

SAE Robot TurtleBot3

Voici un exemple de plan à suivre pour faire fonctionner le robot TurtleBOT3 waffle_pi. Le fichier est basé sur la doc officiel du site TurtleBOT, avec des éléments en plus.

Il se peut que des éléments ne soit plus à jour, ne pas hésiter à vérifier sur internet si une erreur arrive.

Table des matières

Installation de l'environnement	2
Mouvement du robot.....	2
Simulation.....	3
Vérification des données du LIDAR	4
Navigation (évitement d'obstacle)	5
Conduite Autonome.....	5
Personnalisation et tests	5
Programmation du robots (Simulation)	6
Utilisation de la Camera	9
Annexes.....	11
Démarrage du robot	11
Configuration du réseau pour ROS	12
Phase 4 : Navigation et Évitement d'Obstacle (Nav2)	15
4.1 Navigation Autonome (Nav2 Stack)	15
4.2 Conduite Autonome (ROS 2 Humble Simulation)	15

Suivre la doc de TurtleBOT pour initialisation de TurtleBOT sur ubuntu :

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

Choix de la version ROS : ROS 2 HUMBLE

Version de linux : [Ubuntu 22.04 LTS Desktop image \(64-bit\)](#)

Installation de l'environnement

Installation de linux et package turtlebot3

Installation sur la Raspberry PI, connexion en ssh pour utiliser que le PC Distant

Pensez à bien augmenter la ram et les cpu si une machine virtuelle est utilisée

Suivre la documentation pour faire les installations des package ROS 2 TurtleBOT.

Pour la fin de l'installation pour le bringup pensez à regarder la partie du [Démarrage du robot](#) et de la configuration du [Configuration du réseau pour ROS](#).

Mouvement du robot

Une fois l'initialisation faite, suivre les fonctions de base pour contrôler le robot à distance avec le clavier.

TurtleBOT Téléopération

Faire fonctionner moteur avec fonction de bases intégré

Téléopération à distance (avec clavier)

Lecture de données de vitesse, position etc...

2.1 Téléopération et Contrôle

- **A. Lancement des Nœuds du Robot** : Utiliser la commande ROS 2 appropriée pour démarrer les nœuds de base sur la Raspberry Pi
- **B. Contrôle Clavier** : Lancer le nœud de téléopération à partir du PC distant pour commander le robot via le clavier ex :

```
• ros2 run turtlebot3_teleop teleop_keyboard
```

- **C. Visualisation des Données** : Utiliser **RQT (ROS 2 Qt tools)** pour visualiser les messages et les topics :
 - **Topic de commande** : /cmd_vel (type geometry_msgs/msg/Twist).
 - **Topic d'Odométrie** : /odom (position et vitesse).
 - **Topic de Capteurs** : /imu, /battery_state, etc.

Simulation

Pour la simulation un logiciel tiers est utilisé, c'est Gazebo. Pour parvenir à simuler sur les commandes ROS il est utilisé pour faire le lien avec ROS et TurtleBOT. Suivre le lien vers la doc pour l'installation. Pour la version compatible avec ROS HUMBLE, normalement la version Gazebo est la plus adapté, essayer d'utiliser une version récente.

[Gazebo simulation](#)

[Gazebo installation](#)

