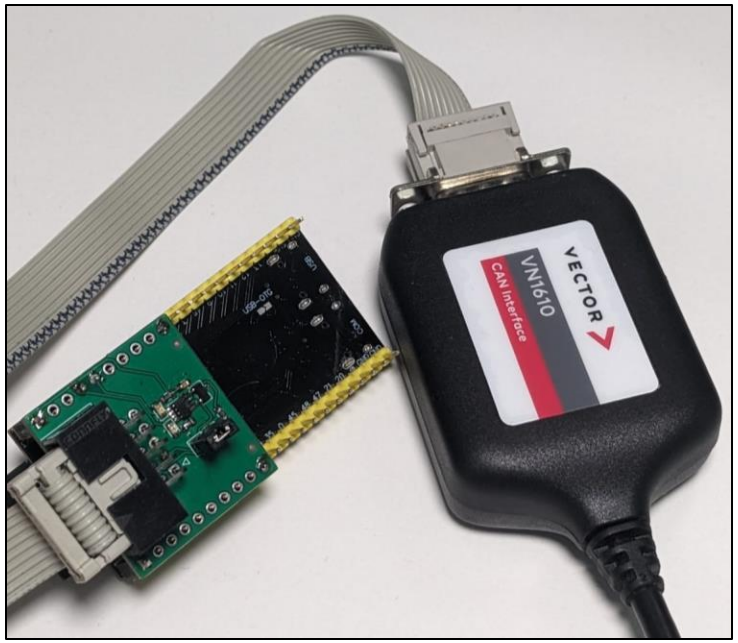
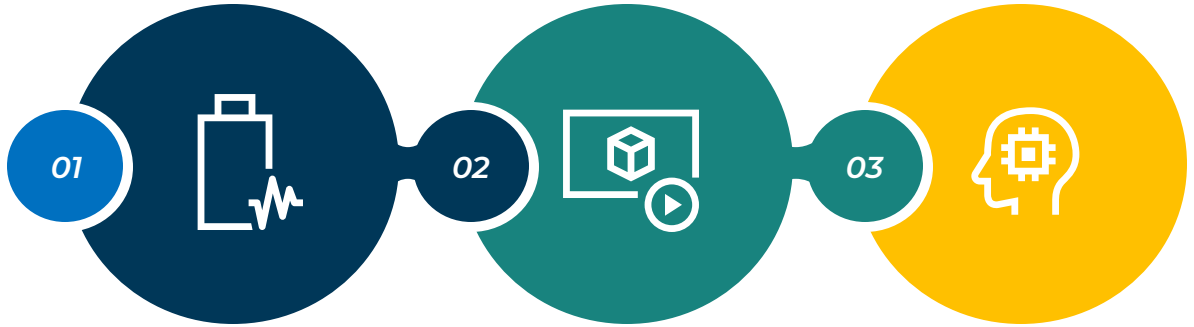
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Maschinenbau und
Mechatronik

