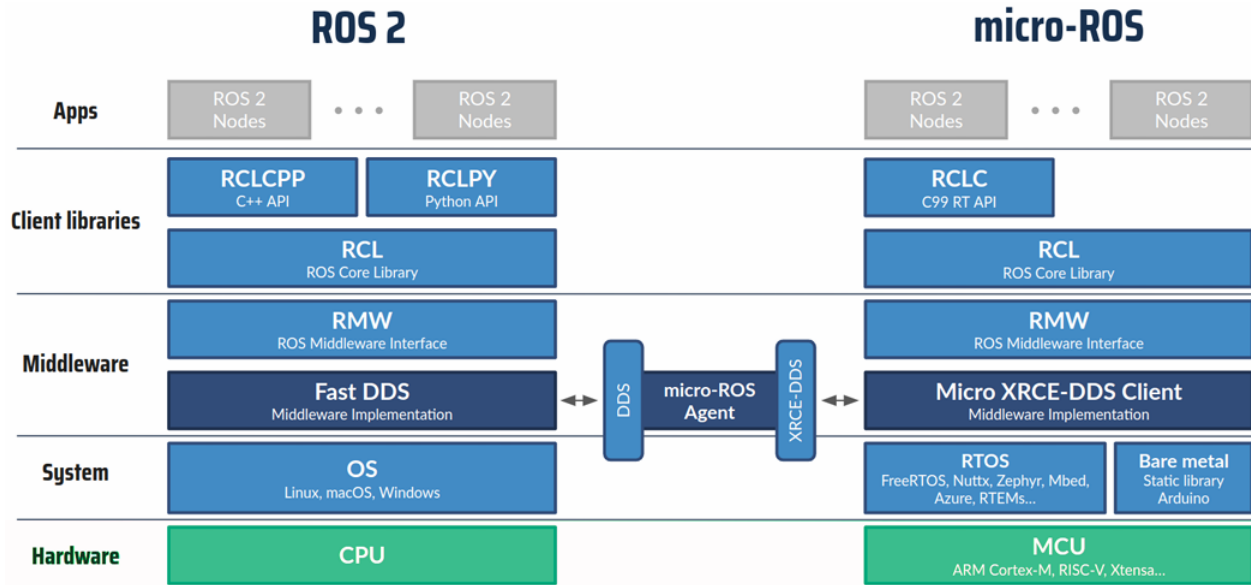


Figure 1: MicroROS communication architecture

With MicroROS it's possible to create **topics to which you can subscribe and from which you can retrieve datas**.

The advantage of this approach is going to be that once DDS is working through different transport protocols, it's possible to exchange datas between sensors, actuators and other devices keeping DDS advantages.

A way of doing that can be through a **central gateway** (generally a computer) which contains an interface for every protocol used. **The central idea is then to replace the usual serial communication with a wireless communication.**

2 Online documentation

MicroROS provides official documentation in order to implement custom transport protocols. All the informations about the implementations can be found in the official MicroROS documentaion:

<https://micro.ros.org/docs/tutorials/advanced/create-custom-transport/>.

A big role is played by **eProsima** which is the primary maintainer and contributor of Fast DDS, an implementation of the Data Distribution Service (DDS) protocol. EProsima provides examples and documentation with a focus on resource constrained environments. Useful informations can be found either on the following repository:

<https://github.com/eProsima/Micro-XRCE-DDS/tree/master>

or in the following website:

https://micro-xrce-dds.docs.eprosima.com/en/latest/transport.html?highlight=hdlc_custom-serial-transport

which also depicts examples for Packet-oriented transports and Stream-oriented transports.

3 Bluetooth + microROS implementation

3.1 Bluetooth/BLE protocol overview

Bluetooth implements wireless communication among nearby devices.

In order to exchange data between two or more devices it's necessary to make a **pairing** between them. The pairing will work in a way that one device is gonna be the **master**, which is gonna listen to nearby devices, and one or more devices are gonna be **slaves**, which are gonna **advertise** the master of their presence and give availability to connect.

It's not possible to connect 2 masters between them as well as connecting two slaves between them. **A master device though, can have multiple slaves** (normally 7 maximum) for data exchange.