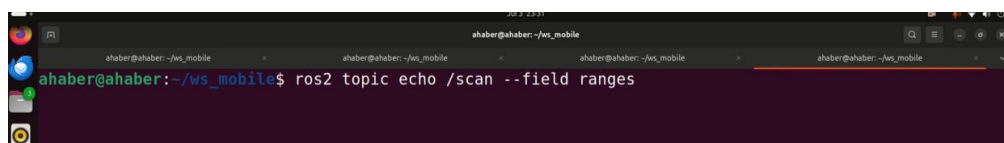
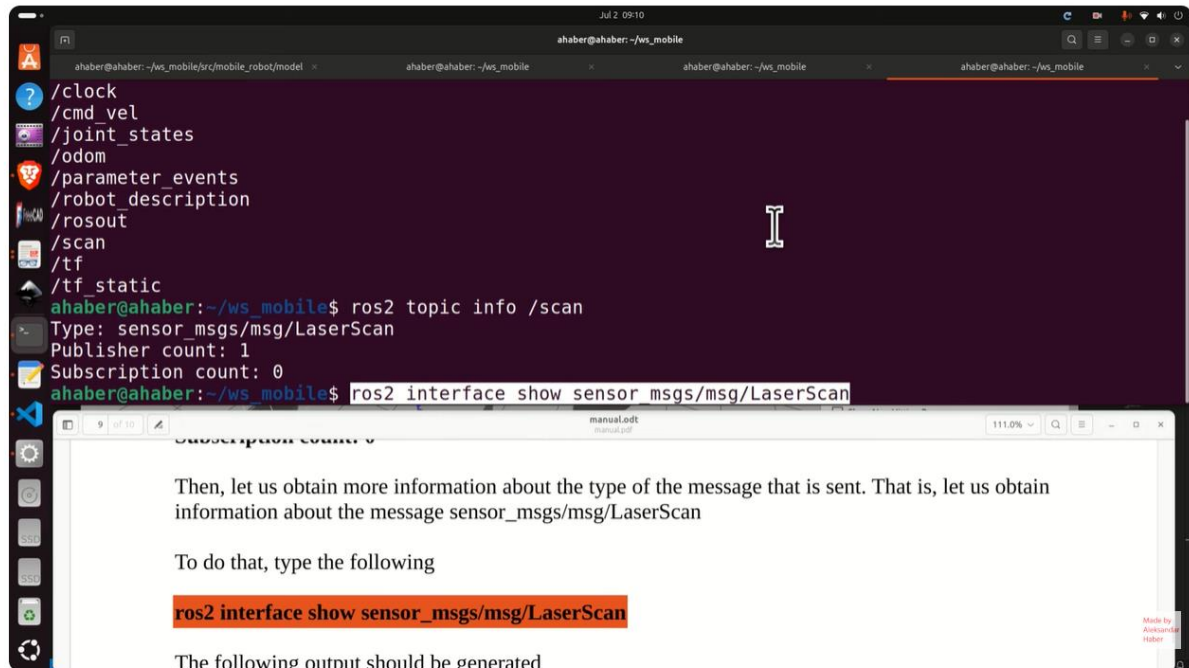
- **B. Création de la Carte** : Lancer le nœud SLAM, explorer l'environnement avec la téléopération et enregistrer la carte générée (.pgm et .yaml). et la visualisé avec RQT

[Dessinez environnement \(Faire une map avec SLAM\)](#)