CAN –
Interface für ESP 32



Agenda

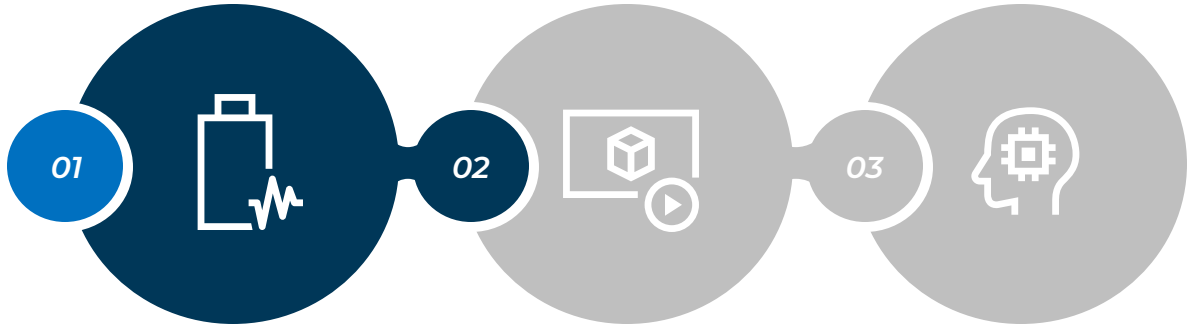


Hardware

Software

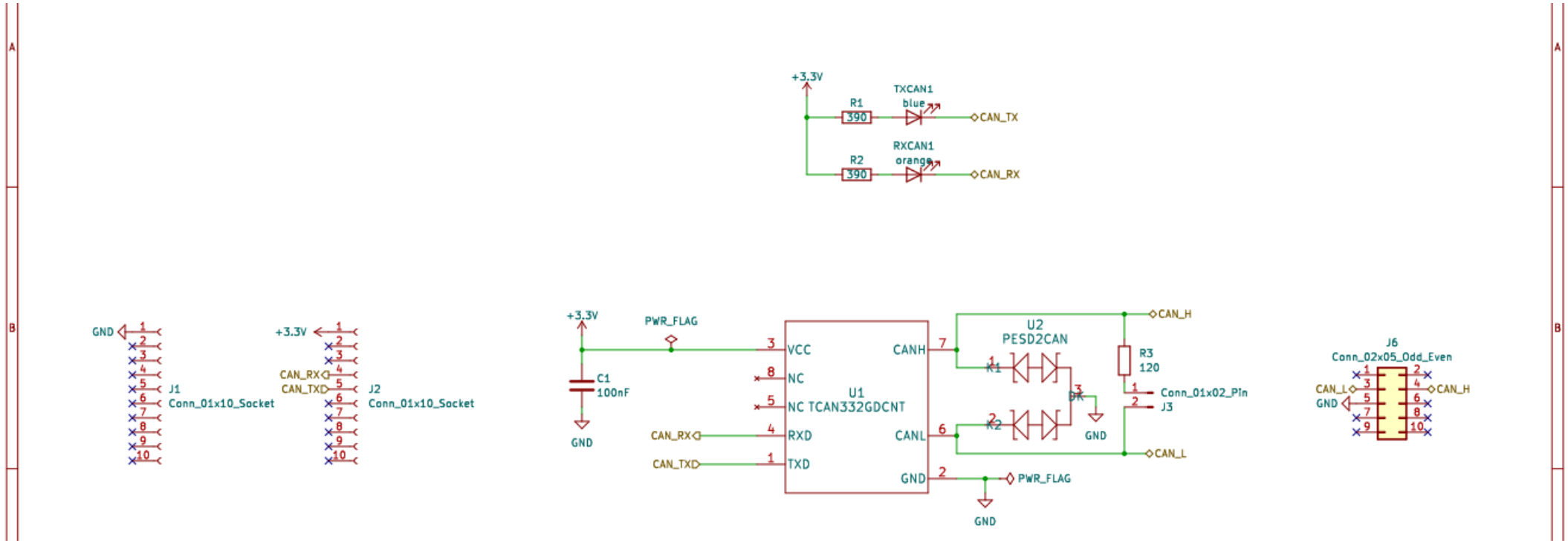
QT-Interface

Agenda

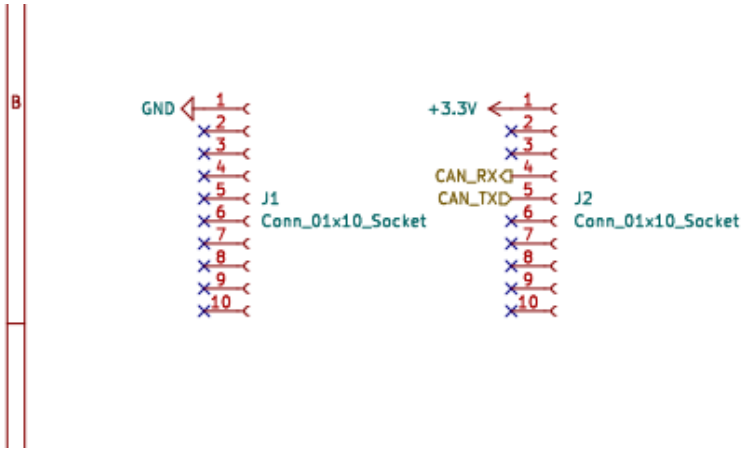


Hardware Software QT-Interface

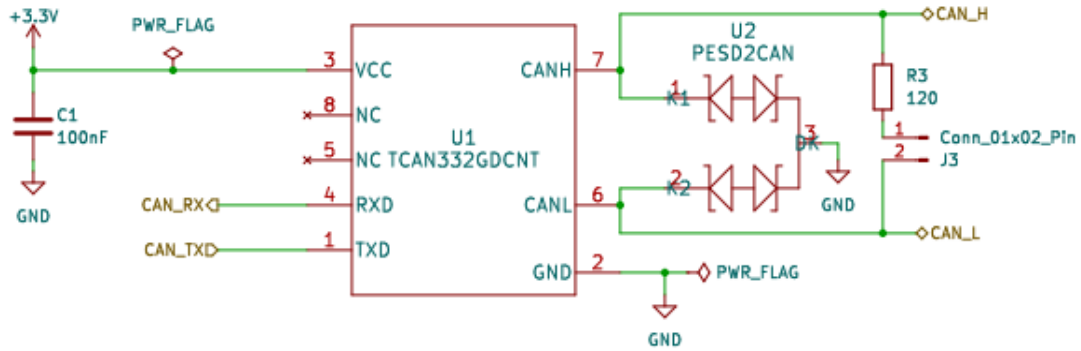
Hardware


+3.3V
TXCAN1
R1
390
blue
CAN_TX
RXCAN1
R2
390
orange
CAN_RX
GND
J1
Conn_01x10_Socket
+3.3V
CAN_RX
CAN_TX
J2
Conn_01x10_Socket
+3.3V
PWR_FLAG
C1
100nF
GND
VCC
NC
NC TCAN332GDCNT
RXD
TXD
CANH
CANL
GND
CAN_RX
CAN_TX
U2
PESD2CAN
CAN_H
CAN_L
GND
R3
120
Conn_01x02_Pin
J3
PWR_FLAG
GND
J6
Conn_02x05_Odd_Even
CAN_L
GND
CAN_H

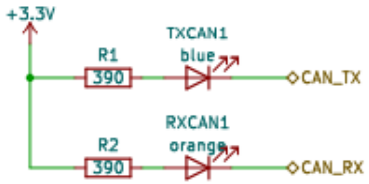
Hardware


B
GND
1
2
3
4
5
6
7
8
9
10
J1
Conn_01x10_Socket
+3.3V
CAN_RX
CAN_TX
J2
Conn_01x10_Socket

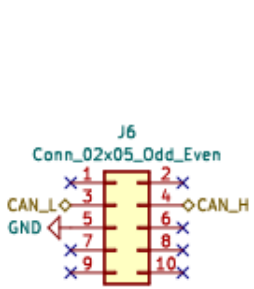
Hardware


+3.3V
PWR_FLAG
C1
100nF
GND
3
VCC
8
NC
5
NC
4
RXD
1
TXD
CAN_RX
CAN_TX
U1
TCAN332GDCNT
CANH
7
CANL
6
GND
2
PWR_FLAG
GND
U2
PESD2CAN
K1
K2
GND
CAN_H
R3
120
Conn_01x02_Pin
J3
CAN_L

