

SS2018 Embedded Systems

LoRa Messnetzwerk

Sinan Ayhan, Nick Dirlein, Holger Herbersagen
Marcel Hizli, René Schießwohl

26. August 2018

Einleitung

Wir haben uns für dieses Projekt entschieden, da wir an dem LoRa-Protokoll sehr interessiert sind. Das LoRa-Protokoll ist eine neue innovative Technologie, die sehr energiesparend ist, sodass es perfekt für unsere kommerzielle Idee geeignet ist. Die Arbeit zeigt unser Vorgehen und den Verlauf des Projekts innerhalb des Semesters. Hierbei gehen wir sowohl auf die Konzeptionierung, Umsetzung und auf den Ausblick ein. Diese Dokumentation geht außerdem sehr genau in die technische Umsetzung ein und beschreibt die verwendeten Technologien. Zudem weiten wir dies aus, indem wir erste Gedanken zur Kommerzialisierung beschreiben. In dem folgenden Bild erkennt man unseren ersten fertigen Prototypen und die einzelnen Bauteile, die verwendet wurden. Hierbei handelt es sich um die gelötete Platine, die Wägezelle, das LoRa-Modul mit der Antenne und ein LCD-Display.

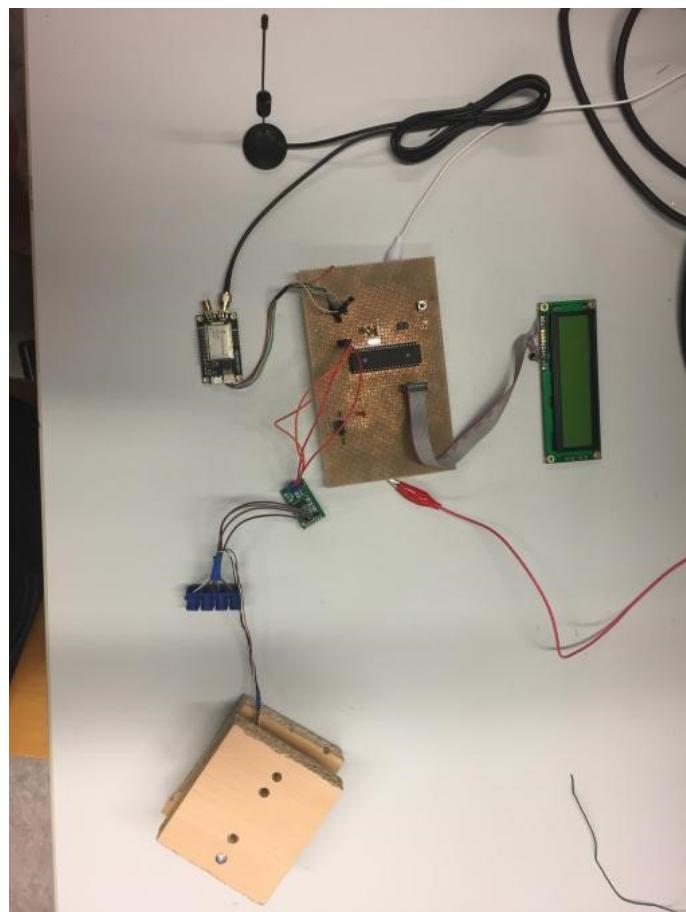


Abbildung 0.1: Überblick des Prototyps

Das Ziel dieser Dokumentation ist, dass klar und verständlich erklärt wird, wie der erste Prototyp funktioniert, welche Stärken dieser besitzt und wie man dies verbessern könnte. Wir bedanken uns für die tatkräftige Unterstützung bei unseren Dozenten Prof. Dr.-Ing. Jürgen Doneit und Ulrich Strauß.

Inhaltsverzeichnis

1 Projektvorstellung	1
2 Ausgangssituation / Beweggründe für die Umsetzung	2
3 Einkaufsliste	4
4 Komponentenmodell	5
5 Überblick der Bauteile	6
5.1 Gateway	6
5.2 Wägezelle	7
5.3 Gewichtssensor	8
5.4 LoRa Node	9
6 Gewichtsmessung mit einer Wägezelle	10
6.1 Funktionsweise der Wägezelle	10
6.2 Versuchsaufbau zur Gewichtsmessung	14
6.3 Programmlogik unseres Microchips	16
6.3.1 Vereinfachte Funktionsweise für momentane Gewichtsmessung .	16
6.3.2 Algorithmus für Rausch- und Driftunterdrückung	16
6.3.3 Algorithmus für Gewichtssendung mit dem LoRa-Modul	17
6.4 Demonstration der Gewichtsmessung	17
6.5 Use-Cases von Straight bar Wägezellen	17
6.6 Alternative Bauformen von Wägezellen	18
6.6.1 Disk-Wägezellen	18
6.6.2 Wägesensoren	19
6.6.3 S-Typ-Wägezellen	19
6.6.4 Kompressions-Wägezellen	20
6.7 Alternative Gewichtssensoren	21
6.7.1 Force Sensitive Resistor (FSR)	21
6.7.2 Piezoelektrischer Sensor	21
7 Drahtlose Verbindung zwischen den Komponenten mit dem LoRaWAN Protokoll	22
7.1 Aufsetzen eines Servers (LoRa-Gateway)	22
7.2 Erstellen einer LoRa-Applikation zum Abrufen der Daten	24
7.3 Aufsetzen eines Clients (LoRa-Node)	27
7.4 Spreading Factor	29
7.5 Security / Datensicherheit	30

