

**Daten von Sensoren ermitteln und mittels ESP8266 Mikrokontroller an
Raspberry Pi 3 senden und auswerten.**

s0546864 - Sylwester Korpik
s0532766 - Nikiema Wendpanga Mohamed
s0554182 - Gires Ntchouayang Nzeunga

Inhaltsverzeichnis

1. Einleitung

2. Grundlagen

2.1 Grundlagen IOT

2.2 Die drei Schichten eines IOT-Modells

2.2.1 Die Ebene der Wahrnehmung

2.2.2 Die Netzwerkschicht

2.2.3 Die Anwendungsschicht

2.3 Message Queuing Telemetry Transport (MQTT)

2.4 Sensoren

2.4.1 Luft- und Feuchtigkeitssensor (DHT-11)

2.4.2 Ultraschallsensor (HC-SR04)

2.5 Microcontroller

2.5.1 ESP8266-01

2.5.2 Raspberry Pi

2.6 Stromversorgung und Programmierung

3. Eigene Arbeit / Umsetzung

3.1 MQTT Broker auf dem Raspberry Pi einrichten

3.2 ESP8266 mit dem FTDI Adapter verbinden

3.3 ESP8266 mit der Arduino IDE programmieren

3.4 HC-SR04 und DHT-11 mit dem ESP8266 verbinden

3.5 Visualisierung mit Node.js einrichten und Daten auswerten

4. Ergebnis/Ausblick

5. Literatur

1. Einleitung

1999 wurde der Begriff „Internet of Things“ das erste mal von Kevin Ashton verwendet und bezeichnet eine Infrastruktur die es ermöglicht Gegenstände miteinander zu vernetzen und zusammenarbeiten zu lassen.

Seitdem wächst das Internet der Dinge stetig und ermöglicht immer mehr Geräten sich miteinander zu vernetzen und Daten auszutauschen. Um dieses Daten auszutauschen braucht man ein geeignetes Protocol zur Übertragung wie z.b HTTP. Das Problem besteht darin das HTTP für diesen gebrauch nicht geeignet ist da es unter anderem zu große Datenmengen verschickt und zu langsam ist.

Die Lösung für dieses Problem lautet MQTT. MQTT ist ein offenes Nachrichtenprotokoll für Machine-to-Machine Kommunikation. Unser Projekt beschäftigt sich mit diesem Protokoll. Die Aufgabe besteht darin einen MQTT Broker einzurichten der als Server dient um Daten zu erhalten und auszuwerten.

Die Daten soll der Broker von Sensoren erhalten. Unter anderem von einem Luft- und Feuchtigkeitssensor sowie einem Ultraschallsensor. Die Sensoren sollen die ermittelten Daten an einen Microcontroller weiterleiten, einen sogenannten ESP8266. Dieser soll die Daten über einen Hotspot bzw. das lokale Netzwerk weiter an den Broker leiten, der die Daten auswertet.

Die ESP Module arbeiten hier als Clients im MQTT Protocol.

2. Grundlagen

2.1 Grundlagen IOT

Der Ausdruck des Internet of Things (IoT) wurde 1999 von Kevin Ashton in einer Präsentation zur Charakterisierung einer Informationsarchitektur auf Basis des Internets zusammengestellt. Danach wurde der Begriff populär und weit verbreitet. Das IoT ist die Verbindung des Internets mit allen Dingen, die uns umgeben, und umfasst die Wissenschaften der Elektronik und der Telekommunikationswissenschaften. Mit all den möglichen Innovationen in diesem Bereich wird das IoT noch leistungsfähiger und seine Definition wird auch flexibler, um der technologischen Entwicklung zu folgen.

Nach dem Wirtschaftslexikon Gabler ist das IoT "die Vernetzung von Gegenständen mit dem Internet, damit diese Gegenstände selbstständig über das Internet kommunizieren und so verschiedene Aufgaben für den Besitzer erledigen können. Der Anwendungsbereich erstreckt sich dabei von einer allg. Informationsversorgung über automatische Bestellungen bis hin zu Warn- und Notfallfunktionen.“ Somit kann IoT also als das neue Bedürfnis aller Entitäten beschrieben werden Maschinen miteinander kommunizieren zu lassen.

2.2 Die drei Schichten eines IOT-Modells

Das Konzept des Internets der Dinge ist seit mehr als einem Jahrzehnt Gegenstand von Forschung, aber dennoch sind viele Aspekte nicht klar definiert. Zum Beispiel gibt es heute keine standardisierte und spezifische Architektur für IoT.

Trotz dieser mangelnden Kompatibilität gibt es eine allgemein bekannte Dreischicht-Architektur (Abbildung 1). Diese Schichten sind: die Wahrnehmungsschicht(auch Physical Layer), die Netzwerkschicht und die Anwendungsschicht.

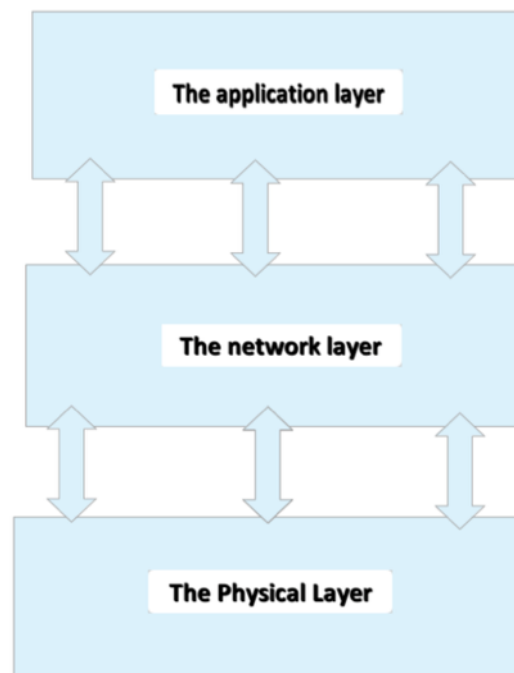


Abbildung 1: Die drei Schichten eines IOT Modells

2.2.1 Die Ebene der Wahrnehmung (Physical Layer)

Die Hauptaufgabe der Wahrnehmungsschicht besteht darin, physikalische Eigenschaften wie z.B. Temperatur, Feuchtigkeit, Lichtstärke, Geschwindigkeit durch verschiedene Sensorvorrichtungen zu erkennen und diese Information in digitale Signale umzuwandeln. Die Objekte dieser Schicht können Erkennungsfähigkeiten und / oder Betätigungsfähigkeiten aufweisen.

Die Wahrnehmungsebene besteht aus:

Sensoren: Erkennen physikalische Eigenschaften und Konvertieren dieser Eigenschaften in digitale Signale.

Aktoren: Empfangen Befehle zum Ausführen von Aktionen zu bestimmte Zeitpunkten.

Endgeräte: Mikrocontroller die Verarbeitungs- und Kommunikationsmöglichkeiten für Sensoren und Aktoren bieten.

2.2.2 Die Netzwerkschicht

Die Netzwerkschicht ist die Schicht, die die Daten, die von der Wahrnehmungsebene empfangen werden zu einer Datenbank, einem Server oder einem Verarbeitungszentrum überträgt.

Die wichtigsten Technologien, um diese Ebene zu erreichen, sind:

2G / 3G / LTE-Mobilfunktechnologien, Wi-Fi, Bluetooth, Zigbee oder Ethernet. Mit diesen verschiedenen Technologien können wir mehrere Objekte bearbeiten, die in Zukunft verbunden werden sollen. Für unsere Projekt wird Wi-Fi verwendet und MQTT als Übertragungsprotokoll. Das Internet der Dinge ist ein riesiges Netzwerk, das nicht nur eine Vielzahl von Objekten verbindet, sondern auch heterogene Netzwerke umfasst.

Die Netzwerkschicht besteht aus:

Kommunikationsprotokolle: welche für die Endgeräte verwendet werden, z.B. MQTT, HTTP

Gateway-Station: um den Informationsfluss zwischen den Endgeräte und dem Internet zu steuern.

2.2.3 Die Anwendungsschicht

Die Anwendungsschicht analysiert die von der Netzwerkschicht empfangenen Informationen. Diese Ebene bietet Anwendungen für alle Arten von technologischen Herausforderungen. Diese Anwendungen favorisieren das Internet der Dinge, weshalb diese Schicht eine wichtige Rolle bei der Verbreitung von IoT spielt.

