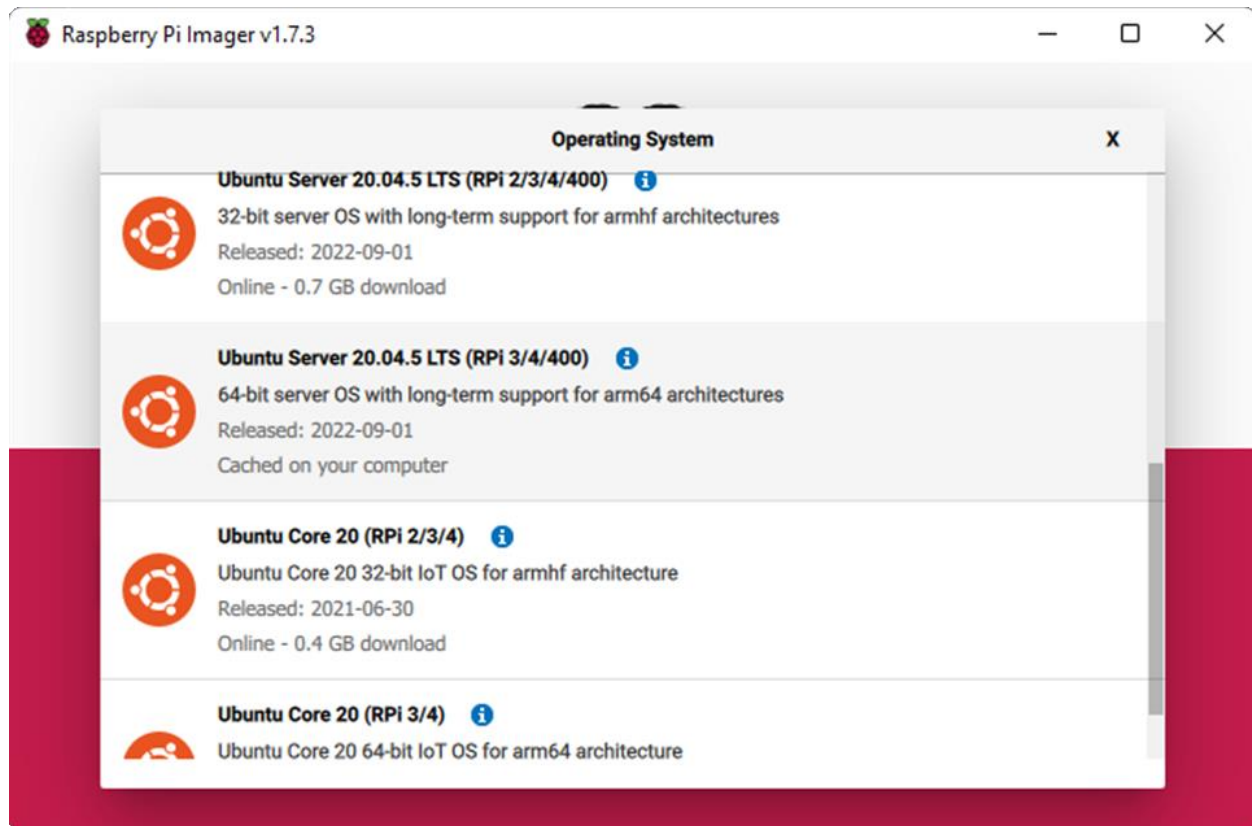


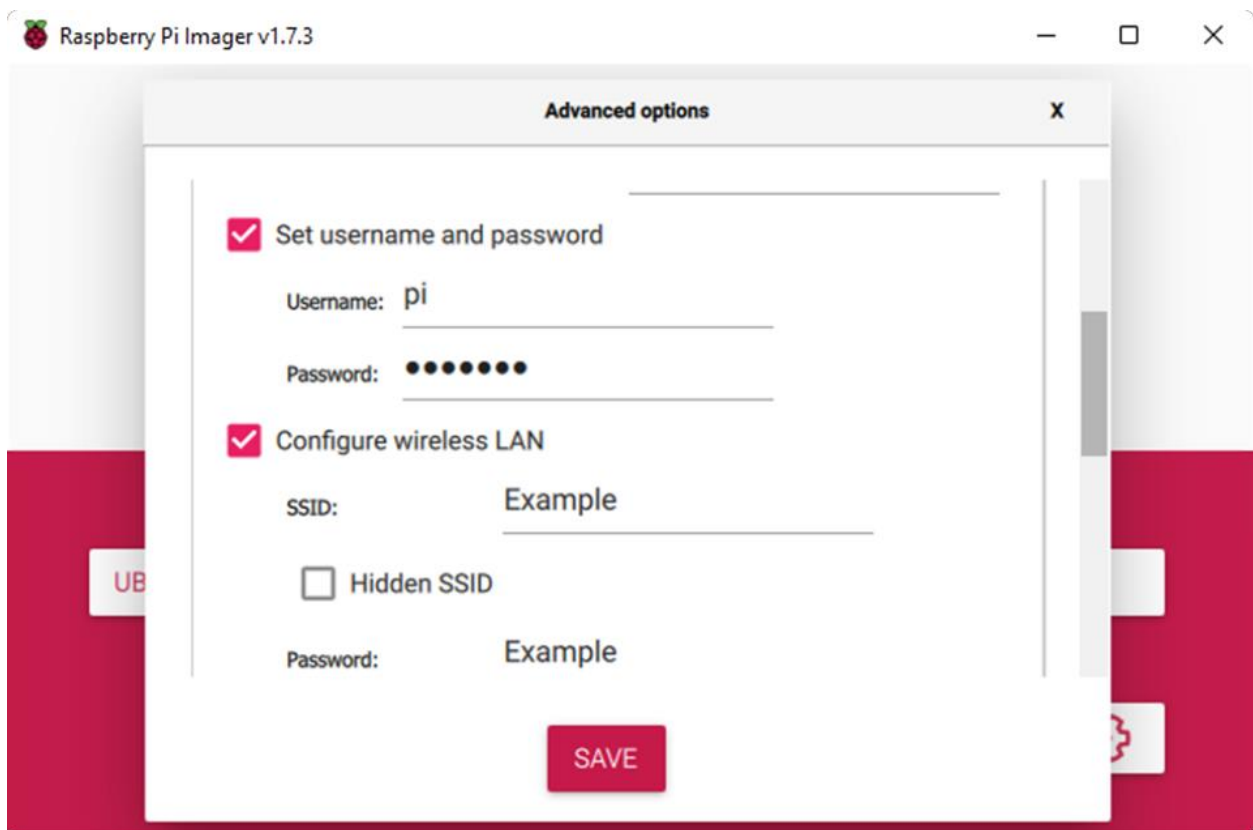