7.6 RN2483- und UART-Methoden und ihre Funktionalität	31
7.6.1 RN2483.c	31
7.6.2 uart.c	32
8 Application Server	33
8.1 Visualisierung der empfangenen Daten mit Angular CLI	33
8.2 NodeJS Server für die Verarbeitung der Datenpakete	34
8.3 The Things Network	34
8.4 Inaktivität der Behälter	36
8.5 Voraussetzungen und Nutzung des Node-Servers	36
9 Schaltplan und Verlöten der Platine	37
10 Ausblick	40
10.1 LoRaWAN Modul Code verbessern	40
10.2 Front-End Lizenzkosten vermeiden	40
10.3 Erstellen eines wasserfesten Gehäuses	41
10.4 Kommerzialisierung und mögliches Geschäftsmodell	41

Abbildungsverzeichnis

0.1 Überblick des Prototyps	
1.1 Use-Case mit den Akteuren Kunde und Kellner und den Komponenten Kaffeekanne und Web-Server	1
2.1 Projektplan	3
4.1 Komponentenmodell, erstellt mit https://www.draw.io/	5
5.1 iC880A-SPI Concentrator Board (links) und Raspberry Pi Model 2B (rechts) bilden zusammen das Gateway	6
5.2 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt	7
5.3 HX711 24-Bit ADC für Gewichtssensoren	8
5.4 Data Link LoRa RN2483 als LoRa-Node	9
6.1 Aufbaubeschreibung einer Wägezelle	10
6.2 Funktionsweise einer Wägezelle bei Gewichtsmessung	11
6.3 Verschaltung der Dehnmessstreifen	12
6.4 Verschaltung der Dehnmessstreifen mit Spannungsmessung	12
6.5 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt	14
6.6 Verdrahtung der Wägezelle	15
6.7 HX711 ADC-Wandler, Ansicht auf Bauteil	16
6.8 Disk-Wägezelle	18
6.9 Wägesensor	19
6.10 S-Typ-Wägezelle	19
6.11 Kompressions-Wägezelle	20
6.12 Force Sensitive Resistor	21
6.13 Piezoelektrischer Sensor	21
7.1 Übersicht eines Beispiel LoRa-Netzwerks	22
7.2 Übersicht der registrierten Gateways im TheThingsNetwork	23
7.3 Übersicht der erstellten Applikationen im TheThingsNetwork	24
7.4 Formular zur Erstellung einer Applikation	24
7.5 Übersicht der registrierten Geräte	25
7.6 Formular zur Registrierung eines Geräts	25
7.7 Aktivierungsmethoden: links Over-the-Air Activation (OTAA), rechts Ac- tivation By Personalization (ABP)	26
7.8 Beispielcode	26

7.9 Spezifikationen für das UART-Interface des RN2483	27
7.10 RN2483 Boot Time unterteilt in die einzelnen Funktionen und der benötigten Zeit in ms	28
7.11 Tabelle mit verschiedenen Spreading Factors und die zugehörige Bitrate	29
7.12 Verschlüsselung und Datensicherheit bei der Kommunikation zwischen verschiedenen LoRa-Geräten	31
8.1 Übersicht der Füllstände der einzelnen Kaffeemaschinen auf der Webseite	33
8.2 Füllstand-Verlauf einer Kaffeemaschine von 12:00 Uhr bis 15:00 Uhr . .	34
9.1 Schaltplan, erstellt mit Fritzing	37
9.2 Rückseite der verlötzten Platine unseres ersten Prototyps	38
9.3 Vorderseite der verlötzten Platine unseres ersten Prototyps	39

Tabellenverzeichnis

3.1 Einkaufsliste	4
-----------------------------	---

1 Projektvorstellung

Wir haben uns in der Veranstaltung Embedded Systems des Studiengangs Software-Engineering bei Prof. Dr. Jürgen Doneit und Herr Ulrich Straus im Sommersemester 2018 für das Projekt LoRa-Messnetzwerk entschieden. Die Aufgabe bestand darin, ein Messnetzwerk aufzubauen, welches Sensordaten entgegennimmt und über Long Range Wide Area Network, kurz LoRaWAN, verschickt werden.

Der daraus resultierende Use-Case:

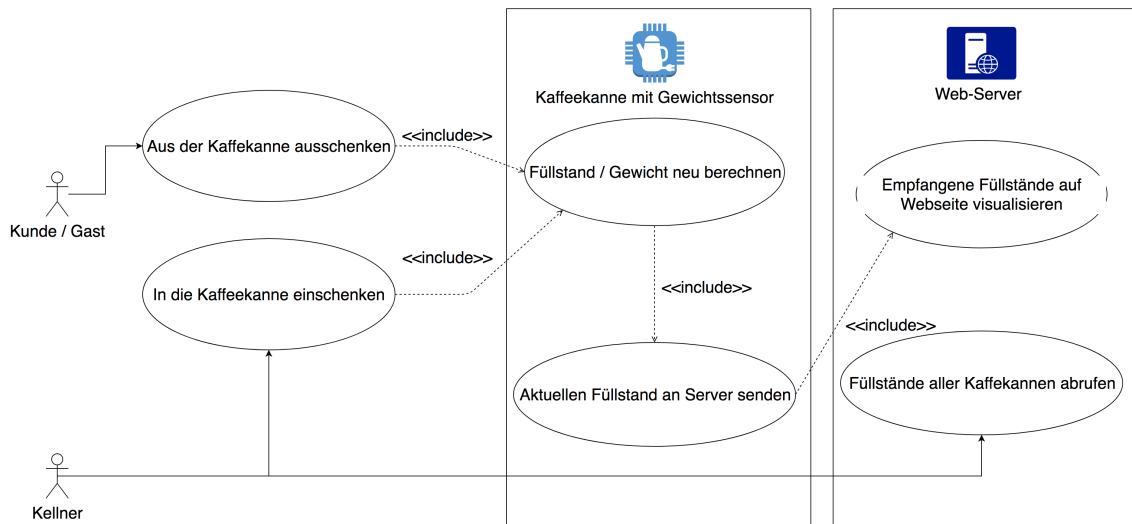


Abbildung 1.1: Use-Case mit den Akteuren Kunde und Kellner und den Komponenten Kaffeekanne und Web-Server