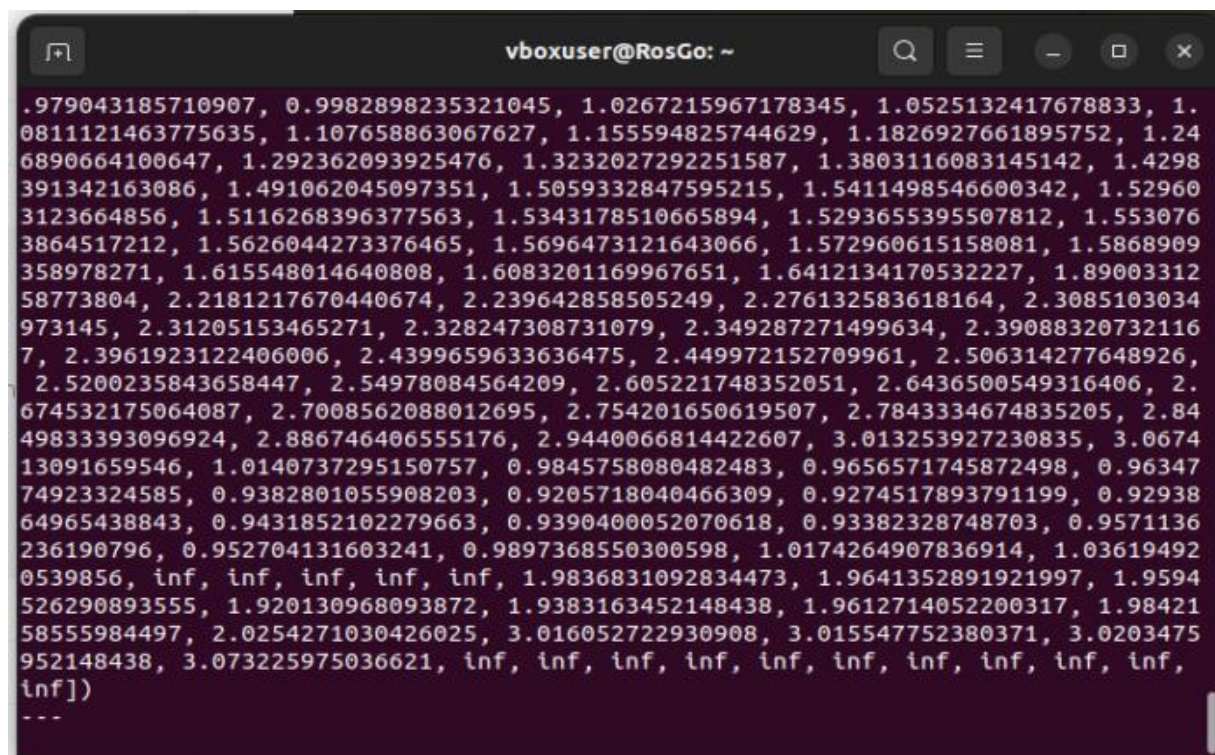
Voir les données du Lidar et obstacle (Voir image pour la commande correcte pour récupérer les bonnes données du LIDAR)

Utilisation de logiciel comme gazebo et Rviz pour un environnement 3D et vérifier la simulation

Vérification des données du LIDAR



On va ainsi voir le tableau rempli des valeurs du LIDAR (inf pour valeur infini)

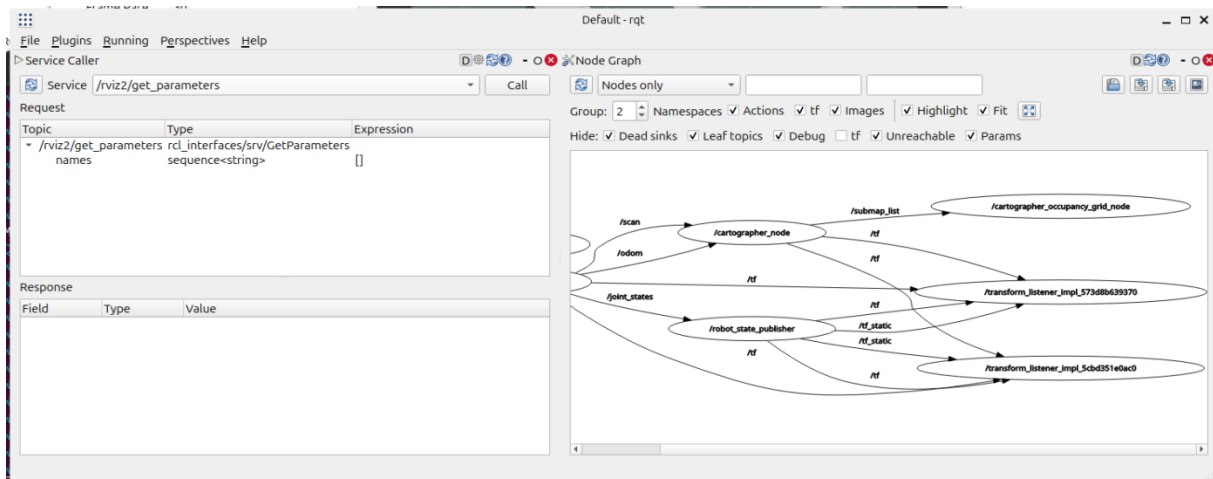


Navigation (évitement d'obstacle)