Hardware



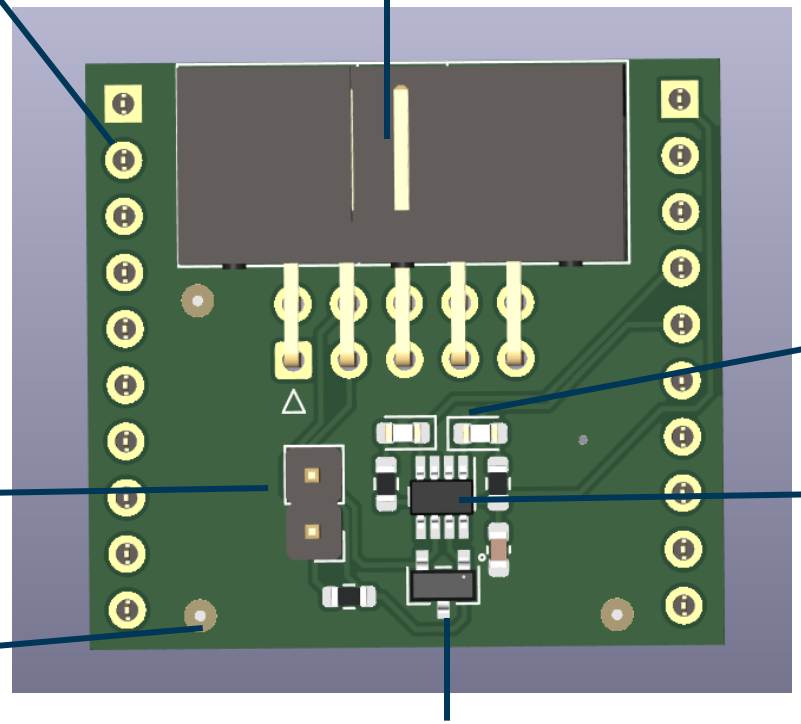
Hardware



Hardware

Connector
10-Pin IDC

Socket
Interface for ESP32



LEDS
For visual debugging

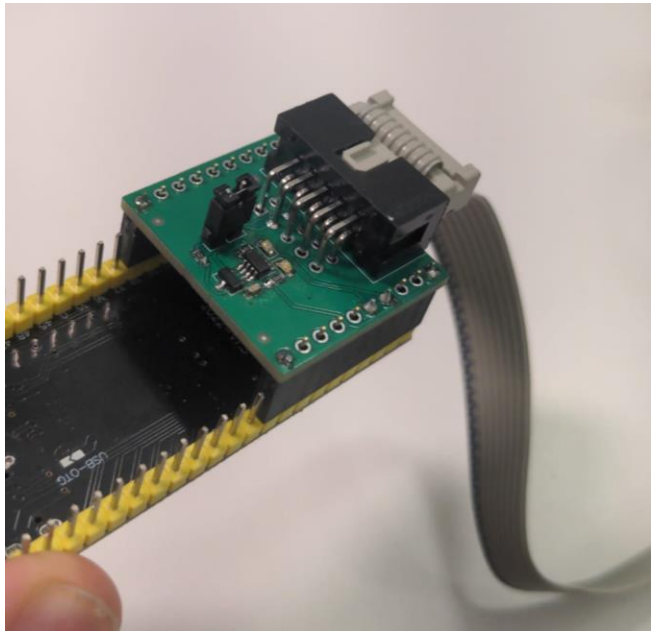
TCAN332GDCNT
CAN transceiver

Jumper
Selectable End-Resistor

Fiducials
For PCBA

PESD2CAN
ESD and transient protection

Hardware

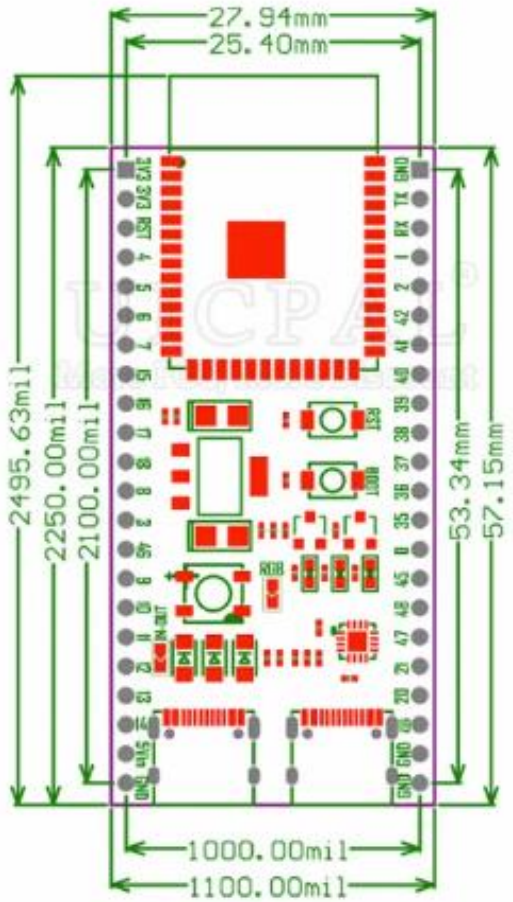


JLCPCB

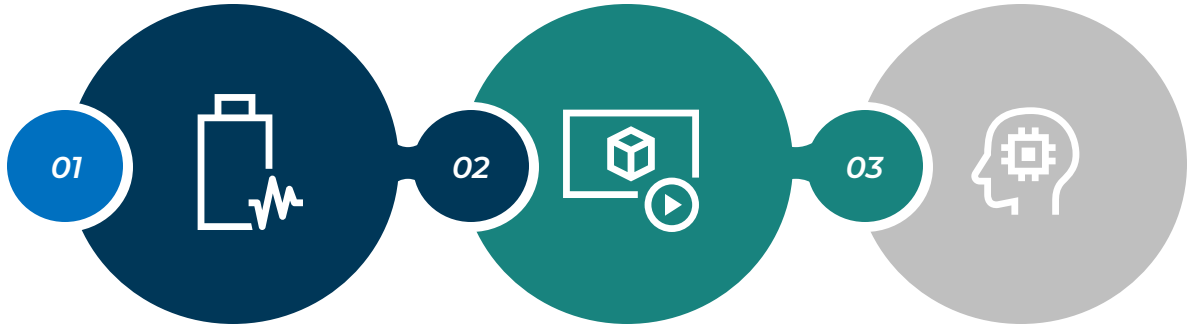
Hardware

Problems during Assembly

- Aliexpress board is a bit wider then original from Espressif
- Cold solder joint at transceiver



Agenda



Hardware **Software** QT-Interface

Software

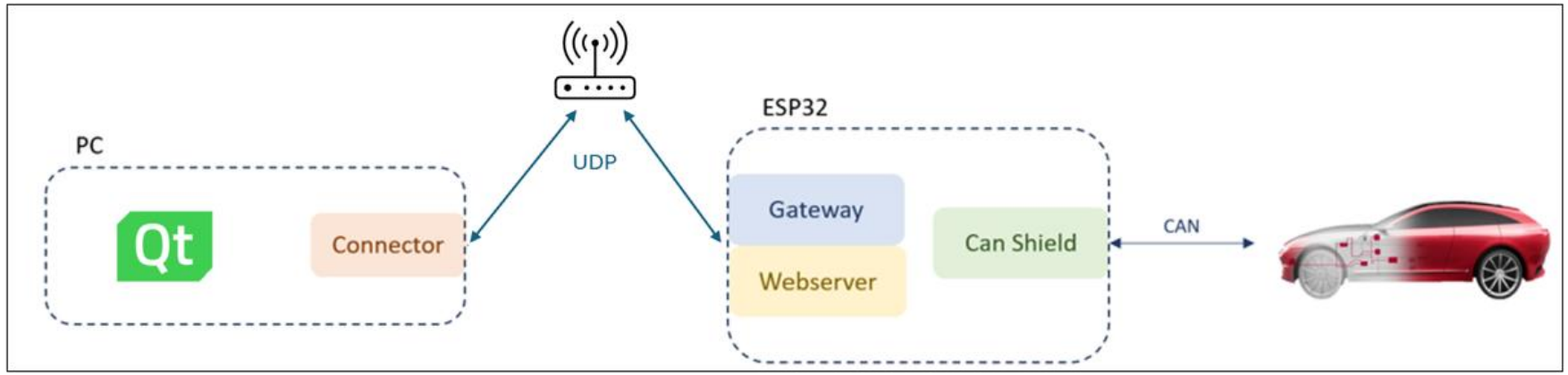
Requirements:

functional

- Connection to PC
- Webserver to configure the interface
- Communication from CAN-Device to PC
- Communication from PC to CAN-Device

technical

- Windows compatible
- Code should be tested
- Code should be maintainable and expandable



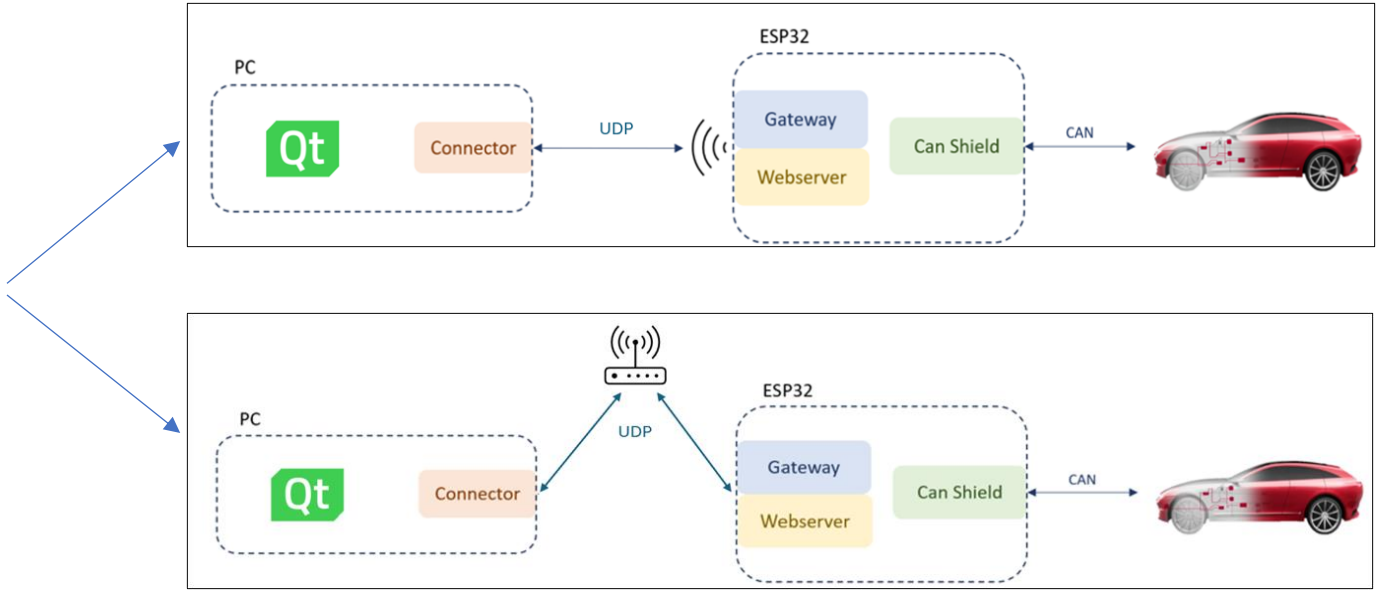
Software

Requirements:

functional

- **Connection to PC**
  - over a predefined network
  - with the ESP32 serving as a hotspot
- Webserver to configure the interface
- Communication from CAN-Device to PC
- Communication from PC to CAN-Device

```
// WiFi and Hotspot settings
//Make sure you adjust the ssid and password in your Secrets.h to your WIFI
//mode = 1: connection with a predefined WIFI
//mode = 2: ESP32 as a WLAN-Access-Point
int mode = 1;
const char *wifi_ssid = WIFI_SSID;
const char *wifi_password = WIFI_PASSWORD;

const char *hotspot_ssid = Hotspot_SSID;
const char *hotspot_password = Hotspot_PASSWORD;
IPAddress remoteIp(192, 168, 42, 182);
```



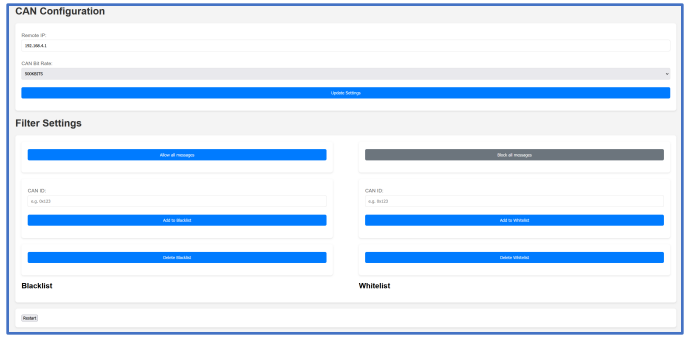
Software

Requirements:

functional

- Connection to PC
- **Webserver to configure the interface**
  - Configure Remote IP
  - Adjusting the Bit Rate
  - Filter Options
- Communication from CAN-Device to PC
- Communication from PC to CAN-Device