In einem Restaurant wird der Inhalt einer Kaffeekanne durch das Gewicht bestimmt. Um die Gewichtsmessung für alle Kannen durchzuführen, werden sie auf einer Plattform abgestellt, in der ein Wäge-Sensor-System integriert ist. Das Sensor-System, welches über einen Akku betrieben wird, besteht aus einem Wäge-Sensor, einem Mikrocontroller und einem Transmitter (dem LoRa-Node). Die Messdaten werden jeweils ausgelesen und an den angeschlossenen LoRa-Node übertragen. Die Gesamtheit aller LoRa-Nodes kommunizieren mit einem Empfänger, dem LoRa-Gateway. Das LoRa-Gateway, angeschlossen ans Internet, empfängt die Sensorgewichtsdaten und leitet diese auf einen Web-Server weiter. Das Server-System verarbeitet die Daten, um die Füllstände der Kaffeekannen schlussendlich auf einer Webseite zu visualisieren. Der plattformübergreifende Zugriff über diverse Endgeräte ist somit möglich.

2 Ausgangssituation / Beweggründe für die Umsetzung

In der Gastronomie üblich, prüft das Personal von Zeit zu Zeit ob in den Behältnissen der Gäste auf den Tischen oder am Buffet noch genügend Inhalt vorhanden ist. Das reicht von der Kaffeekanne auf dem Frühstückstisch, über das Butterbehältnis auf dem Buffet, bis hin zur Weinflasche am Abend im Restaurant. Natürlich gehört dies zum Service eines guten Gastrounternehmens. Auf der anderen Seite kostet es viel Servicekraft die Füllstände zur vollen Zufriedenheit des Kunden und jederzeit im Auge zu behalten.

Unser Ziel ist es bei diesem Prozess eine höhere Effizienz zu erzielen. Es soll zu jeder Zeit und von überall, ohne Sichtprüfung eine klare Aussage über einen nötigen Nachfüllprozess getroffen werden können. Durch das Wegfallen der ständigen Sichtprüfung vor Ort, hat der Kunde eine ruhigere, entspanntere und damit angenehmere Atmosphäre. Er wird nur dann auf den Wunsch des Nachfüllens angesprochen, wenn das Gefäß tatsächlich leer ist. Damit werden deutlich optimalere Geschäftsabläufe gewährleistet:

- Produkte können bedarfsgerechter angerichtet, zubereitet oder hergestellt werden
- Wartezeiten gegenüber dem Kunden lassen sich wahrnehmbar verkürzen, die Zufriedenheit steigt
- und das Personal kann sich auf andere/wichtigere Themen bezüglich des Kunden konzentrieren

Um die Kernfunktionalität im Einsatz zu testen und die Idee zu validieren, beschränken wir uns bei der Entwicklung unseres Prototyps auf die Messung der Füllstände von Kaffeekannen.

Damit das System funktioniert, werden die Kaffeekannen auf einer Plattform abgestellt und durch Wäge-Sensoren gewogen. Die gemessenen Werte und die daraus entstandenen Informationen werden visualisiert und dem Personal über eine Webseite angezeigt. Bei niedrigem Füllstand (oder zu niedriger Temperatur, weil schon zu lange in der Kanne) können sie damit schnell reagieren. Die komplette drahtlose Kommunikation basiert dabei auf LoRa-Nodes, die sehr energiesparend sind. Bei konstanter täglicher Nutzung des Systems ist bei entsprechender Akkugröße eine Laufzeit von einem Jahr möglich. Trotzdem ist zusätzliches Optimierungspotential vorhanden. Denn werden die LoRa-Nodes nach dem Senden der Zeit- und Gewichtsinformationen

in einen Sleep-Mode versetzt, zusätzlich der Sende-Zyklus verlängert und serverseitig ein besserer Abstraktionsalgorithmus implementiert, sind bei gleichem Akku Laufzeiten bis zu drei Jahren denkbar.

Am Anfang des Projektes erstellten wir Bauteildiagramme und diskutierten über Kommunikationswege zwischen den Bauteilen. Danach teilten wir Schwerpunkte beziehungsweise Bauteile auf unsere Teammitglieder auf, welche dann nochmal tiefer vom zugewiesenen Teammitglied recherchiert wurden. Die Zusammensetzung der Bauteile erfolgte erstmal auf einer Entwicklungsplattform um schnelle Tests und Hardwareänderungen durchführen zu können. Nach erfolgreichen Tests auf unserer Entwicklungsplattform wechselten wird die Hardware auf eine selbst konzeptionierten Platine. Nach den letzten Tests dokumentierten wir unsere Ergebnisse.

