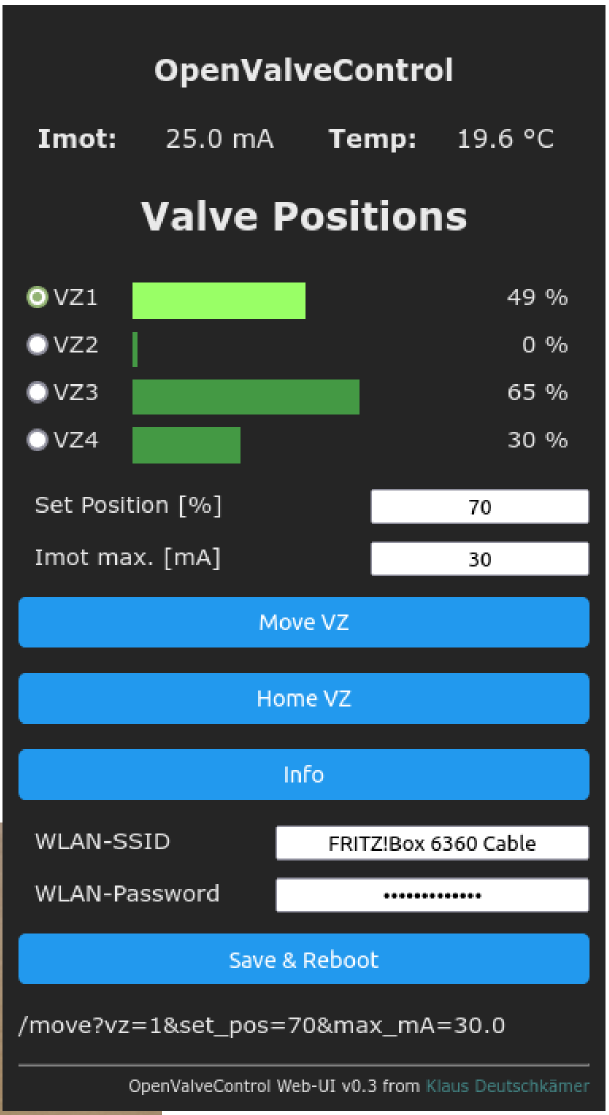
OpenValveControl

User Manual (DRAFT)



OpenValveControl

Imot: 25.0 mA **Temp:** 19.6 °C

Valve Positions

● VZ1	49 %
○ VZ2	0 %
○ VZ3	65 %
○ VZ4	30 %
Set Position [%]	70
Imot max. [mA]	30

WLAN-SSID	FRITZ!Box 6360 Cable
WLAN-Password	•••••••••••••

```
/move?vz=1&set_pos=70&max_mA=30.0
```

OpenValveControl Web-UI v0.3 from Klaus Deutschkämer

Content

1 Initial Setup

Prerequisites

- The board is assembled with all the required components and
- The PIC controller has been flashed (e.g. with a PICkit4) with the latest firmware.

1.1 Flashing the "ESP8266 D1-mini" for the first time

There are many ways to flash the ESP for the first time. They all involve connecting the board via the USB interface. The appropriate driver for the USB interface IC installed on the D1 mini may need to be installed first.

Under Linux, the command 'lsusb' can be used to check this. The name of the USB port can be found in the /dev/ directory:

```
~$ lsusb
Bus 002 Device 006: ID 1a86:7523 QinHeng Electronics CH340 serial converter
...

~$ ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 Dez 15 14:00 /dev/ttyUSB0
~$
```

1.1.1 Arduino IDE (2.2.1)

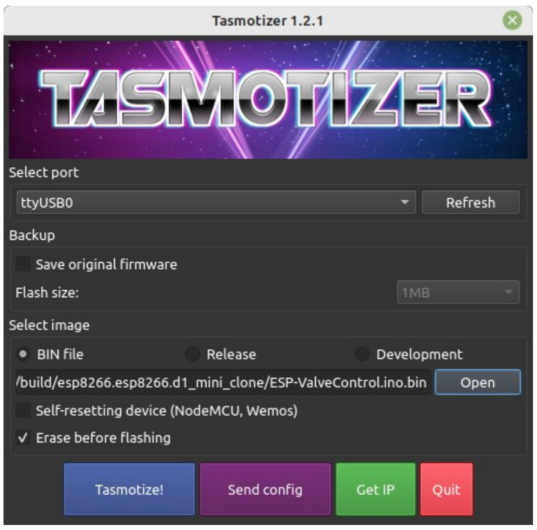
Board: LOLIN(WEMOS) D1 mini
FlashSize: 4 MB (FS: 1MB, OTA: ~1019KB)

1.1.2 tasmotizer

"Tasmotizer" is a Python application and must therefore be started via Python (currently python3). The tool *tasmotizer.py* can be downloaded from github, the path in the following screenshot must be adapted to the download location:

```
~$ python3 .local/bin/tasmotizer.py
esptool.py v2.8
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Erasing flash (this may take a while)...
Chip erase completed successfully in 3.1s
Compressed 383456 bytes to 276588...
Wrote 383456 bytes (276588 compressed) at
0x00000000 in 24.4 seconds (effective 126.0
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```



Select port: ttyUSB0
Save original firmware
Flash size: 1MB
Select image: [✓ BIN file] Release Development
/build/esp8266.esp8266.d1_mini_clone/ESP-ValveControl.ino.bin
Self-resetting device (NodeMCU, Wemos)
✓ Erase before flashing

Figure 1: User interface of „Tasmotizer“ (see https://github.com/tasmota/)

In the graphical user interface, simply select the USB port and the binary program (ESP-ValveControl.ino.bin) with "Open" (navigate to the storage location) and press "Tasmotize!". The rest happens automatically.

1.1.3 Via command line using "esptool.py"

The Python tool "esptool.py" is usually part of the Arduino IDE, but can also be downloaded and installed from "Espressiv" if Phyton is not already installed on the computer. The path information for esptool.py and the ESP-ValveControl binary file must of course be adapted again:

```
~$ python3 ~/.arduino15/packages/esp8266/hardware/esp8266/3.1.2/tools/esptool/esptool.py -p /dev/ttyUSB0
write_flash 0x00000 ~/.git/OpenValveControl/ESP-ValveControl/build/esp8266.esp8266.d1_mini_clone/
ESP-ValveControl.ino.bin
esptool.py v3.0
Serial port /dev/ttyUSB0
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 24:a1:60:3a:bb:83
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 383456 bytes to 276588...
Wrote 383456 bytes (276588 compressed) at 0x00000000 in 24.4 seconds (effective 125.9
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
~$
```

Figure 2: Using esptool in linux console

1.2 WiFi Configuration

If the flashing was successful, the following message[1] should appear in the first two lines of the **OLED display** (as an indication that the ESP is working as an access point):

