

# Digital.auto Starter Kit

## Software-Manual

### Table of Contents

1	Introduction.....	2
2	System Overview.....	2
3	Electronic Control Units .....	2
3.1	Raspberry Pi 5 .....	2
3.2	Arduino Portenta X8 .....	2
3.3	Arduino Portenta H7 .....	2
3.4	Arduino Uno R4 Minima .....	2
3.5	Arduino Uno R4 WiFi .....	3
4	Software Components.....	4
4.1	Mosquitto MQTT Server.....	4
4.2	Node-RED with Dashboard.....	4
4.3	CPP Haptic Force Controller.....	4
4.4	Kuksa Data Broker .....	4
4.5	Kuksa MQTT Gateway .....	4
4.6	UDP-CAN Gateway .....	4
4.7	CAN Display Driver.....	4
4.8	CAN Motor Driver .....	5
5	Data Communication Architecture .....	6
5.1	MQTT Communication .....	6
5.2	Vehicle API Communication .....	6
5.3	UDP Communication .....	6
5.4	CAN Communication.....	6
6	Installation Guide.....	7
6.1	Setting Up Raspberry Pi 5 .....	7
6.2	Installation Mosquitto MQTT Broker on Raspberry Pi .....	10
6.3	Installation Node-Red on Raspberry Pi.....	12
6.4	Setting Up Docker on Raspberry Pi 5 .....	18
6.5	Setting Up Portenta X8.....	20
6.6	Setting up kuksa databroker on portenta X8 .....	25
6.7	Setting up Kuksa MQTT Gateway .....	26
6.8	Installation of the Arduino IDE on the Raspberry Pi 5 .....	28
6.9	Setting up CAN Display Driver on Arduino UNO R4 WiFi.....	32
6.10	Setting up CAN Motor Driver on Arduino UNO R4 Minima.....	34
6.11	Installation Zephyr on the Raspberry Pi 5.....	36
6.12	Setting up UDP CAN Gateway on Arduino Portenta H7.....	39
6.13	Setting up CPP Haptic Force Controller .....	42

# 1 Introduction

The **Digital.auto Starter Kit** is a development platform designed for automotive and industrial automation applications. This manual provides software instructions for setting up and operating the system, detailing each component's role and communication protocols.

## 2 System Overview

The system is structured around a local network with an Ethernet switch connecting various hardware components, each playing a crucial role in data processing, motor control, and communication.

## 3 Electronic Control Units

### 3.1 Raspberry Pi 5

- OS: Ubuntu 24.04 LTS Desktop
- Runs Mosquitto MQTT server and Node-RED dashboard
- Acts as a cloud computer emulation
- Displays data and controls system parameters

### 3.2 Arduino Portenta X8

- OS: Yocto Linux
- Runs the main application: CPP Haptic Force Controller
- Communicates via MQTT with Raspberry Pi
- Exchanges data with MQTT and Vehicle API through Kuksa Data Broker
- Runs Kuksa Data Broker to facilitate data exchange via Vehicle API
- Includes a Python Gateway translating MQTT messages into Vehicle API

### 3.3 Arduino Portenta H7

- Functions as UDP-to-CAN gateway
- Receives motor torque via UDP tunnel from Haptic Force Controller
- Sends motor torque messages to CAN
- Receives actual motor position from CAN and transmits it back via UDP

### 3.4 Arduino Uno R4 Minima

- Controls the motor using Field-Oriented Control (FOC)

- Receives motor angle from SPI magnetic sensor and sends it to the gateway over CAN
- Receives motor torque from CAN and apply this with FOC

### **3.5      Arduino Uno R4 WiFi**

- Connected to the left gateway
- Used to measures roundtrip latency
- Displays latency and jitter on the OLED display

## **4 Software Components**

### **4.1 Mosquitto MQTT Server**

- Runs on Raspberry Pi
- Manages message queues for system communication

### **4.2 Node-RED with Dashboard**

- Provides a web-based user interface
- Allows control and monitoring of system parameters

### **4.3 CPP Haptic Force Controller**

- Main motor control application on Portenta X8
- Calculate desired motor position and motor torque for left and right motor (haptic feedback and position remote control)
- Sends motor torque data over UDP to Portenta H7 gateway
- Send motor position to Vehicle API Kuksa and mqtt
- Receives motor position updates from the gateways
- Measure statistics roundtrip latency, jitter and wall time jitter

### **4.4 Kuksa Data Broker**

- Facilitates data exchange between middleware applications
- Uses Vehicle API for communication

### **4.5 Kuksa MQTT Gateway**

- Converts MQTT messages into Vehicle API data format

### **4.6 UDP-CAN Gateway**

- Runs on Portenta H7
- Manages motor torque, position and statistic data transfer UDP-CAN

### **4.7 CAN Display Driver**

- Runs on Arduino UNO R4 WiFi
- Outputs the real time statistics on the i2c OLED display
- Sends the pong message for roundtrip latency measurement

## **4.8 CAN Motor Driver**

- Runs on Arduino UNO R4 Minima
- Receives Torque Commands via CAN Bus
- Controls the BLDC Motor Using FOC
- Sends Motor Angle Feedback Over CAN

## **5 Data Communication Architecture**

### **5.1 MQTT Communication**

- Used for data transfer between Portenta X8 and Raspberry Pi
- Node-RED interacts with MQTT broker to send control parameters and receive statistic data

### **5.2 Vehicle API Communication**

- Kuksa Data Broker facilitates API-based communication
- Python Gateway translates MQTT messages into Vehicle API format

### **5.3 UDP Communication**

- Torque and position data exchanged between Portenta X8 and Portenta H7
- UDP server on Portenta H7 manages these transactions
- Roundtrip ping and pong messages

### **5.4 CAN Communication**

- Torque commands sent from Portenta H7 to Arduino Uno R4 Minima via CAN
- Motor angle feedback sent back via CAN
- Statistic data
- Roundtrip ping and pong messages

## 6 Installation Guide

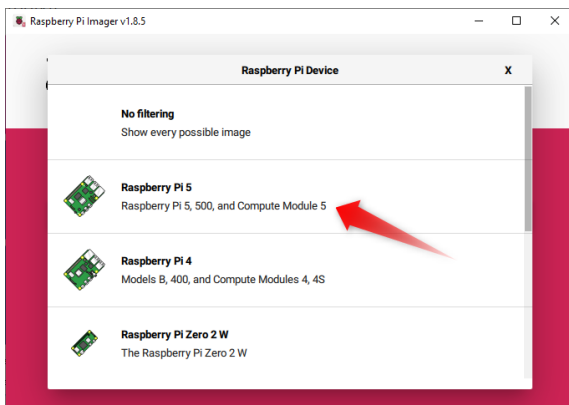
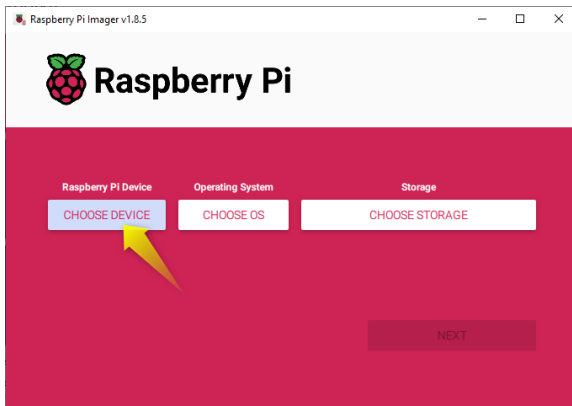
### 6.1 Setting Up Raspberry Pi 5

#### Requirements:

- PC with Windows/macOS/Ubuntu
- SD-Card Reader
- Micro SD-Card (64GB or bigger)
- Router with DHCP LAN (e.g., 192.168.88.1) and Internet Access

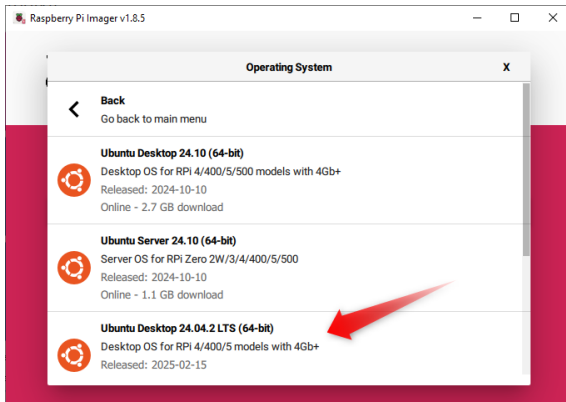
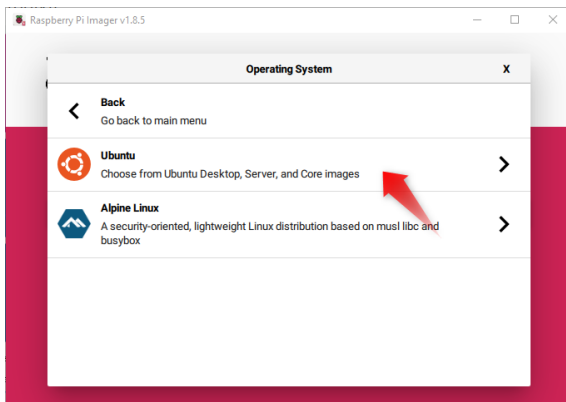
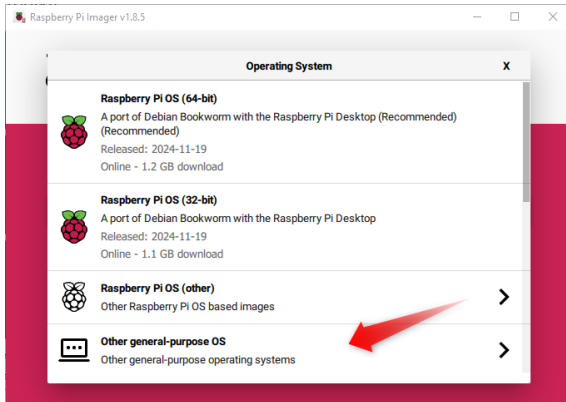
#### Steps:

- 1) Download and install Raspberry Pi Imager from:  
<https://www.raspberrypi.com/software/>
- 2) Insert SD-Card into your PC SD-card reader.
- 3) Run Raspberry Pi Imager, click CHOOSE DEVICE, and select “Raspberry Pi 5.”



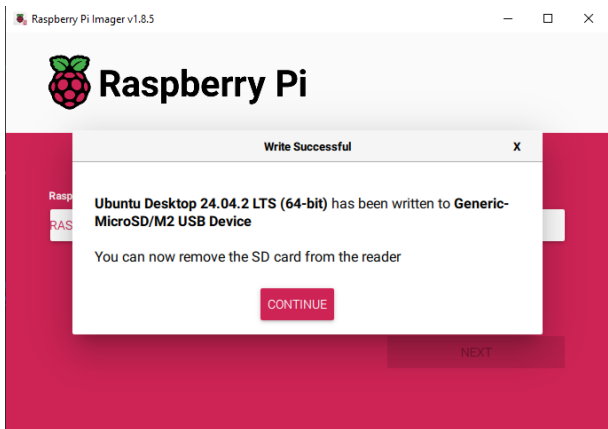
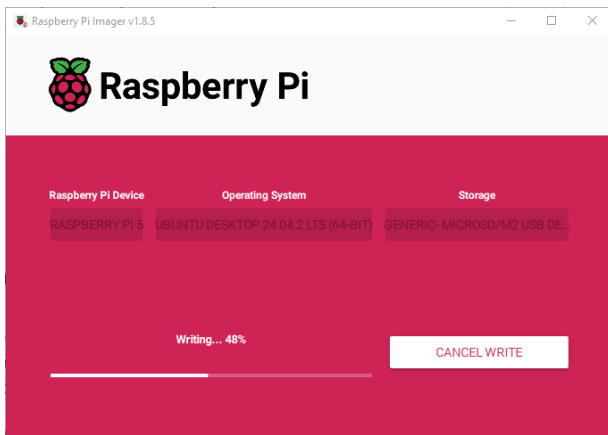
- 4) Click CHOOSE OS and select:

Other general-purpose OS → Ubuntu → Ubuntu Desktop 24.04.x LTS (64-bit)



- 5) Click CHOOSE STORAGE and select your micro SD card storage.
- 6) Click NEXT, then confirm with **YES** to begin installation.





- 7) Once flashing is complete, insert the SD-Card into the Raspberry Pi.
- 8) Connect Raspberry Pi to a display, mouse, keyboard, and router. Plug in the power supply.
- 9) Complete the Ubuntu 24.04 installation by selecting language, location, computer name, and password.