Traditional bluetooth's connection establishes a **continuous** flux of datas among 2 devices. BLE (Bluetooth Low Energy), differently, establishes intermittent transmission of low energy datas which allows the possibility to differentiate the topology (creating **meshes**) and **ensures low power consumptions**.

Implementing MicroROS communication through Bluetooth **will lead to various advantages** as: remote configuration, management and maintainability of MicroROS nodes.

Furthermore, low energy communication and scalability are **keypoints** when working with IoT devices in any application.

BLE exchanges a **STK (Short Term Key)** in order to **pair** 2 devices and then uses it for encrypting new keys (called **secret keys**) which are going to be exchanged **in order to establish a connection**.

3.1.1 Online documentation

Online there is nothing that can be found about current applications, the only available contents are to be referred to the general documentation introduced in chapter 2.

3.2 General workflow

The workflow that has been followed in order to achieve the final task is to firstly **set up and execute a basic ROS2 communication through USB cable**. After understanding how this kind of communication works at the code level, it would be time to **implement a basic RFCOMM communication simulating a bluetooth serial transport**. The last task would then be to take what is needed from both contexts and set up a final project which will allow us to **implement ROS2 communication through bluetooth**.

3.3 Setting up ROS2 communication

For the first application it is first of all required to own a board, in this case a **NUCLEO STM32F401RE** has been used. Setting up the configuration can be achieved from a laptop on which some tools have to be installed.

Regarding ROS, ROS2 iron has to be installed on the laptop while **microROS library** has to be included in the project. The latter will be useful thanks to its API which will help managing ROS2 from board's side.

Regarding STMicroelectronics' supporting tools, it's needed to install **STM Cube MX 6.11** and **STM Cube IDE 1.15.1**. It is important to note that CubeMX comes with **FreeRTOS** which is a supporting tool offering real-time functionalities useful for managing all the needed operations. These tools can be easily downloaded from STMicroelectronics' official website.

Last but not least, it is important to note that **Linux Ubuntu 22.04.2** has been used in dual-boot with **Windows 11** in this application. This avoids some problems regarding Virtual Machines' difficulty to spot laptop's Bluetooth adapter.

3.3.1 ROS 2 installation

- The following commands are launched:

```
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.
key -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings
/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(./etc
/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt
/sources.list.d/ros2.list > /dev/null
sudo apt update && sudo apt install -y
libbullet-dev
python3-pip
python3-pytest-cov
ros-dev-tools
sudo apt update
sudo apt upgrade
sudo rosdep init
sudo apt install ros-iron-desktop
source /opt/ros/iron/setup.bash
```

3.3.2 Agent setup

In order to build and setup the agent:

- A folder called **agent_ws** is created, and inside it a folder called **src**.
- Inside this last one, the following repository is cloned:

```
git clone https://github.com/micro-ROS/micro_ros_setup.git
```

- From now on, it's followed the tutorial described inside the repository:

```
source /opt/ros/$ROS_DISTRO/setup.bash
mkdir uros_ws && cd uros_ws
git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git
src/micro_ros_setup
rosdep update && rosdep install --from-paths src --ignore-src -y
colcon build
source install/local_setup.bash
ros2 run micro_ros_setup create_agent_ws.sh
ros2 run micro_ros_setup build_agent.sh
```

3.3.3 Cube IDE project setup

In order to create a new project from scratch it is possible to go on **File -> New -> STM32 Project** and create a new project by selecting the name of the board actually using.