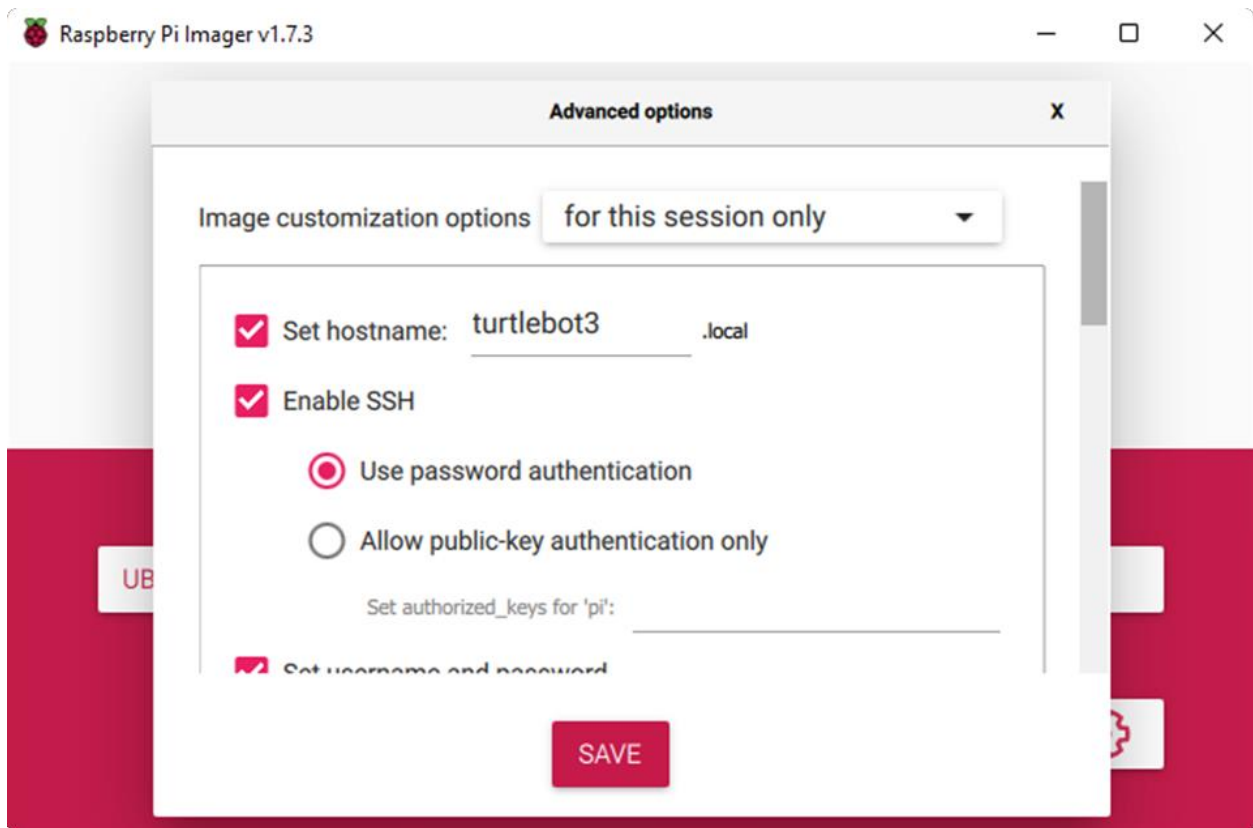
Mise en place de l'architecture ROS2 sur Raspberry Pi 3B+