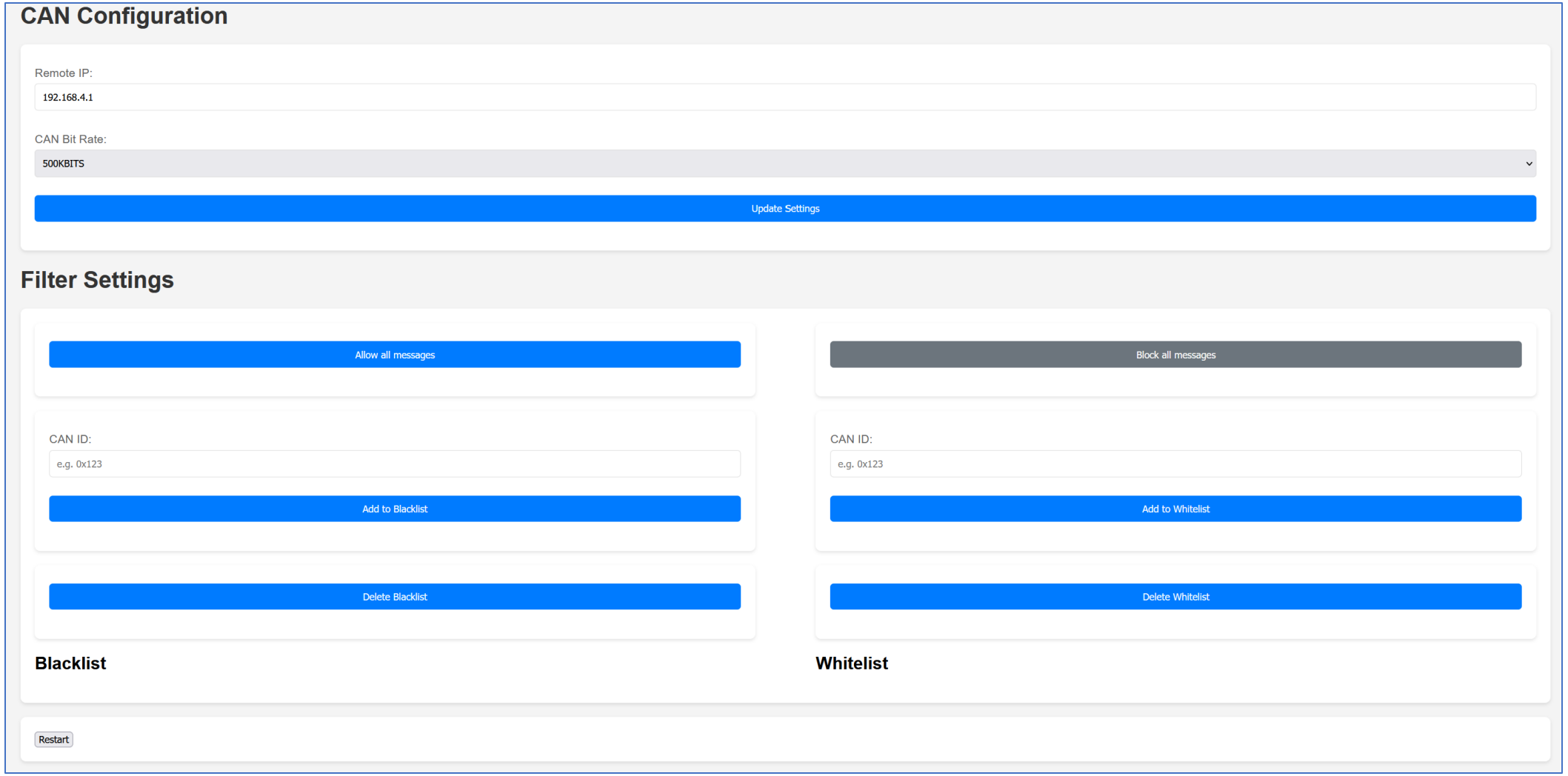
Webserver Frontend → Webserver Backend → UDP Communicator




```
void WebServerManager::begin() {
    server.on("/", HTTP_GET, [this](AsyncWebServerRequest *request) {
        request->send(200, "text/html", getHtmlContent());
    });

    //Update the Settings
    server.on("/update", HTTP_POST, [this](AsyncWebServerRequest *request) {
        this->handleUpdate(request);
    });

    //Restart the ESP32
    server.on("/restart", HTTP_GET, [this](AsyncWebServerRequest *request) {
        request->send(200, "text/html", "ESP Restarted!<br><a href='/'>Go Back</a>");
        delay(1000); // Kurze Verzögerung
        ESP.restart();
    });

    //Allow all messages
    server.on("/allowAll", HTTP_GET, [this](AsyncWebServerRequest *request) {
        udpCommunicator.setAllowAll(true);
        request->send(200, "text/html", "All messages are allowed, except Blacklist.<br><a href='/'>Go Back</a>");
    });
```

- updateRemoteIP
- addToWhitelist
- clearWhitelist
- addToBlacklist
- clearBlacklist
- setAllowAll

Software


CAN Configuration
Remote IP:
192.168.4.1
CAN Bit Rate:
500KBITS
Update Settings
Filter Settings
Allow all messages
Block all messages
CAN ID:
e.g. 0x123
Add to Blacklist
CAN ID:
e.g. 0x123
Add to Whitelist
Delete Blacklist
Delete Whitelist
Blacklist
Whitelist
Restart


1
2
3

Software

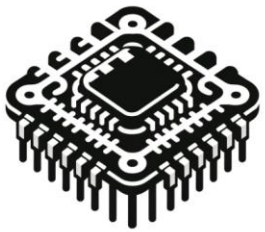
Requirements:

functional

- Connection to PC
- Webserver to configure the interface
- **Communication from CAN-Device to PC**
  - Receive CAN Message
  - Encode CAN Message into CANeth
  - Send CANeth to PC via UDP
- Communication from PC to CAN-Device

ESP32

CAN-Message



- Receive CAN Message
- Encode CAN Message into CANeth – Message
- Send CANeth Message via UDP

CANeth - Message

CANeth – Message

- Protocol Identifier (ISO11898) – 8 byte
- Protocol Version – 1 byte
- Frame Count – 1 byte
- CAN ID – 4 byte
- Data length – 1 byte
- Data – 8 byte
- Extended identifier Flag – 1 byte
- RTR Flag – 1 byte

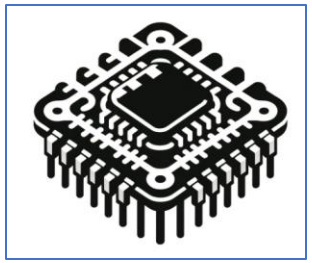
Software

Requirements:

functional

- Connection to PC
- Webserver to configure the interface
- Communication from CAN-Device to PC
- **Communication from PC to CAN-Device**
  - Receive CANeth Message
  - Encode CANeth Message into CAN- Message
  - Send CAN-Message to CAN-Device