As mentioned earlier, it is needed to import microROS library utils in the project. This can be achieved by typing in a terminal:

```
git clone --branch humble https://github.com/micro-ROS/micro_ros_stm32cubemx_utils.git
```

- In the section

Project -> Settings -> C/C++ Build -> Settings -> Build Steps tab

and under

Pre-build steps

The following command is added:

```
docker pull microros/micro_ros_static_library_builder:iron && docker run
--rm -v $workspace_loc:$ProjName:/project --env MICROROS_LIBRARY_FOLDER =
micro_ros_stm32cubemx_utils/microros_static_library_ide microros/micro_ros
_static_library_builder:iron
```

- The path

../micro_ros_stm32cubemx_utils/microros_static_library_ide/libmicroros/include

is added in the section

Project -> Settings -> C/C++ Build -> Settings -> Tool Settings Tab -> MCU GCC Compiler -> Include paths

- In the section

Project -> Settings -> C/C++ Build -> Settings -> MCU GCC Linker -> Libraries

Under command menu

library search path (-L)

the following path can be added:

../micro_ros_stm32cubemx_utils/microros_static_library_ide/libmicroros

- Then it can be added

microros

in the section

Libraries (-l)

- In the end, on the newly project's folder are added some files that can be found on the folder `micro_ros_stm32cubeux_utils -> extra_sources`.

In particular, the following files are moved:

`"extra _sources /microros _time.c";`

`"extra _sources /microros _allocators.c";`

`"extra _sources /custom _memory _manager.c";`

`"extra _sources /microros _transports /dma _transport.c";`

To complete the setup of microROS, the following video becomes helpful <https://www.youtube.com/watch?v=xbWaHARjSmk>

In particular, the steps useful for **configuring the pins and installing the agent** are followed. The agent will be a sort of **Man On the Middle for communication**.

It is important to setup both Cube IDE and Cube MX. After completing the setup of the first, **It is possible to create a new project on the second starting from the embedded.ioc file** (generated by the IDE).

3.3.4 Pins setup on Cube MX

So let's start with the setup in this tool too. It's immediately noticeable that the version of the ST tool currently used is different from the one shown in the tutorial, for example the simple section **Middleware and software packages** present in the in-use tool is only called **Middleware** in the version of the tool used in the tutorial. For the moment the task is reduced on trying to replicate the changes shown by disentangling the process between the different settings and verifying if everything works.

- In the

pinout and configuration section

going to

System Core -> RCC

it's possible to insert

Crystal/Ceramic resonator in High speed clock (HSE).

- Then in the

Middleware and software packages section

it's possible to **install FreeRTOS** (if not already installed).

In the version of Cube MX currently used, it appears as *Xcube-FREERTOS 1.2.0*. This library is not installed and therefore are selected the items specified in the tutorial to proceed, in particular:

`TZ_non_supported`

in the

`Core` item

and

`Heap_1`

in the

`Heap` item.

The choice of the first was mandatory because it is the only available choice, the choice of the heap was arbitrary.

- In the
`CMSIS_RTOS2` section
which opens the
`Task and queues` section
it's possible to modify the task by inserting
`stack size = 3000`
and allocation: `static`.

It's chosen a stack size = 3000. The stack size is calculated in 4 bytes words so by doing so about 12kb are allocated, in fact in github it was expressed the need to have a stack size of at least 10kb for correct operation.

- After that, In the
`Connectivity` section
and in the
`USART3` item
it's selected
`mode: asynchronous`.

- The video shows a different procedure from the one carried out till now because as already said it uses an older version of the tool, in the current version whenever the
`DMA settings`
is clicked, it sends directly to
`GPDMA1`
of the
`System Core` section.

Here can be chosen

`channel 1 { 2 words internal FIFO and channel 3 { 2 words internal FIFO`

in order to reproduce the steps of the video as faithfully as possible.

Are set respectively in the

Request field of the Request configuration board: `usart 3 rx` and `usart 3 tx`.

Only in the

`RX`

it's set the

`circular mode`.

It's set a `very high` priority in both.

- Now it's possible to go back to

`USART3`

where it's possible to see the two channels and also the

`USART3 global interrupt`

that must be set to

`enabled`.

- After that, in the

`Connectivity` section

and in the

`USART3` item

it's selected

`mode: asynchronous`.

- Finally, in the

`System Core` board,

in the

`SYS` section

can be set

`TIM1` as `timebase source`.

- In the

`Project Manager` section,

a project called `f401_microros` it's created

`makefile`

it's selected in the

`toolchain/IDE` item.

- Finally, in the

`Code generator` section

can be activated the

`Generate peripheral initialization` as a pair of `' .c/.h '` files per peripheral item.

