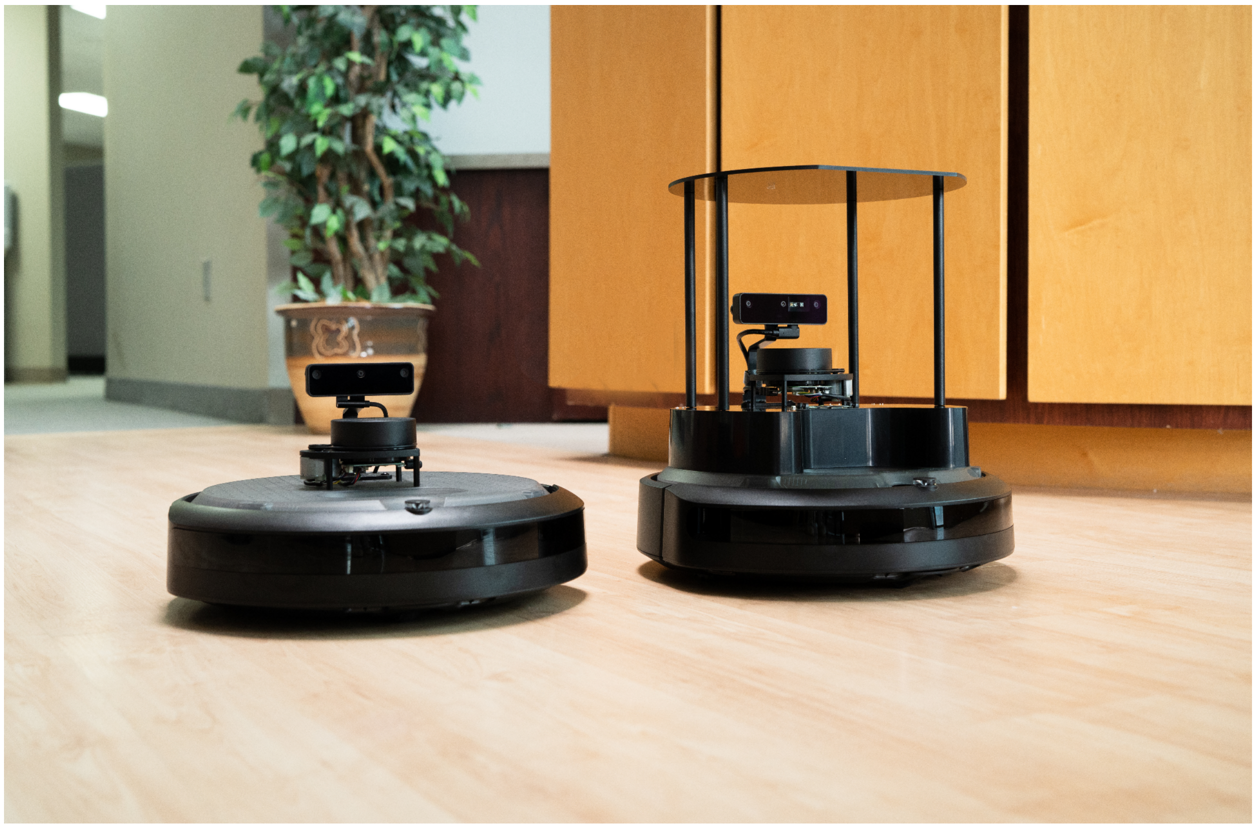
Turtlebot4 User Manual


TurtleBot 4 Lite (left) and TurtleBot 4 (right)

TurtleBot 4 is the next-generation of the world's most popular open source robotics platform for education and research, offering better computing power, better sensors and a world class user experience at an affordable price point.

TurtleBot 4 comes in two models - TurtleBot 4 and TurtleBot Lite. Both are equipped with an iRobot® Create® 3 mobile base, a powerful Raspberry Pi 4 running ROS 2, OAK-D stereo camera, 2D LiDAR and more. All components have been seamlessly integrated to deliver an out-of-the-box development and learning platform. Tap into the thriving open source ROS developer community and get started learning robotics on day one.

Features **9**
TurtleBot 4 9
TurtleBot 4 Lite 11
Hardware Specifications 12
Sensors 14
RPLIDAR A1M8 14
OAK-D-Lite 14
OAK-D-Pro 15

Resources **16**
Software 16
Ubuntu 16
Raspberry Pi 16
ROS2 16
TurtleBot 4 16
iRobot® Create® 3 17
Luxonis 17
SLAMTEC 17

Quick Start **17**
Powering on the robot 18
Installing ROS2 Galactic on your PC 18
Network Configuration 18
WiFi Setup 19
Find the Raspberry Pi IP on your network 20
Create® 3 WiFi Setup 22
TurtleBot 4 Controller Setup 23
Updating the TurtleBot 4 24
Create® 3 24
Update over WiFi 25
Find the IP address of the Create® 3 on your WiFi network. 25
Place the robot into AP mode. 25
Update over USB-C 26
Debian packages 26
Source packages 27
Install latest Raspberry Pi image 27

Overview **29**
Create® 3 29
Raspberry Pi 4 30

User PC 30

TurtleBot 4 Packages 31

TurtleBot 4 31

Installation 31

Debian installation 31

Source installation 31

Description 32

Messages 32

Actions 32

Messages 32

Services 33

Navigation 33

TurtleBot 4 Navigator 35

Node 35

Functions 36

Configuration 36

Buttons 37

Example 38

LEDs 38

Examples 39

Display 42

Menu Control 42

TurtleBot 4 Robot 43

Installation 43

Source installation 43

Base 44

GPIO Interface 44

I2C Interface 45

SSD1306 45

Configuration 45

Robot Upstart 46

Bringup 47

Diagnostics 47

Diagnostics Updater 47

Tests 48

ROS Tests 48

TurtleBot 4 Desktop 51

Installation 51

Debian installation 51

Source installation 51

Visualisation 52
TurtleBot 4 Simulator 52
Installation 52
Dev Tools 52
Ignition Edifice 52
Debian installation 53
Source installation 53
Ignition Bringup 53
Ignition GUI Plugins 54
Ignition Toolbox 55

Sensors 55
RPLIDAR A1M8 56
Connecting 56
Installing 56
Running 56
OAK-D 56
Connecting 57
Installing 57
Running 57
Create® 3 57
Cliff 57
Bumper 58
Wheeldrop 58
IR Proximity 58
Slip and Stall 58
Kidnap 58

Rviz2 58
View Model 59
View Robot 59
Rviz2 Displays 60
LaserScan 61
Camera 62
TF 63

SLAM 64
Synchronous SLAM 65
Asynchronous SLAM 65
Saving the map 66

Nav2 66
Launching Navigation 67

Launch files 67
Parameters 67
Configuration 67
Examples 67
Navigating with Rviz2 69

Simulation 69
Installing Ignition Gazebo 70
Launching Ignition Gazebo 70

TurtleBot 4 72
Attaching Accessories to the Top Integration Plate 72
Removing the Top Integration Plate 72
Making Modifications to the Top Integration Plate 73
Attaching Accessories to the Base Unit 73
Removing the PCBA 74
Removing the Create® 3 Integration Plate and Shell 75

TurtleBot 4 Lite 76
Attaching Accessories to the Base Unit 76
Removing the Create® 3 Integration Plate 77
Accessing the Raspberry Pi Computer 78

Payloads Over 9kg 79
Mechanical Modification 79
Lower payload height 79
Example 81
Software Modifications 83
Acceleration Limits 83
Velocity Limits 83
Keyboard Teleoperation 83
Joystick Teleoperation 83
Command Velocity 84
Create® 3 Actions 84
Nav2 84

Create® 3 85

Raspberry Pi 4B 86

User Interface PCBA 87
Overview 87
User I/O 87
User Power 90
Molex Picoblade 6-Pin cable assembly 91

User USB-C Ports 92

Power Budget 92

Driving your TurtleBot 4 95

Keyboard Teleoperation 95

Joystick Teleoperation 96

Command Velocity 97

Create® 3 Actions 97

Creating your first node (C++) 98

Create a workspace 98

Create a package and node 98

Write your node 98

Add your dependencies 99

Create a class 100

Subscribe to the Create® 3 interface buttons 101

Test Create® 3 Button 1 102

Create a lightring publisher 103

Publish the lightring command with a button press 107

Toggle the lightring 108

Your first C++ Node 109

Creating your first node (Python) 113

Create a workspace 113

Create a package and node 113

Write your node 113

Add your dependencies 114

Create a class 114

Subscribe to the Create® 3 interface buttons 115

Test Create® 3 Button 1 116

Create a lightring publisher 118

Publish the lightring command with a button press 121

Toggle the lightring 121

Your first Python Node 123

Generating a map 126

Launch SLAM 126

Launch Rviz2 126

Drive the TurtleBot 4 127

Save the map 127

View the map 128

Navigation 128

SLAM vs Localization 129
SLAM 129
Localization 129
Nav2 129
Launching navigation 129
Interacting with Nav2 130
2D Pose Estimate 130
Publish Point 131
Nav2 Goal 132
TurtleBot 4 Navigator 133
Navigate to Pose 133
Code breakdown 133
Initialise the node 134
Dock the robot 134
Set the initial pose 134
Wait for Nav2 135
Set the goal pose 135
Undock the robot and go to the goal pose 135
Watch navigation progress in Rviz 136
Navigate Through Poses 136
Code breakdown 136
Set goal poses 138
Navigate through the poses 138
Watch navigation progress in Rviz 138
Follow Waypoints 139
Code breakdown 139
Watch navigation progress in Rviz 140
Create Path 141
Code breakdown 142
Create your path 143
Set initial pose and clear costmaps 143
Follow the path 143
Creating a path with Rviz 144

Diagnostics 145

ROS2 Tests 147

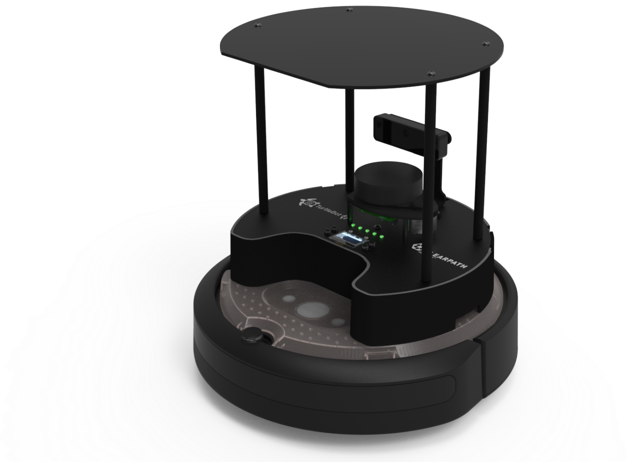
FAQ 149
Common issues with the Raspberry Pi 4B 150
1. Access point is not visible 150
Check that the Raspberry Pi is powered 150
Check for obstructions 150

Restart the robot 151
Access the RPi over Ethernet 151
1. Waiting to connect to bluetoothd… 152
2. No default controller available 152
Common issues with the user PC 153
1. ros2: command not found 153
2. Create® 3 topics are not visible 153

Overview

Features

TurtleBot 4



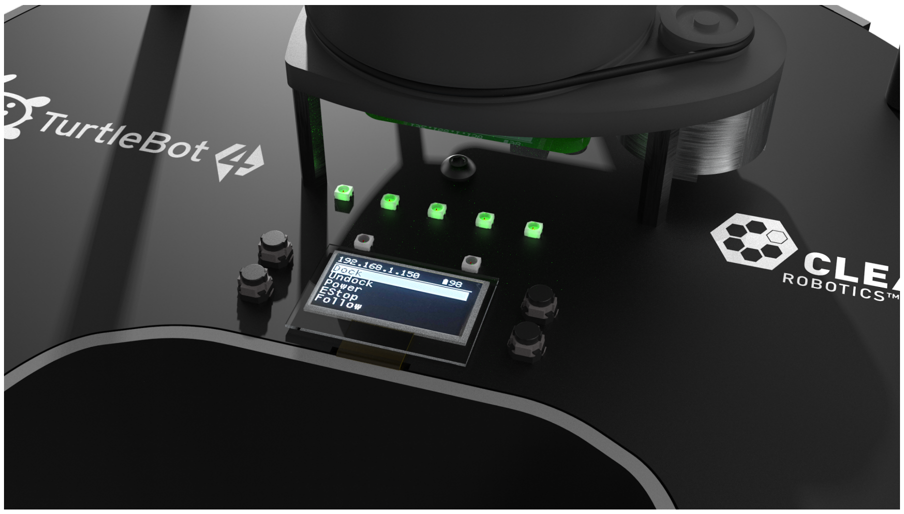
TurtleBot 4

The TurtleBot 4 is a ROS2-based mobile robot intended for education and research. The TurtleBot 4 is capable of mapping its surroundings, navigation autonomously, running AI models on its camera, and more.

It uses a Create® 3 as the base platform, and builds on it with the TurtleBot 4 shell and User Interface (UI) board. Inside the shell sits a Raspberry Pi 4B which runs the TurtleBot 4 software.

Raspberry Pi 4B

The UI Board offers status and user LEDs, user buttons, and a 128x64 user display. Additionally, it exposes 4 USB 3.0 (type C) ports, as well as additional power ports and some Raspberry Pi pins for the user.



TurtleBot 4 UI Board

On top of the UI board sits a RPLIDAR A1M8 360 degree lidar, and an OAK-D-Pro camera. Above the sensors is the sensor tower, which allows the user to customize their TurtleBot4 with additional sensors or payloads.

TurtleBot 4 Lite

TurtleBot 4 Lite

The TurtleBot 4 Lite is a barebones version of the TurtleBot 4. It has just the necessary components for navigation, mapping, and AI applications. The TurtleBot 4 has the same Raspberry Pi 4B, which sits in the cargo bay of the Create® 3, as well as the same RPLIDAR A1M8. The camera on the TurtleBot 4 Lite is the OAK-D-Lite. Additional sensors and payloads can be attached to the Create® 3 faceplate, or placed inside the cargo bay.

Hardware Specifications

Feature	TurtleBot 4 Lite	TurtleBot 4
Size (L x W x H)	342 x 339 x 192 mm	342 x 339 x 351 mm
Weight	3270 g	3945 g
Base platform	iRobot® Create® 3	iRobot® Create® 3
Wheels (Diameter)	72 mm	72 mm
Ground Clearance	4.5 mm	4.5 mm
On-board Computer	Raspberry Pi 4B 4GB	Raspberry Pi 4B 4GB
Maximum linear velocity	0.31 m/s in safe mode, 0.46 m/s without safe mode	0.31 m/s in safe mode, 0.46 m/s without safe mode
Maximum angular velocity	1.90 rad/s	1.90 rad/s
Maximum payload	9 kg	9 kg
Operation time	2h 30m - 4h depending on load	2h 30m - 4h depending on load
Charging time	2h 30m	2h 30m
Bluetooth Controller	Not Included	TurtleBot 4 Controller
Lidar	RPLIDAR A1M8	RPLIDAR A1M8
Camera	OAK-D-Lite	OAK-D-Pro

User Power	VBAT @1.9A 5V @ Low current 3.3V @ Low current	VBAT @ 300 mA 12V @ 300 mA 5V @ 500 mA 3.3v @ 250 mA
USB Expansion	USB 2.0 (Type A) x2 USB 3.0 (Type A) x2	USB 2.0 (Type A) x2 USB 3.0 (Type A) x1 USB 3.0 (Type C) x4
Programmable LEDs	Create® 3 Lightring	Create® 3 Lightring User LED x2
Status LEDs	-	Power LED Motors LED WiFi LED Comms LED Battery LED
Buttons and Switches	Create® 3 User buttons x2 Create® 3 Power Button x1	Create® 3 User buttons x2 Create® 3 Power Button x1 User Buttons x4
Battery	26 Wh Lithium Ion (14.4V nominal)	26 Wh Lithium Ion (14.4V nominal)
Charging Dock	Included	Included

Sensors

RPLIDAR A1M8

RPLIDAR A1M8

The RPLIDAR A1M8 is a 360 degree Laser Range Scanner with a 12m range. It is used to generate a 2D scan of the robots surroundings. Both the TurtleBot 4 and TurtleBot 4 Lite use this sensor. For more information, click here.

OAK-D-Lite

OAK-D-Lite

The OAK-D-Lite camera from Luxonis uses a 4K IMX214 colour sensor along with a pair of OV7251 stereo sensors to produce high quality colour and depth images. The on-board Myriad X VPU gives the camera the power to run computer vision applications, object tracking, and run AI models. For more information, visit the Luxonis documentation.

OAK-D-Pro

OAK-D-Pro

The OAK-D-Pro offers all of the same features the OAK-D-Lite has, but uses higher resolution OV9282 stereo sensors and adds an IR laser dot projector and an IR illumination LED. This allows the camera to create higher quality depth images, and perform better in low-light environments. For more information, visit the Luxonis documentation.

Resources

Software

Ubuntu

- Ubuntu 20.04 LTS Desktop (Focal Fossa): https://releases.ubuntu.com/20.04/

Raspberry Pi

- Raspberry Pi Imager: https://www.raspberrypi.com/software/
- Raspberry Pi Pinout: https://pinout.xyz/

ROS2

- Documentation: https://docs.ros.org/en/galactic/index.html
- Debian Install: https://docs.ros.org/en/galactic/Installation/Ubuntu-Install-Debians.html
- Nav2
  - Documentation: https://navigation.ros.org/
  - Github: https://github.com/ros-planning/navigation2
- SLAM
  - slam_toolbox: https://github.com/SteveMacenski/slam_toolbox

TurtleBot 4

- Common package: https://github.com/turtlebot/turtlebot4
- Desktop visualization package: https://github.com/turtlebot/turtlebot4_desktop
- Simulator package: https://github.com/turtlebot/turtlebot4_simulator
- Robot package: https://github.com/turtlebot/turtlebot4_robot
- TurtleBot 4 images: http://download.ros.org/downloads/turtlebot4/
- TurtleBot 4 hardware: https://github.com/turtlebot/turtlebot4-hardware

iRobot® Create® 3

- Product details: https://edu.irobot.com/what-we-offer/create3
- Hardware overview: https://iroboteducation.github.io/create3_docs/hw/overview/
- Electrical overview: https://iroboteducation.github.io/create3_docs/hw/electrical/
- Create® 3 Simulator: https://github.com/iRobotEducation/create3_sim
- irobot_create_msgs: https://github.com/iRobotEducation/irobot_create_msgs

Luxonis

- OAK-D-Lite product details:
  https://docs.luxonis.com/projects/hardware/en/latest/pages/DM9095.html
- OAK-D-Pro product details:
  https://docs.luxonis.com/projects/hardware/en/latest/pages/DM9098pro.html
- Depthai-ROS: https://github.com/luxonis/depthai-ros/tree/main
- Depthai-ROS Examples: https://github.com/luxonis/depthai-ros-examples
- API Documentation: https://docs.luxonis.com/projects/api/en/latest/

SLAMTEC

- RPLIDAR A1M8: https://www.slamtec.com/en/Lidar/A1
- Rplidar ROS: https://github.com/allenh1/rplidar_ros

Quick Start

Powering on the robot

To power the robot, place it on the charging dock. The Create® 3 lightring will turn on and the Raspberry Pi will be powered as well. To power off the robot, remove it from the dock and press and hold the Power button on the Create® 3. The lightring will flash 3 times, and the Create® 3 will play a sound before turning off.

Installing ROS2 Galactic on your PC

Follow these instructions to install ROS2 Galactic Desktop on your PC.

Also, install useful tools with this command:

```
sudo apt update && sudo apt install -y \
  Build-essential \
  cmake \
  git \
  python3-colcon-common-extensions \
  python3-flake8 \
  python3-pip \
  python3-pytest-cov \
  python3-rosdep \
  python3-setuptools \
  python3-vcstool \
  wget
```

Network Configuration

ROS2 Galactic supports two middlewares: CycloneDDS and FastRTPS. The default is CycloneDDS.

The Create® 3 and Raspberry Pi both use the `usb0` and `wlan0` network interfaces to communicate. As a result, CycloneDDS needs to be configured on the user PC in order to see the robot topics properly.

CycloneDDS is configured in an XML file, and that configuration should be applied to the `CYCLONEDDS_URI` environment variable.

Add this line to your `~/.bashrc` file to automatically configure CycloneDDS each time you open your terminal:

```
export CYCLONEDDS_URI='<CycloneDDS><Domain><General><DontRoute>true</></></></>'
```

For more CycloneDDS configuration options, visit the CycloneDDS documentation.

If you wish to switch middlewares or want more information on configuring the Create® 3, check out the Create® 3 Docs.

WiFi Setup

- On the first boot, the Raspberry Pi will enter AP mode which will allow you to connect to it over WiFi.
- On a PC, connect to the `Turtlebot4` WiFi network. The password is also `Turtlebot4`.
- Once connected, you can SSH into the Raspberry Pi to configure its WiFi.

```
ssh ubuntu@10.42.0.1
```

- The default password is `turtlebot4`
- In `/usr/local/bin` there will be a script called `wifi.sh` which can be used to configure the Raspberry Pi's WiFi:

```
sudo wifi.sh -s '<WIFI_SSID>' -p '<WIFI_PASSWORD>' -r <REGULATORY_DOMAIN> && sudo reboot
```

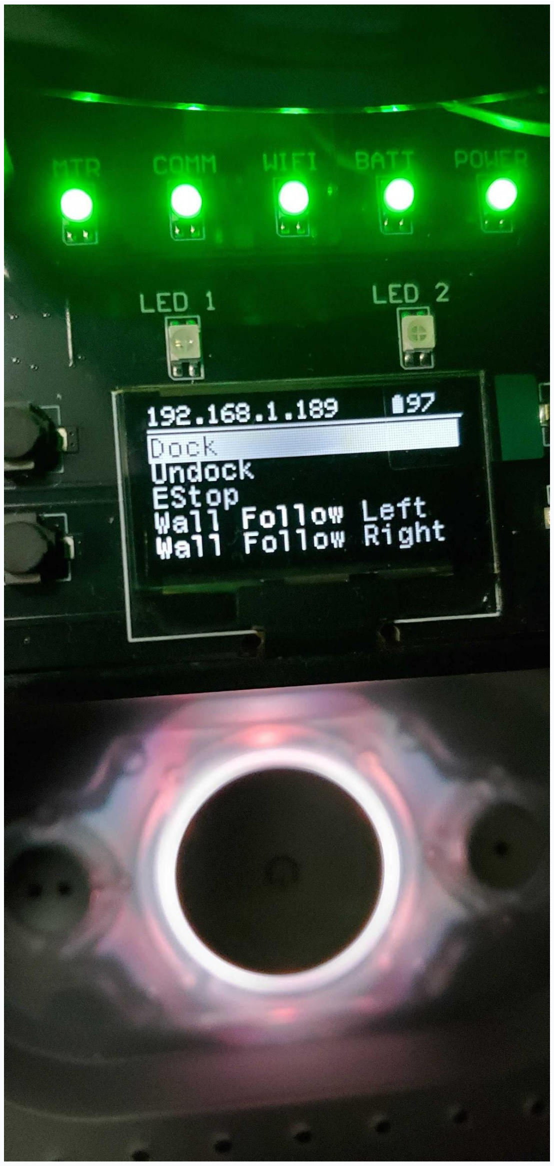
Note

The Regulatory Domain is based on the country you live in. USA: `US`, Canada: `CA`, UK: `GB`, Germany: `DE`, Japan: `JP3`, Spain: `ES`. For a full list, click here.

- Your Raspberry Pi will reboot and connect to your WiFi network.

Find the Raspberry Pi IP on your network

The TurtleBot 4 will display its WiFi IP address on the display.



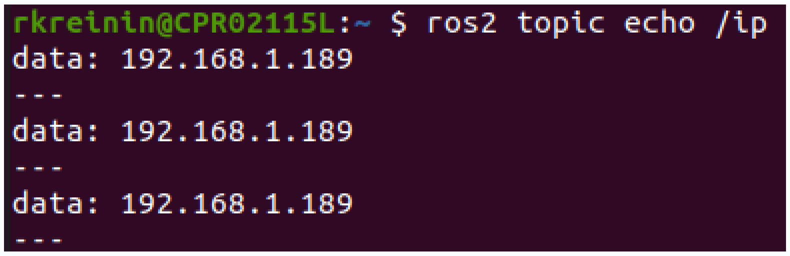
WiFi IP address on a TurtleBot 4

For the TurtleBot 4 Lite, you will need to check the `/ip` topic for the new address.

On your PC, run the following commands:

```
source /opt/ros/galactic/setup.bash
ros2 topic echo /ip
```

You should see the IP address printed out in your terminal periodically.



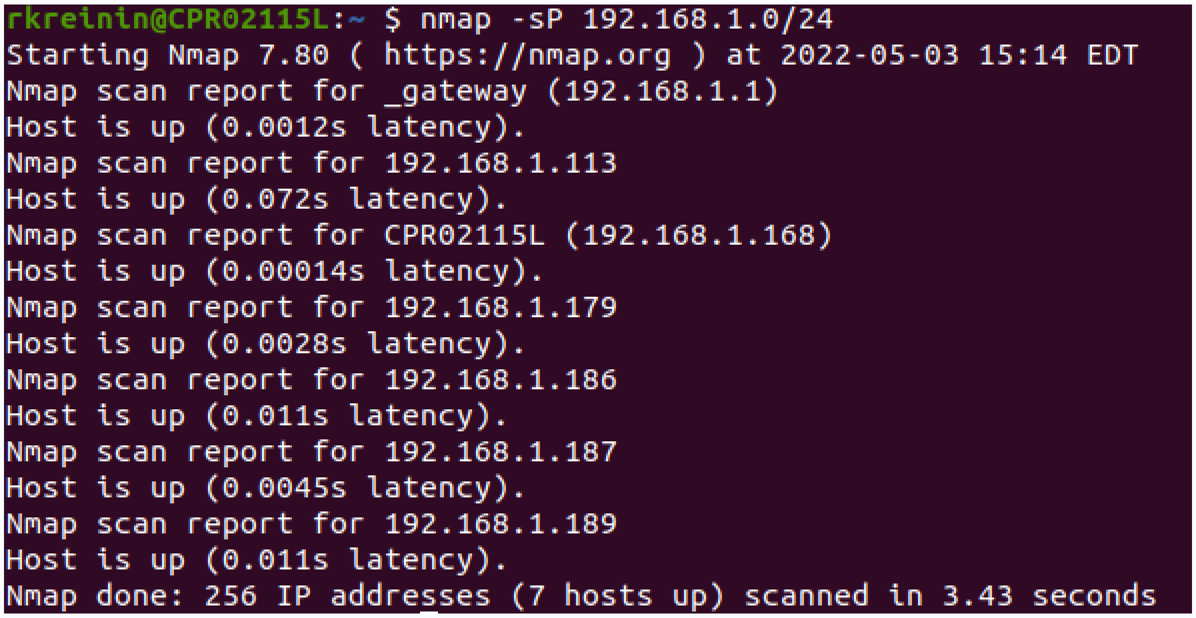
```
rkreinin@CPR02115L:~ $ ros2 topic echo /ip
data: 192.168.1.189
---
data: 192.168.1.189
---
data: 192.168.1.189
---
```

Echoing the IP address of a TurtleBot 4

If you are unable to find the IP address with the previous methods, you can try to use:

```
nmap -sP 192.168.1.0/24
```

Make sure to replace `192.168.1` with your subnet.



```
rkreinin@CPR02115L:~ $ nmap -sP 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2022-05-03 15:14 EDT
Nmap scan report for _gateway (192.168.1.1)
Host is up (0.0012s latency).
Nmap scan report for 192.168.1.113
Host is up (0.072s latency).
Nmap scan report for CPR02115L (192.168.1.168)
Host is up (0.00014s latency).
Nmap scan report for 192.168.1.179
Host is up (0.0028s latency).
Nmap scan report for 192.168.1.186
Host is up (0.011s latency).
Nmap scan report for 192.168.1.187
Host is up (0.0045s latency).
Nmap scan report for 192.168.1.189
Host is up (0.011s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 3.43 seconds
```

Scanning IP Addresses with nmap

Once you have found the IP address, you can now ssh back into the robot with it.

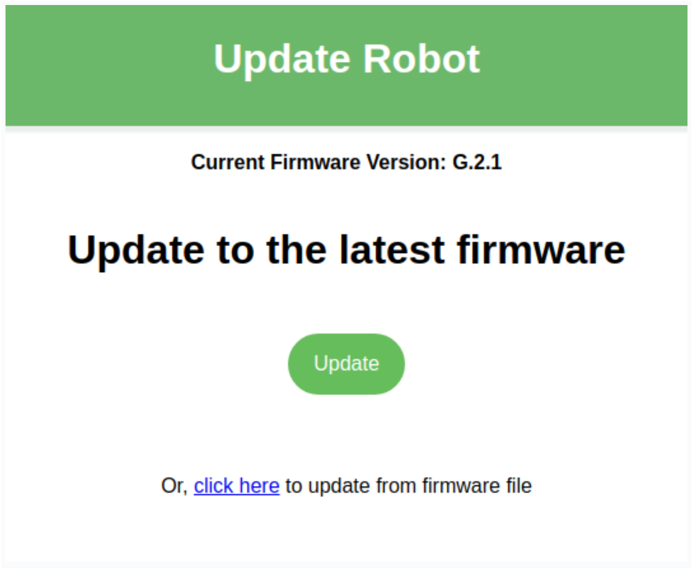
```
ssh ubuntu@xxx.xxx.xxx.xxx
```

If you wish to put the Raspberry Pi back into AP mode, you can call

```
sudo wifi.sh -a
```

Create® 3 WiFi Setup

- Press both Create® 3 button 1 and 2 simultaneously until light ring turns blue



Update Robot

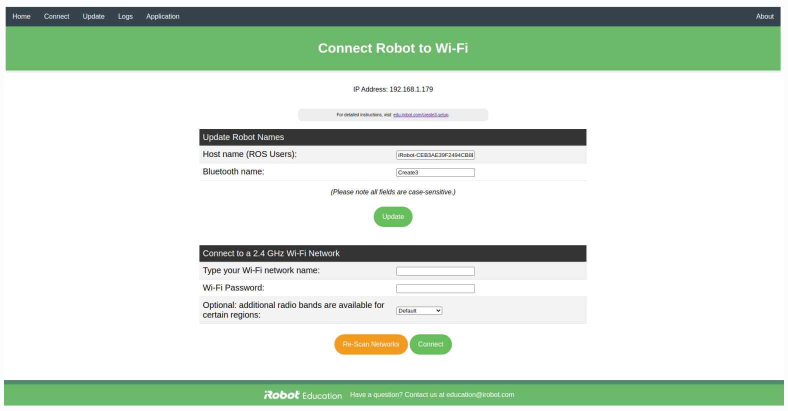
Current Firmware Version: G.2.1

Update to the latest firmware

Or, click here to update from firmware file

Putting the Create® 3 in AP mode

- The Create® 3 is now in AP mode. Connect to its WiFi network called 'Create-XXXX'
- In a browser go to `192.168.10.1`
- Go to the Connect tab, enter your WiFi ssid and password, and then click 'Connect'



Connect Robot to Wi-Fi

IP Address: 192.168.1.179

For detailed instructions, visit edu.irobot.com/create3-setup.

Update Robot Names	
Host name (ROS Users):	iRobot-CEB3AE39F2494CB8I
Bluetooth name:	Create3

(Please note all fields are case-sensitive.)

Connect to a 2.4 GHz Wi-Fi Network	
Type your Wi-Fi network name:	
Wi-Fi Password:	
Optional: additional radio bands are available for certain regions:	Default

iRobot Education Have a question? Contact us at education@irobot.com

Connecting the Create® 3 to WiFi

- Wait for it to connect to WiFi and play a chime
- On your PC, run `ros2 topic list` to ensure that the Create® 3 is publishing its topics

TurtleBot 4 Controller Setup

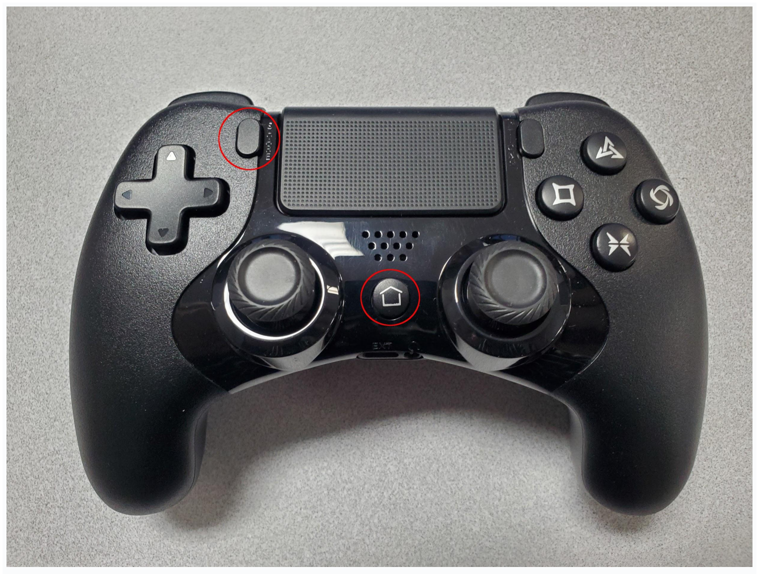
The TurtleBot 4 comes with an included TurtleBot 4 Controller. It is paired in advance with the Raspberry Pi.

If you wish to manually pair a controller, follow these instructions:

- SSH into the TurtleBot 4

```
sudo bluetoothctl --agent=NoInputNoOutput
```

- The `bluetoothd` CLI interface will start.
- Type `scan on` and press enter.
- Press and hold both the home and share buttons on the TurtleBot 4 controller until the light starts blinking.