```
OVC-Access-Point
OVC-password
```

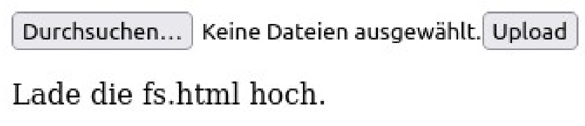
Shortly afterwards, the message changes and the access point’s IP address is displayed:

```
IP: 192.168.4.1
21.1 C          0.0 mA
```

1 If the OLED display does not work, the hardware and especially the connection (and polarity) of the OLED must be checked.

Now you can connect via WiFi to this access point "OVC-Access-Point" and the password "OVC-password" (case sensitive!) and enter the displayed IP address in a browser.

The following message should appear in your browser:


Durchsuchen… Keine Dateien ausgewählt. Upload

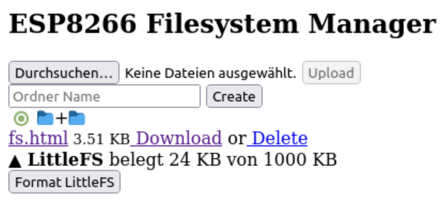
Lade die fs.html hoch.

http://192.168.4.1

Figure 3: Appearance in browser prior upload of filesystem user interface

The file "fs.html" must be selected from the ESP directory with "Browse" and then copied to the ESP file system by clicking the button "Upload".

The page then updates itself and shows the ESP8266 file manager with the content "fs.html", the file that has just been uploaded:


ESP8266 Filesystem Manager

Durchsuchen… Keine Dateien ausgewählt. Upload

Ordner Name Create

fs.html 3.51 KB Download or Delete

▲ LittleFS belegt 24 KB von 1000 KB

Format LittleFS

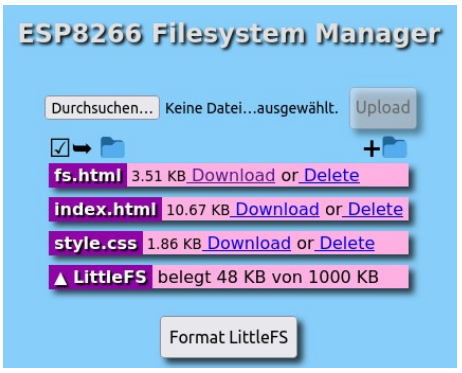
http://192.168.4.1/fs.html

Figure 4: Appearance in browser after upload of basic filesystem user interface (fs.html)

In the same way, we then upload the files

"**style.css**" (css formatting for fs.html) and
"**index.html**" (the ESP Valve Control user interface)

to the ESP file system:


ESP8266 Filesystem Manager

Durchsuchen… Keine Datei...ausgewählt. Upload

fs.html 3.51 KB Download or Delete

index.html 10.67 KB Download or Delete

style.css 1.86 KB Download or Delete

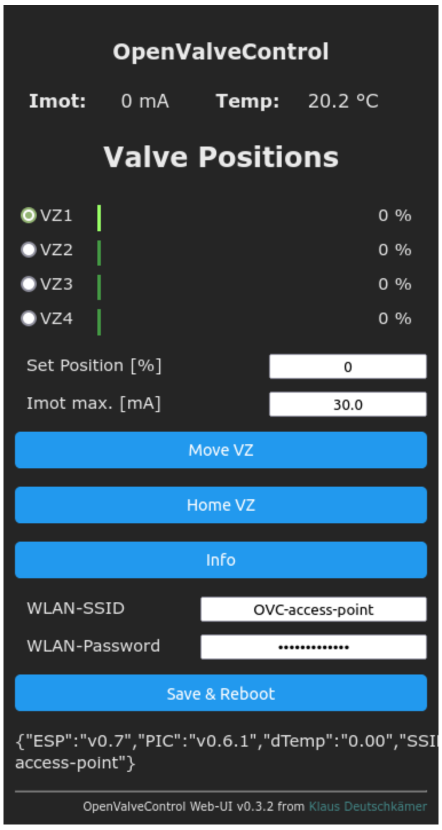
▲ LittleFS belegt 48 KB von 1000 KB

Format LittleFS

http://192.168.4.1/fs.html

Figure 5: Appearance in browser after upload of fs.html, stylesheet (style.css) and web-UI (index.html)

After entering the top URL (in our example: http://192.168.4.1) into your browser will display the graphical user interface of ESP-ValveControl (it is loaded from the ESP file system):



Imot: 0 mA Temp: 20.2 °C

VZ1	0 %
VZ2	0 %
VZ3	0 %
VZ4	0 %
Set Position [%]	0
Imot max. [mA]	30.0
WLAN-SSID	OVC-access-point
WLAN-Password	•••••••••••••

{"ESP":"v0.7","PIC":"v0.6.1","dTemp":"0.00","SSI
access-point"}

OpenValveControl Web-UI v0.3.2 from Klaus Deutschkämer

http://192.168.4.1 (IP of access point)

← Enter here the WiFi SSID of your home network,
← and the WiFi password.
← Press this button to save the settings.

Figure 6: OpenValveControl Graphical User Interface in Web-Browser

1.3 Reboot and Login to your home network

After the reboot, the ESP attempts to log into the home network with the data entered. If this is not successful within about 2 minutes, the access point is activated again.

i If problems occur, please check whether your router is configured so that only known devices are allowed to connect. Change this setting temporarily if necessary.

When restarting, the SSID you entered for your home network is displayed in the top line of the OLED. This line flashes while the connection has not yet been established.

The version of the ESP8266 software is displayed in the second line:

Example:

```
FRITZ!Box 6390 Cable
ESP v0.7
```

After successful login, the first row of the OLED now shows the IP address as assigned by your router. The second line shows the actual temperature[2] and motor current.

Example:

```
IP: 192.168.2.131
19.5 C         0.0 mA
```

You can now connect to the home network and enter the IP address displayed. The ESP-ValveControl graphical user interface should then be displayed again, but now with the SSID of your home network.

1.4 Additional settings (ovc.ini)

After entering and saving the WiFi parameters, an "ovc.ini" file is created in which the settings are saved permanently. Even after a firmware update, the data is normally still available unless you specify that the flash is to be completely erased (e.g. with Tasmotizer: "Erase before flashing").

However, a loss is not a problem, the installation described can be repeated at any time.

An example file **ovc.ini** is available in github in the folder "/ESP-ValveControl/".