L'installation commence par flasher une image d'Ubuntu Server 20.04 64-bit.

Sur la Raspberry 3B+, la version Desktop est à éviter car l'interface graphique demande beaucoup de ressources à la carte, même si celle-ci est allégée (xubuntu ou lubuntu desktop).



On peut, afin de ne pas avoir besoin de travailler sur la Raspberry, programmer directement l'accès SSH, le nom d'utilisateur, le mot de passe ainsi que la connexion sans-fil afin de pouvoir prendre accès à distance dès la mise en route.



Puis on flash la carte SD après avoir choisi le support.

Si les pré-paramétrages Wi-Fi n'ont pas fonctionné, il est toujours possible de raccorder la carte au réseau Ethernet et de la retrouver avec un scan d'ip avant de reparamétrer le Wi-Fi.

?	192.168.1.16			
	192.168.1.17	132 ms	turtlebot3.home	[n/a]
?	192.168.1.18			

Pour reparamétrer le Wi-Fi il suffira de rentrer la commande

```
$ ls /etc/netplan
```

pour connaître le nom du fichier réseau, puis de le modifier avec

```
$ sudo nano
```

```
pi@turtlebot3: ~  
pi@turtlebot3:~$ ls /etc/netplan  
50-cloud-init.yaml  
pi@turtlebot3:~$ sudo nano /etc/netplan/50-cloud-init.yaml  
pi@turtlebot3:~$
```

```
pi@turtlebot3: ~  
GNU nano 4.8 /etc/netplan/50-cloud-init.yaml Modified  
# This file is generated from information provided by the datasource. Changes  
# to it will not persist across an instance reboot. To disable cloud-init's  
# network configuration capabilities, write a file  
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:  
# network: {config: disabled}  
network:  
  version: 2  
  wifis:  
    renderer: networkd  
    wlan0:  
      access-points:  
        Example_SSID:  
          password: Example_Password  
      dhcp4: true  
      optional: true  
  
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos  
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

Enfin, on valide avec :

