

Mit MQTT einen Roboter steuern [Teil 2]

Im ersten Teil dieser Blogserie haben Sie alles Wissenswerte über MQTT gelernt. Zudem wurde auf einem Raspberry Pi ein Server, der sogenannte Broker, eingerichtet und mit Kommandozeileingaben Nachrichten an den Broker gesendet und auch empfangen.

In diesem Teil wird der Aufbau etwas komplexer, sodass die Daten von diversen NodeMCUs und einem Arduino Uno mit Ethernetshield, den sogenannten Clients, gesendet und empfangen werde. Hintergrund wird sein, dass Sie auf den gängigsten MicroControllern die Az-Delivery vertreibt einen MQTT-Client betreiben werden. In den Hardware-Voraussetzungen ist nachzulesen, welche Teile zwingend zum Abschluss der Reihe benötigt werden. Einige Komponenten werden nur für diesen Blogteil benötigt, finden aber in diversen anderen Blogbeiträgen von az-delivery.de Verwendung.

Hardware-Voraussetzung

Für diesen Blogbeitrag brauchen Sie nur wenige Bauteile, siehe Tabelle 1.

Pos	Anzahl	Bauteil	Link
1	1	Raspberry Pi (erforderlich)	https://www.az-delivery.de/
2	1	Passendes Netzteil	
3	1	NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F (erforderlich)	https://www.az-delivery.de/
4	1	ESP32 NodeMCU Module WLAN WiFi Development Board (erforderlich)	https://www.az-delivery.de/
5	1	Mikrocontroller Board mit USB-Kabel (optional)	https://www.az-delivery.de/
6	1	Ethernet Shield W5100 (optional)	https://www.az-delivery.de/
7	1	Widerstände Sortiment (optional)	https://www.az-delivery.de/
8	2	Potentiometer (erforderlich)	https://www.az-delivery.de/
9	1	LCD Display 4x20 Zeichen Blau i2c (optional)	https://www.az-delivery.de/
10	1	LED einfarbig (optional)	https://www.az-delivery.de/

Tabelle 1: Benötigte Hardware

Denken Sie bei dem Pi daran, dass Sie neben der oben genannten Hardware auch eine MicroSD-Karte brauchen. Hierzu sollten Sie raspbian als Image auf die Karte installieren.

Software-Voraussetzung

Die benötigte Software für dieses Projekt ist überschaubar:

- Arduino IDE (<https://www.arduino.cc/en/Main/Software>), hier am besten die aktuelle Version herunterladen
- Die Bibliothek **PubSubClient** mit allen Abhängigkeiten

- Die Bibliothek **LiquidCrystal_I2C** mit allen Abhängigkeiten

Wie Sie Libraries über die Bibliotheksverwaltung installieren können, ist unter <https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/arduino-ide-programmieren-fuer-einsteiger-teil-1> Abschnitt Bibliotheksverwaltung, näher beschrieben.

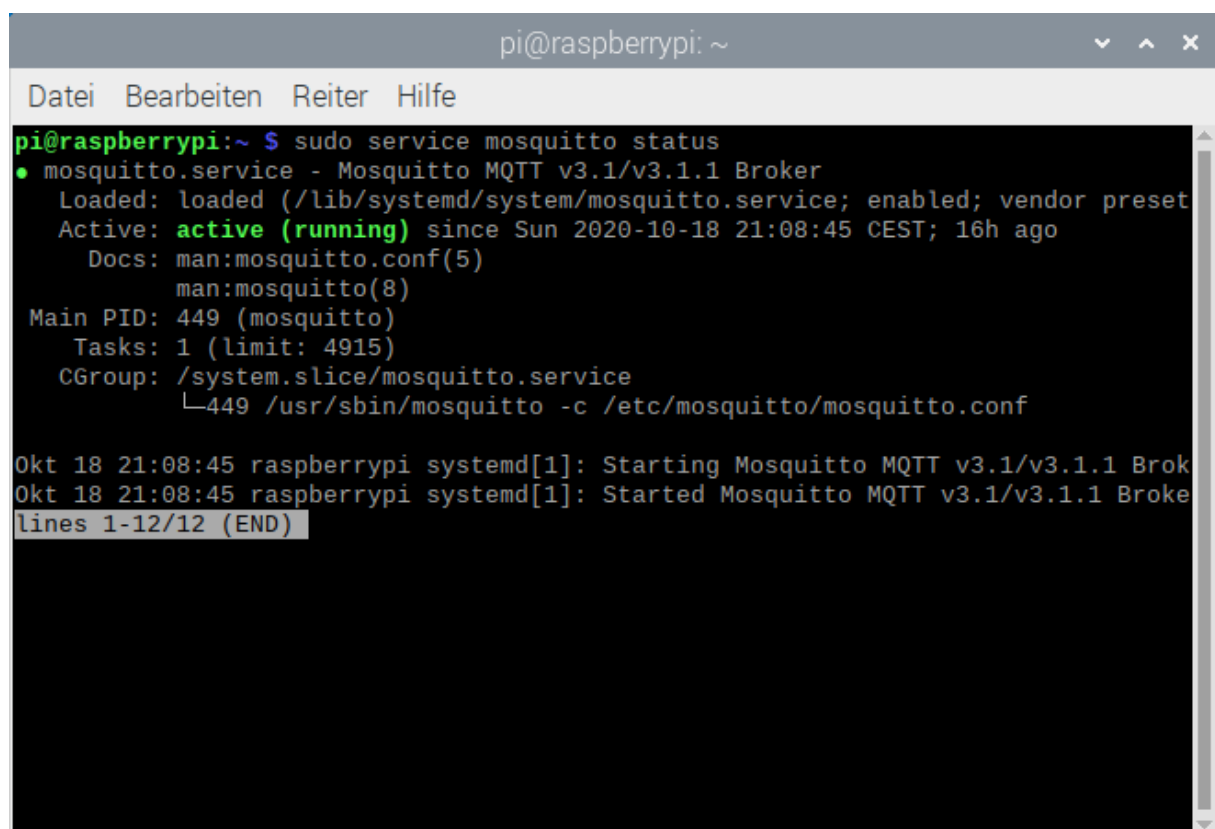
Grundvoraussetzung

Damit dieser und auch die nachfolgenden Blogbeiträge funktionieren, braucht es einen Raspberry Pi mit installiertem MQTT-Broker. Wie das genau geht und welche Kommandos sieh dafür eingeben müssen, können Sie in [Teil 1 dieser Blogserie](#) genau nachlesen. Prüfen Sie bitte zusätzlich, ob der Broker auch beim Hochfahren vom Raspberry Pi gestartet wurde. Geben Sie dazu den Befehl aus Code 1 ins Terminal ein.

```
sudo service mosquitto status
```

Code 1: Abfrage im Terminal ob mosquitto gestartet ist

Sollte die Ausgabe ein **active (running)** zeigen, siehe Abbildung 1, sind keine weiteren Kommandos nötig.



```
pi@raspberrypi: ~
Datei Bearbeiten Reiter Hilfe
pi@raspberrypi:~ $ sudo service mosquitto status
• mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
  Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset
  Active: active (running) since Sun 2020-10-18 21:08:45 CEST; 16h ago
  Docs: man:mosquitto.conf(5)
        man:mosquitto(8)
  Main PID: 449 (mosquitto)
  Tasks: 1 (limit: 4915)
  CGroup: /system.slice/mosquitto.service
          └─449 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Okt 18 21:08:45 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Brok
Okt 18 21:08:45 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broke
lines 1-12/12 (END)
```

Abbildung 1: Status von mosquitto-Broker im Terminal

Sollte sich die Ausgabe unterscheiden, so prüfen Sie bitte noch einmal, ob mosquitto korrekt installiert wurde und auch der Dienst korrekt in den Autostart eingebunden wurde. Die genaue Anleitung finden Sie in [Teil 1](#).