Die Anwendungsebene enthält:

IoT-Cloud-Plattformen: virtuelle Online-Datenbanken, die Informationen speichern und eine Visualisieren dieser Informationen (z.B. Tabellen, Grafiken) für Endbenutzer zur Verfügung stellen

Software-Anwendungen: für Smartphones, Tablets, Desktop-PCs mit grafischen Oberflächen (GUI) zur Überwachung und Steuerung von Endgeräten.

The Internet of Things

Get in on the next big thing

WHAT IS THE INTERNET OF THINGS (IoT)?

The IoT has three main parts:

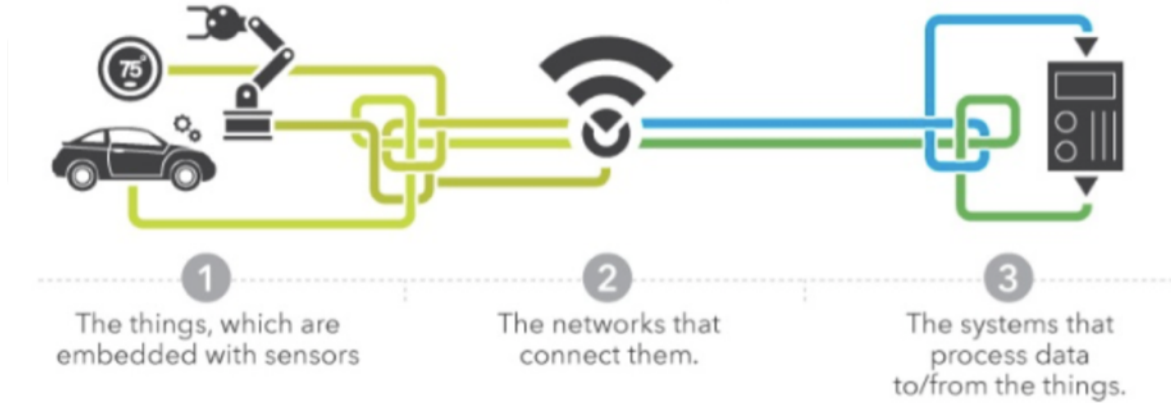


Abbildung 2: drei Schichten Modell

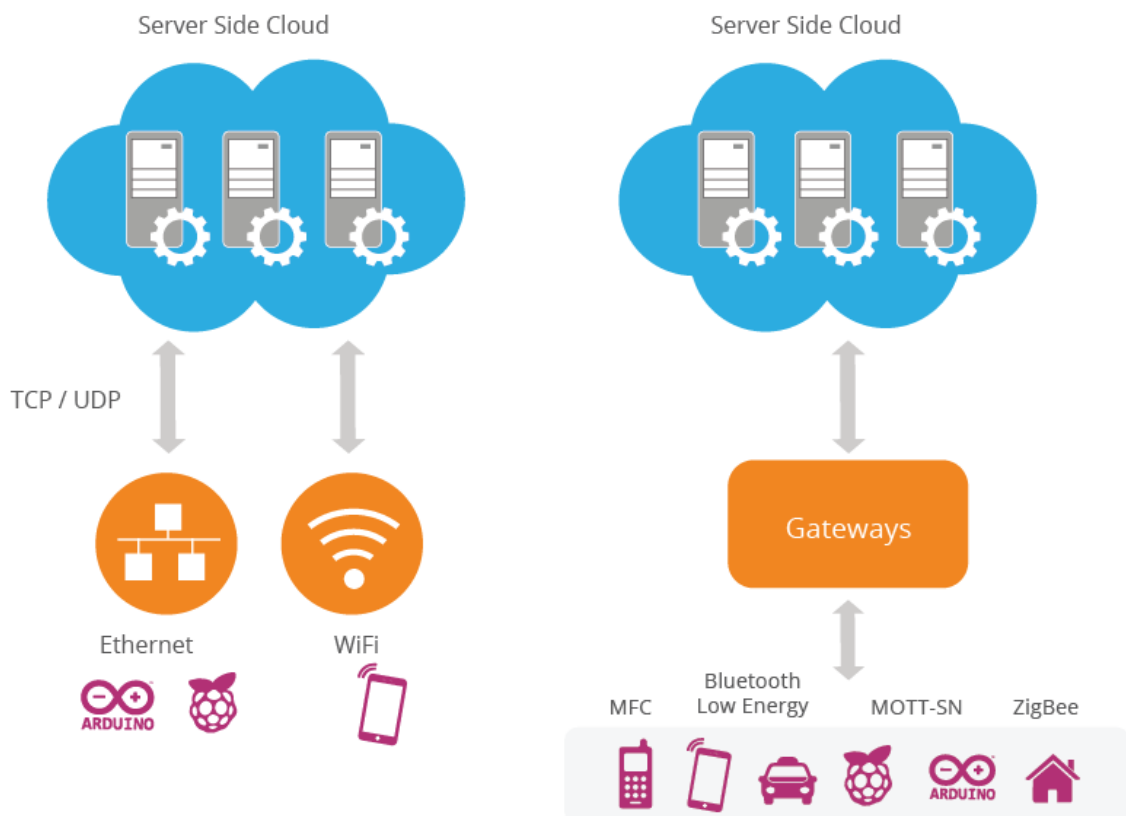


Abbildung 3: Referenz Architektur für das Internet der Dinge

2.3 Message Queuing Telemetry Transport (MQTT)

MQTT (Message Queuing Telemetry Transport) ist ein offenes Nachrichtenprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkten Netzwerken. Entsprechende Geräte reichen von Sensoren und Aktoren, Mobiltelefonen, Eingebetteten Systemen in Fahrzeugen oder Laptops bis zu voll entwickelten Rechnern. Das Protokoll schlägt mit einem ereignisgesteuerten Push-Ansatz einen anderen Weg ein als beispielsweise HTTP, das auf Request/Response basiert.

1) MQTT ist komplett **datenagnostisch**. Es ist daher geeignet, Daten jeder Art zu übertragen, egal ob es sich um Text oder binärkodierte Inhalte handelt.

2) MQTT ist **einfach**. Die Konzepte sind leicht zu erlernen und eigene Clientimplementierungen problemlos zu realisieren.

3) MQTT **erfindet das Rad nicht neu**. Es baut auf TCP auf, und die Übertragung kann jederzeit mittels SSL/TLS verschlüsselt werden.

4) MQTT **ermöglicht echte Push-Kommunikation**: Anders als bei anderen Protokollen gibt es bei MQTT kein Polling. Nachrichten werden sofort verteilt, wenn ein Ereignis auftritt. Das schont Bandbreite und CPU, da MQTT-Clientanwendungen reagieren können, sobald eine Nachricht ankommt, anstatt beim Server nach neuen Nachrichten zu fragen.

2.4 Sensoren

2.4.1 DHT11 Sensor

Der DHT11 ist ein Sensor zum messen von Temperatur und Luftfeuchtigkeit und wird entweder über vier Pins oder über ein Breakoutboard mit drei Pins ausgeliefert und arbeitet mit 3,5 bis 5,5 Volt Gleichstromversorgung. Er kann Temperaturen in Bereich von 0 bis 50 Grad Celsius mit einer Genauigkeit von $\pm 2^{\circ}\text{C}$ und die relative Luftfeuchtigkeit von 20 bis 95% mit einer Genauigkeit von $\pm 5\%$ messen. Der Sensor verbraucht zwischen 0,5 mA und 2,5 mA und hat eine Dimension von (Länge, Breite, Höhe): 15.5mm, 12mm, 5.5mm. Laut dem technischen Datenblatt besitzt es eine ausgezeichnete Langzeitstabilität und eine sehr schnelle Reaktionszeit.

