# NavigateT: TurtleBot

*Linux software commands*

| | |
|---|---|
| Edited by: | L.MICHAUD |
| Creation date: | October 5, 2023 |
| Status | Final |
| Version | 5.0 |

| Version | Reference | Changes | Date |
|---|---|---|---|
| 0.1 | TurtleBot3-Linux-Commands-v0.1 | Creation | 05/10/2023 |
| 1.0 | TurtleBot3-Linux-Commands-v1.0 | Installation of ROS2 Humble and Gazebo and TurtleBot3 | 15/10/2023 |
| 2.0 | TurtleBot3-Linux-Commands-v2.0 | Creation of a workspace and code a first Node in Python | 25/10/2023 |
| 3.0 | TurtleBot3-Linux-Commands-v3.0 | How to use CasADI for MPC | 30/10/2023 |
| 4.0 | TurtleBot3-Linux-Commands-v4.0 | How to install and use Mocap for ROS2 Qualisys | 30/10/2023 |
| 5.0 | TurtleBot3-Linux-Commands-v5.0 | Setting up separate TurtleBot controls | 10/12/2023 |

# Table of contents

# 1. Purpose of the document

The aim of this document is to gather the *Linux* commands required to install *ROS 2 Humble*, *Gazebo* and *TurtleBot3*. In addition, we've included an example of *ROS* code and the configuration of a new workspace for developing and testing *ROS* nodes from *Visual Studio*. We've also included some commands for using *CasADi* and *Mocap ROS 2 Qualisys* to retrieve the *TurtleBot's* position from *Esynov's* cameras, as well as commands for connecting to a real *TurtleBot* from its IP address and other useful commands. Please note that this installation is compatible with *Ubuntu 22.04* and may be obsolete for future versions. You'll need to adapt your installation accordingly.

# 2. Ubuntu installation

First, go to the official Ubuntu website to download the installation image (IOS file). You can find the official link below: [https://www.ubuntu-fr.org/download/](https://www.ubuntu-fr.org/download/)

You can then install the ISO on a virtual environment or on a main computer. Some useful commands for installing and updating packages:

- **Software update:** *sudo apt update*
- **Software upgrade**: *sudo apt upgrade*
- **Install specific package:** *sudo apt-get install package_name*
- **Uninstall specific program:** *sudo apt-get purge program_name**

# 3. ROS 2 Humble installation

Once you've installed the Ubuntu environment, you can install a set of software libraries and tools for building robotic applications: the Robot Operating System (ROS). The following installation commands correspond to the ROS 2 Humble version. You can find the official link of ROS 2 Humble below: [https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html](https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html)

**<u>Some requirements before the installation (Python 3 Colcon and Curl installation):</u>**

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

**<u>Installation of ROS 2 Humble:</u>**

- **Add the ROS 2 apt repository to your system:**

```
sudo apt install software-properties-common
sudo add-apt-repository universe
```

- **Add the ROS 2 GPG key with apt:**

```
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
```

- **Add the repository to your sources list:**

```
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
```

- **Install ROS 2 packages (ROS, Rviz, Demos, Tutorials):**

```
sudo apt update
sudo apt upgrade
sudo apt install ros-humble-desktop
```

- **ROS-Base install: Communication libraries, message packages, command line tools (No GUI tools):**

```
sudo apt install ros-humble-ros-base
```

- **Development tools: Compilers and other tools to build ROS packages**

```
sudo apt install ros-dev-tools
```

- **Environment setup**

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

**Check installation:**

- **In one terminal, source the setup file and then run a C++ talker:**

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_cpp talker
```

- **In another terminal source the setup file and then run a Python listener:**

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_py listener
```

**Uninstall ROS 2 Humble:**

```
sudo apt remove ~nros-humble-* && sudo apt autoremove
sudo rm /etc/apt/sources.list.d/ros2.list
sudo apt update
sudo apt autoremove
sudo apt upgrade
```

## 4. Gazebo installation

Once you've installed the ROS 2 Humble, you can install the Gazebo simulator to simulate the TurtleBot in a simulation environment. You can find the official link of Gazebo for ROS 2 below: https://installati.one/install-gazebo-ubuntu-22-04/