Simples MQTT-Subscribe mit NodeMCU Lua Amica Modul V2 ESP8266

Gleich ein Hinweis an dieser Stelle, für das hier beschriebene Beispiel wird der NodeMCU Lua Amica Modul V2 ESP8266 verwendet.

Im ersten Beispiel, siehe Code 2, soll der NodeMCU eine Verbindung mit dem MQTT-Broker aufbauen und alle Daten empfangen. Damit erhalten Sie auch direkt einen Einblick, wie die generelle Kommunikation mit der Bibliothek PubSubClient funktioniert. Damit Sie den Quellcode besser verstehen, wurde jede Funktion genau beschrieben.

Damit der Quellcode auch bei Ihnen läuft, müssen Sie ihre WLAN-ID und das Passwort im Quellcode abändern.

```
//-----  
// Example 1 NodeMCU with MQTT  
// Autor: Joern Weise  
// License: GNU GPI 3.0  
// Created: 20. Oct 2020  
// Update: 25. Oct 2020  
//-----  
#include <ESP8266WiFi.h> //Lib for Wifi  
#include <PubSubClient.h> //Lib for MQTT Pub and Sub  
//  
#ifndef STASSID  
#define STASSID "....."  
#define STAPSK "....."  
#endif  
  
#define ADVANCEDIAG 1  
const char* MQTT_BROKER = "raspberrypi"; //Name of the mqtt broker  
const char* SUBTOPIC = "/#";  
String clientID = "NodeMCU_1"; //Clientname for MQTT-Broker  
WiFiClient espClient;  
PubSubClient mqttClient(espClient);  
  
void setup()  
{  
  Serial.begin(115200); //Start Serial monitor baudrate 115200  
  delay(50);  
  writeAdvanceDiag("SerialMonitor enabled", true);  
  setupWifi();  
  writeAdvanceDiag("Set MQTT-Server", true);  
  mqttClient.setServer(MQTT_BROKER,1883);  
  writeAdvanceDiag("Set Callback-function", true);  
  mqttClient.setCallback(callback);  
  writeAdvanceDiag("Finish setup()-Function", true);  
}  
  
/*  
* =====  
* Function:  setupWifi  
* Returns:   void  
* Description: Setup wifi to connect to network  
* =====  
*/
```

```

void setupWifi()
{
  Serial.println("Connection to: " + String(STASSID));
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // put your main code here, to run repeatedly:
  if(!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();
}

/*
* =====
* Function:   callback
* Returns:    void
* Description: Will automatical called, if a subscribed topic
*              has a new message
* topic:      Returns the topic, from where a new msg comes from
* payload:    The message from the topic
* length:     Length of the msg, important to get content
* =====
*/
void callback(char* topic, byte* payload, unsigned int length)
{
  String stMessage = "";
  writeAdvanceDiag("Message arrived from topic: " + String(topic), true);
  writeAdvanceDiag("Message length: " + String(length), true);
  for (int i = 0; i < length; i++)
    stMessage += String((char)payload[i]);
  writeAdvanceDiag("Message is: " + stMessage, true);
}

/*
* =====
* Function:   reconnectMQTT
* Returns:    void
* Description: If there is no connection to MQTT, this function is
*              called. In addition, the desired topic is registered.
* =====
*/

```

```

void reconnectMQTT()
{
  while(!mqttClient.connected())
  {
    writeAdvanceDiag("Login to MQTT-Broker", true);
    if(mqttClient.connect(clientID.c_str()))
    {
      Serial.println("Connected to MQTT-Broker " +String(MQTT_BROKER));
      writeAdvanceDiag("Subscribe topic '" + String(SUBTOPIC)+ "'", true);
      mqttClient.subscribe(SUBTOPIC,1); //Subscribe topic "SUBTOPIC"
    }
    else
    {
      writeAdvanceDiag("Failed with rc=" +String(mqttClient.state()), true);
      Serial.println("Next MQTT-Connect in 3 sec");
      delay(3000);
    }
  }
}

/*
* =====
* Function:   writeAdvanceDiag
* Returns:    void
* Description: Writes advance msg to serial monitor, if
*              ADVANCEDIAG >= 1
* msg:        Message for the serial monitor
* newLine:    Message with linebreak (true)
* =====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
  if(bool(ADVANCEDIAG)) //Check if advance diag is enabled
  {
    if(newLine)
      Serial.println(msg);
    else
      Serial.print(msg);
  }
}

```

Code 2: Beispiel NodeMCU MQTT-Daten empfangen lassen

Am Anfang vom Code wird ein Objekt `espClient` erzeugt und dem ebenfalls erzeugten Objekt `mqttClient` übergeben. Damit kann bei einer aktiven WiFi-Verbindung eine generelle Kommunikation hergestellt werden. Startet der NodeMCU wird zunächst das WiFi, hier in der Funktion `setupWifi()`, eingerichtet und anschließend in der `setup()`-Funktion die Verbindung zum MQTT-Broker hergestellt. Mittels „`mqttClient.setServer(MQTT_BROKER,1883);`“ wird die Adresse und der Port vom MQTT-Broker konfiguriert und über „`mqttClient.setCallback(callback);`“ die Funktion zur Abarbeitung der empfangenen Daten bestimmt, dazu weiter unten mehr.

Die `loop()`-Funktion macht genau zwei Dinge:

1. Prüfung, ob der NodeMCU mit dem MQTT-Broker verbunden ist. Sollte das nicht der Fall sein, wird die Funktion `reconnectMQTT()` aufgerufen
2. Die Funktion `mqttClient.loop()` wird aufgerufen, um die Kommunikation mit MQTT zu durchzuführen.

Der erste Schritt ist die Anmeldung beim MQTT-Broker mit einem eindeutigen Client-Namen. Diesen können Sie über die globale Variable „clientID“ frei bestimmen. War die Anmeldung erfolgreich, wird im nächsten Schritt das gewünschte Topic abonniert. Dies geschieht mittels „`mqttClient.subscribe(SUBTOPIC,1)`“, wobei in diesem Beispiel SUBTOPIC das Topic „/#“ (alle Topics) abonniert und die „1“ für ein QoS größer 0 steht. Damit wird der NodeMCU mit Nachrichten vom Broker versorgt, sobald es Änderungen bei einem Topic gibt.

Im Fehlerfall versucht sich der NodeMCU alle 3 Sekunden mit dem Broker neu zu verbinden.

Damit kommen wir zu dem Punkt, an dem der NodeMCU vom Broker Daten empfängt., sofern der NodeMCU verbunden ist. Mittels der Funktion `callback()` wird, sofern es neue Nachrichten gibt, die Nachrichten ausgewertet. Dabei wird der `callback()`-Funktion drei Parameter übergeben:

1. `topic`: Welches den absoluten Topicpfad der empfangenen Nachricht wiedergibt
2. `payload`: Die Nachricht, wobei diese in Byte-Format vorliegt
3. `length`: Die Länge der Nachricht

Mit diesen drei Informationen können Sie auswerten, wofür Sie das gewählte Topic brauchen. Im hier vorliegenden Fall geben wir das Topic über den seriellen Monitor aus. Direkt danach wandeln wir die Byte-Nachricht, mittels der in Code 3 gezeigten for-Schleife, in einen String um. Damit haben Sie eine normal lesbare Nachricht.

```
for (int i = 0; i < length; i++)  
stMessage += String((char)payload[i]);
```

Code 3: Codefragment um MQTT-Nachricht in String zu konvertieren

Diese Nachricht im String-Format wird ebenfalls über den seriellen Monitor ausgegeben.