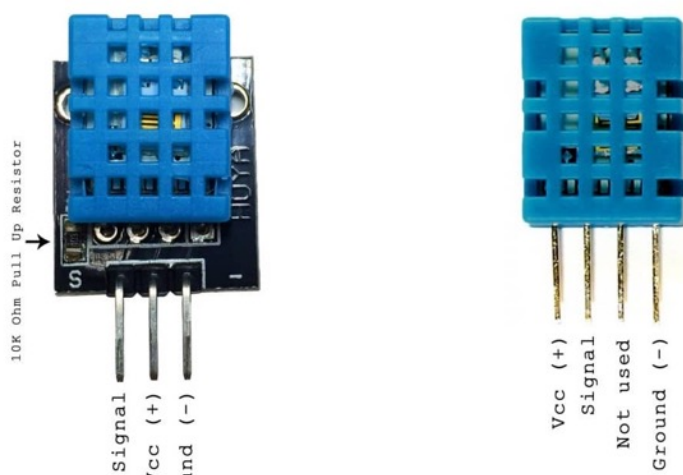


Abbildung 4: DHT11 mit Breakoutboard und 3 Pins und DHT11 mit 4Pins

2.4.2 Ultraschall Sensor (HC-SR04)

Der Ultraschallsensor HC-SR04 wird zur Entfernungsmessung bzw. Abstandsbestimmung zwischen 2cm und ca. 3m mit einer Auflösung von 3mm verwendet und sendet einen hochfrequenten Signalton aus. Er misst die Zeit t , bis das von einem Objekt reflektierte Signal wieder bei ihm eintrifft. Diese Zeitmessung ist abhängig von der Ausbreitungsgeschwindigkeit von Schall in Luft. Es benötigt nur eine einfache Versorgungsspannung von 5V bei einer Stromaufnahme von $< 2\text{mA}$. Nach „Triggerung“ mit einer fallenden Flanke (TTL - Pegel) misst das Modul selbstständig die Entfernung und wandelt diese in ein PWM Signal welches am Ausgang zur Verfügung steht. Ein Messintervall hat eine Dauer von 20ms. Es können also 50 Messungen pro Sekunde durchgeführt werden.

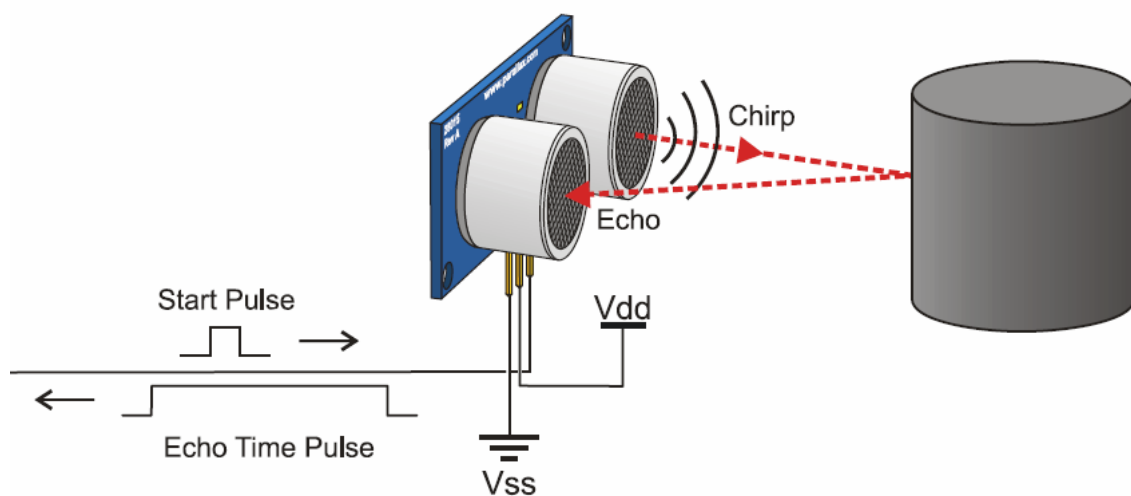


Abbildung 5: Sensor Ping Operation

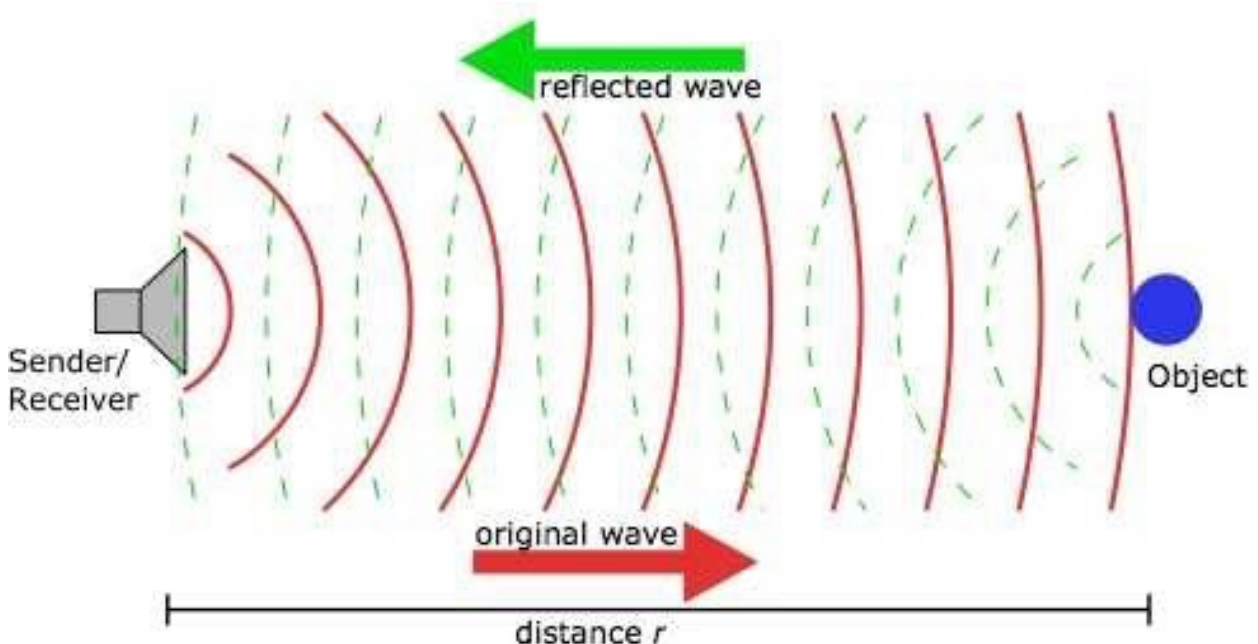


Abbildung 6: Funktionsweise des HC-SR04

2.5 Microcontroller

2.5.1 ESP8266-01

Der ESP8266 ist ein programmierbares WLAN-Funkmodul mit zahlreichen Schnittstellen wie UART, I²C und SPI.

Es gibt es ihn in verschiedenen Versionen, die sich größtenteils in der Anzahl der I/O Pins und der Größe des Flash-Speichers unterscheiden. An einigen der Module kann man neben der internen zudem auch eine externe Antenne anschließen. Wir benutzen in unserem Projekt den ESP8266-01 BLACK. Dieses Modul verfügt über gerade einmal zwei GPIO Ports, einen 2×4 poligen Pinheader, eine rote Power-LED und eine blaue UART-Transmit-LED.

Trotz dieser Beschränkungen reicht der ESP8266-01 aus, der gerade einmal die Größe einer zwei Euro Münze hat, um einen Sensor anzuschließen und so die Sensordaten per WLAN an einen Server zu senden.

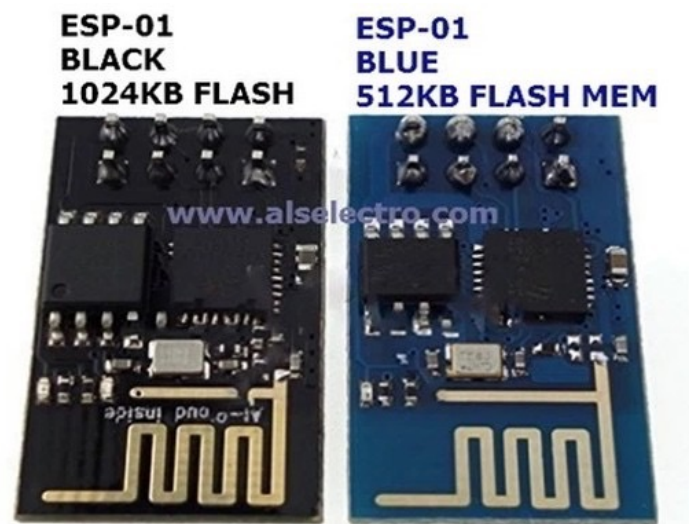
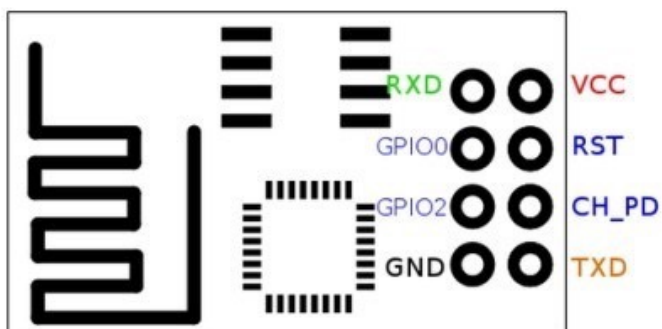