Gefährdet	Aufgabenname	Start Datum	Ende Datum	Zugewiesen	% vollständig
Projektstart		26.03.18	03.04.18		100%
Kick-Off	26.03.18	26.03.18	Alle	100%	
Konzeptionierung	26.03.18	26.03.18	Alle	100%	
Arbeitsteilung	26.03.18	26.03.18	Alle	100%	
Recherche	27.03.18	03.04.18	Rene	100%	
Kraftsensor und Verteilung des gewichts	27.03.18	03.04.18	Holger	100%	
LORA Protokoll	27.03.18	03.04.18	Nick	100%	
LORA Server	27.03.18	03.04.18	Sinan	100%	
Stromversorgung	27.03.18	03.04.18	Marcel	100%	
Grobe Realisierung		03.04.18	17.04.18		100%
Bestellung	03.04.18	03.04.18	Alle	100%	
Steckboard bestücken	03.04.18	17.04.18	Nick/Holger	100%	
LORA Server	03.04.18	17.04.18	Sinan	100%	
Prototypische Implementierung		16.04.18	25.06.18		100%
Bestellte Teile einbauen	16.04.18	03.05.18	Alle	100%	
Übertrag auf Steckplatine	07.05.18	15.06.18	René/Marcel	100%	
Schaltplan erstellen	07.05.18	25.06.18	René/Marcel	100%	
Produkt auf Platine umziehen	21.05.18	25.06.18	René/Marcel	100%	
Testen		11.05.18	02.07.18		100%
Hardwaretest	11.05.18	21.05.18	Alle	100%	
Hardwaretest mit Platine	25.06.18	02.07.18	Alle	100%	
Softwaretest	21.05.18	02.07.18	Alle	100%	
Dokumentation		29.06.18	06.07.18		100%
Zusammenführen	29.06.18	06.07.18	Alle	100%	
Formatieren	29.06.18	06.07.18	Alle	100%	
Rechtschreibprüfung	29.06.18	06.07.18	Alle	100%	

Abbildung 2.1: Projektplan

3 Einkaufsliste

Tabelle 3.1: Einkaufsliste

Komponente	Preis
Web-Server für NodeJS VPS oder Cloud Server mieten oder einen vorhandenen, netzwerkfähigen Computer benutzen	ab 5.00 €/ Monat -
LoRa-Gateway iC880A-SPI + Antenne + Pigtail cable + Raspberry Pi 2 B	167.90 €
LoRa-Node Data Link LoRa RN2483 + Antenne	Ca. 70 €
Wägezelle + HX711	10.71 €

Für unseren ersten Prototyp benötigten wir das LoRa-Node Modul mit Antenne, um Daten an das Gateway (Server) verschicken zu können. Außerdem haben wir eine Wägezelle mit einem AD-Wandler (HX711) an einen Mikrocontroller (Atmega 644PA) angeschlossen, um damit das Gewicht von der Wägezelle messen und übertragen zu können.

Durch weitere Optimierungen kann der Atmega 644PA oder HX711 weggelassen werden, was den Preis nochmal verringern würde.

4 Komponentenmodell

Hier sieht man unser aktuelles Komponentenmodell. Dort ist zu erkennen, welche Komponenten und Bauteile von uns verwendet werden. Auch wie diese miteinander kommunizieren kann man aus dem Modell entnehmen.