**Installation:**

- **Install some necessary tools:**

```
sudo apt-get update
sudo apt-get install lsb-release wget gnupg
```

- **Install Gazebo for Ubuntu 22.04**

```
sudo apt install gazebo
sudo apt install libgazebo-dev
```

**Uninstall Gazebo:**

```
sudo apt-get remove gazebo
sudo apt-get -y autoremove gazebo
sudo apt-get -y purge gazebo
sudo apt-get -y autoremove
```

## 5. TurtleBot3 Humble packages installation

Then, to simulate the TurtleBot with ROS 2 Humble and Gazebo, you have to install specific packages for using TurtleBot3. You can find the official link of Turtlebot3 simulation below: https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/

**Installation:**

- **Install Simulation package:**

```
sudo apt install ros-humble-gazebo-*
sudo apt install ros-humble-cartographer
sudo apt install ros-humble-cartographer-ros
sudo apt install ros-humble-navigation2
sudo apt install ros-humble-nav2-bringup

sudo apt remove ros-humble-turtlebot3-msgs
sudo apt remove ros-humble-turtlebot3
mkdir -p ~/turtlebot3_ws/src
cd ~/turtlebot3_ws/src/
git clone -b humble-devel https://github.com/ROBOTIS-GIT/DynamixelSDK.git
```

```
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
cd ~/turtlebot3_ws
colcon build

echo 'source ~/turtlebot3_ws/install/setup.bash' >> ~/.bashrc
source ~/.bashrc
echo 'export TURTLEBOT3_MODEL=burger' >> ~/.bashrc
source ~/.bashrc

cd ~/turtlebot3_ws/src/
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
cd ~/turtlebot3_ws && colcon build
```

# 6. Simulation and connection with the TurtleBot 3

Now you can do some simulations from Gazebo and also connect to the TurtleBot3 with a SSH Connection. To know the IP Address of the TurtleBot3, try to do a Wi-Fi hotspot from a computer and try to display Raspberry PI interface from another computer.

- **Launch Simulation World**
    - **Empty World**

```
ros2 launch turtlebot3_gazebo empty_world.launch.py
```

- **Turtlebot3 World**

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

- **Operate Turtlebot3**

```
ros2 run turtlebot3_teleop teleop_keyboard
```

- **Cartographer SLAM**

```
ros2 launch turtlebot3_cartographer cartographer.launch.py
```

- **Save Map**

```
ros2 run nav2_map_server map_saver_cli -f ~/map
```

- **TurtleSim**

```
ros2 run turtlesim turtlesim_node
```

- **Teleop keyboard for TurtleSim**

```
ros2 run turtlesim turtle_teleop_key
```

**SSH Connection:**

- **Configure a Wifi hotspot from a computer:**

*nmcli con add type wifi ifname <ip of hotspot wireless interface) con-name <name> autoconnect yes ssid <name SSID> ap ipv4.method shared*

**In our case**: *nmcli con add type wifi ifname wlp2s0 con-name Co4Sys autoconnect yes ssid Co4Sys 802-11-wireless.mode ap ipv4.method shared*

**Notice**: Normaly, on the Ionela's PC, the hotspot is configured, you just have to check if the Wifi network is On and in hotspot mode

- **Connection to the hotspot created before:**

*nmcli con up <name of the wireless network>*

**In our case***: nmcli con up Co4Sys*

- **Configure the SSH settings of the RaspberryPi to connect it to the Wifi hotspot:**

Login: ubuntu

Password: turtlebot

- **Get the Turtlebot3 address:**

```
ip n
```

**Notice:** turtlebot3 must be "reachable"

- **Connect to Turtlebot3:**

*ssh ubuntu@ip_address*

**In our case**: *ssh* *ubuntu@10.42.0.216* or *ubuntu@10.42.0.56* (depending which TurtleBot IP is configure on the Turtlebot you use)

- **Launch the Turtlebot3 and then the keyboard on another terminal:**

```
ros2 launch turtlebot3_bringup robot.launch.py
ros2 run turtlebot3_teleop teleop_keyboard
```

**Useful commands:**

- **IP Information:** *ifconfig*
- **List of wireless interfaces:** *ip a*
- **Wireless scan:** *sudo arp-scan-localnet -I<wireless interface with ip a>*
- **Display your own IP Address:** *hostname -I*
- **Display all the hotspot connection:** *nmcli connection show*