Das dieser Sketch auch so reibungsfrei funktioniert, können Sie in Abbildung 2 sehen.

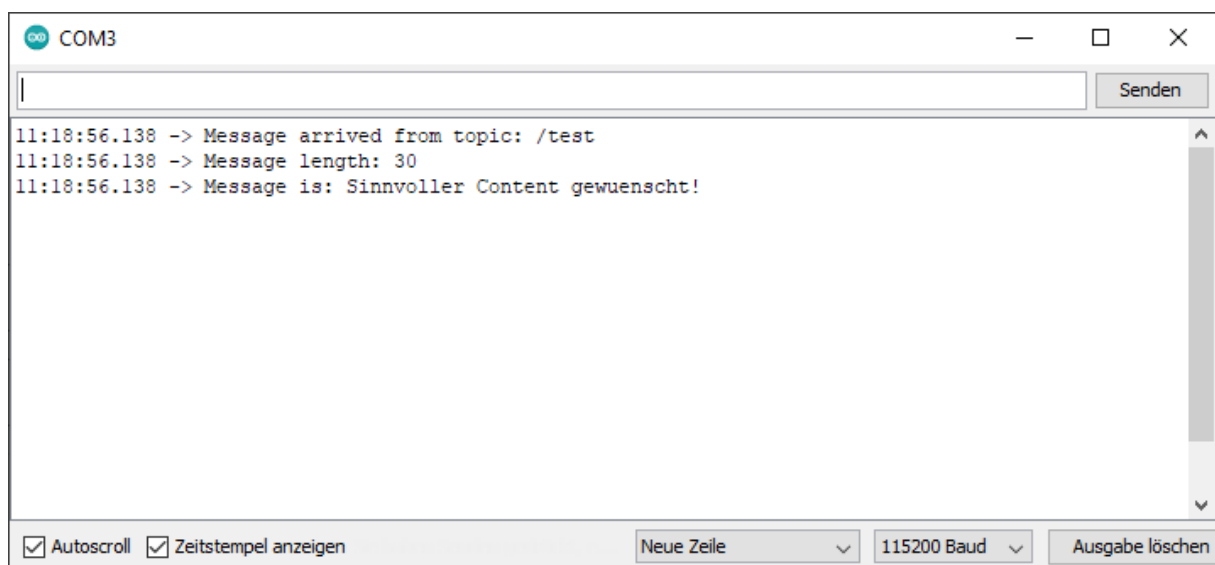


Abbildung 2: Empfangene Nachricht vom MQTT-Broker

Bitte bedenken Sie gleich im Vorfeld Ihrer Nachrichten, dass unsere deutschen Umlaute nicht funktionieren. Diese werden dann durch andere Zeichen dargestellt.

Simple MQTT-Subscribe mit ESP32 NodeMCU

Hinweis an dieser Stelle, für das folgende Beispiel wird das ESP32 NodeMCU Module WLAN WiFi Development Board genutzt.

Im Beispiel von Code 4 soll der ESP32 NodeMCU Daten an den MQTT-Broker senden. Da es sehr einfach ist starten wir damit, dass alle zwei Sekunden die Laufzeit vom ESP32 NodeMCU Module WLAN WiFi Development Board an den Broker gesendet wird.

Auch in diesem Beispiel sind im Code viele Kommentare vorhanden, damit Sie diesen leichter verstehen. Damit der Quellcode auch bei Ihnen läuft, müssen Sie ihre WLAN-ID und das Passwort im Quellcode abändern.

```
//-----  
// Example 2 ESP-NodeMCU with MQTT  
// Autor: Joern Weise  
// License: GNU GPI 3.0  
// Created: 20. Oct 2020  
// Update: 25. Oct 2020  
//-----  
#include <WiFi.h>  
#include <PubSubClient.h> //Lib for MQTT Pub and Sub  
//  
#ifndef STASSID  
#define STASSID "....." //Enter Wfi-Name  
#define STAPSK "....." //Enter Passkey  
#endif  
  
#define ADVANCEDIAG 1  
const char* MQTT_BROKER = "raspberrypi"; //Name of the mqtt broker  
const char* PubTopic = "/Client/ESP32"; //Topic where to publish  
String clientID = "ESP-DevKit_1"; //Clientname for MQTT-Broker  
WiFiClient espClient;  
PubSubClient mqttClient(espClient);  
  
unsigned long lLastMsg = 0;  
int iTimeDelay = 2000; //Set delay for next msg to 2 seconds  
#define MSG_BUFFER_SIZE (50)  
char msg[MSG_BUFFER_SIZE];  
  
void setup()  
{  
  Serial.begin(115200); //Start Serial monitor baudrate 115200  
  delay(50);  
  writeAdvanceDiag("SerialMonitor enabled", true);
```

```

setupWifi();
writeAdvanceDiag("Set MQTT-Server", true);
mqttClient.setServer(MQTT_BROKER,1883);
writeAdvanceDiag("Finish setup()-Function", true);
}

/*
* =====
* Function:   setupWifi
* Returns:    void
* Description: Setup wifi to connect to network
* =====
*/
void setupWifi()
{
  Serial.println("Connection to: " + String(STASSID));
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // put your main code here, to run repeatedly:
  if(!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();

  if(millis() - lLastMsg > iTimeDelay)
  {
    lLastMsg = millis();
    sprintf(msg,MSG_BUFFER_SIZE, "%1d",millis()); //Convert message to char
    mqttClient.publish(PubTopic,msg,true); //Send to broker
  }
}

/*
* =====
* Function:   reconnectMQTT
* Returns:    void
* Description: If there is no connection to MQTT, this function is

```



```

*          called. In addition, the desired topic is registered.
* =====
*/
void reconnectMQTT()
{
    while(!mqttClient.connected())
    {
        writeAdvanceDiag("Login to MQTT-Broker", true);
        if(mqttClient.connect(clientID.c_str()))
        {
            Serial.println("Connected to MQTT-Broker " +String(MQTT_BROKER));
        }
        else
        {
            writeAdvanceDiag("Failed with rc=" +String(mqttClient.state()), true);
            Serial.println("Next MQTT-Connect in 3 sec");
            delay(3000);
        }
    }
}

/*
* =====
* Function:   writeAdvanceDiag
* Returns:    void
* Description: Writes advance msg to serial monitor, if
*             ADVANCEDIAG >= 1
* msg:        Message for the serial monitor
* newLine:     Message with linebreak (true)
* =====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
    if(bool(ADVANCEDIAG)) //Check if advance diag is enabled
    {
        if(newLine)
            Serial.println(msg);
        else
            Serial.print(msg);
    }
}

```

Code 4: ESP32 MQTT-Daten empfangen lassen

Wie schon im ersten Beispiel wird ein Objekt `espClient` erzeugt und dem ebenfalls erzeugten Objekt `mqttClient` übergeben. Beim Start vom ESP32 NodeMCU Module WLAN WiFi Development Board wird das WiFi über die Funktion `setupWiFi()` aufgebaut. Direkt im Anschluss der `setup()`-Funktion wird der MQTT-Broker mittels „`mqttClient.setServer(MQTT_BROKER,1883);`“ vorkonfiguriert. Eine `callback()`-Funktion braucht es an dieser Stelle nicht, da keine Daten empfangen und verarbeitet werden müssen.

Die loop()-Funktion ist daher fast identisch zum ersten Beispiel, mit der Ausnahme von der Zeile die Code 5 zeigt.

```
if(millis() - lLastMsg > iTimeDelay)
{
    lLastMsg = millis();
    sprintf(msg,MSG_BUFFER_SIZE,"%1d",millis()); //Convert message to char
    mqttClient.publish(PubTopic,msg,true); //Send to broker
}
```

Code 5: Codefragment zum Senden der aktiven Zeit

Dieser Teil sendet alle 2 Sekunden die aktuelle Laufzeit an das Topic `"/Client/ESP32"`. Das dieses Beispiel funktioniert, zeigt Abbildung 3. In diesem Fall zeigt MQTT.fx die Nachrichten vom MQTT-Broker an.