```
## ovc.ini
# As default (or when it cannot connect within 2 minutes to the WiFi using the current
# settings), the ESP creates an access point with the credentials shown below.
# Please change the SSID and PSK to meet the settings of your WiFi, then upload the file into
# the ESP's file system, which can be accessed in your browser with the URI <local IP>/fs.html

# WLAN credentials
#SSID = Your SSID
#PSK = Your WiFi password
SSID = OVC-access-point
PSK = OVC-password

# MQTT: server (IP) and token to publish data every period (seconds) as "<mqtt_prefix>/status"
MQTT_HOST   = 192.168.2.72
MQTT_PREFIX = OVC-1
MQTT_PERIOD = 900

# ValveZone aliases (not used yet)
VZ1 = ESS
VZ2 = KÜ
VZ3 = BAD
VZ4 = WZ1

# Adjust temperature sensor
dTemp = -0.5
```

Figure 7: Example file **ovc.ini**

2 If no DS18B20 is connected, -128.2 C is displayed.

It means:

SSID	The name of your home network
PSK	Password of your home network
MQTT_HOST	IP-Adresse of your MQTT server.
MQTT_PREFIX	Prefix for the sent data (to differentiate between several identical devices (all data is sent with the MQTT token "<mqtt_prefix>/status").
MQTT_PERIOD	Time interval in seconds for sending the MQTT data ("publishing").
VZ1 VZ2 VZ3 VZ4	Alternative identifiers for the valve zones (not yet used)
dTemp	Adjustment value for the DS18B20 temperature sensor

As a further alternative, you can also download the „ovc.ini“ file to your computer and save it or change it with a text editor and add further information (e.g. for MQTT).

You can also duplicate the settings on several devices or skip entering the WiFi settings in the access point and upload a prepared „ovc.ini“ file using the Filesystem Manager. After the reboot, all settings are then immediately active.

1.5 Optional JavaScript file (customize.js)

Starting from WebUI v0.4 you can upload a JavaScript file ‚customize.js‘ to the ESP file system. This script can replace some default headers and labels by your own texts, e.g.

- Change title (e.g. 1st Floor, to differentiate between several devices)
- Replace valve identifiers (VZ1, VZ2,..) with more meaningful names.

The script is read at the end of <head> section in the WebUI’s „index.html“, but the function ***CustomizeLabels()*** is called once at the end of <body>, when the complete HTML document has been read by the browser.

The script also can be extended by your own instructions.

An example file **customize.js** is available in github in the folder "/ESP-ValveControl/".

```
/* customize.js
   Upload this script into the ESP filesystem.
   It replaces some default headers and labels by the given texts.
*/
function CustomizeLabels ()
{
  document.getElementById('header1').innerHTML = '1<sup>st</sup> Floor';   // -> OpenValveControl

  document.getElementById('header2').innerHTML = 'Valve Positions';        // -> Valve Positions

  document.getElementById('radio1').labels[0].innerHTML = '1-Living';      // -> VZ1
  document.getElementById('radio2').labels[0].innerHTML = '2-Dilbert';     // -> VZ2
  document.getElementById('radio3').labels[0].innerHTML = 'VZ3';
  document.getElementById('radio4').labels[0].innerHTML = 'VZ4';
}
```

Figure 8: Example file **customize.js**

As an example, you will find the first replacement item in the <body> section of index.html:

```
<h2 id='header1'>OpenValveControl</h2>
```

The header text „`OpenValveControl`“ will be replaced with the given text in the script, because the `<h2>` expression was given the attribut `id =‘header1‘`:

```
<h2 id='header1'>1<sup>st</sup> Floor</h2>
```

As you see, you can even enter html attributes like superscript <sup> (superscript) to get „1st Floor“ or similar HTML expressions.

Hence, basically this script overwrites the default headers to differentiate several devices and to give the valve radio buttons more meaningful (room) names.

These texts are preserved even if the index.html file is updated.

1.6 First driving trials

A connected motor without a valve can move beyond its end position if it does not experience any resistance when moving. This may result in damage to the mechanics. The control unit is operated at your own risk.

1.6.1 Delivery status of the drives

The following photo shows an actuator in its as-delivered condition. You can see that the round plunger (in the center) is offset a few millimeters inwards. This corresponds to the OPEN valve position on my Rotex heating circuit manifold and makes it easier to install the motor.

When referencing (homing), the actuator moves the plunger out, so when installed it pushes the valve pin in, which corresponds to the CLOSED valve position (on my Rotex heating circuit manifold). When the valve is closed, the plunger presses against the spring force of the valve pin, which is why installing the motor in motor position CLOSED is not recommended.

This may be different with other devices!
OpenValveControl does not currently allow the direction of rotation to be changed!

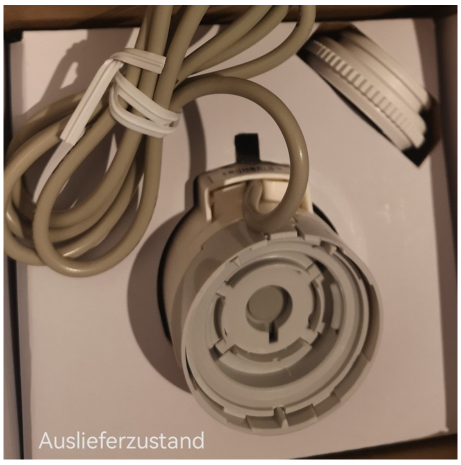


Figure 9: As-delivered condition

1.6.2 Checking the reference run

First of all, you should carry out a reference run - WITHOUT installing the motor in the heating circuit manifold. The easiest way to do this is on the desk. Connect the motor to the left-hand socket, for example, as shown in the next picture. This corresponds to valve zone VZ1 in the user interface.

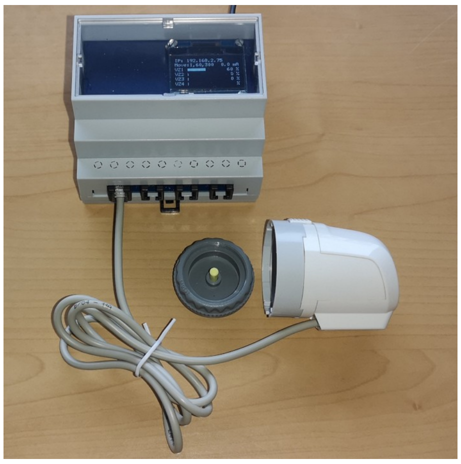
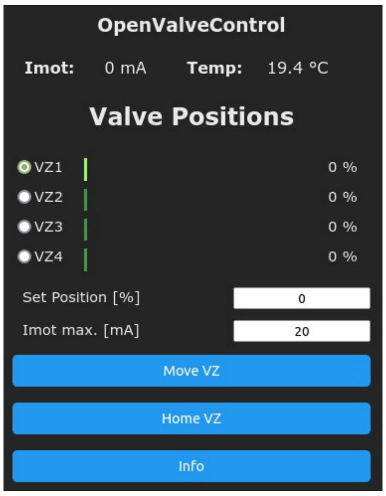


Figure 10: Connection of a motor as VZ1 to test the reference run.

- Now select „VZ1“ in the user interface (by clicking with the mouse)
- Then enter the (small) value 20 for „Imot max [mA]“
- Now click on the "HOME VZ" button.



OpenValveControl

Imot: 0 mA **Temp:** 19.4 °C

Valve Positions

VZ1 0 %
VZ2 0 %
VZ3 0 %
VZ4 0 %

Set Position [%] 0
Imot max. [mA] 20

The motor should then move the plunger "out" and display the motor current during this process. The LED on the PCB also lights up as an indicator.

The reference run is terminated when the motor current exceeds the entered limit value. Valve position "CLOSED" is then assumed and the actual position is set to 0 %.

You can easily simulate this with such a low limit value by pressing against the plunger with a suitable tool or with your thumb while it is being moved out.

Figure 11: Graphical User Interface

Caution! The plunger must not move out so far that the guide lug comes completely out of the groove, otherwise the plunger cannot be moved in without manual support.

The reference run is therefore aborted after approx. 2 minutes, even if the current limit value is not exceeded. The LED goes out.

In case of problems, simply unplug the drive!

If this "reference run" has worked, you can specify a „Set Position“ of e.g. 20 % for the motor and move to this position using the "Move VZ" button.

If the motor stops prematurely, you can increase the current limit value in small steps. Proceed in a similar way once you have installed the motor. For comparison: I have entered a current limit value of 55 mA for my Rotex heating manifold.

2 Controlling with node-RED[3]

Node-RED is the "Swiss army knife" of the Internet of Things (IoT) and the ideal tool when it comes to exchanging data between sensors and actuators. In this chapter, we want to control *OpenValveControl* with a node-RED „flow“.

In node-RED, "flows" refer to the graphical representations and data connections of "nodes". A node can, for example, represent an interface to MQTT and is represented as a block (node) with inputs and outputs. The properties of these nodes are defined via parameters (e.g. the IP address of the MQTT server).

You can install node-RED on a RaspberryPi, for example, so that it is permanently available (24/7). On the same RaspberryPi, a mosquitto server can run in parallel as an MQTT host and other applications (e.g. grafana).

I have been running this configuration on a 3rd generation Pi for years without any problems.

2.1 Adjusting the valves using node-RED

2.1.1 Task definition

As we have seen, we can adjust our valves via the web interface. To do this, we need to know the IP address of the web UI.

Then we can select the ValveZone (the actuator or valve), enter the target position (0 to 100%), enter the maximum motor current in milliamperes and press the "Move" button.

This sends an "http GET" command to the ESP, which contains the parameters mentioned. The command is displayed in plain text at the bottom of the Web UI screen.

Example

Valve 1 (vz=1) should be moved to 26%. If the motor current exceeds the value 30 mA, the process should be aborted (blocking protection):

```
/move?vz=1&set_pos=26&max_mA=30.0
```

We can achieve exactly the same thing by sending the GET command via a web browser on our LAN. We just need to prefix it with the IP address of our control device (the IP address will be displayed on the OLED after the ESP has successfully logged into the home network):

```
http://192.168.2.131/move?vz=1&set_pos=26&max_mA=30.0
```

3 **node-RED** is a flow-based programming tool, originally developed by IBM’s Emerging Technology Services team (https://emerging-technology.co.uk/) and now a part of the OpenJS Foundation (https://openjsf.org/).

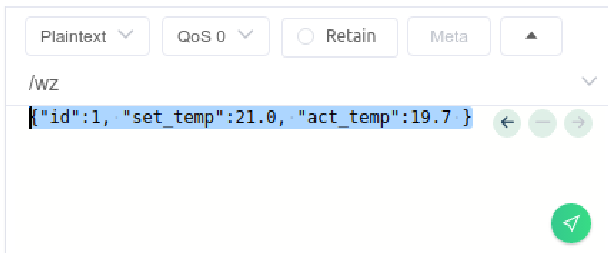
2.1.2 Specifying a setpoint

In order to solve the task in a practical way, we first have to specify the target value.

We assume that we have a temperature sensor with display in our living room for a reasonable individual room control as well as a possibility for a setpoint specification. This could be realized with a small ESP or with a commercially available industrial product (display of actual and setpoint temperature, 2 buttons ⇅ for the setpoint).

This ESP could now send the described GET command directly to our controller via WiFi and, for example, control the valve to 80% if it is too cold in the room or to 20% if it is too warm (simple "two-point" control). In addition, each thermostat must of course be informed of the valve assignment, the maximum current and possibly the control algorithm. However, there is other information that is useful for control, e.g. outdoor temperature, solar radiation, night setback, extended absence (vacation) and others. It therefore makes sense to only report the actual and setpoint temperatures of the respective room and to manage all rooms centrally, e.g. with node-RED.

To do this, we send a sensor ID, the actual temperature and setpoint value to an MQTT server (also known as a broker, e.g. mosquitto). Until we have programmed the sensor, we can simulate this with the MQTTX[4] tool:



/wz