Abbildung 7: ESP8266-01 Black und ESP8266-01 Blue



Label	Signal
VCC	3.3V (3.6V max) supply voltage
GND	Ground
TXD	Transmit Data (3.3V level)
RXD	Receive Data (3.3V level)
CH_PD	Chip Power down: (LOW = power down active)
GPIO0	General Purpose I / O 0
GPIO2	General Purpose I / O 2
RST	Reset (reset = LOW active)

Abbildung 8: ESP8266-01 Pin Belegung

2.5.2 Raspberry Pi

Das Raspberry Pi Development Board [3] ist ein kleiner Broadcom BCM 2835 SoC-basierter ARM11 Power Minicomputer. Der Raspberry Pi kann dank seiner eingebauten GPU und audiovisuellen Fähigkeiten einfach in den Monitor eingesteckt werden. Der Raspberry Pi ist leicht programmierbar und durch leistungsfähige Sprachen wie C, Python etc., ermöglicht dieser die Daten zu speichern und zu analysieren. Die integrierte WLAN, BLE(Bluetooth Low Energy), Speicherkapazität dieses Boards und der verfügbare RAM sind im Vergleich zu anderen Boards sehr groß und ermöglichen es, in den meisten Fällen als IoT Server zu fungieren.

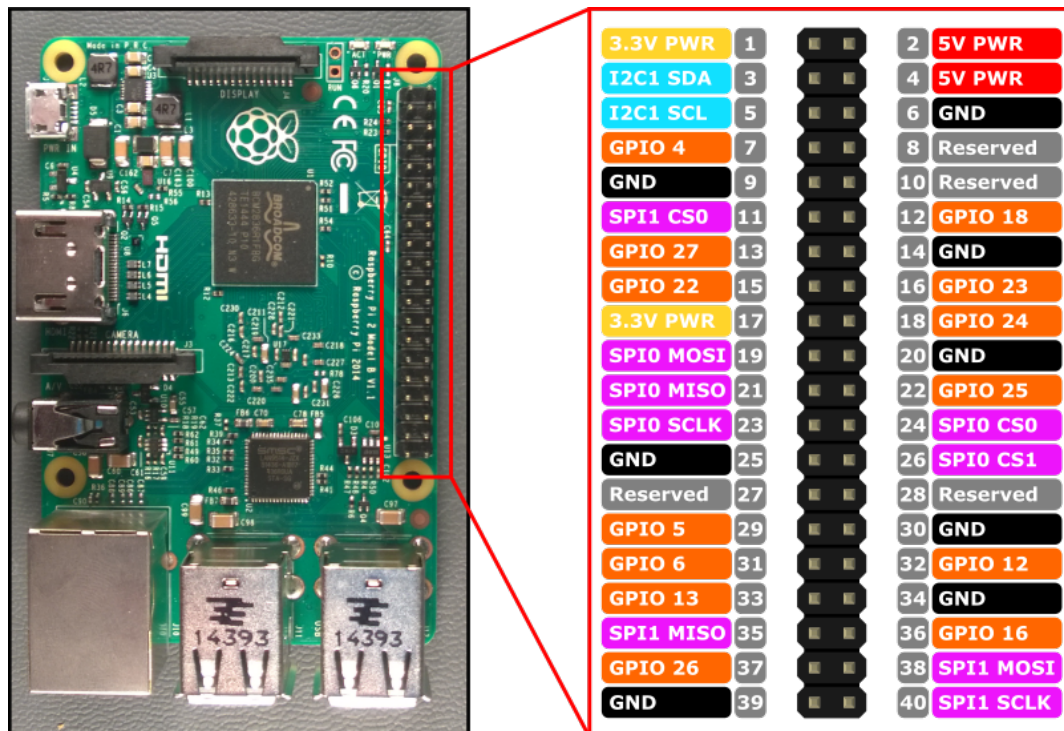


Abbildung 9: Raspberry Pi 3 Pin Belegung

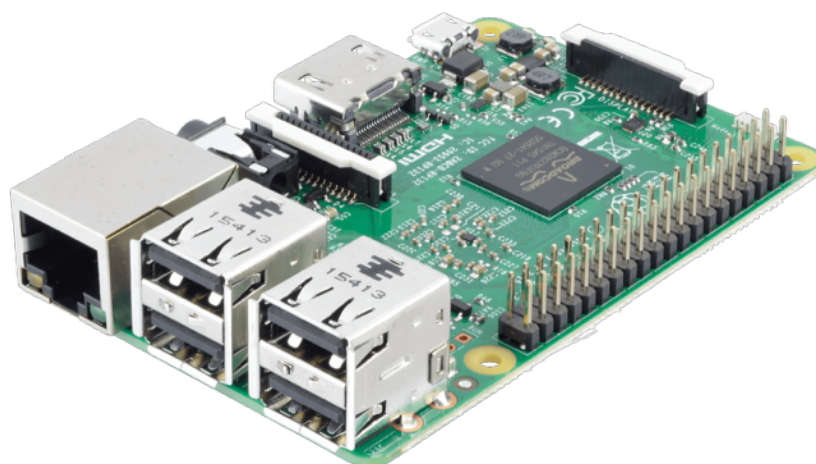


Abbildung 10: Raspberry Pi 3

2.6 Stromversorgung und Programmierung

Der ESP8266-01 wird über ein FTDI Kabel programmiert und mit Strom versorgt. FTDI bedeutet Future Technology Devices International und ist der dazugehörige Firmenname. FTDI wurde für ihre USB-RS232-Interface-Chips bekannt, mit denen es möglich ist eine serielle Schnittstelle über USB verfügbar zu machen. Diese FTDI Kabel gibt es in 5V oder 3,3V Versionen. Wenn man die 5V Version besitzt, muss man unbedingt darauf achten, dass der ESP8266 nur mit 3,3V betrieben wird, die Spannung auf 3,3V mit einem Spannungsregler herunter regelt, einen Spannungsteiler aus zwei Widerständen baut oder ein externes Netzteil verwenden.

Dazu muss auch darauf geachtet werden das der VCC Pin des FTDI nicht mit dem ESP8266 verbunden ist. Wenn man ein FTDI Kabel mit 3,3V Ausgang besitzt, kann der VCC des FTDI direkt mit dem VCC Pin des ESP8266 verbunden werden. Programmiert wird der ESP8266-01 über die Arduino IDE. Damit dies gelingt muss jedoch zuvor die ESP8266 library installieren werden. Wird der FTDI Adapter wie in Abbildung.12 angeschlossen kann der ESP8266 ganz leicht in den Programmiermodus versetzen werden indem der „Prog – Button“ gedrückt wird, diesen gedrückt hält, den „Reset-Button“ kurz betätigt und anschließend den „Prog-Button“ loslässt. Die blaue LED am ESP8266 sollte kurz aufleuchten. Der Widerstand am Reset-Button ist ein 10kOhm Widerstand.

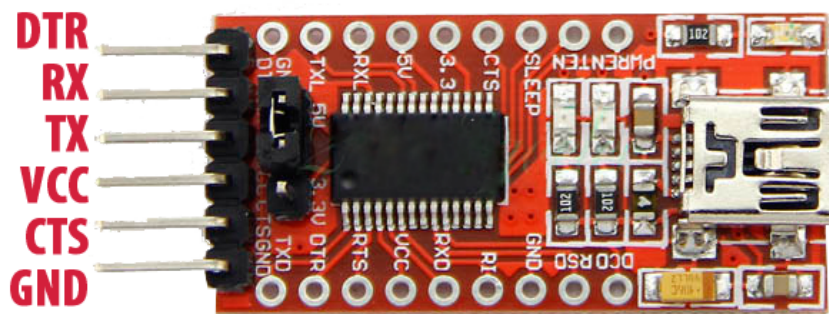


Abbildung 11: FTDI Adapter

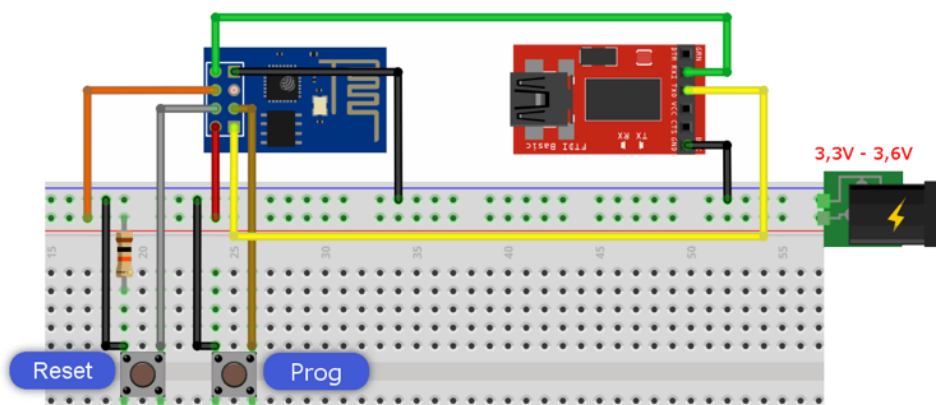


Abbildung 12: Verkabelung FTDI mit ESP8266

3. Eigene Arbeit/Umsetzung

3.1 MQTT Broker für Raspberry Pi einrichten

Es gibt verschiedene MQTT-Implementationen. Sehr einfach lässt sich auf dem Raspberry Pi unter Raspbian Mosquitto nutzen. Installiert wird es folgendermaßen:

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

Nun sollte der Mosquitto-Server bereits laufen. Das kann wie folgt einfach getestet werden:

```
mosquitto_sub -h localhost -v -t test
```

Mit diesem Befehl wird ein MQTT Subscriber gestartet, der auf Nachrichten im "Test"-Kanal wartet.

3.2 ESP8266 mit dem FTDI Adapter verbinden

Um den ESP8266 programmieren zu können muss dieser zunächst mit dem FTDI Adapter verkabelt werden (z.b. wie in Abbildung 12) und an einen PC / Laptop angeschlossen werden auf welchem die Arduino IDE läuft.

Zum Programmieren haben wir den ESP-Modul wie folgt verbunden:

- SC_PD/CHIP_EN auf VCC (um dem Modul zu sagen, dass es laufen soll)
- Tx und Rx über Kreuz an ein USB-Serial Modul (FTDI)
- VCC an 3,3V
- GND natürlich mit der Masse verbunden
- GPIO0 muss zum Programmieren auf Masse gezogen werden
- RST kann offen bleiben

Bei dem Tx Pegel sollte man auch drauf achten, dass dieser nicht auf 5V läuft. Einige Adapter haben zwar einen Jumper, den man von 5V auf 3V3 umlöten kann, teilweise wird damit allerdings nur VCC und nicht der Datenpegel geändert. Allerdings wäre dies weitaus harmloser, als den ESP mit 5V an VCC zu versorgen. Die einfachste Abhilfe schafft ein Spannungsteiler mit zwei Widerständen, deren Werte etwa im Verhältnis 1:2 sind.

Sollte alles einwandfrei angeschlossen sein sollte der ESP8266 leuchten und auf AT Befehle reagieren.

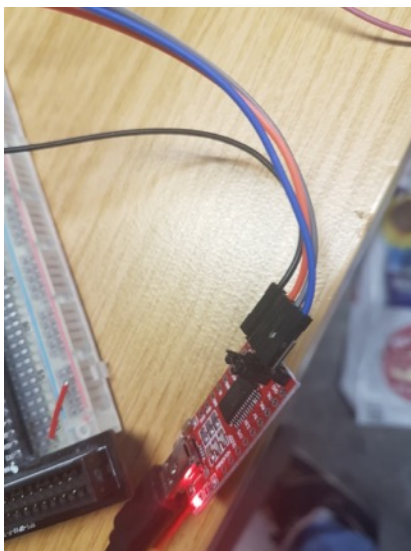


Abbildung 13: verkabelter FTDI Adapter

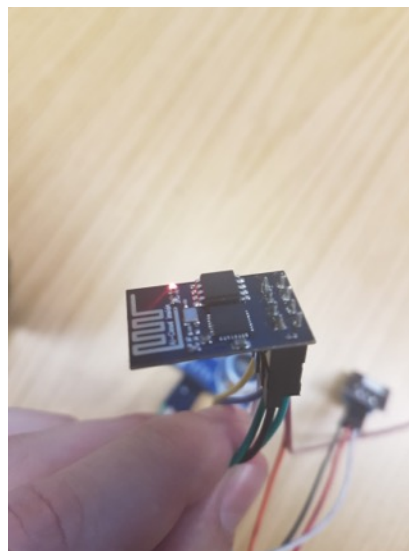


Abbildung 14: funktionsbereiter ESP8266

3.3 ESP8266 mit der Arduino IDE programmieren

Zunächst einmal muss die Arduino IDE installiert werden wenn diese nicht schon installiert ist. Die Arduino IDE unterstützt nach der Installation eine ganze Reihe von verschiedenen Arduino-Boards. Da wir aber kein Arduino Board sondern das ESP8266 programmieren möchten, müssen wir die Boardverwaltung der IDE um ein Paket bereichern. Das geht sehr einfach, wir rufen dazu im **Menü Datei** den **Menüpunkt Voreinstellungen** auf.

Dort gibt es das Eingabefeld **Zusätzliche Boardverwalter URL's**. Dort haben wir die unten aufgeführte Url eingegeben und schließt danach den Dialog wieder mit OK.

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Das sollte dann folgendermaßen aussehen:

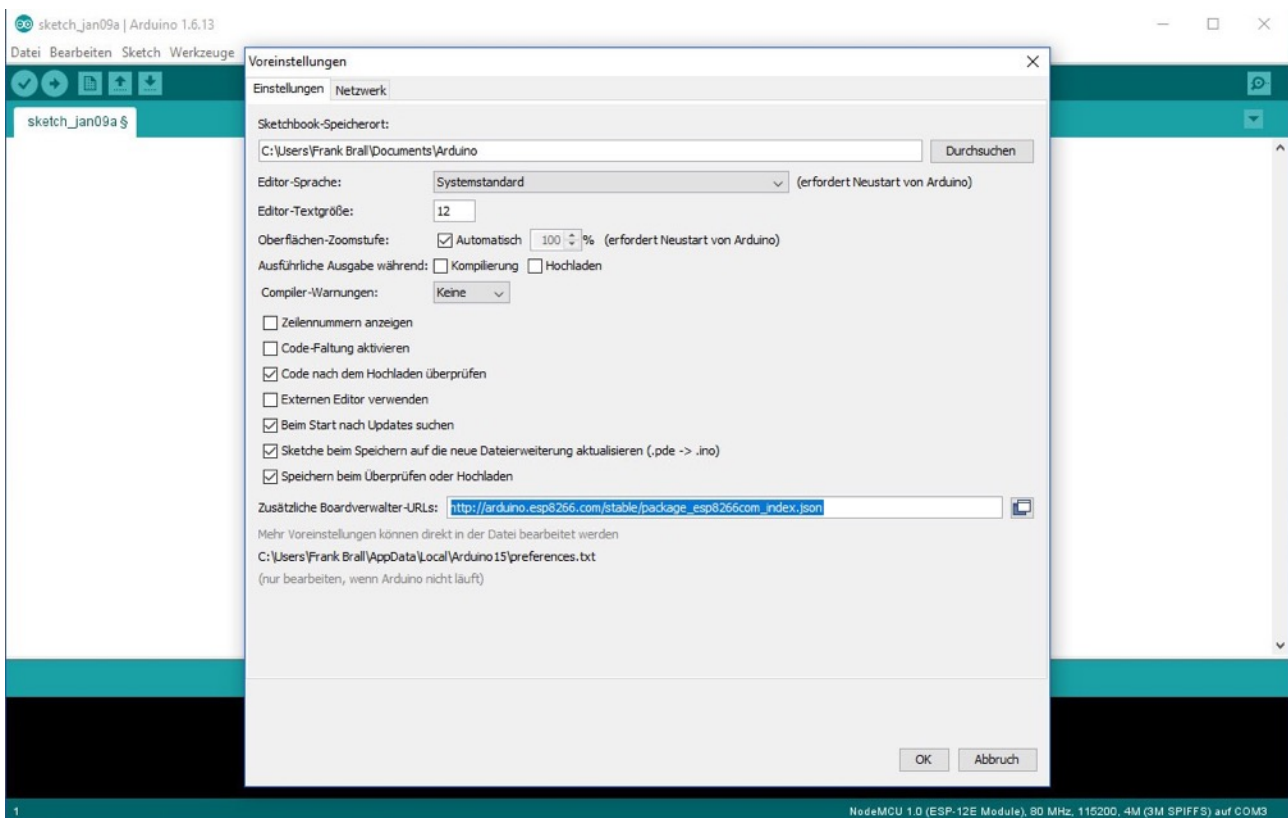


Abbildung 15: ESP8266-Module in die Boardverwaltung aufnehmen

Nachdem Ihr die Voreinstellungen wieder mit OK verlassen habt, könnt Ihr im Menü **Werkzeuge** den Menüpunkt **Board** und danach **Boardverwaltung** auswählen.