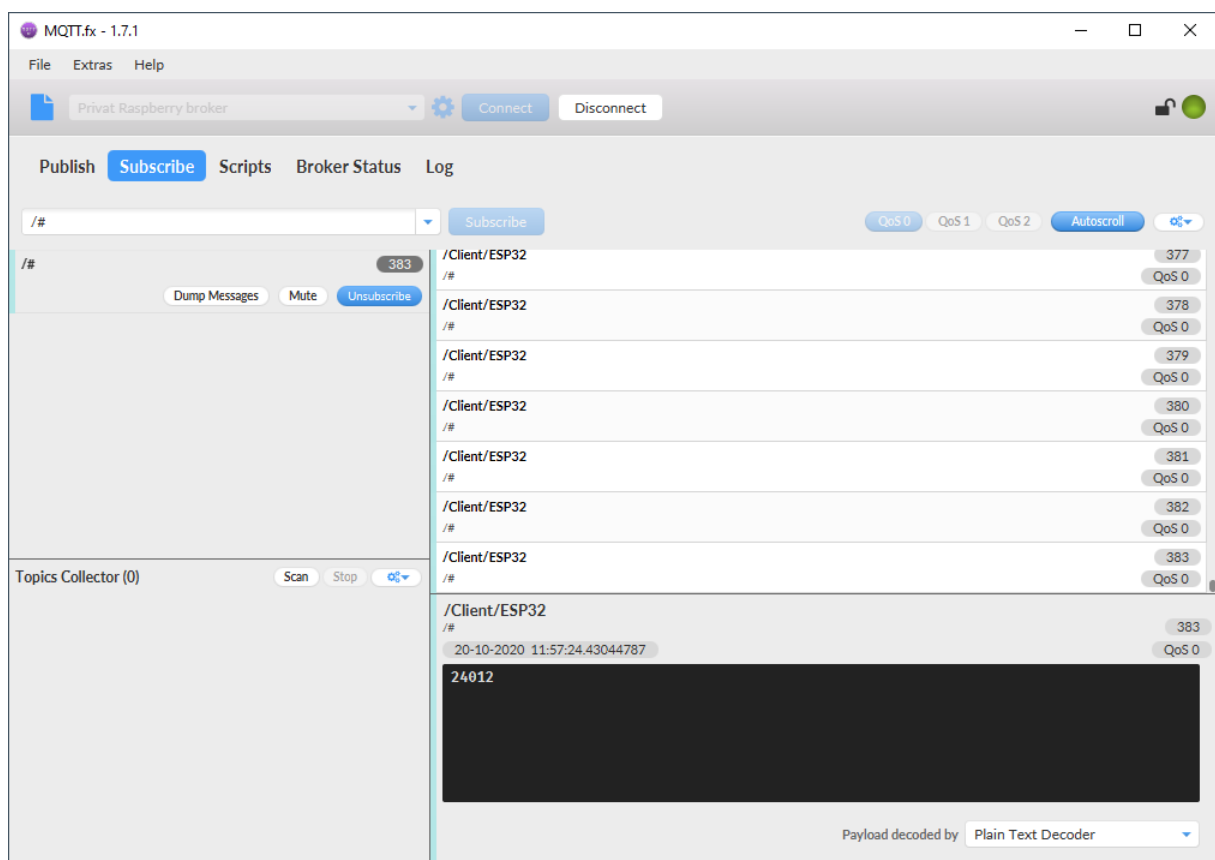


Abbildung 3: MQTTfx zeigt gesendete Daten vom ESP32 NodeMCU

MQTT-Beispiel mit mehreren Clients

In den letzten Beispielen, auch im ersten Blogbeitrag, sind wir immer von einem Broker und einem Client ausgegangen. Die Realität bei solchen MQTT-Projekten sieht aber oft anders aus.

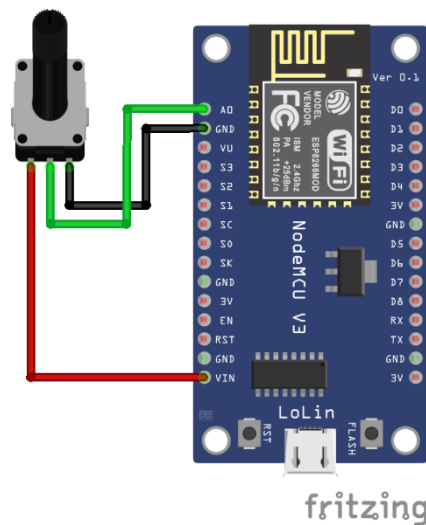
In der Praxis gibt es meist Clients, die Daten von Sensoren liefern, Clients, welche die gepushten Daten empfangen und weiterverarbeiten, und zu guter Letzt Clients, die ausführen.

Für das jetzt kommende Beispiel, wird ein NodeMCU Lua Amica Modul V2 einen Potentiometerwert überwachen und bei Änderung der neue Wert an den Broker gesenden.

Ein ESP32 NodeMCU Module WLAN WiFi Development Board subscribt diese Daten und mappt diese von 0-1024 auf 0-255, zudem published es von zwei BME/BMP280-Sensoren die Temperatur und den Luftdruck.

Diese gesammelten Daten subscribt ein Arduino Uno mit Ethernet-Shield, LED und I2C-LCD-Display und visualisiert die Daten. Die LED liefert das gemappte Signal vom Potentiometer und das Display zeigt die Temperatur und den Luftdruck der beiden Sensoren an.

Wir beginnen mit der NodeMCU Lua Amica Modul V2 und dem Potentiometer. Verkabeln Sie beide entsprechend der Abbildung 4. Dieser Teil ist einfach, da es nur drei Drähte angeschlossen werden müssen.



```

#define UPDATETIME 200
#define MINVALUECHANGE 1

const char* MQTT_BROKER = "raspberrypi"; //Name of the mqtt broker
String clientID = "NodeMCU_1"; //Clientname for MQTT-Broker
const char* PubTopicPoti = "/Client/ESP32/Poti/Value"; //Topic where to publish
const int iAnalogPin = A0; //Set analog pin
int iSensorValue = 0;
int iLastValue = 0;

//Create objects for mqtt
WiFiClient espClient;
PubSubClient mqttClient(espClient);
unsigned int iLastTime = 0;
char msg[MSG_BUFFER_SIZE];
void setup()
{
  Serial.begin(115200); //Start Serial monitor baudrate 115200
  delay(50);
  writeAdvanceDiag("SerialMonitor enabled", true);
  setupWifi();
  writeAdvanceDiag("Set MQTT-Server", true);
  mqttClient.setServer(MQTT_BROKER, 1883);
  writeAdvanceDiag("Set Callback-function", true);
  writeAdvanceDiag("Finish setup()-Function", true);
}

/*
=====
Function:  setupWifi
Returns:  void
Description: Setup wifi to connect to network
=====
*/
void setupWifi()
{
  Serial.println("Connection to: " + String(STASSID));
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