It is now possible to generate the code.

To make sure that ros2 works correctly, it's needed to open a new terminal and enter the command:

```
ros2 topic list
```

- . An empty string appears, which means that **ros2 is not activated**.
- To make ros2 work it's possible to enter the command

```
gedit .bashrc
```

- and as the last line of the file insert:
source /opt/ros/iron/setup.bash.
- Now trying to re-launch the command `ros2 topic list` can be seen:
`/parameter _events` and `/rosout`.

3.3.5 Missing snippets

By comparing the actual generated code with the generated code from other working projects, some lines essential for the correct development of the tool appear to be missing. They are implemented on **main.c**:

```
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rcl/rclc.h>
#include <rclc/executor.h>
#include <uxr/client/transport.h>
#include <rmw_microxrcedds_c/config.h>
#include <rmw_microros/rmw_microros.h>
#include <std_msgs/msg/int32.h>
#include <geometry_msgs/msg/twist.h>
bool cubemx_transport_open(struct uxrCustomTransport * transport);
bool cubemx_transport_close(struct uxrCustomTransport * transport);
size_t cubemx_transport_write(struct uxrCustomTransport * transport,
const uint8_t * buf, size_t len, uint8_t * err);
size_t cubemx_transport_read(struct uxrCustomTransport * transport,
uint8_t* buf, size_t len, int timeout, uint8_t* err);
void * microros_allocate(size_t size, void * state);
void microros_deallocate(void * pointer, void * state);
void * microros_reallocate(void * pointer, size_t size, void * state);
void * microros_zero_allocate(size_t number_of_elements,
size_t size_of_element, void * state);
void (* rclc_subscription_callback)(const void *);
void subscription_callback(const void * msgin){
    // Cast received message to used type
```

```

const geometry_msgs__msg__Twist * msg =
(const geometry_msgs__msg__Twist *)msgin;
HAL_GPIO_TogglePin(LD2_GPIO_Port , LD2_Pin);
// Process message
printf(" Received:  \n", msg->linear.x);
}
void timer_callback(rcl_timer_t * timer, int64_t last_call_time){
    RCLC_UNUSED(last_call_time);
}

```

3.3.6 Running the communication

It is now possible to flash the code on STM cube IDE. It's important in this case to spot the name of the USB port where the board is connected. In this case, the port's name is `/dev/ttyACM0`.

After flashing the code, it is time to **launch the agent** from a new terminal. More specifically, everytime a new terminal is opened, it is needed to surf into the `"agent_ws"` folder previously created and launch the following commands:

```

colcon build
source install/local_setup.bash
ros2 run micro_ros_agent micro_ros_agent serial -b 115200 --dev /dev/ttyACM0

```

```

leg3214@leg3214-virtual-machine:~/agent_ws/src$ ros2 run micro_ros_agent micro_ros_agent serial -b 115200 --dev /dev/ttyACM0
1727522539.286318] info | TermiosAgentLinux.cpp | init | running... | fd: 3
1727522539.286705] info | Root.cpp | set_verbose_level | logger setup | verbose_level: 4
1727522554.183977] info | Root.cpp | create_client | create | client_key: 0x5851F42D, session_id: 0x81
1727522554.184051] info | SessionManager.hpp | establish_session | session established | client_key: 0x5851F42D, address: 0
1727522554.268997] info | ProxyClient.cpp | create_participant | participant created | client_key: 0x5851F42D, participant_id: 0x000(1)
1727522558.858976] info | SessionManager.hpp | establish_session | session re-established | client_key: 0x5851F42D, address: 0
1727522558.889582] info | ProxyClient.cpp | create_topic | topic created | client_key: 0x5851F42D, topic_id: 0x000(2), participant_id: 0x000(1)
1727522558.900406] info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x5851F42D, publisher_id: 0x000(3), participant_id: 0x000(1)
1727522558.914289] info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x5851F42D, datawriter_id: 0x000(5), publisher_id: 0x000(3)
1727522558.933856] info | ProxyClient.cpp | create_topic | topic created | client_key: 0x5851F42D, topic_id: 0x001(2), participant_id: 0x000(1)
1727522558.944542] info | ProxyClient.cpp | create_subscriber | subscriber created | client_key: 0x5851F42D, subscriber_id: 0x000(4), participant_id: 0x000(1)
1727522558.958677] info | ProxyClient.cpp | create_datareader | datareader created | client_key: 0x5851F42D, datareader_id: 0x000(6), subscriber_id: 0x000(4)