Intégrer une stratégie d'évitement d'obstacle

Allez d'un point A à B (par choix ou balise)

Visualisation des nœuds avec RQT



Pour les paramètres qui sont modifiables, ce fichier à l'air de fonctionner

```
vboxuser@RosGo:~$ nano turtlebot3_ws/src/turtlebot3/turtlebot3_navigation2/param/humble/waffle_pi.yaml
```

Conduite Autonome

Autonomous Driving

La version de ROS2 Humble doit être utilisée, mais seulement en simulation donc utilisé que sur une version du PC et pas sur le vrai TurtleBOT.

Avoir une version de conduite Autonome sur le vrai TurtleBOT (niveau avancé ou trouvé projet existant)

Personnalisation et tests

Modifier paramètres du robot avec le fichier .yaml : vitesse, capteurs

Utilisation d'autres type de SLAM : Hector, Cartographer, Lancer plusieurs robots en même temps en simulation pour les faire se suivre par exemple

Retour Automatique à une position initiale

Exploration automatique d'une carte

Programmation du robots (Simulation)

Il est possible de faire ses propres programmes en python ou C++ pour déplacer le robot lire les capteurs

Etape pour faire écrire son fichier en python pour simplement avancer les moteurs avec ROS :

Créer un espace de travail :

```
Mkdir -p ~/turtlebot_ps/src  
cd ~/turtlebot_ps  
colcon build  
source install/setup.bash
```

Créer un package

```
cd src  
ros2 pkg create --build-type ament_python turtlebot_control
```

Créer un fichier move_forward.py dans -> turtlebot_control/turtlebot_control et y mettre ce programme :

Vérifier la commande /cmd_vel quel topic est utilisé si c'est Twit ou TwistStamped, avec la commande *info*

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import TwistStamped

class MoveForward(Node):
    def __init__(self):
        super().__init__('move_forward')
        # Publier sur le bon topic avec le bon type
        self.publisher_ = self.create_publisher(TwistStamped,
        '/cmd_vel', 10)

        timer_period = 0.5 # secondes

        self.timer = self.create_timer(timer_period,
        self.timer_callback)

    def timer_callback(self):
        twist_stamped = TwistStamped()
        twist_stamped.header.frame_id = "base_link"
        twist_stamped.twist.linear.x = 0.2 # Avancer
        twist_stamped.twist.angular.z = 0.0 # Pas de rotation
        self.publisher_.publish(twist_stamped)
        self.get_logger().info('Moving forward...')

def main(args=None):
    rclpy.init(args=args)
    node = MoveForward()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

Rendre le fichier exécutable

```
chmod +x turtlebot_control/move_forward.py
```

Mets à jour setup.py :

```
entry_points={
    'console_scripts': [
        'move_forward = turtlebot_control.move_forward:main',
    ],
},
```

Rebuild :

```
cd ~/turtlebot_ws
colcon build
source install/setup.bash
```

Lancer :

```
ros2 run turtlebot_control move_forward
```

Utilisation de la Camera

Initialisation à la fin du SBC Setup : [Camera Setup](#)

Reconnaissance d'objet

Traitement

Utilisation des paramètres avancé et la Calibration caméra

More Info

For detailed specifications and advanced settings, please check the [13.More Info - 13.1.Appendixes - Raspberry Pi Camera](#) for a comprehensive guide on hardware capabilities and software features.

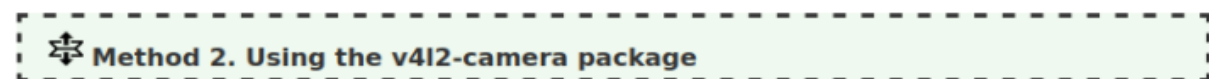
Camera Calibration

If you plan to use advanced vision features like camera calibration, you can find the detailed instructions [here](#).

Trouble Shooting

If the error message `Unable to open camera calibration file` appears, check the solution [here](#).