```
{"id":1, "set_temp":21.0, "act_temp":19.7 }
```

If we are connected to the MQTT broker (e.g. as test@raspi3:1880), we can define a **topic** (e.g. "/wz") and send the entered **payload** to the MQTT broker by clicking on the green arrow. The data can be changed as required.

Figure 12: Using MQTTX to simulate data source

2.1.3 flow: "Subscribe" thermostat values from our MQTT broker

In node-RED, we now place an "mqtt in" node in our "flow". Double-clicking on the node opens the "Properties" window, in which we enter the following data:

I have entered "localhost:1883" as the **server** because both node-RED and mosquitto (the MQTT broker) run on the same RaspberryPi. Otherwise, the corresponding IP address of the MQTT server must be entered instead of "localhost".

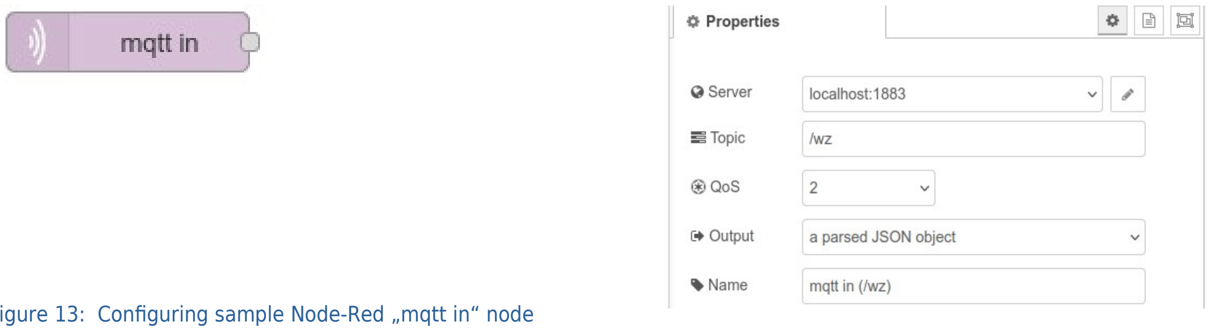
4 „Your All-in-One MQTT Client Toolbox“ (https://mqttx.app/). MQTTX is a free open-source project available under the Apache-2.0 LICENSE.

We can assign the **topic** "/wz" (deutsch: WohnZimmer = Living Room) in the thermostat (or temperature sensor). We have entered this in MQTTX as a test and it is used to differentiate between the various rooms or temperature sensors.

Quality of Service "**QoS**" describes the increasing degree of reliability in the delivery of messages, the highest value is 2.

As **output**, we want a "parsed JSON object", i.e. the same message that was sent from the sensor to the MQTT broker.

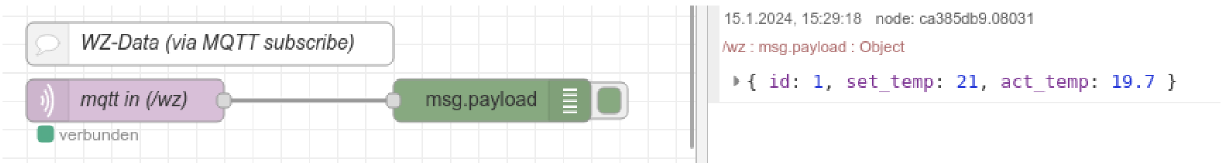
The **name** can be chosen at will and is primarily used to make the flow clearer.



Server: localhost:1883
Topic: /wz
QoS: 2
Output: a parsed JSON object
Name: mqtt in (/wz)

Figure 13: Configuring sample Node-Red „mqtt in“ node

We now add a "debug" node so that we can check the output of the "mqtt in" node. As soon as we send a data record to the MQTT broker via MQTTX, the subscribed data is sent to node-RED and is available at the output of the "mqtt in" node:



```
15.1.2024, 15:29:18   node: ca385db9.08031
/wz : msg.payload : Object
▸{ id: 1, set_temp: 21, act_temp: 19.7 }
```

Figure 14: Using Node-Red „debug“ node

2.1.4 flow: Convert temperature difference into valve position

We now know the room ID, the setpoint and actual temperature and can now make a decision for the valve position.

To simplify matters, we take the room ID as the valve zone (vz) and program the two-point controller described above, which we want to formulate in JavaScript as follows:

If the deviation is more than 0.5 degrees, we switch the valve closed (20%) or open (80%), otherwise we leave it as it is.

Note: The simplified assumption is that the flow temperature is weather-compensated and we only make a fine adjustment with a valve. Of course, we can also program an arbitrarily complicated control algorithm here!

For this purpose, node-RED has the "function" node, in which JavaScript can be executed almost arbitrarily. A "message" is accepted as input, which can be modified (almost) at will by the program and output at one or more outputs. The instruction `return (x)` is used for this purpose, whereby x can also be `null`, in which case no output is generated. We use this hysteresis to avoid unnecessary adjustments if the temperature is reasonably correct.

At the start of the JavaScript program, our message has the following values, for example:

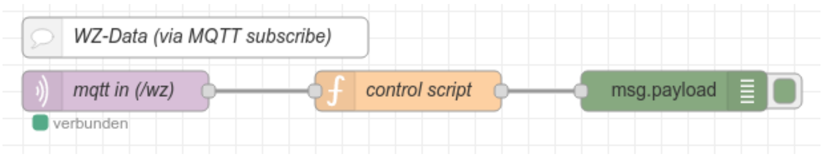
```
msg.payload = {"id":1,"set_temp":21,"act_temp":19.7}
```

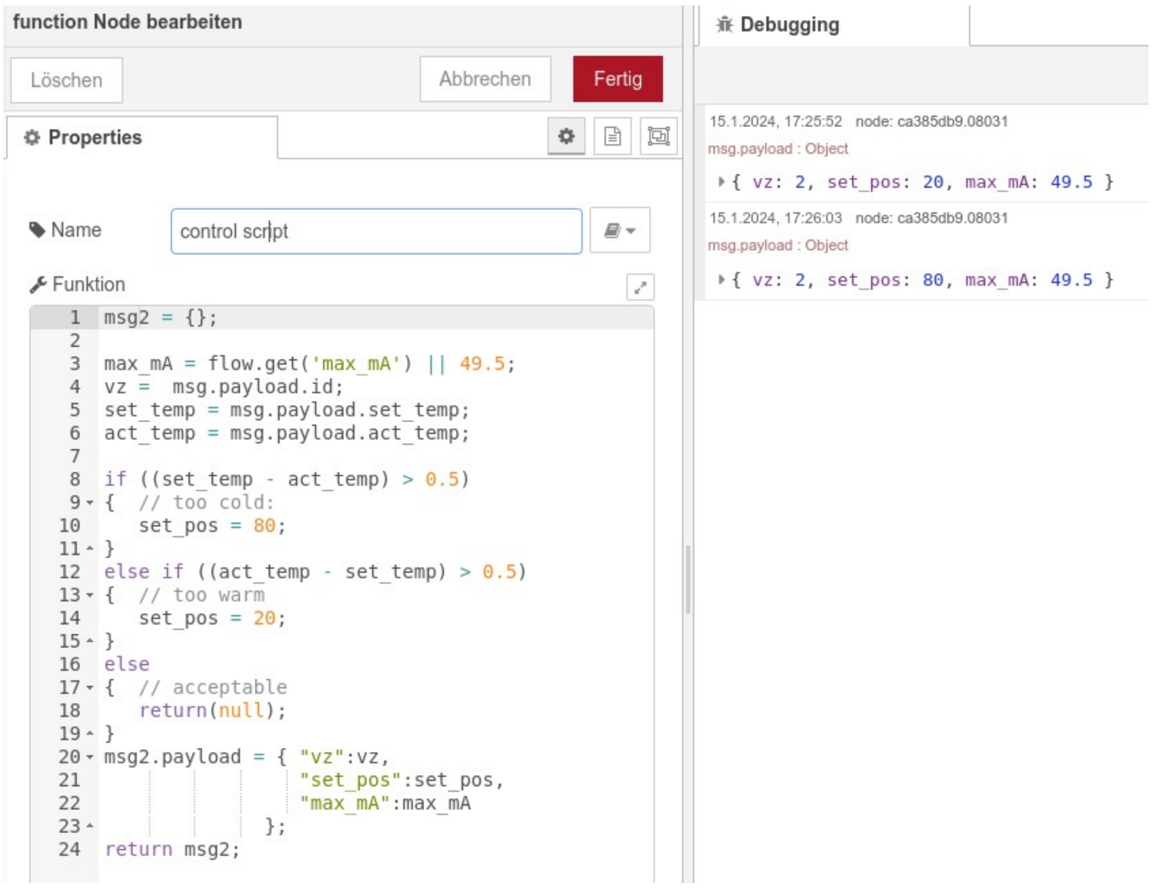
At the output, we want to have the appropriate parameters for the "http GET" request:

```
msg2.payload = {"id":1, "set_pos":80, "max_mA":50.0}
```

All we have to do is generate a new message msg2. We temporarily take the missing value for "max_mA" from a variable that was somehow saved in the flow with the instruction `flow.set("max_mA", <value>)`, for example.

Our JavaScript program could then look like this:





Name: control script

Funktion

```
msg2 = {};

max_mA = flow.get('max_mA') || 49.5;
vz =  msg.payload.id;
set_temp = msg.payload.set_temp;
act_temp = msg.payload.act_temp;

if ((set_temp - act_temp) > 0.5)
{  // too cold:
   set_pos = 80;
}
else if ((act_temp - set_temp) > 0.5)
{  // too warm
   set_pos = 20;
}
else
{  // acceptable
   return(null);
}
msg2.payload = { "vz":vz,
                 "set_pos":set_pos,
                 "max_mA":max_mA
               };
return msg2;
```

```
15.1.2024, 17:25:52   node: ca385db9.08031
msg.payload : Object
▸{ vz: 2, set_pos: 20, max_mA: 49.5 }
15.1.2024, 17:26:03   node: ca385db9.08031
msg.payload : Object
▸{ vz: 2, set_pos: 80, max_mA: 49.5 }
```

Figure 15: Configuring sample Node-Red „function“ node

The 2 outputs in the debugger were the responses to the following 3 messages:

```
{"id":2, "set_temp":18.0, "act_temp":19.7 }        // too warm
{"id":2, "set_temp":22.0, "act_temp":19.7 }        // too cold
{"id":2, "set_temp":20.0, "act_temp":19.7 }        // acceptable
```

This is a very good way of testing that no output is produced on the third input as intended.

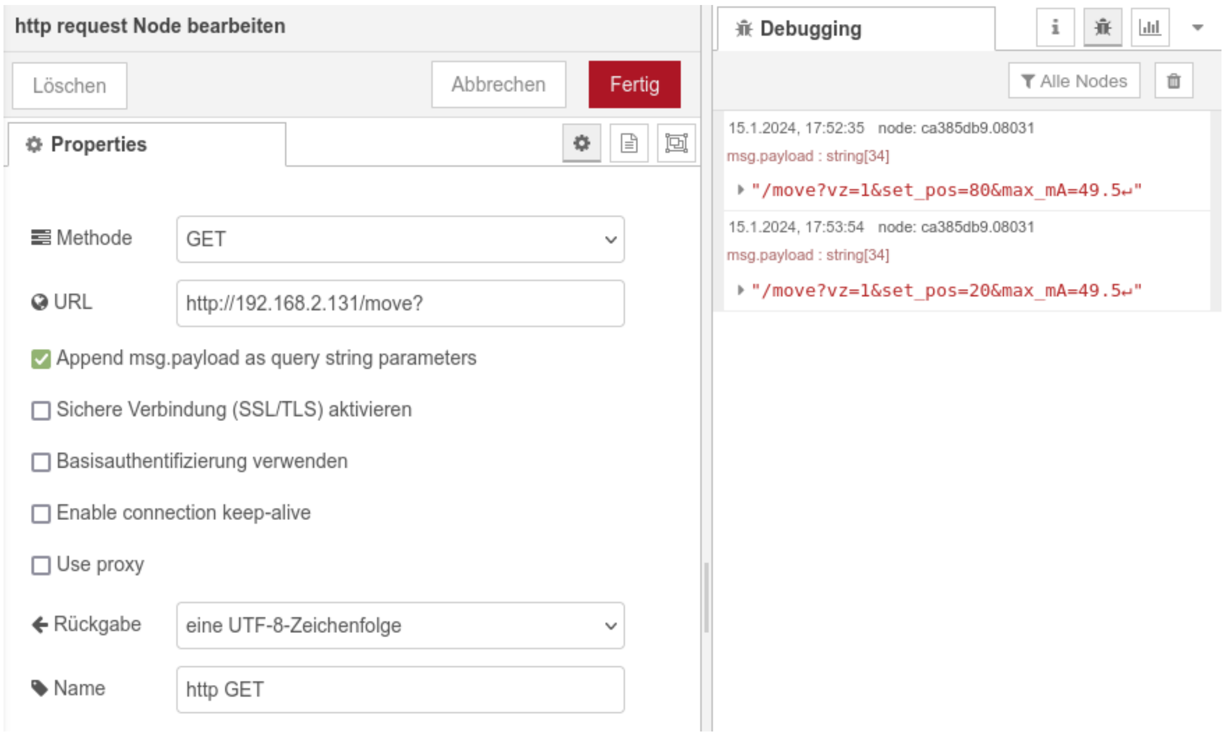
Note

To prevent an error message from occurring when reading the max_mA variable or an invalid value being used if no value has (yet) been saved, a logical AND has been appended, which assigns a default value in this case:

```
max_mA = flow.get('max_mA') || 49.5;
```

2.1.5 flow: Output the query parameters as http GET request

There is another (network) node for sending the "http request". We place it at the output of the JavaScript "function" node and connect both nodes. We move the debug node to the output of the http request. Then we fill in the properties as follows:


http request Node bearbeiten
Löschen | Abbrechen | Fertig
Properties
Methode: GET
URL: http://192.168.2.131/move?
☑ Append msg.payload as query string parameters
☐ Sichere Verbindung (SSL/TLS) aktivieren
☐ Basisauthentifizierung verwenden
☐ Enable connection keep-alive
☐ Use proxy
Rückgabe: eine UTF-8-Zeichenfolge
Name: http GET

Debugging
Alle Nodes

```
15.1.2024, 17:52:35   node: ca385db9.08031
msg.payload : string[34]
"/move?vz=1&set_pos=80&max_mA=49.5↵"
15.1.2024, 17:53:54   node: ca385db9.08031
msg.payload : string[34]
"/move?vz=1&set_pos=20&max_mA=49.5↵"
```

Figure 16: Configuring sample Node-Red node „http request“

The debug output when publishing our thermostat values shows the response of our ESP server, when it receives the GET request. In our case, fortunately the ESP echoes back the request (watch the CR ↲ at the end of the debugged payload). So we can see that all parameters from the message were appended to the URL with the addition "/move?" for the move command.

Similarily, the commands "home" or "info" or "status" can also be implemented and further processed in node-RED.

Of course, you can delete or deactivate the debug node if you no longer need it.

This completes the task and we have our finished flow:

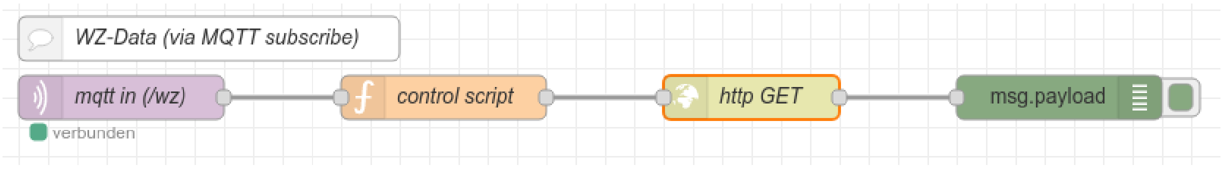


Figure 17: Complete sample in Node-Red for valve control

3 Driving the motors by PWM

3.1 Introduction to the H-bridge

In our application we are using two „MX1508 Brushed DC Motor Drivers“. Each board can drive two motors independently. So we can run up to 4 motors in total.

These drivers incorporate an H-bridge for each motor and can drive the motors in both directions. A simpified schematic of each H-bridge is shown in the following figure:

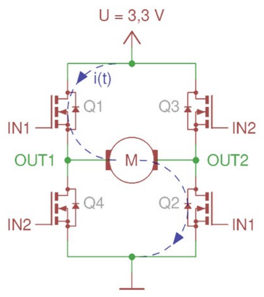


Figure 18: Simplified schematic of each H-bridge

The motor is connected to outputs OUT1 and OUT2.

When IN1 is high and IN2 is low, a current flows from the voltage source through Q1, the motor and through Q2 to GND, moving the motor in one direction.

When IN1 is low and IN2 is high, the current flows from the voltage source through Q3, the motor and through Q4 to GND, moving the motor in the opposite direction.

If both inputs are low, all transistors are high-impedance and no current flows.

Internal logic inhibits a short between U and GND, when both inputs are high. Instead, transistors Q1 and Q3 are switched off, Q2 and Q4 are switched on, shorting the current through the motor. This has the same effect as a brake.

To drive a motor in one direction, we also can apply a pulse width modulated (PWM) signal intead of a constantly high level. In this case, we are able to control the power driving the motor by the duty cycle of the PWM signal: p(t) = u(t) * i(t).

But there is yet another advantage: During the period in which the H-bridge is switched off, we want to measure the motor’s „Back EMF“ (Electro Magnetic Force), a voltage almost proportional to the motor speed with a characteristic shape, allowing to count the number of armature segments that the motor has passed over.

And the PWM period is a good trigger to measure the motor current, which is required to switch off the motor in the event of a blockage.

3.2 PWM generation and interrupts

After a couple of experiments we configured two PWM signals with a period of 8 ms, named *slice1, output1 (SaOUT1)* and *slice1, output2 (SaOUT2)*, following the naming convention of the PIC datasheet.

SaOUT1 ist used to enable the H-bridge, either asserted to IN1 or IN2, depending on the desired motor direction. The signal is set 7.2 ms high and 800 µs Low. When the signal is low, At a low level, no current is driven through the motor and so the motor acts like a generator where we can measure the back EMF voltage (V_{BEMF}). Therefore we enable a PWM interrupt with the falling edge of SaOUT1.

To measure the motor current, we have to wait until the current has settled to a stable value. This is realized with the synchronized PWM output SaOUT2: The signal is set 2 ms high and the falling edge also triggers the PWM interrupt.

The interrupt service routine can detect, which signal was the source of interrupt and perform the desired data acquisition.

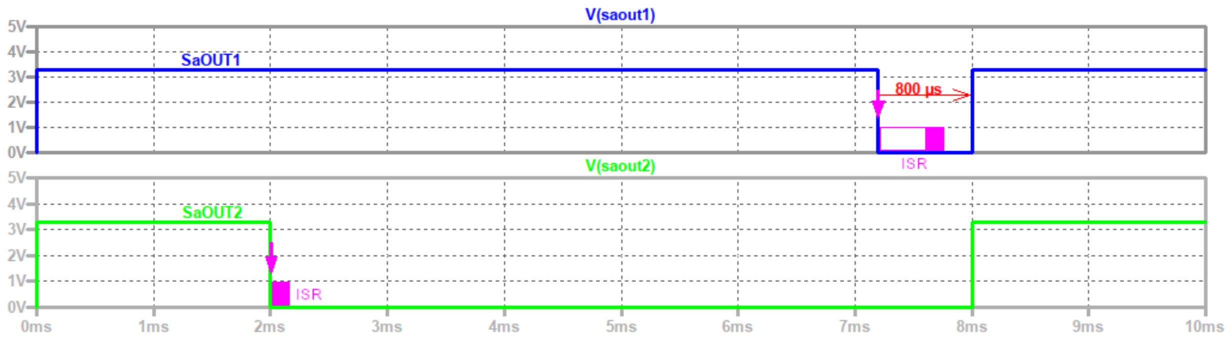


Figure 19: Timing of PWM outputs SaOUT1 and SaOUT2 (LTspice simulation)

3.3 Acquisition of motor current

As shown in the schematics, the motor current is measured by an INA219 breakout board, where the small voltage drop across a shunt is sensed and can be read via the I2C interface. Our INA219 board uses a 0.1 Ohm shunt, but may be different on others. 100 mA motor current would then generate a voltage drop of only 10 mV. Due to the resolution of 10 µV, current changes of 0.1 mA can be measured.

These defaults are working pretty good for our application, as we can simply read the INA216 register at address 1, which continuously is updated with the shunt voltage measurement data using the default configuration. No further initialisation or configuration is required.

The shunt voltage register is in signed 16 bit format and LSB = 10 µV, which is equivalent to 0.1 mA (I = U / R).

As previously explained, we read the motor current in the interrupt service routine, on the falling edge of our PWM signal *slice1, output2 (SaOUT2)*. This is 2 ms after the H-bridge is switched on and allows the current to stabilize.

To read INA216 register 0x01, we first have to write the desired register address (0x01) to the configuration register at address 0x00. After this, we can start two I2C read cycles from the INA219. The most significant byte (MSB) is read first.

This measurement is carried out by the PIC with a I2C clock of 400 kHz. Address setup and 16 bit read of the shunt voltage register require 138.5 µs (add some microseconds for data handling and cleanup).

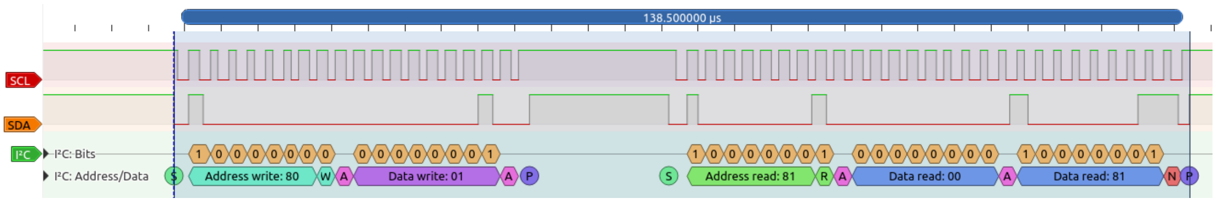


Figure 20: Visualisation of this I2C communication recorded with a logic analyzer

The reading is repeated every PWM cycle (8 ms), hence we also have to read the current, when the PIC is in the idle state, so that we can also see the current at standstill.

This is the code to read the motor current from INA219 module:

```
/* Read motor current via INA219 Shunt Current Register
 * (1 LSB = 10µV, Rs = 0,1 Ohm => I / 0.1 mA = Us / 10µV).
 * Exectime ~171 µs.
 */
ina219_reg(1);               // exec time ca. 49 µs (@SCL 400 kHz)
g_mAx10 = ina219_read();     // exec time ca. 72 µs (@SCL 400 kHz)
if (g_mAx10 < 0) g_mAx10 = 0;  // offset may cause negative readings
```

3.3.1 Exemplary measurement results

The following oscillograms have been recorded using the PIC’s digital to analog converter (DAC). This DAC has only 8 bit, and the motor was expected to reach up to 50 mA, which produces a (16 bit) register value of up to 500 (x 0.1 mA). So the read 16-bit value was shifted 2 bit to the right, which corresponds to a „division by 4“ and truncated to 8 bit.

50 mA → 500 = 0x1F4,   divide by 4 → 0x7D (unsigned) = 125

This value results in U_{DAC} = 3.3 V * 125 / 255 = 1.618 V at the output.

Accordingly, we obtain 30.9 mA/volt in the oscillogram..

These are the results in both directions, with no load (free running on the desk) and with some load (applied with my thumb pressing onto the plunger).

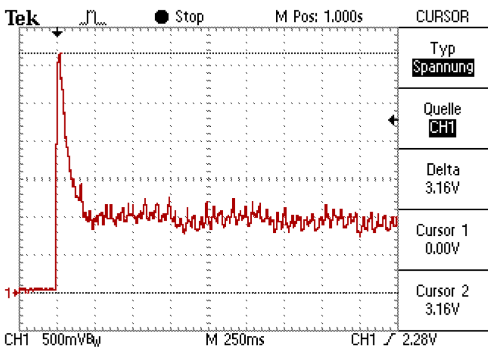


Figure 21: Motor current, open direction, no load

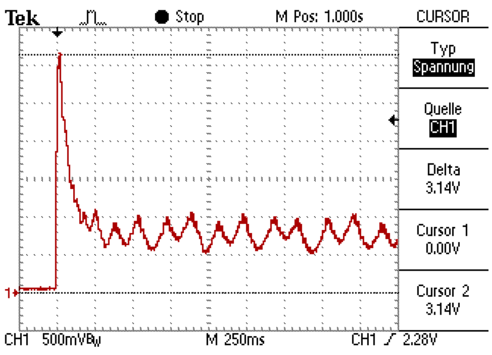


Figure 22: Motor current, open direction, with load

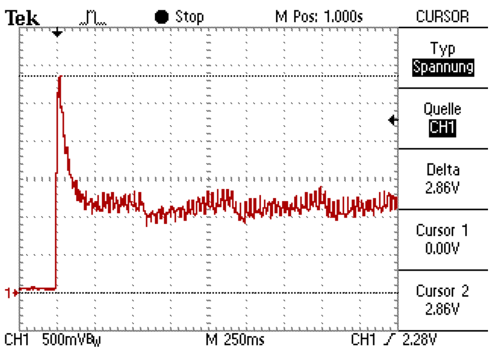


Figure 23: Motor current, close direction, with load

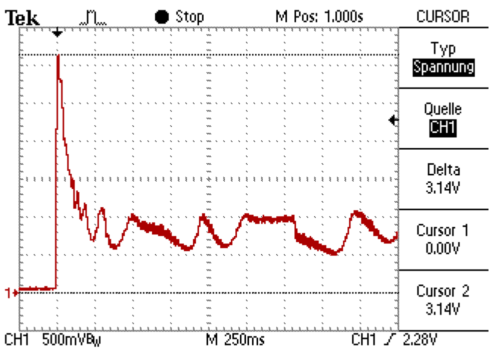


Figure 24: Motor current, close direction, no load

As result we can see, that the motor start-up is completed after approx. 150 ms, causing a peak current close to 100 mA. Without load the motor draws approx. 15 to 30 mA, with moderate load approx. 10 mA more. In reality, when the motor runs against a resistance, the current consumption may increase, hence with a ohmic resistance of R = 30 Ohm and U = 3.3 V - 2 x U_{DS} (H-bridge), the current cannot exceed 100 mA, which is well below the max. current of 120 mA of the VDMOT. And as soon the motor turns, the back EMF counteracts the supply voltage until the system reachs a stable state.