## 7. Configure Visual Studio workspace and code Python node

If no workspace is already set, you can do yours and code your first node in Python. If you are interested. If you're interested, there's a series of videos on the ROS 2 Humble environment for building your workspace and coding ROS nodes. The video playlist link is as follows:

[https://www.youtube.com/watch?v=0aPbWsyENA8&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&ab_channel=RoboticsBack-End](https://www.youtube.com/watch?v=0aPbWsyENA8&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&ab_channel=RoboticsBack-End)

**Create a ROS2 Workspace and code your own node in Python:**

- **Install colcon extensions:** *sudo apt install python3-colcon-common-extensions*

- **Autocompletion:** *gedit ~/.bashrc* **and add the following line at the end:** *source /usr/share/colcon_argcomplete/hook/ colcon-argcomplete.bash*

- **Make a workspace directory:** *mkdir ros2_ws*

- **Make a source directory inside the workspace:** *cd ros2_ws && mkdir src*

- **Build the workspace:** *colcon build*

- **Source the workspace:** *cd && source ~/ros2_ws/install/setup.bash && gedit ~/.bashrc* **and add in the file the following line at the end :** *source ~/ros2_ws/install/setup.bash*

- **Create a ROS2 package into the workspace:** *cd ros2_ws/src/ && ros2 pkg create name_package –build-type ament_python –dependencies rclpy*

- **Make sure Visual Studio Code is installed with Python and ROS2 extensions**

- **Check installation:** *sudo snap install code --classic*

- **Launch the workspace from the terminal:** *code .*

- **Build the package with colcon:** *cd ros2_ws/ && colcon build*

- **Fix "SetuptoolsDeprecationWarning: setup.py install is deprecated" in colcon build:** *sudo apt install python3-pip && pip3 install setuptools==58.2.0*

- **Create a python file to write a node and make it executable:** *cd ros2_ws/src/name_package/name_package/touch name_file_node.py && chmod +x name_file_node.py*

- **Execute the python node file created:** *cd ros2_ws/src/name_package/name_package/ && ./name_file_node.py*

<u>**Example of a Python node code:**</u>

- ***my_first_node.py***

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node

class MyNode(Node):

    def __init__(self):
        super().__init__("first_node") #Initialization of the Node
        self.get_logger().info("Hello from ROS2") #Display logger
into the terminal

def main(args=None):
    rclpy.init(args=args) #Initialization ROS2 communication
    node = MyNode() #Create a node from the MyNode class
    rclpy.spin(node) #Keep alive the node


    rclpy.shutdown() #Close the ROS2 communication

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

- **setup.py**

```
entry_points={
        'console_scripts': [
            "test_node = my_robot_controller.my_first_node:main"
        ],
    },
```

- **Auto-compilation without colcon build every time**

```
cd ros2_ws/
colcon build –symlink-install
source ~/.bashrc
ros2 run my_robot_controller test_node
```

- **Useful commands:**

```
ros2 node list
ros2 node info /name_of_the_node
ros2 topic list
ros2 topic info /name_of_the_topic
rqt_graph
ros2 interface show <type of ros2 topic info >
```

## 8. CasADi for MPC

If you want to use implement *MPC* functions, you can use *CasADI* in Python:

__**Install CasADi for Python:**__

- Verify pip installation (should be ≥8.1) : *pip -V*
- Install CasADi: *pip install casadi*
- **Python** script to verify the installation:
  *from casadi import **
  *x = MX.sym("x")*
  *print(jacobian(sin(x),x))*

__**Notice:**__ If CasADi is well installed, the script should display "*cos(x)*"

## 9. Mocap for ROS 2

If you want to use the coordinates of the cameras in Esynov, you can retrieve them using Qualisys and use Mocap for ROS 2:

**Install and use Mocap4Ros2 from the following GitHub:** [https://github.com/MOCAP4ROS2-Project](https://github.com/MOCAP4ROS2-Project)

```
mkdir -p mocap_ws/src && cd mocap_ws/src
git clone --recursive https://github.com/MOCAP4ROS2-Project/mocap4ros2_qualisys.git
vcs import < mocap4ros2_qualisys/dependency_repos.repos
cd .. && colcon build --symlink-install
source install/setup.bash
mocap_ws/src/mocap4ros2_qualisys/qualisys_driver/config/qualisys_driver_params.yaml
ros2 launch qualisys_driver qualisys.launch.py
ros2 launch mocap_marker_viz mocap_marker_viz.launch.py mocap_system:=qualisys
```

**Other useful command:**

```
ros2 topic echo /rigid_bodies
source ~/mocap_ws/install/setup.bash
```

## 10. Configuration of multiple TurtleBot

The goal of this part is to control multiple TurtleBot separately. In this way, even if multiple TurtleBots are connected to the same Wi-Fi hotspot (with a unique IP Address), it will possible to give different orders. To do that, we have to give a namespace to each TurtleBot. Otherwise, each order will be addressed on both TurtleBots and it will be not possible to launch different nodes at the same time.

For this tutorial, we will use "tb3_1" as namespace to configure a TurtleBot. Make sure before to be connected in *ssh* to the Raspberry Pi and you applied the following Linux commands in the terminal dedicated to the Raspberry PI.

- **1st step: Create a new ros2 package**

Start by going to the *"src"* directory on your Raspberry Pi, which contains the turtlebot3 and utils packages. Next, create two empty directories in the new package and go to the launch directory and create a new ***Bringup*** launch file. These are the Linux commands associated:

```
cd ~/turtlebot3_ws/src && ros2 pkg create tb3_1
cd ~/turtlebot3_ws/src/tb3_1 &&  mkdir launch && mkdir param
cd launch && touch tb3_1_bringup.launch.py
```

- **2nd step: Copy and modify the contents of the Tb3 *Bringup* package in your package**

In the *"turtlebot3/turtlebot3_bringup"* directory of the ros2 package, copy the contents of *robot.launch.py* into the file *tb3_1_bringup.launch.py*. Note you can use the *cp* and *nano* Linux commands to copy then edit the file. Then you can apply the following modifications in the following code:

```python
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import
PythonLaunchDescriptionSource
```

```python
from launch.substitutions import LaunchConfiguration
from launch.substitutions import ThisLaunchFileDir
from launch_ros.actions import Node

def generate_launch_description():
    TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']
    LDS_MODEL = os.environ['LDS_MODEL']
    LDS_LAUNCH_FILE = '/hlds_laser.launch.py'

    usb_port = LaunchConfiguration('usb_port', default='/dev/ttyACM0')

    tb3_param_dir = LaunchConfiguration(
        'tb3_param_dir',
        default=os.path.join(
            get_package_share_directory('tb3_1'),# CHANGE THIS!
            'param',
            TURTLEBOT3_MODEL + '.yaml'))

    if LDS_MODEL == 'LDS-01':
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',

default=os.path.join(get_package_share_directory('hls_lfcd_lds_driver'),
'launch'))
    elif LDS_MODEL == 'LDS-02':
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',

default=os.path.join(get_package_share_directory('ld08_driver'),
'launch'))
        LDS_LAUNCH_FILE = '/ld08.launch.py'
    else:
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',

default=os.path.join(get_package_share_directory('hls_lfcd_lds_driver'),
'launch'))

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')

    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value=use_sim_time,
            description='Use simulation (Gazebo) clock if true'),

        DeclareLaunchArgument(
            'usb_port',
            default_value=usb_port,
            description='Connected USB port with OpenCR'),

        DeclareLaunchArgument(
            'tb3_param_dir',
            default_value=tb3_param_dir,
            description='Full path to turtlebot3 parameter file to
load'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [ThisLaunchFileDir(),
'/turtlebot3_state_publisher.launch.py']),
```

```
            launch_arguments={'use_sim_time': use_sim_time}.items(),
        ),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([ThisLaunchFileDir(),
'/ld08.launch.py']), # CHANGE THIS
            launch_arguments={'port': '/dev/ttyUSB0', 'frame_id':
'base_scan'}.items(),
        ),

        Node(
            package='turtlebot3_node',
            executable='turtlebot3_ros',
            namespace='tb3_1',  # ADD THIS!
            parameters=[tb3_param_dir],
            arguments=['-i', usb_port],
            output='screen'),
    ])
```