La méthode 2 est celle qui a l'air plus fonctionnelle.

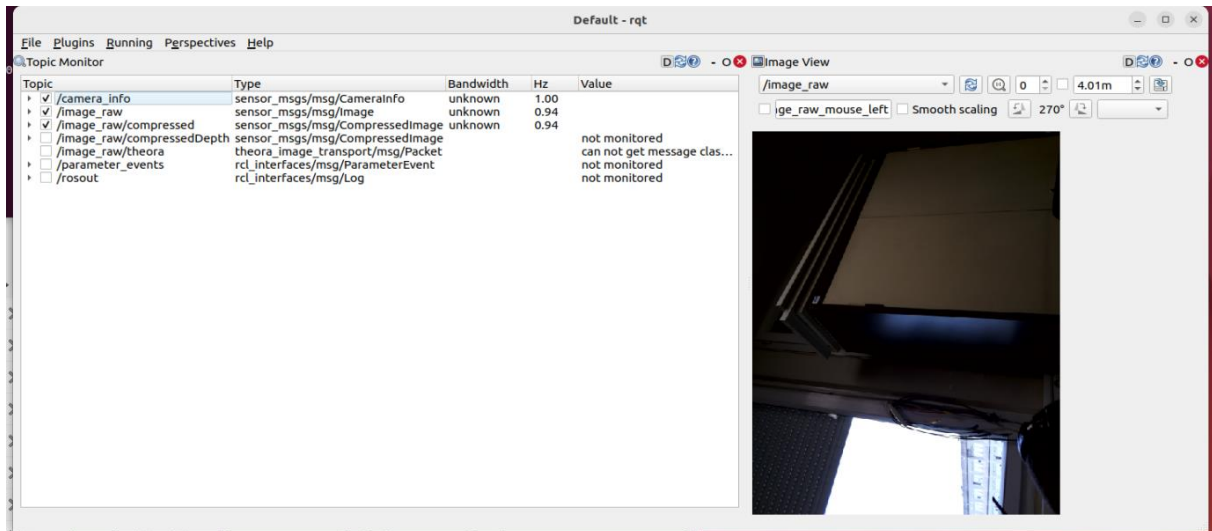


Bien pensé à source le bash pour avoir les bonnes données pour faire fonctionner la caméra :

```
geii@ubuntu:~$ source ~/.bashrc
```

Voici les infos que l'on reçoit après avoir lancer la node pour la camera

```
geii@ubuntu: ~
d: 4, for control: Force Key Frame
[WARN] [1763559779.171252397] [v4l2_camera]: Control type not currently supporte
d: 6, for control: Camera Controls
[WARN] [1763559779.171400617] [v4l2_camera]: Control type not currently supporte
d: 9, for control: Auto Exposure, Bias
[WARN] [1763559779.171534467] [v4l2_camera]: Control type not currently supporte
d: 9, for control: ISO Sensitivity
[WARN] [1763559779.171705002] [v4l2_camera]: Control type not currently supporte
d: 6, for control: JPEG Compression Controls
[INFO] [1763559779.171804815] [v4l2_camera]: Requesting format: 1024x768 YUYV
[INFO] [1763559779.178208051] [v4l2_camera]: Success
[INFO] [1763559779.178308012] [v4l2_camera]: Requesting format: 640x480 YUYV
[INFO] [1763559779.178769579] [v4l2_camera]: Success
[INFO] [1763559779.181013508] [v4l2_camera]: Starting camera
[WARN] [1763559779.947559330] [v4l2_camera]: Image encoding not the same as requ
ested output, performing possibly slow conversion: yuv422_yuy2 => rgb8
[INFO] [1763559780.056365778] [v4l2_camera]: using default calibration URL
[INFO] [1763559780.056593793] [v4l2_camera]: camera calibration URL: file:///hom
e/geii/.ros/camera_info/mmal_service_16.1.yaml
[ERROR] [1763559780.056916918] [camera_calibration_parsers]: Unable to open cam
era calibration file [/home/geii/.ros/camera_info/mmal_service_16.1.yaml]
[WARN] [1763559780.057012490] [v4l2_camera]: Camera calibration file /home/geii/
.ros/camera_info/mmal_service_16.1.yaml not found
```



Voici l'image qui est bien récupérée du robot que l'on peut voir dans le logiciel rqt.