ESP32



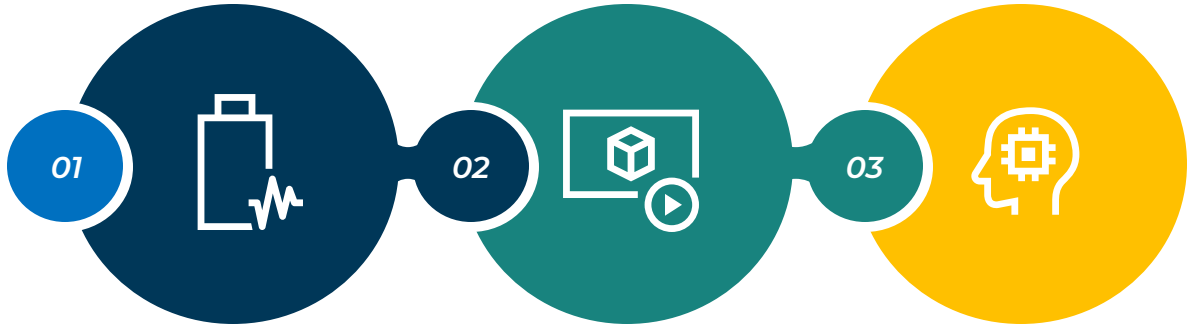
- Receive CANeth Message
- Decode CANeth Message into CAN – Message
- Send CAN Message via **UDP**

CAN-Message

CANeth - Message

User **D**atagram **P**rotocol
+ Speed
+ Simplicity
- No Error corretion
- No congestion control

Agenda



Hardware

Software

QT-Interface

QT-Interface

Requirements:

functional
- Visualization of received CANeth messages
- Logging of received CANeth messages
- Sending CANeth messages
- Filtering CANeth messages
- A user interface (UI) shall enable the functions

technical
- Independent of a python interpreter
- Windows compatible
- Code should be tested
- Code should be maintainable and expandable

Implementation of technical requirements:

- Executable Application based on Python 3.9 compiled with PyInstaller for Windows
  - Backend with default libraries
  - Frontend with PyQT5 library
- Testing of Function and Class with edge case input variable
- Using github and providing open source code