```
$ sudo netplan --debug apply
```

```
pi@turtlebot3: ~  
pi@turtlebot3:~$ sudo netplan --debug apply  
** (generate:3361): DEBUG: 20:37:59.169: starting new processing pass  
** (generate:3361): DEBUG: 20:37:59.170: wlan0: adding wifi AP 'Ohmega'  
** (generate:3361): DEBUG: 20:37:59.179: We have some netdefs, pass them through  
a final round of validation  
** (generate:3361): DEBUG: 20:37:59.179: Configuration is valid  
** (generate:3361): DEBUG: 20:37:59.183: Generating output files..  
** (generate:3361): DEBUG: 20:37:59.183: Creating wpa_supplicant config  
** (generate:3361): DEBUG: 20:37:59.184: wlan0: Creating wpa_supplicant configur  
ation file run/netplan/wpa-wlan0.conf  
** (generate:3361): DEBUG: 20:37:59.184: Creating wpa_supplicant unit /run/syste  
md/system/netplan-wpa-wlan0.service  
(generate:3361): GLib-DEBUG: 20:37:59.185: posix_spawn avoided (workdir specifie  
d) (fd close requested)  
** (generate:3361): DEBUG: 20:37:59.332: Creating wpa_supplicant service enablem  
ent link /run/systemd/system/systemd-networkd.service.wants/netplan-wpa-wlan0.se  
rvice  
** (generate:3361): DEBUG: 20:37:59.333: openvswitch: definition wlan0 is not fo  
r us (backend 1)  
** (generate:3361): DEBUG: 20:37:59.334: NetworkManager: definition wlan0 is not  
for us (backend 1)  
(generate:3361): GLib-DEBUG: 20:37:59.334: posix_spawn avoided (fd close request  
ed)  
(generate:3361): GLib-DEBUG: 20:37:59.392: posix_spawn avoided (fd close request
```

Une fois connecté en SSH on arrive donc sur le terminal de la carte. On peut passer le clavier en azerty à l'aide de :

```
$ sudo loadkeys fr
```

L'installation de ROS2 m'as été donnée par

(https://www.youtube.com/watch?v=AmULiA840fA&ab_channel=Ubuntu)

(<http://docs.ros.org.ros.informatik.uni-freiburg.de/en/foxy/Installation/Ubuntu-Install-Debian.html>)

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

```
$ sudo apt-key adv --fetch-keys https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc
```

```
$ sudo apt-add-repository http://packages.ros.org/ros2/ubuntu
```

Suite à cet ajout de domaine nous pouvons installer la base ROS2 que nous utiliserons afin de contrôler et améliorer la turtlebot 3.