Voici maintenant le fichier de config pour la caméra que l'on modifie pour faire fonctionner avec la méthode 2.

```
sudo nano /boot/firmware/config.txt
```

Si la caméra ne fonctionne pas essayer avec ce fichier :

```
kernel=vmlinuz cmdline=cmdline.txt initramfs initrd.img followkernel [pi4]
dtoverlay=vc4-fkms-v3d max_framebuffers=2 arm_boost=1 [all] # Enable the audio
output, I2C and SPI interfaces on the GPIO header. As these # parameters related to the
base device-tree they must appear *before* any # other dtoverlay= specification
dtparam=audio=on dtparam=i2c_arm=on dtparam=spi=on disable_overscan=1 # If you
have issues with audio, you may try uncommenting the following line # which forces the
HDMI output into HDMI mode instead of DVI (which doesn't # support audio output)
#hdmi_drive=2 # Enable the serial pins enable_uart=1 # Autoload overlays for any
recognized cameras or displays that are attached # to the CSI/DSI ports. Please note
this is for libcamera support, *not* for # the legacy camera stack start_x=1 start_x=1
#display_auto_detect=1 # Config settings specific to arm64 arm_64bit=1
#dtoverlay=imx219 [cm4] # Enable the USB2 outputs on the IO board (assuming your
CM4 is plugged into # such a board) #dtoverlay=dwc2,dr_mode=host [all]
gpu_mem=128 dtoverlay=disable-bt
```

```
geli@ubuntu:~$ ros2 run v4l2_camera v4l2_camera_node --ros-args -p image_encodin
g:=yuv422_yuy2
```

Pour remettre encoding par défaut, il fait la conversion dans le lancement de la node normalement

Annexes

Les bases de ros2 pour bien comprendre les principes de bases :

https://wiki.ares.asso-ensea.fr/index.php/Les_bases_de_ROS2

Démarrage du robot

Après avoir démarré le robot la RPI et que toutes les installations sont faites ainsi que le programme est sur la carte OpenCR et le pc distant voici un résumé des actions et commande à faire pour lancer son fonctionnement de base.

```
geii@ubuntu:~$ source ~/.bashrc
```

Pour initialiser tous les scripts et les variables sur pc distant et le robot

```
export TURTLEBOT3_MODEL=waffle_pi
```

Voilà les variables qui pourrait fonctionner selon la configuration du robot, qui sont censées être à la fin du fichier ~/.bashrc quand le tutoriel est terminé autant dans le fichier du Pc distant que dans la Raspberry PI:

```
source ~/turtlebot3_ws/install/setup.bash
export ROS_DOMAIN_ID=30 #TURTLEBOT3
source /usr/share/gazebo/setup.sh
source /opt/ros/humble/setup.bash
```

Robot TurtleBot3 Waffle Pi

Système d'Exploitation Ubuntu 22.04 LTS (64-bit Desktop)

Middleware Robotique ROS 2 humble

Objectif Principal Maîtriser l'environnement ROS 2, la simulation, la téléopération et la navigation autonome.

Phase 1 : Installation et Initialisation de l'Environnement

Objectif : Mettre en place les systèmes hôte et robotique, et établir la communication.

