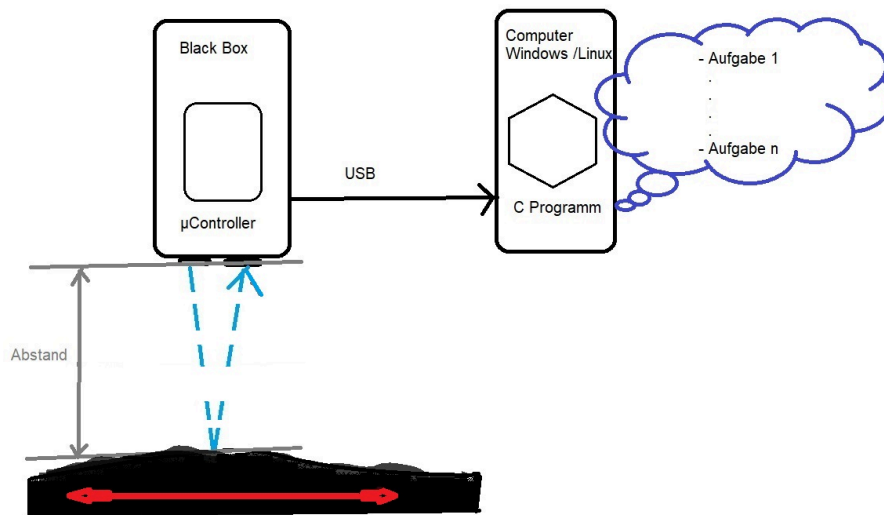


## Laboraufgabe.

Es soll eine Höhenmesser entwickelt werden, der ein Höhenprofil aufnehmen kann. Dazu wird in diesem Labor die Informatik - Komponente entwickelt.

---



Handskizze

Für die Entwicklung ist schon ein Hardware-Aufbau (Black-Box) vorhanden. Diese bietet verschiedene Grundfunktionen an, die anhand der Aufgaben erweitert werden sollen.