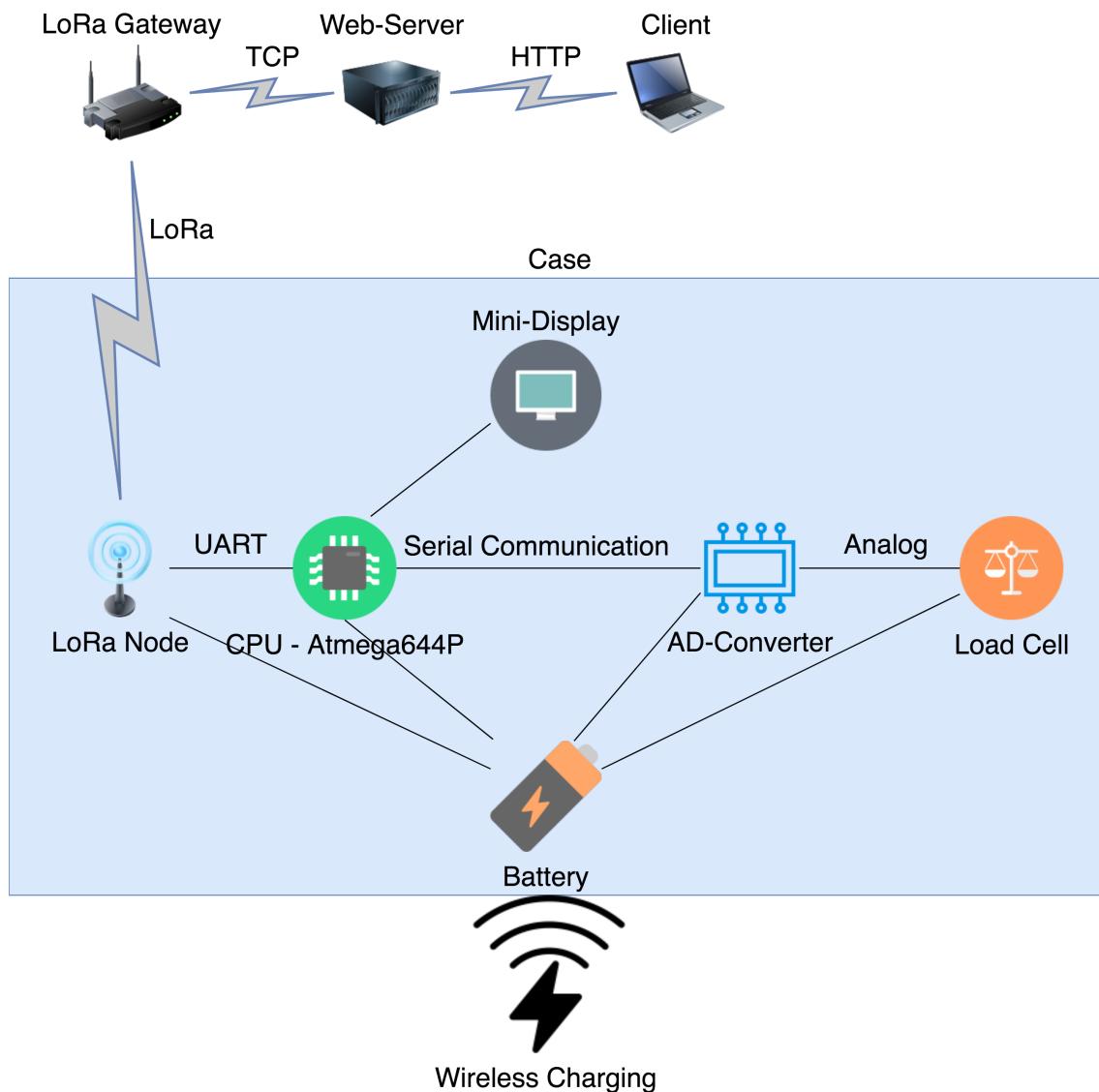


Abbildung 4.1: Komponentenmodell, erstellt mit <https://www.draw.io/>

5 Überblick der Bauteile

5.1 Gateway

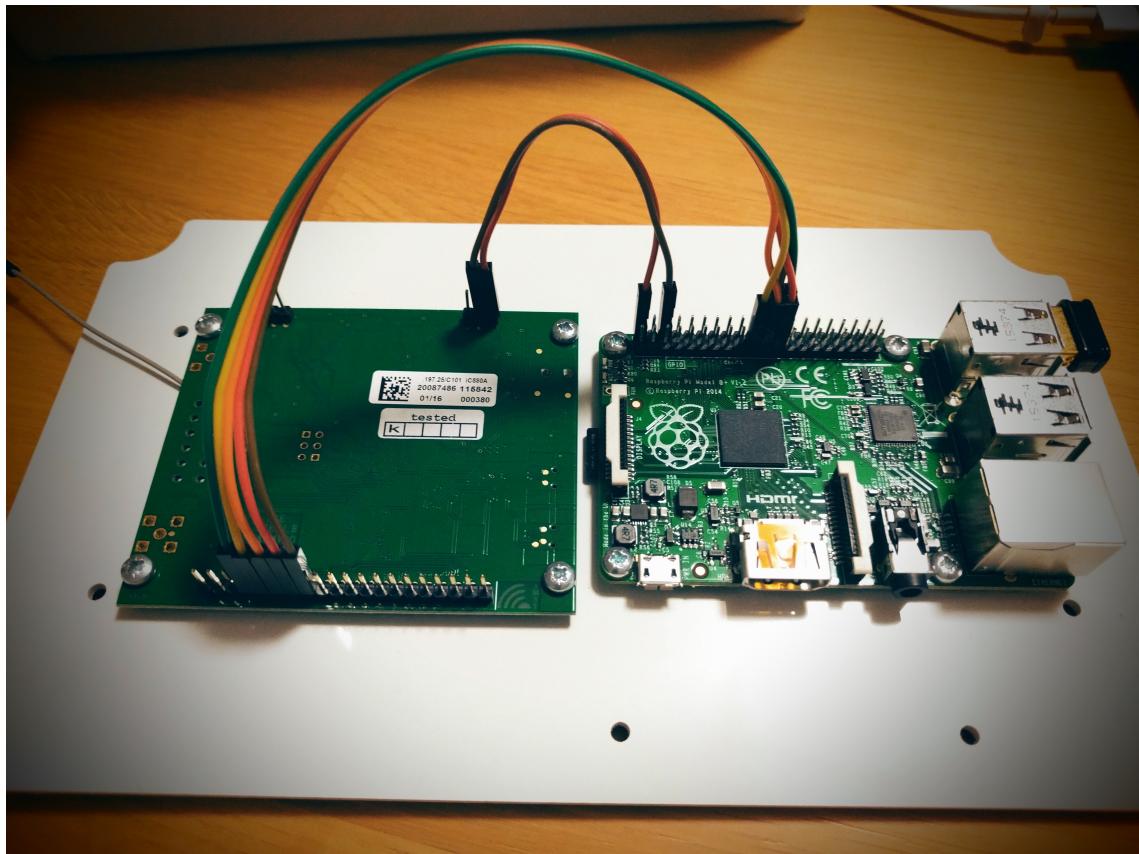


Abbildung 5.1: iC880A-SPI Concentrator Board (links) und Raspberry Pi Model 2B (rechts) bilden zusammen das Gateway

Quelle: <https://raw.githubusercontent.com/ttn-zh/ic880a-gateway/spi/images/mounted-boards.jpg>