Putting the TurtleBot 4 in pair mode

- In the CLI look for a *Wireless Controller* device to be found. It will have a MAC address similar to `A0:5A:5C:DF:4D:7F`.
- Copy the MAC address.
- In the CLI enter `trust MAC_ADDRESS`, replacing `MAC_ADDRESS` with the controllers address.
- Then, enter `pair MAC_ADDRESS`.
- Finally, enter `connect MAC_ADDRESS`.
- The CLI should report that the controller has been connected and the light on the controller will turn blue.
- Enter `exit` to exit the CLI.

Updating the TurtleBot 4

It is recommended to update both the Create® 3 and the Raspberry Pi when you first use it to receive the latest bug fixes and improvements.

Create® 3

Check the [Create® 3 software releases] to see if a newer firmware version is available. You can check the firmware version of your robot by visiting the Create® 3's webserver.

Update over WiFi

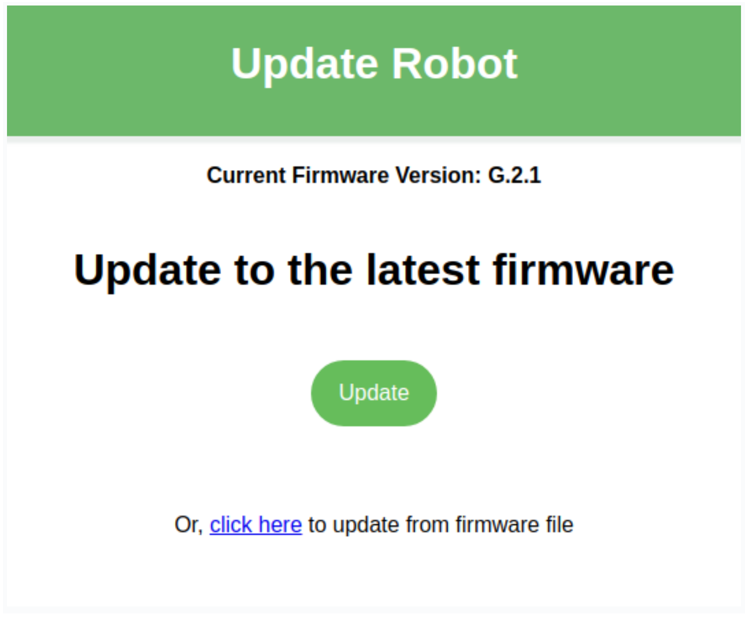
The Create® 3 can be updated through its webserver. There are two options to connect to the webserver:

Find the IP address of the Create® 3 on your WiFi network.

This can be done by going to your routers portal and viewing connected devices. You should see the Create® 3 in your Wireless Clients if it is connected.

Enter the IP address into a browser. You will be taken to the Create® 3 webserver.

Go to the Update tab and click the 'Update' button. The robot will automatically download and install the latest firmware.


Update Robot
Current Firmware Version: G.2.1
Update to the latest firmware
Update
Or, click here to update from firmware file

Updating the Create® 3 over WiFi

Place the robot into AP mode.

If you cannot find the IP address of the Create® 3 on your WiFi network, you can alternatively put it into AP mode and connect directly to it with your PC:

- Download the latest firmware from http://edu.irobot.com/create3-latest-fw.
- Place the robot into AP mode and access the webserver. See Create® 3 WiFi Setup.

- Go to the Update tab and click on the link to update from firmware file.
- Upload the latest firmware and wait for the robot to be updated.

Update over USB-C

Download the latest firmware from http://edu.irobot.com/create3-latest-fw.

Copy the firmware to the Raspberry Pi:

```
sudo scp ~/Downloads/Create3-G.X.Y.swu ubuntu@xxx.xxx.xxx.xxx:/home/ubuntu/
```

SSH into the Raspberry Pi and update the Create® 3 firmware over USB-C:

```
sudo create_update.sh Create3-G.X.Y.swu
```

or

```
curl -X POST --data-binary @Create3-G.X.Y.swu http://192.168.186.2/api/firmware-update
```

This may take a few minutes.

Debian packages

Debian packages can be updated by calling:

```
sudo apt update
sudo apt install <PACKAGE>
```

For example, updating the `turtlebot4_desktop` package can be done like this:

```
sudo apt update
sudo apt install ros-galactic-turtlebot4-desktop
```

Source packages

To update a source package you will need to use a terminal to manually pull changes.

For example, updating the `turtlebot4_robot` package on the `galactic` branch:

```
cd ~/turtlebot4_ws/src/turtlebot4_robot
git checkout galactic
git pull origin galactic
```

You will then need to rebuild the packages:

```
cd ~/turtlebot4_ws
source /opt/ros/galactic/setup.bash
colcon build --symlink-install
source install/setup.bash
```

Install latest Raspberry Pi image

Warning

Installing a new image on the Raspberry Pi will delete any changes you may have made. Save your changes before proceeding.

If you wish to install the latest image onto your robot, follow these instructions.

The latest TurtleBot 4 Raspberry Pi images are available at http://download.ros.org/downloads/turtlebot4/.

- Download the latest image for your robot model and extract it.
- Power off your robot and then remove the microSD card from the Raspberry Pi.
- Insert the microSD card into your PC. You may need an adapter.
- Install the imaging tool `dcfldd`

```
sudo apt install dcfldd
```

- Identify your SD card

```
sudo fdisk -l
```

- The SD card should have a name like `/dev/mmcblk0` or `/dev/sda`.
- If you wish to backup your current image, do so now:

```
sudo dd if=/dev/<SD_NAME> of=<IMAGE_PATH> bs=1M
```

Note

SD_NAME is the device name (`mmcblk0`, `sda`, etc.).

IMAGE_PATH is the path to where you want the image saved – e.g., `~/turtlebot4_images/backup_image`.

- Get the SD flash script from `turtlebot4_setup` and flash the SD card:

```
wget
https://raw.githubusercontent.com/turtlebot/turtlebot4_setup/galactic/scripts/sd_flash
.sh
bash sd_flash.sh /path/to/downloaded/image
```

- Follow the instructions and wait for the SD card to be flashed. Remove the SD card from your PC.
- Ensure your Raspberry Pi 4 is not powered on before inserting the flashed SD card.
- Follow WiFi Setup to configure your WiFi.

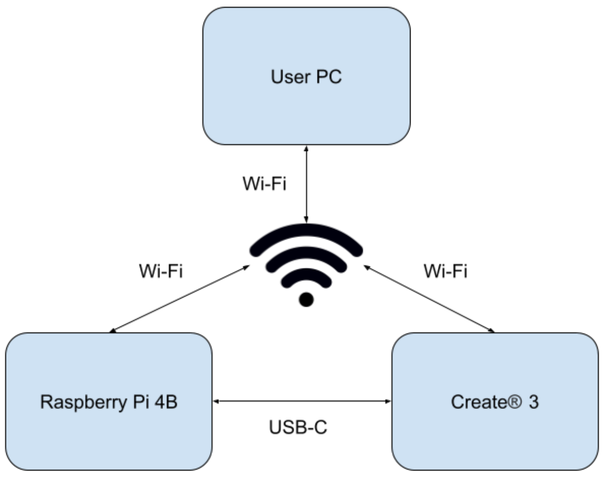
Software

- Overview
- TurtleBot 4 Packages
- Sensors
- Rviz2
- SLAM
- Nav2
- Simulation

Overview

The TurtleBot 4 runs on Ubuntu 20.04 LTS (Focal Fossa) and currently only supports ROS2 Galactic. The TurtleBot 4 software is entirely open source under the Apache 2.0 license, and is available on the TurtleBot Github.

There are 2 main computers that run software used by TurtleBot 4: the onboard Raspberry Pi 4, the Create® 3 onboard processor. The user can also connect to the robot with their own PC to visualise sensor data, configure the robot, and more. Each computer is required to run Ubuntu 20.04 with ROS2 Galactic.



TurtleBot 4 computer connections

Create® 3

The Create® 3 exposes ROS2 topics, actions, and services over both WiFi and the USB-C cable powering the Raspberry Pi. This gives users access to the battery state, sensor data, docking actions, and more. While the Create® 3 can be used with just the USB-C interface, in order to view the robot model on Rviz or run software such as SLAM or Nav2 from a user PC, the Create® 3 will require a WiFi connection.

Raspberry Pi 4

The Raspberry Pi 4 on both the TurtleBot 4 and TurtleBot 4 Lite comes preinstalled with Ubuntu 20.04 Server, ROS2 Galactic, and TurtleBot 4 software. The latest TurtleBot 4 images can be found here. The purpose of the Raspberry Pi 4 is to run the TurtleBot 4 ROS nodes, run sensor ROS nodes, use bluetooth to connect to the TurtleBot 4 controller, access GPIO, and more.

User PC

The user's PC is used to configure the robot, visualise sensor data, run the TurtleBot 4 simulation, and run additional software. The PC is required to run Ubuntu 20.04 with ROS2 Galactic installed, or to use a Virtual Machine running Ubuntu 20.04. A typical laptop or desktop will offer significantly higher processing performance than the Raspberry Pi can, so running applications such as Nav2 or SLAM on the PC will provide significant performance improvements.

TurtleBot 4 Packages

The TurtleBot 4 has 4 main repositories for software: turtlebot4, turtlebot4_robot, turtlebot4_desktop, and turtlebot4_simulator. Each repository is also a metapackage and contains one or more ROS2 packages.

TurtleBot 4

The turtlebot4 repository contains common packages that are used by both turtlebot4_robot and turtlebot4_simulator.

Installation

Source code is available here.

Note

The turtlebot4 packages are automatically installed when either of turtlebot4_robot or turtlebot4_simulator is installed.

Debian installation

Individual packages can be installed through apt:

```
sudo apt update
sudo apt install ros-galactic-turtlebot4-description \
ros-galactic-turtlebot4-msgs \
ros-galactic-turtlebot4-navigation \
ros-galactic-turtlebot4-node
```

Source installation

To manually install this metapackage from source, clone the git repository:

```
cd ~/turtlebot4_ws/src
git clone https://github.com/turtlebot/turtlebot4.git
```

Install dependencies:

```
cd ~/turtlebot4_ws
vcs import src < src/turtlebot4/dependencies.repos
rosdep install --from-path src -yi
```

Build the packages:

```
source /opt/ros/galactic/setup.bash
colcon build --symlink-install
```

Description

The turtlebot4_description package contains the URDF description of the robot and the mesh files for each component.

The description can be published with the robot_state_publisher.

Messages

The turtlebot4_msgs package contains the custom messages used on the TurtleBot 4:

- UserButton: User Button states.
- UserLed: User Led control.
- UserDisplay: User Display data.

The TurtleBot 4 can also use all of the actions, messages, and services that the iRobot® Create® 3 platform supports:

Actions

- AudioNoteSequence: Play a given set of notes from the speaker for a given number of iterations.
- DockServo: Command the robot to dock into its charging station.
- DriveArc: Command the robot to drive along an arc defined by radius.
- DriveDistance: Command the robot to drive a defined distance in a straight line.
- LedAnimation: Command the lights to perform specified animation.
- NavigateToPosition: Command the robot to drive to a goal odometry position using simple approach that rotates to face goal position then translates to goal position then optionally rotates to goal heading.
- RotateAngle: Command the robot to rotate in place a specified amount.
- Undock: Command the robot to undock from its charging station.
- WallFollow: Command the robot to wall follow on left or right side using bump and IR sensors.

Messages

- AudioNote: Command the robot to play a note.
- AudioNoteVector: Command the robot to play a sequence of notes.

- Button: Status for a button.
- Dock: Information about the robot sensing the its dock charging station.
- HazardDetection: An hazard or obstacle detected by the robot.
- HazardDetectionVector: All the hazards and obstacles detected by the robot.
- InterfaceButtons: Status of the 3 interface buttons on the Create® robot faceplate.
- IrIntensity: Reading from an IR intensity sensor.
- IrIntensityVector: Vector of current IR intensity readings from all sensors.
- IrOpcode: Opcode detected by the robot IR receivers. Used to detect the dock and virtual walls.
- KidnapStatus: Whether the robot has been picked up off the ground.
- LedColor: RGB values for an LED.
- LightringLeds: Command RGB values of 6 lightring lights.
- Mouse: Reading from a mouse sensor.
- SlipStatus: Whether the robot is currently slipping or not.
- StopStatus: Whether the robot is currently stopped or not.
- WheelStatus: Current/PWM readings from the robot's two wheels in addition to whether wheels are enabled.
- WheelTicks: Reading from the robot two wheels encoders.
- WheelVels: Indication about the robot two wheels current speed.

Services

- EStop: Set system EStop on or off, cutting motor power when on and enabling motor power when off.
- RobotPower: Power off robot.

See irobot_create_msgs for more details.

Note

When publishing or subscribing to topics, make sure that the QoS that you use matches that of the topic.

Navigation

The turtlebot4_navigation packages contains launch and configuration files for using SLAM and navigation on the TurtleBot 4. It also contains the TurtleBot 4 Navigator Python node.

Launch files:

- Nav Bringup: Launches navigation. Allows for launch configurations to use SLAM, Nav2, and localization.
- SLAM Sync: Launches slam_toolbox with online synchronous mapping. Recommended for use on a PC.

- SLAM Async: Launches slam_toolbox with online asynchronous mapping. Recommended for use on the Raspberry Pi 4.

Nav Bringup launch configuration options:

- **namespace**: Top-level namespace.
    - default: *None*
- **use_namespace**: Whether to apply a namespace to the navigation stack.
    - options: *true, false*
    - default: *false*
- **slam**: Whether to launch SLAM.
    - options: *off, sync, async*
    - default: *off*
- **localization**: Whether to launch localization.
    - options: *true, false*
    - default: *false*
- **nav2**: Whether to launch Nav2.
    - options: *true, false*
    - default: *false*
- **map**: Full path to map yaml file to load.
    - default: */path/to/turtlebot4_navigation/maps/depot.yaml*
- **use_sim_time**: Use simulation (Gazebo) clock if true.
    - options: *true, false*
    - default: *false*
- **param_file**: Full path to the ROS2 parameters file to use for nav2 and localization nodes.
    - default: */path/to/turtlebot4_navigation/config/nav2.yaml*
- **autostart**: Automatically startup the nav2 stack.
    - options: *true, false*
    - default: *true*
- **use_composition**: Whether to use composed bringup.
    - options: *true, false*
    - default: *true*

Running synchronous SLAM:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py slam:=sync
```

Running asynchronous SLAM with Nav2:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py slam:=async
```

Running Nav2 with localization and existing map:

ros2 launch turtlebot4_navigation nav_bringup.launch.py localization:=true slam:=off map:=/path/to/map.yaml

TurtleBot 4 Navigator

The TurtleBot 4 Navigator is a Python node that adds TurtleBot 4 specific functionality to the Nav2 Simple Commander. It provides a set of Python methods for navigating the TurtleBot 4. This includes docking, navigating to a pose, following waypoints, and more. Visit the Navigation Tutorials for examples.

Node

The turtlebot4_node package contains the source code for the rclcpp node turtlebot4_node that controls the robots HMI as well as other logic. This node is used by both the physical robot and the simulated robot.

Publishers:

- **/hmi/display**: *turtlebot4_msgs/msg/UserDisplay*
  - description: The current information that is to be displayed (TurtleBot 4 model only).
- **/ip**: *std_msgs/msg/String*
  - description: The IP address of the Wi-Fi interface.

Subscribers:

- **/battery_state**: *sensor_msgs/msg/BatteryState*
  - description: Current battery state of the Create® 3.
- **/hmi/buttons**: *turtlebot4_msgs/msg/UserButton*
  - description: Button states of the TurtleBot 4 HMI (TurtleBot 4 model only).
- **/hmi/display/message**: *std_msgs/msg/String*
  - description: User topic to print custom message to display (TurtleBot 4 model only).
- **/hmi/led**: *turtlebot4_msgs/msg/UserLed*
  - description: User topic to control User LED 1 and 2 (TurtleBot 4 model only).
- **/interface_buttons**: *irobot_create_msgs/msg/InterfaceButtons*
  - description: Button states of Create® 3 buttons.
- **/joy**: *sensor_msgs/msg/Joy*
  - description: Bluetooth controller button states (TurtleBot 4 model only).
- **/wheel_status**: *irobot_create_msgs/msg/WheelStatus*
  - description: Wheel status reported by Create® 3.

Service Clients:

- **/e_stop**: *irobot_create_msgs/srv/EStop*

    - description: Enable or disable motor stop.
- **/robot_power**: *irobot_create_msgs/srv/RobotPower*
    - description: Power off the robot.

Action Clients:

- **/dock**: *irobot_create_msgs/action/DockServo*
    - description: Command the robot to dock into its charging station.
- **/wall_follow**: *irobot_create_msgs/action/WallFollow*
    - description: Command the robot to wall follow on left or right side using bump and IR sensors.
- **/undock**: *irobot_create_msgs/action/Undock*
    - description: Command the robot to undock from its charging station.

Functions

The node has a set of static functions that can be used either with a button or through the display menu.

Currently, the supported functions are:

- **Dock**: Call the */dock* action.
- **Undock**: Call the */undock* action.
- **Wall Follow Left**: Call the */wall_follow* action with direction FOLLOW_LEFT and a duration of 10 seconds.
- **Wall Follow Right**: Call the */wall_follow* action with direction FOLLOW_RIGHT and a duration of 10 seconds.
- **Power**: Call the */robot_power* service and power off the robot.
- **EStop**: Call the */e_stop* service and toggle EStop state.

The TurtleBot 4 also supports the following menu functions:

- **Scroll Up**: Scroll menu up.
- **Scroll Down**: Scroll menu down.
- **Back**: Exit message screen or return to first menu entry.
- **Select**: Select currently highlighted menu entry.
- **Help**: Print help statement.

Configuration

This node can be configured using a parameter *.yaml* file. The default robot parameters can be found here.

Parameters:

- **wifi.interface**: The Wi-Fi interface being used by the computer. This is used to find the current IP address of the computer.
- **menu.entries**: Set menu entries to be displayed. Each entry must be one of the support functions.
- **buttons**: Set the function of Create® 3 and HMI buttons.
- **controller**: Set the function of TurtleBot 4 Controller buttons.

Buttons

The Buttons class in turtlebot4_node provides functionality to all buttons on the robot. This includes the Create® 3 buttons, HMI buttons, and TurtleBot 4 Controller buttons. The node receives button states from the */interface_buttons*, */hmi/buttons*, and */joy* topics.

Each button can be configured to have either a single function when pressed, or two functions by using a short or long press. This is done through configuration.

Supported buttons:

```
buttons:
  create3_1:
  create3_power:
  create3_2:
  hmi_1:
  hmi_2:
  hmi_3:
  hmi_4:

controller:
  a:
  b:
  x:
  y:
  up:
  down:
  left:
  right:
  l1:
  l2:
  l3:
  r1:
  r2:
  r3:
  share:
  options:
  home:
```

Example

Lets say we want the TurtleBot 4 to have the following button functions:

- Make a short press of Create® 3 button 1 toggle EStop.
- Power off robot with 5 second press of Home on the TurtleBot 4 Controller.
- Short press of HMI button 1 performs Wall Follow Left, long press of 3 seconds performs Wall Follow Right.

Create a new yaml file:

```
cd /home/ubuntu/turtlebot4_ws
touch example.yaml
```

Use your favourite text editor and paste the following into example.yaml:

```
turtlebot4_node:
  ros__parameters:
    buttons:
      create3_1: ["EStop"]
      hmi_1: ["Wall Follow Left", "Wall Follow Right", "3000"]

    controller:
      home: ["Power", "5000"]
```

Launch the robot with your new configuration:

```
ros2 launch turtlebot4_bringup standard.launch.py
param_file:=/home/ubuntu/turtlebot4_ws/example.yaml
```

The buttons should now behave as described in example.yaml.

LEDs

The Leds class in turtlebot4_node controls the states of the HMI LEDs on the TurtleBot 4. It is not used for the TurtleBot 4 Lite.

There are 5 status LEDs which are controlled by the node: POWER, MOTOR, COMMS, WIFI, and BATTERY. There are also 2 user LEDs: USER_1 and USER_2 which are controlled by the user via the */hmi/led* topic. The BATTERY and USER_2 LEDs consist of a red and green LED which allows them to be turned on as either green, red, or yellow (red + green). The rest are green only.

Status LEDs:

- **POWER**: Always ON while turtlebot4_node is running.
- **MOTOR**: ON when wheels are enabled, OFF when wheels are disabled.
    - Wheel status is reported on the */wheel_status* topic.
- **COMMS**: ON when communication with Create® 3 is active. OFF otherwise.
    - Receiving data on the */battery_state* topic implies that communication is active.
- **WIFI**: ON when an IP address can be found for the Wi-Fi interface specified in the configuration.
- **BATTERY**: Colour and pattern will vary based on battery percentage.
    - Battery percentage is received on */battery_state* topic.

User LEDs:

The user LEDs can be set by publishing to the */hmi/led* topic with a UserLed message.

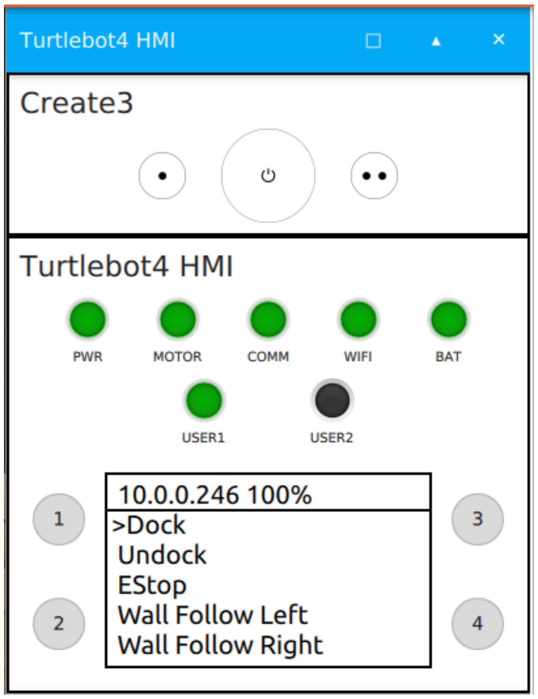
UserLed message:

- **led**: Which available LED to use.
    - uint8 USER_LED_1 = 0
    - uint8 USER_LED_2 = 1
- **color**: Which color to set the LED to.
    - uint8 COLOR_OFF = 0
    - uint8 COLOR_GREEN = 1
    - uint8 COLOR_RED = 2
    - uint8 COLOR_YELLOW = 3
- **blink_period**: Blink period in milliseconds.
    - uint32 ms
- **duty_cycle**: Percentage of blink period that the LED is ON.
    - float64 (0.0 to 1.0)

Examples

Set USER_1 to solid green:

```
ros2 topic pub /hmi/led turtlebot4_msgs/msg/UserLed "led: 0
color: 1
blink_period: 1000
duty_cycle: 1.0" --once
```

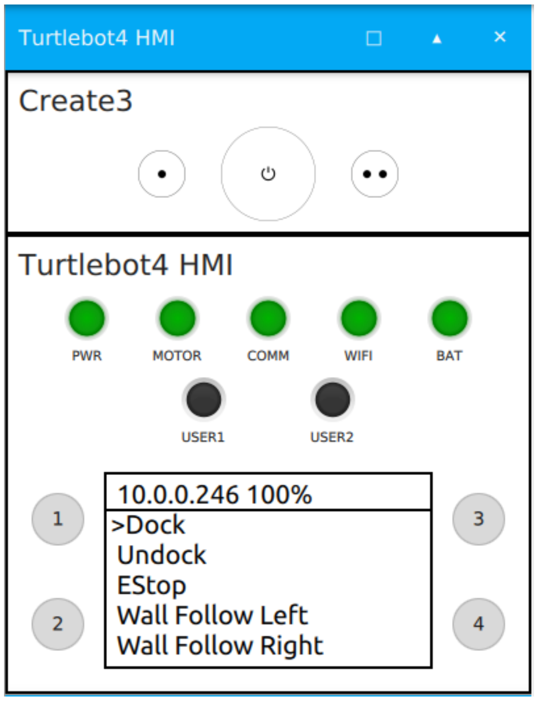


10.0.0.246 100%
>Dock
Undock
EStop
Wall Follow Left
Wall Follow Right

User 1: Solid Green

Set USER_1 OFF:

```
ros2 topic pub /hmi/led turtlebot4_msgs/msg/UserLed "led: 0
color: 0
blink_period: 1000
duty_cycle: 1.0" --once
```

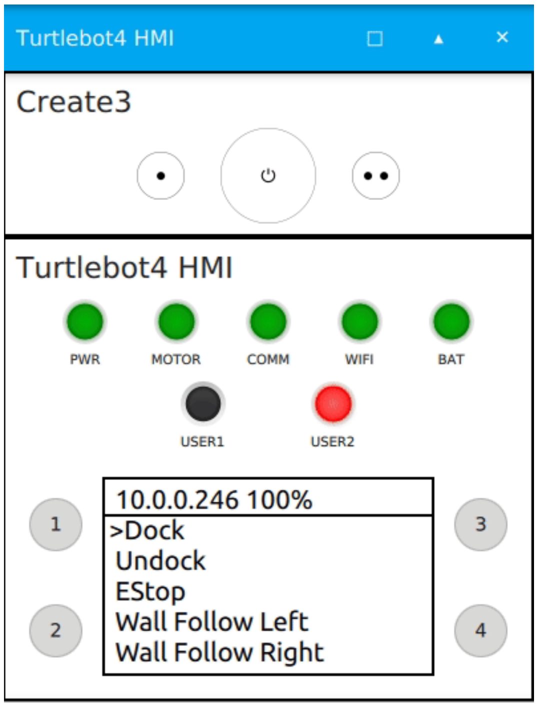


10.0.0.246 100%
>Dock
Undock
EStop
Wall Follow Left
Wall Follow Right

User 1: Off

Blink USER_2 red at 1hz with 50% duty cycle:

```
ros2 topic pub /hmi/led turtlebot4_msgs/msg/UserLed "led: 1
color: 2
blink_period: 1000
duty_cycle: 0.5" --once
```



User 2: Red, 1hz, 50%

Display

The Display class in turtlebot4_node controls the HMI display of the TurtleBot 4. The physical display is a 128x64 OLED which is controlled over I2C with a SSD1306 driver.

The display has a header line which contains the IP address of the Wi-Fi interface specified in configuration, as well as the battery percentage received on the */battery_state* topic. The display also has 5 additional lines which are used for the menu by default. The menu entries are specified in configuration and are a set of the available functions. The 5 menu lines can be overwritten by publishing to the */hmi/display/message* with a **String** message.

Note

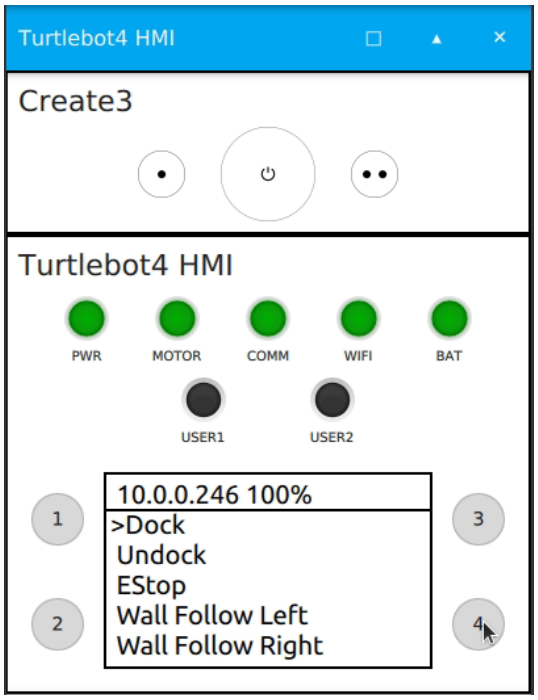
The menu can have any number of entries. If there are more than 5 entries, the user will have to scroll down to see the entries that do not fit on the 5 menu lines.

Menu Control

The TurtleBot 4 display has a simple scrolling menu. There are 4 control functions for the menu: Scroll up, Scroll down, Select, and Back.

- Scroll up and down allow the users to navigate through the menu entries and by default are mapped to user buttons 3 and 4 respectively.

- The select function will call the currently selected menu entry. This can trigger an action such as docking, a service such as EStop, or display a message such as the Help message. This function is mapped to user button 1 by default.
- The back function allows the user to return back to the menu from a message screen. If the menu is already showing the menu entries, it will return to showing the first 5 menu entries and the first entry will be highlighted.



```
10.0.0.246 100%
>Dock
 Undock
 EStop
 Wall Follow Left
 Wall Follow Right
```

TurtleBot 4 Menu Controls

TurtleBot 4 Robot

Source code is available here.

Note

The turtlebot4_robot metapackage can be installed on a Raspberry Pi 4B running Ubuntu Server 20.04 with ROS2 Galactic.

Installation

The turtlebot4_robot metapackage is pre-installed on the TurtleBot 4 Raspberry Pi image.

Source installation

To manually install this metapackage from source, clone the git repository:

```
cd ~/turtlebot4_ws/src
git clone https://github.com/turtlebot/turtlebot4_robot.git
```

Install dependencies:

```
cd ~/turtlebot4_ws
vcs import src < src/turtlebot4_robot/dependencies.repos
rosdep install --from-path src -yi
```

Build the packages:

```
source /opt/ros/galactic/setup.bash
colcon build --symlink-install
```

Base

The turtlebot4_base package contains the source code for the rclcpp node turtlebot4_base_node which runs on the physical robot. This node interfaces with the GPIO lines of the Raspberry Pi which allows it to read the state of the buttons, as well as write to the LEDs and display.

Publishers:

- **/hmi/buttons**: *turtlebot4_msgs/msg/UserButton*
  - description: Button states of the TurtleBot 4 HMI (TurtleBot 4 model only).

Subscribers:

- **/hmi/display**: *turtlebot4_msgs/msg/UserDisplay*
  - description: The current information that is to be displayed (TurtleBot 4 model only).
- **/hmi/led/_<led>**: *std_msgs/msg/Int32*
  - description: Hidden topics indicating the state of each LED.

GPIO Interface

The TurtleBot 4 uses *libgpiod* to interface with the GPIO lines of the Raspberry Pi. The gpiochip0 device represents the 40-pin header of the Raspberry Pi and is used for reading and writing to these pins.

I2C Interface

The linux I2C drivers are used to read and write data on the I2C buses of the Raspberry Pi. The display's SSD1306 driver is connected to the i2c-3 device by default, but other buses are available too.

SSD1306

The SSD1306 is a driver for OLED displays. It receives commands over a communication bus (I2C for the TurtleBot 4) and controls how the physical display behaves. The TurtleBot 4 uses a modified version of this STM32 SSD1306 driver to write pixels, shapes and characters to the display.

Configuration

Warning

Do NOT change pin definitions if you are using the standard PCBA or do not know what you are doing.

The turtlebot4_base_node pin definitions can be set with ROS parameters. The default configuration is:

```
turtlebot4_base_node:
  ros__parameters:
    # GPIO definition for HMI. Do NOT change if you are using the standard PCBA.
    gpio:
      user_button_1: 13
      user_button_2: 19
      user_button_3: 16
      user_button_4: 26

      led_green_power: 17
      led_green_motors: 18
      led_green_comms: 27
      led_green_wifi: 24
      led_green_battery: 22
      led_red_battery: 23
      led_green_user_1: 25
      led_green_user_2: 6
      led_red_user_2: 12

      display_reset: 2
```

Note

The value for each GPIO device is the GPIO number, NOT the pin number.

Robot Upstart

The robot uses the robot_upstart package to install the bringup launch files as a background process that launches when the robot starts. The launch files are located under the turtlebot4_bringup package.

To check if the TurtleBot 4 service is running, use this command on the Raspberry Pi:

```
systemctl | grep turtlebot4
```

If the service is active, the CLI will echo turtlebot4.service loaded active running "bringup turtlebot4".

To read the most recent logs from the service, call:

```
sudo journalctl -u turtlebot4 -r
```

To stop the service, call:

```
sudo systemctl stop turtlebot4.service
```

This will kill all of the nodes launched by the launch file.

Note

The service will automatically start again on reboot. To fully disable the service, uninstall the job.

To start the service again, call:

```
sudo systemctl start turtlebot4.service
```

The launch files are installed on the TurtleBot 4 with this command:

```
ros2 run robot_upstart install turtlebot4_bringup/launch/standard.launch.py --job turtlebot4 --rmw
rmw_cyclonedds_cpp --rmw_config /etc/cyclonedds_rpi.xml
```

and on the TurtleBot 4 Lite with this command:

```
ros2 run robot_upstart install turtlebot4_bringup/launch/lite.launch.py --job turtlebot4 --rmw
rmw_cyclonedds_cpp --rmw_config /etc/cyclonedds_rpi.xml
```

To uninstall, use this command:

```
ros2 run robot_upstart uninstall turtlebot4
```

Once uninstalled, the launch file will no longer be launched on boot.

Bringup

The turtlebot4_bringup package contains the launch and configuration files to run the robots software.

Launch files:

- Joy Teleop: Launches nodes to enable the bluetooth controller.
- OAKD: Launches the OAK-D nodes.
- RPLIDAR: Launches the RPLIDAR node.
- Robot: Launches the TurtleBot 4 nodes.
- Lite: Launches all necessary nodes for the TurtleBot 4 Lite.
- Standard: Launches all necessary nodes for the TurtleBot 4.

Config files:

- TurtleBot 4 Controller: Configurations for the TurtleBot 4 controller.
- TurtleBot 4: Configurations for the turtlebot4_node and turtlebot4_base_node.