```

```

void loop() {
  // put your main code here, to run repeatedly:
  if (!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();

  if (millis() - iLastTime > UPDATETIME)
  {
    iSensorValue = analogRead(iAnalogPin); //Read analog value
    if (iSensorValue != iLastValue)
    {
      if (abs(iSensorValue - iLastValue) > MINVALUECHANGE) // Check if change is high enough
      {
        Serial.println("Sensorvalue: " + String(iSensorValue));
        snprintf(msg, MSG_BUFFER_SIZE, "%d", iSensorValue); //Convert message to char
        mqttClient.publish(PubTopicPoti, msg); //Send to broker
        iLastValue = iSensorValue;
      }
    }
    iLastTime = millis();
  }
}

/*
=====
Function:   reconnectMQTT
Returns:   void
Description: If there is no connection to MQTT, this function is
             called. In addition, the desired topic is registered.
=====
*/
void reconnectMQTT()
{
  while (!mqttClient.connected())
  {
    writeAdvanceDiag("Login to MQTT-Broker", true);
    if (mqttClient.connect(clientID.c_str()))
    {
      Serial.println("Connected to MQTT-Broker " + String(MQTT_BROKER));
    }
    else
    {
      writeAdvanceDiag("Failed with rc=" + String(mqttClient.state()), true);
      Serial.println("Next MQTT-Connect in 3 sec");
      delay(3000);
    }
  }
}

```

```

    }
}

/*
=====
Function:  writeAdvanceDiag
Returns:   void
Description: Writes advance msg to serial monitor, if
            ADVANCEDIAG >= 1
msg:       Message for the serial monitor
newLine:   Message with linebreak (true)
=====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
    if (bool(ADVANCEDIAG)) //Check if advance diag is enabled
    {
        if (newLine)
            Serial.println(msg);
        else
            Serial.print(msg);
    }
}

```

Code 6: NodeMCU Lua Amica Modul V2

Damit Sie schneller den Quellcode verstehen, sind viele Kommentare und Hinweise zu den Funktionen vermerkt worden.

Der nächste MicroController, der hier aufgebaut und seine Funktion bekommen soll, ist das ESP32 NodeMCU Module WLAN WiFi Development Board. Dieser erhält, da er zwei I2C-Schnittstellen besitzt, gleich zwei BME/BMP280-Sensoren und übernimmt das Mapping der analogen Potentiometerwerte. Auch hier ist die Verdrahtung einfach, siehe Abbildung 5.

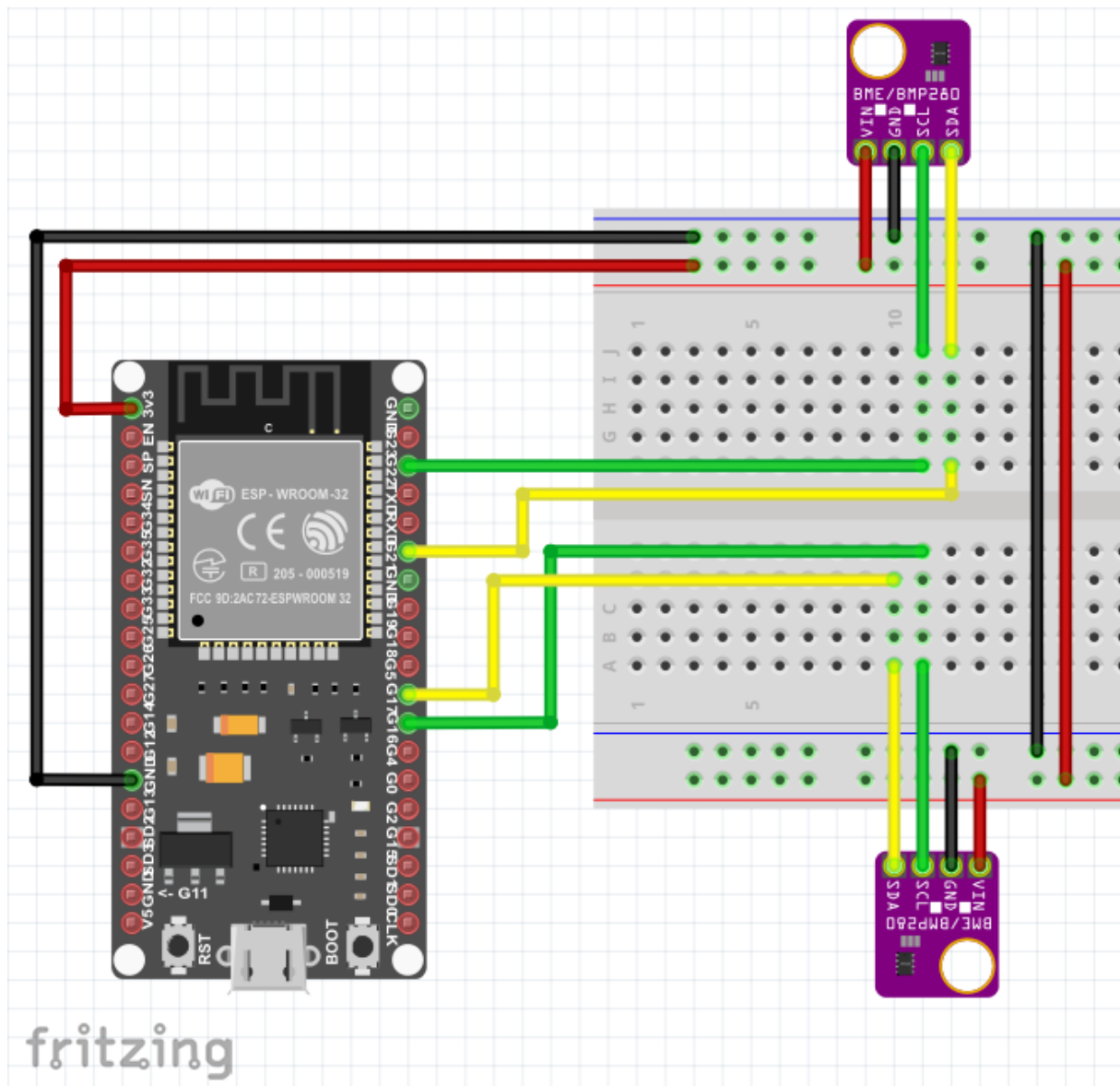


Abbildung 5: Verdrahtung ESP32 NodeMCU Module WLAN WiFi Development Board

Der Code für diese Aufgaben, siehe Code 7, ist etwas umfangreicher. Zunächst wird die WLAN-Verbindung, die MQTT-Verbindung und die Sensoren initialisiert. In der callback-Funktion wird der neu empfangene analoge Potentiometerwert auf einen Wert zwischen 0 bis 255 gemappt und wieder an den Broker gesendet. Hat einer oder beide BME/BMP280 neue Messdaten, die von den vorherigen Werten abweichen, werden diese ebenfalls an den Broker übermittelt. Damit der Quellcode auch bei Ihnen läuft, müssen Sie ihre WLAN-ID und das Passwort im Quellcode abändern.

```
//-----
// Example 3 ESP-NodeMCU with two BME transfer to
// mqtt-broker and mapping analog input
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 20. Oct 2020
// Update: 25. Oct 2020
//-----
```

```

#include <Adafruit_BME280.h>
#include <Wire.h>
#include <WiFi.h>
#include <PubSubClient.h> //Lib for MQTT Pub and Sub

//Define WiFi-Settings
#ifndef STASSID
#define STASSID "ENTER-WIFI-HERE" //Enter Wfi-Name
#define STAPSK "ENTER-PASS-HERE" //Enter Passkey
#endif

#define ADVANCEDIAG 1

#define I2C_SDA1 21
#define I2C_SCL1 22
#define I2C_SDA2 17
#define I2C_SCL2 16
#define NEXTUPDATE 2000

//Objects for I2C and BME
TwoWire I2Cone = TwoWire(0);
TwoWire I2Ctwo = TwoWire(1);
Adafruit_BME280 bmeOne;
Adafruit_BME280 bmeTwo;
unsigned long lastTime = 0;

const char* MQTT_BROKER = "raspberrypi"; //Name of the mqtt broker
const char* PubTopicTempOne = "/Client/ESP32/TempOne"; //Topic first temp
const char* PubTopicTempTwo = "/Client/ESP32/TempTwo"; //Topic second temp
const char* PubTopicPresOne = "/Client/ESP32/PressOne"; //Topic first pressure
const char* PubTopicPresTwo = "/Client/ESP32/PressTwo"; //Topic second pressure
const char* PubTopicPotiMap = "/Client/ESP32/PotiMapValue"; //Topic second pressure
const char* SUBTOPIC = "/Client/ESP32/Poti/Value"; //Topic subscribe poti value
String clientID = "ESP-DevKit_1"; //Clientname for MQTT-Broker

int iLastTempOne,iLastTempTwo,iLastPressOne,iLastPressTwo;

//Create objects for mqtt
WiFiClient espClient;
PubSubClient mqttClient(espClient);

#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("BME280 test");
  Serial.println("Init both I2C-Connections");