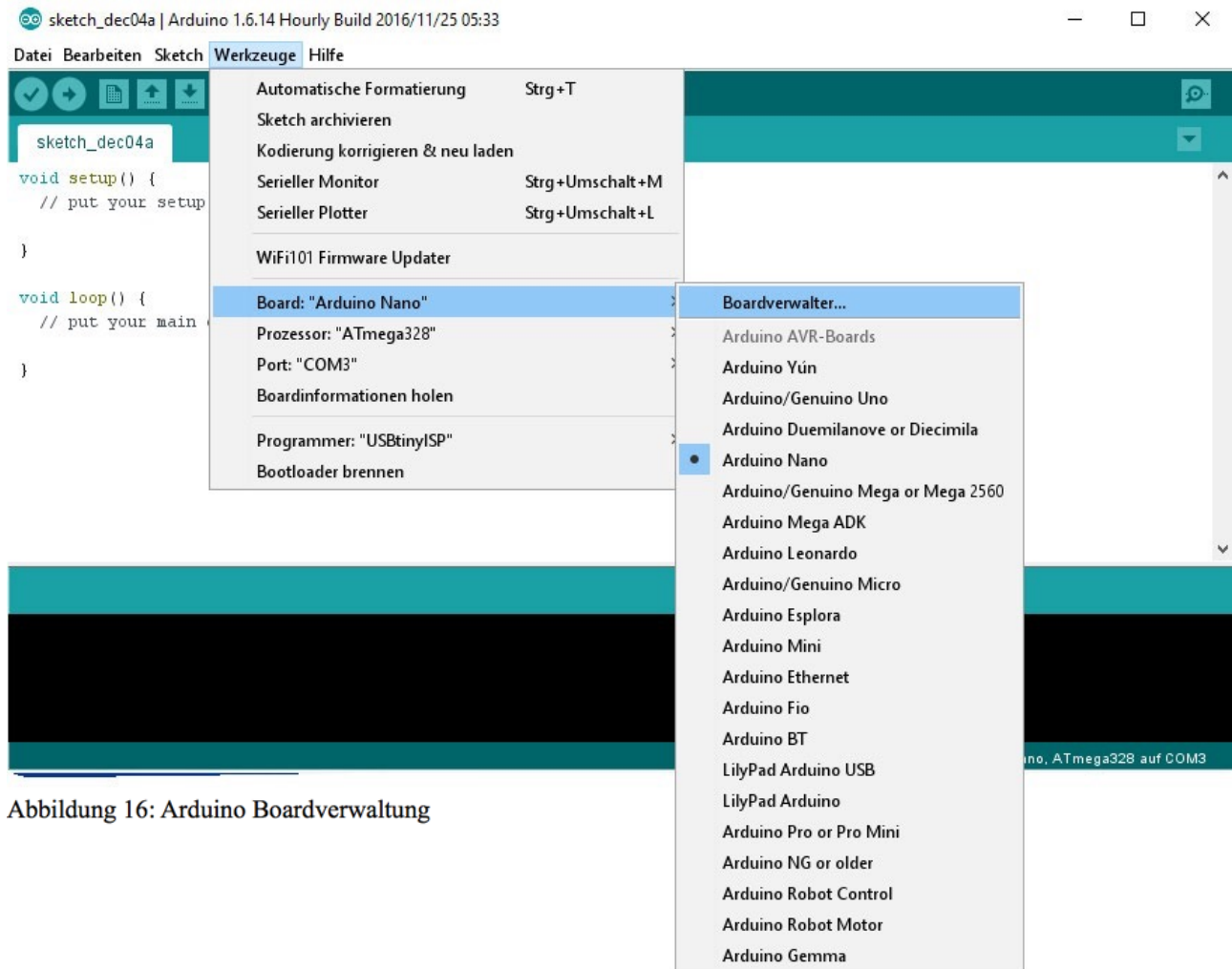


Abbildung 16: Arduino Boardverwaltung



Abbildung 17: Arduino Boardverwalter

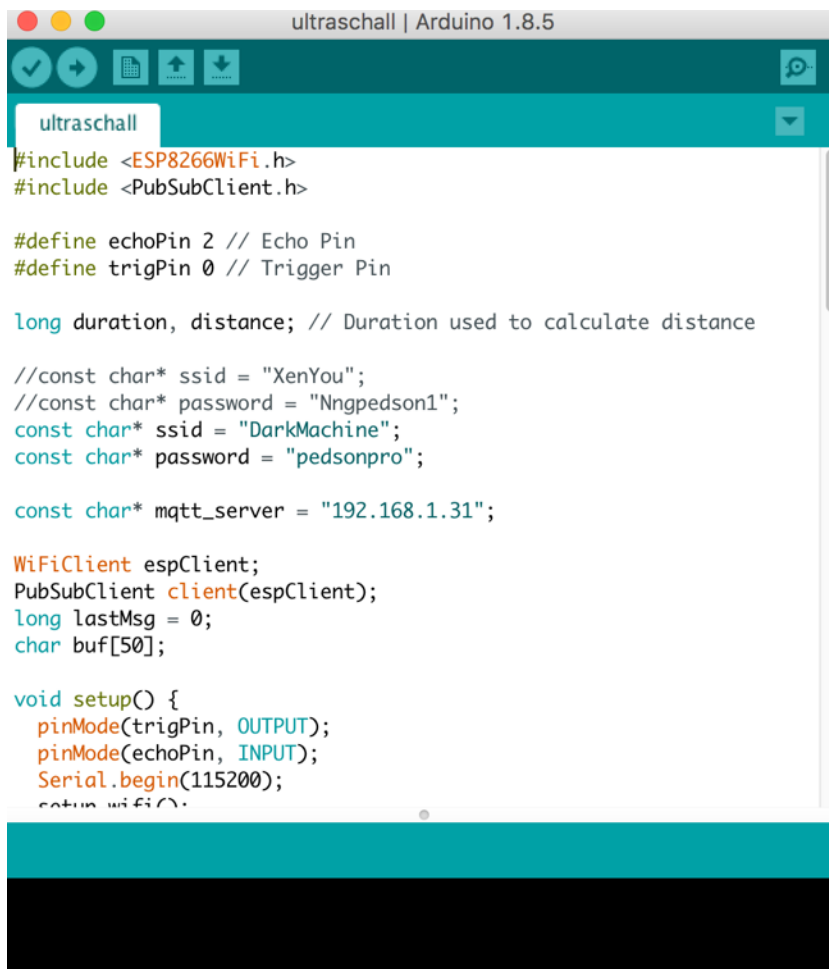
Die Installation des ESP-Paketes besteht aus sehr vielen Dateien, daher kann der Installationsvorgang je nach Rechner durchaus 1 bis 2 Minuten dauern. Danach haben wir die Boardverwaltung wieder geschlossen, die Arduino IDE ist nun bereit um zahlreiche ESP8266-Module, aber auch um das NodeMCU-Board zu programmieren.

Nun müssen wir der Arduino-IDE noch mitteilen welches Board wir genau programmieren möchten. Dazu gehen wir noch mal ins Menü **Werkzeuge** und **Board** und wählen das entsprechende Board aus.

Als nächstes müssen wir der IDE noch mitteilen an welchem COM-Port das Modul angeschlossen ist, denn der USB-Treiber gibt die Daten über einen virtuellen COM-Port an die IDE weiter. Man könnte jetzt im Gerätemanager von Windows nachschauen welche COM-Ports dort bereitgestellt werden. Gewöhnlich reicht es aber wenn Ihr in der Arduino IDE über **Werkzeuge** auf den Menüpunkt **Port** geht. Gewöhnlich wird dort angezeigt welcher COM-Port jetzt bereit steht. Wenn es nur einer ist, dann ist die Auswahl ja sehr einfach.