## 6.2 Installation Mosquitto MQTT Broker on Raspberry Pi

Steps:

- 1) Run command to install Mosquitto MQTT Broker:

```
sudo apt update -y  
  
sudo apt install mosquitto -y
```

- 2) Set up a username and password for MQTT Broker

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd app1
```

Enter a password: **app1password** for the user name **app1** at the prompt and press **ENTER**.

Re-enter the password and press **ENTER**.

- 3) Update Mosquitto Configuration File

```
echo -e "allow_anonymous false\npassword_file  
/etc/mosquitto/passwd\nlistener 1883 0.0.0.0" | sudo tee  
/etc/mosquitto/conf.d/default.conf > /dev/null  
  
sudo systemctl restart mosquitto
```

- 4) Test MQTT Broker with MQTT Client

Install MQTT Client

```
sudo apt-get install mosquitto-clients
```

Enter:

```
mosquitto_sub -t "test" -u "app1" -P "app1password"
```

Open a second terminal window and enter:

```
mosquitto_pub -t "test" -u "app1" -P "app1password" -m "Test message"
```

Select the first terminal window. View the message: message from mosquitto\_pub client in the first Terminal window:

```
bluebox@bluebox:~$ mosquitto_sub -t "test" -u "app1" -P "app1password"  
Test message from mosquitto_pub client
```

Press **Ctrl+C** to exit the subscription.

## 6.3 Installation Node-Red on Raspberry Pi

Steps:

1) Install Node.js:

```
sudo apt-get update

sudo apt-get install -y ca-certificates curl gnupg

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-
repo.gpg.key | sudo gpg --dearmor -o
/etc/apt/keyrings/nodesource.gpg

NODE_MAJOR=20

echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
https://deb.nodesource.com/node_${NODE_MAJOR}.x nodistro main"
| sudo tee /etc/apt/sources.list.d/nodesource.list

sudo apt-get update

sudo apt-get install nodejs -y
```

After installation, check the installed versions of Node.js and npm to confirm they are installed correctly:

```
node -v

npm -v
```

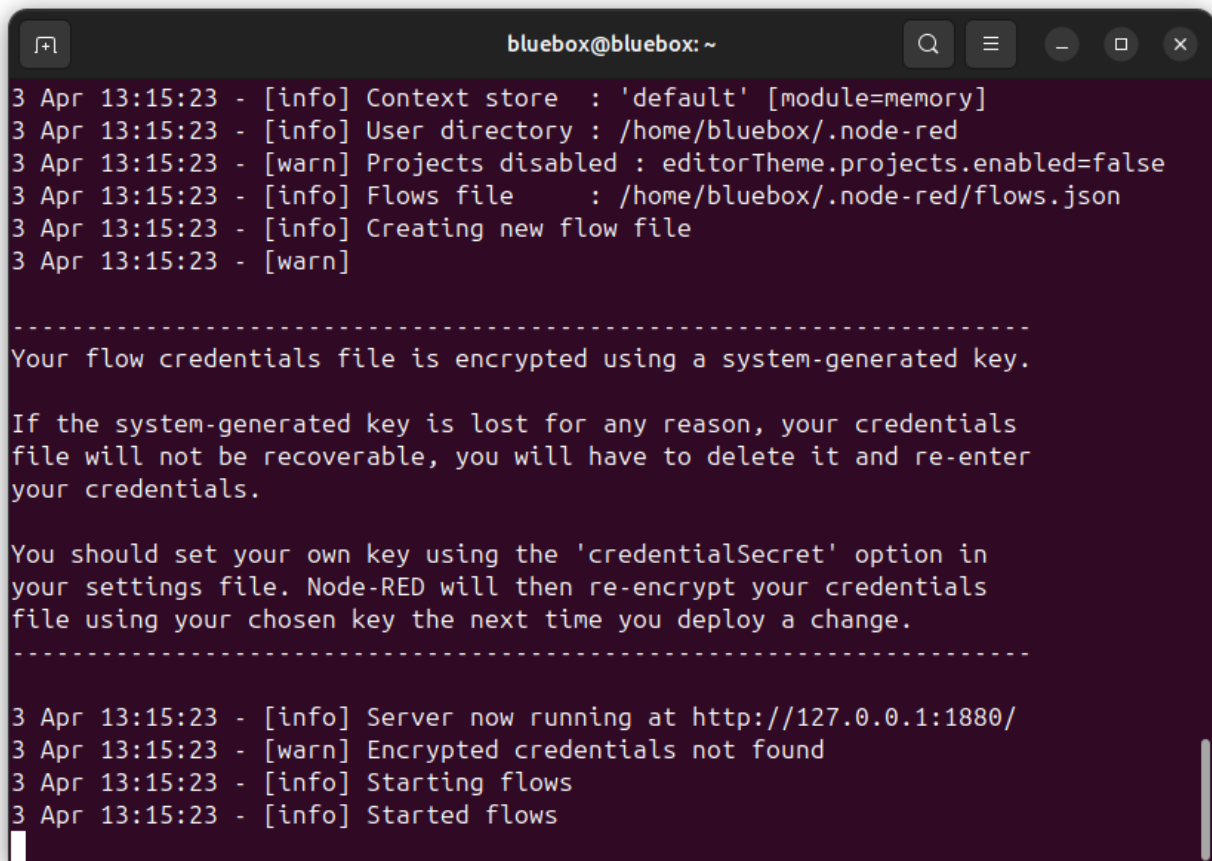
```
bluebox@bluebox:~$ node -v
v20.19.0
npm -v
10.8.2
```

2) Install Node-RED globally using npm:

```
sudo npm install -g --unsafe-perm node-red
```

Once the installation is complete, start Node-RED by running:

```
node-red
```

A terminal window titled 'bluebox@bluebox: ~' showing the output of the 'node-red' command. The logs include information about the context store, user directory, projects status, flows file, and flow credentials. It also displays a warning about encrypted credentials and a message that the server is now running at http://127.0.0.1:1880/.

```
3 Apr 13:15:23 - [info] Context store : 'default' [module=memory]
3 Apr 13:15:23 - [info] User directory : /home/bluebox/.node-red
3 Apr 13:15:23 - [warn] Projects disabled : editorTheme.projects.enabled=false
3 Apr 13:15:23 - [info] Flows file      : /home/bluebox/.node-red/flows.json
3 Apr 13:15:23 - [info] Creating new flow file
3 Apr 13:15:23 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

3 Apr 13:15:23 - [info] Server now running at http://127.0.0.1:1880/
3 Apr 13:15:23 - [warn] Encrypted credentials not found
3 Apr 13:15:23 - [info] Starting flows
3 Apr 13:15:23 - [info] Started flows
```

Press **Ctrl+C** to exit

3) Set up a pm2 service to manage Node-RED's automatic startup:

Install PM2:

```
sudo npm install -g pm2
```

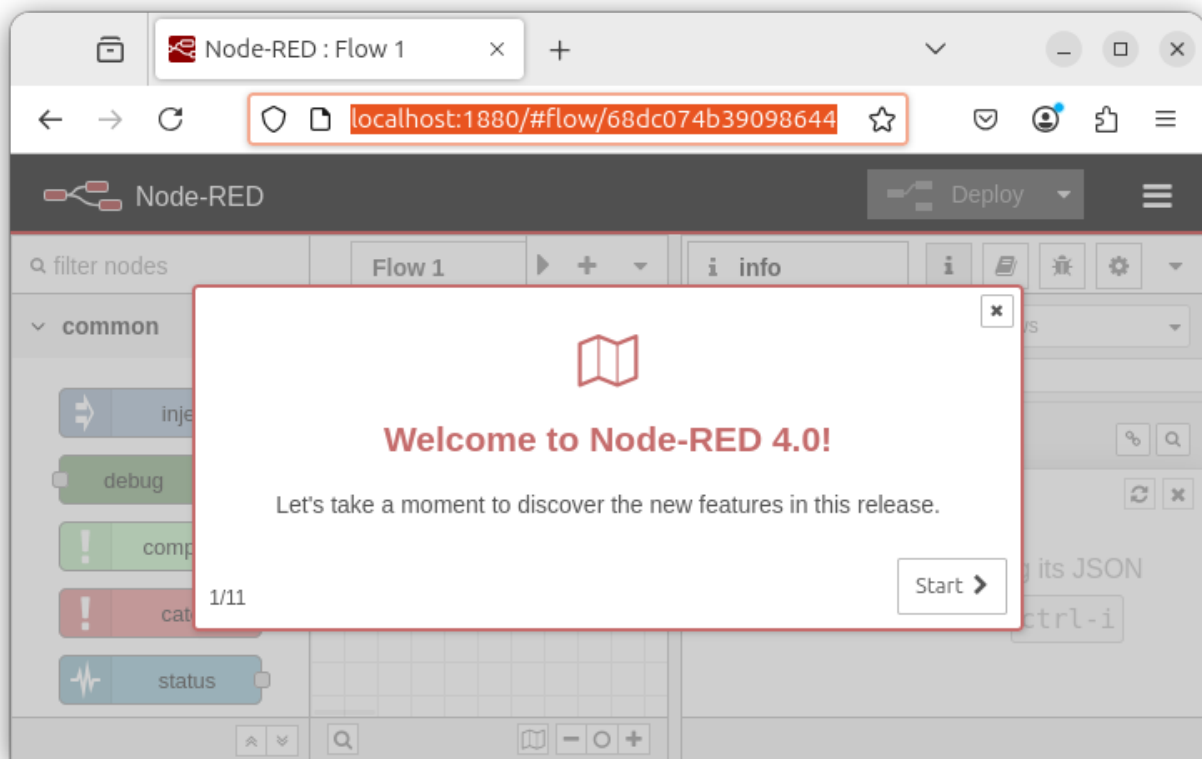
Start Node-RED with PM2:

```
pm2 start /usr/bin/node-red -- -v
```

Save the current process list:

```
pm2 save
```

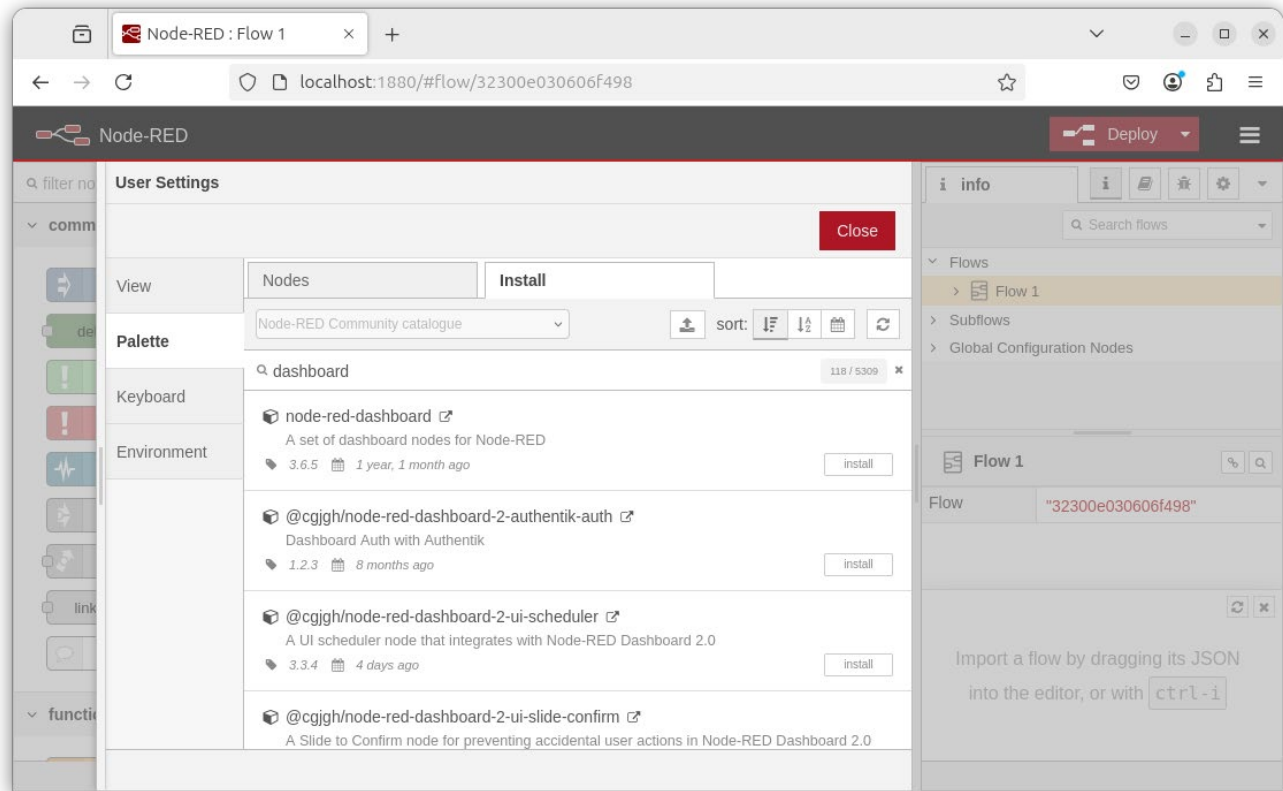
- 4) Open a web browser and navigate to <http://localhost:1880> to access the Node-RED flow editor.



- 5) Install the Node-RED Dashboard

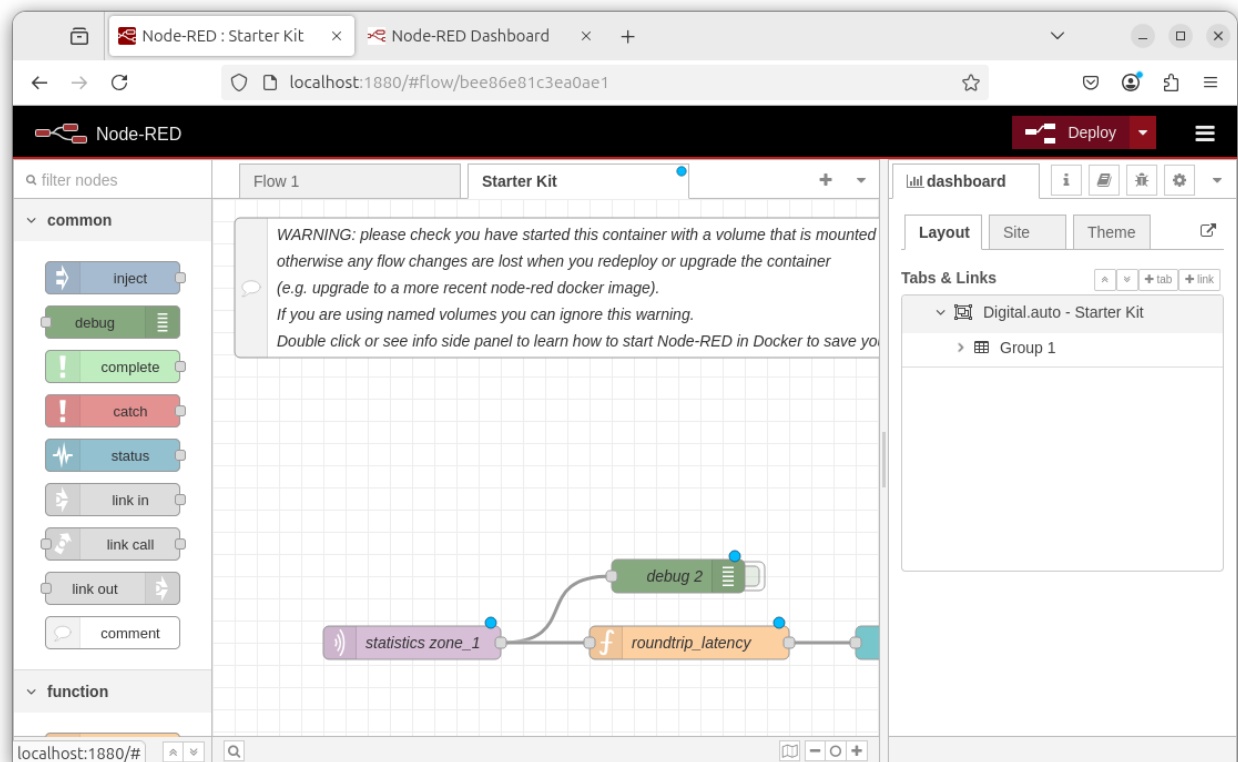
- Click on the menu (≡) > Manage palette.
- Go to the Install tab.

- Search for node-red-dashboard.
- Click Install.



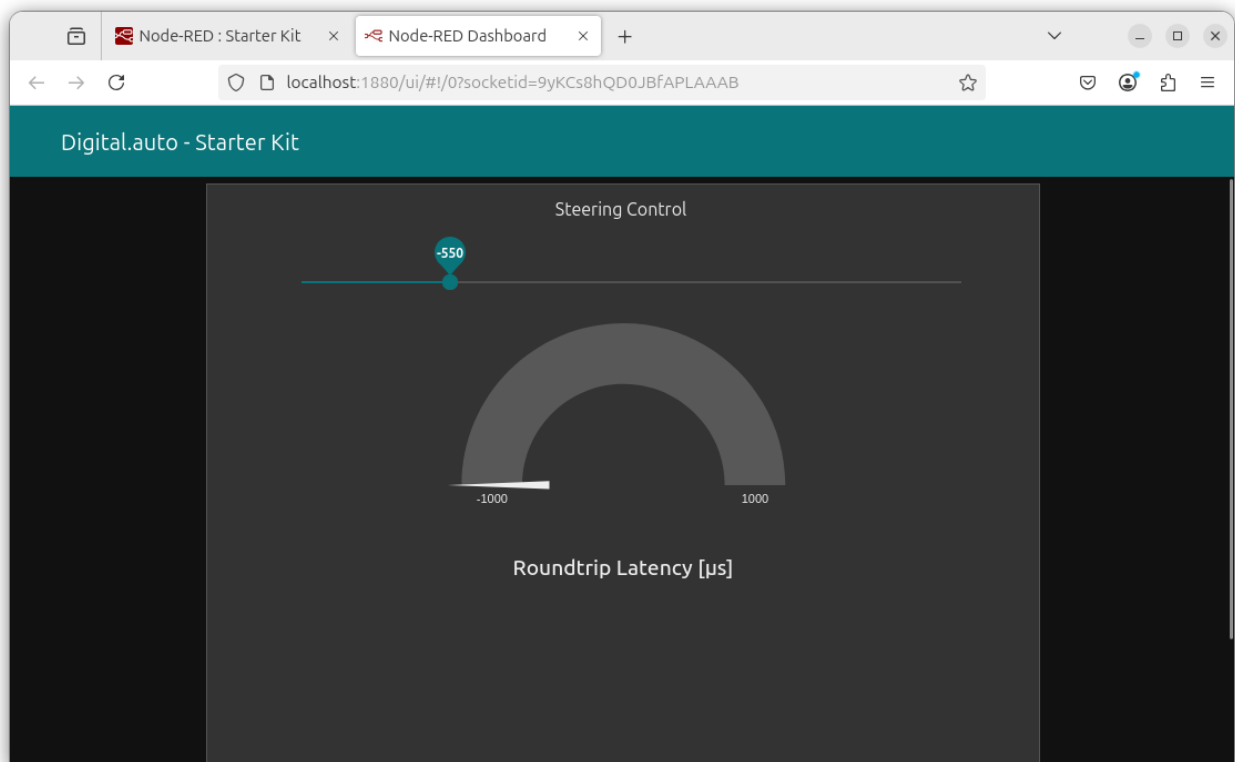
6) Download or clone node-red-starterkit-flow repository and import flow file in node-red:

- Click on the menu (≡) > Import.
- Click on **select file to import**.
- Select a flows.json file from node-red-starterkit-flow
- Click **Import**. Import Copy



- 7) Deploy the Flow and open a web browser and navigate to <http://localhost:1880/ui> to access the Node-RED Dashboard:





## 6.4 Setting Up Docker on Raspberry Pi 5

### Steps:

- 1) Install Docker (Official Script Method)

```
curl -fsSL https://get.docker.com -o get-docker.sh  
  
sudo sh get-docker.sh
```

- 2) Add Your User to the Docker Group

```
sudo usermod -aG docker $USER  
  
newgrp docker
```

- 3) Test Docker

```
docker run hello-world
```

A terminal window titled 'bluebox@bluebox: ~' with standard window controls. The terminal shows the command 'docker run hello-world' being executed. The output displays a 'Hello from Docker!' message, confirming the installation. It then lists four steps Docker took: contacting the daemon, pulling the 'hello-world' image (arm64v8), creating a new container, and streaming output to the terminal. Finally, it provides instructions on how to run an Ubuntu container and links to Docker's documentation and image hub.

```
bluebox@bluebox:~$ docker run hello-world  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (arm64v8)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
  
bluebox@bluebox:~$
```

#### 4) Enable Docker to Start on Boot

```
sudo systemctl enable docker
```

#### 5) Reboot Raspberry Pi 5

```
sudo reboot -f
```

## 6.5 Setting Up Portenta X8

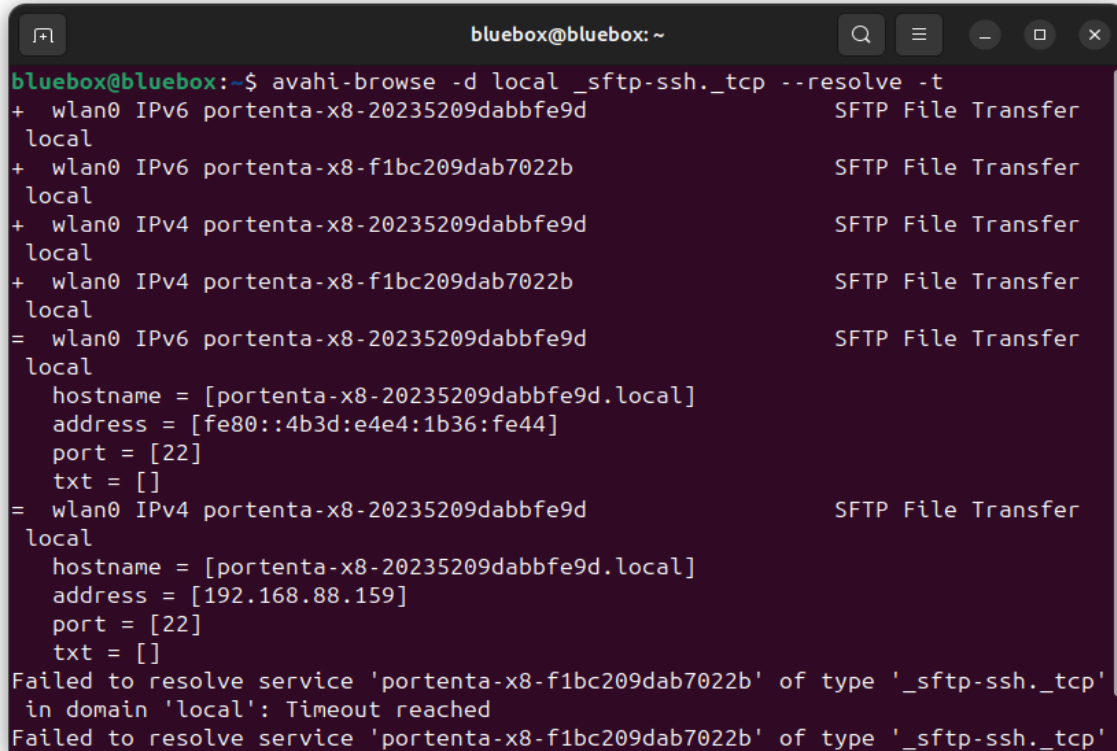
### Steps:

- 1) Connect Portenta X8 Hat Carrier to Ethernet and 12V power supply.
- 2) Open Terminal in Raspberry Pi 5.
- 3) Install avahi-utils:

```
sudo apt install avahi-utils
```

- 4) Find out IP-address of the Portenta X8 host in LAN:

```
avahi-browse -d local _sftp-ssh._tcp --resolve -t
```

A terminal window titled 'bluebox@bluebox: ~' showing the output of the command 'avahi-browse -d local \_sftp-ssh.\_tcp --resolve -t'. The output lists several services on the local network, including 'portenta-x8-20235209dabbfe9d' and 'portenta-x8-f1bc209dab7022b' on both IPv6 and IPv4. The IPv4 entry for 'portenta-x8-20235209dabbfe9d' shows the IP address '192.168.88.159'. The terminal also shows error messages for the other service: 'Failed to resolve service 'portenta-x8-f1bc209dab7022b' of type '\_sftp-ssh.\_tcp' in domain 'local': Timeout reached'.

For example, shown above, the IP-address is **192.168.88.159**

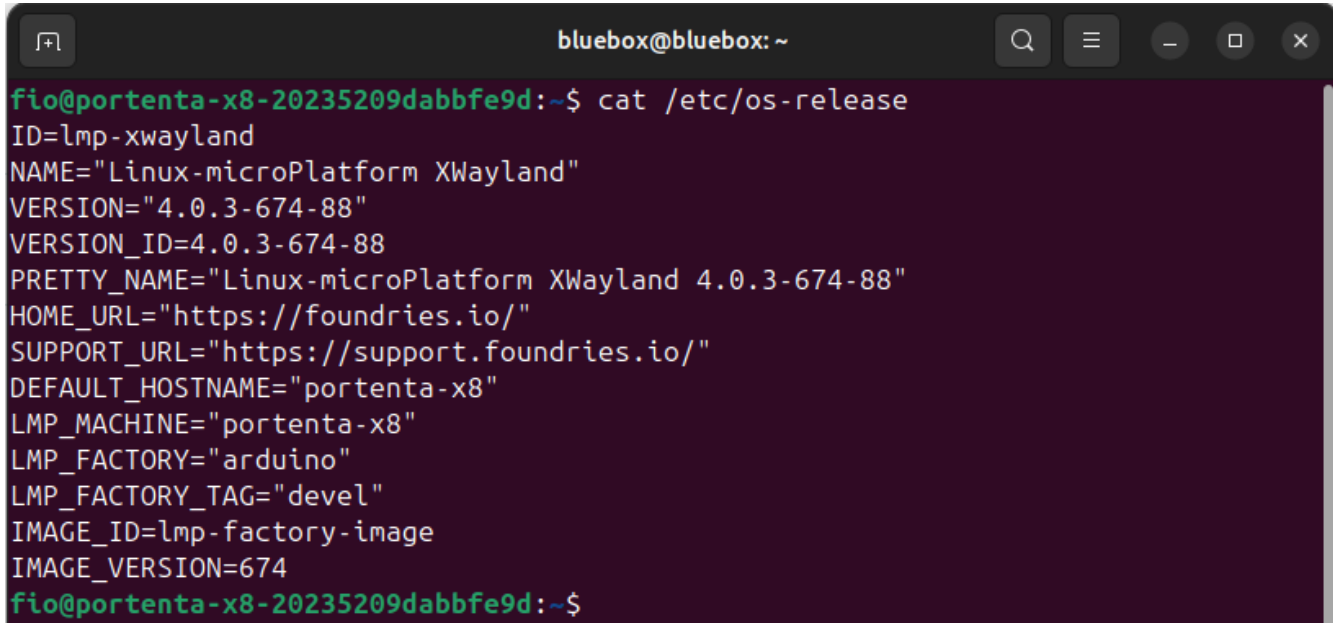
- 5) Open SSH session with Portenta X8 using the following command:

```
ssh fio@192.168.88.159
```

The default password in official image OS for user fio is **fio**

6) Check OS release currently running on the Portenta X8:

```
cat /etc/os-release
```

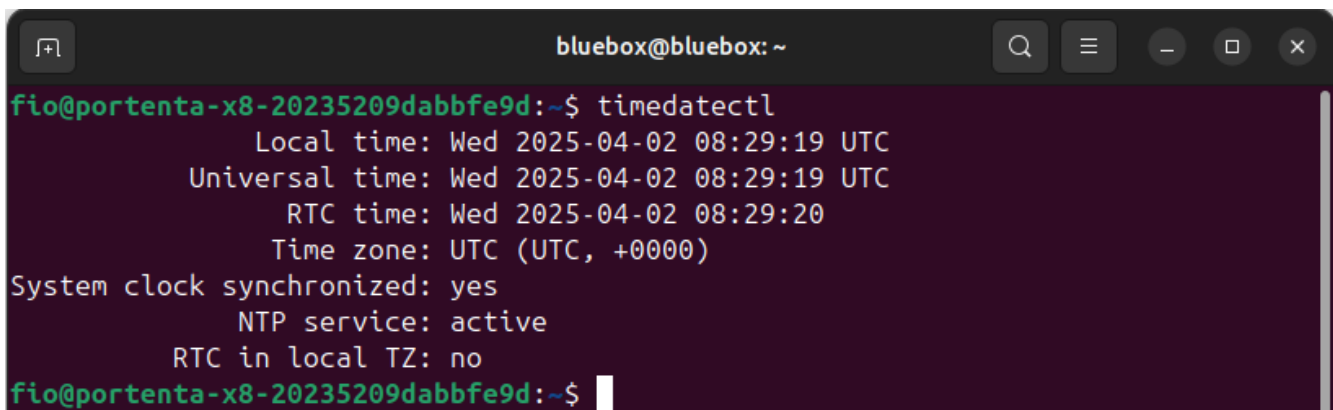


```
bluebox@bluebox: ~  
fio@portenta-x8-20235209dabbfe9d:~$ cat /etc/os-release  
ID=lmx-wayland  
NAME="Linux-microPlatform XWayland"  
VERSION="4.0.3-674-88"  
VERSION_ID=4.0.3-674-88  
PRETTY_NAME="Linux-microPlatform XWayland 4.0.3-674-88"  
HOME_URL="https://foundries.io/"  
SUPPORT_URL="https://support.foundries.io/"  
DEFAULT_HOSTNAME="portenta-x8"  
LMP_MACHINE="portenta-x8"  
LMP_FACTORY="arduino"  
LMP_FACTORY_TAG="devel"  
IMAGE_ID=lmx-factory-image  
IMAGE_VERSION=674  
fio@portenta-x8-20235209dabbfe9d:~$
```

You can check for update of the OS and find update instruction on the official Arduino documentation:  
<https://docs.arduino.cc/tutorials/portenta-x8/image-flashing/>

7) Check and set time and date

```
timedatectl
```



```
bluebox@bluebox: ~  
fio@portenta-x8-20235209dabbfe9d:~$ timedatectl  
          Local time: Wed 2025-04-02 08:29:19 UTC  
          Universal time: Wed 2025-04-02 08:29:19 UTC  
             RTC time: Wed 2025-04-02 08:29:20  
            Time zone: UTC (UTC, +0000)  
System clock synchronized: yes  
              NTP service: active  
          RTC in local TZ: no  
fio@portenta-x8-20235209dabbfe9d:~$
```

8) Set CPU Governor to Performance.  
Create systemd service file:

```
echo -e "[Unit]\nDescription=Set CPU Governor to\nPerformance\nAfter=multi-\nuser.target\n\n[Service]\nType=oneshot\nExecStart=/bin/sh -c\n'echo performance | tee\n/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor'\nRema\ninAfterExit=yes\n\n[Install]\nWantedBy=multi-user.target" |  
sudo tee /etc/systemd/system/cpu-governor.service >  
/dev/null
```

Reload the systemd manager configuration to read the new service file and enable the service so that it starts on boot:

```
sudo systemctl daemon-reload  
  
sudo systemctl enable cpu-governor
```

Start the service and check the current cpu-governor settings

```
sudo systemctl start cpu-governor  
  
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```



```
bluebox@bluebox: ~  
fio@portenta-x8-20235209dabbfe9d:~$ sudo systemctl start cpu-governor  
fio@portenta-x8-20235209dabbfe9d:~$ cat /sys/devices/system/cpu/cpu0/cpufreq/sca  
ling_governor  
performance  
fio@portenta-x8-20235209dabbfe9d:~$
```

## 9) Change real time OS settings:

Add configuration for real-time priority (rtprio), memory lock (memlock), and priority for the @realtime group to the /etc/security/limits.conf:

```
echo -e
"*\tsoft\tmemlock\t102400\n*\thard\tmemlock\t102400\n*\tsoft
\ttrprio\t99\n*\thard\ttrprio\t99\n*\tsoft\tpriority\t99\n*\th
ard\tpriority\t99" | sudo tee -a /etc/security/limits.conf >
/dev/null
```

Add session required pam\_limits.so to /etc/pam.d/common-session:

```
echo "session required pam_limits.so" | sudo tee -a
/etc/pam.d/common-session > /dev/null

echo "vm.overcommit_memory = 1" | sudo tee -a
/etc/sysctl.conf > /dev/null

sudo sysctl -p
```

A terminal window with a dark background. The title bar shows 'bluebox@bluebox: ~'. The prompt is 'fio@portenta-x8-20235209dabbfe9d:~\$'. The user enters 'echo "vm.overcommit\_memory = 1" | sudo tee -a /etc/sysctl.conf > /dev/null'. The prompt changes to 'fio@portenta-x8-20235209dabbfe9d:~\$' and the user enters 'sudo sysctl -p'. The output shows several system parameters being reloaded, including 'net.ipv4.tcp\_syncookies = 1', 'net.ipv4.conf.all.log\_martians = 1', 'net.ipv4.conf.all.accept\_redirects = 0', 'net.ipv4.conf.all.accept\_source\_route = 0', 'net.ipv4.icmp\_echo\_ignore\_broadcasts = 1', and 'vm.overcommit\_memory = 1'.

```
bluebox@bluebox: ~
fio@portenta-x8-20235209dabbfe9d:~$ echo "vm.overcommit_memory = 1" | sudo tee -a
a /etc/sysctl.conf > /dev/null
fio@portenta-x8-20235209dabbfe9d:~$ sudo sysctl -p
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
vm.overcommit_memory = 1
```

Change system default limits:

```
sudo mkdir -p /etc/systemd/system.conf.d

echo -e
"[Manager]\nDefaultLimitMEMLOCK=102400\nDefaultLimitRTPRIO=9
9" | sudo tee -a /etc/systemd/system.conf.d/limits.conf

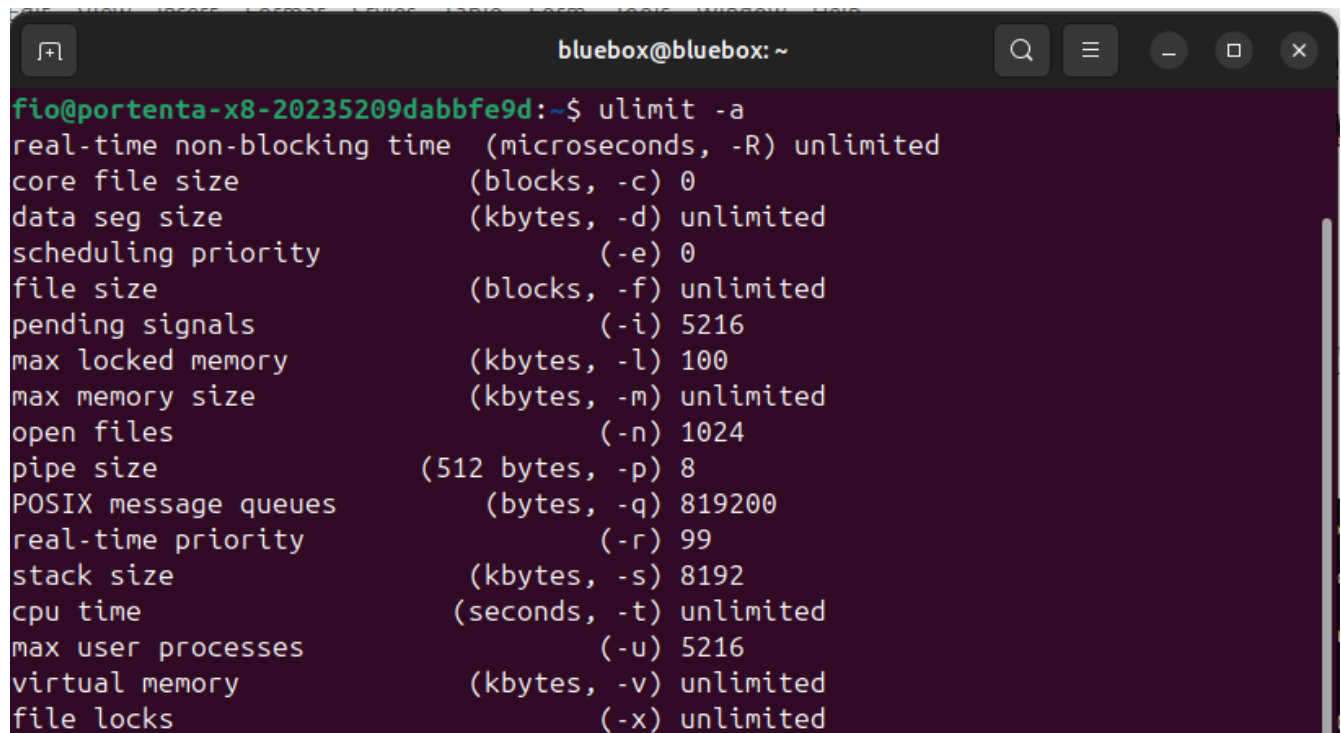
sudo systemctl daemon-reload
```

Reboot Portenta X8 to apply real time settings:

```
sudo reboot -f
```

Check the real time OS settings after reboot:

```
ulimit -a
```

A terminal window titled 'bluebox@bluebox: ~' with standard window controls. The prompt is 'fio@portenta-x8-20235209dabbfe9d:~\$'. The command 'ulimit -a' has been executed, displaying a list of system limits. The output shows various limits such as 'real-time non-blocking time' (unlimited), 'core file size' (0), 'data seg size' (unlimited), 'scheduling priority' (0), 'file size' (unlimited), 'pending signals' (5216), 'max locked memory' (100), 'max memory size' (unlimited), 'open files' (1024), 'pipe size' (8), 'POSIX message queues' (819200), 'real-time priority' (99), 'stack size' (8192), 'cpu time' (unlimited), 'max user processes' (5216), 'virtual memory' (unlimited), and 'file locks' (unlimited).

```
fio@portenta-x8-20235209dabbfe9d:~$ ulimit -a
real-time non-blocking time (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority         (-e) 0
file size                   (blocks, -f) unlimited
pending signals             (-i) 5216
max locked memory           (kbytes, -l) 100
max memory size             (kbytes, -m) unlimited
open files                  (-n) 1024
pipe size                   (512 bytes, -p) 8
POSIX message queues        (bytes, -q) 819200
real-time priority          (-r) 99
stack size                  (kbytes, -s) 8192
cpu time                    (seconds, -t) unlimited
max user processes          (-u) 5216
virtual memory              (kbytes, -v) unlimited
file locks                  (-x) unlimited
```

The Portenta X8 is ready for setting up of the Starter Kit software.