```



```

I2Cone.begin(I2C_SDA1, I2C_SCL1, 400000);
I2Ctwo.begin(I2C_SDA2, I2C_SCL2, 400000);
Serial.println("Make first BME talking to us");
bool bStatus;
//Init first sensor
bStatus = bmeOne.begin(0x76, &I2Cone);
if (!bStatus)
{
  Serial.println("Could not find a valid BME280 - 1 sensor, check wiring!");
  while (1);
}
else
  Serial.println("Valid BME280 - 1 sensor!");

//Init second sensor
bStatus = bmeTwo.begin(0x76, &I2Ctwo);
if (!bStatus)
{
  Serial.println("Could not find a valid BME280 - 2 sensor, check wiring!");
  while (1);
}
else
  Serial.println("Valid BME280 - 2 sensor!");
writeAdvanceDiag("Init Wifi", true);
setupWifi();
writeAdvanceDiag("Init Wifi - DONE", true);
writeAdvanceDiag("Set MQTT-Server", true);
mqttClient.setServer(MQTT_BROKER,1883);
writeAdvanceDiag("Set Callback-function", true);
mqttClient.setCallback(callback);
writeAdvanceDiag("Finish setup()-Function", true);
}

void loop() {
  // put your main code here, to run repeatedly:
  int iTempOne,iTempTwo,iPressOne,iPressTwo;
  if(!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();
  //Check after "NEXTUPDATE" if values has changed
  if(millis() - lastTime > NEXTUPDATE)
  {
    iTempOne = int(bmeOne.readTemperature()); //Get temp one
    iTempTwo = int(bmeTwo.readTemperature()); //Get temp two
    iPressOne = int(bmeOne.readPressure() / 100.0F); //Get press one
    iPressTwo = int(bmeTwo.readPressure() / 100.0F); //get press two
    if(iTempOne != iLastTempOne) //Check temp one changed and send
    {

```

```

    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iTempOne); //Convert message to char
    mqttClient.publish(PubTopicTempOne,msg,true); //Send to broker
    writeAdvanceDiag("Send Temp one: " + String(iTempOne), true);
    iLastTempOne = iTempOne;
}
if(iTempTwo != iLastTempTwo) //Check temp two changed and send
{
    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iTempTwo); //Convert message to char
    mqttClient.publish(PubTopicTempTwo,msg,true); //Send to broker
    writeAdvanceDiag("Send Temp two: " + String(iTempTwo), true);
    iLastTempTwo = iTempTwo;
}
if(iPressOne != iLastPressOne) //Check pressure one changed and send
{
    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iPressOne); //Convert message to char
    mqttClient.publish(PubTopicPresOne,msg,true); //Send to broker
    writeAdvanceDiag("Send Press one: " + String(iPressOne), true);
    iLastPressOne = iPressOne;
}
if(iPressTwo!= iLastPressTwo) //Check pressure two changed and send
{
    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iPressTwo); //Convert message to char
    mqttClient.publish(PubTopicPresTwo,msg,true); //Send to broker
    writeAdvanceDiag("Send Press two: " + String(iPressTwo), true);
    iLastPressTwo = iPressTwo;
}
lastTime = millis();
}

}

/*
* =====
* Function:   callback
* Returns:    void
* Description: Will automatical called, if a subscribed topic
*              has a new message
* topic:      Returns the topic, from where a new msg comes from
* payload:    The message from the topic
* length:     Length of the msg, important to get content
* =====
*/
void callback(char* topic, byte* payload, unsigned int length)
{
    String stMessage = "";
    writeAdvanceDiag("Message arrived from topic: " + String(topic), true);
    writeAdvanceDiag("Message length: " + String(length), true);
    for (int i = 0; i < length; i++)
        stMessage += String((char)payload[i]);
}

```

```

writeAdvanceDiag("Message is: " + stMessage, true);
//Map value and send the mapped value to mqtt broker
int iValue,iMapValue;
iValue = stMessage.toInt();
iMapValue = map(iValue,0,1024,0,255);
snprintf(msg,MSG_BUFFER_SIZE, "%1d",iMapValue); //Convert message to char
writeAdvanceDiag("Send mapped PotiValue: " + String(iMapValue), true);
mqttClient.publish(PubTopicPotiMap,msg,true); //Send to broker

}

/*
* =====
* Function:   setupWifi
* Returns:    void
* Description: Setup wifi to connect to network
* =====
*/
void setupWifi()
{
  Serial.println("Connection to: " + String(STASSID));
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

/*
* =====
* Function:   writeAdvanceDiag
* Returns:    void
* Description: Writes advance msg to serial monitor, if
*             ADVANCEDIAG >= 1
* msg:        Message for the serial monitor
* newLine:     Message with linebreak (true)
* =====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
  if(bool(ADVANCEDIAG)) //Check if advance diag is enabled
  {
    if(newLine)

```

```

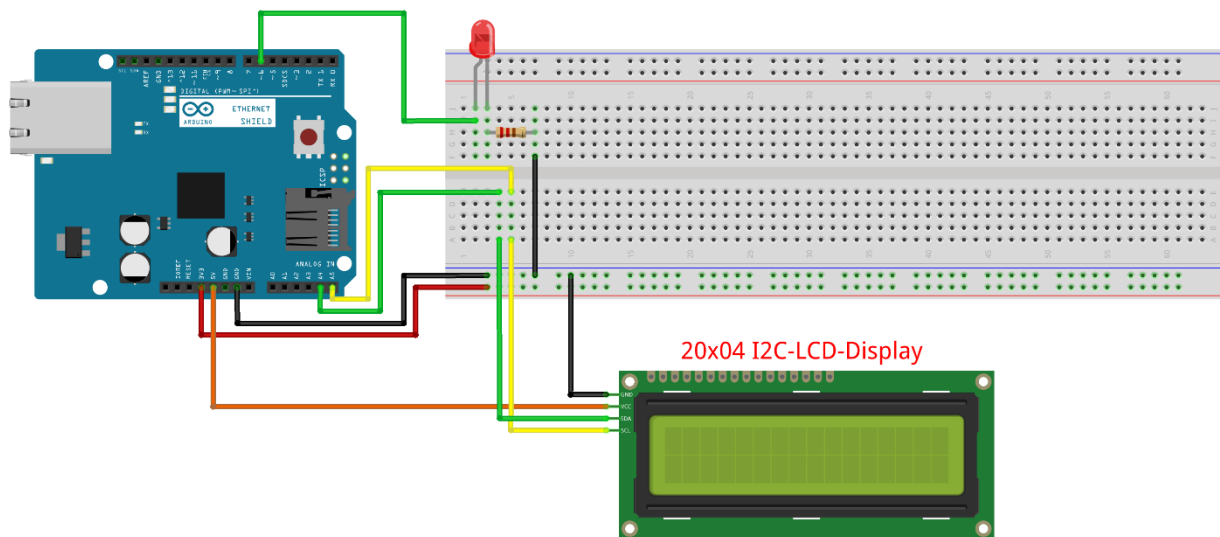
        Serial.println(msg);
    else
        Serial.print(msg);
    }
}

/*
 * =====
 * Function:   reconnectMQTT
 * Returns:    void
 * Description: If there is no connection to MQTT, this function is
 *              called. In addition, the desired topic is registered.
 * =====
 */
void reconnectMQTT()
{
    while(!mqttClient.connected())
    {
        writeAdvanceDiag("Login to MQTT-Broker", true);
        if(mqttClient.connect(clientID.c_str()))
        {
            Serial.println("Connected to MQTT-Broker " +String(MQTT_BROKER));
            writeAdvanceDiag("Subscribe topic '" + String(SUBTOPIC)+ "'", true);
            mqttClient.subscribe(SUBTOPIC,1); //Subscribe topic "SUBTOPIC"
        }
        else
        {
            writeAdvanceDiag("Failed with rc=" +String(mqttClient.state()), true);
            Serial.println("Next MQTT-Connect in 3 sec");
            delay(3000);
        }
    }
}