## Beschreibung Hardware-Blackbox:

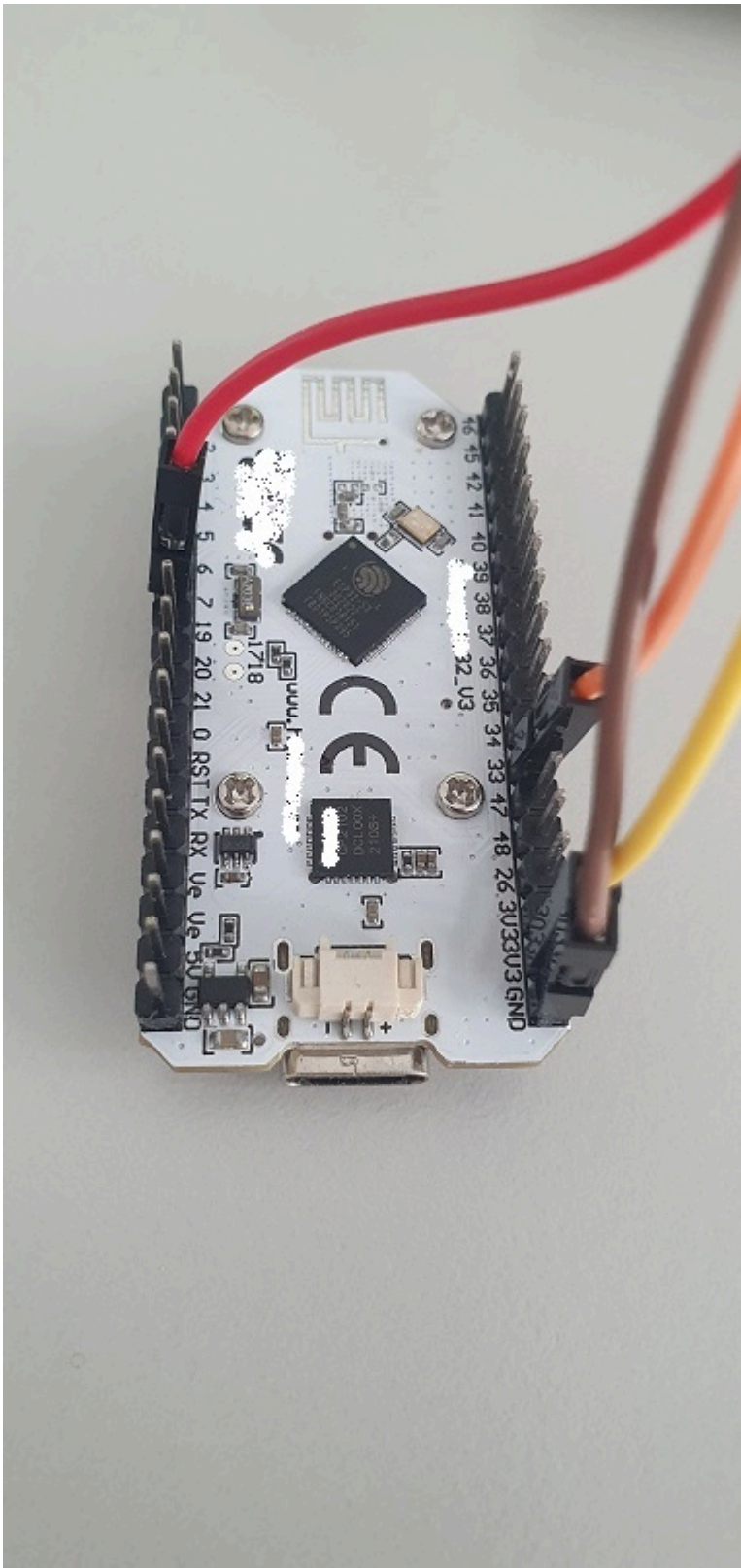
---



Ultraschal Sensor



*µController Vorderseite*



*µController Rückseite*

Figure 1: Komponenten der Black-Box.



Ultraschal Sensor

Die Blackbox ist ein Gehäuse in dem ein  $\mu$  Controller Signale von einem Ultraschall Sensor (US-Sensor) erfasst diese verarbeitet und über die USB Schnittstelle ausgibt. Die gesendeten Daten sind im INT (Integer) Format und werden mit einer Frequenz von 1Hz ausgegeben Die Spannungsversorgung erfolgt auch über den USB Anschluss (5V/0,500mA).

Die Daten die von der Black Box über die serielle Schnittstelle (USB) gesendet werden können sowohl unter Windows als auch unter einem Linux System bearbeitet werden.

## Daten im **Windows** Betriebs-System empfangen und auslesen/speichern

### Schritt-für-Schritt-Anleitung

#### 1. COM-Port herausfinden

Öffne den Windows Geräte-Manager und suche unter **"Anschlüsse (COM & LPT)"** nach der angeschlossenen Black Box, z. B. *"Silicon Labs CP210x USB to UART Bridge Port (COM5)"*.