Diagnostics

The turtlebot4_diagnostics packages contains the source code and launch files for the TurtleBot 4 diagnostics updater.

Launch files:

- **Diagnostics**: Launches the turtlebot4 diagnostics updater and the diagnostic aggregator node.

Diagnostics Updater

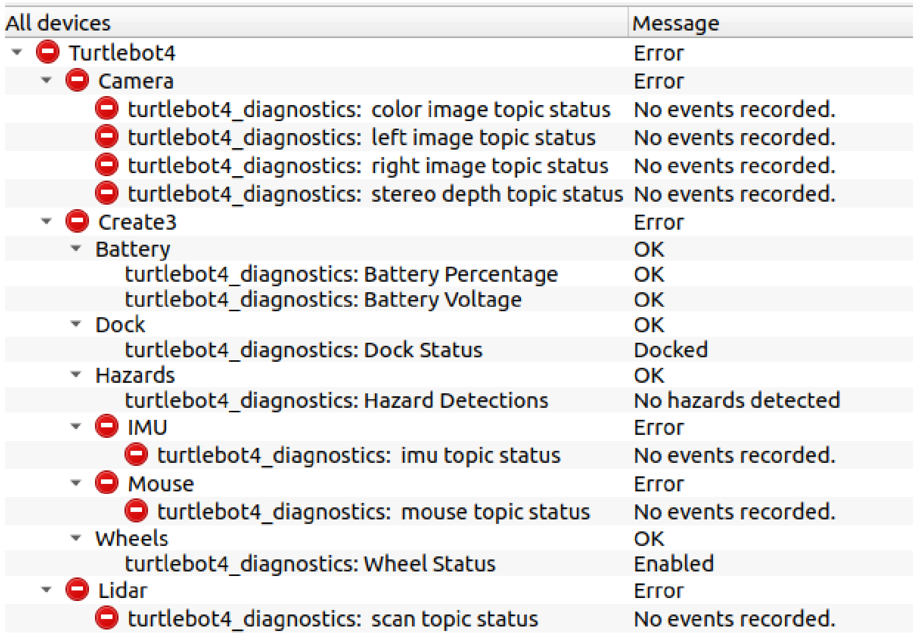
The diagnostics updater is a Python3 node that runs on the robot. It subscribes to diagnostic topics records statistics specific to each topic. The diagnostic data is viewable with rqt_robot_monitor.

Diagnostic topics:

- **/battery_state**: Check battery voltage and percentage.
- **/wheel_status**: Check if wheels are enabled.
- **/dock**: Check if the robot is docked.
- **/scan**: Check the frequency of laser scans from the RPLIDAR.
- **/left/image**: Check the frequency of images from the left OAK-D camera.
- **/right/image**: Check the frequency of images from the right OAK-D camera.
- **/color/image**: Check the frequency of images from the OAK-D colour sensor.
- **/stereo/image**: Check the frequency of depth images from the OAK-D.
- **/hazard_detection**: Check for detected hazards.
- **/imu**: Check the frequency of IMU messages.
- **/mouse**: Check the frequency of Mouse messages.

Viewing diagnostics:

ros2 launch turtlebot4_viz view_diagnostics.launch.py



All devices	Message
Turtlebot4	Error
Camera	Error
turtlebot4_diagnostics: color image topic status	No events recorded.
turtlebot4_diagnostics: left image topic status	No events recorded.
turtlebot4_diagnostics: right image topic status	No events recorded.
turtlebot4_diagnostics: stereo depth topic status	No events recorded.
Create3	Error
Battery	OK
turtlebot4_diagnostics: Battery Percentage	OK
turtlebot4_diagnostics: Battery Voltage	OK
Dock	OK
turtlebot4_diagnostics: Dock Status	Docked
Hazards	OK
turtlebot4_diagnostics: Hazard Detections	No hazards detected
IMU	Error
turtlebot4_diagnostics: imu topic status	No events recorded.
Mouse	Error
turtlebot4_diagnostics: mouse topic status	No events recorded.
Wheels	OK
turtlebot4_diagnostics: Wheel Status	Enabled
Lidar	Error
turtlebot4_diagnostics: scan topic status	No events recorded.

Diagnostics data captured with rqt_robot_monitor

Tests

The turtlebot4_tests packages contains the source code for the TurtleBot 4 system test scripts. These scripts test basic functionality of the robot and are useful for troubleshooting issues.

ROS Tests

The ROS tests use ROS topics and actions to test various system functionality. Test results are saved to ~/turtlebot4_test_results/Y_m_d-H_M_S where Y_m_d-H_M_S is the date and time of the test. A rosbag is also recorded for the duration of the test and saved to the same location.

Currently supported tests:

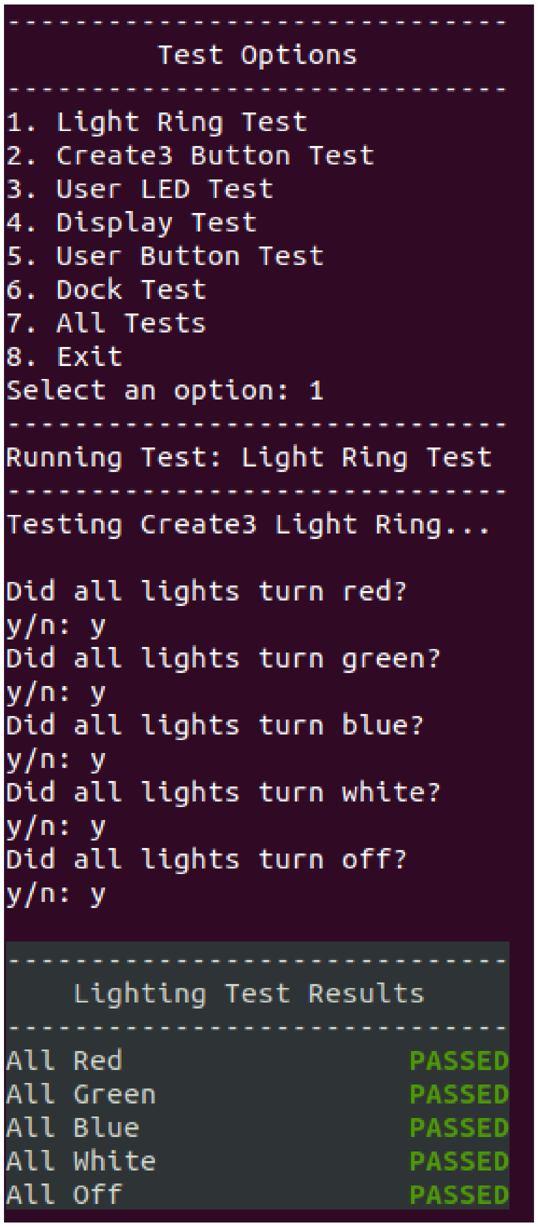
- **Light Ring**: Test the Create® 3 light ring
- **Create® 3 Button**: Test the Create® 3 buttons
- **User LED**: Test the HMI LEDs (TurtleBot 4 model only)
- **User Button**: Test the HMI buttons (TurtleBot 4 model only)
- **Display**: Test the HMI display (TurtleBot 4 model only)
- **Dock**: Test the robots ability to undock and dock.

Running the tests:

```
ros2 run turtlebot4_tests ros_tests
```

This will launch a CLI menu where the different tests can be run.

Enter the index of the test and hit enter to start the test. Some tests will run automatically while others require user input.



```
------------------------------
         Test Options
------------------------------
1. Light Ring Test
2. Create3 Button Test
3. User LED Test
4. Display Test
5. User Button Test
6. Dock Test
7. All Tests
8. Exit
Select an option: 1
------------------------------
Running Test: Light Ring Test
------------------------------
Testing Create3 Light Ring...

Did all lights turn red?
y/n: y
Did all lights turn green?
y/n: y
Did all lights turn blue?
y/n: y
Did all lights turn white?
y/n: y
Did all lights turn off?
y/n: y

------------------------------
    Lighting Test Results
------------------------------
All Red                 PASSED
All Green               PASSED
All Blue                PASSED
All White               PASSED
All Off                 PASSED
```

Running the Light Ring test

TurtleBot 4 Desktop

The turtlebot4_desktop metapackage contains packages used for visualising and interfacing with the TurtleBot 4 from a PC.

Installation

Source code is available here.

Note

The turtlebot4_desktop metapackage can be installed on a PC running Ubuntu Desktop 20.04 with ROS2 Galactic.

Debian installation

To install the metapackage through apt:

```
sudo apt update
sudo apt install ros-galactic-turtlebot4-desktop
```

Source installation

To manually install this metapackage from source, clone the git repository:

```
cd ~/turtlebot4_ws/src
git clone https://github.com/turtlebot/turtlebot4_desktop.git
```

Install dependencies:

```
cd ~/turtlebot4_ws
rosdep install --from-path src -yi
```

Build the packages:

```
source /opt/ros/galactic/setup.bash
colcon build --symlink-install
```

Visualisation

The turtlebot4_viz package contains launch files and configurations for viewing the robot in Rviz2, and viewing the diagnostics.

Launch files:

- View Diagnostics: Launches rqt_robot_monitor to view diagnostic data.
- View Model: Launches rviz2. Used to view the model and sensor data.
- View Robot: Launches rviz2. Used to view the robot while navigating.

TurtleBot 4 Simulator

The turtlebot4_simulator metapackage contains packages used to simulate the TurtleBot 4 in Ignition Gazebo.

Installation

Source code is available here.

Note

The turtlebot4_simulator metapackage can be installed on a PC running Ubuntu Desktop 20.04 with ROS2 Galactic.

Dev Tools

```
sudo apt install -y \
python3-colcon-common-extensions \
python3-rosdep \
python3-vcstool
```

Ignition Edifice

Ignition Edifice must be installed:

```
sudo apt-get update && sudo apt-get install wget
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update && sudo apt-get install ignition-edifice
```

Debian installation

To install the metapackage through apt:

```
sudo apt update
sudo apt install ros-galactic-turtlebot4-simulator ros-galactic-irobot-create-nodes
```

Source installation

To manually install this metapackage from source, clone the git repository:

```
cd ~/turtlebot4_ws/src
git clone https://github.com/turtlebot/turtlebot4_simulator.git
```

Install dependencies:

```
cd ~/turtlebot4_ws
vcs import src < src/turtlebot4_simulator/dependencies.repos
rosdep install --from-path src -yi
```

Build the packages:

```
source /opt/ros/galactic/setup.bash
colcon build --symlink-install
```

Ignition Bringup

The turtlebot4_ignition_bringup package contains launch files and configurations to launch Ignition Gazebo.

Launch files:

- Ignition: Launches Ignition Gazebo and all required nodes to run the simulation.
- ROS Ignition Bridge: Launches all of the required ros_ign_bridge nodes to bridge Ignition topics with ROS topics.
- TurtleBot 4 Nodes: Launches the turtlebot4_node and turtlebot4_ignition_hmi_node required to control the HMI plugin and robot behaviour.

Ignition launch configuration options:

- **model**: Which TurtleBot 4 model to use.
    - options: *standard, lite*
    - default: *standard*
- **rviz**: Whether to launch rviz.
    - options: *true, false*
    - default: *false*
- **slam**: Whether to launch SLAM.
    - options: *off, sync, async*
    - default: *off*
- **nav2**: Whether to launch Nav2.
    - options: *true, false*
    - default: *false*
- **param_file**: Path to parameter file for turtlebot4_node.
    - default: */path/to/turtlebot4_ignition_bringup/config/turtlebot4_node.yaml*
- **world**: Which world to use for simulation.
    - default: *depot*
- **robot_name**: What to name the spawned robot.
    - default: *turtlebot4*

Running the simulator with default settings:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py
```

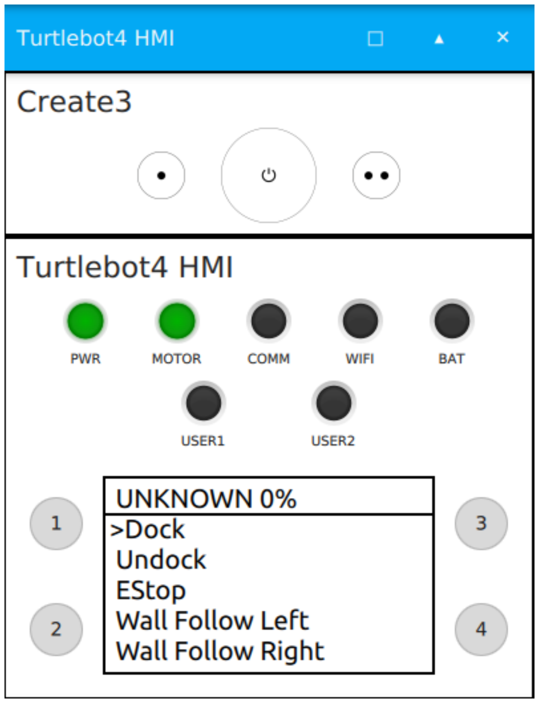
Running synchronous SLAM with Nav2:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py slam:=sync nav2:=true rviz:=true
```

Ignition GUI Plugins

The turtlebot4_ignition_gui_plugins package contains the source code for the TurtleBot 4 HMI GUI plugin.

The TurtleBot 4 HMI GUI plugin is only used for the standard model. The lite model uses the Create® 3 HMI GUI plugin.



TurtleBot 4 HMI GUI plugin

Ignition Toolbox

The turtlebot4_ignition_toolbox package contains the source code for the TurtleBot 4 HMI node. The TurtleBot 4 HMI node acts as a bridge between the turtlebot4_node and ros_ign_bridge to convert the custom TurtleBot 4 messages into standard messages such as Int32 and String.

Sensors

RPLIDAR A1M8

Connecting

The RPLIDAR connects to the TurtleBot 4 with a micro USB to USB-A cable. The sensor does not require high data throughput, so using a USB 2.0 port is sufficient.

Once connected, the RPLIDAR should register on the Raspberry PI as a USB device. If the udev rules are installed, the RPLIDAR will appear as /dev/RPLIDAR. Otherwise it will be /dev/ttyUSB0.

To check that the USB device exists, use the command

```
ls /dev/RPLIDAR
```

If the device exists, the terminal will echo /dev/RPLIDAR.

Installing

The RPLIDAR drivers are installed by default on all TurtleBot 4's. To manually install, run:

```
sudo apt install ros-galactic-rplidar-ros
```

Running

```
ros2 launch turtlebot4_bringup rplidar.launch.py
```

The laserscan will be published to the */scan* topic by default.

OAK-D

Connecting

The OAK-D cameras are connected to the Raspberry Pi with a USB-C to USB-A cable. The cameras requires high data throughput so using a USB 3.0 port is highly recommended.

Installing

The OAK-D drivers are installed by default on all TurtleBot 4's. To manually install, follow the instructions on the DepthAI ROS github.

Running

The default node used by the TurtleBot 4 can be launched:

```
ros2 launch turtlebot4_bringup oakd.launch.py
```

Other nodes are available in the DepthAI ROS examples package.

For example:

```
ros2 launch depthai_examples mobile_publisher.launch.py
```

AI examples are available on the DepthAI github. To view the images from these examples you will need to ssh into the robot with a -X flag.

```
ssh ubuntu@192.168.0.15 -X
```

Create® 3

The Create® 3 comes with several sensors for safety, object detection, and odometry. For more information on the physical location of the sensors, read the Create® 3 Hardware Overview. Hazards detected by the robot are published to the */hazard_detection* topic, although some sensors also have their own individual topics

Cliff

The Create® 3 has 4 cliff sensors located on the front half of the robot. These sensors measure the distance from the robot to the ground, and prevent the robot from falling off of cliffs.

Bumper

The bumper is used by the Create® 3 to detect objects or walls that the robot has run in to. It can trigger reflexes to recoil from the object, or use the information to follow the wall.

Wheeldrop

The wheeldrop is the spring on which the Create® 3 wheels sit. When the robot is lifted off of the ground, the spring is decompressed and the wheeldrop hazard is activated.

IR Proximity

The IR proxmity sensors are located on the front of the bumper and are used for the wall follow action. The sensor data can be viewed on the */ir_intensity* topic.

Slip and Stall

Wheel slip and stall is also detected by the Create® 3. The status can be viewed on the */slip_status* and */stall_status* topics.

Kidnap

The robot uses a fusion of sensor data to detect when it has been picked up and "kidnapped". Motors will be disabled in this state, and will re-enable when placed on the ground again. The */kidnap_status* topic shows the current kidnap state.

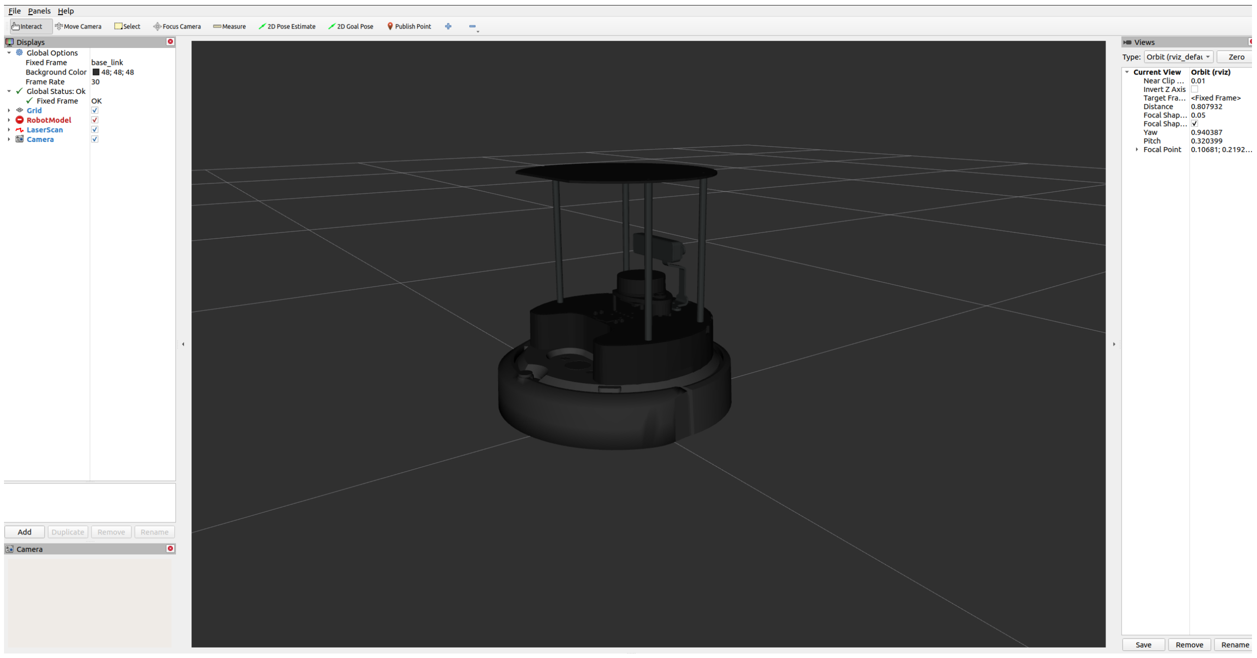
Rviz2

Rviz2 is a port of Rviz to ROS2. It provides a graphical interface for users to view their robot, sensor data, maps, and more. It is installed by default with ROS2 and requires a desktop version of Ubuntu to use.

turtlebot4_desktop provides launch files and configurations for viewing the TurtleBot 4 in Rviz2.

View Model

To inspect the model and sensor data, run ros2 launch turtlebot4_viz view_model.launch.py.



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
LaserScan	✓
Camera	✓

Type: Orbit (rviz_defau

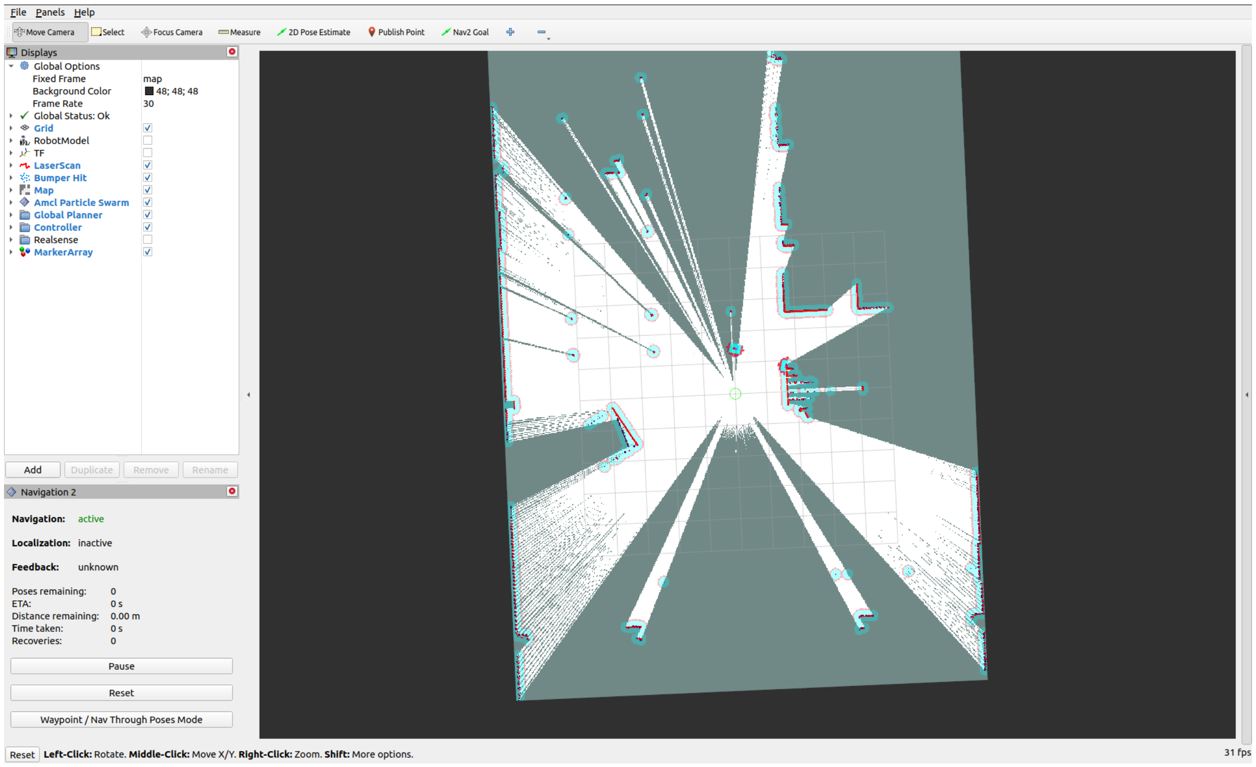
Current View	**Orbit (rviz)**
Near Clip …	0.01
Invert Z Axis	☐
Target Fra…	<Fixed Frame>
Distance	0.807932
Focal Shap…	0.05
Focal Shap…	✓
Yaw	0.940387
Pitch	0.320399
Focal Point	0.10681; 0.2192…

Rviz2 launched with the View Model configuration

View Robot

For a top down view of the robot in its environment, run ros2 launch turtlebot4_viz view_robot.launch.py.

This is useful when mapping or navigating with the robot



Displays

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Grid	✓
RobotModel	☐
TF	☐
LaserScan	✓
Bumper Hit	✓
Map	✓
Amcl Particle Swarm	✓
Global Planner	✓
Controller	✓
Realsense	☐
MarkerArray	✓

Navigation 2

Navigation: active

Localization: inactive

Feedback: unknown

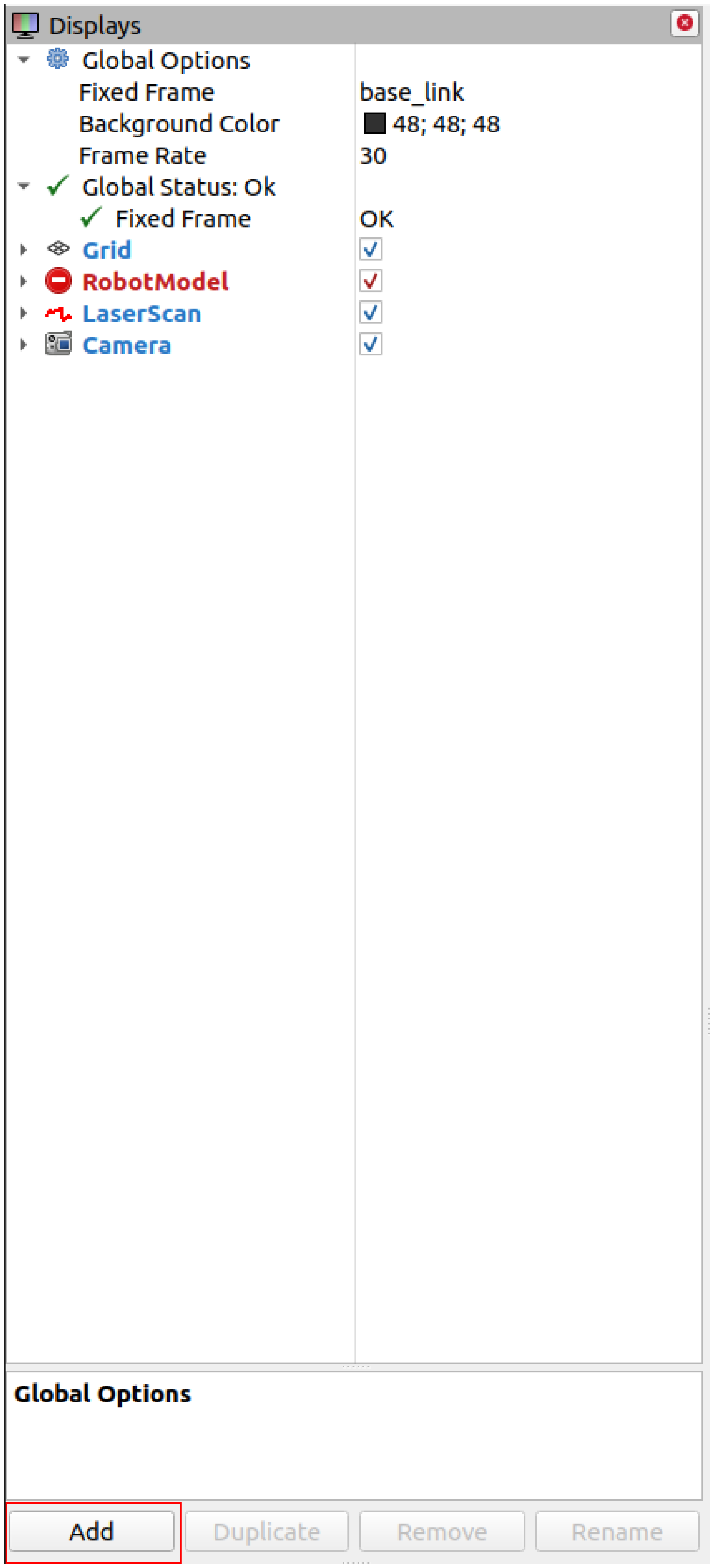
```
Poses remaining:     0
ETA:                 0 s
Distance remaining:  0.00 m
Time taken:          0 s
Recoveries:          0
```

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

Rviz2 launched with the View Robot configuration

Rviz2 Displays

Rviz2 offers support for displaying data from various sources. Displays can be added using the "Add" button.

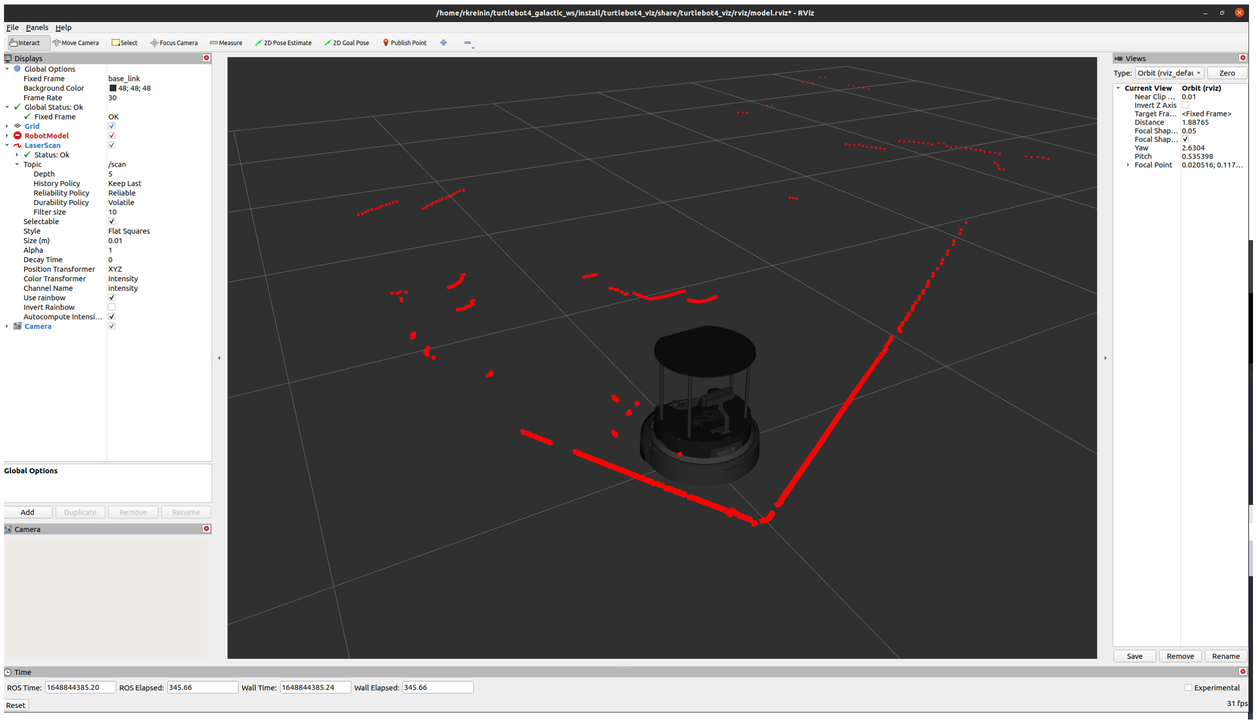


Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
LaserScan	✓
Camera	✓

Adding Displays in Rviz2

LaserScan

The LaserScan display shows data for sensor_msgs/msg/LaserScan messages. On the TurtleBot 4 the RPLIDAR supplies this data on the /scan topic.


/home/rkreinin/turtlebot4_galactic_ws/install/turtlebot4_viz/share/turtlebot4_viz/rviz/model.rviz* - RViz

```
Displays
▾ Global Options
    Fixed Frame                 base_link
    Background Color            ■ 48; 48; 48
    Frame Rate                  30
▾ ✓ Global Status: Ok
      ✓ Fixed Frame             OK
▸ Grid                          ✓
▸ RobotModel                    ✓
▾ LaserScan                     ✓
  ▸ ✓ Status: Ok
  ▾ Topic                       /scan
        Depth                   5
        History Policy          Keep Last
        Reliability Policy      Reliable
        Durability Policy       Volatile
        Filter size             10
    Selectable                  ✓
    Style                       Flat Squares
    Size (m)                    0.01
    Alpha                       1
    Decay Time                  0
    Position Transformer        XYZ
    Color Transformer           Intensity
    Channel Name                intensity
    Use rainbow                 ✓
    Invert Rainbow              ☐
    Autocompute Intensi…        ✓
▸ Camera                        ✓
```

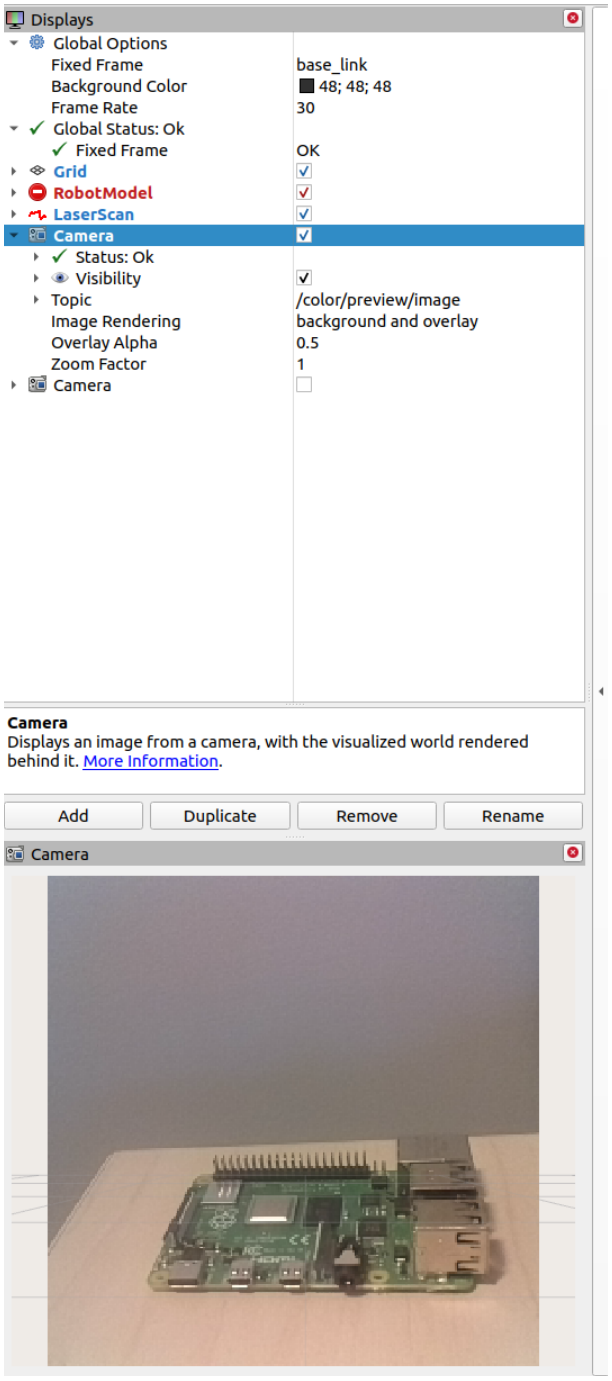
```
Views
Type: Orbit (rviz_defau
▾ Current View      Orbit (rviz)
    Near Clip …     0.01
    Invert Z Axis   ☐
    Target Fra…     <Fixed Frame>
    Distance        1.88765
    Focal Shap…     0.05
    Focal Shap…     ✓
    Yaw             2.6304
    Pitch           0.535398
  ▸ Focal Point     0.020516; 0.117…
```

```
Time
ROS Time: 1648844385.20   ROS Elapsed: 345.66   Wall Time: 1648844385.24   Wall Elapsed: 345.66        ☐ Experimental
                                                                                                        31 fps
```

LaserScan displayed in Rviz2

Camera

The Camera display shows camera images from sensor_msgs/msg/Image messages. The OAK-D cameras publish images on the /color/preview/image and /stereo/depth topics.