QT-Interface Architecture

frontend

displaying information, starting multithreads

.py and .ui file

user input

display data

backend

data storage

CANeth filter

Sending CANeth

receiving CANeth

data type conversion

Windows socket

.py file

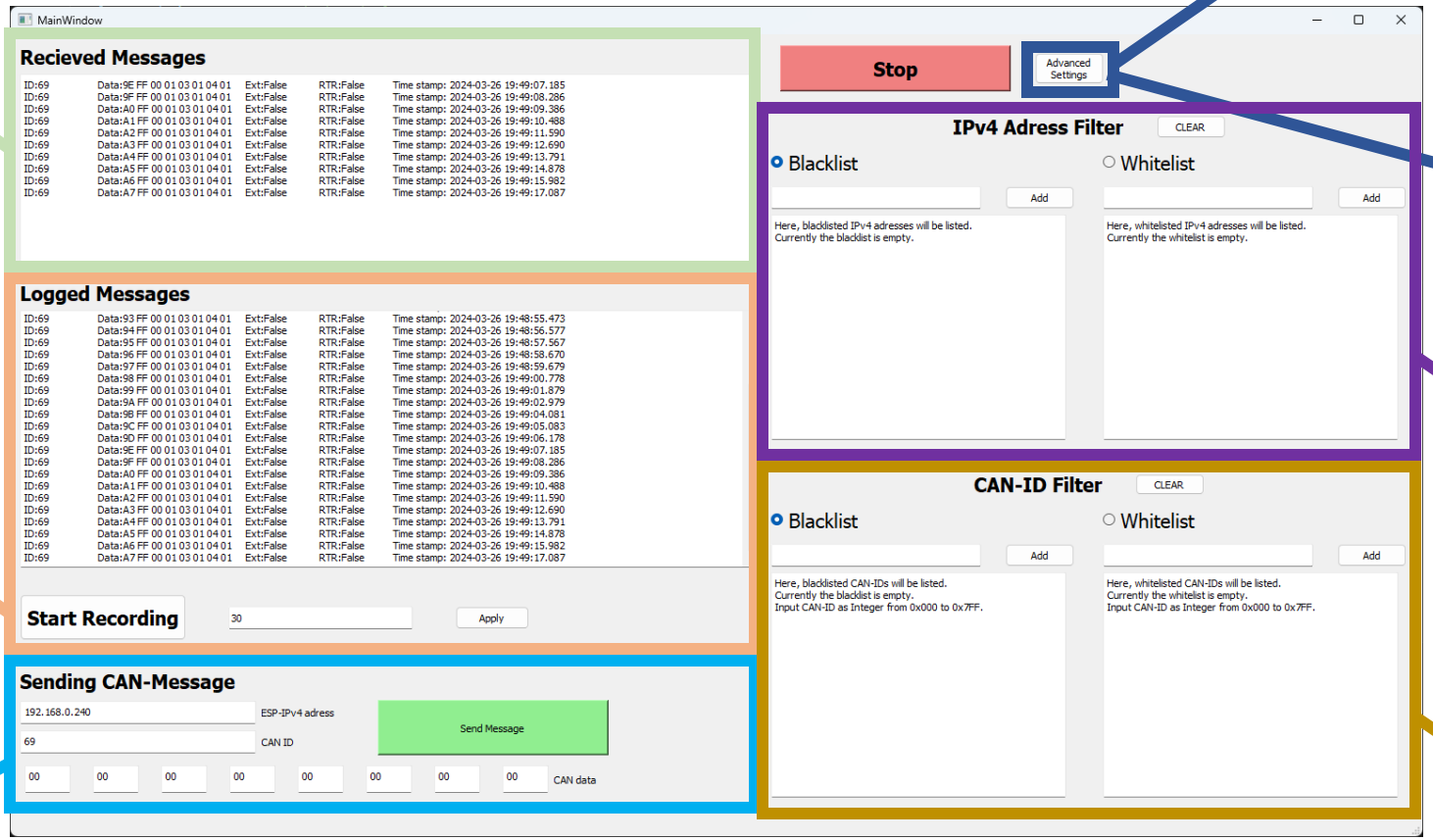
ConnectorApp.exe

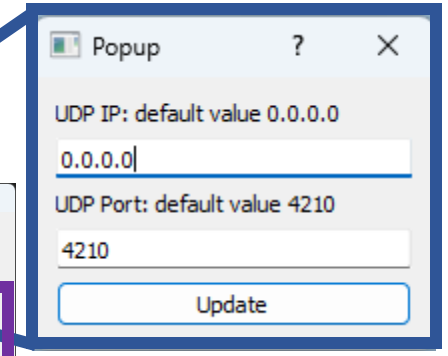
QT-Interface: UI functionalities

Live feed of received, filtered CANeth messages

Display area of received, filtered, logged CANeth messages

Sending single CANeth messages





Socket settings

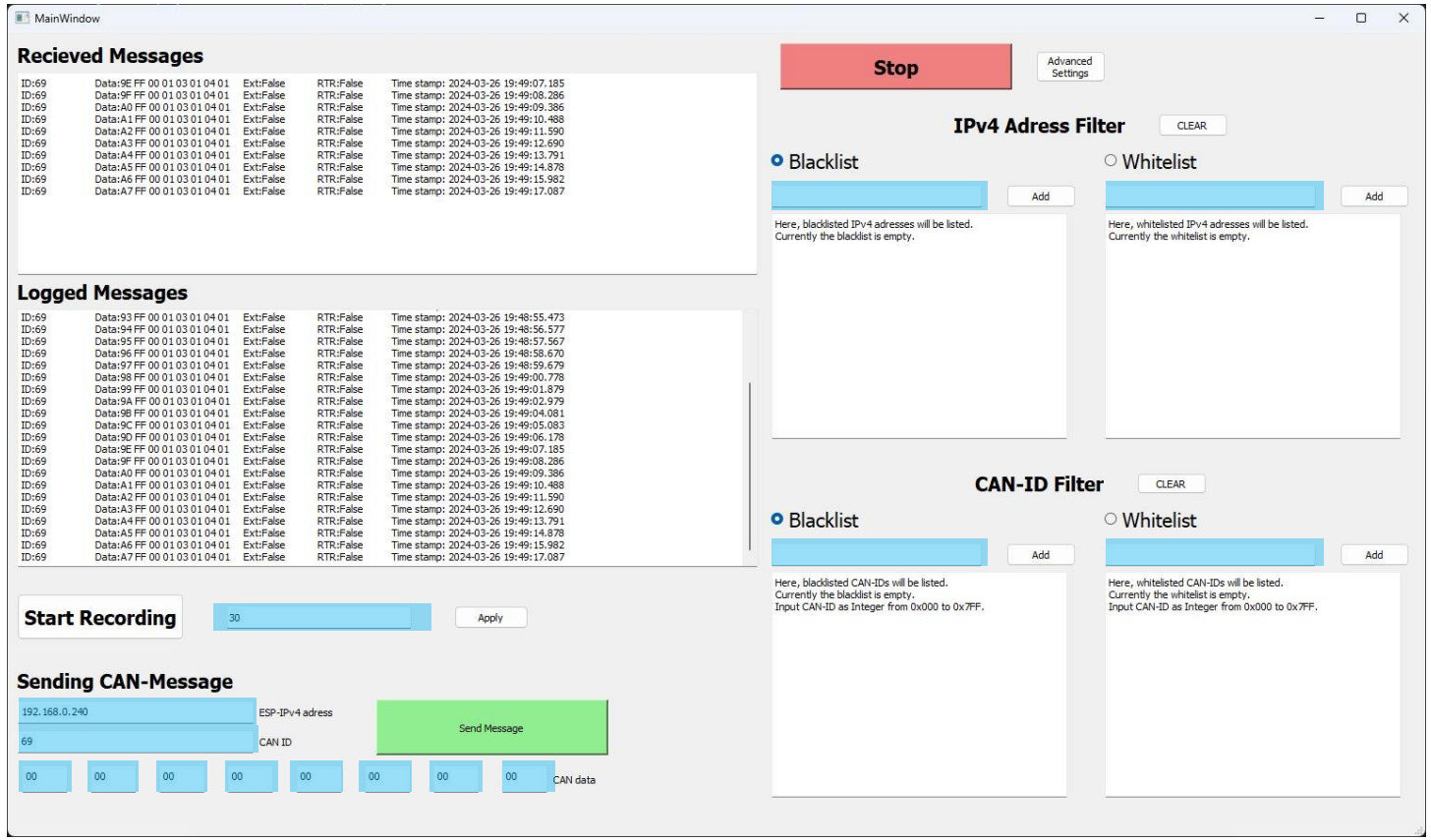
UDP IP **0.0.0.0** allows to receive messages on all IPv4 addresses

IPv4 address Filter

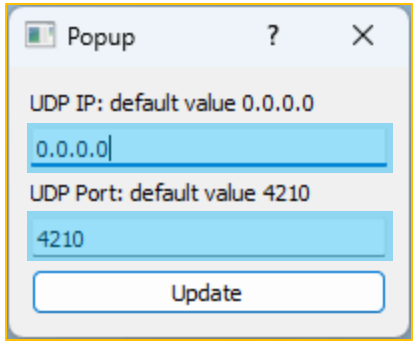
Only useful if multiple ESP32s are connected to an existing WiFi