#### 2. C-Code zum Auslesen eines COM-Ports

Speichere folgenden C-Code in eine Datei, z. B. `read_com_port.c`:

```
//Programm zum auslesen eines USB Port's
#include <windows.h>
#include <stdio.h>

int main() {
    HANDLE hSerial;
    DCB dcbSerialParams = {0};
    COMMTIMEOUTS timeouts = {0};
    char portName[] = "\\.\COMX"; // Ändere X auf die Port Nr.
    char buffer[1024];
    DWORD bytesRead;
```

```
// COM-Port öffnen
hSerial = CreateFileA(portName, GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
if (hSerial == INVALID_HANDLE_VALUE) {
    fprintf(stderr, "Fehler beim Öffnen des Ports %s\n", portName);
    return 1;
}

// Serielle Einstellungen
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
if (!GetCommState(hSerial, &dcbSerialParams)) {
    fprintf(stderr, "Fehler beim Lesen der Port-Einstellungen\n");
    CloseHandle(hSerial);
    return 1;
}

dcbSerialParams.BaudRate = CBR_Y;    // Baudrate ggf. anpassen z.B. Y= 115200
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity    = NOPARITY;

if (!SetCommState(hSerial, &dcbSerialParams)) {
    fprintf(stderr, "Fehler beim Setzen der Port-Einstellungen\n");
    CloseHandle(hSerial);
    return 1;
}

// Timeout-Einstellungen
timeouts.ReadIntervalTimeout = 50;
timeouts.ReadTotalTimeoutConstant = 50;
timeouts.ReadTotalTimeoutMultiplier = 10;

if (!SetCommTimeouts(hSerial, &timeouts)) {
    fprintf(stderr, "Fehler beim Setzen der Timeouts\n");
    CloseHandle(hSerial);
    return 1;
}

// Datei zum Speichern öffnen
FILE *outFile = fopen("daten.txt", "w");
if (!outFile) {
    fprintf(stderr, "Konnte Ausgabedatei nicht öffnen\n");
    CloseHandle(hSerial);
    return 1;
}

printf("Lese vom USB-Gerät...\n");

while (1) {
    if (ReadFile(hSerial, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {
        if (bytesRead > 0) {
            buffer[bytesRead] = '\0'; // Null-terminieren
            printf("%s", buffer);     // Ausgabe auf Konsole
            fprintf(outFile, "%s", buffer); // Schreiben in Datei
            fflush(outFile);          // Sicherstellen, dass sofort geschrieben wird
        }
    }
}
```

```
    }  
    } else {  
        fprintf(stderr, "Fehler beim Lesen vom COM-Port\n");  
        break;  
    }  
}  
  
fclose(outFile);  
CloseHandle(hSerial);  
return 0;  
}
```

### 3. Kompilieren

Öffne die Eingabeaufforderung und kompiliere mit z. B.:

```
gcc read_com_port.c -o read_com_port.exe
```

(Erfordert, dass `gcc` installiert und in der PATH-Variable ist.)

### 4. Ausführen

```
read_com_port.exe
```

Wenn dein USB-Gerät Daten sendet, siehst du sie auf der Konsole und sie werden in `daten.txt` gespeichert.

## Daten im **Linux** Betriebs-System empfangen und auslesen/speichern

### Schritt-für-Schritt-Anleitung

```
// usb: USB0 Baud; Rate 115200  
  
// mit "cat /dev/ttyUSB0" kann der Port Testweise gelesen werden.  
  
#include <iostream>  
#include <fcntl.h>  
#include <termios.h>  
#include <unistd.h>  
#include <cstring>  
  
int main() {  
    const char* port = "/dev/ttyUSB0"; // Port Nr.  
    int serial_port = open(port, O_RDWR);  
  
    .....  
}
```

# Rahmenbedienung

- Es muss in C programmiert werden.
- Verwendung einer sqlite-Datenbank
- Verwendung von `structs`
- Aufteilen des Codes in Header (.h) und Source (.c) Dateien
- Verwendung von CMake als Buildsystem
- Jede Laboraufgabe wird auf einem eigenen Branch in github oder gitlab entwickelt.
- Die Implementierung erfolgt in dem während der Vorlesung erstellten C-Projekt-Template.

## Laborsetup

Implementierung erfolgt auf den Jetsons aus dem Digitalisierungslabor. Diese wird durch die Dozenten bereitgestellt und supportet. Ein Austausch des Codes erfolgt über git (github oder gitlab).

Das C-Project kann auf den Jetsons ohne devcontainer verwendet werden, da es schon ein Linux als host System ist.