Displays

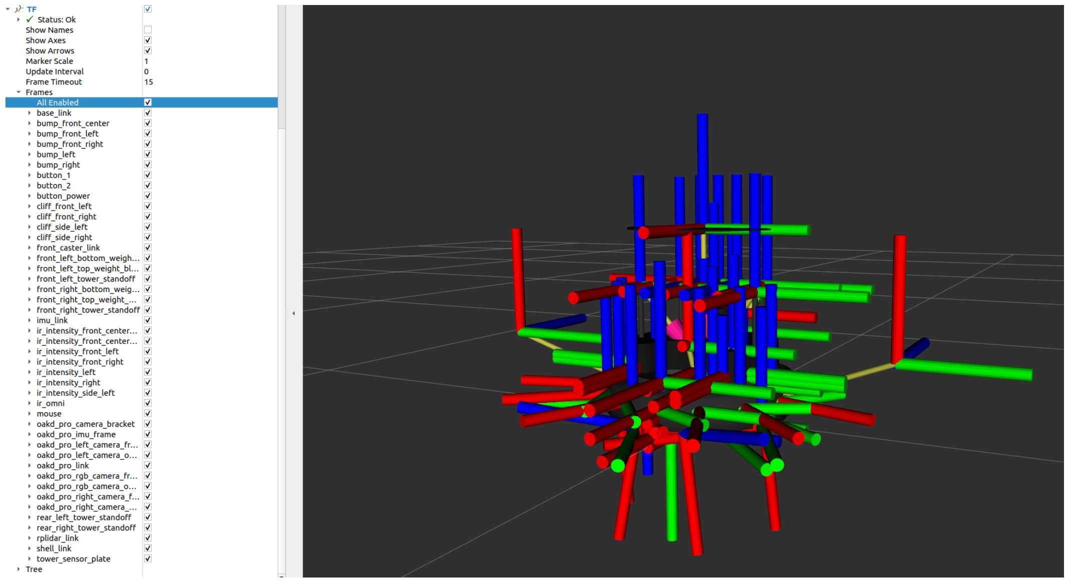
Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
LaserScan	✓
Camera	✓
✓ Status: Ok	
Visibility	✓
Topic	/color/preview/image
Image Rendering	background and overlay
Overlay Alpha	0.5
Zoom Factor	1
Camera	☐

Camera
Displays an image from a camera, with the visualized world rendered behind it. More Information.

Camera image displayed in Rviz2

TF

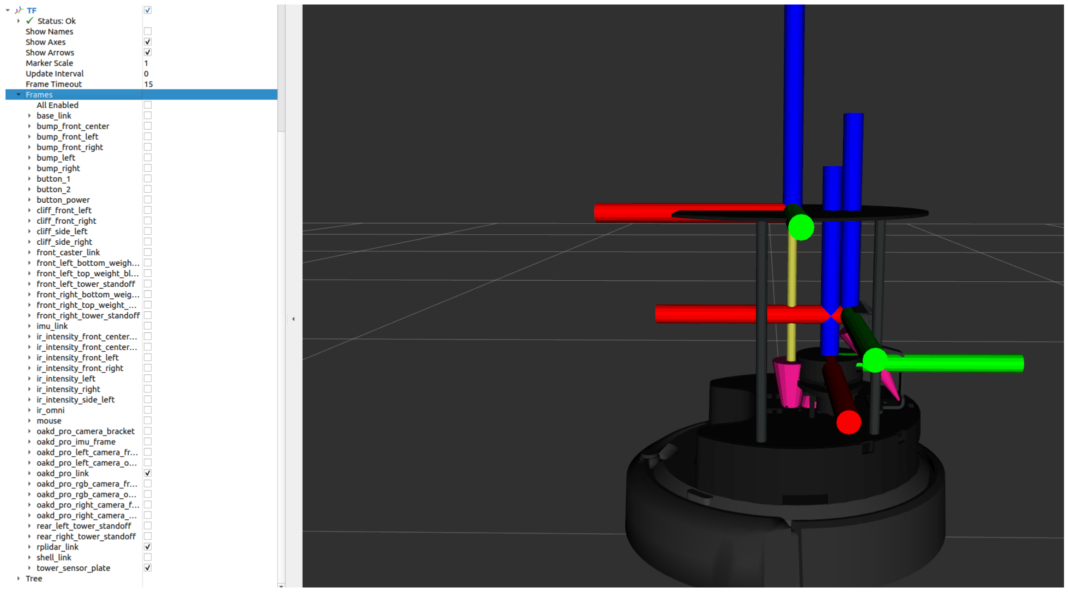
The TF display can be used to visualise the links that make up the robot. When you first add the TF display, it will show every link that makes up the robot.



TF	✓
✓ Status: Ok	
Show Names	☐
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	✓
base_link	✓
bump_front_center	✓
bump_front_left	✓
bump_front_right	✓
bump_left	✓
bump_right	✓
button_1	✓
button_2	✓
button_power	✓
cliff_front_left	✓
cliff_front_right	✓
cliff_side_left	✓
cliff_side_right	✓
front_caster_link	✓
front_left_bottom_weigh…	✓
front_left_top_weight_bl…	✓
front_left_tower_standoff	✓
front_right_bottom_weig…	✓
front_right_top_weight_…	✓
front_right_tower_standoff	✓
imu_link	✓
ir_intensity_front_center…	✓
ir_intensity_front_center…	✓
ir_intensity_front_left	✓
ir_intensity_front_right	✓
ir_intensity_left	✓
ir_intensity_right	✓
ir_intensity_side_left	✓
ir_omni	✓
mouse	✓
oakd_pro_camera_bracket	✓
oakd_pro_imu_frame	✓
oakd_pro_left_camera_fr…	✓
oakd_pro_left_camera_o…	✓
oakd_pro_link	✓
oakd_pro_rgb_camera_fr…	✓
oakd_pro_rgb_camera_o…	✓
oakd_pro_right_camera_f…	✓
oakd_pro_right_camera_…	✓
rear_left_tower_standoff	✓
rear_right_tower_standoff	✓
rplidar_link	✓
shell_link	✓
tower_sensor_plate	✓
Tree	

TF with default settings

You can uncheck the "All Enabled" box, and then select the links you wish to see.

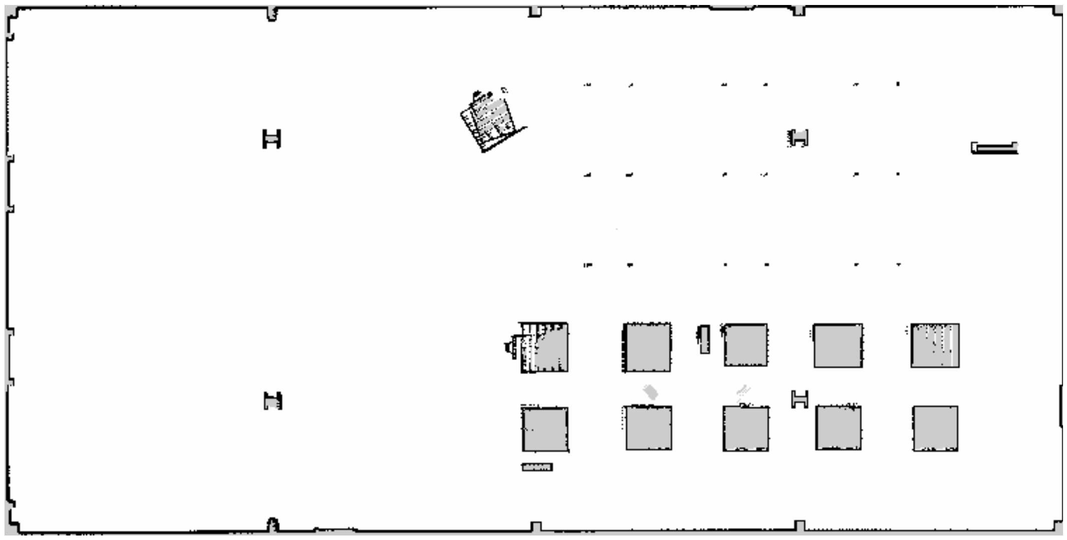


TF	✓
✓ Status: Ok	
Show Names	☐
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	☐
base_link	☐
bump_front_center	☐
bump_front_left	☐
bump_front_right	☐
bump_left	☐
bump_right	☐
button_1	☐
button_2	☐
button_power	☐
cliff_front_left	☐
cliff_front_right	☐
cliff_side_left	☐
cliff_side_right	☐
front_caster_link	☐
front_left_bottom_weigh…	☐
front_left_top_weight_bl…	☐
front_left_tower_standoff	☐
front_right_bottom_weig…	☐
front_right_top_weight_…	☐
front_right_tower_standoff	☐
imu_link	☐
ir_intensity_front_center…	☐
ir_intensity_front_center…	☐
ir_intensity_front_left	☐
ir_intensity_front_right	☐
ir_intensity_left	☐
ir_intensity_right	☐
ir_intensity_side_left	☐
ir_omni	☐
mouse	☐
oakd_pro_camera_bracket	☐
oakd_pro_imu_frame	☐
oakd_pro_left_camera_fr…	☐
oakd_pro_left_camera_o…	☐
oakd_pro_link	✓
oakd_pro_rgb_camera_fr…	☐
oakd_pro_rgb_camera_o…	☐
oakd_pro_right_camera_f…	☐
oakd_pro_right_camera_…	☐
rear_left_tower_standoff	☐
rear_right_tower_standoff	☐
rplidar_link	✓
shell_link	☐
tower_sensor_plate	✓
Tree	

TF with selected links

SLAM

Simultaneous localization and mapping (SLAM) is a method used in robotics for creating a map of the robots surroundings while keeping track of the robots position in that map. The TurtleBot 4 uses slam_toolbox to generate maps by combining odometry data from the Create® 3 with laser scans from the RPLIDAR. slam_toolbox supports both synchronous and asynchronous SLAM nodes.


Map generated by slam_toolbox

Synchronous SLAM

Synchronous SLAM requires that the map is updated everytime new data comes in. This results in maps with high accuracy and detail. The downside to synchronous SLAM is that it requires high processing power from the computer running it to keep up with the sensor data. This approach is ideal for use on a PC, whether it is for the simulator or for getting better SLAM performance on the physical robot.

Launching synchronous SLAM:

```
ros2 launch turtlebot4_navigation slam_sync.launch.py
```

Asynchronous SLAM

Asynchronous SLAM will update the map as fast as the processor running it can handle. This may cause it to drop some laser scans or odometry data. Maps created with asynchronous SLAM may have reduced accuracy and detail, but this method requires significantly less proccessing power. This approach is ideal for use on the TurtleBot 4's Raspberry Pi. The default parameters for asynchronous SLAM use a reduced map resolution to further improve performance on the Pi.

Launching asynchronous SLAM:

```
ros2 launch turtlebot4_navigation slam_async.launch.py
```

Saving the map

Once you have driven the robot around and generated the map, you can use the following call to save the map to your current directory:

```
ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name:
  data: 'map_name' "
```

This will generate two files: map_name.yaml and map_name.pgm. You can open the .pgm file with an image editor to view your map.

Nav2

Nav2 is the official navigation stack in ROS2. Nav2 can be used to calculate and execute a travel path for the robot by using a map of its surroundings. The map can be loaded at launch or generated with SLAM while navigating.

Launching Navigation

Launch files

- **Nav Bringup**: Launches Nav2 nodes, with the option to launch SLAM or Localization as well.

Parameters

- **nav2**: Whether to launch Nav2 nodes.
  - *options*: [true, false]
  - *default*: true
- **slam**: Launch SLAM along with Nav2.
  - *options*: [off, sync, async]
  - *default*: off
- **localization**: Launch localization with an existing map
  - *options*: [true, false]
  - *default*: false
- **map**: Path to existing map.
  - *default*: /path/to/turtlebot4_navigation/maps/depot.yaml
- **params_file**: Full path to parameter file for Nav2 and localization nodes.
  - *default*: /path/to/turtlebot4_navigation/config/nav2.yaml

Configuration

The default TurtleBot 4 configuration can be found here. It is a slightly modified version of the default configuration from the Nav2 github. The configuration file allows the user to modify parameters such as velocity while pathing, the radius of the robot, costmap update frequencies and resolutions, and more. For more information, read the Nav2 configuration guide.

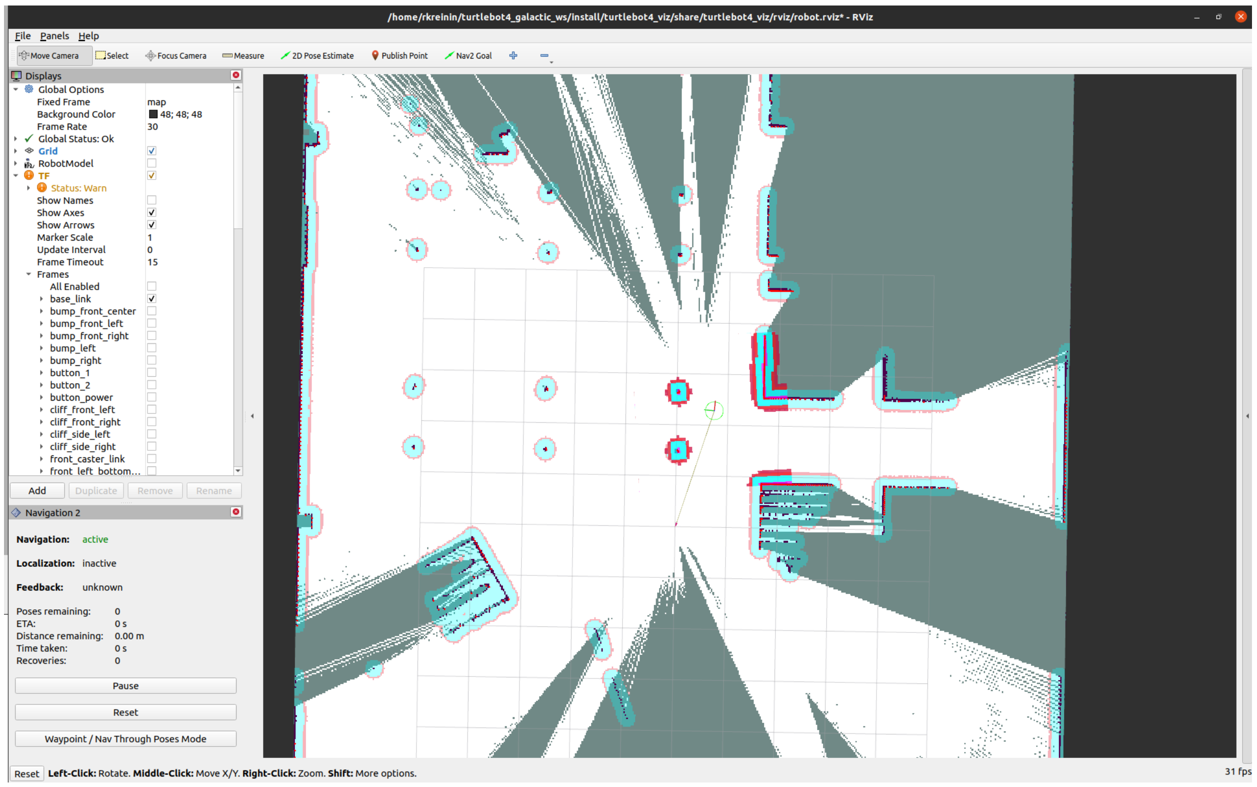
Examples

Launching Nav2 with synchronous SLAM:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py slam:=sync
```

The map and costmaps can be viewed in Rviz2:

```
ros2 launch turtlebot4_viz view_robot.launch.py
```



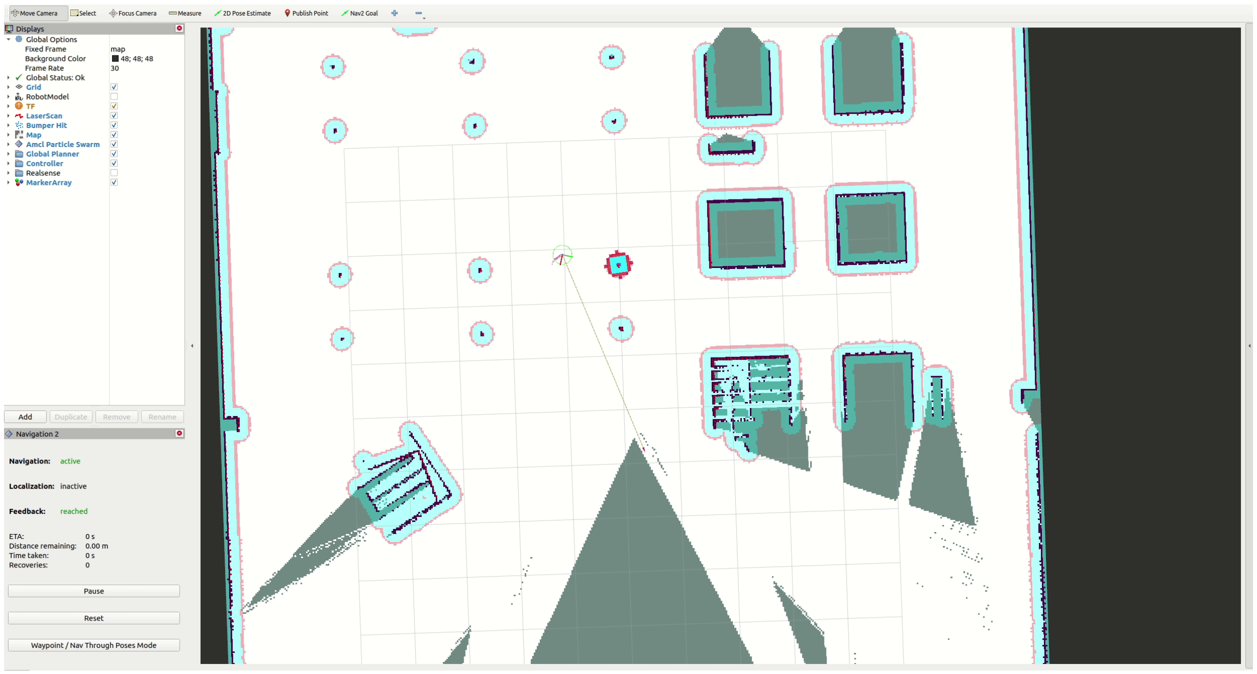
```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
Grid                   ✓
RobotModel             ☐
TF                     ✓
  Status: Warn
  Show Names           ☐
  Show Axes            ✓
  Show Arrows          ✓
  Marker Scale         1
  Update Interval      0
  Frame Timeout        15
  Frames
    All Enabled        ☐
    base_link          ✓
    bump_front_center  ☐
    bump_front_left    ☐
    bump_front_right   ☐
    bump_left          ☐
    bump_right         ☐
    button_1           ☐
    button_2           ☐
    button_power       ☐
    cliff_front_left   ☐
    cliff_front_right  ☐
    cliff_side_left    ☐
    cliff_side_right   ☐
    front_caster_link  ☐
    front_left_bottom… ☐

Navigation:     active
Localization:   inactive
Feedback:       unknown

Poses remaining:      0
ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
```

Nav2 with SLAM

Obstacles that are detected on the map will have a padding around with a radius equivalent to the radius of the robot. When navigating, Nav2 will drive the robot outside of the padded area to avoid hitting obstacles.

Navigating with Rviz2

The easiest way to set a navigation goal is to use **Nav2 Goal** in Rviz2. With Nav2 running, select the Nav2 Goal tool at the top of Rviz2, and click the location on the map where you would like to navigate to.



```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
Grid                   ✓
RobotModel
TF                     ✓
LaserScan              ✓
Bumper Hit             ✓
Map                    ✓
Amcl Particle Swarm    ✓
Global Planner         ✓
Controller             ✓
Realsense
MarkerArray            ✓
```

```
Navigation:    active
Localization:  inactive
Feedback:      reached

ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0
```

Navigating with Rviz2

Simulation

The simulator allows the user to test the robot without the need for a physical robot. It has all of the same functionality as the real robot. The TurtleBot 4 can be simulated using Ignition Gazebo. Unlike Gazebo, Ignition Gazebo does not natively support ROS. Instead, it has its own transport stack with a similar topic and node implementation. To communicate with ROS, we can use the ros_ign_bridge. This ROS node translates data from ROS to Ignition, and vice versa.

Installing Ignition Gazebo

Requirements:

- Ubuntu 20.04
- ROS2 Galactic

Recommended:

- PC with dedicated GPU

Follow the installation instructions described here.

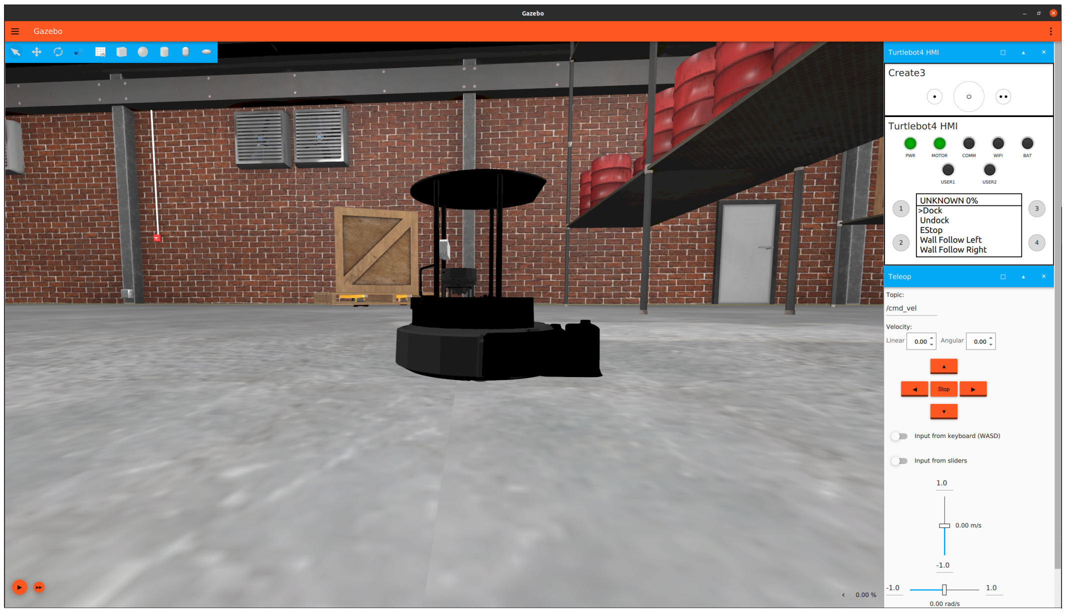
Launching Ignition Gazebo

The ignition.launch.py launch file has several launch configurations that allow the user to customize the simulation.

Default TurtleBot 4 launch:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py
```

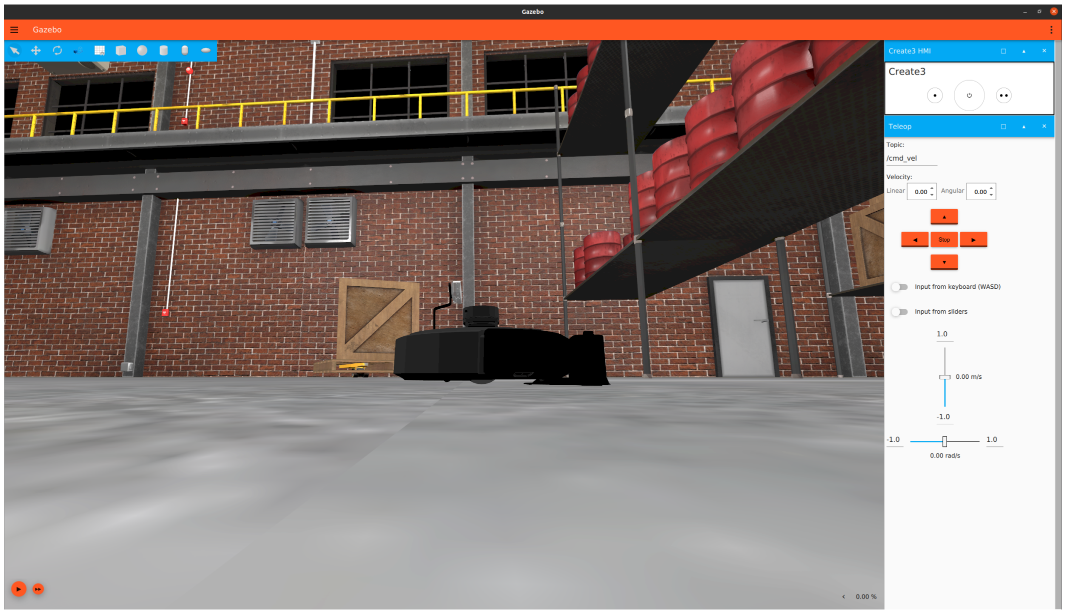
Ignition Gazebo will launch and spawn the TurtleBot 4 in the default world along with all of the necessary nodes.