5.2 Wägezelle



Abbildung 5.2: 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt

5.3 Gewichtssensor

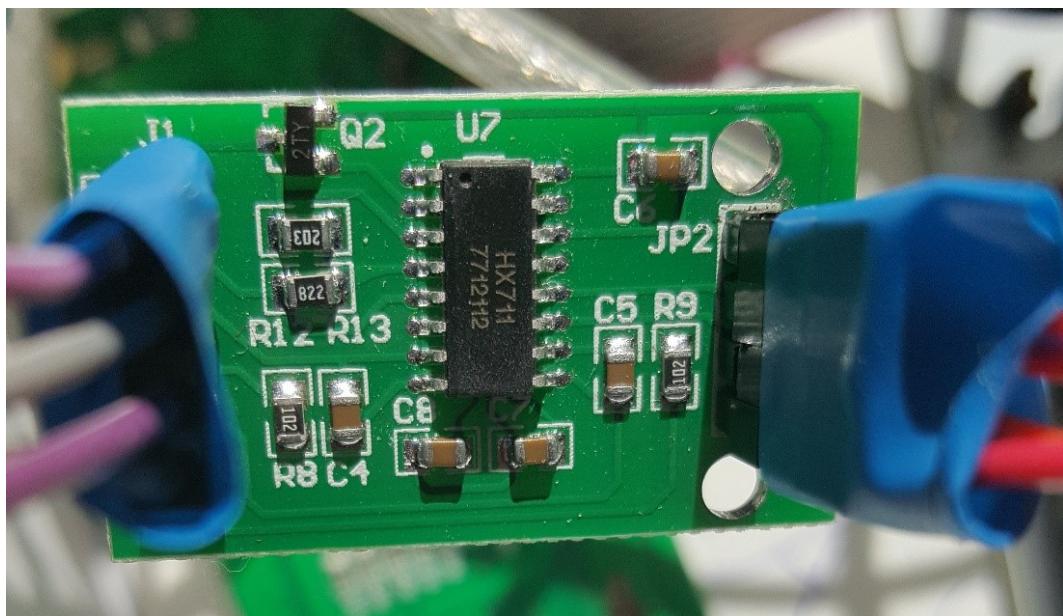


Abbildung 5.3: HX711 24-Bit ADC für Gewichtssensoren

5.4 LoRa Node

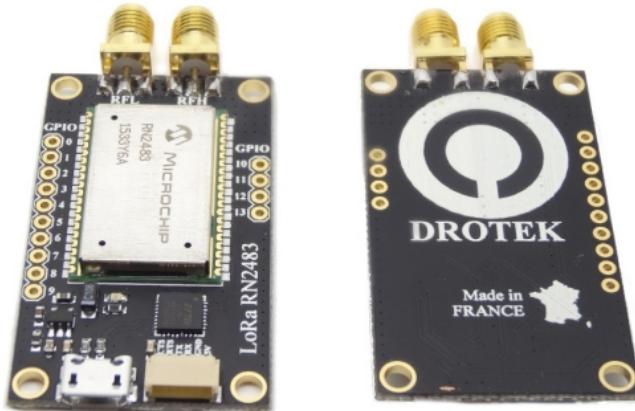


Abbildung 5.4: Data Link LoRa RN2483 als LoRa-Node

Quelle:

https://drotek.com/shop/2643-thickbox_default/data-link-lora-rn2483.jpg

6 Gewichtsmessung mit einer Wägezelle

6.1 Funktionsweise der Wägezelle

Wägezellen sind Sensoren, die das Gewicht über Verformung ihres Materials mit Hilfe von Dehnungsmessstreifen messen können. Es sind vier Dehnungsmessstreifen, wie im Bild unten zu sehen, an der Wägezelle angebracht.

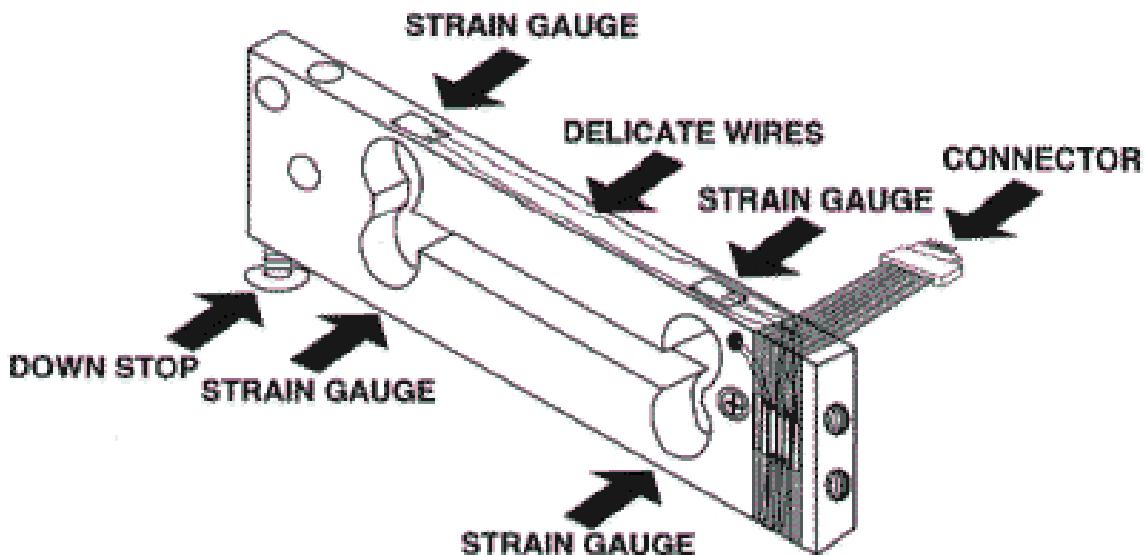


Abbildung 6.1: Aufbaubeschreibung einer Wägezelle

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Bei Auflage von Gewicht bzw. Verformung der Wägezelle messen zwei Dehnungsmessstreifen die Kompression und die anderen zwei die Spannung. Bei Kompression des Dehnungsmessstreifens werden die Leiter dicker (höherer Leiterquerschnitt) und kürzer (geringere Leiterlänge). Laut der Formel für den Leiterwiderstand

$$R = \rho \cdot \frac{I}{A}$$

$$\text{Leiterwiderstand} = \text{spezifischer Widerstand} \cdot \frac{\text{Leiterlänge}}{\text{Leiterquerschnitt}}$$

verringert sich der Widerstand des Dehnungsmessstreifens. Bei Spannung des Dehnungsmessstreifens werden die Leiter dünner und länger, somit erhöht sich auch der Widerstand des Dehnungsmessstreifens.