## 6.6 Setting up kuksa databroker on portenta X8

### Steps:

- 1) Clone or download **docker\_kuksa\_databroker\_mix\_vss** repository
- 2) Run following command to start Kuksa Databroker in docker container:  
(This container will start automatically at system start up)

```
cd docker_kuksa_databroker_mix_vss  
  
./rundatabrokerEW2025.sh
```

- 3) Check the log file to see the Kuksa Databroker is running:

```
docker logs -f databroker
```

```
bluebox@bluebox: ~  
e36f23594ba292718ef7ee616916b018c014846756caef5bf383aa805a66aca7  
If you are interested in logs, just run 'docker logs -f databroker'  
fio@portenta-x8-20235209dabbfe9d:~/docker_kuksa_databroker_mix_vss$ docker logs  
-f databroker  
2025-04-02T10:42:05.118141Z INFO databroker: Init logging from RUST_LOG (enviro  
nment variable not found)  
2025-04-02T10:42:05.118235Z INFO databroker: Starting Kuksa Databroker 0.5.0  
2025-04-02T10:42:05.118254Z INFO databroker: Using 4 threads with 4 cores avail  
able on the system  
2025-04-02T10:42:05.121049Z INFO databroker: Populating metadata from file 'vss  
/vssEWCombined.json'  
2025-04-02T10:42:05.173619Z WARN databroker: TLS is not enabled. Default behavi  
or of accepting insecure connections when TLS is not configured may change in th  
e future! Please use --insecure to explicitly enable this behavior.  
2025-04-02T10:42:05.173703Z WARN databroker: Authorization is not enabled.  
2025-04-02T10:42:05.174054Z INFO databroker::grpc::server: Listening on 0.0.0.0  
:55555  
2025-04-02T10:42:05.174060Z INFO databroker::viss::server: VISSv2 (websocket) s  
ervice listening on 0.0.0.0:8090  
2025-04-02T10:42:05.174085Z INFO databroker::broker: Starting housekeeping task  
2025-04-02T10:42:05.174105Z INFO databroker::grpc::server: TLS is not enabled  
2025-04-02T10:42:05.174118Z INFO databroker::grpc::server: Authorization is not  
enabled.
```

## 6.7 Setting up Kuksa MQTT Gateway

Steps:

- 1) Open Console on Raspberry Pi 5.
- 2) Clone or download the **mqtt\_kuksa\_gw** repository
- 3) View the Raspberry Pi 5 Ip-address

```
ip a
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou  
p default qlen 1000  
    link/ether 2c:cf:67:6d:0b:5a brd ff:ff:ff:ff:ff:ff  
    inet 192.168.88.111/24 brd 192.168.88.255 scope global eth0
```

In this example the IP-address is 192.168.88.111

- 4) Change the MQTT Broker IP-address to Raspberry Pi 5

```
cd mqtt_kuksa_gw  
  
nano mqtt-kuksa-gw.py
```

Update MQTT\_BROKER parameter to your Raspberry Pi 5 IP-address:

```
# MQTT broker configuration  
MQTT_BROKER = "192.168.88.111"  
MQTT_PORT = 1883
```

Save and exit Ctr+O and Ctrl+X

5) Build a Docker image from a Dockerfile:

```
docker build -t mqtt-gw-image .
```

6) Export (save) a Docker image to a .tar archive file:

```
docker save mqtt-gw-image:latest > ~/mqtt-gw-image.tar
```

7) Transfer the mqtt-gw-image.tar file to Portenta X8 machine over SSH.

The IP-address of Portenta X8 see step 4) §6.5

```
scp ~/mqtt-gw-image.tar fio@192.168.88.159:/home/fio
```

8) Open SSH session with Portenta X8 using the following command:

```
ssh fio@192.168.88.159
```

9) load a Docker image from the mqtt-gw-image.tar file into the local Docker engine.

```
docker load -i ~/mqtt-gw-image.tar
```

10) Start the docker container with Kuksa MQTT Gateway on Portenta X8

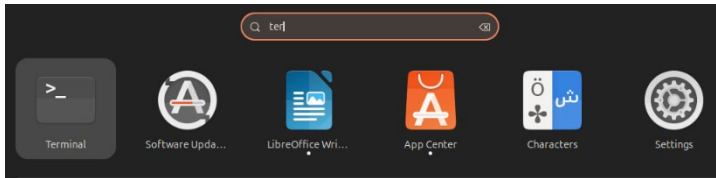
(This container will start automatically at system start up)

```
docker run -d --net=host --restart always --name mqtt-gw-  
container mqtt-gw-image
```

## 6.8 Installation of the Arduino IDE on the Raspberry Pi 5

Steps:

- 1) Connect Arduino Uno R4 WiFi Board to the USB port of the Raspberry Pi 5.
- 2) Open Terminal In Raspberry Pi Ubuntu Desktop



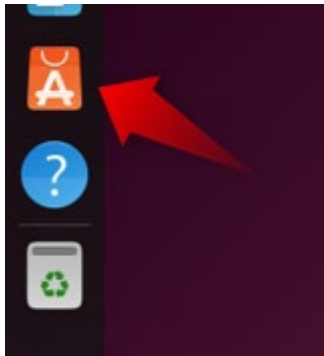
- 3) Set the permissions of the `/dev/ttyACM0` device file

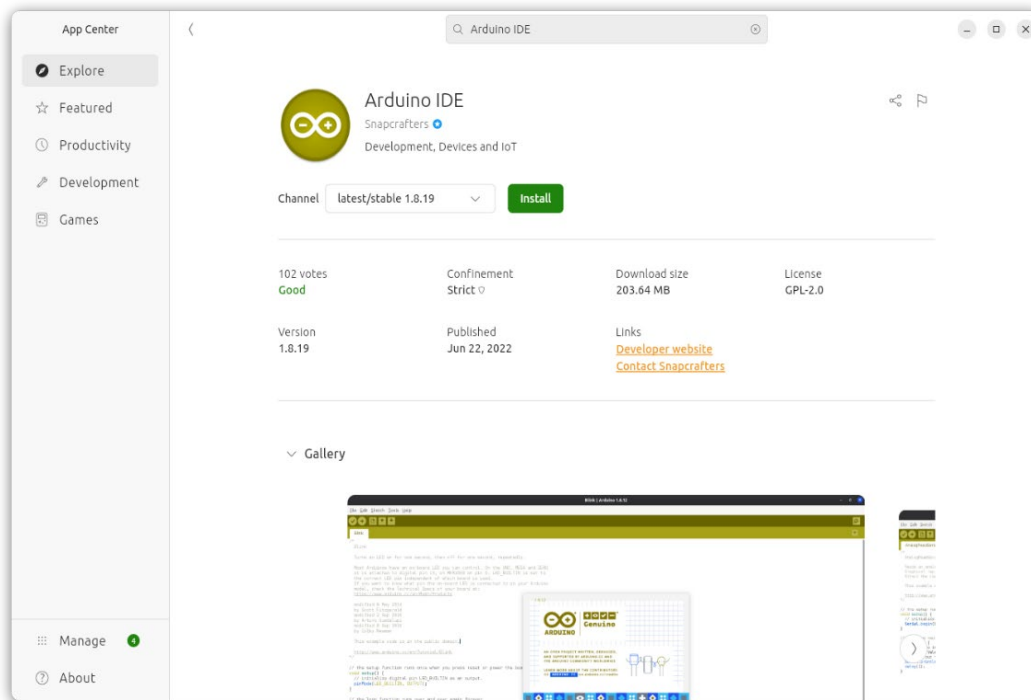
```
sudo chmod 666 /dev/ttyACM0
```

- 4) Enter:

```
echo 'SUBSYSTEMS=="usb", ATTRS{idVendor}=="2341",  
ATTRS{idProduct}=="0069", GROUP="plugdev", MODE="0666"  
SUBSYSTEMS=="usb", ATTRS{idVendor}=="2341",  
ATTRS{idProduct}=="0369", GROUP="plugdev", MODE="0666"' |  
sudo tee /etc/udev/rules.d/99-arduino-uno-r4.rules >  
/dev/null
```

- 5) Open App Center and search for Arduino IDE, then install it.

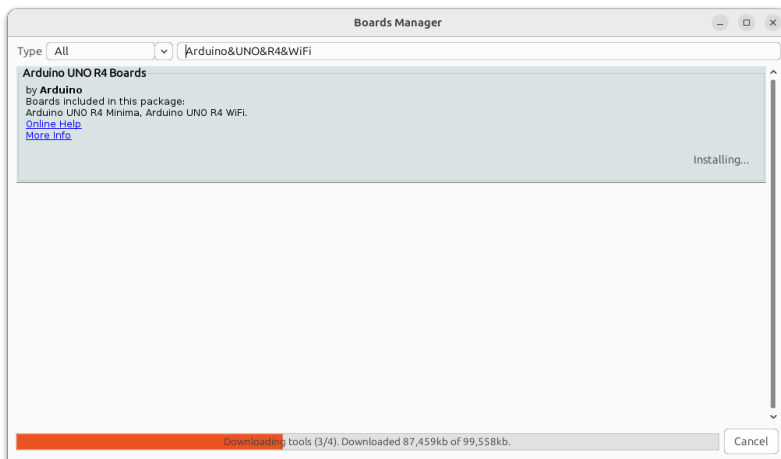




6) Run Arduino IDE.