TurtleBot 4 in Ignition Gazebo

TurtleBot 4 Lite launch:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py model:=lite
```



TurtleBot 4 Lite in Ignition Gazebo

Mechanical

TurtleBot 4

TurtleBot 4 is designed to be modified to meet your needs and make it possible to attach additional sensors and accessories.

Attaching Accessories to the Top Integration Plate

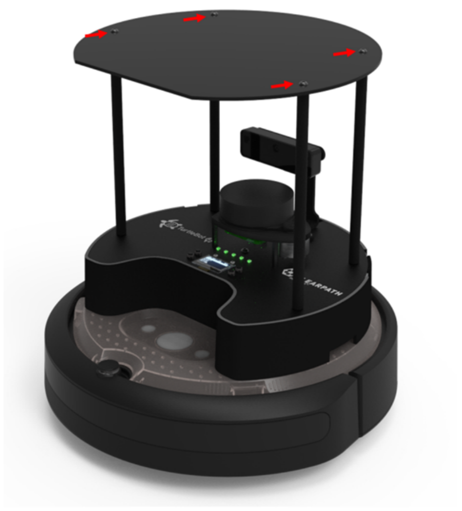
The TurtleBot 4 is equipped with an acrylic plate at the top that is easy to modify in order to attach additional sensors and peripherals.

Warning

Modifications to the plate should only be done when it is removed from the robot, attempting to modify the plate while mounted can cause the plastic to crack.

Removing the Top Integration Plate

The plate is attached to the robot by four Round head M4x0.7 screws. Remove the screws by using a 2.5mm hex key.



TurtleBot 4 Top Integration Plate screw locations

Making Modifications to the Top Integration Plate

Modifying the plate can be done roughly by hand, however a 3D model and 2D drawing of the part is available at Github. When cutting or drilling into the plate, ensure proper safety precautions are taken; wear safety glasses, be familiar with your tools, fasten the plate securely to a work surface. When working with acrylic, it is best to start with a smaller hole (Ø3mm) and enlarge it to the desired size incrementally.

To reattach the plate, apply a low to medium strength thread locker (e.g. Loctite 222). Insert the screws and torque them to 25N-cm.

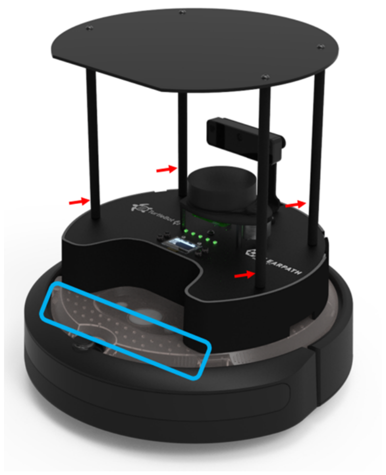
Attaching Accessories to the Base Unit

There is space for sensors to be attached to the base unit above the PCBA as well as near the Create® 3 User Buttons and inside the Shell.

To attach accessories above the PCBA, mounting brackets can be designed for the desired accessory and attached to the TurtleBot 4 by securing them to the standoffs. 3D models of the TurtleBot4 are available on Github that can help in the design process.

To attach accessories inside the Shell, the PCBA and the Create® 3 Integration Plate should be removed. Inside, the existing holes can be used or additional mounting features can be machined or 3D printed.

To attach accessories to the Create® 3, the existing holes in the plate can be used, or additional holes can be drilled by removing the Create® 3 Integration Plate.

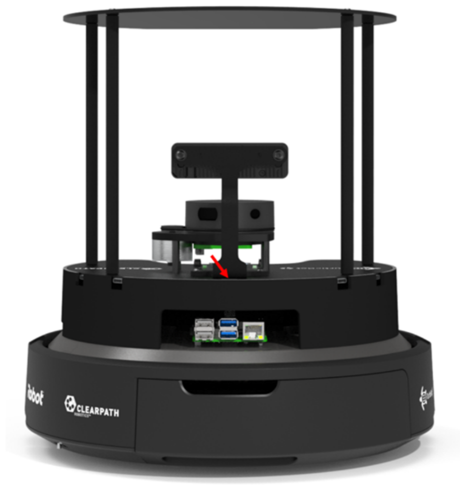


TurtleBot 4 Attachment Locations

Removing the PCBA

To access the inside of the "Shell" of the TurtleBot4, the PCBA can be removed. Ensure that you have a safe spot to place the PCBA when it is removed to prevent damage to the components. It is recommended that this procedure is done on an Electrostatic Discharge Mat to protect the PCBA from damage caused by static electricity. Follow the steps below to remove the PCBA.

1. Remove the Top Integration Plate and the four standoffs.
2. Disconnect the USB cables connected to the Oak-D Camera and the RPLIDAR. Feed these cables through the opening at the back of the Robot.



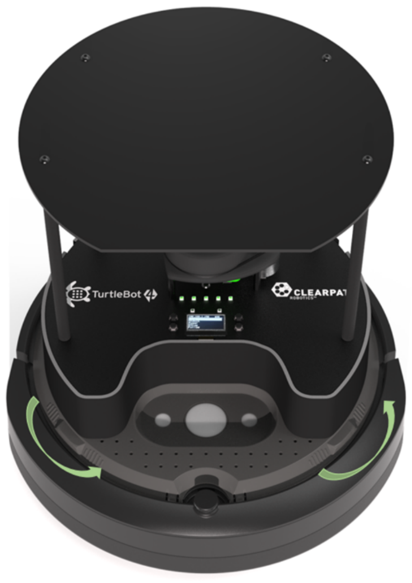
TurtleBot 4 Cable passthrough

3. Carefully lift the PCBA by holding the camera bracket or the RPLIDAR base.
4. Disconnect the USB-B cable, the power harness, the 40 pin ribbon cable, and the fan cable, as well as any other cables that may have been attached.

The PCBA should now be free to be removed and placed safely.

Removing the Create® 3 Integration Plate and Shell

The Create® 3 Integration Plate and Shell can be removed with the rest of the assembly on or off. To remove the Create® 3 Integration Plate, first open and remove the rear Create® 3 tray. Then disconnect the USB C cable and the power harness from the Create® 3 base. Feed these cables through the slot at the back of the Create® 3. Using the tabs on the Create® 3, twist the plate counter-clockwise until it snaps to unlock it and remove the plate.



Create® 3 Integration Plate Removal

To reattach the plate, place the plate slightly angled such that the posts fit into the tabs. Then, twist the plate clockwise until it snaps back into place.

TurtleBot 4 Lite

TurtleBot Lite is built on the iRobot Create® 3 learning platform which features an easy to modify integration plate.

Attaching Accessories to the Base Unit

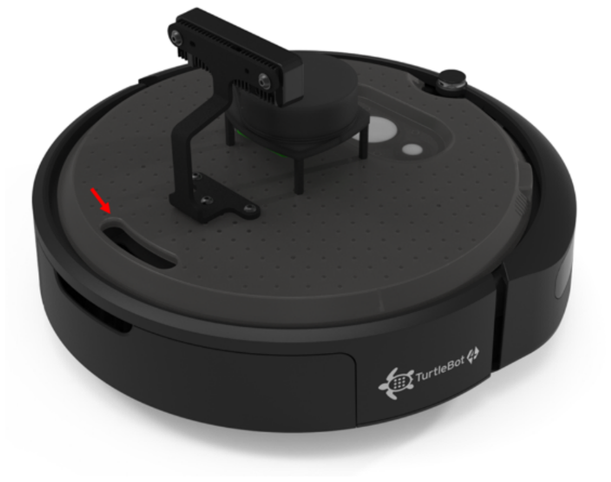
There is space for sensors and accessories to be attached to the base unit around the Oak Camera and RPILIDAR. To attach accessories to the Create® 3, the existing holes in the plate can be used. These Ø3.5mm holes are spaced apart 10mm. Alternatively, holes can be drilled by removing the Create® 3 Integration Plate. 3D models of the robot are available on Github which can help in the design.

TurtleBot 4 Lite Integration

Removing the Create® 3 Integration Plate

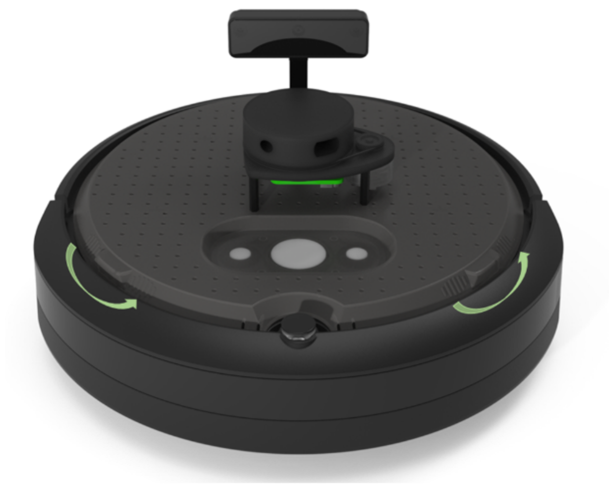
The Create® 3 Integration Plate can be removed. To remove the Create® 3 Integration Plate follow the steps below.

1. Disconnect the USB cables connected to the Oak-D Camera and the RPLIDAR. Feed these cables through the slot opening at the back of the Robot.



TurtleBot 4 Lite Cable Passthrough

1. Using the tabs on the Create® 3, twist the plate counter-clockwise until it snaps to unlock it and remove the plate.

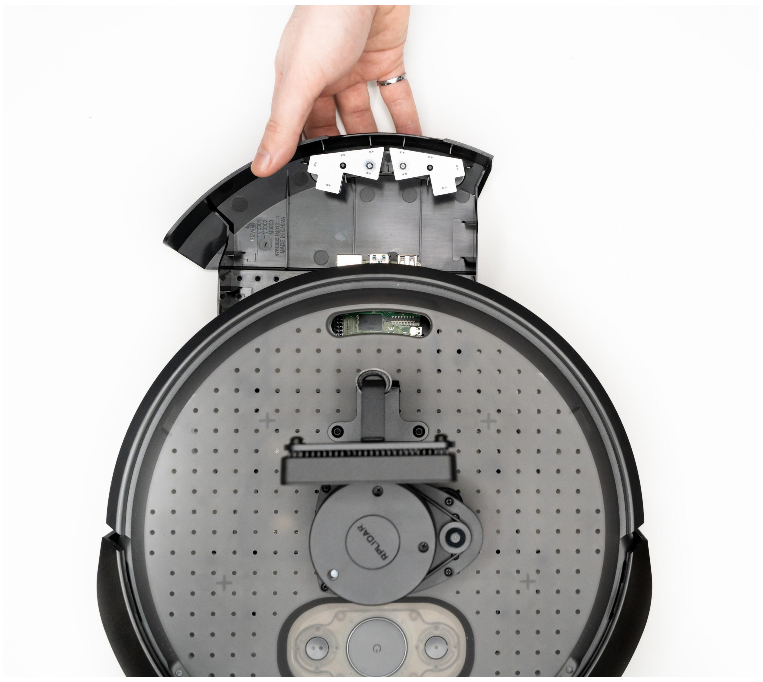


Create® 3 Integration Plate Removal

To reattach the plate, place the plate slightly angled such that the posts fit into the tabs. Then, twist the plate clockwise until it snaps back into place. Open the rear tray and feed the USB cables that were previously disconnected through the slot. Attach the USB-C cable to the Oak-Camera and connect the USB Micro cable to the RPLIDAR.

Accessing the Raspberry Pi Computer

The Raspberry Pi is found in the rear tray of the robot. To fully access the Raspberry Pi, disconnect the USB cables connected to the Oak Camera and RPLIDAR and feed them through the slot opening at the rear of the robot. You can now carefully slide out the cargo bay.



Create® 3 Cargo Bay removal

Payloads Over 9kg

The TurtleBot4 is able to perform with heavier payloads over 9 kg, however some mechanical and software changes must be made for ideal operation. If these changes are not used the system may become unstable and difficult to control.

Mechanical Modification

In order to ensure the robot is as stable as possible during operation, it is important to mount the payload such that its center of gravity (COG) is fully supported by the wheelbase. This can be achieved through two methods:

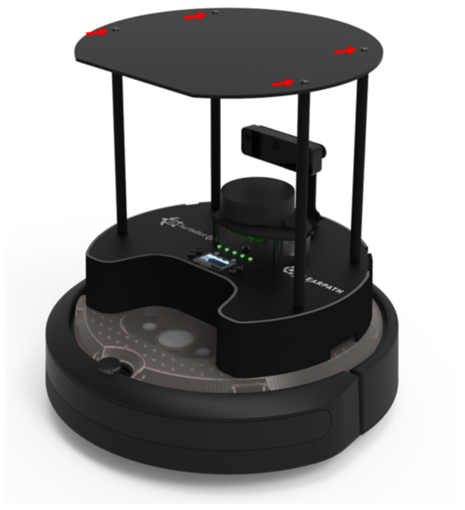
1. Design the payload mounting to accommodate for the offset COG. This can be done by using the existing acrylic integration plate or by using a custom plate.
2. Add an additional caster wheel to the rear of the Create 3. This will ensure that the payload is fully supported and balanced. Detailed instructions as well as design files are available here.

Additionally, the payload height can be lowered to ensure better stability for the robot, particularly if an additional caster wheel is not used and the system is balanced on three wheels. While the acceleration and speed can be modified to accommodate for the elevated COG (see below), lowering it will optimize robot performance.

Lower payload height

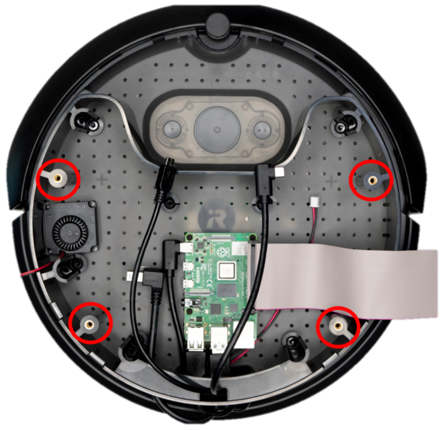
Lowering the integration plate of the Standard TurtleBot4, and therefore the payload, is the simplest way to make the COG as low as possible. Follow the steps below:

1. Remove the four M4 screws using a 2.5mm hex key. The integration plate should now be free; set it aside.


TurtleBot 4 Top Integration Plate screw locations

1. Remove the four round standoffs by hand and set them aside. Note that the PCBA is now free to move and care should be taken to ensure it is not damaged.

2. Use alternative standoffs (e.g. from Mcmaster Carr) or other mounting options. The threaded inserts of the robot are M4x0.7mm. The mounting pattern can be taken from 3D CAD available on Github or it can be transferred from the integration plate itself.


TurtleBot 4 Mounting Inserts

When reassembling the robot with new hardware, ensure that all fasteners are torqued according to the fastener and mating threads and that a threadlocker is used.

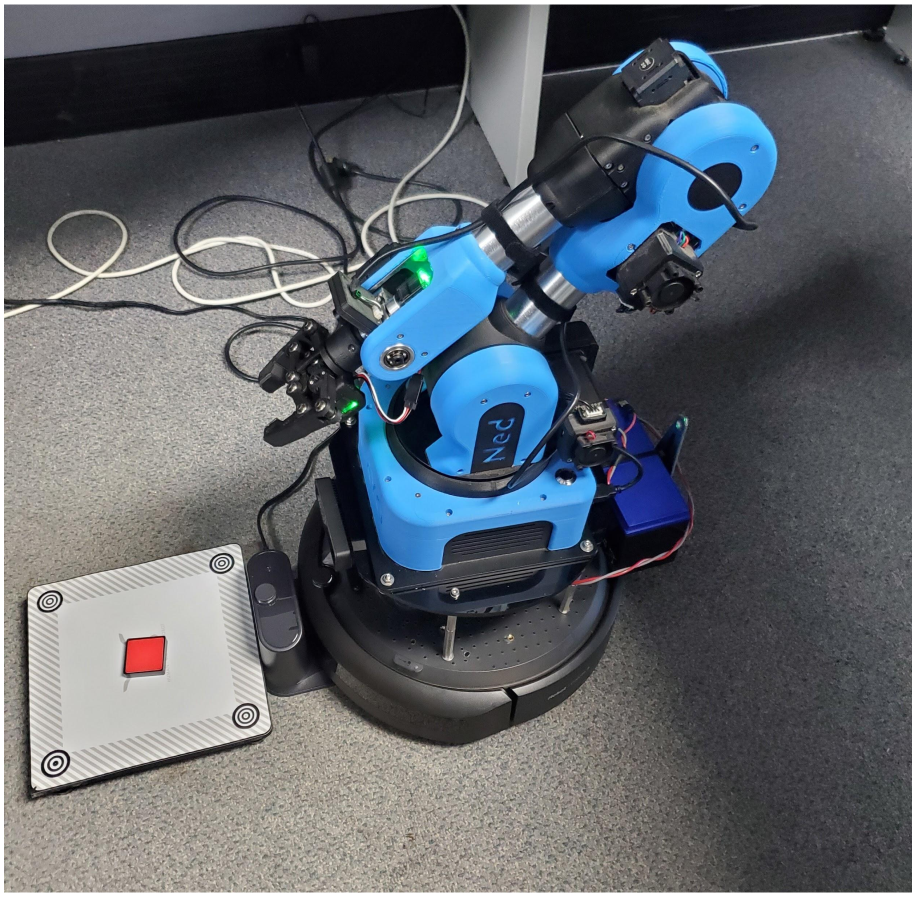
For the TurtleBot4 Lite, the Create 3 plate can be used to secure payloads. Tips for mounting directly to this plate can be found on the Create 3 Github.

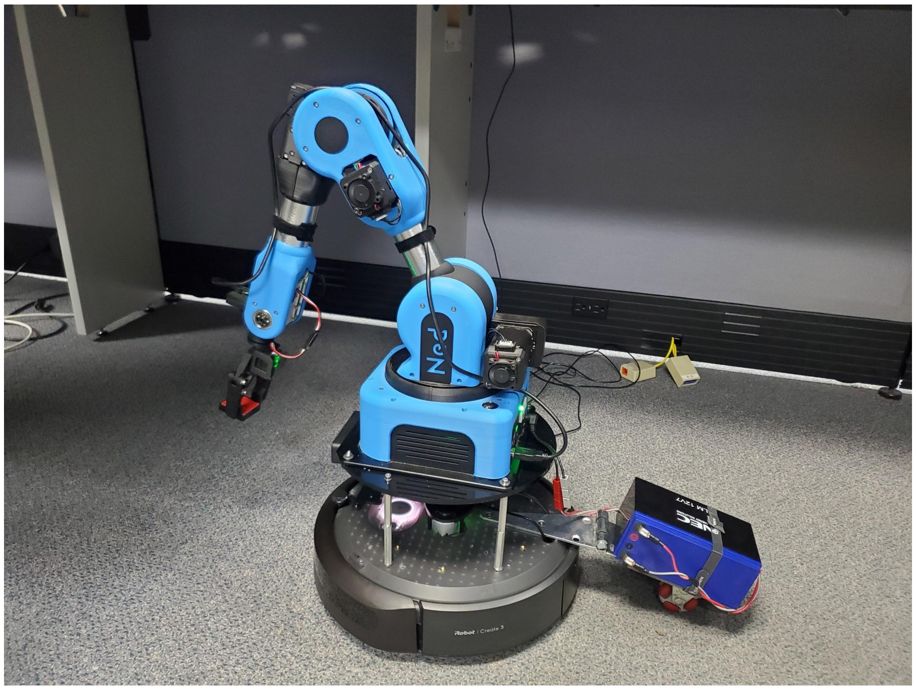
Note

The minimum clearance to accommodate the OAK-D-Pro is 108mm. For the OAK-D-Lite it is 102mm.

Example

The pictures below show a Clearpath Robotics Hackday project where a NED2 Robot manipulator (8.9kg) mounted on top of a TurtleBot4 Lite using 4X2 M4 standoffs (Mcmaster Carr P/N 98952A450) and an integration plate from a TurtleBot4 Standard.The manipulator was mounted such that the COG was further forward and supported by the front caster wheel. An external NEC ALM 12V7 Battery (0.9kg) was also attached to power the device. In this particular case the battery was installed on a hinge that would allow the battery to sit closer to the robot or to be laid out and supported on its own omni-wheel. This was a design choice to allow the demonstration of either a larger weight mounted to the unit or a payload hitched to the robot as an alternative payload mounting design method.





TurtleBot 4 Lite with a NED2 arm

Software Modifications

Velocity and acceleration limits should be set appropriately to maintain stability when driving the robot. Otherwise you may find that the robot will shake, stall, or not drive as commanded.

Acceleration Limits

The acceleration limit on the Create® 3 can be changed using the wheel_accel_limit parameter of the motion_control node.

```
ros2 param set /motion_control wheel_accel_limit 300
```

The acceleration value can be between 1 and 900.

Note

The wheel_accel_limit is applied to both acceleration and deceleration. If you set it too low, the robot will stop very slowly.

Velocity Limits

The Create® 3 has linear velocity limits of 0.31 m/s with safety enabled, and 0.46 m/s with safety overridden. The angular velocity is limited to 1.9 rad/s. For heavy payloads you may want to limit this further. The method to do this will vary based on how you are driving the robot.

Keyboard Teleoperation

If you are using the teleop_twist_keyboard ROS2 node, you can follow the CLI instructions to reduce linear and angular velocities.

Joystick Teleoperation

To limit the teleop_twist_joy velocities you will need create a modified version the TurtleBot4 controller config file. The scale_linear.x value limits the linear velocity, and the scale_angular.yaw value limits the angular velocity.

Once you have created the config file, you can launch joy_teleop and set the controller_config parameter to the full path of your new config file.

```
ros2 launch turtlebot4_bringup joy_teleop.launch.py controller_config:=/path/to/config.yaml
```

Command Velocity

If you are manually sending the velocity through the /cmd_vel topic, simply reduce the velocity values to an appropriate level.

Create® 3 Actions

If you are driving the robot through one of the Create® 3 actions, you can set velocity limits in the action goal.

Nav2

To limit velocity during navigation, you can create a modified nav2.yaml configuration file. Changing parameters such as controller_server.FollowPath.max_vel_x will limit the velocity commands sent by the Nav2 stack.

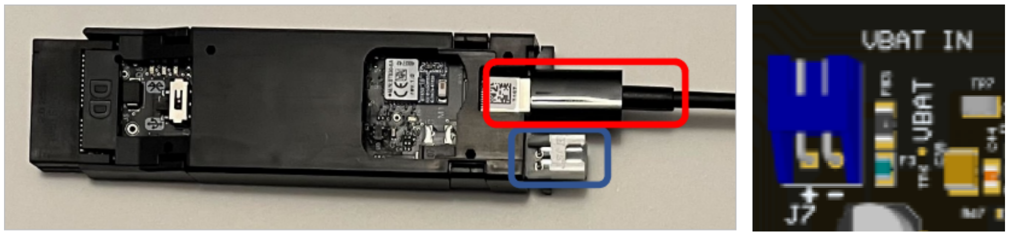
Launch Nav2 with your modified parameter file:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py params_file:=/path/to/your/nav2.yaml
```

Electrical

Create® 3

On the TurtleBot 4, the connection of the User Interface board with the Create® 3 robot is only through the VBAT connector J7, which uses JST XH-style connector. The connector has been marked with Positive and Negative signs on the board (Positive being pin 1). The VBAT line is fused with a PTC fuse rated at 2A.



Create® 3 Power Adapter (left) and J7 connector (right)

The Create® 3 power adapter also supplies the Raspberry Pi 4 with power and communication through a USB 2.0 (Type C) on both the TurtleBot 4 and TurtleBot 4 Lite. The USB interface can supply up to 3A at 5V.

For more details, visit the Create® 3 Documentation.

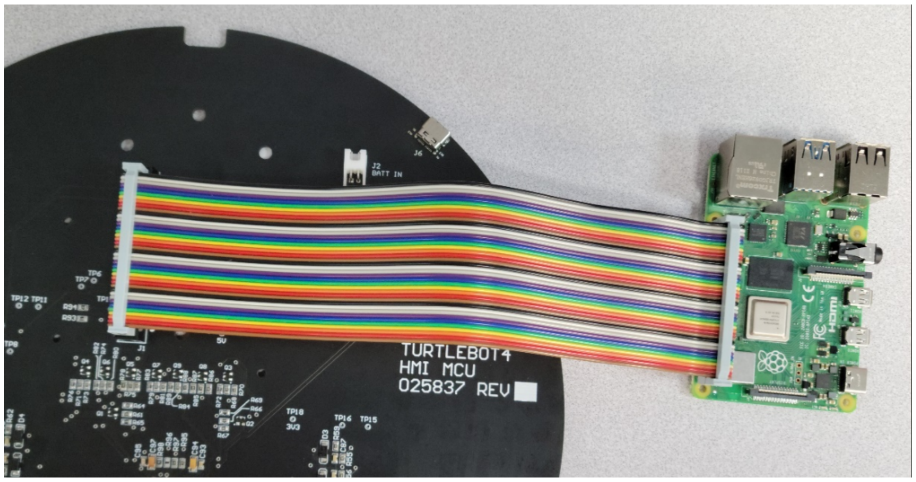
Warning

It is recommended to not drain the robot below 20% as VBAT voltage begin to decline sharply at this level.

Raspberry Pi 4B

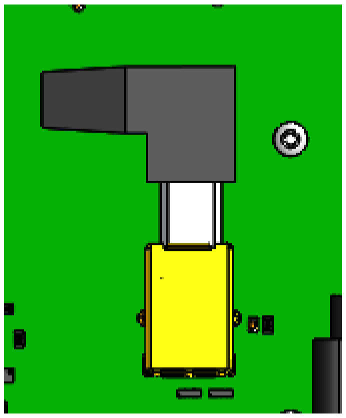
The Raspberry Pi 4 is present on both the TurtleBot 4 and TurtleBot 4 Lite. On the TurtleBot 4 it can be found inside the shell, while on the Lite it is mounted in the Create® 3 cargo bay.

The TurtleBot 4 connects the Raspberry Pi with the User Interface Board through a 40 pin connector and a USB 3.0 (Type B) cable. The USB 3.0 type cable enables communication to 4 USB-C ports on the UI board, while the ribbon cable passes the 40 GPIO pins of the Raspberry Pi through to the UI Board.



TurtleBot 4 UI Board to Raspberry Pi connector

The TurtleBot 4 comes with a USB-A to USB-B 3.0 cable to connect the UI board and Raspberry Pi. Without this connection the USB-C ports will only be able to supply power, but not communication.



TurtleBot 4 UI Board USB type B connector

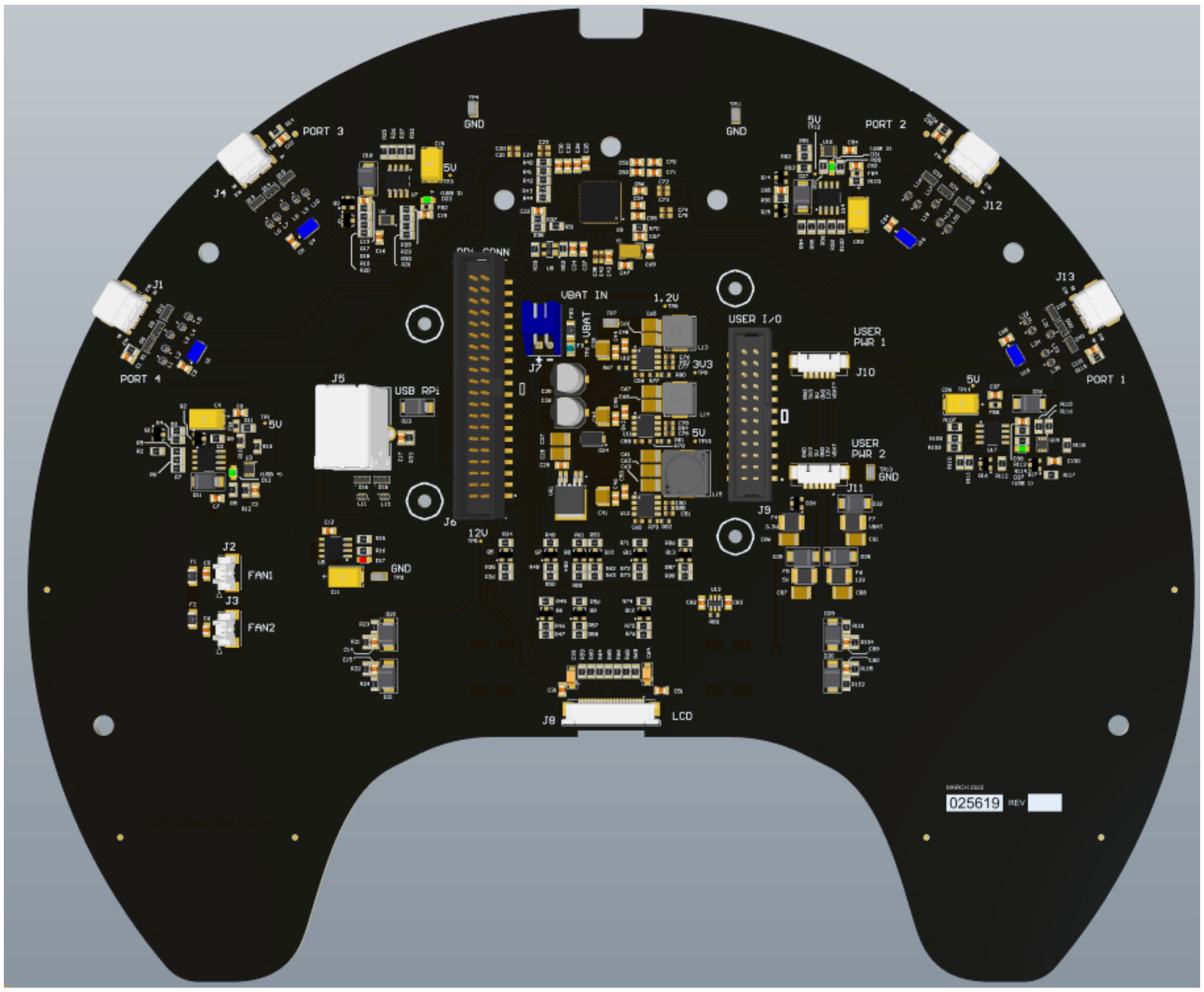
User Interface PCBA

Note

The User Interface PCBA is only available on the TurtleBot 4 and NOT the TurtleBot 4 Lite.

Overview

The TurtleBot 4 comes with an additional User Interface board that expands on the Raspberry Pi 4 functionality to give the user ease of control over the Create 3 robot and Raspberry Pi and to act as an expansion board for addons, sensors, gadgets the user might have in mind to utilize.



TurtleBot 4 User Interface PCBA

User I/O

The TurtleBot 4 has a 2x20 pin internal connector connecting it to the Raspberry Pi via a flex cable, and another 2x12 pin connector allowing the user to access the remaining GPIOs and a set of 5V, and 3.3V power pins coming from the Raspberry Pi.

The IO interface between the 2x20 connector and 2x12 connector and the available GPIOs to the user are shown in Table 1, and 2. The GPIO numbers are a direct match to the Raspberry Pi 4 GPIO.

Table 1: 2x20 RPi Connector Pinout

GPIO #	Function	Pin #	Pin #	Function	GPIO #
	3V3_RPi	1	2	5V_RPi	
GPIO2	USER_PORT	3	4	5V_RPi	
GPIO3	USER_PORT	5	6	GND	
GPIO4	SDA	7	8	USER_PORT	GPIO14
	GND	9	10	USER_PORT	GPIO15
GPIO17	PWR_LED	11	12	MTR_LED	GPIO18
GPIO27	COMM_LED	13	14	GND	
GPIO22	BATT_GRN_LED	15	16	BATT_RED_LED	GPIO23
	3V3_RPi	17	18	WIFI_LED	GPIO24
GPIO10	USER_PORT	19	20	GND	
GPIO9	USER_PORT	21	22	USER1_GRN_LED	GPIO25
GPIO11	USER_PORT	23	24	USER_PORT	GPIO8
	GND	25	26	USER_PORT	GPIO7
GPIO0	EEPROM_SD	27	28	EEPROM_SC	GPIO1
GPIO5	SCL	29	30	GND	

GPIO6	USER2_GRN_LED	31	32	USER_PORT	GPIO12
GPIO13	DISPLAY-RST	33	34	GND	
GPIO19	USER_SW1	35	36	USER_SW2	GPIO16
GPIO26	USER_SW3	37	38	USER_SW4	GPIO20
	GND	39	40	USER2_RED_LED	GPIO21

Note

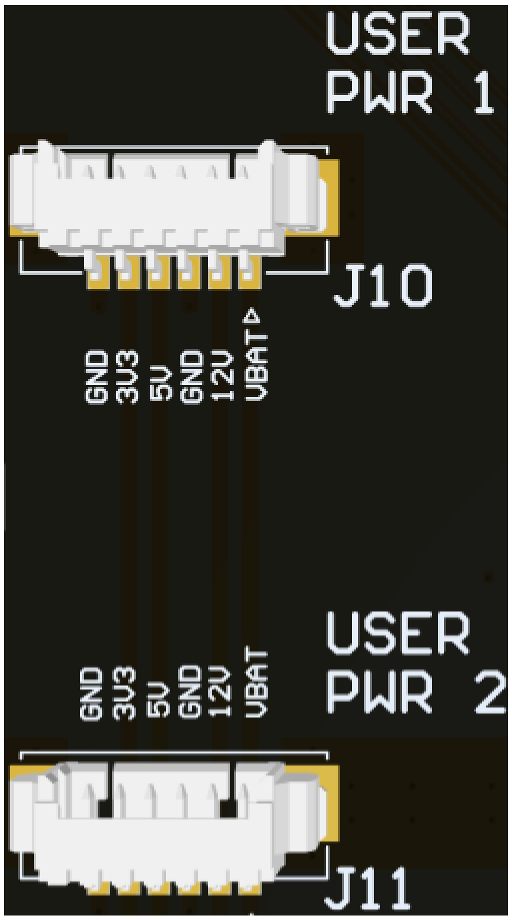
ALL USER_PORTs are routed to the 2X12 Auxiliary connectors

Table 2: 2x12 User I/O Pinout

GPIO #	**Function**	**Pin #**	**Pin #**	**Function**	**GPIO #**
	3V3_RPi	1	2	5V_RPi	
GPIO2	USER_PORT	3	4	5V_RPi	
GPIO3	USER_PORT	5	6	GND	
	GND	7	8	USER_PORT	GPIO14
	3V3_RPi	9	10	USER_PORT	GPIO15
GPIO0	EEPROM_SD	11	12	EEPROM_SC	GPIO1
GPIO10	USER_PORT	13	14	GND	
GPIO9	USER_PORT	15	16	GND	
GPIO11	USER_PORT	17	18	USER_PORT	GPIO8
	GND	19	20	USER_PORT	GPIO7
	GND	21	22	USER_PORT	GPIO12
	GND	23	24	GND	

User Power

In addition to these GPIO ports, the user has two additional power ports available supplying 3.3V, 5V, 12V, VBAT (14.4V), and two grounds each.



TurtleBot 4 Additional Power Ports

The pinout and power ratings can be found in Table 3.

Table 3: User Power Port Pinout

Pinout	Source	Max current output (mA)	Fuse Hold at (mA)
1	VBAT	300	350
2	12V	300	350
3	GND		
4	5V	500	500
5	3V3	250	300
6	GND		

Molex Picoblade 6-Pin cable assembly

The two connectors are both 6-Pin Molex PicoBlade P/N 0532610671. The cable assembly needed to use these connectors are P/N 0151340602.

Molex PicoBlade: Connector 0532610671 (left) and Cable 0151340602 (right)

User USB-C Ports

The are 4 USB-C ports that go through an integrated hub on the User Interface board and connect to the Raspberry Pi through a single USB 3.0 cable. The current available to all 4 ports is 3A. Additionally, each individual port is current limited to 3A. In other words, each port is capable of supplying 3A if the others aren't in use, or the available 3A is shared amongst ports that are in use. The bandwidth for communication is split among 4 dynamically depending on how many of the ports are communication at once, and is limited by the USB 3.0 connection to the Raspberry Pi.

Note

On the REV 2 board only port 4 can supply 3A. The other three ports can supply 2.6A +/- 0.1A. The revision and port numbers are labeled on the underside of the PCBA.

Power Budget

The total power made available by the Create 3 output power adapter is 28.8W. This supplies the USB-C connector mated to Raspberry Pi, and the two pin auxiliary VBAT connector combined. Since the two connectors share this power amongst them, a rise in consumption of one will lead to reduction of available power for the other. Thus, although maximum theoretical power consumption of individual components is mentioned in Table 2, the true limiting factor is the remaining power available to the whole system.

Table 1: Nominal power consumption

Source	**TurtleBot 4 Lite (W)**	**TurtleBot 4 (W)**
Raspberry Pi 4B	4	4
OAK-D-Pro	-	5
OAK-D-Lite	3.5	-
RPLIDAR A1M8	2.3	2.3
Fan	0.8	1.3
UI Board	-	2.4
User Power	-	*Limited By Remaining Power
USB-C Ports	-	*Limited By Remaining Power
Total Power Draw	**10.6**	**15**
Total Available Power	**28.8**	**28.8**
***Remaining Power**	**18.2**	**13.8**

Table 2: Maximum power consumption of individual components and systems

Source	**Operating Voltage (V)**	**Max current draw (A)**	**Max Power (W)**
User Interface Board			~40
4 USB-C ports	5	3	15
USB Hub Controller	1.2, 3.3	1.3	1.8
OLED Display	12	0.031	0.372
User LEDs	5	0.007	0.17
USER PWR Ports	VBAT	0.3	4.32
	12	0.3	3.6
	5	0.5	2.5
	3.3	0.25	0.825
OAK-D-Lite	5	1	5
OAK-D-Pro (connected to RPi)	5	1	5
OAK-D-Pro (connected to UI Board)	5	1.5	7.5
RPLIDAR A1M8	5	0.6	3
Blower Fan	5	0.25	1.25
Axial Fan	5	0.15	0.75
Raspberry Pi 4B	5	1.2	6

Note

Not accounting for inefficiencies of components, and power loss. Assuming USB hub speed operating with all ports at SuperSpeed, and OLED is set to max brightness.

Tutorials

Driving your TurtleBot 4

There are several methods to get your TurtleBot 4 moving.

Note

The robot must first be set up and connected to Wi-Fi before it can be driven. Check out the Quick Start section if you have not already.

Keyboard Teleoperation

The simplest way to get your robot driving is to use a keyboard application on your PC.

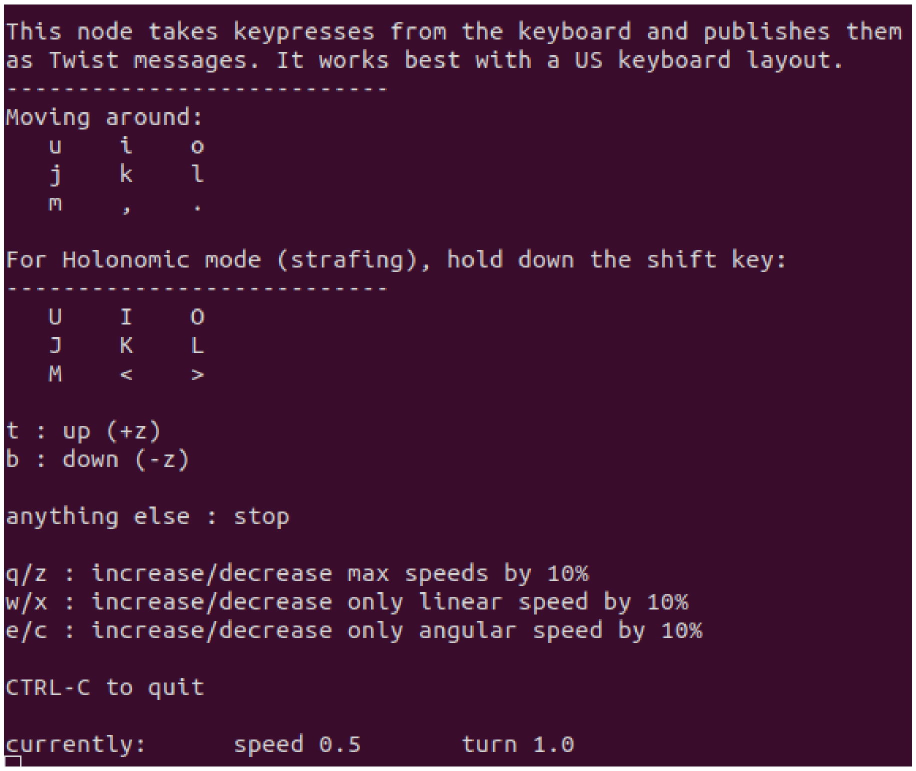
You can install the teleop_twist_keyboard package on your PC by running the following commands:

```
sudo apt update

sudo apt install ros-galactic-teleop-twist-keyboard
```

Once installed, run the node by calling:

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

ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

This will start a CLI interface which allows you to press keys to command the robot to drive.



```
This node takes keypresses from the keyboard and publishes them
as Twist messages. It works best with a US keyboard layout.
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

teleop_twist_keyboard CLI

Press **i** to drive forward, **j** to rotate left, and so on. You can also adjust linear and angular speeds on the go.

Joystick Teleoperation

If you have a TurtleBot 4 controller or have your own Bluetooth controller, you can drive the robot with it.

First, make sure that your controller is paired and connects to the robot. If you have a TurtleBot 4 controller, press the home button and check that the controller's light turns blue. If your controller is not paired or connecting, refer to the Controller Setup section.

Note

If you are using a TurtleBot 4 Lite, the Bluetooth packages will not be installed by default. To install them, SSH into the Raspberry Pi and call `sudo bluetooth.sh` and then reboot the Pi. Then follow the Controller Setup instructions.

Once your controller is connected, make sure that the `joy_teleop` nodes are running. These are launched as part of the Standard and Lite launch files under `turtlebot4_bringup`. If it is not running, you can run it manually by calling:

```
ros2 launch turtlebot4_bringup joy_teleop.launch.py
```

Note

The default configuration for the joy_teleop nodes will only work for the TurtleBot 4 controller and PS4 controllers. You may need to create your own config file if the button mappings on your controller differ.

To drive the robot, press and hold either **L1** or **R1**, and move the left joystick. By default, **L1** will drive the robot at ‘normal' speeds, and **R1** will drive the robot at ‘turbo' speeds. The buttons can be changed in the configuration file.

Command Velocity

Both the keyboard and joystick teleop methods work by sending velocity commands the the robot through the /cmd_vel topic. This topic uses a geometry_msgs/Twist message to tell the robot what linear and angular velocities should be applied.

You can manually publish to this topic through the command line by calling:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist \
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

Set the linear.x value to drive the robot forwards or backwards, and the angular.z value to rotate left or right.

Create® 3 Actions

The Create® 3 provides a set of ROS2 Actions for driving the robot. You can use the DriveDistance, DriveArc, and RotateAngle actions to tell the robot exactly how far and how fast to drive or rotate.

For example, command the robot to drive 0.5 m forwards at 0.3 m/s:

```
ros2 action send_goal /drive_distance irobot_create_msgs/action/DriveDistance \
"distance: 0.5
max_translation_speed: 0.3"
```

Creating your first node (C++)

This tutorial will go through the steps of creating a ROS2 package and writing a ROS2 node in C++. For a Python example, click here. These steps are similar to the ROS2 Tutorial, but focus on interacting with the TurtleBot 4. For source code, click here.

Note

You can follow this tutorial on either the Raspberry Pi of your TurtleBot 4, or your PC.

Create a workspace

If you do not already have a workspace, open a terminal and create one in the directory of your choice:

```
mkdir ~/turtlebot4_ws/src -p
```

Create a package and node

You will need to create a ROS2 package to hold your files. For this tutorial, we will create a package called turtlebot4_cpp_tutorials with a node called turtlebot4_first_cpp_node.