```

Figure 2: Visualization of working micro-ros agent

Everything seems working. By launching the command

```
ros2 topic list
```

it is possible to check the communication, it's indeed obtained:

```
/cmd_vel
/cubemx_publisher
/parameter_events
/rosout
```

Figure 3: Active ros topics

Ros2 is able to find the publisher initialized in the default task. So if now a new terminal opened and the following command launched:

```
ros2 topic echo /cubemx_publisher
```

The result is to see some progressive numbers which are a proof of the correct functioning of the setup (figure 4).

```
aleg3214@aleg3214-virtual-machine:~$ ros2 topic echo /cubemx_publisher
data: 319
---
data: 320
---
data: 321
---
data: 322
---
data: 323
---
data: 324
---
data: 325
---
data: 326
---
data: 327
---
data: 328
---
data: 329
```

Figure 4: Visualization of communication from subscriber's side

3.4 Setting up Bluetooth communication

The in-use board's manual, **shows that Bluetooth it's not built-in**. In order to implement this functionality it's then needed to use an external hardware which is a **HC-05 Bluetooth module**.

3.4.1 Working with Bluetooth Module HC-05

Online, can be found a tutorial on how to make HC-05 communicate with the mobile application **"Serial Bluetooth Terminal"**:

<https://www.youtube.com/watch?v=ikIdYuI0hnE>

which is followed step by step.

It's important to not that in the video is shown an application connecting the smartphone with a different board (**STM32F103RB**) from the actually one in use. The steps have been reproduced as shown, taking care of the differences with the board actually using.

Overall, the application is pretty linear: the first step is to **create a new project on STM Cube MX** and setup the pins of the board in order to have a transmission trough **UART**.

The idea is to **initialize an RX (receiver) and a TX (transmitter)** which are then going to communicate with the **RX and TX of the bluetooth module**.

Once pins are set up, the **physical connection has to be made**: according to the used pins, it's necessary to link board's TX and RX with module's TX and RX and connect 2 more wires between grounds and between VCCs. In this case 5V pin is used to power on the module. **Figure 5** shows the devices in use.

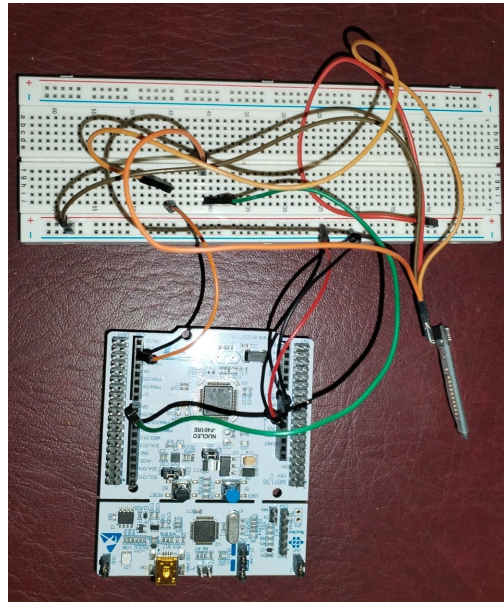


Figure 5: STM32F401RE connected to HC-05 through breadboard

Once the setup is completed and pins are connected, the tutorial shows an easy implementation of a function to check the correct functioning of the communication which includes **turning on a**

LED on the board when pressing 'O' in the mobile application and turning the same LED off when pressing 'X'.

Setting it up it's simple, the first thing to do is to pair the smartphone with the module using the normal bluetooth interface of the smartphone. After pairing it, **HC-05 module** can be found on the mobile application as well among the available devices. As soon as it is clicked from the app, the connection is started.