1.1 Préparation des Systèmes

- **A. Installation d'Ubuntu 22.04 LTS** (PC distant).
 - **B. Installation de ROS 2 Humble** (sur PC et sur Raspberry Pi).
- **C. Installation des paquets TurtleBot3** (turtlebot3, turtlebot3_msgs, turtlebot3_simulations) sur les deux machines. (suivre la doc de TurtleBOT)
- **D. Configuration de la Raspberry Pi :** Installation de l'OS (ex. Ubuntu Server) et des paquets ROS 2. Installation de ROS Bare Bones pour la RPI et pas la version ROS desktop

Configuration du réseau pour ROS

Faire attention a peut-être change aussi config ip du pc distant

Utiliser un wifi qui soit bien fonctionnelle, si cela ne se connecte pas à la RPI, utilisé un autre WIFI

Si on veut garder la même adresses IP local pour ne pas à avoir à changer à chaque pour la connexion au ssh et les paramètres réseau pour la liaison ROS de la RPI et du PC distant

On va choisir une IP libre sur ton réseau (par ex. 192.168.1.50)

On suppose que ta passerelle (box/routeur) est 192.168.1.1

Et que ton interface réseau est eth0 (Ethernet) ou wlan0 (Wi-Fi)

Modifier le NetPlan

```
sudo nano /etc/netplan/50-cloud-init.yaml
```

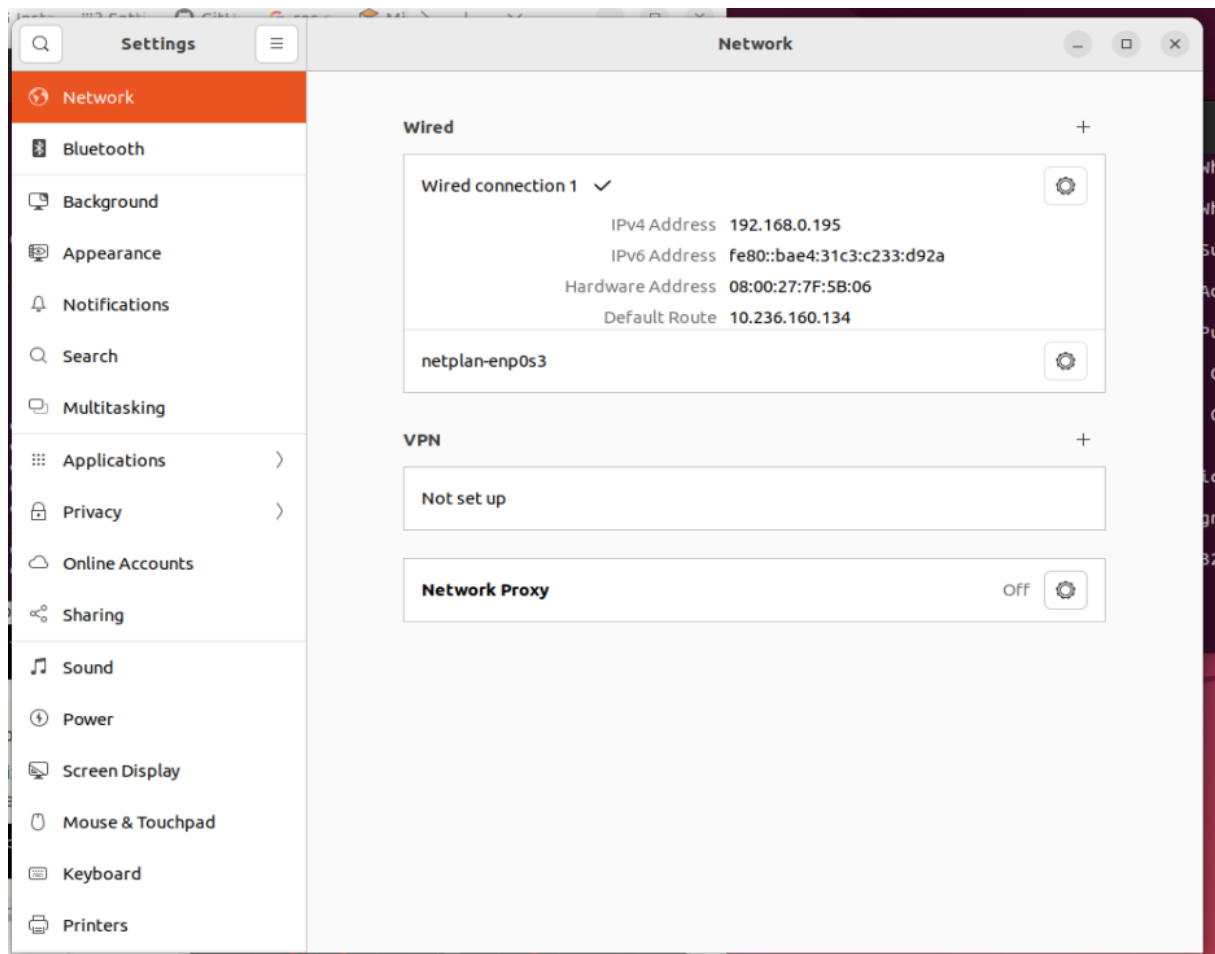
Ajouter les paramètres voulus et mettre l'adresse IP et la bonne gateways

```
network:
  version: 2
  wifis:
    wlan0:
      dhcp4: no
      addresses:
        - 192.168.1.51/24
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
      access-points:
        "NomDuWifi":
          password: "MotDePasseDuWifi"
```