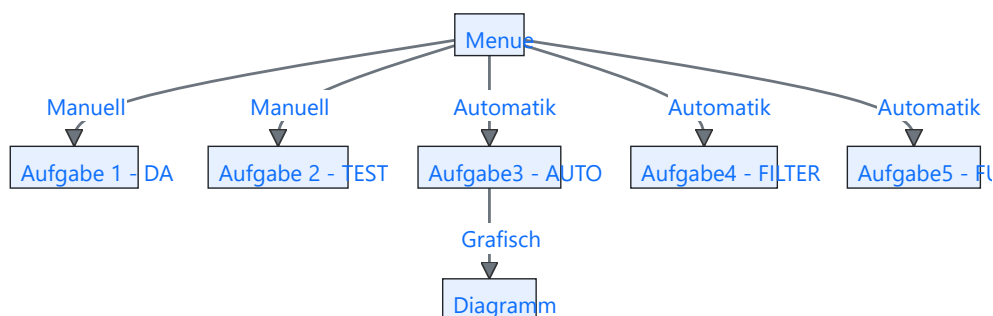
Um die Entwicklung auf Windows-PCs zu ermöglichen und in dem bekannten Setup mit Visual Studio Code und cmake zu arbeiten gibt es folgende Möglichkeiten:

1. Mingw Installation auf Windows mit cmake und gcc ()
2. Windows Subsystem for Linux (WSL) mit Ubuntu installieren und dort cmake und gcc installieren.
3. Serielle Kommunikation zur Blackbox in devcontainer einrichten. Siehe Anhang

## Abnahme

- Die Laboraufgaben werden einzeln abgenommen.
- Der Code wie auch die erzeugten Visualisierungen müssen erklärt werden.
- Der Code muss sauber mit Methoden, structs und unterschiedlichen .h und .c Dateien strukturiert sein. (vgl. Vorlesungsprojekt)
- Die weiteren Rahmenbedingungen müssen eingehalten werden.

Programm Struktur



## Aufgabe 1

Plausibilität der eingelesenen Werte. Modus "DA" - DatenAufnahme

Ziel: Es ist ein C Programm zu entwickeln, mit dem der von Hand gemessene Abstand und der Blackbox gelieferter Wert auf Tastendruck in einer csv-Datei **messung\_x.csv** speichert.

## Labortask Beschreibung

1. US-Sensor in einer festen Position zu einem Objekt abstellen
2. Den Abstand vom Sensor zur Fläche mit einem Lineal messen und über die Kommandozeile (in Meter) eingeben und mit Enter bestätigen. Dieser Wert soll als realer Abstandswert in der csv-Datei gespeichert werden. (siehe nachfolgende Tabelle)
3. In der nachfolgenden Spalte den vom Sensor gelieferte Wert speichern (auch Wert in Meter)
4. In der nachfolgenden Spalte die berechnete Differenz der beiden Werte eintragen
5. 20 Messungen Anfang = 0m, Schrittweite = 0,002 m durchführen.
6. Nach 20 Punkten soll die Eingabe nicht mehr möglich sein und das Programm springt automatisch zum Menue über
7. Speichern der Daten in einer Datenbank (sqlite). Das Datenbank Layout ist passend gewählt. Aus der Datenbank kann eine Tabelle als csv exportiert werden.
8. Option: Plot mit reeler und den vom Sensor gelieferte Werte in Excel programmieren und graphisch ausgeben.