```
source /opt/ros/galactic/setup.bash
cd ~/turtlebot4_ws/src
ros2 pkg create --build-type ament_cmake --node-name turtlebot4_first_cpp_node
turtlebot4_cpp_tutorials
```

This will create a turtlebot4_cpp_tutorials folder and populate it with a basic "Hello World" node, as well as the CMakeLists.txt and package.xml files required for a ROS2 C++ package.

Write your node

The next step is to start coding. For this tutorial, our goal will be to use the Create® 3 interface button 1 to change the colour of the Create® 3 lightring. Open up the "Hello World" .cpp file located at ~/turtlebot4_ws/src/turtlebot4_cpp_tutorials/src/turtlebot4_first_cpp_node.cpp in your favourite text editor.

Add your dependencies

For this tutorial, we will need to use the rclcpp and irobot_create_msgs packages. The rclcpp package allows us to create ROS2 nodes and gives us full access to all the base ROS2 functionality in C++. The irobot_create_msgs package gives us access to the custom messages used by the Create® 3 for reading the button presses and controlling the lightring.

In your CMakeLists.txt file, add these lines under find_package(ament_cmake REQUIRED):

```
find_package(rclcpp REQUIRED)
find_package(irobot_create_msgs REQUIRED)
```

and add this line under add_executable(turtlebot4_first_cpp_node src/turtlebot4_first_cpp_node.cpp):

```
ament_target_dependencies(turtlebot4_first_cpp_node rclcpp irobot_create_msgs)
```

In package.xml, add these lines under <buildtool_depend>ament_cmake</buildtool_depend>:

```
<depend>rclcpp</depend>
<depend>irobot_create_msgs</depend>
```

Finally, in your nodes .cpp file you will need to include these headers:

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"

#include "irobot_create_msgs/msg/interface_buttons.hpp"
#include "irobot_create_msgs/msg/lightring_leds.hpp"
```

Create a class

Now that the dependencies are set, we can create a class that inherits from the rclcpp::Node class. We will call this class TurtleBot4FirstNode.

```
class TurtleBot4FirstNode : public rclcpp::Node
{
public:
  TurtleBot4FirstNode()
  : Node("turtlebot4_first_cpp_node")
  {}
};
```

Notice that our class calls the Node constructor and passes it the name of our node, turtlebot4_first_cpp_node.

We can now create our node in the main function and spin it. Since our node is empty, the node will be created but it won't do anything.

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<TurtleBot4FirstNode>());
  rclcpp::shutdown();
  return 0;
}
```

Subscribe to the Create® 3 interface buttons

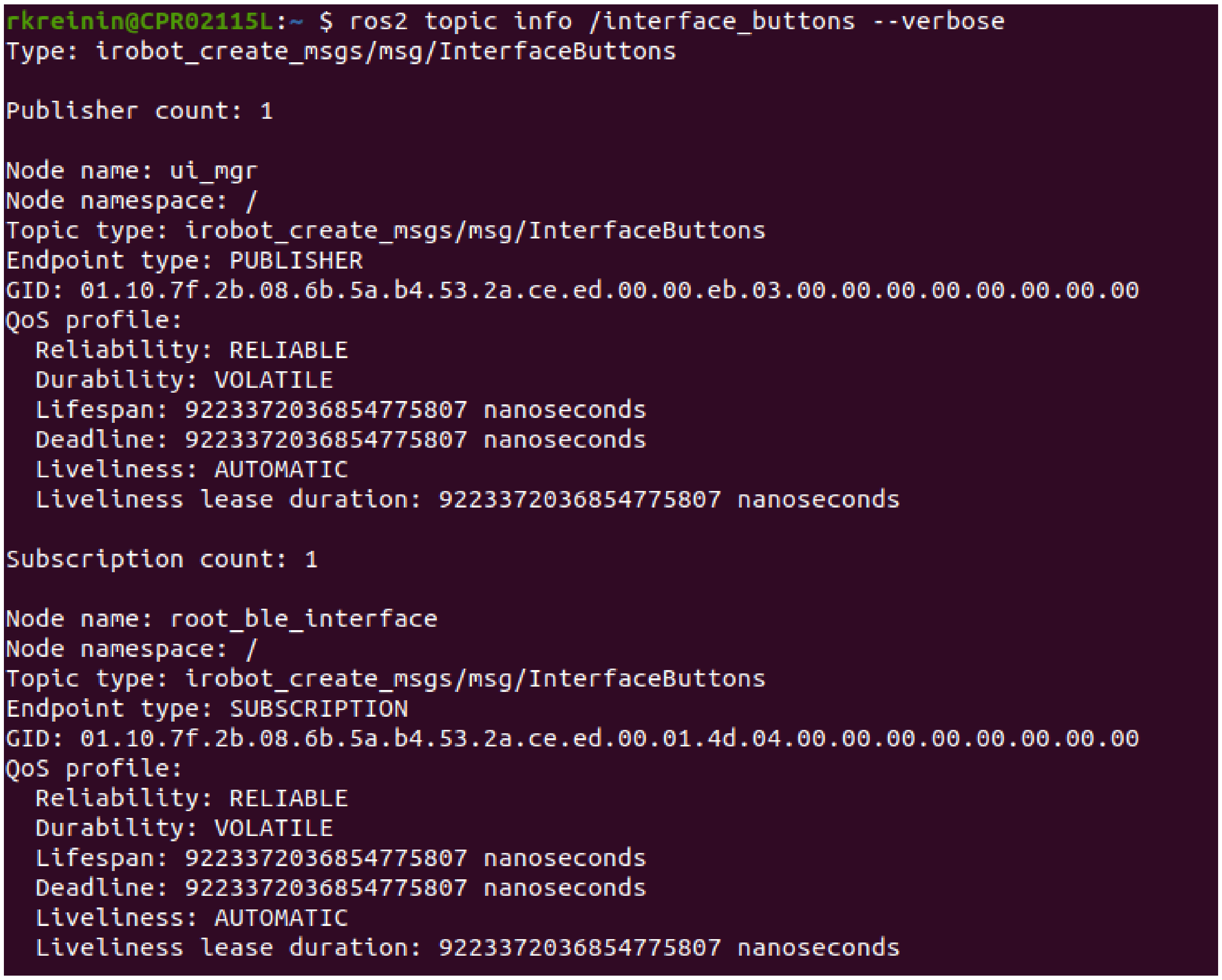
Our next step is to subscribe to the Create® 3 interface buttons topic to receive button presses.

We will need to create a rclcpp::Subscription as well as a callback function for the subscription. The callback function will be called every time we receive a message on the interface buttons topic.

```
class TurtleBot4FirstNode : public rclcpp::Node
{
public:
  TurtleBot4FirstNode()
  : Node("turtlebot4_first_cpp_node")
  {
    // Subscribe to the /interface_buttons topic
    interface_buttons_subscriber_ =
      this->create_subscription<irobot_create_msgs::msg::InterfaceButtons>(
      "/interface_buttons",
      rclcpp::SensorDataQoS(),
      std::bind(&TurtleBot4FirstNode::interface_buttons_callback, this, std::placeholders::_1));
  }

private:
  // Interface buttons subscription callback
  void interface_buttons_callback(
    const irobot_create_msgs::msg::InterfaceButtons::SharedPtr create3_buttons_msg)
  {}

  // Interface Button Subscriber
  rclcpp::Subscription<irobot_create_msgs::msg::InterfaceButtons>::SharedPtr
interface_buttons_subscriber_;
};
```

Notice that the interface_buttons_subscriber_ uses the InterfaceButtons message type, and the quality of service is rclcpp::SensorDataQoS(). These parameters must match the topic, otherwise the subscription will fail. If you are unsure what message type or QoS a topic is using, you can use the ROS2 CLI to find this information.

Call ros2 topic info /<topic> --verbose to get the full details.



```
rkreinin@CPR02115L:~ $ ros2 topic info /interface_buttons --verbose
Type: irobot_create_msgs/msg/InterfaceButtons

Publisher count: 1

Node name: ui_mgr
Node namespace: /
Topic type: irobot_create_msgs/msg/InterfaceButtons
Endpoint type: PUBLISHER
GID: 01.10.7f.2b.08.6b.5a.b4.53.2a.ce.ed.00.00.eb.03.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

Subscription count: 1

Node name: root_ble_interface
Node namespace: /
Topic type: irobot_create_msgs/msg/InterfaceButtons
Endpoint type: SUBSCRIPTION
GID: 01.10.7f.2b.08.6b.5a.b4.53.2a.ce.ed.00.01.4d.04.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds
```

ROS2 topic information

Test Create® 3 Button 1

Now that we are subscribed, lets test out our node by printing a message every time button 1 is pressed.

Edit the interface_buttons_callback function to look like this:

```
// Interface buttons subscription callback
void interface_buttons_callback(
  const irobot_create_msgs::msg::InterfaceButtons::SharedPtr create3_buttons_msg)
{
  // Button 1 is pressed
  if (create3_buttons_msg->button_1.is_pressed) {
    RCLCPP_INFO(this->get_logger(), "Button 1 Pressed!");
  }
}
```

Now every time we receive a message on the /interface_buttons topic we will check if button 1 is pressed, and if it is then the node will print a message.

To test this out, we will need to build our package using colcon:

```
cd ~/turtlebot4_ws

colcon build --packages-select turtlebot4_cpp_tutorials

source install/local_setup.bash
```

The --packages-select flag allows you to enter any number of packages that you want to build, in case you don't want to build all packages in your workspace.

Now, try running the node:

```
ros2 run turtlebot4_cpp_tutorials turtlebot4_first_cpp_node
```

When you run it, nothing will happen until you press button 1 on your TurtleBot 4.

Press the button, and you should see this message in your terminal:

```
[INFO] [1652379086.090977658] [turtlebot4_first_cpp_node]: Button 1 Pressed!
```

Tip

Printing messages like this is a great way to debug your code.

Create a lightring publisher

Now that we can receive a button press, lets create a lightring publisher.

```
class TurtleBot4FirstNode : public rclcpp::Node
{
public:
  TurtleBot4FirstNode()
  : Node("turtlebot4_first_cpp_node")
  {
    // Subscribe to the /interface_buttons topic
    interface_buttons_subscriber_ =
      this->create_subscription<irobot_create_msgs::msg::InterfaceButtons>(
      "/interface_buttons",
      rclcpp::SensorDataQoS(),
      std::bind(&TurtleBot4FirstNode::interface_buttons_callback, this, std::placeholders::_1));

    // Create a publisher for the /cmd_lightring topic
    lightring_publisher_ = this->create_publisher<irobot_create_msgs::msg::LightringLeds>(
      "/cmd_lightring",
      rclcpp::SensorDataQoS());
  }

private:
  // Interface buttons subscription callback
  void interface_buttons_callback(
    const irobot_create_msgs::msg::InterfaceButtons::SharedPtr create3_buttons_msg)
  {
    // Button 1 is pressed
    if (create3_buttons_msg->button_1.is_pressed) {
      RCLCPP_INFO(this->get_logger(), "Button 1 Pressed!");
    }
  }

  // Interface Button Subscriber
  rclcpp::Subscription<irobot_create_msgs::msg::InterfaceButtons>::SharedPtr
interface_buttons_subscriber_;
  // Lightring Publisher
  rclcpp::Publisher<irobot_create_msgs::msg::LightringLeds>::SharedPtr lightring_publisher_;
};
```

Note
The Lightring publisher uses the LightringLeds message type.
Next, lets create a function that will populate a LightringLeds message, and publish it.
Add this code below your interface_buttons_callback function:

```
 // Perform this function when Button 1 is pressed.
```

```
void button_1_function()
{
  // Create a ROS2 message
  auto lightring_msg = irobot_create_msgs::msg::LightringLeds();
  // Stamp the message with the current time
  lightring_msg.header.stamp = this->get_clock()->now();

  // Override system lights
  lightring_msg.override_system = true;

  // LED 0
  lightring_msg.leds[0].red = 255;
  lightring_msg.leds[0].blue = 0;
  lightring_msg.leds[0].green = 0;

  // LED 1
  lightring_msg.leds[1].red = 0;
  lightring_msg.leds[1].blue = 255;
  lightring_msg.leds[1].green = 0;

  // LED 2
  lightring_msg.leds[2].red = 0;
  lightring_msg.leds[2].blue = 0;
  lightring_msg.leds[2].green = 255;

  // LED 3
  lightring_msg.leds[3].red = 255;
  lightring_msg.leds[3].blue = 255;
  lightring_msg.leds[3].green = 0;

  // LED 4
  lightring_msg.leds[4].red = 255;
  lightring_msg.leds[4].blue = 0;
  lightring_msg.leds[4].green = 255;

  // LED 5
  lightring_msg.leds[5].red = 0;
  lightring_msg.leds[5].blue = 255;
  lightring_msg.leds[5].green = 255;
  // Publish the message
  lightring_publisher_->publish(lightring_msg);
}
```

This function creates a LightringLeds message and populates the parameters.

We first stamp the message with the current time:

```
lightring_msg.header.stamp = this->get_clock()->now();
```

Then we set the override_system parameter to true so that our command overrides whatever commands the Create® 3 is sending to the lightring.

```
lightring_msg.override_system = true;
```

Next, we populate the 6 LEDs in the leds array with whatever colours we want.

```
// LED 0
lightring_msg.leds[0].red = 255;
lightring_msg.leds[0].blue = 0;
lightring_msg.leds[0].green = 0;

// LED 1
lightring_msg.leds[1].red = 0;
lightring_msg.leds[1].blue = 255;
lightring_msg.leds[1].green = 0;

// LED 2
lightring_msg.leds[2].red = 0;
lightring_msg.leds[2].blue = 0;
lightring_msg.leds[2].green = 255;

// LED 3
lightring_msg.leds[3].red = 255;
lightring_msg.leds[3].blue = 255;
lightring_msg.leds[3].green = 0;

// LED 4
lightring_msg.leds[4].red = 255;
lightring_msg.leds[4].blue = 0;
lightring_msg.leds[4].green = 255;

// LED 5
lightring_msg.leds[5].red = 0;
lightring_msg.leds[5].blue = 255;
lightring_msg.leds[5].green = 255;
```

Tip

Each RGB value can be set between 0 and 255. You can look up the RGB value of any color and set it here.

Finally, we publish the message.

```
self.lightring_publisher.publish(lightring_msg)
```

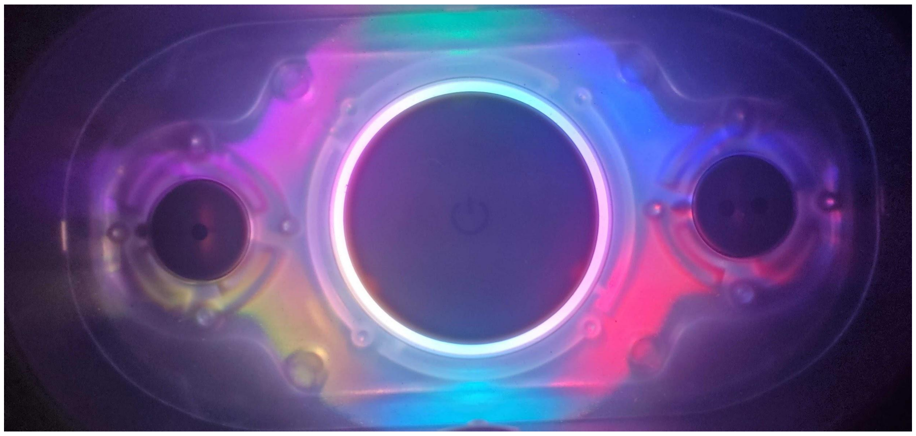
Publish the lightring command with a button press

Now we can connect our interface button subscription to our lightring publisher. Simply call button_1_function inside the interface_buttons_callback.

```
# Interface buttons subscription callback
def interface_buttons_callback(self, create3_buttons_msg: InterfaceButtons):
  # Button 1 is pressed
  if create3_buttons_msg.button_1.is_pressed:
    self.get_logger().info('Button 1 Pressed!')
    self.button_1_function()
```

Test this out by running the node like before.

Press button 1 and the lightring light should look like this:


Lightring colours controlled with the press of a button!

Toggle the lightring

You will notice that once you have set the lightrings LEDs they will remain like that forever. Lets make the button toggle the light on or off each time we press it.

Add a boolean to keep track of the light state:

```
bool lights_on_;
```

Initialize the boolean in the class constructor:

```
TurtleBot4FirstNode()
  : Node("turtlebot4_first_cpp_node"), lights_on_(false)
```

And modify button_1_function to toggle the light:

```
 void button_1_function()
  {
    // Create a ROS2 message
    auto lightring_msg = irobot_create_msgs::msg::LightringLeds();
    // Stamp the message with the current time
    lightring_msg.header.stamp = this->get_clock()->now();

    // Lights are currently off
    if (!lights_on_) {
      // Override system lights
      lightring_msg.override_system = true;

      // LED 0
      lightring_msg.leds[0].red = 255;
      lightring_msg.leds[0].blue = 0;
      lightring_msg.leds[0].green = 0;

      // LED 1
      lightring_msg.leds[1].red = 0;
      lightring_msg.leds[1].blue = 255;
      lightring_msg.leds[1].green = 0;

      // LED 2
      lightring_msg.leds[2].red = 0;
      lightring_msg.leds[2].blue = 0;
      lightring_msg.leds[2].green = 255;

      // LED 3
      lightring_msg.leds[3].red = 255;
      lightring_msg.leds[3].blue = 255;
      lightring_msg.leds[3].green = 0;

      // LED 4
      lightring_msg.leds[4].red = 255;
      lightring_msg.leds[4].blue = 0;
      lightring_msg.leds[4].green = 255;

      // LED 5
      lightring_msg.leds[5].red = 0;
      lightring_msg.leds[5].blue = 255;
      lightring_msg.leds[5].green = 255;
    }
    // Lights are currently on
    else {
      // Disable system override. The system will take back control of the lightring.
      lightring_msg.override_system = false;
    }
    // Publish the message
    lightring_publisher_->publish(lightring_msg);
    // Toggle the lights on status
    lights_on_ = !lights_on_;
  }
```

Now the Create® 3 will regain control of the lightring if we press button 1 again.

Your first C++ Node

You have finished writing your first C++ node! The final .cpp file should look like this:

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"

#include "irobot_create_msgs/msg/interface_buttons.hpp"
#include "irobot_create_msgs/msg/lightring_leds.hpp"

class TurtleBot4FirstNode : public rclcpp::Node
{
public:
  TurtleBot4FirstNode()
  : Node("turtlebot4_first_cpp_node"), lights_on_(false)
  {
    // Subscribe to the /interface_buttons topic
    interface_buttons_subscriber_ =
      this->create_subscription<irobot_create_msgs::msg::InterfaceButtons>(
      "/interface_buttons",
      rclcpp::SensorDataQoS(),
      std::bind(&TurtleBot4FirstNode::interface_buttons_callback, this, std::placeholders::_1));

    // Create a publisher for the /cmd_lightring topic
    lightring_publisher_ = this->create_publisher<irobot_create_msgs::msg::LightringLeds>(
      "/cmd_lightring",
      rclcpp::SensorDataQoS());
  }

private:
  // Interface buttons subscription callback
  void interface_buttons_callback(
    const irobot_create_msgs::msg::InterfaceButtons::SharedPtr create3_buttons_msg)
  {
    // Button 1 is pressed
    if (create3_buttons_msg->button_1.is_pressed) {
      RCLCPP_INFO(this->get_logger(), "Button 1 Pressed!");

      button_1_function();
    }
  }

  // Perform a function when Button 1 is pressed.
  void button_1_function()
  {
    // Create a ROS2 message
    auto lightring_msg = irobot_create_msgs::msg::LightringLeds();
    // Stamp the message with the current time
    lightring_msg.header.stamp = this->get_clock()->now();

    // Lights are currently off
    if (!lights_on_) {
      // Override system lights
      lightring_msg.override_system = true;

      // LED 0
      lightring_msg.leds[0].red = 255;
      lightring_msg.leds[0].blue = 0;
      lightring_msg.leds[0].green = 0;

      // LED 1
      lightring_msg.leds[1].red = 0;
      lightring_msg.leds[1].blue = 255;
      lightring_msg.leds[1].green = 0;

      // LED 2
      lightring_msg.leds[2].red = 0;
      lightring_msg.leds[2].blue = 0;
      lightring_msg.leds[2].green = 255;

      // LED 3
      lightring_msg.leds[3].red = 255;
      lightring_msg.leds[3].blue = 255;
      lightring_msg.leds[3].green = 0;

      // LED 4
      lightring_msg.leds[4].red = 255;
      lightring_msg.leds[4].blue = 0;
      lightring_msg.leds[4].green = 255;

      // LED 5
      lightring_msg.leds[5].red = 0;
      lightring_msg.leds[5].blue = 255;
      lightring_msg.leds[5].green = 255;
    }
    // Lights are currently on
    else {
      // Disable system override. The system will take back control of the lightring.
      lightring_msg.override_system = false;
    }
    // Publish the message
    lightring_publisher_->publish(lightring_msg);
    // Toggle the lights on status
    lights_on_ = !lights_on_;
  }

  // Interface Button Subscriber
  rclcpp::Subscription<irobot_create_msgs::msg::InterfaceButtons>::SharedPtr
    interface_buttons_subscriber_;
  // Lightring Publisher
  rclcpp::Publisher<irobot_create_msgs::msg::LightringLeds>::SharedPtr lightring_publisher_;
```

```
  // Lights on status
  bool lights_on_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<TurtleBot4FirstNode>());
  rclcpp::shutdown();
  return 0;
}
```

Don't forget to build the package again before running the node.

Creating your first node (Python)

This tutorial will go through the steps of creating a ROS2 package and writing a ROS2 node in Python. For a C++ example, click here. These steps are similar to the ROS2 Tutorial, but focus on interacting with the TurtleBot 4. For source code, click here.

Note

You can follow this tutorial on either the Raspberry Pi of your TurtleBot 4, or your PC.

Create a workspace

If you do not already have a workspace, open a terminal and create one in the directory of your choice:

```
mkdir ~/turtlebot4_ws/src -p
```

Create a package and node

You will need to create a ROS2 package to hold your files. For this tutorial, we will create a package called turtlebot4_python_tutorials with a node called turtlebot4_first_python_node.

```
source /opt/ros/galactic/setup.bash
cd ~/turtlebot4_ws/src
ros2 pkg create --build-type ament_python --node-name turtlebot4_first_python_node
turtlebot4_python_tutorials
```

This will create a turtlebot4_python_tutorials folder and populate it with a basic "Hello World" node, as well as the setup and package.xml files required for a ROS2 Python package.

Write your node

The next step is to start coding. For this tutorial, our goal will be to use the Create® 3 interface button 1 to change the colour of the Create® 3 lightring. Open up the "Hello World" .py file located at ~/turtlebot4_ws/src/turtlebot4_python_tutorials/turtlebot4_python_tutorials/turtlebot4_first_python_node.py in your favourite text editor.

Add your dependencies

For this tutorial, we will need to use the rclpy and irobot_create_msgs packages. The rclpy package allows us to create ROS2 nodes and gives us full access to all the base ROS2 functionality in Python. The irobot_create_msgs package gives us access to the custom messages used by the Create® 3 for reading the button presses and controlling the lightring.

In package.xml, add these lines under <buildtool_depend>ament_cmake</buildtool_depend>:

```
<depend>rclpy</depend>
<depend>irobot_create_msgs</depend>
```

In your .py file, import these packages:

```
from irobot_create_msgs.msg import InterfaceButtons, LightringLeds

import rclpy
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data
```

Create a class

Now that the dependencies are set, we can create a class that inherits from the rclpy.Node class. We will call this class TurtleBot4FirstNode.

```
class TurtleBot4FirstNode(Node):
    def __init__(self):
        super().__init__('turtlebot4_first_python_node')
```

Notice that our class calls the super() constructor and passes it the name of our node, turtlebot4_first_python_node.

We can now create our node in the main function and spin it. Since our node is empty, the node will be created but it won't do anything.

```
def main(args=None):
    rclpy.init(args=args)
    node = TurtleBot4FirstNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

Subscribe to the Create® 3 interface buttons

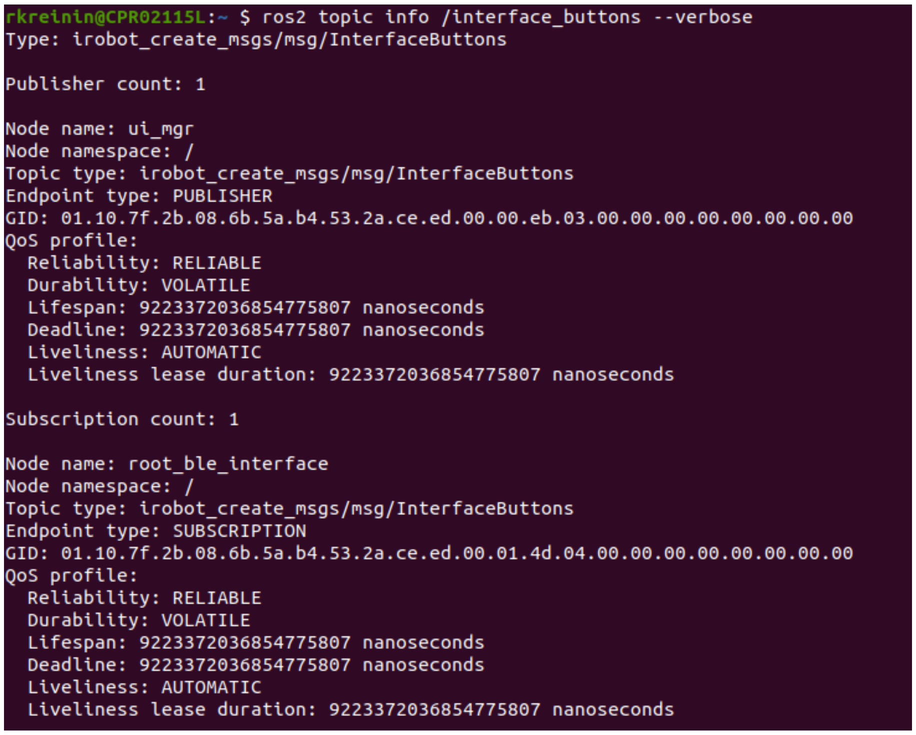
Our next step is to subscribe to the Create® 3 interface buttons topic to receive button presses.

We will need to create a rclpy.Subscription as well as a callback function for the subscription. The callback function will be called every time we receive a message on the interface buttons topic.

```
class TurtleBot4FirstNode(Node):
    lights_on_ = False

    def __init__(self):
        super().__init__('turtlebot4_first_python_node')

        # Subscribe to the /interface_buttons topic
        self.interface_buttons_subscriber = self.create_subscription(
            InterfaceButtons,
            '/interface_buttons',
            self.interface_buttons_callback,
            qos_profile_sensor_data)

    # Interface buttons subscription callback
    def interface_buttons_callback(self, create3_buttons_msg: InterfaceButtons):
```

Notice that the interface_buttons_subscriber uses the InterfaceButtons message type, and the quality of service is qos_profile_sensor_data. These parameters must match the topic, otherwise the subscription will fail. If you are unsure what message type or QoS a topic is using, you can use the ROS2 CLI to find this information.

Call `ros2 topic info /<topic> --verbose` to get the full details.



```
rkreinin@CPR02115L:~ $ ros2 topic info /interface_buttons --verbose
Type: irobot_create_msgs/msg/InterfaceButtons

Publisher count: 1

Node name: ui_mgr
Node namespace: /
Topic type: irobot_create_msgs/msg/InterfaceButtons
Endpoint type: PUBLISHER
GID: 01.10.7f.2b.08.6b.5a.b4.53.2a.ce.ed.00.00.eb.03.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds

Subscription count: 1

Node name: root_ble_interface
Node namespace: /
Topic type: irobot_create_msgs/msg/InterfaceButtons
Endpoint type: SUBSCRIPTION
GID: 01.10.7f.2b.08.6b.5a.b4.53.2a.ce.ed.00.01.4d.04.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  Durability: VOLATILE
  Lifespan: 9223372036854775807 nanoseconds
  Deadline: 9223372036854775807 nanoseconds
  Liveliness: AUTOMATIC
  Liveliness lease duration: 9223372036854775807 nanoseconds
```

ROS2 topic information

Test Create® 3 Button 1

Now that we are subscribed, lets test out our node by printing a message every time button 1 is pressed.

Edit the interface_buttons_callback function to look like this:

```
# Interface buttons subscription callback
def interface_buttons_callback(self, create3_buttons_msg: InterfaceButtons):
    # Button 1 is pressed
    if create3_buttons_msg.button_1.is_pressed:
        self.get_logger().info('Button 1 Pressed!')
```

Now every time we receive a message the one /interface_buttons topic we will check if button 1 is pressed, and if it is then the node will print a message.

To test this out, we will need to build our package using colcon:

```
cd ~/turtlebot4_ws
colcon build --symlink-install --packages-select turtlebot4_python_tutorials
source install/local_setup.bash
```

The --symlink-install allows us to install a symbolic link to our Python script, rather than a copy of the script. This means that any changes we make to the script will be applied to the installed script, so we don't need to rebuild the package after each change.

The --packages-select flag allows you to enter any number of packages that you want to build, in case you don't want to build all packages in your workspace.

Now, try running the node:

```
ros2 run turtlebot4_python_tutorials turtlebot4_first_python_node
```

When you run it, nothing will happen until you press button 1 on your TurtleBot 4.

Press the button, and you should see this message in your terminal:

```
[INFO] [1652384338.145094927] [turtlebot4_first_python_node]: Button 1 Pressed!
```

Tip

Printing messages like this is a great way to debug your code.

Create a lightring publisher

Now that we can receive a button press, lets create a lightring publisher.

```
class TurtleBot4FirstNode(Node):
    def __init__(self):
        super().__init__('turtlebot4_first_python_node')

        # Subscribe to the /interface_buttons topic
        self.interface_buttons_subscriber = self.create_subscription(
            InterfaceButtons,
            '/interface_buttons',
            self.interface_buttons_callback,
            qos_profile_sensor_data)

        # Create a publisher for the /cmd_lightring topic
        self.lightring_publisher = self.create_publisher(
            LightringLeds,
            '/cmd_lightring',
            qos_profile_sensor_data)
```

Note

The Lightring publisher uses the LightringLeds message type.

Next, lets create a function that will populate a LightringLeds message, and publish it.

Add this code below your interface_buttons_callback function:

```
def button_1_function(self):
    # Create a ROS2 message
    lightring_msg = LightringLeds()
    # Stamp the message with the current time
    lightring_msg.header.stamp = self.get_clock().now().to_msg()

    # Override system lights
    lightring_msg.override_system = True

    # LED 0
    lightring_msg.leds[0].red = 255
    lightring_msg.leds[0].blue = 0
    lightring_msg.leds[0].green = 0
    # LED 1
    lightring_msg.leds[1].red = 0
    lightring_msg.leds[1].blue = 255
    lightring_msg.leds[1].green = 0

    # LED 2
    lightring_msg.leds[2].red = 0
    lightring_msg.leds[2].blue = 0
    lightring_msg.leds[2].green = 255

    # LED 3
    lightring_msg.leds[3].red = 255
    lightring_msg.leds[3].blue = 255
    lightring_msg.leds[3].green = 0

    # LED 4
    lightring_msg.leds[4].red = 255
    lightring_msg.leds[4].blue = 0
    lightring_msg.leds[4].green = 255

    # LED 5
    lightring_msg.leds[5].red = 0
    lightring_msg.leds[5].blue = 255
    lightring_msg.leds[5].green = 255

    # Publish the message
    self.lightring_publisher.publish(lightring_msg)
```

This function creates a LightringLeds message and populates the parameters.

We first stamp the message with the current time:

```
lightring_msg.header.stamp = self.get_clock().now().to_msg()
```

Then we set the override_system parameter to True so that our command overrides whatever commands the Create® 3 is sending to the lightring.