Sachant que gateway peut ne plus fonctionner, préférer cette version pour remplacer gateway4 avec *routes*

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.0.195/24]
      routes:
        - to: default
          via: 10.236.160.134
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

Pour modifier l'adresse IP du PC distant, on peut le faire directement depuis les paramètres.



Pour trouver l'adresse du routeur

```
ip route
```

Pour appliquer les modifications

```
sudo netplan apply
```

Pour voir les changements appliquer et l'adresse IP modifié

```
ip addr
```

Si cela ne fonctionne pas et avec plusieurs changement l'ip ne se change pas, on peut essayer

```
$ sudo rm -f /etc/resolv.conf
```

```
$ sudo ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

Etape 3 : Simulation, Cartographie (SLAM) et Visualisation

Objectif : Utiliser Gazebo et RViz pour modéliser le robot et son environnement.

3.1 Simulation avec Gazebo

3.2 Cartographie (SLAM - Simultaneous Localization and Mapping)

- **A. Choix du SLAM :** Installer et utiliser un algorithme moderne de ROS 2 : **slam_toolbox** (recommandé pour sa robustesse).
 - *Étape optionnelle avancée :* Tester et comparer avec **Cartographer** (plus gourmand en ressources, mais précis) ou **Hector SLAM** (pour de petits environnements plats).

3.3 Visualisation et Débogage

- **A. Utilisation de RViz (ROS Visualization) :**
 - Visualiser le modèle 3D du robot.
 - Afficher la carte générée.
 - **Données LIDAR :** Afficher le topic des scans laser. Le *tableau rempli des valeurs du LIDAR* est l'output du topic, que vous pouvez visualiser de manière graphique dans RViz et textuellement via

```
ros2 topic echo /scan
```

- **B. Analyse d'Obstacle :** Utiliser **RQt Graph** pour voir la chaîne de transformation (tf) des coordonnées et les flux de données entre les capteurs et le SLAM.

Phase 4 : Navigation et Évitement d'Obstacle (Nav2)

Objectif : Implémenter la capacité du robot à se localiser et naviguer de manière autonome.

4.1 Navigation Autonome (Nav2 Stack)

Évitement d'Obstacle : Le planificateur local de Nav2 gère l'évitement d'obstacle en temps réel en utilisant les données du LIDAR.

Visualisation : Utiliser le panneau "**2D Goal Pose**" dans RViz pour envoyer des objectifs de navigation au robot.

4.2 Conduite Autonome (ROS 2 Humble Simulation)

- *Justification :* [Autonomous Driving](#) n'a été que testé en simulation sur la version **ROS 2 Humble**.

Annexes et Documentation

- **Documentation de Base ROS 2 :** https://wiki.ares.asso-ensea.fr/index.php/Les_bases_de_ROS2
- **Documentation TurtleBot Officielle :** <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>