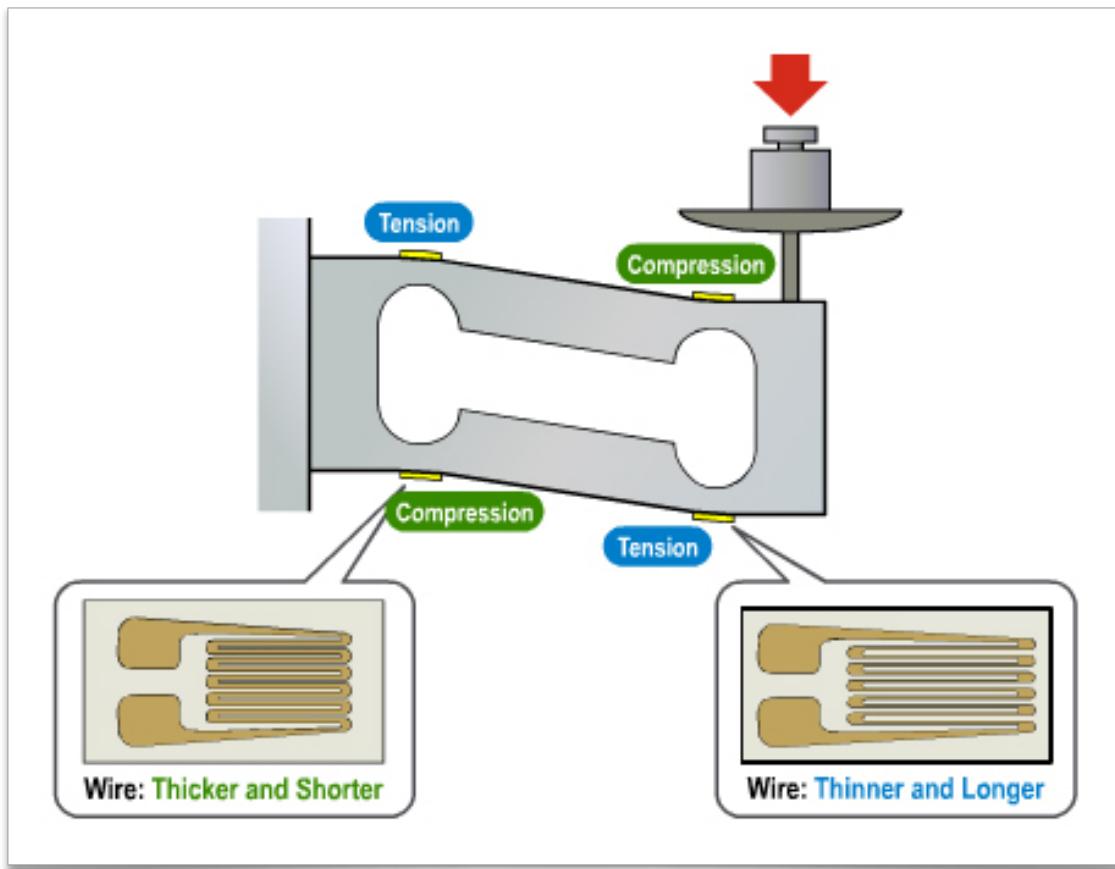


Abbildung 6.2: Funktionsweise einer Wägezelle bei Gewichtsmessung

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Um die Widerstandsveränderungen der Dehnungsmessstreifen messen zu können, sind die Dehnungsmessstreifen innerhalb der Wägezelle in einer Wheatstone-Brücken-Formation geschalten.

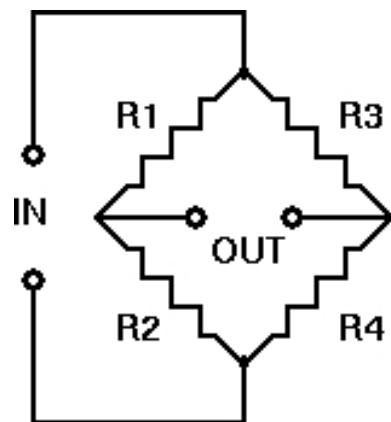


Abbildung 6.3: Verschaltung der Dehnmessstreifen

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Wenn gilt

$$\frac{R1}{R2} = \frac{R3}{R4}$$

dann misst man 0V an Out. Falls sich ein Widerstand verändern sollte, zum Beispiel durch Verformung eines Dehnmessstreifens, lässt sich eine Spannung an Out laut dieser Formel messen:

$$V_{out} = \left(\frac{R3}{(R3 + R4)} - \frac{R2}{(R1 + R2)} \right) \cdot V_{in}$$

Full-bridge strain gauge circuit

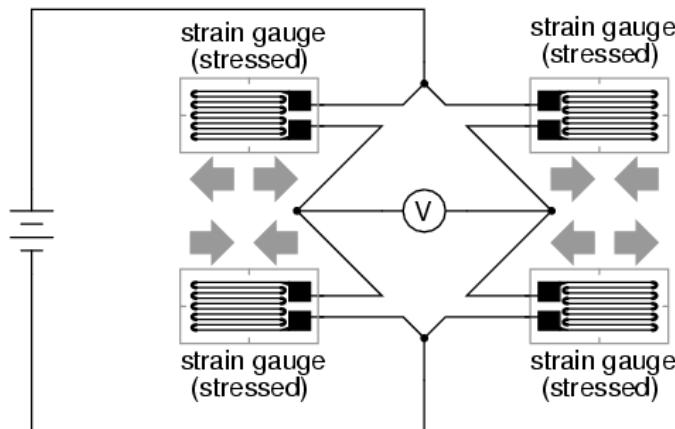


Abbildung 6.4: Verschaltung der Dehnmessstreifen mit Spannungsmessung

Quelle: <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>

Die an V_{out} gemessene Spannung ist sehr klein und analog, deshalb benutzen wir den HX711, welcher das Signal verstärkt und in digitale Werte [0 bis 16777217] umwandelt.