```
$ sudo apt install ros-foxy-ros-base
```

Puis nous l'ajoutons aux sources afin que ses packages soient reconnus dans les terminaux :

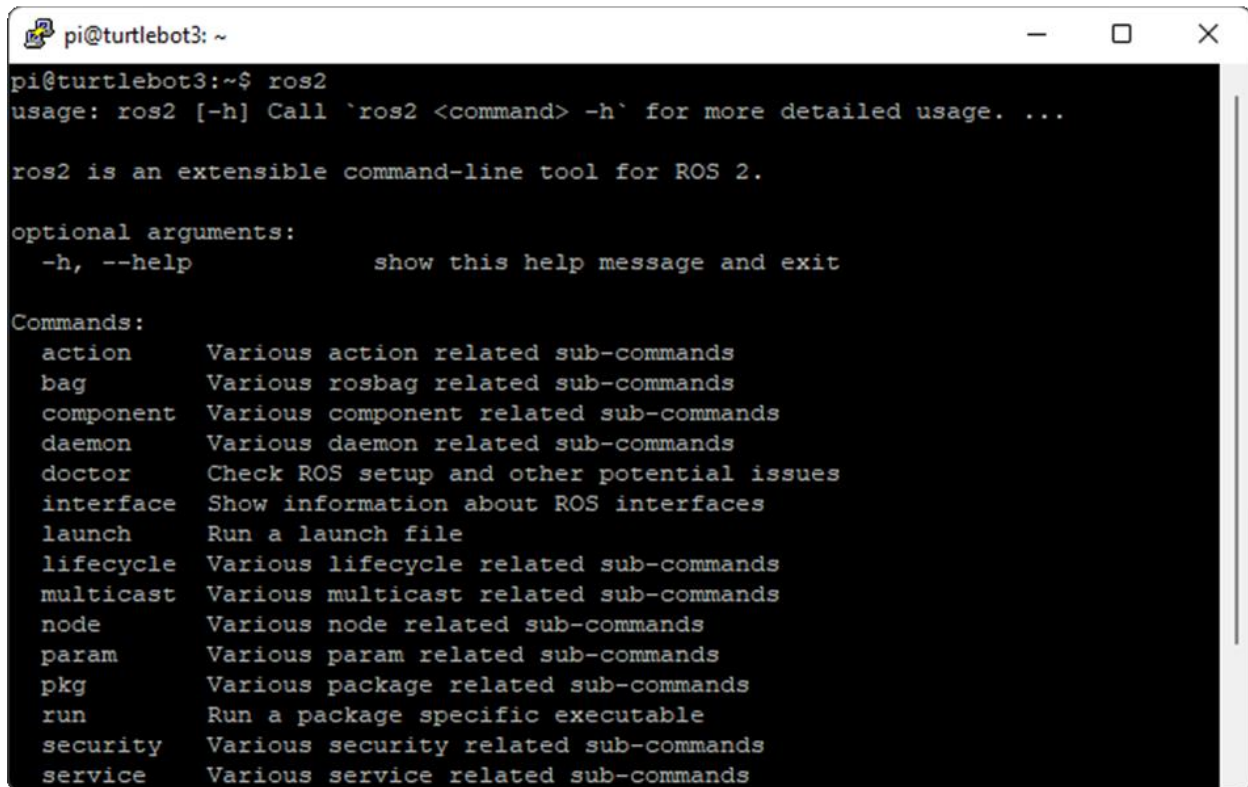
```
$ echo "source /opt/ros/foxy/setup.bash" >> ~/.bashrc
```

Relancer le terminal ou effectuer :

```
$ source ~/.bashrc
```

Et vérifier que ROS2 est bien installé en utilisant :

```
$ ros2
```



```
pi@turtlebot3:~$ ros2
usage: ros2 [-h] Call `ros2 <command> -h` for more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

optional arguments:
  -h, --help            show this help message and exit

Commands:
  action      Various action related sub-commands
  bag         Various rosbag related sub-commands
  component   Various component related sub-commands
  daemon      Various daemon related sub-commands
  doctor       Check ROS setup and other potential issues
  interface    Show information about ROS interfaces
  launch      Run a launch file
  lifecycle    Various lifecycle related sub-commands
  multicast    Various multicast related sub-commands
  node         Various node related sub-commands
  param        Various param related sub-commands
  pkg          Various package related sub-commands
  run          Run a package specific executable
  security     Various security related sub-commands
  service      Various service related sub-commands
```

J'ai préféré ensuite ajouter les dépendances de ROS ainsi que les packages qui nous aideront par la suite :

```
$ sudo apt install python3-argcomplete
```

```
$ sudo apt install python3-colcon-common-extensions
```

```
$ sudo apt-get install ros-foxy-gazebo-*
```

```
$ sudo apt install ros-foxy-cartographer
```

```
$ sudo apt install ros-foxy-cartographer-ros
```

```
$ sudo apt install ros-foxy-navigation2
```

```
$ sudo apt install ros-foxy-nav2-bringup
```

Ces packages ne sont probablement pas nécessaires mais les inclure me permet d'effectuer plus de tests sur le robot sans risquer des erreurs de compilation.

Et enfin, nous installons et sourçons les packages dédiés au turtlebot :