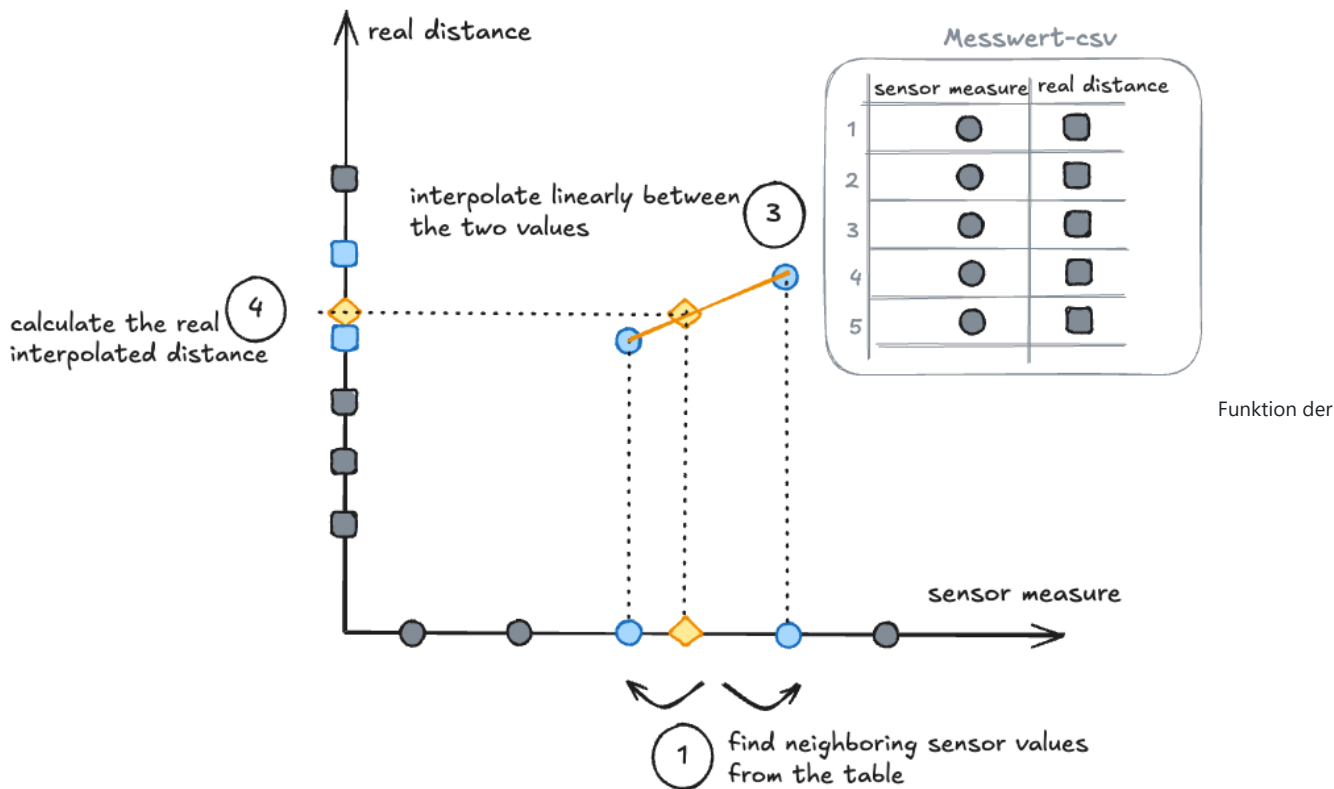
Messung Nr.	ABSTAND	Abweichung
	Reeler Wert	Vom Sensor erfasster Wert
	über Tastatureingabe	über USB eingelesen
1		
2		
...		
n-1		
n		

## Aufgabe 2

Kalibrierung des Sensors mit einer Look-Up Tabelle. Modus "TEST"

- Die Ermittlung soll auf Basis der **messung\_1.csv** erfolgen.
- Der Sensorwert soll in den realen Abstandswert mit einer Look-Up Tabelle umgerechnet werden.
- Durch den Tausch der Messwert-csv-Datei ist es möglich das Programm auf einen anderen Wertebereich festzulegen oder einen andere Blackbox zu verwenden..
- Zwischen den Datenpunkten aus der Look-Up Tabelle soll linear interpoliert werden, um aus dem Sensorwert den korrekten Abstand zu berechnen.

Siehe folgende Skizze:



Look-Up Tabelle:

1. Der Sensorwert wird eingelesen.
2. Die nächsten zwei Sensorwerte aus der Look-Up Tabelle werden gesucht.
3. Zwischen diesen beiden Werten wird linear interpoliert, um den realen Abstand zu berechnen.
4. Der berechnete reale Abstand wird ausgegeben.