It's possible to customize macro buttons in order to insert 'O' and 'X' which are gonna be the buttons that allow us to verify the correct functioning of the communication.

3.4.2 Connecting to HC-05 on Windows 11

The process of establishing Bluetooth communication between the HC-05 module and a laptop builds upon the initial setup performed with a smartphone. **Following, are described the configuration steps carried out to achieve the communication.** All procedures were performed on a system running Windows 11, focusing on adjusting system settings, pairing the HC-05 module, and utilizing terminal software to control the connection.

System Configuration in Windows 11

To enable the laptop to detect and communicate with the HC-05 module, **certain adjustments are required in the system's Bluetooth settings.**

The configuration process began by navigating to the **Settings** application within the operating system and accessing the **Bluetooth and devices** section.

Within this interface, the following actions are performed:

- In the **Other Bluetooth settings** menu, the option "Allow Bluetooth devices to discover this PC" is enabled.
- Under the **Device discovery** option in the **Devices** menu, the setting is adjusted to **Advanced**.

These adjustments ensure that the **laptop is properly configured to be discoverable by external devices** such as the HC-05 module.

Device Pairing and Software Integration

After modifying the system settings, the **HC-05 module is made discoverable** (by ensuring the led blinks fast) and successfully paired with the laptop through the *Add device* function.

At this point, a **terminal application is required to interact with the HC-05 module.** For this purpose, **PuTTY**, a widely used terminal emulator, is selected.

PuTTY is configured to establish a serial connection with the HC-05 module. The required parameters are a **baud rate of 9600** (according to the flashed code in the board) and to **enable local echo** in order to see a response from the terminal. The latter is achieved by navigating to the **Terminal** tab within the software settings and setting the **Local echo** option to "Force on".

These adjustments will allow the terminal to communicate effectively with the HC-05 module, ensuring proper data transmission and reception.

Determining the COM Port and Establishing Connection

To finalize the setup, **the COM port associated with the HC-05 module needs to be identified**. This is achieved through the *Control Panel* by accessing *Device Manager* -> *Ports (COM and LPT)*, where the assigned COM port number is noted. The identified port is then given in input to *PuTTY*, completing the connection setup.

Once the connection is established, the HC-05 module is successfully controlled through the terminal. Specifically, sending the command '0' will turn the LED connected to the module on, while sending 'X' will turn it off.

This mirrors the functionality achieved during the smartphone setup, confirming that the HC-05 module is operational across both devices.

3.4.3 Connecting to HC-05 on Ubuntu 22.04.2

ROS2 with the relative agent, which is gonna connect with microROS, it's installed on **Linux Ubuntu 22.04.2**.

The whole process of setting up bluetooth communication then **must be done on this operative system as well**.

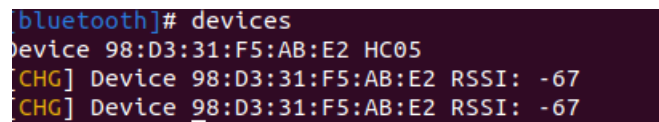
The main focus is to recreate the bluetooth connection through HC-05 module as well as what has been done on Windows.

As soon as the project on STM Cube IDE is imported and built, the code can be run without problems and flashed inside the NUCLEO board. **HC-05 now blinks fast which means it is ready to be paired**.

After completing the setup phase, it's time to explore further the functionalities offered by linux. First of all it's important to note that **laptop's bluetooth adapter correctly spots HC-05 bluetooth module**, this can be verified by launching the following commands:

```
sudo systemctl start bluetooth
bluetoothctl
scan on
devices
```

Which results are shown in **figure 6**.



```
bluetooth]# devices
Device 98:D3:31:F5:AB:E2 HC05
[CHG] Device 98:D3:31:F5:AB:E2 RSSI: -67
[CHG] Device 98:D3:31:F5:AB:E2 RSSI: -67
```

Figure 6: Devices found by bluetooth

After HC-05 is found, it's possible to pair to it, trust it and bind it a specific serial port (which is interpreted as a serial device from linux) from which it can be recognizable for future configurations.