Bei den meisten NodeMCU-Boards können Programme mit 115200 Baud übertragen werden, diese Übertragungsgeschwindigkeit ist gewöhnlich bereits in der IDE vorgegeben. Es soll aber auch NodeMCU Boards geben, die auf eine andere Geschwindigkeit eingestellt sind, sollte es also später Probleme mit dem Upload geben, versucht man eine andere Geschwindigkeit unter Menüpunkt **Werkzeuge/Upload Speed**.

Als nächsten Schritt haben wir unsere Programme, die auf dem ESP8266 Module laufen sollen, geschrieben und auf den ESP8266 übertragen.



```
ultraschall
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define echoPin 2 // Echo Pin
#define trigPin 0 // Trigger Pin

long duration, distance; // Duration used to calculate distance

//const char* ssid = "XenYou";
//const char* password = "Nngpedson1";
const char* ssid = "DarkMachine";
const char* password = "pedsonpro";

const char* mqtt_server = "192.168.1.31";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char buf[50];

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(115200);
  setup_wifi();
```

Abbildung 18: HC-SR04 Code

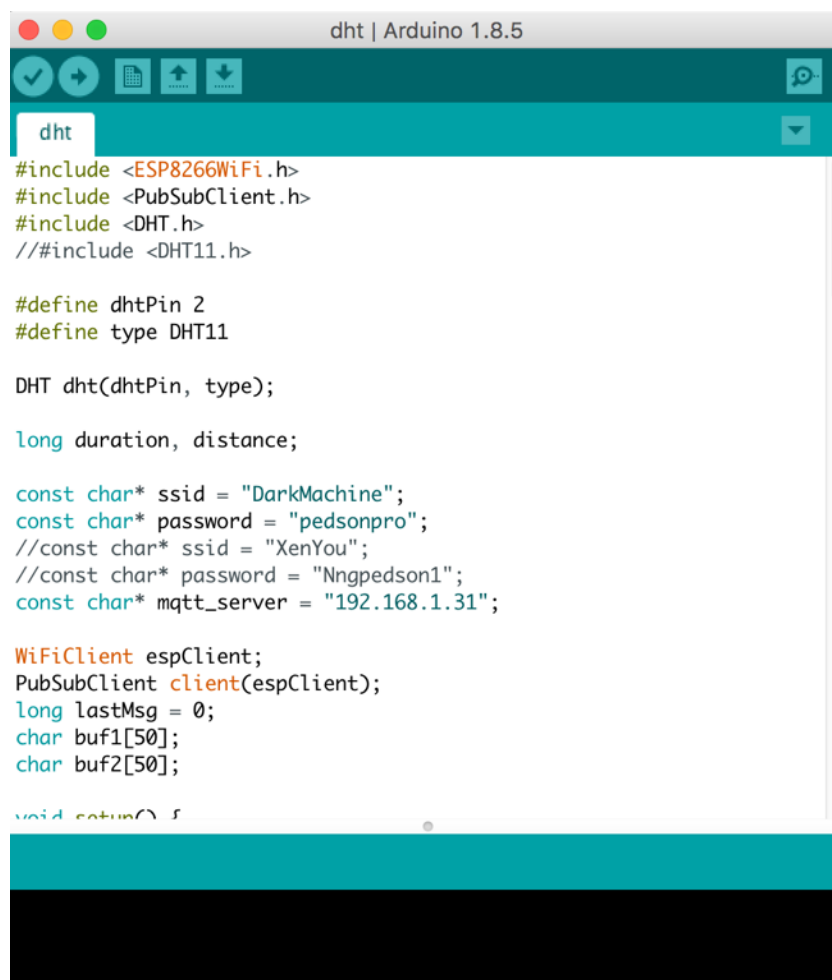
Die Compilierung und Übertragung eines Programmes dauert bei den ESP-Modulen als auch bei dem NodeMCU-Board wesentlich länger als bei den kleinen Arduino-Programmen, da im Hintergrund wesentlich mehr Code compiliert werden muss. Die Compilierung kann je nach Rechner schon mal 30 bis 60 Sekunden dauern.

In welchem Modus der ESP8266 Arbeitet wird durch unseren Code bestimmt der auf den ESP8266 aufgespielt worden ist. In unserem Fall arbeitet der ESP als Client im MQTT Protokoll.

Er verbindet sich mit dem Server „DarkMachine“ auf der IP Adresse 192.168.1.31.

Das ist unserer MQTT Broker (RaspberryPi 3) der die Daten erhält und auswertet.

Die Verbindung wird über Sockets realisiert und läuft über Port 1883.



```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
// #include <DHT11.h>

#define dhtPin 2
#define type DHT11

DHT dht(dhtPin, type);

long duration, distance;

const char* ssid = "DarkMachine";
const char* password = "pedsonpro";
//const char* ssid = "XenYou";
//const char* password = "Nngpedson1";
const char* mqtt_server = "192.168.1.31";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char buf1[50];
char buf2[50];

void setup() {
```

Abbildung 19: DHT-11 Code

3.4 HC-SR04 und DHT-11 mit dem ESP8266 verbinden

Im nächsten Schritt verbinden wir die Sensoren mit den ESP Modulen. Als Stromquelle verwenden wir die FTDI Adapter. Die folgenden beiden Abbildungen veranschaulichen die Verkabelung zwischen dem ESP8266 und dem jeweiligen Sensor.