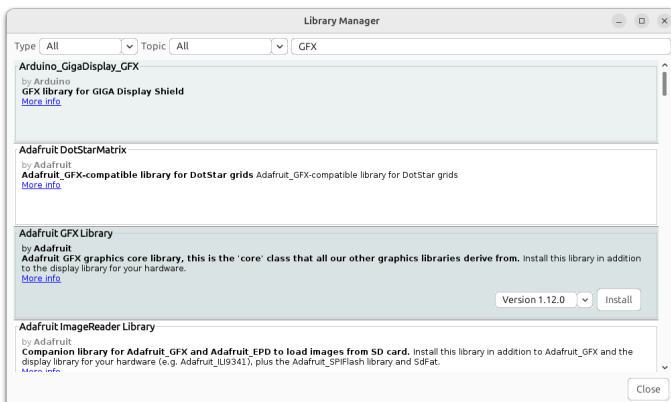
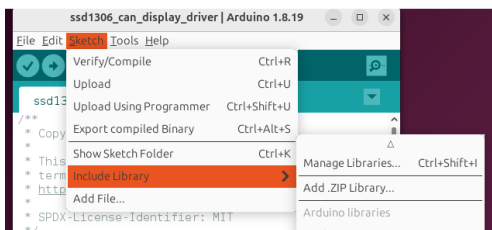
7) Open Boards Manager and install Arduino Uno R4 Boards  
Tools → Board: → Boards Manager

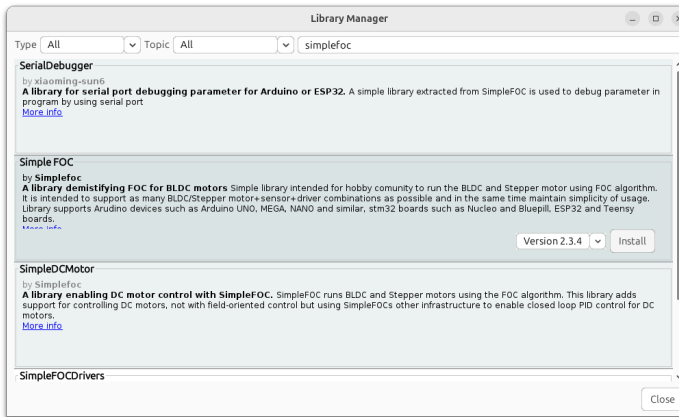
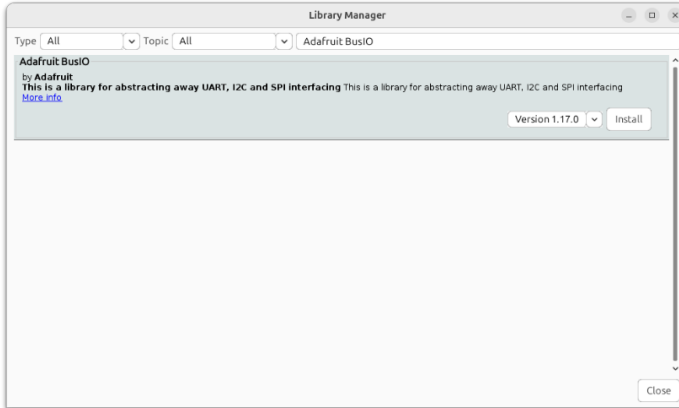
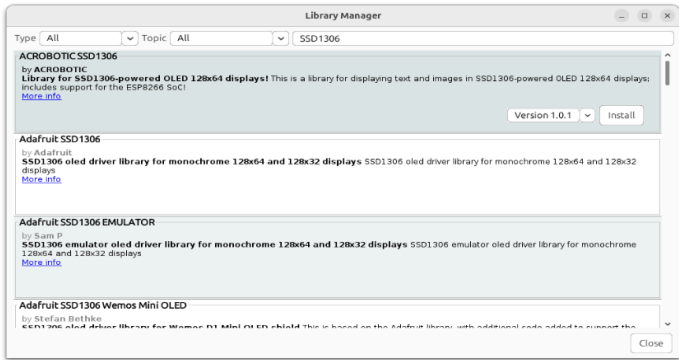


8) Install the libraries:

- Adafruit GFX
- Adafruit SSD1306
- Adafruit BusIO
- Simple FOC

Sketch → Include Library → Manage Libraries





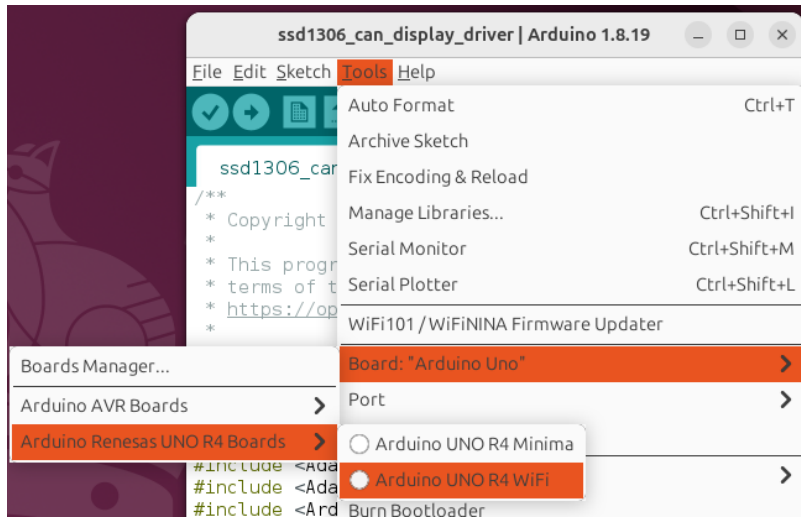


## 6.9 Setting up CAN Display Driver on Arduino UNO R4 WiFi

Steps:

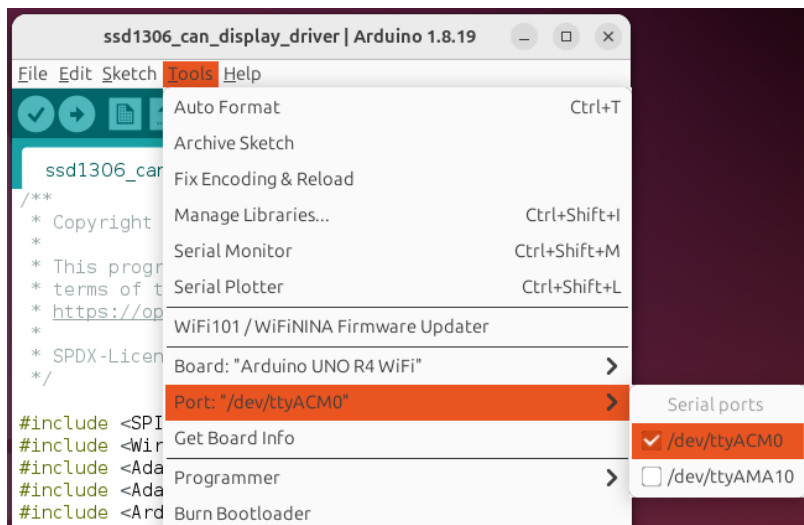
- 1) Connect Arduino Uno R4 WiFi Board to the USB port of the Raspberry Pi 5.
- 2) Select Board: **Arduino Uno R4 WiFi**

Tools→Board→Arduino Renesas UNO R4 Boards→Arduino UNO R4 WiFi

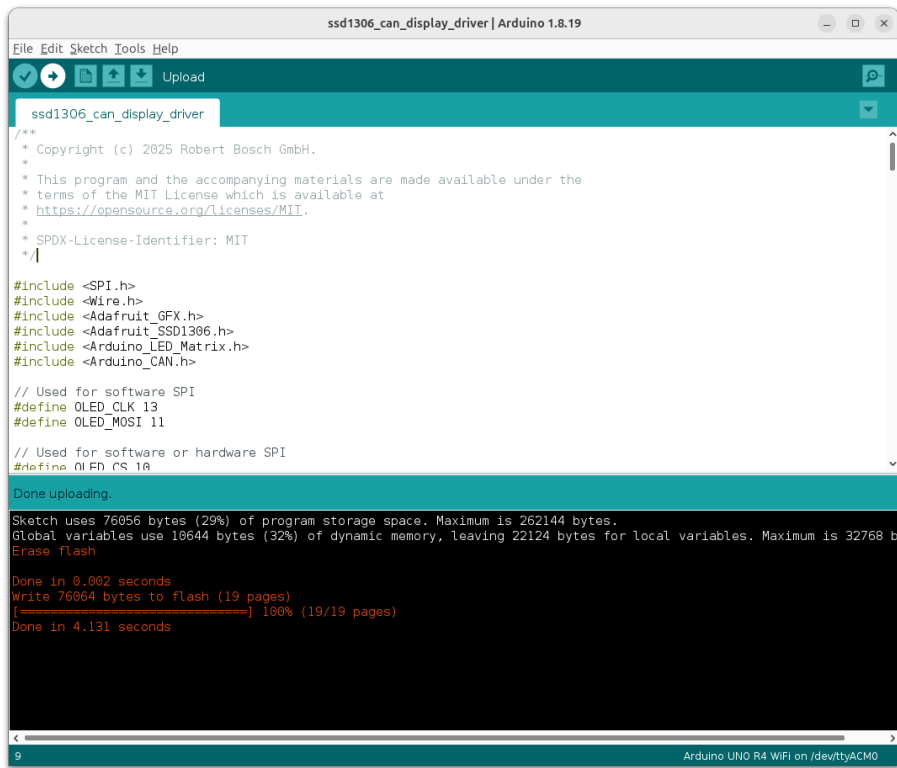


- 3) Select Port: **/dev/ttyACM0**

Tools → Port → /dev/ttyACM0



- 4) Clone or download **arduino\_can\_display\_driver** repository and open `ssd1306_can_display_driver.ino` in Arduino IDE.
- 5) Compile and upload the software



The screenshot shows the Arduino IDE interface with the sketch 'ssd1306\_can\_display\_driver' open. The code in the editor includes copyright information, license details, and various library inclusions and pin definitions. The status bar at the bottom indicates the board is 'Arduino UNO R4 WiFi on /dev/ttyACM0'.

```
File Edit Sketch Tools Help
Upload

ssd1306_can_display_driver

/**
 * Copyright (c) 2025 Robert Bosch GmbH.
 *
 * This program and the accompanying materials are made available under the
 * terms of the MIT License which is available at
 * https://opensource.org/licenses/MIT.
 *
 * SPDX-License-Identifier: MIT
 */

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Arduino_LED_Matrix.h>
#include <Arduino_CAN.h>

// Used for software SPI
#define OLED_CLK 13
#define OLED_MOSI 11

// Used for software or hardware SPI
#define OLEF_CS 10

Done uploading.

Sketch uses 76056 bytes (29%) of program storage space. Maximum is 262144 bytes.
Global variables use 10644 bytes (32%) of dynamic memory, leaving 22124 bytes for local variables. Maximum is 32768 bytes.
Erase Flash

Done in 0.002 seconds
Write 76064 bytes to flash (19 pages)
[=====] 100% (19/19 pages)
Done in 4.131 seconds

9 Arduino UNO R4 WiFi on /dev/ttyACM0
```

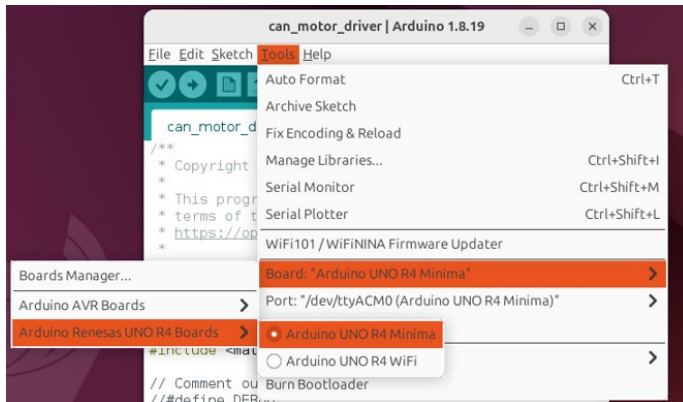
6) After uploading, disconnect the Arduino Uno R4 WiFi from Raspberry Pi.

## 6.10 Setting up CAN Motor Driver on Arduino UNO R4 Minima

Steps:

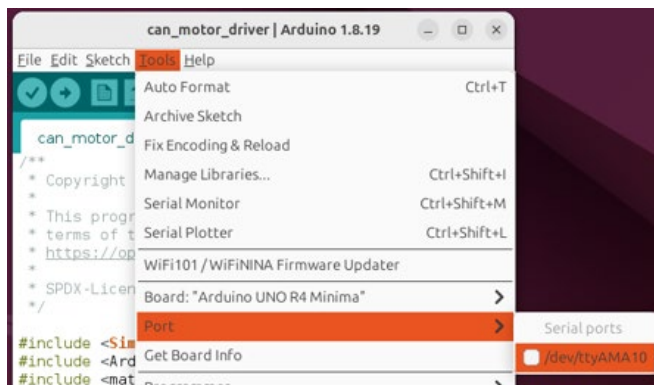
- 1) Connect Arduino Uno R4 Minima board to the USB port of the Raspberry Pi 5.
- 2) Select Board: **Arduino Uno R4 Minima**

Tools→Board→Arduino Renesas UNO R4 Boards→Arduino UNO R4 Minima



- 3) Select Port: **/dev/ttyAMA10**

Tools → Port → /dev/ttyAMA10



- 4) Clone or download **arduino\_can\_motor\_driver** repository and open **can\_motor\_driver.ino** in Arduino IDE
- 5) Download and open **can\_motor\_driver.ino** in Arduino IDE.



If an upload error occurs, disconnect and reconnect the Arduino board, then quickly press the reset button on the Arduino board twice and retry.

- 6) After uploading, disconnect the Arduino Uno R4 Minima from Raspberry Pi.
- 7) Repeat the procedure for the second Arduino UNO R4 Minima Board.