By default it's not necessary to always assign it to a specific port but, during the configuration process, problems connecting to the native given port has been found so it has been decided to bind the bluetooth module with port **rfcomm0** to avoid complications.

```
pair 98:D3:31:F5:AB:E2
trust 98:D3:31:F5:AB:E2
exit
sudo rfcomm bind /dev/rfcomm0 98:D3:31:F5:AB:E2
```

In case native port wants to be used, the following command can be launched:

```
sudo dmesg | grep tty
```

The output will reveal that the port is assigned and that can be used for further configurations.

Once the pairing has been done and the port is known, it's possible to reproduce what has been done on Windows 11: **a terminal is going to be installed in order connect HC-05 module to the laptop and launch commands through bluetooth.**

After checking once more the presence of the serial port, a terminal is run passing as parameters the name of the port and the baud rate (remind that the baud rate has been set on the code flashed on the NUCLEO board).

Looking online, many terminals can be found. Once again, PuTTY has been used.

The installation is done by:

```
sudo apt install putty
putty
```

Once it's run though, the following errors appear:



```
leg3214@aleg3214-Prestige-15-A125C:~$ putty
(putty:5578): Gtk-CRITICAL **: 16:59:44.254: gtk_box_gadget_distribute: assertion 'size >= 0' failed
in GtkScrollbar
(putty:5578): Gtk-CRITICAL **: 16:59:44.254: gtk_box_gadget_distribute: assertion 'size >= 0' failed
in GtkScrollbar
(putty:5578): Gtk-CRITICAL **: 16:59:44.255: gtk_box_gadget_distribute: assertion 'size >= 0' failed
in GtkScrollbar
PuTTY: unable to load font "server:fixed"
```

Figure 7: Error when launching "PuTTY" terminal

The error looks like it's coming from the UI, more precisely from the font. It's indeed possible to solve it by going opening again PuTTY, going to **Category -> Window -> Fonts** and replace **server:fixed** with another one (ex. *Noto Mono*). Now everything works fine.

Some commands are launched in order to ensure that the bluetooth module is effectively communicating with the laptop:

```
sudo rfcomm release all
sudo rfcomm bind /dev/rfcomm0 98:D3:31:F5:AB:E2
ls -l /dev/rfcomm0
sudo echo "Test" > /dev/rfcomm0
```

After launching the `ls` command, the device is seen. After launching the last command, this error appears:

bash: /dev/rfcomm0: Permission denied

For any reason the command cannot be launched by the root user, this permission problem is solved launching these commands:

```
ls -l /dev/rfcomm0
sudo usermod -a -G dialout $USER
groups
```

After the first command it's possible to check how the device is owned by the **dialout group**. It's then needed to add the user to dialout and reboot the system. After the reboot, when launching "groups" command, dialout can be seen listed.

If the echo command is run again, now the terminal echoes with "Test".

With further reasearches, **the correct communication is accomplished even from raw commands sent to the terminal**. In order to achieve that, it's needed to launch the following command on a terminal:

```
cat /dev/rfcomm0
```

This will be the terminal for receveing datas.

It's now needed to open a new terminal for sending datas and launch these commands in order to either turn on off on the LED:

```
echo -n "0" > /dev/rfcomm0
echo -n "X" > /dev/rfcomm0
```

As expected, LED is toggled accordingly.

3.5 Implementing ROS communication through Bluetooth

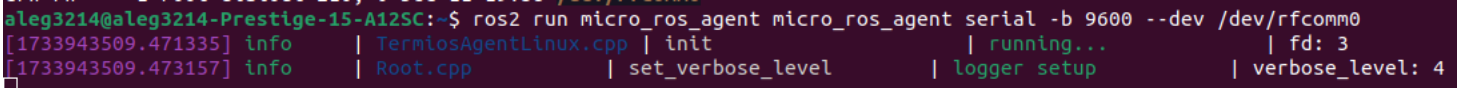
At the moment, there are 2 stand-alone working projects both for microROS and Bluetooth communication.

The idea at this point would be to replace the USB cable which is connecting the NUCLEO board with the laptop and replacing it with the bluetooth connection already tested which simulates a serial transport. When running the scripts for microROS communication, it's needed to explicitly set in the command the baud rate and the name of the port to be used. If previously, the port to which the USB cable was written, now the serial bluetooth port is going to be used.