```

Code 7: ESP32 NodeMCU Module WLAN WiFi Development Board

Im letzten Schritt sollen die gesammelten Daten visualisiert werden. Dazu wird der Arduino Uno mit aufgesteckten Ethernetshield, das 20x04 LCD I2C-Display, sowie eine einfarbige LED verwendet. Die genaue Verdrahtung können Sie in Abbildung 6 sehen. Bedenken Sie, dass das 20x04 LCD I2C-Display eine Spannungsversorgung von 5V benötigt.



fritzing

Abbildung 6: Arduino Uno Ethernetshield für Ausgabe

Auch bei der Programmierung des Arduino Uno gibt es keine großen Überraschungen. Zunächst wird das Netzwerk initialisiert und die Verbindung überprüft. Im nächsten Schritt werden das Display und die Verbindung zum MQTT-Broker aufgebaut. Für die Ausgabe der Daten ist die callback()-Funktion relevant, die alle Daten vom Broker empfängt und bei Änderungen einer oder mehrerer Werte diese im Display oder über die LED anzeigt.

```
//-----
// Example 3 Arduino Uno Ethernetshield to receive
// data from broker and show on LCD and LED
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 25. Oct 2020
// Update: 26. Oct 2020
//-----
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <PubSubClient.h> //Lib for MQTT Pub and Sub
#include <SPI.h>
#include <Ethernet.h>

#define ADVANCEDIAG 1

LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x27 for a 16 chars and 2 line display
const int iAnalogOut = 6;
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; //MAC-Adress for shield
IPAddress ip(192, 168, 178, 177); //Static IP-Address for arduino
IPAddress myDns(192, 168, 178, 1); //IP-Adress router
char server[] = "www.google.com"; //Check if Arduino is online

String clientID = "Arduino_Uno"; //Clientname for MQTT-Broker
//Topics for subscribe
const char* MQTT_BROKER = "raspberrypi"; //Name of the mqtt broker
const char* SubTopicTempOne = "/Client/ESP32/TempOne"; //Topic first temp
```

```
const char* SubTopicTempTwo = "/Client/ESP32/TempTwo"; //Topic second temp
const char* SubTopicPresOne = "/Client/ESP32/PressOne"; //Topic first pressure
const char* SubTopicPresTwo = "/Client/ESP32/PressTwo"; //Topic second pressure
const char* SubTopicPotiMap = "/Client/ESP32/PotiMapValue"; //Topic mapped Poti
```

```
//Objects for ethernet-com
EthernetClient client;
PubSubClient mqttClient(client);
```

```
//Some vars for update
bool bUpdateDisplay = false;
bool bUpdatePWM = false;
int iLastTempOne,iLastTempTwo,iLastPressOne,iLastPressTwo,iLastPotiMap;
```

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.println("Arduino Uno - Monitor");
  pinMode(iAnalogOut, OUTPUT);
  Ethernet.init(10); // Most Arduino shields use digital Pin 10
  lcd.init(); //Init LCD
  lcd.backlight(); //Backlight on
  lcd.clear(); //Clear old content
  bUpdateDisplay = true;
  Ethernet.begin(mac, ip); //Init ethernet-shield
  // Check for Ethernet hardware present
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("Ethernet shield was not found. Sorry, can't run without hardware. :(");
    while (true) {
      delay(1); // do nothing, no point running without Ethernet hardware
    }
  }
  //Check if there is com to router
  while (Ethernet.linkStatus() == LinkOFF) {
    Serial.println("Ethernet cable is not connected.");
    delay(500);
  }

  // give the Ethernet shield a second to initialize:
  delay(1000);
  Serial.println("connecting...");

  //Check if system is able to communicate
  if (client.connect(server, 80)) {
    Serial.print("connected to ");
    Serial.println(client.remoteIP());
    // Make a HTTP request:
```

```

client.println("GET /search?q=arduino HTTP/1.1");
client.println("Host: www.google.com");
client.println("Connection: close");
client.println();
} else {
    // if you didn't get a connection to the server:
    Serial.println("connection failed");
}
//Init MQTT
writeAdvanceDiag("Set MQTT-Server", true);
mqttClient.setServer(MQTT_BROKER,1883);
writeAdvanceDiag("Set Callback-function", true);
mqttClient.setCallback(callback);
writeAdvanceDiag("Finish setup()-Function", true);
}

void loop() {
    // put your main code here, to run repeatedly:
    if(!mqttClient.connected())
        reconnectMQTT();

    mqttClient.loop();

    if(bUpdateDisplay)
    {
        UpdateDisplay();
        bUpdateDisplay = false;
    }
    if(bUpdatePWM)
    {
        analogWrite(iAnalogOut, iLastPotiMap); //Write new analog value to LED-Pin
        bUpdatePWM = false;
    }
}

/*
* =====
* Function:   writeAdvanceDiag
* Returns:    void
* Description: Writes advance msg to serial monitor, if
*              ADVANCEDIAG >= 1
* msg:        Message for the serial monitor
* newLine:    Message with linebreak (true)
* =====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
    if(bool(ADVANCEDIAG)) //Check if advance diag is enabled
    {

```

```

    if(newLine)
        Serial.println(msg);
    else
        Serial.print(msg);
}
}

/*
 * =====
 * Function:   reconnectMQTT
 * Returns:    void
 * Description: If there is no connection to MQTT, this function is
 *              called. In addition, the desired topic is registered.
 * =====
 */
void reconnectMQTT()
{
    while(!mqttClient.connected())
    {
        writeAdvanceDiag("Login to MQTT-Broker", true);
        if(mqttClient.connect(clientID.c_str()))
        {
            Serial.println("Connected to MQTT-Broker " +String(MQTT_BROKER));
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicTempOne)+ "'", true);
            mqttClient.subscribe(SubTopicTempOne,1); //Subscribe topic "SubTopicTempOne"
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicTempTwo)+ "'", true);
            mqttClient.subscribe(SubTopicTempTwo,1); //Subscribe topic "SubTopicTempTwo"
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicPresOne)+ "'", true);
            mqttClient.subscribe(SubTopicPresOne,1); //Subscribe topic "SubTopicPresOne"
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicPresTwo)+ "'", true);
            mqttClient.subscribe(SubTopicPresTwo,1); //Subscribe topic "SubTopicPresTwo"
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicPotiMap)+ "'", true);
            mqttClient.subscribe(SubTopicPotiMap,1); //Subscribe topic "SubTopicPotiMap"
        }
        else
        {
            writeAdvanceDiag("Failed with rc=" +String(mqttClient.state()), true);
            Serial.println("Next MQTT-Connect in 3 sec");
            delay(3000);
        }
    }
}