```
$ sudo apt install ros-foxy-dynamixel-sdk
$ sudo apt install ros-foxy-turtlebot3-msgs
$ sudo apt install ros-foxy-turtlebot3
$ echo 'export ROS_DOMAIN_ID=30 #TURTLEBOT3' >> ~/.bashrc
$ echo 'export TURTLEBOT3_MODEL=burger' >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt update
```

(<https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/#pc-setup>)

Ca y est ! Notre turtlebot contient désormais les packages principaux requis afin d'être fonctionnel.

On peut le vérifier en le pilotant à distance, après avoir lancé le programme :

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

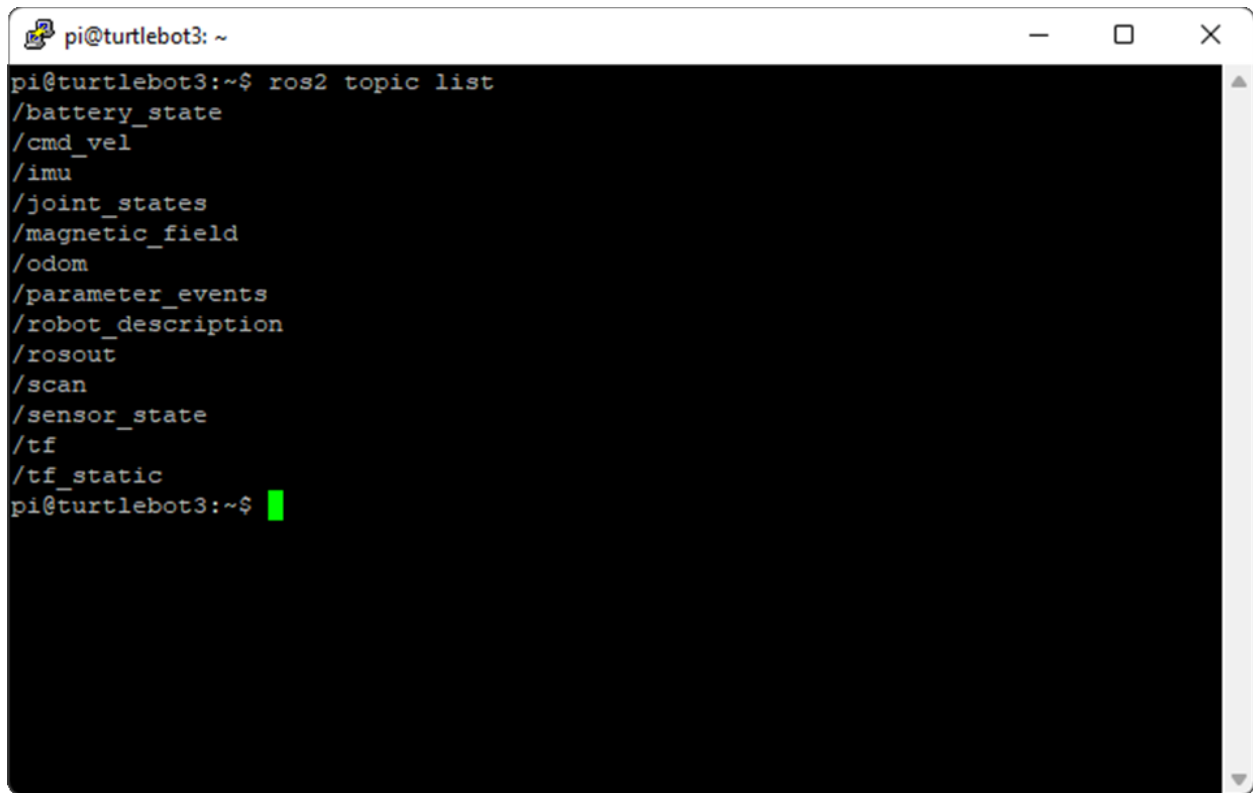
Sur le robot, puis :

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

Sur un ordinateur du même DOMAIN_ID.

On peut vérifier la bonne initialisation du robot à l'aide de :

```
$ ros2 topic list
```

A terminal window titled 'pi@turtlebot3: ~' with standard window controls. The command 'ros2 topic list' has been executed, resulting in a list of ROS2 topics. The topics are: /battery_state, /cmd_vel, /imu, /joint_states, /magnetic_field, /odom, /parameter_events, /robot_description, /rosout, /scan, /sensor_state, /tf, and /tf_static. The prompt 'pi@turtlebot3:~\$' is followed by a green cursor.