ROS' command for running the communication can be then launched like this:

```
ros2 run micro_ros_agent micro_ros_agent serial -b 9600 --dev /dev/rfcomm0
```

It's needed to specify the baud rate (9600 in this case since the bluetooth module HC-05 has been tested on this baud rate) and the name of the serial port. In this case `/dev/rfcomm0` is the port binded with HC-05 module.



```
aleg3214@aleg3214-Prestige-15-A125C:~$ ros2 run micro_ros_agent micro_ros_agent serial -b 9600 --dev /dev/rfcomm0
[1733943509.471335] info | TermiosAgentLinux.cpp | init | running... | fd: 3
[1733943509.473157] info | Root.cpp | set_verbose_level | logger setup | verbose_level: 4
```

Figure 8: Terminal output when running ROS2 agent using bluetooth serial port

It's clear that the script does not work properly since it does not initialize all the required functions.

Some debugging has been done on the code which lead to **problems regarding the communication initialization**.

In order to enable ROS 2 to deal with bluetooth connection, it's needed to make some reasonings about the means of communication:

- **In the traditional ROS2 application, only one USART** was enabled on the pinout configuration. This USART was used for letting the agent connect the laptop with the board through the USB cable.
- **In the bluetooth application**, one USART was used to exchange data received from bluetooth, connecting HC-05 bluetooth module with the nucleo.
- **In the latest case scenario** in which we wanna implement the custom transport (bluetooth) keeping ROS 2 functionalities, sounds meaningful to enable both USARTs at the same time: one would connect the bluetooth module with the nucleo and the other one would connect the laptop with the bluetooth module.

This idea is pursued and developed, leading to persistent errors and **impossibility to come to a concrete result**.

After going deeper into the code and the functions implementing both the scripts uniquely for bluetooth connection and microros connection, **a new idea popped out**.

The bluetooth script, at code level, does not implement any function nor library to manage the connection between the module and the laptop. Everything it's done through some commands launched in the terminal. These commands **simulate a serial connection**, meaning that the laptop thinks something is connected through a normal USB cable so **it doesn't bother to manage it**. This key concept is implemented, by making the process more straight forward.

The steps made in order to execute it are:

- **Remove one USART** (the one which was supposed to connect the bluetooth module with the laptop) and setup appropriately the only one remaining as followed:
 - The only needed USART would be the one connecting RXs and TXs of the nucleo and the module among them;
 - To set it up properly, it's needed to go on **STM cube MX**, in the section **Connectivity** -> **USART1** -> **Mode** and set **Asynchronous**;
 - In **configuration tab** -> **DMA Settings** it's needed to add both **USART1_RX** and **USART1_TX**. In particular, **USART1_RX** requires **Mode Circular**;
 - In the tab **NVIC Settings**, **USART1 global interrupt** has to be **enabled**;
 - Finally, ensure a **baud rate = 9600** is set.
- **At code level**, by going in the file **freertos.c** and in particular in the function **void StartDefaultTask()** it's needed to replace the parameter passed to the function **rmw_uoros_set_custom_transport()** with **huart1** as shown in **figure 9**:

```
void StartDefaultTask(void *argument)
{
    /* USER CODE BEGIN StartDefaultTask */
    // micro-ROS configuration

    rmw_uoros_set_custom_transport(
        true,
        (void *) &huart1,
        cubemx_transport_open,
        cubemx_transport_close,
        cubemx_transport_write,
        cubemx_transport_read);
}
```

Figure 9: Setting USART1 to handle the communication

- From terminal, **bind the bluetooth module with /dev/rfcomm0** as done before many times;
- Flash the code, disconnect the nucleo from the laptop and connect it somewhere else (in order to be sure bluetooth as being used) and run microros setting the right port and the right baud rate.

After running the code it might be necessary to click the **reset button** on the board to correctly let ROS2 initialize all the functions needed.

It's now possible to create new nodes, publish data into them and read the datas by subscribing to them.