Next, copy the file *turtlebot3_state_publisher.launch.py* from the *"turtlebot3_bringup/launch directory"* into your launch package directory. Make sure it has the same name! Once finished, make the following changes as indicated by the following comments:

```
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():
    TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    urdf_file_name = 'turtlebot3_' + TURTLEBOT3_MODEL + '.urdf'

    print("urdf_file_name : {}".format(urdf_file_name))

    urdf = os.path.join(
        get_package_share_directory('turtlebot3_description'),
        'urdf',
        urdf_file_name)

    # Major refactor of the robot_state_publisher
    # Reference page: https://github.com/ros2/demos/pull/426
    with open(urdf, 'r') as infp:
        robot_desc = infp.read()

    rsp_params = {'robot_description': robot_desc}

    # print (robot_desc) # Printing urdf information.

    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use simulation (Gazebo) clock if true'),
```

```
            description='Use simulation (Gazebo) clock if true'),
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='robot_state_publisher',
            namespace='tb3_1',  # ADD THIS!
            output='screen',
            parameters=[rsp_params, {'use_sim_time': use_sim_time}])
    ])
```

Finally, copy the file *ld08.launch.py* from the package located into the launch directory of your package. Again, make sure it has the same name. Modify the launch file with the following changes marked by the comments below:

```
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    return LaunchDescription([
        Node(
            package='ld08_driver',
            executable='ld08_driver',
            name='ld08_driver',
            namespace='tb3_1',  #ADD THIS!
            output='screen'),
    ])
```

- **3rd step: modify the YAML parameters file**

Now copy the *burger.yaml* file located in the *"param"* directory of the *turtlebot3_bringup* package into your "*param*" directory in "*tb3_1*" package, and make the following change at the top:

```
tb3_1:
  turtlebot3_node:
    ros__parameters:

      opencr:
        id: 200
        baud_rate: 1000000
        protocol_version: 2.0

      wheels:
        separation: 0.160
        radius: 0.033

      motors:
        profile_acceleration_constant: 214.577
        profile_acceleration: 0.0

      sensors:
        bumper_1: false
        bumper_2: false

        illumination: false
```

```
        ir: false

        sonar: false

tb3_1:
  diff_drive_controller:
    ros__parameters:

      odometry:
        publish_tf: true
        use_imu: true
        frame_id: "odom"
        child_frame_id: "base_footprint"
```

As you can see, the highest parameter was the node name (*turtlebot3_node* and *diff_drive_controller*). For the node namespace you've added to work, you'll need to add the node namespace *"tb3_1"* one level above the node name. In step 2, we've already modified the launch file to point to this *yaml* file instead of the one in the *turtlebot3_bringup* package.

- **4th step: modify the CMakeLists file**

In this section, we'll add a code snippet to our *CMakeLists.txt* that will install the launch content *"param"* of our *tb3_1r* package. Add this extract just before the "if(BUILD_TESTING)" section of the CMakeLists.txt file:

```
install(DIRECTORY
  launch
  param
  DESTINATION share/${PROJECT_NAME}/
)
```

- **5th step: Compilation and execution**

Finally, compile the code on your TurtleBot3:

```
cd ~/turtlebot3_ws && colcon build --symlink-install --parallel-workers 1 && . install/setup.bash

export TURTLEBOT3_MODEL=burger && ros2 launch tb3_1_bringup.launch.py
```

To check if it's working, you can execute a *ros2 topic list* and you should see something like that:

```
/tb3_1/battery_state
/tb3_1/cmd_vel
/tb3_1/joint_states
/tb3_1/magnetic_field
/tb3_1/odom
/tb3_1/parameter_events
/tb3_1/robot_description
/tb3_1/rosout
/tb3_1/scan
/tb3_1/sensor_state
/tb3_1/tf
```

You can repeat these procedures with other *TurtleBot3* robots with different namespaces so that several robots work on your network. It's possible that when you reboot the *TurtleBot* and want to launch the *bringup* you've just created, you'll need to relaunch a "*colcon build*" from the *"tb3_1"* package and then run the *"ros2 launch"* command.