```
pi@turtlebot3:~$ ros2 topic list
/battery_state
/cmd_vel
/imu
/joint_states
/magnetic_field
/odom
/parameter_events
/robot_description
/rosout
/scan
/sensor_state
/tf
/tf_static
pi@turtlebot3:~$
```

Si la carte de pilotage STM32 / OpenCR n'as as été initialisée, vous pourrez suivre le tutoriel fourni par Robotis pour la paramétrer :

https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/#opencr-setup

La procédure a été effectuée sous cette configuration et est validée comme fonctionnelle.

Si vous possédez un LIDAR LDS-02, il faudrait utiliser colcon afin de construire une autre workspace et de charger le driver requis.

Dans son état actuel, le driver n'est malheureusement pas fonctionnel et ne peut pas être construit sous ROS2. (Des changements récents sur la branche de développement semblent avoir rendu le logiciel inconstructible sous cette configuration de ROS2 Foxy.)

Le tutoriel à suivre est censé être le suivant :

https://emanual.robotis.com/docs/en/platform/turtlebot3/sbc_setup/#sbc-setup

On commence par construire un nouveau workspace et y installer le driver :

```
$ echo 'export LDS_MODEL=LDS-02' >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt update
$ sudo apt install libudev-dev
$ cd
$ mkdir custom_ws
$ cd ~/custom_ws
$ mkdir src
$ cd src
$ git clone -b ros2-devel https://github.com/ROBOTIS-GIT/lid08_driver.git
```

Puis on construit la workspace :

```
$ cd ~/custom_ws && colcon build --symlink-install
```

A ce stade, le build stagne à 83%. Normalement, une fois ce build produit, nous pourrions rajouter le `local_setup.bash` de notre workspace au `bashrc` puis utiliser `Turtlebot_cartographer` pour visualiser la cartographie.

Certaines personnes trouvent des manières de compiler tout de même le driver, malheureusement des fixes n'existent pas encore pour ROS2 Foxy :

(https://github.com/ROBOTIS-GIT/lid08_driver/pull/4)

Mes essais m'ont permis de cibler ce driver car la première version fonctionne correctement, le topic `/scan` fonctionne correctement mais son Publisher (par défaut celui du lidar LDS-01) ne poste rien malgré un démarrage correct.

```
colcon build [0/1 done] [1 ongoing]
pi@turtlebot3:~/custom_ws$ cd src
pi@turtlebot3:~/custom_ws/src$ git clone -b ros2-devel https://github.com/ROBOTI
S-GIT/ld08_driver.git
Cloning into 'ld08_driver'...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 82 (delta 32), reused 76 (delta 26), pack-reused 0
Unpacking objects: 100% (82/82), 32.41 KiB | 162.00 KiB/s, done.
pi@turtlebot3:~/custom_ws/src$ cd ld08_driver/
pi@turtlebot3:~/custom_ws/src/ld08_driver$ sudo nano C
pi@turtlebot3:~/custom_ws/src/ld08_driver$ sudo nano CMakeLists.txt
pi@turtlebot3:~/custom_ws/src/ld08_driver$ sudo nano CMakeLists.txt
pi@turtlebot3:~/custom_ws/src/ld08_driver$ cd ..
pi@turtlebot3:~/custom_ws/src$ ..
.: command not found
pi@turtlebot3:~/custom_ws/src$ cd ..
pi@turtlebot3:~/custom_ws$ colcon build --symlink-install
Starting >>> ld08_driver
[Processing: ld08_driver]
[Processing: ld08_driver]
[Processing: ld08_driver]
[Processing: ld08_driver]
[2h 37min 43.5s] [0/1 complete] [ld08_driver:build 83% - 2h 37min 40.7s]
```