CAN-ID Filter

QT-Interface: UI functionalities
Limitations

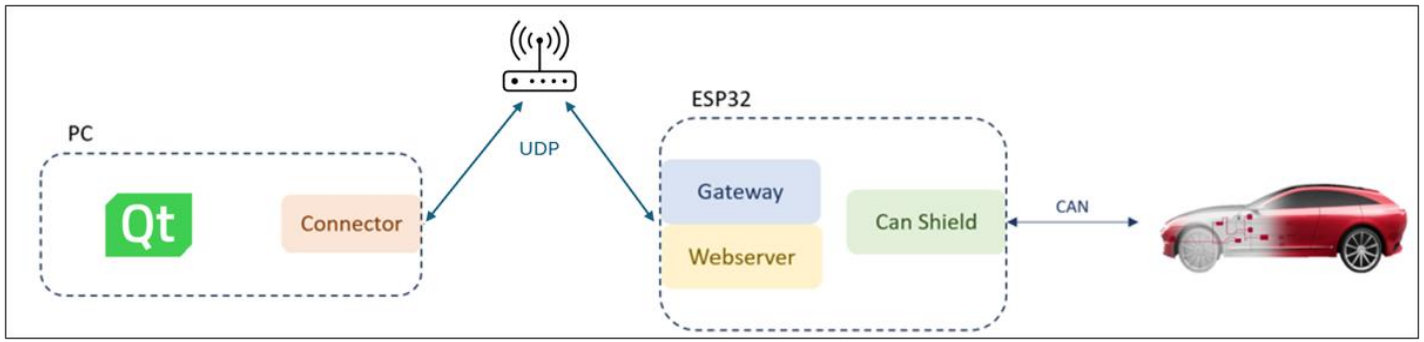


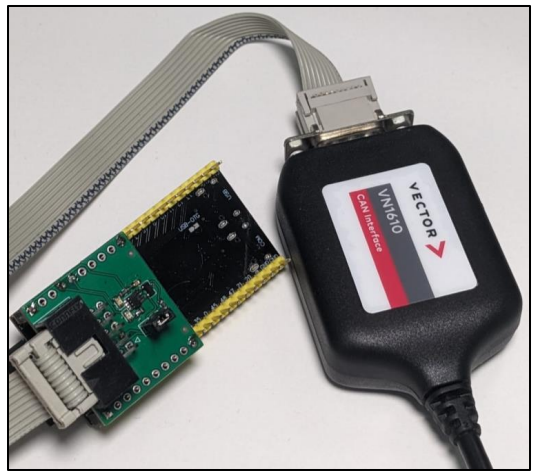
- All LineEdits do not check for input errors



- Logged messages can not be saved
- Blacklist and Whitelist of Filters can only be cleared completely
- CANeth messages with only the standard 16 hexbit can be sent once a time
- Display resolution and ratio compromise application layout

Tools: CAN Device





- CAN Device simulated via VECTOR box VN1610
- CAN Device sends changing CAN messages to the CAN shield while listening to CAN messages

CAN ID: 0x45

CAN message: data_change 0xFF 0x00 0x01 0x03 0x01 0x04 0x01

data_change: 0x00 to 0xFF

Ext: False

RTR: False

Sent on configured Application Channel of the VN1610 with **bitrate of 500 000**

Tools: CAN Device

- CAN Device simulated via VECTOR VN1610
- Requirements:
  - Python-can library — https://anaconda.org/conda-forge/python-can
    https://pypi.org/project/python-can/
  - Vector XL Driver Library — https://www.vector.com/int/en/download/xl-driver-library-203014/
  - Python Interpreter, e.g. VS Code
- VECTOR box needs to be configured with an application channel, so the py-script can access it
  - Use the Vector – Hardware Manger from the Vector XL Driver Library download to do so

Configured Application Channels

Application	Application Channel	Device	Device Channel
CANalyzer	CAN 1	VN1610 [1]	Channel 1

```
if __name__ == "__main__":
    with can.Bus(interface='vector', app_name='CANalyzer', channel=0, bitrate=500000) as bus:
        listen_and_send(bus)
```

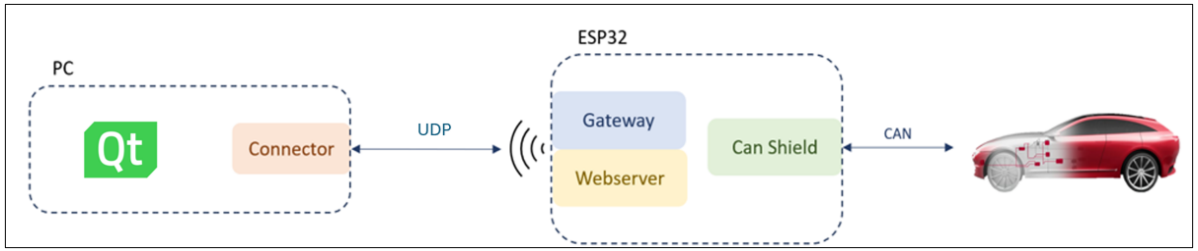
- If the vxlapi64.dll can not be found while running the canDevice.py, the file path hast to be hardcoded in to the python-can library in the xldriver.py file

C:\Users*#YOURUSERNANME#*\anaconda3\Lib\site-packages\can\interfaces\vector\xldriver.py

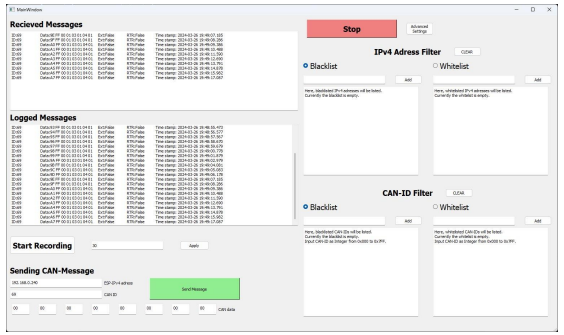
C:\Users*#YOURUSERNAME#*\AppData\Local\Programs\Python\Python311\Lib\site-packages\can\interfaces\vector\xldriver.py

```
# Load Windows DLL
DLL_NAME = "vxlapi64" if platform.architecture()[0] == "64bit" else "vxlapi"
#_xlapi_dll = ctypes.windll.LoadLibrary(DLL_NAME) # original w
_xlapi_dll = ctypes.windll.LoadLibrary('C:\\Users\\#YOURUSERNAME#\\FuE_2_esp_can\\ESP32-CAN-Shield\\Tools\\vxlapi64.dll')
```

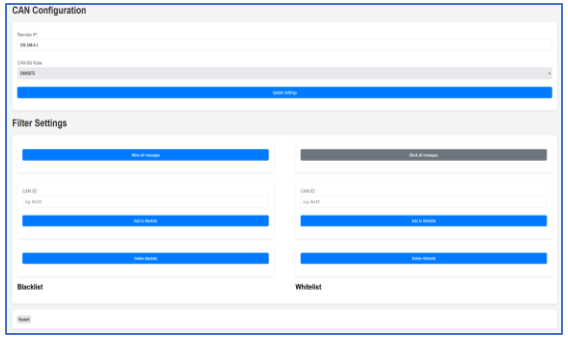
Conclusion & Outlook



Software

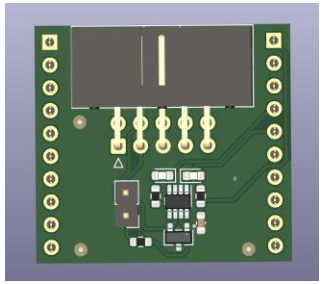


- Improvement of known Limitations
- Reverse engineering of CAN messages
- Diagnostic programs



- Improvement redundant functions

Hardware



- Power supply from CAN interface
- Power supply from battery

Comprehensive documentation in text and video