## 6.11 Installation Zephyr on the Raspberry Pi 5

Please find official Zephyr installation instructions here:

[https://docs.zephyrproject.org/latest/develop/getting\\_started/index.html](https://docs.zephyrproject.org/latest/develop/getting_started/index.html)

Steps:

- 1) Open Terminal in Raspberry Pi5 Desktop and update OS:

```
sudo apt update -y & sudo apt upgrade
```

- 2) Install the required dependencies:

```
sudo apt install --no-install-recommends git cmake ninja-  
build gperf \  
  
ccache dfu-util device-tree-compiler wget \  
  
python3-dev python3-pip python3-setuptools python3-tk  
python3-wheel xz-utils file \  
  
make gcc libsdl2-dev libmagic1
```

- 3) Verify the versions of the main dependencies:

```
cmake --version  
  
python3 --version  
  
dtc --version
```

```
bluebox@bluebox:~$ cmake --version  
python3 --version  
dtc --version  
cmake version 3.28.3  
  
CMake suite maintained and supported by Kitware (kitware.com/cmake).  
Python 3.12.3  
Version: DTC 1.7.0
```

4) Install Python venv package:

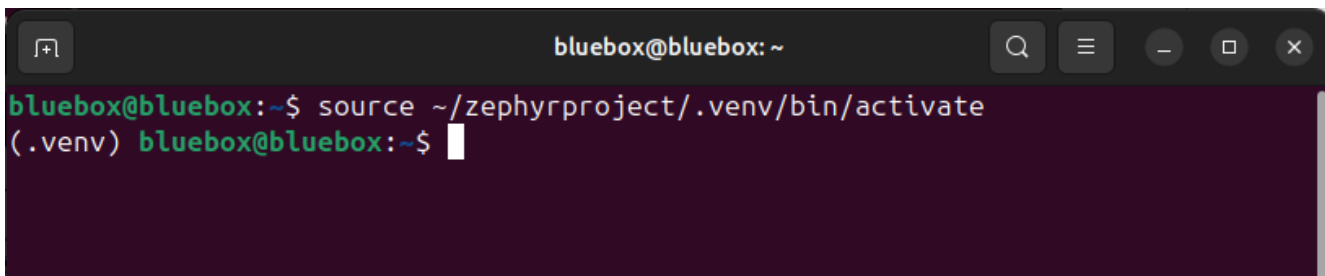
```
sudo apt install python3-venv
```

5) Create a new virtual environment:

```
python3 -m venv ~/zephyrproject/.venv
```

6) Activate the virtual environment:

```
source ~/zephyrproject/.venv/bin/activate
```

A terminal window with a dark background. The title bar shows 'bluebox@bluebox: ~'. The prompt is 'bluebox@bluebox:~\$'. The command 'source ~/zephyrproject/.venv/bin/activate' has been entered. The output shows the prompt changed to '(.venv) bluebox@bluebox:~\$' with a cursor at the end.

```
bluebox@bluebox:~$ source ~/zephyrproject/.venv/bin/activate  
(.venv) bluebox@bluebox:~$
```

7) Install west:

```
pip install west
```

8) Get the Zephyr source code:

```
west init ~/zephyrproject  
cd ~/zephyrproject  
west update
```

9) Export a Zephyr CMake package:

```
west zephyr-export
```

10) Install Python dependencies:

```
west packages pip --install
```

#### 11) Install the Zephyr SDK:

```
cd ~/zephyrproject/zephyr  
  
west sdk install
```

#### 12) Add configuration for Arduino Portenta H7:

```
echo 'SUBSYSTEMS=="usb", ATTR{idVendor}=="2341",  
MODE=="0666"' > 20-portenta.rules  
  
sudo mv 20-portenta.rules /etc/udev/rules.d/  
  
sudo udevadm control --reload-rules && sudo udevadm trigger
```

## 6.12 Setting up UDP CAN Gateway on Arduino Portenta H7

Steps:

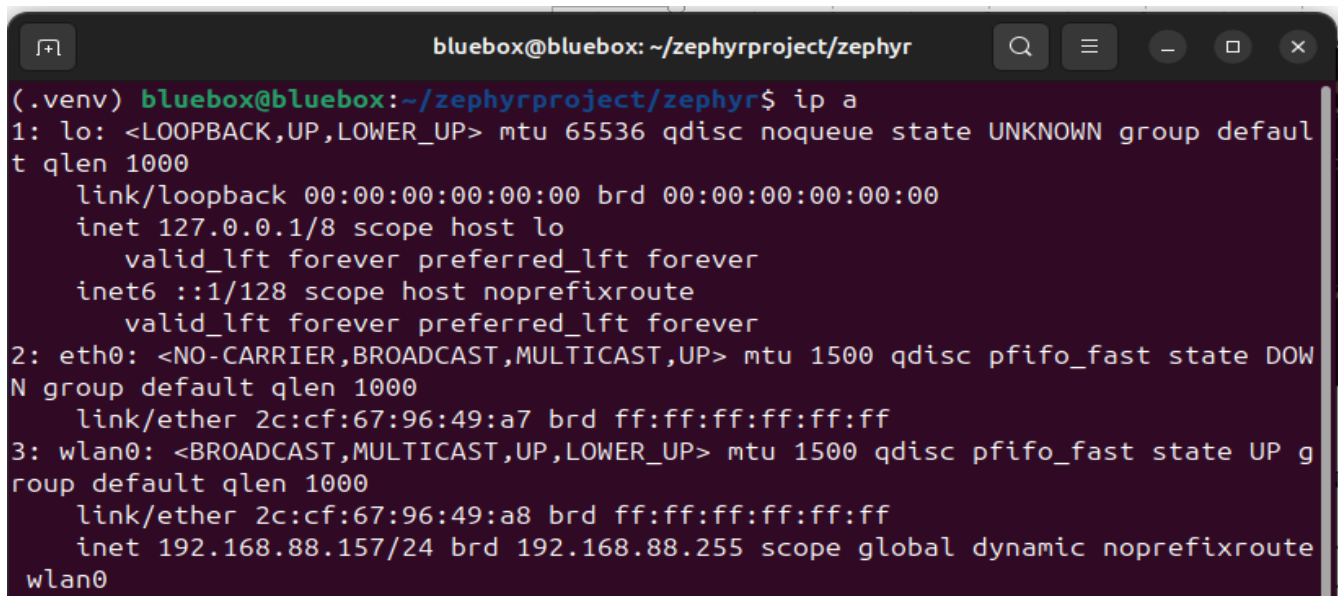
- 1) Open Terminal in Raspberry Pi5 Desktop and run command:

```
cd ~/zephyrproject/zephyr/  
  
source ~/zephyrproject/.venv/bin/activate
```

- 2) Download or clone the **zephyr\_udp\_can\_gateway** repository.
- 3) Choose gateway IP-Address:

IP address of the gateway should be within your LAN range. Enter **ip a** to find your LAN IP address:

```
ip a
```



```
bluebox@bluebox: ~/zephyrproject/zephyr  
(.venv) bluebox@bluebox:~/zephyrproject/zephyr$ ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host noprefixroute  
        valid_lft forever preferred_lft forever  
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000  
    link/ether 2c:cf:67:96:49:a7 brd ff:ff:ff:ff:ff:ff  
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000  
    link/ether 2c:cf:67:96:49:a8 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.88.157/24 brd 192.168.88.255 scope global dynamic noprefixroute wlan0
```

For example above the following IP addresses for the gateways could be chosen e.g.:

- Left gateway: 192.168.88.21
- Right gateway: 192.168.88.22

Note: These IP-addresses shall be updated accordingly in the configuration of the `haptic_force_controller` application (`starterkit_config.yaml`)

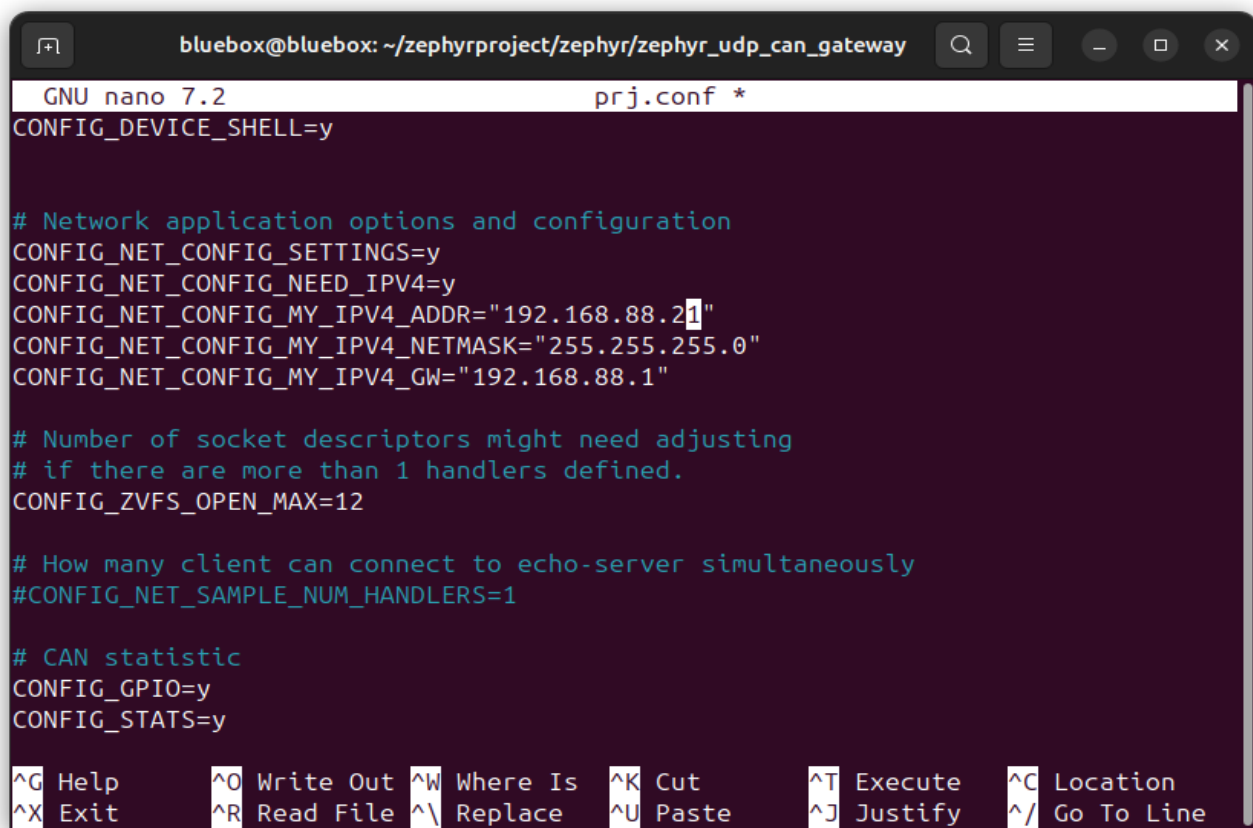


4) Set the gateway IP address:

```
cd zephyr_udp_can_gateway  
  
nano prj.conf
```

Update the IP-addresses accordingly in this file:

```
CONFIG_NET_CONFIG_MY_IPV4_ADDR="192.168.88.21"  
CONFIG_NET_CONFIG_MY_IPV4_NETMASK="255.255.255.0"  
CONFIG_NET_CONFIG_MY_IPV4_GW="192.168.88.1"
```



```
bluebox@bluebox: ~/zephyrproject/zephyr/zephyr_udp_can_gateway  
GNU nano 7.2 prj.conf *  
CONFIG_DEVICE_SHELL=y  
  
# Network application options and configuration  
CONFIG_NET_CONFIG_SETTINGS=y  
CONFIG_NET_CONFIG_NEED_IPV4=y  
CONFIG_NET_CONFIG_MY_IPV4_ADDR="192.168.88.21"  
CONFIG_NET_CONFIG_MY_IPV4_NETMASK="255.255.255.0"  
CONFIG_NET_CONFIG_MY_IPV4_GW="192.168.88.1"  
  
# Number of socket descriptors might need adjusting  
# if there are more than 1 handlers defined.  
CONFIG_ZVFS_OPEN_MAX=12  
  
# How many client can connect to echo-server simultaneously  
#CONFIG_NET_SAMPLE_NUM_HANDLERS=1  
  
# CAN statistic  
CONFIG_GPIO=y  
CONFIG_STATS=y  
  
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location  
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify  ^_ Go To Line
```