/*
 * =====
 * Function:   callback
 * Returns:    void
 * Description: Will automatical called, if a subscribed topic
 *              has a new message

```



```

* topic:    Returns the topic, from where a new msg comes from
* payload:   The message from the topic
* length:    Length of the msg, important to get content
* =====
*/
void callback(char* topic, byte* payload, unsigned int length)
{
    String stMessage = "";
    for (int i = 0; i < length; i++)
        stMessage += String((char)payload[i]);

    //Check if temp one has changed
    if(String(topic) == "/Client/ESP32/TempOne")
    {
        if(iLastTempOne != stMessage.toInt())
        {
            iLastTempOne = stMessage.toInt();
            bUpdateDisplay = true;
        }
    }
    //Check if temp two has changed
    if(String(topic) == "/Client/ESP32/TempTwo")
    {
        if(iLastTempTwo != stMessage.toInt())
        {
            iLastTempTwo = stMessage.toInt();
            bUpdateDisplay = true;
        }
    }
    //Check if pressure one has changed
    if(String(topic) == "/Client/ESP32/PressOne")
    {
        if(iLastPressOne != stMessage.toInt())
        {
            iLastPressOne = stMessage.toInt();
            bUpdateDisplay = true;
        }
    }
    //Check is pressure two has changed
    if(String(topic) == "/Client/ESP32/PressTwo")
    {
        if(iLastPressTwo != stMessage.toInt())
        {
            iLastPressTwo = stMessage.toInt();
            bUpdateDisplay = true;
        }
    }
    //Check if mapped poti value has changed
    if(String(topic) == "/Client/ESP32/PotiMapValue")

```

```

{
  if(iLastPotiMap != stMessage.toInt())
  {
    iLastPotiMap = stMessage.toInt();
    bUpdatePWM = true;
  }
}
}

/*
* =====
* Function:   UpdateDisplay
* Returns:    void
* Description: Display new values on I2C-Display
* =====
*/
void UpdateDisplay()
{
  lcd.clear();
  lcd.home();
  lcd.print("Temp one[C]: ");
  lcd.print(iLastTempOne);
  lcd.setCursor(0,1);
  lcd.print("Temp two[C]: ");
  lcd.print(iLastTempTwo);
  lcd.setCursor(0,2);
  lcd.print("Press one[hPa]: ");
  lcd.print(iLastPressOne);
  lcd.setCursor(0,3);
  lcd.print("Press two[hPa]: ");
  lcd.print(iLastPressTwo);
}

```

Code 8: Arduino Uno zur Anzeige der MQTT-Daten

Ist alles korrekt verdrahtet und der Code auf alle MicroController überspielt, sollten Sie schnell ein Ergebnis auf dem LCD-Display und der angeschlossenen LED sehen. Beim Drehen vom Potentiometer wird die Helligkeit der LED geändert und die aktuelle Temperatur und Druck beider Sensoren sollten visualisiert werden. Wie das aussehen kann, zeigt Abbildung 7.

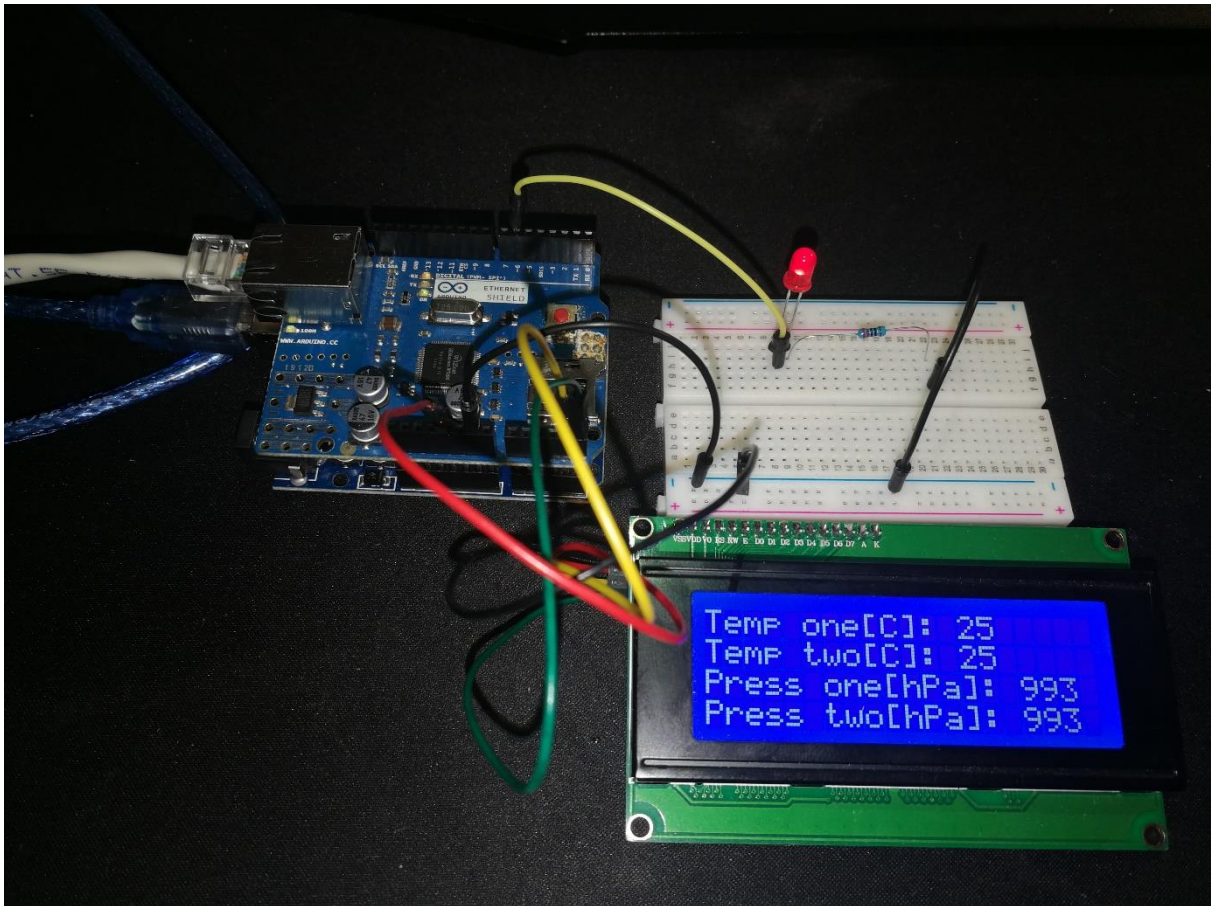


Abbildung 7: Ausgabe von Sensordaten und LED-Helligkeit

Wollen Sie alle Daten ihres Brokers sehen, kann ich Ihnen das Tool [MQTT.fx](https://mqtt.fx) empfehlen. Damit können Sie sowohl Daten vom Broker subscriben, als auch publishen. Probieren Sie gerne auch aus weiterer Hardware an die MicroController anzuschließen und die Daten an den Broker zu senden. Der Arduino Uno kann, über eine Schleife, die Daten abwechselnd anzeigen.

Mit Abschluss dieses Beitrages haben Sie sowohl die Grundlagen, als auch ein komplexeres praktisches Beispiel erhalten. Im nächsten Teil werden die Grundlagen und das praktische Beispiel genutzt, um einen Rollroboter zu steuern.

Dieses und weitere Projekte finden sich auf GitHub unter <https://github.com/M3taKn1ght/Blog-Repo>