### Labortask Beschreibung

1. US-Sensor in einer festen Position zu einem Objekt abstellen
2. Das messen des Abstands vom Sensor zur Fläche wird mit einem Tastendruck auf Enter ausgelöst. Dieser Wert wird als gelesener Abstandswert (in Meter) csv-Datei **messung\_2.csv** gespeichert werden. (siehe nachfolgende Tabelle)
3. In der nachfolgenden Spalte den interpolierten Wert eintragen (in Meter)
4. Mehrere Messungen durchführen ( minimum 20 ) . Abstand ist diesmal willkürlich ausgewählt
5. Der Modus soll so lange laufen, bis ein spezieller Tastendruck (z.B. "q") erfolgt. das Programm springt automatisch zum Menue über
6. Option: Plot mit Soll und Ist Wert in Excel programieren und graphisch ausgeben.

Messung Nr.	ABSTAND	Abweichung
	Vom Sensor Wert	Interpolierter Wert
	über USB eingelesen	Berechnet
1		
2		
...		
n-1		
n		

### Aufgabe 3

Automatisch messen. Modus "AUTO"

### Labortask Beschreibung

1. für ca. 10 Sekunden eine/n Messung/Bewegungsablauf starten
2. Zeitstempel der erfassten Werte und die Werte in einer .csv Tabelle **messung\_3.csv** speichern
3. Visualisieren der Höhenkarte anhand der Zeitstempel - in C

4. Ist die Zeit zwischen zwei Zeitstempeln immer gleich?
5. Wenn nein die max und min sowie auch die durchnits Abtast Zeit berechnen.
6. Im Windows mehrere Task's starten (youtube Video, Maus bewegen usw. ) und gleiche Mesung noch mal durchführen.
7. Wie verändern sich jetzt die Werte der Zeitstempel? Speichern Sie die Daten in einer neuen .csv **messung\_4.csv** Tabelle "Messung Nr. 3"

## Aufgabe 4

Daten Filtern. Modus "FILTER"

- Filter Sie die in **messwert3.csv** gespeicherten Abstands Werte.
- Programmieren Sie ein Mittelwert Filter über 3 Werte z.B.  $u(m) = (\text{SUMME}(y(m-1):y(m+1)))/3$
- Ploten Sie beide Werte Reihen in einem Diagramm und die dazugehörige Differenz zwischen den beiden Wertereihen in einem neuem Diagramm

## Aufgabe 5

Fun: Make it a game. Modus "FUN"

### Labortask Beschreibung

1. ASCII Art level durch Höhenkarte. Das C Programm gibt eine Höhenkarte auf der Kommandozeile aus.
2. U-Boot mit W - up, S - down gesteuert durch die Höhenkarte navigieren.
3. von links nach rechts fährt das Uboot automatisch.
4. Beenden Sie das Spiel wenn das U-Boot auf Grund läuft (also mit der Höhenkarte kollidiert).

## Abgabe Ausarbeitung

- Laden Sie auf Moodle folgende Dateien hoch
- 4x .csv Dateien ( messung\_1.csv, messung\_2.csv, messung\_3.csv, messung\_4.csv )
- C Programm
- Screenshot der ausgegebenen grafik in .png oder .jpeg Format

Abgabefrist wird im Labor bekannt gegeben

## Spielregeln

- Labortermine sind Pflichttermine.
- Ausarbeitung daheim ist erlaubt.
- Team Work, max 3 Teildehmer ist erlaubt. Allerdings ist ein gesamt Kenntnisstand vom jedem Teilnehmer erfragt.

## Anhang Installationsvorgehen zur Verwendung von USB serial Kommunikation auf Windows host Systemen mit WSL 2 und devcontainer

Accessing a USB Serial Device in a VS Code Dev Container on WSL 2

This tutorial explains how to use a USB-connected serial adapter (for example an FTDI or debug probe) from within a Visual Studio Code development container. The scenario assumes you are running Docker on Windows via WSL 2, have a project that already uses a Dev Container with CMake and GCC, and you want to connect to a device that appears on Windows as COM4. The solution uses usbipd-win to share the USB device with WSL 2 and passes it through to the container using Docker's `--device` flag.