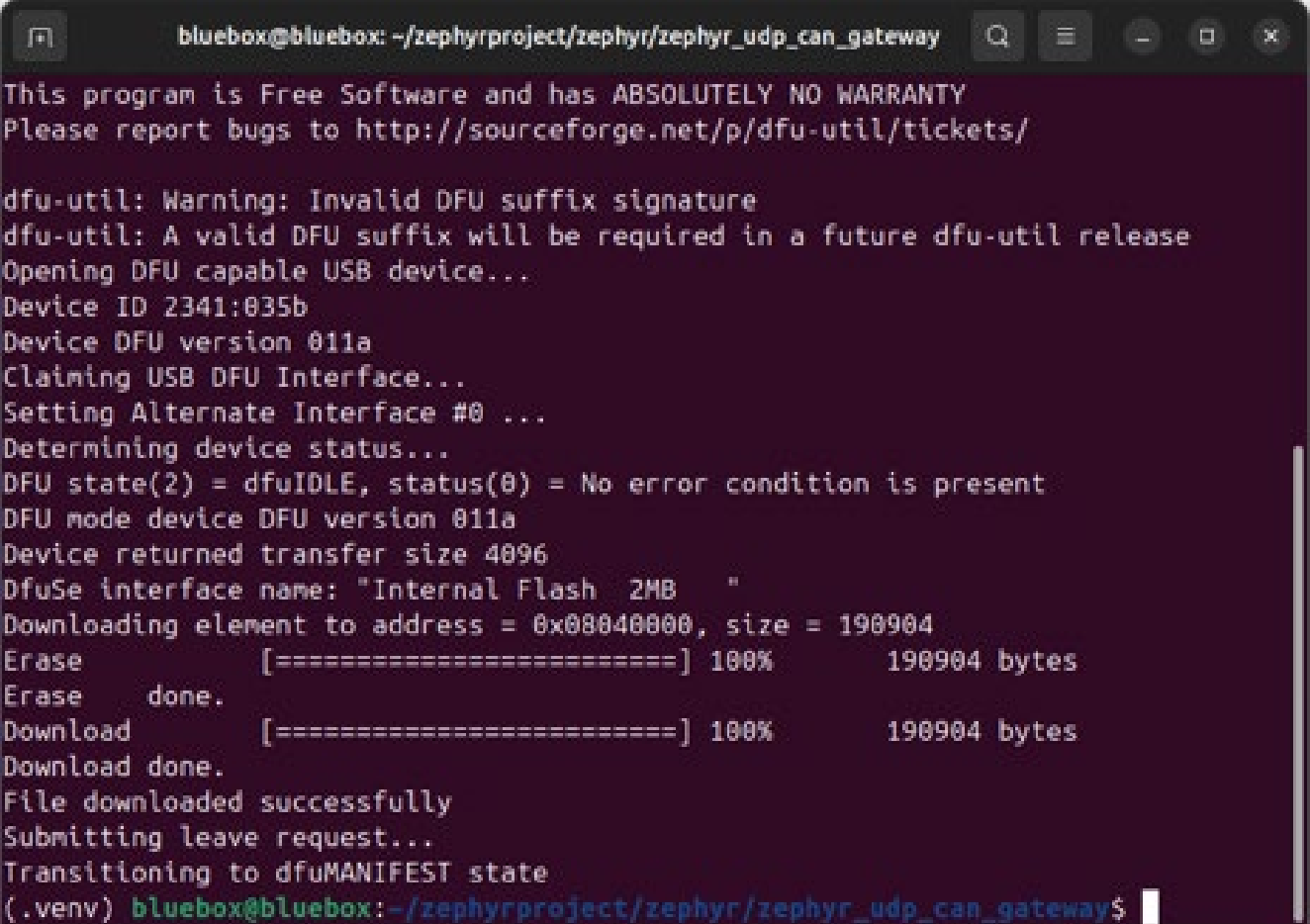
Save and exit Ctr+O and Ctrl+X

5) Build the gateway software:

```
west build -p always -b arduino_portenta_h7/stm32h747xx/m7 .
```

- 6) Connect the USB port of the Portenta H7 to the Raspberry Pi and press the reset button on the Arduino Portenta board twice. The LED will start flashing green, indicating flashing mode.
- 7) Upload the software:

```
west flash
```



```
bluebox@bluebox: ~/zephyrproject/zephyr/zephyr_udp_can_gateway
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

dfu-util: Warning: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release
Opening DFU capable USB device...
Device ID 2341:035b
Device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Interface #0 ...
Determining device status...
DFU state(2) = dfuIDLE, status(0) = No error condition is present
DFU mode device DFU version 011a
Device returned transfer size 4096
DfuSe interface name: "Internal Flash  2MB  "
Downloading element to address = 0x08040000, size = 190904
Erase      [=====] 100%      190904 bytes
Erase done.
Download   [=====] 100%      190904 bytes
Download done.
File downloaded successfully
Submitting leave request...
Transitioning to dfuMANIFEST state
(.venv) bluebox@bluebox:~/zephyrproject/zephyr/zephyr_udp_can_gateway$
```

- 8) Repeat steps 3...6 for the **right** gateway Arduino Portenta H7.

## 6.13 Setting up CPP Haptic Force Controller

Steps:

- 1) Open Console on Raspberry Pi 5.
- 2) Clone or download the **cpp\_haptic\_force\_controller** repository
- 3) View the Raspberry Pi 5 Ip-address

```
ip a
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 2c:cf:67:6d:0b:5a brd ff:ff:ff:ff:ff:ff
    inet 192.168.88.111/24 brd 192.168.88.255 scope global eth0
```

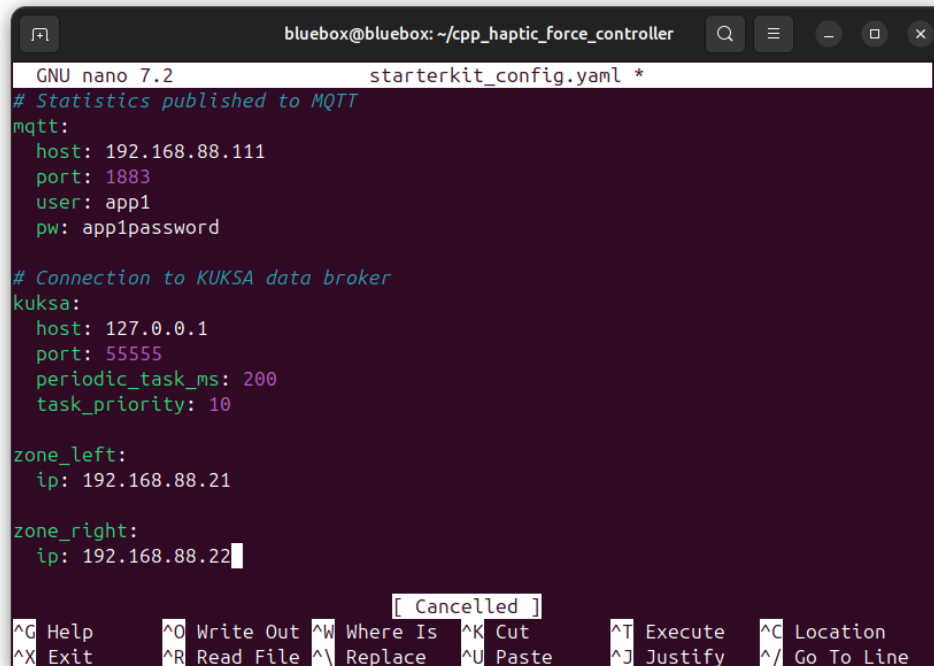
In this example the IP-address is 192.168.88.111

- 4) Set the IP-Addresses of MQTT Broker and gateway IP address in the starterkit\_config.yaml file :

```
cd cpp_haptic_force_controller

nano starterkit_config.yaml
```

The gateway IP-addresses has been chosen in step 4) §6.12



```
bluebox@bluebox: ~/cpp_haptic_force_controller
GNU nano 7.2 starterkit_config.yaml *
# Statistics published to MQTT
mqtt:
  host: 192.168.88.111
  port: 1883
  user: app1
  pw: app1password

# Connection to KUKSA data broker
kuksa:
  host: 127.0.0.1
  port: 55555
  periodic_task_ms: 200
  task_priority: 10

zone_left:
  ip: 192.168.88.21

zone_right:
  ip: 192.168.88.22
[Cancelled]
```

Help Write Out Where Is Cut Execute Location  
Exit Read File Replace Paste Justify Go To Line

Save and exit Ctr+O and Ctrl+X

- 5) Build a Docker image from a Dockerfile:

```
docker build -t cpp_hfc_image:latest -f  
dockerfiles/Dockerfile.ew .
```

- 6) Export (save) a Docker image to a .tar archive file:

```
docker save cpp_hfc_image:latest -o ~/cpp-hfc-image.tar
```

- 7) Transfer the `cpp-hfc-image.tar` file to Portenta X8 machine over SSH.  
The IP-address of Portenta X8 see step 4) §6.5

```
scp ~/cpp-hfc-image.tar fio@192.168.88.159:/home/fio
```

- 8) Open SSH session with portenta X8 using the following command:

```
ssh fio@192.168.88.159
```

- 9) Load a Docker image from the `mqtt-gw-image.tar` file into the local Docker engine.

```
docker load -i ~/cpp-hfc-image.tar
```

- 10) Start the docker container interactively with CPP Haptic Force Controller on Portenta X8

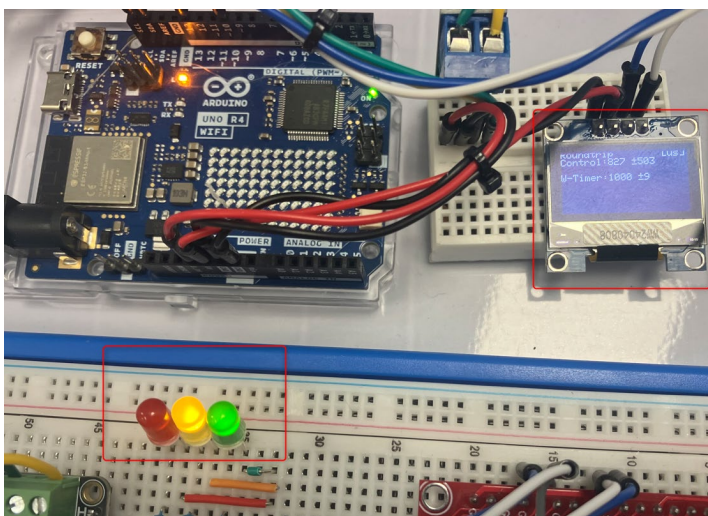
```
docker run -it --rm --network host --cap-add=ALL --  
security-opt seccomp=unconfined --privileged --name  
cpp_haptic_controller_run cpp_hfc_image:latest
```

```
bluebox@bluebox: ~/mqtt_kuksa_gw
Invalid socket index[DEBUG] Scheduler Policy: 1 | Min Prio: 1 | Max Prio: 99
[DEBUG THREAD] Scheduler Policy: 1 | Priority: 80
5 is out of range! Ignoring...
[DEBUG] Applied Policy:
0 | Applied Priority: 0[WARNING] Core [DEBUG] Available CPU Cores: 4

6[WARNING] Core 4 is out of range! Ignoring...
[WARNING] Core 5 is out of range! Ignoring...
is out of range! Ignoring...[DEBUG THREAD] Scheduler Policy: 1 | Priority: 10

[WARNING] Core 7[WARNING] Core [DEBUG] Available CPU Cores: 4
6 is out of range! Ignoring...
[WARNING] Core 7 is out of range! Ignoring...
is out of range! Ignoring...
Walltimer Avg: 999, Jitter: 37
Left -> Roundtrip Avg: 1242, Jitter: 832
Right -> Roundtrip Avg: 0, Jitter: 0
CPU Usage: 91%
Memory Usage: 19.3392%
Walltimer Avg: 1000, Jitter: 420
Left -> Roundtrip Avg: 1222, Jitter: 1407
Right -> Roundtrip Avg: 0, Jitter: 0
CPU Usage: 91%
Memory Usage: 19.364%
```

Now the application is running, you should see the Status LED is lighting yellow, that indicates the communication of the Arduino UNOs with Portenta X8. On OLED you should see the real time statistic data.



Turn the left or right motor and see that the second motor is turning accordingly to the position of the first motor.

Open a web browser and navigate to <http://localhost:1880/ui> to see the Node-RED dashboard created in step 7) §6.3



### Troubleshooting:

- LED is lighting yellow, but motors are not turning reset the Arduino UNO R4 Minima