```
lightring_msg.override_system = True
```

Next, we populate the 6 LEDs in the leds array with whatever colours we want.

```
# LED 0
lightring_msg.leds[0].red = 255
lightring_msg.leds[0].blue = 0
lightring_msg.leds[0].green = 0

# LED 1
lightring_msg.leds[1].red = 0
lightring_msg.leds[1].blue = 255
lightring_msg.leds[1].green = 0

# LED 2
lightring_msg.leds[2].red = 0
lightring_msg.leds[2].blue = 0
lightring_msg.leds[2].green = 255

# LED 3
lightring_msg.leds[3].red = 255
lightring_msg.leds[3].blue = 255
lightring_msg.leds[3].green = 0

# LED 4
lightring_msg.leds[4].red = 255
lightring_msg.leds[4].blue = 0
lightring_msg.leds[4].green = 255

# LED 5
lightring_msg.leds[5].red = 0
lightring_msg.leds[5].blue = 255
lightring_msg.leds[5].green = 255
```

Tip

Each RGB value can be set between 0 and 255. You can look up the RGB value of any color and set it here.

Finally, we publish the message.

```
self.lightring_publisher.publish(lightring_msg)
```

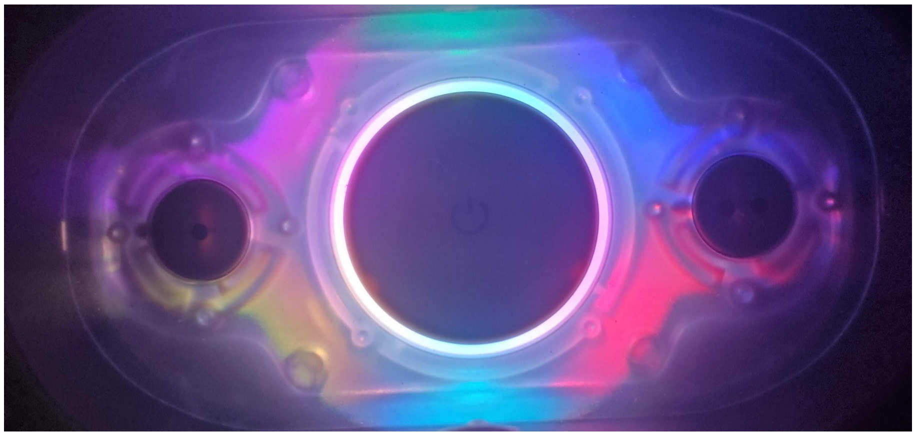
Publish the lightring command with a button press

Now we can connect our interface button subscription to our lightring publisher. Simply call button_1_function inside the interface_buttons_callback.

```
# Interface buttons subscription callback
def interface_buttons_callback(self, create3_buttons_msg: InterfaceButtons):
    # Button 1 is pressed
    if create3_buttons_msg.button_1.is_pressed:
        self.get_logger().info('Button 1 Pressed!')
        self.button_1_function()
```

Test this out by running the node like before.

Press button 1 and the lightring light should look like this:


Lightring colours controlled with the press of a button!

Toggle the lightring

You will notice that once you have set the lightrings LEDs they will remain like that forever. Lets make the button toggle the light on or off each time we press it.

Add a boolean to keep track of the light state:

```
class TurtleBot4FirstNode(Node):
    lights_on_ = False

    def __init__(self):
```

And modify button_1_function to toggle the light:

```
# Perform a function when Button 1 is pressed
def button_1_function(self):
    # Create a ROS2 message
    lightring_msg = LightringLeds()
    # Stamp the message with the current time
    lightring_msg.header.stamp = self.get_clock().now().to_msg()

    # Lights are currently off
    if not self.lights_on_:
        # Override system lights
        lightring_msg.override_system = True

        # LED 0
        lightring_msg.leds[0].red = 255
        lightring_msg.leds[0].blue = 0
        lightring_msg.leds[0].green = 0

        # LED 1
        lightring_msg.leds[1].red = 0
        lightring_msg.leds[1].blue = 255
        lightring_msg.leds[1].green = 0

        # LED 2
        lightring_msg.leds[2].red = 0
        lightring_msg.leds[2].blue = 0
        lightring_msg.leds[2].green = 255

        # LED 3
        lightring_msg.leds[3].red = 255
        lightring_msg.leds[3].blue = 255
        lightring_msg.leds[3].green = 0

        # LED 4
        lightring_msg.leds[4].red = 255
        lightring_msg.leds[4].blue = 0
        lightring_msg.leds[4].green = 255

        # LED 5
        lightring_msg.leds[5].red = 0
        lightring_msg.leds[5].blue = 255
        lightring_msg.leds[5].green = 255
    # Lights are currently on
    else:
        # Disable system override. The system will take back control of the lightring.
        lightring_msg.override_system = False

    # Publish the message
    self.lightring_publisher.publish(lightring_msg)
    # Toggle the lights on status
    self.lights_on_ = not self.lights_on_
```

Now the Create® 3 will regain control of the lightring if we press button 1 again.

Your first Python Node

You have finished writing your first Python node! The final .py file should look like this:

```
from irobot_create_msgs.msg import InterfaceButtons, LightringLeds

import rclpy
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data


class TurtleBot4FirstNode(Node):
    lights_on_ = False

    def __init__(self):
        super().__init__('turtlebot4_first_python_node')

        # Subscribe to the /interface_buttons topic
        self.interface_buttons_subscriber = self.create_subscription(
            InterfaceButtons,
            '/interface_buttons',
            self.interface_buttons_callback,
            qos_profile_sensor_data)
        # Create a publisher for the /cmd_lightring topic
        self.lightring_publisher = self.create_publisher(
            LightringLeds,
            '/cmd_lightring',
            qos_profile_sensor_data)

    # Interface buttons subscription callback
    def interface_buttons_callback(self, create3_buttons_msg: InterfaceButtons):
        # Button 1 is pressed
        if create3_buttons_msg.button_1.is_pressed:
            self.get_logger().info('Button 1 Pressed!')
            self.button_1_function()

    # Perform a function when Button 1 is pressed
    def button_1_function(self):
        # Create a ROS2 message
        lightring_msg = LightringLeds()
        # Stamp the message with the current time
        lightring_msg.header.stamp = self.get_clock().now().to_msg()

        # Lights are currently off
        if not self.lights_on_:
            # Override system lights
            lightring_msg.override_system = True

            # LED 0
            lightring_msg.leds[0].red = 255
            lightring_msg.leds[0].blue = 0
            lightring_msg.leds[0].green = 0

            # LED 1
            lightring_msg.leds[1].red = 0
            lightring_msg.leds[1].blue = 255
            lightring_msg.leds[1].green = 0

            # LED 2
            lightring_msg.leds[2].red = 0
            lightring_msg.leds[2].blue = 0
            lightring_msg.leds[2].green = 255

            # LED 3
            lightring_msg.leds[3].red = 255
            lightring_msg.leds[3].blue = 255
            lightring_msg.leds[3].green = 0
            # LED 4
            lightring_msg.leds[4].red = 255
            lightring_msg.leds[4].blue = 0
            lightring_msg.leds[4].green = 255

            # LED 5
            lightring_msg.leds[5].red = 0
            lightring_msg.leds[5].blue = 255
            lightring_msg.leds[5].green = 255
        # Lights are currently on
        else:
            # Disable system override. The system will take back control of the lightring.
            lightring_msg.override_system = False

        # Publish the message
        self.lightring_publisher.publish(lightring_msg)
        # Toggle the lights on status
        self.lights_on_ = not self.lights_on_


def main(args=None):
    rclpy.init(args=args)
    node = TurtleBot4FirstNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


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

Generating a map

In this tutorial we will be mapping an area by driving the TurtleBot 4 around and using SLAM. Start by making sure that the area you will be mapping is clear of unwanted obstacles. Ideally, you don't want people or animals moving around the area while creating the map.

Launch SLAM

First, make sure that the RPLIDAR and description nodes are running on the TurtleBot 4.

Then run SLAM. It is recommended to run synchronous SLAM on a remote PC to get a higher resolution map.

```
ros2 launch turtlebot4_navigation slam_sync.launch.py
```

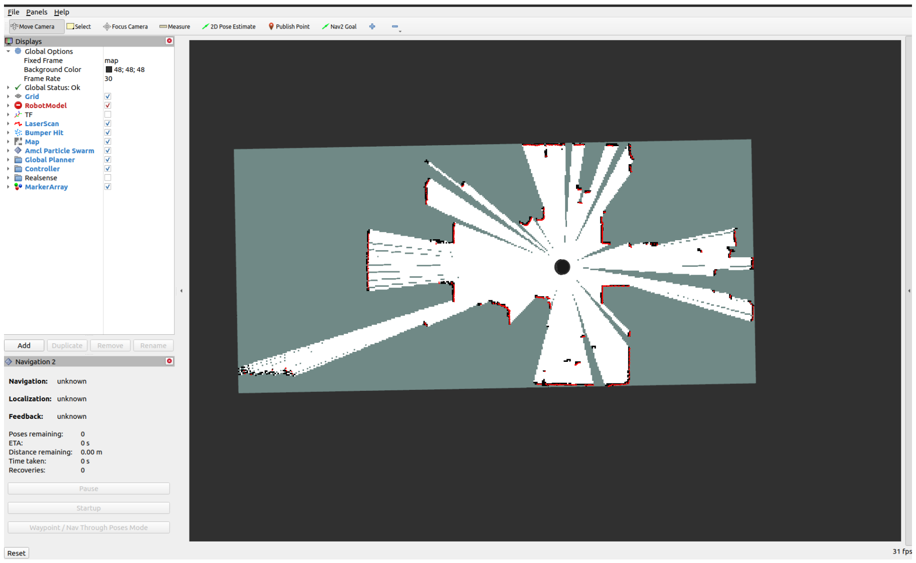
Asynchronous SLAM can be used as well.

```
ros2 launch turtlebot4_navigation slam_async.launch.py
```

Launch Rviz2

To visualise the map, launch Rviz2 with the view_robot launch file.

```
ros2 launch turtlebot4_viz view_robot.launch.py
```



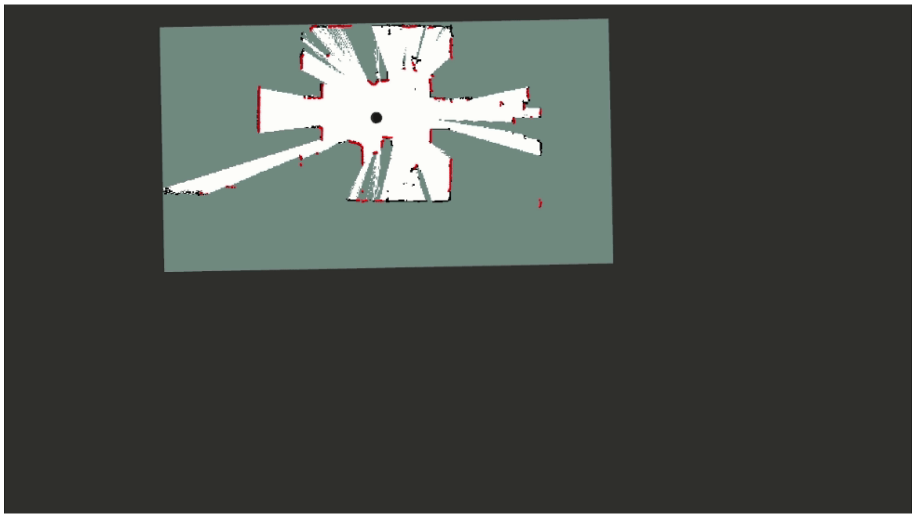
```
Global Options
  Fixed Frame        map
  Background Color   48; 48; 48
  Frame Rate         30
Global Status: Ok
Grid                 ✓
RobotModel           ✓
TF                   
LaserScan            ✓
Bumper Hit           ✓
Map                  ✓
Amcl Particle Swarm  ✓
Global Planner       ✓
Controller           ✓
Realsense            
MarkerArray          ✓

Navigation:    unknown
Localization:  unknown
Feedback:      unknown

Poses remaining:      0
ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0
```

Rviz2 showing a map generate by SLAM

Drive the TurtleBot 4

Use any method to drive the robot around the area you wish to map. Check out the driving tutorial if you are unsure of how to drive the robot.

Keep watch of RVIZ as you drive the robot around the area to make sure that the map gets filled out properly.



Generating a map by driving the TurtleBot 4

Save the map

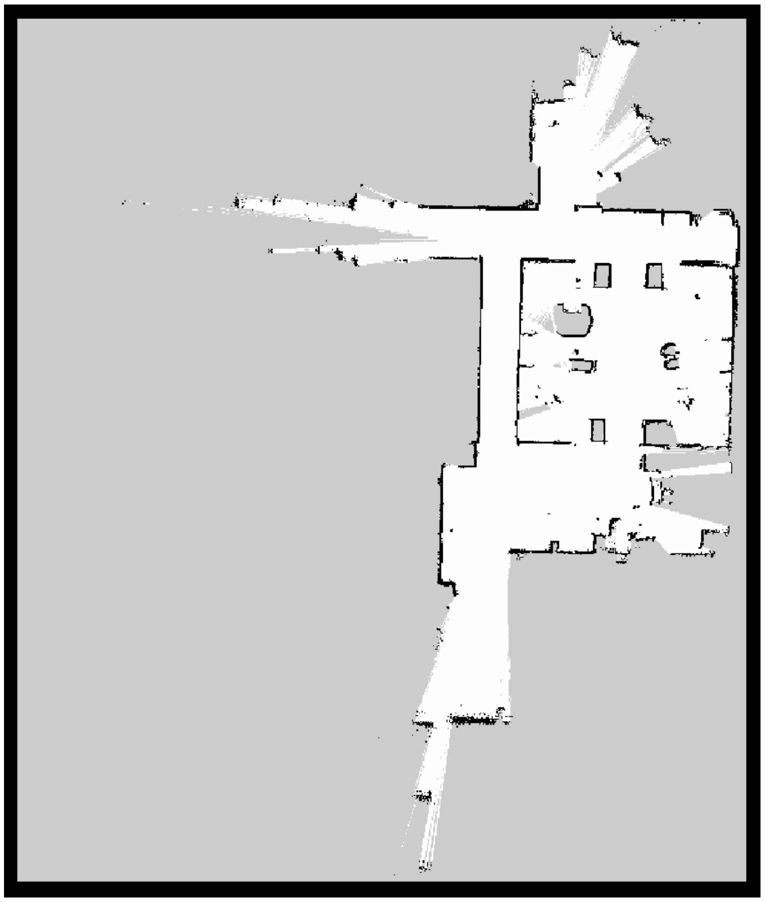
Once you are happy with your map, you can save it with the following command:

```
ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name:
  data: 'map_name'"
```

This will save the map to your current directory.

View the map

Once the map is saved it will generate a map_name.pgm file which can be viewed in an image editor. A map_name.yaml file is also created. You can edit this file to adjust the map parameters.


Generated map image

Navigation

This tutorial will cover various methods of navigating with the TurtleBot 4 and Nav2.

SLAM vs Localization

There are two localization methods we can use to figure out where the robot is on the map: SLAM or Localization. SLAM allows us to generate the map as we navigate, while localization requires that a map already exists.

SLAM

SLAM is useful for generating a new map, or navigating in unknown or dynamic environments. It updates the map as it detects and changes, but cannot see areas of the environment that it has not discovered yet.

Localization

Localization uses an existing map along with live odometry and laserscan data to figure out the position of the robot on the given map. It does not update the map if any changes have been made to the environment, but we can still avoid new obstacles when navigating. Because the map doesn't change, we can get more repeatable navigation results.

For this tutorial, we will be using localization to navigate on a map generated with SLAM.

Nav2

The TurtleBot 4 uses the Nav2 stack for navigation.

Launching navigation

For this tutorial we can launch navigation with Nav Bringup.

On a physical TurtleBot 4, call:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py slam:=off localization:=true
map:=office.yaml
```

Replace office.yaml with your own map.

If you are using the simulator, call:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py nav:=true slam:=off localization:=true
```

This will launch the simulation in the default depot world and will use the existing depot.yaml file for the map. If you are using a different world you will need to create a map for it and pass that in as a launch argument.

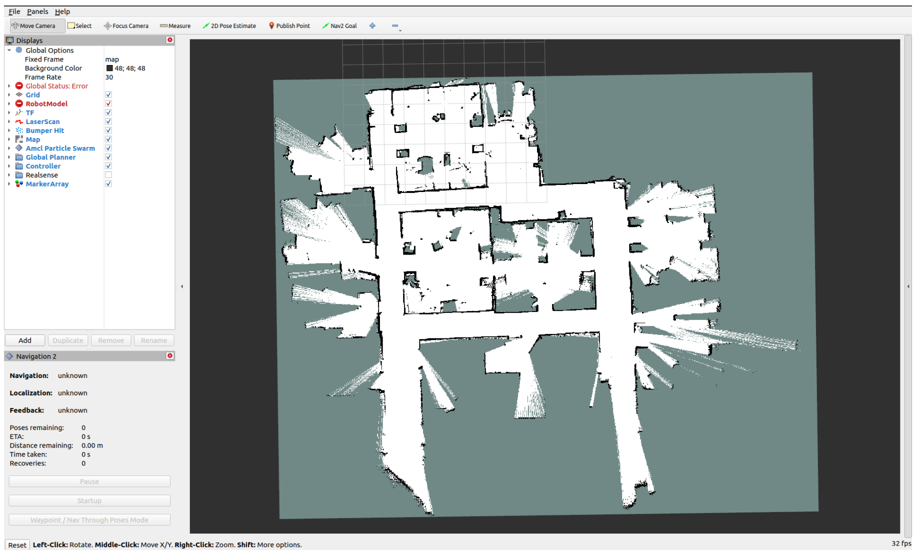
For example:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py nav:=true slam:=off localization:=true
world:=classroom map:=classroom.yaml
```

Interacting with Nav2

In a new terminal, launch Rviz so that you can view the map and interact with navigation:

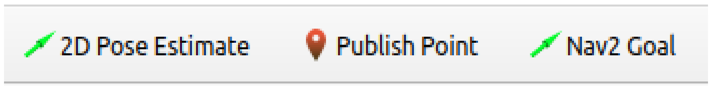
```
ros2 launch turtlebot4_viz view_robot.launch.py
```



```
Fixed Frame        map
Background Color   48; 48; 48
Frame Rate         30

Navigation:     unknown
Localization:   unknown
Feedback:       unknown

Poses remaining:      0
ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
```

Office Map shown in Rviz

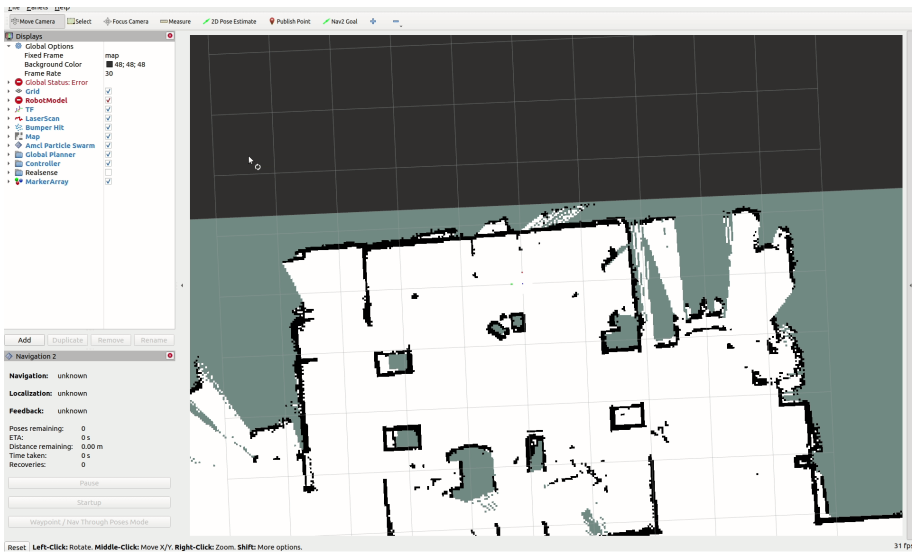
At the top of the Rviz window is the toolbar. You will notice that there are three navigation tools available to you.



Navigation tools in Rviz

2D Pose Estimate

The 2D Pose Estimate tool is used in localization to set the approximate initial pose of the robot on the map. This is required for the Nav2 stack to know where to start localizing from. Click on the tool, and then click and drag the arrow on the map to approximate the position and orientation of the robot.



Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30

Navigation: unknown
Localization: unknown
Feedback: unknown

Poses remaining: 0
ETA: 0 s
Distance remaining: 0.00 m
Time taken: 0 s
Recoveries: 0

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

Setting the initial pose

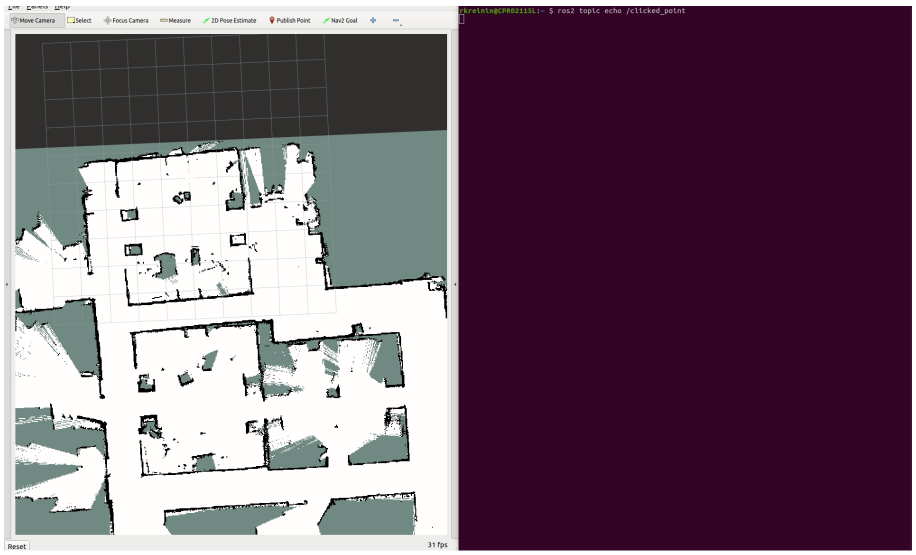
Publish Point

The Publish Point tool allows you to click on a point on the map, and have the coordinates of that point published to the /clicked_point topic.

Open a new terminal and call:

```
ros2 topic echo /clicked_point
```

Then, select the Publish Point tool and click on a point on the map. You should see the coordinates published in your terminal.

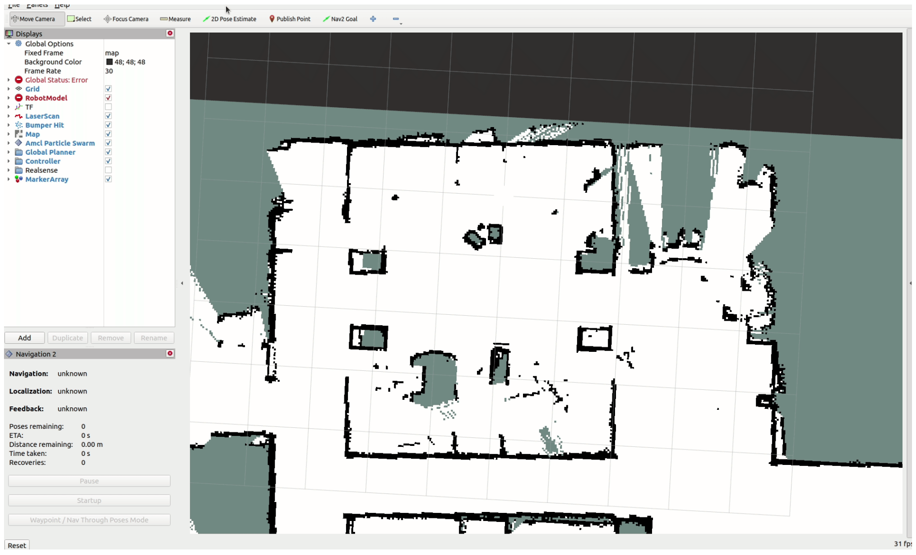


```
rkreinin@CPR02115L:~ $ ros2 topic echo /clicked_point
```

Getting a point coordinate

Nav2 Goal

The Nav2 Goal tool allows you to set a goal pose for the robot. The Nav2 stack will then plan a path to the goal pose and attempt to drive the robot there. Make sure to set the initial pose of the robot before you set a goal pose.



```
Global Options
    Fixed Frame          map
    Background Color     48; 48; 48
    Frame Rate           30
Global Status: Error
Grid                     ✓
RobotModel               ✓
TF                       ☐
LaserScan                ✓
Bumper Hit               ✓
Map                      ✓
Amcl Particle Swarm      ✓
Global Planner           ✓
Controller               ✓
Realsense                ☐
MarkerArray              ✓
```

```
Navigation 2

Navigation:    unknown

Localization:  unknown

Feedback:      unknown

Poses remaining:      0
ETA:                  0 s
Distance remaining:   0.00 m
Time taken:           0 s
Recoveries:           0
```

Driving the TurtleBot4 with a Nav2 Goal

TurtleBot 4 Navigator

The TurtleBot 4 Navigator is a Python node that adds on to the Nav2 Simple Commander. It includes TurtleBot 4 specific features such as docking and undocking, as well as easy to use methods for navigating.

Note

TurtleBot 4 Navigator requires at least version 1.0.11 of Nav2 Simple Commander

The code for the following examples is available at https://github.com/turtlebot/turtlebot4_tutorials. For each example, the robot starts on a dock at the origin of the map.

Navigate to Pose

This example demonstrates the same behaviour as Nav2 Goal. The Nav2 stack is given a pose on the map with which it calculates a path. The robot then attempts to drive along the path. This example is demonstrated in the depot world of the TurtleBot 4 simulation.

To run this example, start the Ignition simulation:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py nav:=true slam:=off localization:=true
```

Once the simulation has started, open another terminal and run:

```
ros2 run turtlebot4_python_tutorials nav_to_pose
```

Code breakdown

The source code for this example is available here.

Lets take a look at the main function.

```
def main():
    rclpy.init()

    navigator = TurtleBot4Navigator()

    # Start on dock
    if not navigator.getDockedStatus():
        navigator.info('Docking before intialising pose')
        navigator.dock()
    # Set initial pose
    initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
    navigator.setInitialPose(initial_pose)

    # Wait for Nav2
    navigator.waitUntilNav2Active()

    # Set goal poses
    goal_pose = navigator.getPoseStamped([13.0, 5.0], TurtleBot4Directions.EAST)

    # Undock
    navigator.undock()

    # Go to each goal pose
    navigator.startToPose(goal_pose)

    rclpy.shutdown()
```

Initialise the node

We start by initialising rclpy and creating the TurtleBot4Navigator object. This will initialise any ROS2 publishers, subscribers and action clients that we need.

```
rclpy.init()

navigator = TurtleBot4Navigator()
```

Dock the robot

Next, we check if the robot is docked. If it is not, we send an action goal to dock the robot. By docking the robot we guarantee that it is at the [0.0, 0.0] coordinates on the map.

```
if not navigator.getDockedStatus():
    navigator.info('Docking before intialising pose')
    navigator.dock()
```

Set the initial pose

Now that we know the robot is docked, we can set the initial pose to [0.0, 0.0], facing "North".

```
initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
navigator.setInitialPose(initial_pose)
```

The TurtleBot 4 Navigator uses cardinal directions to set the orientation of the robot relative to the map. You can use actual integers or floating points if you need a more precise direction.

```
class TurtleBot4Directions(IntEnum):
    NORTH = 0
    NORTH_WEST = 45
    WEST = 90
    SOUTH_WEST = 135
    SOUTH = 180
    SOUTH_EAST = 225
    EAST = 270
    NORTH_EAST = 315
```

Note

These cardinal directions are relative to the map, not the actual magnetic north pole. Driving north is equivalent to driving upwards on the map, west is driving left, and so on.

Wait for Nav2

Once the initial position has been set, the Nav2 stack will place the robot at that position on the map and begin localizing. We want to wait for Nav2 to be ready before we start sending navigation goals.

```
navigator.waitUntilNav2Active()
```

Note

This call will block until Nav2 is ready. Make sure you have launched nav bringup in a separate terminal.

Set the goal pose

Now we can create a geometry_msgs/PoseStamped message. The getPoseStamped method makes it easy for us. All we have to do is pass in a list describing the x and y position that we want to drive to on the map, and the direction that we want the robot to be facing when it reaches that point.

```
goal_pose = navigator.getPoseStamped([13.0, 5.0], TurtleBot4Directions.EAST)
```

Undock the robot and go to the goal pose

We are ready to drive to the goal pose. We start by undocking the robot so that it does not attempt to drive through the dock, and then send the goal pose. As the robot drives to the goal pose, we will be receiving feedback from the action. This feedback includes the estimated time of arrival.

navigator.undock()

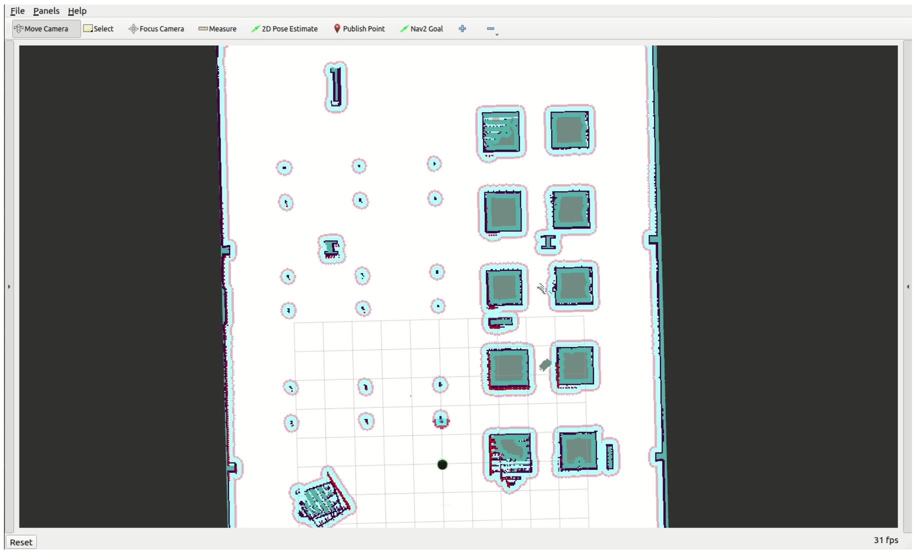
navigator.startToPose(goal_pose)

Once the robot has reached the goal, we call rclpy.shutdown() to gracefully destroy the rclpy context.

Watch navigation progress in Rviz

You can visualise the navigation process in Rviz by calling:

ros2 launch turtlebot4_viz view_robot.launch.py



Navigate to a pose

Navigate Through Poses

This example demonstrates the [Navigate Through Poses](#) behaviour tree. The Nav2 stack is given a set of poses on the map and creates a path that goes through each pose in order until the last pose is reached. The robot then attempts to drive along the path. This example is demonstrated in the depot world of the TurtleBot 4 simulation.

To run this example, start the Ignition simulation:

ros2 launch turtlebot4_ignition_bringup ignition.launch.py nav:=true slam:=off localization:=true

Once the simulation has started, open another terminal and run:

ros2 run turtlebot4_python_tutorials nav_through_poses

Code breakdown

The source code for this example is available here.

Lets take a look at the main function.

```
def main():
    rclpy.init()

    navigator = TurtleBot4Navigator()

    # Start on dock
    if not navigator.getDockedStatus():
        navigator.info('Docking before intialising pose')
        navigator.dock()

    # Set initial pose
    initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
    navigator.setInitialPose(initial_pose)

    # Wait for Nav2
    navigator.waitUntilNav2Active()

    # Set goal poses
    goal_pose = []
    goal_pose.append(navigator.getPoseStamped([0.0, -1.0], TurtleBot4Directions.NORTH))
    goal_pose.append(navigator.getPoseStamped([1.7, -1.0], TurtleBot4Directions.EAST))
    goal_pose.append(navigator.getPoseStamped([1.6, -3.5], TurtleBot4Directions.NORTH))
    goal_pose.append(navigator.getPoseStamped([6.75, -3.46],
TurtleBot4Directions.NORTH_WEST))
    goal_pose.append(navigator.getPoseStamped([7.4, -1.0], TurtleBot4Directions.SOUTH))
    goal_pose.append(navigator.getPoseStamped([-1.0, -1.0], TurtleBot4Directions.WEST))

    # Undock
    navigator.undock()

    # Navigate through poses
    navigator.startThroughPoses(goal_pose)

    # Finished navigating, dock
    navigator.dock()

    rclpy.shutdown()
```

This example starts the same as navigate to pose. We initialse the node, make sure the robot is docked, and set the initial pose. Then we wait for Nav2 to become active.