### Update WSL 2 and install usbipd-win on Windows

Ensure WSL is up-to-date. Open a PowerShell terminal and run:

```
``powershell
wsl --update
``
```

The Microsoft command-line blog notes that WSL needs to be on a 5.10 kernel or newer and suggests updating before using USB/IP.

### Install usbipd-win.

Use winget<sup>1001</sup> to install the latest release. In an elevated PowerShell prompt run:

```
winget install --id=Microsoft.usbipd-win
```

[Installing usbipd-win registers a service on Windows that shares local USB devices to WSL.](#)

### List available USB devices.

From an elevated command prompt run:

```
usbipd wsl list
```

This command prints each device's **BUSID**, vendor/product IDs and current state. Note the **BUSID** corresponding to the serial adapter you want to share.

## Bind and attach the device.

First, make the device sharable with `usbipd bind --busid=<BUSID>`. Then attach it to WSL using:

```
usbipd wsl attach --busid <BUSID>
```

The usbipd documentation explains that bind is persistent, whereas attach connects the device to the running WSL VM.

## Install USB/IP client tools inside WSL

In order for WSL to communicate with the usbipd service, install the Linux USB/IP user-space tools:

```
bashsudo apt update # Install the appropriate linux-tools package for your kernel and the hwdatabasudo apt install linux-tools-$(uname -r) hwdatabusbutils # Register the usbip client binary if your distribution does not do so automaticallysudo update-alternatives --install /usr/local/bin/usbip usbip \    /usr/lib/linux-tools-$(uname -r)/usbip 20
```

Microsoft's guidance demonstrates installing `linux-tools` and `hwdatab`, then registering `usbip` with `update-alternatives`. The `usbutils` package provides the `lsusb` utility used later.

After attaching the device from Windows, verify that WSL can see it:

```
``bash lsusb
```

You should see an entry corresponding to your serial adapter. Depending on the driver, it will also create a character device such as `/dev/ttyUSB0` or `/dev/ttyACM0`. You can find it with:

```
```bash
ls -l /dev/ttyUSB* /dev/ttyACM* 2>/dev/null
```

## Pass the USB device through to the Dev Container

Docker cannot automatically see devices attached to the WSL VM. You must tell Docker to map the device into the container. The following modifications are required.

### Dockerfile

A sample Dockerfile is provided in this repository. It uses the `mcr.microsoft.com/devcontainers/base:ubuntu-22.04` base image, installs the compiler toolchain and USB utilities, and adds the unprivileged `vscode` user to the `dialout` group so it can open serial ports:

```
FROM mcr.microsoft.com/devcontainers/cpp:1-debian-12

RUN apt-get update
RUN apt-get install -y sqlite3 libsqlite3-dev ninja-build

# New entries

USER root

RUN apt-get install -y --no-install-recommends usbutils minicom

RUN usermod -aG dialout vscode

USER vscode
WORKDIR /workspaces
```

This configuration installs `usbutils` and `minicom` so you can inspect USB buses and open serial ports within the container. Adding the `vscode` user to the `dialout` group avoids running commands as root.

### devcontainer.json

Extend the json:

```
"runArgs": [
  "--privileged",
```

```
    "--device=/dev/bus/usb"  
  ],  
  "remoteUser": "vscode",  
  "postCreateCommand": "sudo usermod -a -G dialout $(whoami) && ls -l /dev/ttyUSB* || true",  
  "workspaceFolder": "/workspaces/${localWorkspaceFolderBasename}"
```

If you know the exact device file (/dev/ttyUSB0 or similar) you may instead map only that device using "--device=/dev/ttyUSB0" for finer control. The example above maps all USB devices.