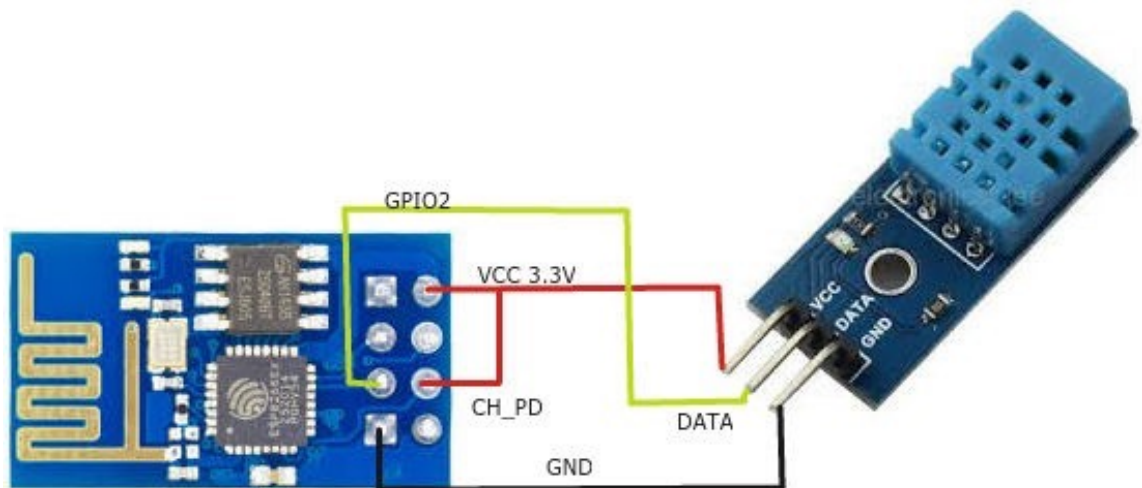


Abbildung 20: ESP8266-01 Verkabelung mit DHT-11

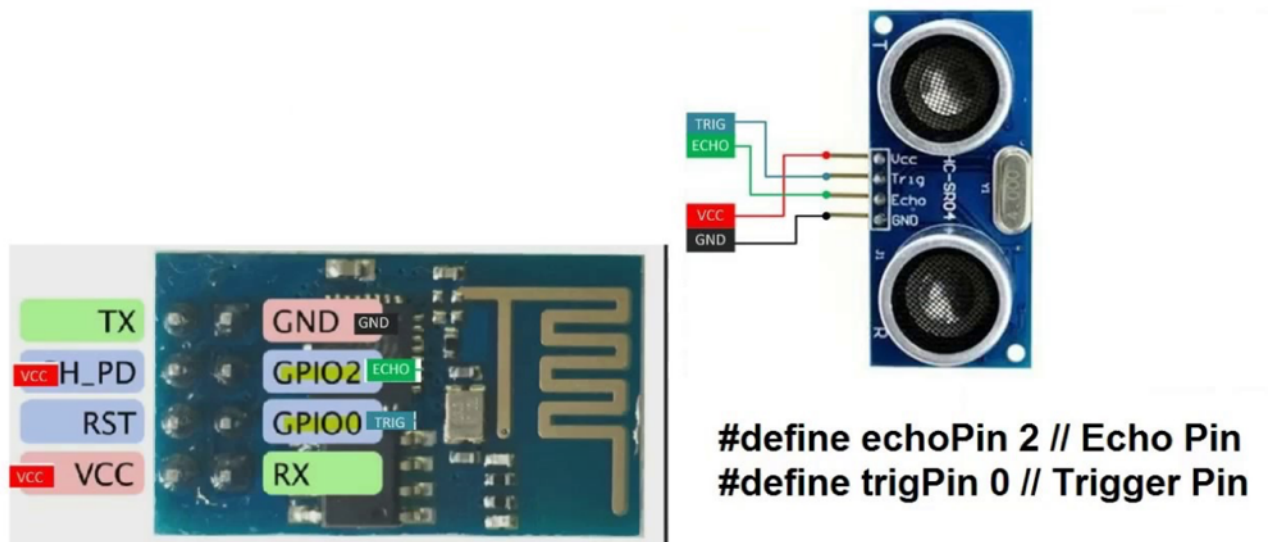


Abbildung 21: ESP8266-01 Verkabelung mit HC-SR04

3.5 Visualisierung mit Node.js einrichten und Daten auswerten

Ab hier läuft der Code der Arduino IDE, und wir können auch in der Konsole beobachten, wie sich die ESP-Module alle zwei Sekunden neue Werte auslesen und diese an den MQTT-Broker senden. Wir wollen nun diese Werte auch in einem Web Browser angezeigt bekommen. Das heißt, wir haben einen Webserver eingerichtet. Dieser meldet sich an den Broker an und abonniert die von den ESP-Module erzeugten Topics. Der Server soll die Werte vom Broker erhalten und diese dann an alle verbundenen Clients verbreiten. Der Vorteil einer Webapplikation mit Node.js ist in unserem Fall, dass Node.js auf JavaScript basiert und somit Ereignis-basierte Entwicklung ermöglicht. Außerdem brauchen wir zusätzliche Module, um das gezielte Ergebnis zu erhalten. Zur Verwaltung der Module gibt es den Node Packet Manager, kurz „**npm**“ den wir benutzen um alle notwendige Module zu installieren.

Bevor wir mit der eigentlichen Arbeit am Projekt beginnen, muss dieser zunächst initialisiert werden. Diese Initialisierung beinhaltet vor allem die Erzeugung der Projektkonfiguration in Form der package.json. Diese Konfigurationsdatei wird auf der Kommandozeile mit dem Kommando **npm init** erzeugt, gefolgt von einer Reihe von Fragen bezüglich des Projekts, die man beantworten muss. Danach können einfach mit dem Kommando **npm install —save „modul“** beliebige Module installiert werden die benötigt werden. In unserem Fall sind es:

- 1) **Express.js** - als Applikation Framework,
- 2) **Mqtt.js** - für die MQTT-Verbindung,
- 3) **Socket.io.js** - für die bidirektionale Kommunikation zwischen dem Server und die Clients,
- 4) **Vis.js** - für die Visualisierung.

Wir haben für die Visualisierung das JavaScript Package Vis.js genutzt, weil es viele Funktionen bietet und auch individuell angepasst werden kann, was für uns sehr gut ist, da wir die Daten live angezeigt bekommen wollen, sobald diese vom Server durch die Websocket-Verbindung gesendet werden. Nun werden die Daten wie in Abbildung 22 angezeigt und können von uns ausgewertet werden.

ESP8266 | Sensordaten mit Mqtt visualisieren

Sensorentyp: DHT11, HCSR04

Mqtt topics: sensor/humidity, sensor/distance, sensor/temperature.

Anzeigetyp:

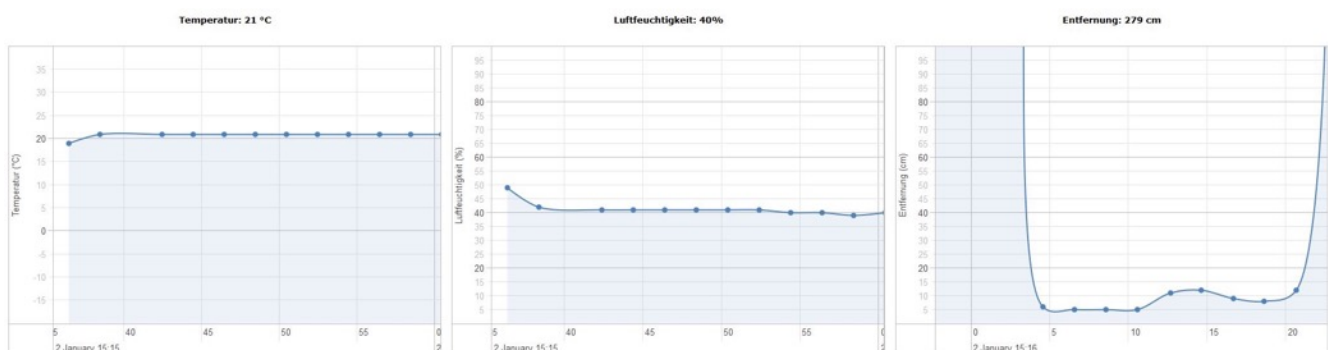


Abbildung 22: Visualisierung der Daten durch Node.js

4. Ergebnis/Ausblick

Dieses Projekt stellt die Funktionsweise des Internet der Dinge dar und ist ein optimales Projekt um erste Erfahrungen mit Sensoren und dem Internet der Dinge zu erhalten.

Jedoch gab es auch einige Schwierigkeiten auf die wir hier näher eingehen möchten.

Nach anfänglichen Problemen mit dem ESP8266-01 würden wir empfehlen in zukünftigen Projekten mit dem Node MCU zu arbeiten. Im Grunde ist ein NodeMCU-Entwicklungsboard nichts anderes als ein ESP8266 der um eine Usb-Schnittstelle (Uart/USB-Wandler auf dem Board) und eine Spannungsstabilisierung (ESP-Module arbeiten mit 3,3V) erweitert wurde. Zudem wurde die Pin Belegung des NodeMCU vereinheitlicht und durchnummeriert. Weiterhin wurde die Programmierung erheblich vereinfacht. Da das programmieren über den FTDI Adapter sich als extrem schwierig und zeitaufwändig herausgestellt hat. Außerdem könnte man mehrere Sensoren an einen größeren ESP anschließen.

Des Weiteren könnte man noch mehrere und alternative Sensoren verwenden um noch mehr Daten zum MQTT Broker senden um diese auswerten zu lassen.

Node.js ist zur Zeit nur für die Ausgabe auf dem Laptop optimiert. Als Ausblick für zukünftige Projekte kann man die Ausgabe für das Smartphone optimieren.

Für ein nächstes Projekt kann auch Node.red verwendet werden.

Unser Projekt stellt eine Verbindung über das lokale WLAN Netzwerk her, um die Daten zu übertragen, da wir Probleme hatten es über einen Hotspot zum laufen zu kriegen.

Desweiteren verwenden wir die FTDI Adapter mit Verbindung eines Laptops als Stromquelle für die ESP8266 und die Sensoren. Als nächstes Schritt könnte man eine externe Stromquelle verwenden.

5. Literatur / Quellen

https://de.wikipedia.org/wiki/Internet_der_Dinge

<https://de.wikipedia.org/wiki/MQTT>

<http://mqtt.org>

<https://de.wikipedia.org/wiki/Aktor>

<http://fluux.de/2016/08/einfuehrung-esp8266-wifi-modul/>

<http://wirtschaftslexikon.gabler.de/Definition/internet-der-dinge.html>

<https://www.open-homeautomation.com/de/2016/06/09/install-an-mqtt-broker-on-your-raspberry-pi/>

<https://pishawk.org/peripherals/8266>

<http://www.mikrocontroller-elektronik.de/nodemcu-esp8266-tutorial-wlan-board-arduino-ide/>

<https://arduino-hannover.de/2015/04/08/arduino-ide-mit-dem-esp8266/>

<https://www.youtube.com/watch?v=NPanke6uSvU>

<http://www.i-programmer.info/programming/hardware/10037-raspberry-pi-wifi-with-the-esp8266-.html?start=2>

<https://pubsubclient.knolleary.net/api.html>

<https://www.baldengineer.com/mqtt-tutorial.html>

[Kofler, Kühnast, Scherbeck : „Raspberry Pi - Das umfassende Handbuch“, Rheinwerk Computing, 2015](#)