Set goal poses

The next step is to create a list of PoseStamped messages which represent the poses that the robot needs to drive through.

```
goal_pose = []
goal_pose.append(navigator.getPoseStamped([0.0, -1.0], TurtleBot4Directions.NORTH))
goal_pose.append(navigator.getPoseStamped([1.7, -1.0], TurtleBot4Directions.EAST))
goal_pose.append(navigator.getPoseStamped([1.6, -3.5], TurtleBot4Directions.NORTH))
goal_pose.append(navigator.getPoseStamped([6.75, -3.46],
TurtleBot4Directions.NORTH_WEST))
goal_pose.append(navigator.getPoseStamped([7.4, -1.0], TurtleBot4Directions.SOUTH))
goal_pose.append(navigator.getPoseStamped([-1.0, -1.0], TurtleBot4Directions.WEST))
```

Navigate through the poses

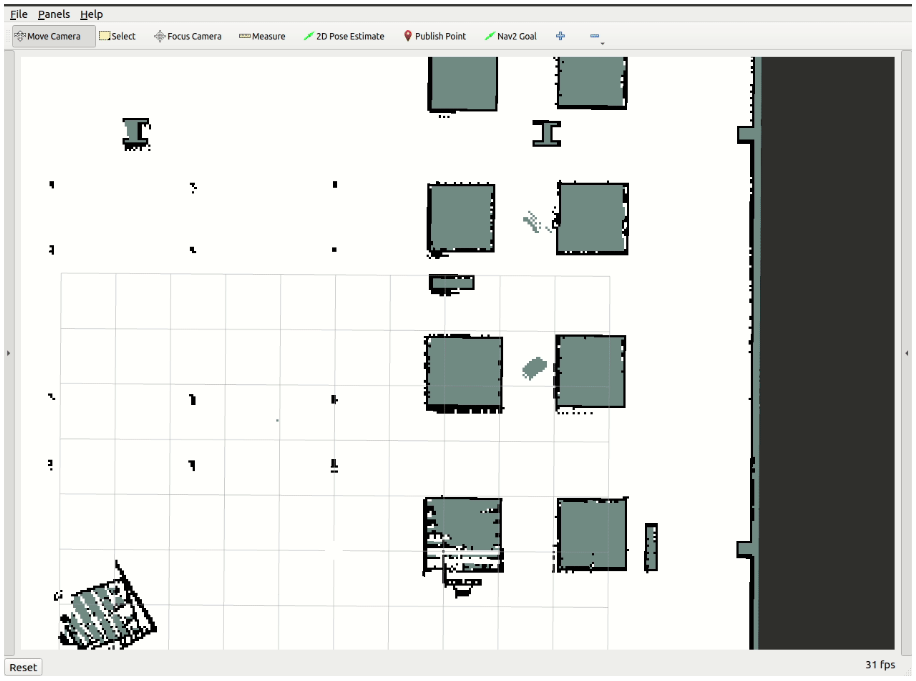
Now we can undock the robot and begin navigating through each point. Once the robot has reached the final pose, it will then return to the dock.

```
navigator.undock()

navigator.startThroughPoses(goal_pose)

navigator.dock()
```

Watch navigation progress in Rviz

You can visualise the navigation process in Rviz by calling:

```
ros2 launch turtlebot4_viz view_robot.launch.py
```



Navigate through a set of poses

Follow Waypoints

This example demonstrates how to follow waypoints. The Nav2 stack is given a set of waypoints on the map and creates a path that goes through each waypoint in order until the last waypoint is reached. The robot then attempts to drive along the path. The difference between this example and Navigating Through Poses is that when following waypoints the robot will plan to reach each waypoint individually, rather than planning to reach the last pose by driving through the other poses. This example is demonstrated in the depot world of the TurtleBot 4 simulation.

To run this example, start the Ignition simulation:

```
ros2 launch turtlebot4_ignition_bringup ignition.launch.py nav:=true slam:=off localization:=true
```

Once the simulation has started, open another terminal and run:

```
ros2 run turtlebot4_python_tutorials follow_waypoints
```

Code breakdown

The source code for this example is available here.

Lets take a look at the main function.

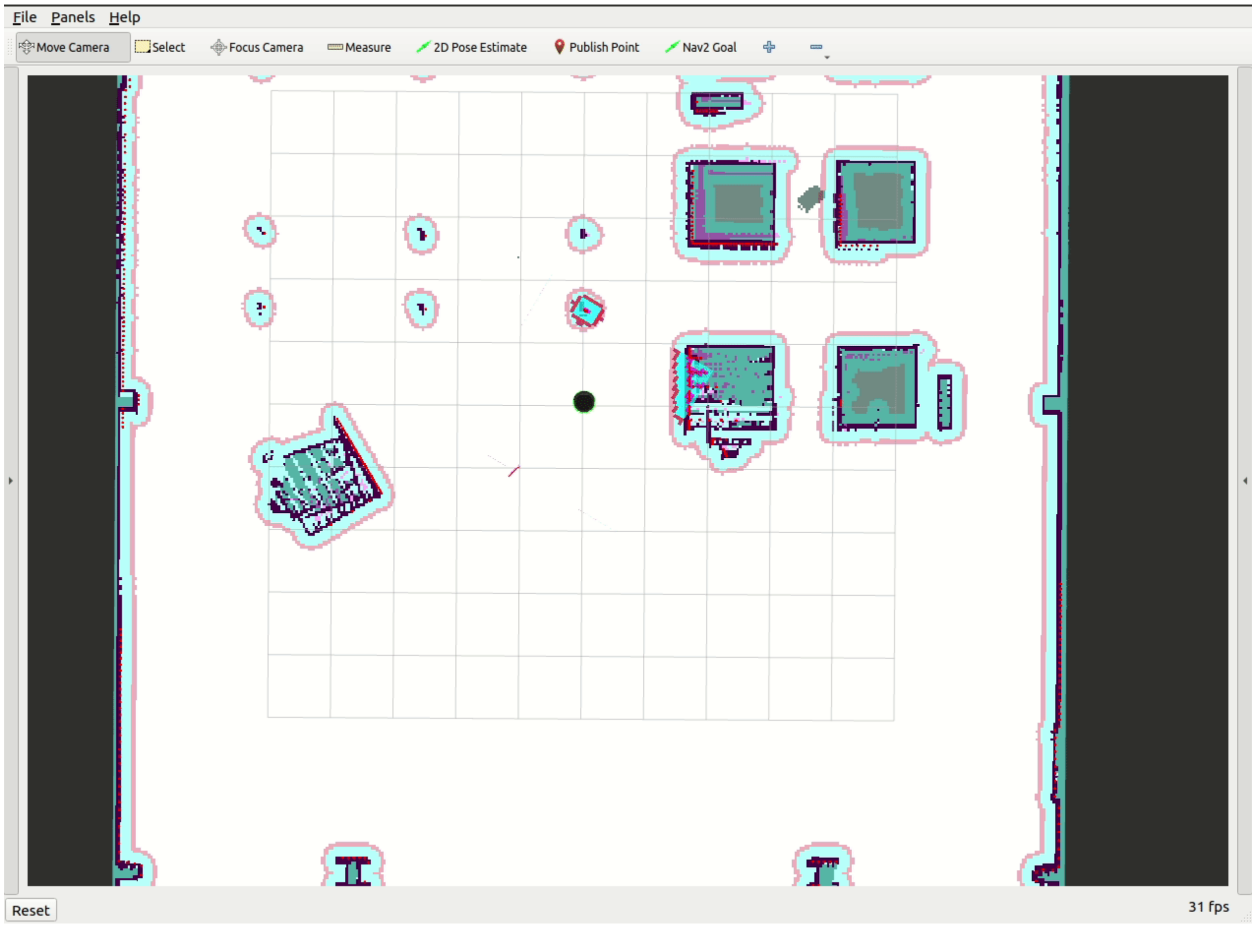
```
def main():
    rclpy.init()

    navigator = TurtleBot4Navigator()

    # Start on dock
    if not navigator.getDockedStatus():
        navigator.info('Docking before intialising pose')
        navigator.dock()

    # Set initial pose
    initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
    navigator.setInitialPose(initial_pose)

    # Wait for Nav2
    navigator.waitUntilNav2Active()

    # Set goal poses
    goal_pose = []
    goal_pose.append(navigator.getPoseStamped([-3.3, 5.9], TurtleBot4Directions.NORTH))
    goal_pose.append(navigator.getPoseStamped([2.1, 6.3], TurtleBot4Directions.EAST))
    goal_pose.append(navigator.getPoseStamped([2.0, 1.0], TurtleBot4Directions.SOUTH))
    goal_pose.append(navigator.getPoseStamped([-1.0, 0.0], TurtleBot4Directions.NORTH))

    # Undock
    navigator.undock()

    # Follow Waypoints
    navigator.startFollowWaypoints(goal_pose)

    # Finished navigating, dock
    navigator.dock()

    rclpy.shutdown()
```

This example is very similar to Navigate Through Poses. The difference is that we are using different poses as our waypoints, and that we use the startFollowWaypoints method to perform our navigation behaviour.

Watch navigation progress in Rviz

You can visualise the navigation process in Rviz by calling:

```
ros2 launch turtlebot4_viz view_robot.launch.py
```



Follow a set of Waypoints

Create Path

This example demonstrates how to create a navigation path in Rviz during runtime. It uses the [2D Pose Estimate] tool to pass the TurtleBot 4 Navigator a set of poses. Then we use the [Follow Waypoints] behaviour to follow those poses. This example was run on a physical TurtleBot 4.

To run this example, start nav bringup on your PC or on the Raspberry Pi:

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py slam:=off localization:=true
map:=office.yaml
```

Replace office.yaml with the map of your environment.

Once the navigation has started, open another terminal and run:

```
ros2 run turtlebot4_python_tutorials create_path
```

On your PC you will need to start Rviz:

```
ros2 launch turtlebot4_viz view_robot.launch.py
```

Code breakdown

The source code for this example is available here.

Lets take a look at the main function.

```
def main():
    rclpy.init()

    navigator = TurtleBot4Navigator()

    # Set goal poses
    goal_pose = navigator.createPath()

    if len(goal_pose) == 0:
        navigator.error('No poses were given, exiting.')
        exit(0)

    # Start on dock
    if not navigator.getDockedStatus():
        navigator.info('Docking before intialising pose')
        navigator.dock()

    # Set initial pose
    initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
    navigator.clearAllCostmaps()
    navigator.setInitialPose(initial_pose)

    # Wait for Nav2
    navigator.waitUntilNav2Active()

    # Undock
    navigator.undock()

    # Navigate through poses
    navigator.startFollowWaypoints(goal_pose)
    # Finished navigating, dock
    navigator.dock()

    rclpy.shutdown()
```

This example begins the same as the others by initialising the TurtleBot 4 Navigator.

Create your path

After initialisation, the user is prompted to create their path by using the 2D Pose Estimate tool. You must set at least one pose. Once all of the poses have been set, the user can press CTRL + C to stop creating the path and begin navigating.

```
goal_pose = navigator.createPath()

if len(goal_pose) == 0:
    navigator.error('No poses were given, exiting.')
    exit(0)
```

Set initial pose and clear costmaps

Next we set the initial pose and clear all costmaps. We clear costmaps because the 2D Pose Estimate tool is subscribed to by the Nav2 stack, and every time we use it Nav2 assumes that the robot is in that position, when it is not. Clearing the costmaps will get rid of any false costmaps that may have spawned when creating the path.

```
if not navigator.getDockedStatus():
    navigator.info('Docking before intialising pose')
    navigator.dock()

initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
navigator.clearAllCostmaps()
navigator.setInitialPose(initial_pose)

navigator.waitUntilNav2Active()
```

We also wait for Nav2 to be active before continuing.

Follow the path

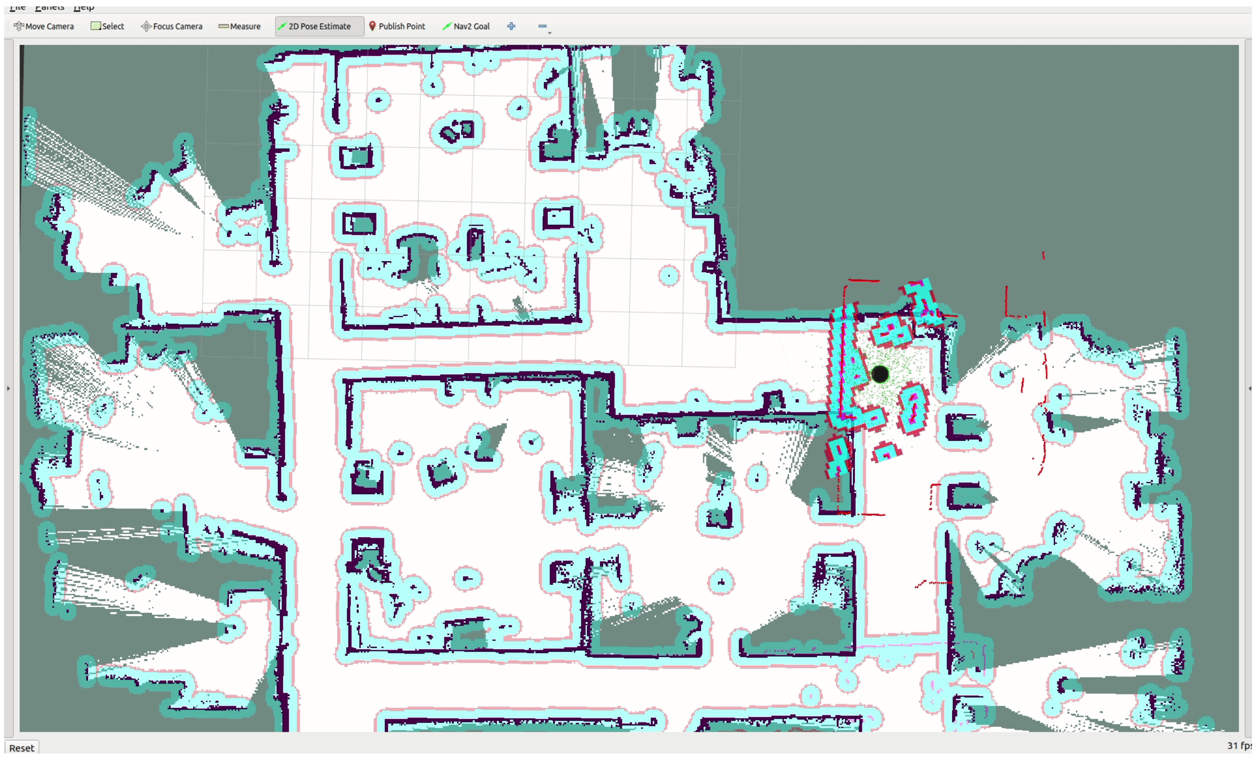
Now we can undock and follow the created path. In this example we use the Follow Waypoints behaviour, but this can easily be replaced with Navigate Through Poses.

```
navigator.undock()

navigator.startFollowWaypoints(goal_pose)

navigator.dock()
```

We finish the example by docking the robot. This assumes that the last pose in the created path is near the dock. If it is not, you can remove this action.

Creating a path with Rviz

Running this example will look something like this:



Creating a path and following it

Note

As the path is created, you will see the robot being placed at the position you click on. This is normal and gets cleared up when the initial pose is set by the TurtleBot 4 Navigator.

Troubleshooting

Diagnostics

The TurtleBot 4 and TurtleBot 4 both run diagnostics updater and aggregator nodes by default. The updater records diagnostics data and the aggregator formats it so that it can be used with rqt_robot_monitor. This is a tool that can be used to monitor various robot topics to ensure that they are publishing data at the expected frequency.

To check that diagnostics are running properly, call

ros2 node list

You should see a node called turtlebot4_diagnostics. Additionally, calling

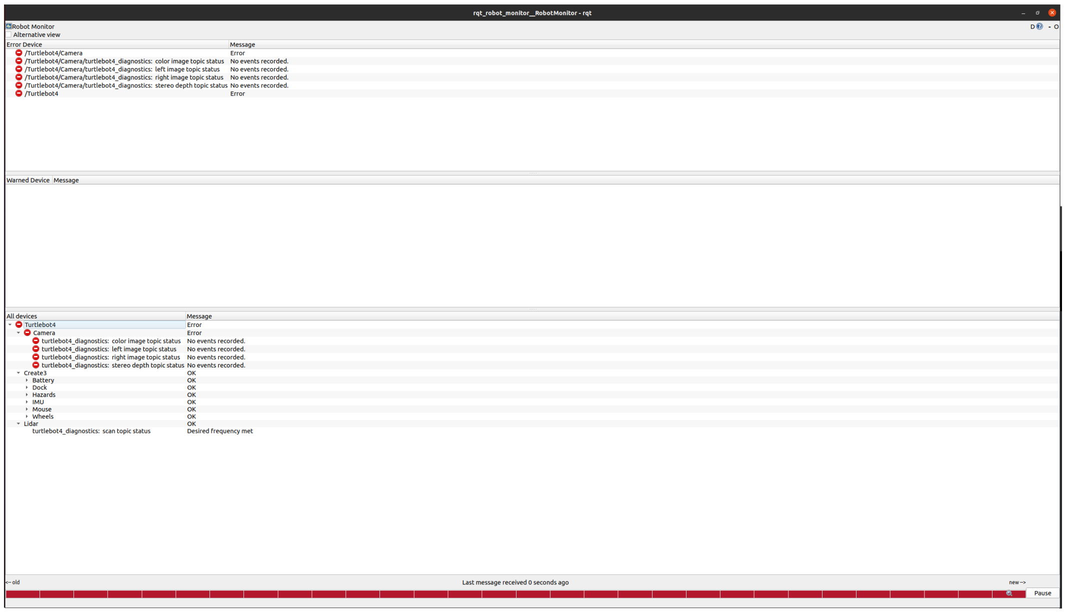
ros2 topic list

should list topics such as /diagnostics, /diagnostics_agg, and /diagnostics_toplevel_state. If diagnostics are not running, you can manually run them by calling

ros2 launch turtlebot4_diagnostics diagnostics.launch.py

Once diagnostics are running, you can view them with rqt_robot_monitor. Ensure that turtlebot4_desktop is installed on your PC, then call

ros2 launch turtlebot4_viz view_diagnostics.launch.py



rqt_robot_monitor__RobotMonitor - rqt

Robot Monitor

Alternative view

Error Device	Message
/Turtlebot4/Camera	Error
/Turtlebot4/Camera/turtlebot4_diagnostics: color image topic status	No events recorded.
/Turtlebot4/Camera/turtlebot4_diagnostics: left image topic status	No events recorded.
/Turtlebot4/Camera/turtlebot4_diagnostics: right image topic status	No events recorded.
/Turtlebot4/Camera/turtlebot4_diagnostics: stereo depth topic status	No events recorded.
/Turtlebot4	Error

Warned Device	Message

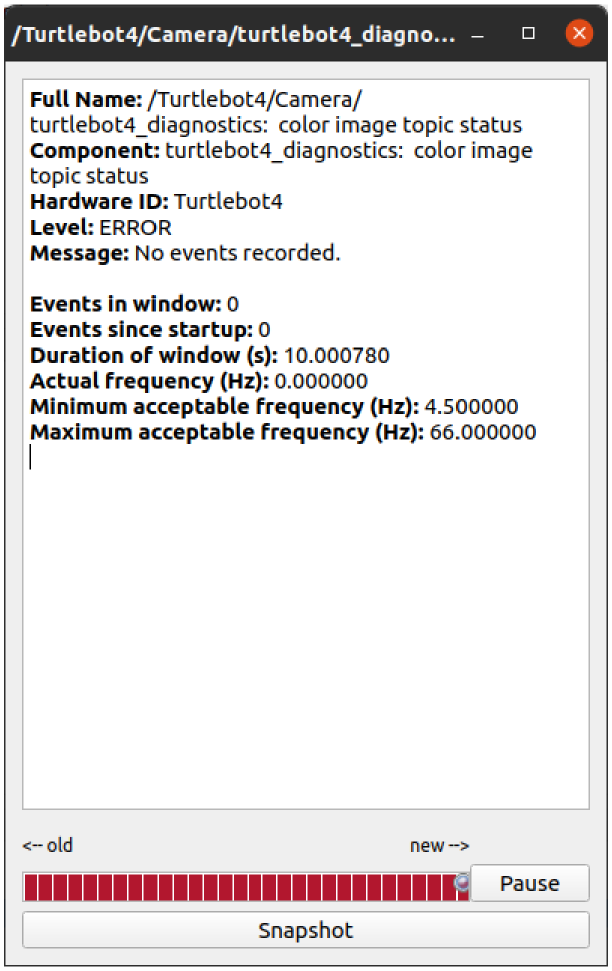
All devices	Message
Turtlebot4	Error
Camera	Error
turtlebot4_diagnostics: color image topic status	No events recorded.
turtlebot4_diagnostics: left image topic status	No events recorded.
turtlebot4_diagnostics: right image topic status	No events recorded.
turtlebot4_diagnostics: stereo depth topic status	No events recorded.
Create3	OK
Battery	OK
Dock	OK
Hazards	OK
IMU	OK
Mouse	OK
Wheels	OK
Lidar	OK
turtlebot4_diagnostics: scan topic status	Desired frequency met

<-- old   Last message received 0 seconds ago   new -->

rqt_robot_monitor with TurtleBot 4 diagnostics

The monitor will display any errors in the first window, any warnings in the second window, and a summary of all topics in the "All devices" section at the bottom. Each topic has a status level of OK, WARNING, ERROR, or STALE. There is also a more detailed message included as well. You can click on each topic to view more information.

In this example, the OAK-D node is not running, so the camera topics are not being published.


/Turtlebot4/Camera/turtlebot4_diagno...

Full Name: /Turtlebot4/Camera/turtlebot4_diagnostics: color image topic status
Component: turtlebot4_diagnostics: color image topic status
Hardware ID: Turtlebot4
Level: ERROR
Message: No events recorded.

Events in window: 0
Events since startup: 0
Duration of window (s): 10.000780
Actual frequency (Hz): 0.000000
Minimum acceptable frequency (Hz): 4.500000
Maximum acceptable frequency (Hz): 66.000000

Color camera diagnostics

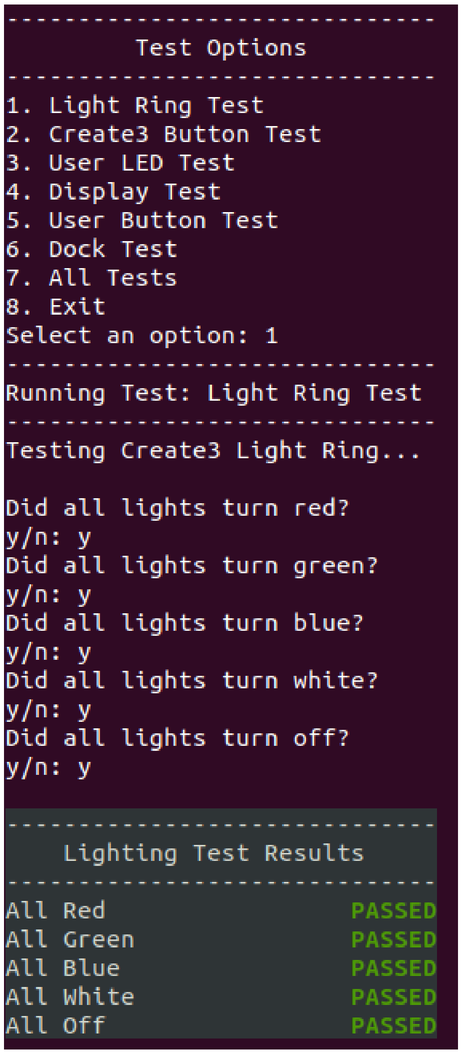
ROS2 Tests

Both TurtleBot 4 models have the turtlebot4_tests package installed by default. This package provides some tests that can be run from CLI to test basic system functions.

Each test uses a ROS2 topic, action, or service to perform the action. To run Create® 3 tests, the Create® 3 must be connected to the Raspberry Pi over either WiFi or USB-C.

To run the tests, call

ros2 run turtlebot4_tests ros_tests



```
------------------------------
         Test Options
------------------------------
1. Light Ring Test
2. Create3 Button Test
3. User LED Test
4. Display Test
5. User Button Test
6. Dock Test
7. All Tests
8. Exit
Select an option: 1
------------------------------
Running Test: Light Ring Test
------------------------------
Testing Create3 Light Ring...

Did all lights turn red?
y/n: y
Did all lights turn green?
y/n: y
Did all lights turn blue?
y/n: y
Did all lights turn white?
y/n: y
Did all lights turn off?
y/n: y

------------------------------
    Lighting Test Results
------------------------------
All Red                 PASSED
All Green               PASSED
All Blue                PASSED
All White               PASSED
All Off                 PASSED
```

Running the Light Ring test

Test results are saved to ~/turtlebot4_test_results/Y_m_d-H_M_S where Y_m_d-H_M_S is the date and time of the test. A rosbag is also recorded for the duration of the test and saved to the same location.

FAQ

Common issues with the Raspberry Pi 4B

1. Access point is not visible

If your Raspberry Pi is in AP mode, a Turtlebot4 WiFi network should become visible about 30 seconds after the robot has been powered on. If it does not, try the following.

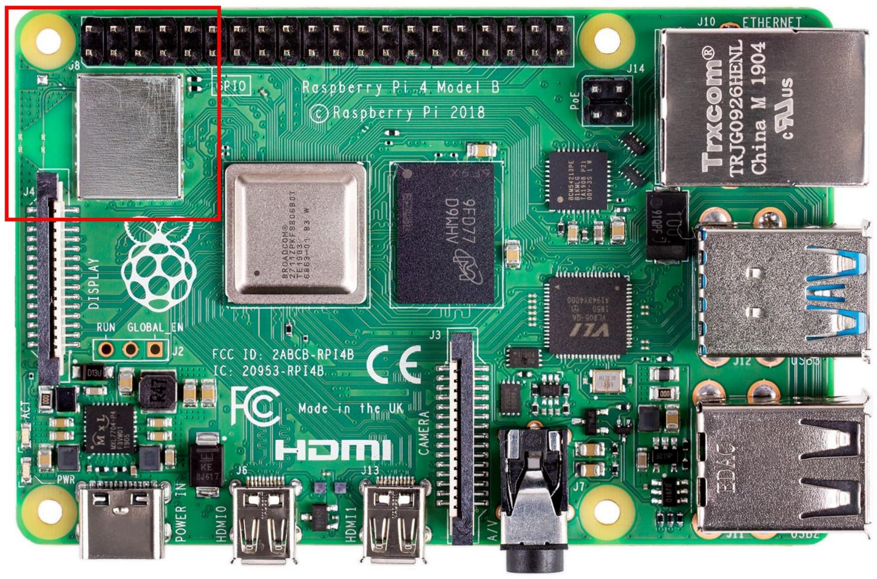
Check that the Raspberry Pi is powered

The Raspberry Pi has a Power LED near the USB-C port. Make sure it is illuminated.

If the LED is not on, then the Raspberry Pi is not powered. Check the USB-C connection and make sure the Create® 3 power adapter board is inserted fully.

Check for obstructions

If the Pi is on but you cannot see the access point, make sure that any wires in the robot are not obstructing the WiFi module of the Raspberry Pi. This includes the ribbon cable connecting the RPi to the UI PCBA on the TurtleBot 4, and the wires powering the fans on both models.



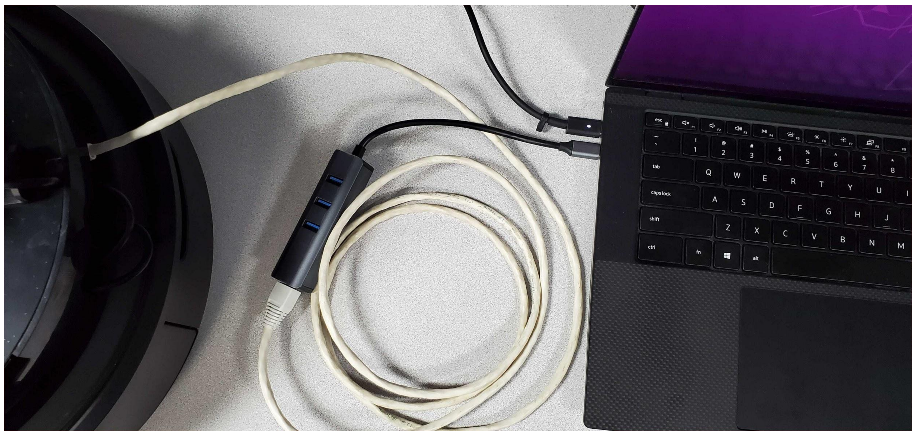
Raspberry Pi 4B WiFi module and antenna

Restart the robot

If the WiFi module is unobstructed, try restarting the robot. Take the robot off of its dock and press and hold the Power button on the Create® 3 until it is off. Wait a few seconds and place the robot back on its dock.

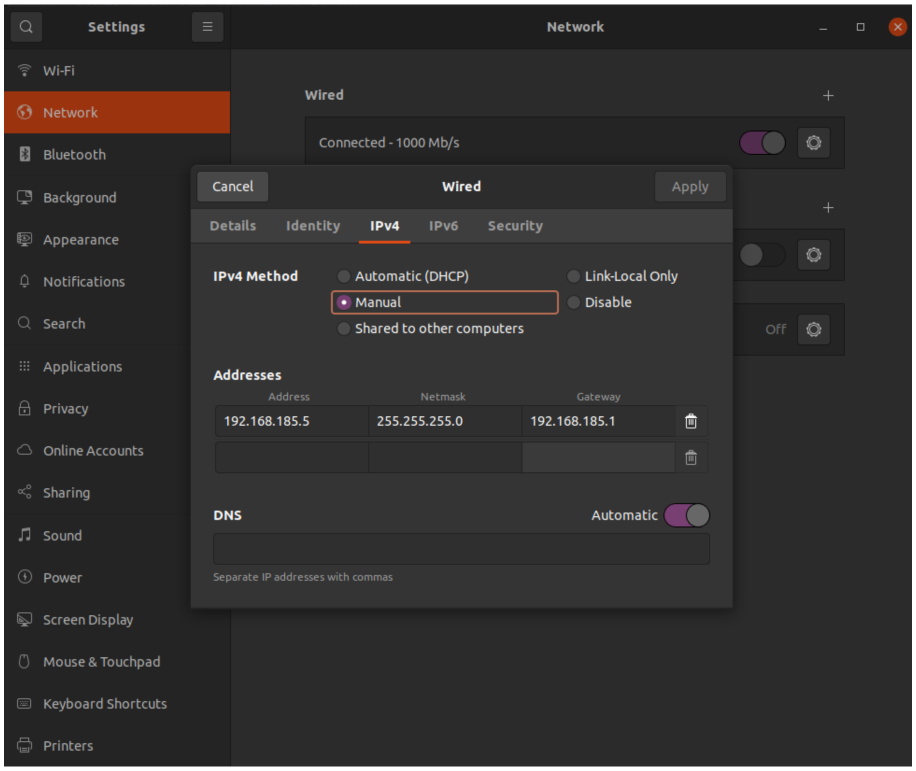
Access the RPi over Ethernet

If you are still unable to see the Turtlebot4 access point, you can connect directly to the RPi using an ethernet cable. You may need a USB to Ethernet adapter for your PC.


Connecting to the TurtleBot 4 over Ethernet

The Raspberry Pi uses a static IP address for the ethernet interface, 192.168.185.3. You will need to configure your wired connection to use the same subnet:

- Go to your wired connection settings.
- Set your IPv4 Method to Manual and set your static IP. The IP address cannot be the same as the Raspberry Pi.



Wired

Connected - 1000 Mb/s

IPv4 Method: Automatic (DHCP) [✓ Manual] Shared to other computers Link-Local Only Disable

Addresses

Address	Netmask	Gateway
192.168.185.5	255.255.255.0	192.168.185.1

DNS — Automatic

Separate IP addresses with commas

Configure your PC's wired IP

- Click 'Apply'

You can now go to your terminal and SSH into the robot by typing:

ssh ubuntu@192.168.185.3

1. Waiting to connect to bluetoothd…

This issue is usually a result of the bluetooth service being stopped.

To start the service again, run sudo systemctl start bluetooth.

2. No default controller available

This error occurs if you are attempting to connect a bluetooth device to the Raspberry Pi with sudo bluetoothctl and the hciuart service throws errors.

To fix this, call sudo systemctl disable hciuart and then reboot the Pi with sudo reboot.

Once the Pi has restarted, call sudo systemctl restart hciuart. Now you can run sudo bluetoothctl again and the bluetooth controller should be found.

Common issues with the user PC

1. ros2: command not found

Make sure you have sourced ROS2 galactic:

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

If you are building packages from source, you will also want to source the workspace:

```
source /path/to/ws/install/setup.bash
```

2. Create® 3 topics are not visible

First, check that the Create® 3 is connected to your WiFi network. You should be able to access the Create® 3 portal by entering the Create® 3 IP address in a browser. For information on how to connect the Create® 3 to WiFi, check the quick start guide.

If it is connected to WiFi, check if you can see Create® 3 topics on the Raspberry Pi.

If topics are visible on the Raspberry Pi, ensure that your PC has the following configuration set for CycloneDDS:

```
<CycloneDDS>
  <Domain>
    <General>
      <DontRoute>true</DontRoute>
    </General>
  </Domain>
</CycloneDDS>
```

To set this configuration automatically, add the following line to your ~/.bashrc file.

```
export
CYCLONEDDS_URI='<CycloneDDS><Domain><General><DontRoute>true</></></></>'
```

If you have set a ROS_DOMAIN_ID for the Create® 3, your terminal will have to have the same ID. You can set the ID by using this command:

```
export ROS_DOMAIN_ID=#
```

Replace # with the ID.

If topics are not visible on the Raspberry Pi, you may need to restart the Create® 3 application through the portal, or reboot the robot.