6.2 Versuchsaufbau zur Gewichtsmessung

Die Wägezelle wurde an zwei gegenüberliegenden Punkten an zwei Holzplatten geschraubt, damit sich das Gewicht auf alle Dehnungsmesstreifen auswirkt.



Abbildung 6.5: 20Kg Wägezelle zwischen zwei Spanholzplatten befestigt

Die Wägezelle wurde wie folgt an das HX711-9Modul angeschlossen:

- Rot an E+ (Excitation+/Vin+)
- Schwarz an E- (Excitation-/Vin-)
- Weiß an A+ (Output+/Vout+)
- Grün an A- (Output-/Vout-)

Veranschaulichung der Verdrahtung (Achtung! Gelbe Verbindung ist bei uns weiß):
Die A-Anschlüsse und B-Anschlüsse sind für verschiedene Verstärkungskanäle. Der

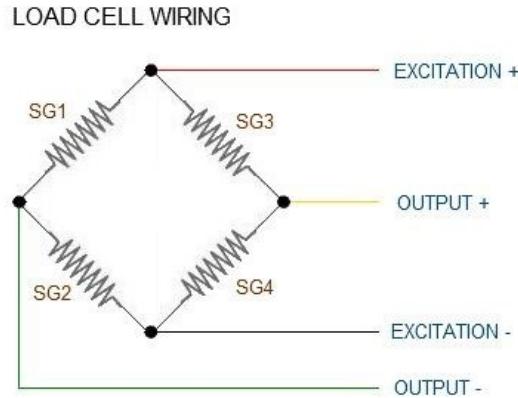


Abbildung 6.6: Verdrahtung der Wägezelle

Quelle: [https://learn.sparkfun.com/tutorials/
load-cell-amplifier-hx711-breakout-hookup-guide](https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide)

A-Kanal hat einen programmierbaren Verstärkungsfaktor von 128 oder 64 und der
B-Kanal hat einen festen Verstärkungsfaktor von 32.

Die rechte Seite des HX711 wird wie folgt angeschlossen:

- GND an Ground
- VCC an Versorgungsspannung in unserem Fall 5V
- SCK an PD5 unseres Microcontrollers/Atmega 644PA
- DT (Data) an PD6 unseres Microcontrollers/Atmega 644PA

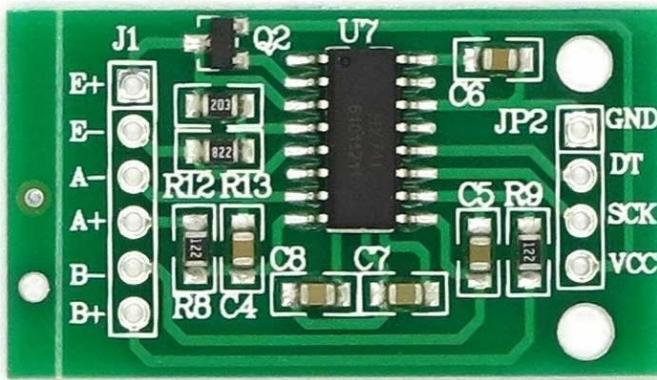


Abbildung 6.7: HX711 ADC-Wandler, Ansicht auf Bauteil
Quelle: <https://tinyurl.com/y8grwwdb>

6.3 Programmlogik unseres Microchips

Das Programm basiert auf getsiddd's HX711 AVR Bibliothek (<https://github.com/getsiddd/HX711>). Die wichtigsten Additionen von uns sind die Möglichkeit, negative Gewichtsveränderungen wahrnehmen zu können, auf Tastendruck eine Tar-Funktion auszuführen und der Gewichts-Sende-Algorithmus in Verbindung mit dem LoRa-Modul.

6.3.1 Vereinfachte Funktionsweise für momentane Gewichtsmessung

Beim vom HX711 gelesenen Gewicht wird das getarte Gewicht (Tar Funktion/Gewichtsnullung) abgezogen und durch die Kalibration dividiert, um einen lesbaren Wert in Kilogramm zu erhalten.

6.3.2 Algorithmus für Rausch- und Driftunterdrückung

Das Differenzgewicht von der letzten und der momentanen Gewichtsmessung wird mit einer vorher gewählten Veränderungsrate verglichen. Wenn das Differenzgewicht höher als die gewählte Veränderungsrate ist, wird der momentan gemessene Wert verwendet und angezeigt, im anderen Fall wird der Wert verworfen, weil er vermutlich

durch Rauschen oder Drift verursacht wurde.

Der Veränderungswert gibt auch die kleinstmögliche messbare Veränderung an. Am Beispiel unseres Projektes könnte bei einem Veränderungswert von ca. 5g, jemand mit einem Strohhalm Flüssigkeit aus dem Behältnis saugen, ohne dass das Programm die Gewichtsveränderung wahrnimmt.

6.3.3 Algorithmus für Gewichtssendung mit dem LoRa-Modul

Mit Hilfe eines Interrupts, wird jede Sekunde eine Gewichtsmessung durchgeführt. Beim Ablauf des Sendeintervalls (in unserem Projekt momentan eingestellt auf fünfzehn Sekunden), wird in der Menge der Gewichtsmessungen (in unserem Fall fünfzehn Messungen, jede Sekunde eine), nach den letzten fünf Gewichtsmessungen gesucht, die den gleichen Wert haben. Wenn die Messungen diese Bedingung nicht erfüllen, wird kein Wert gesendet.

6.4 Demonstration der Gewichtsmessung

Hier eine Demo zur Gewichtsmessung, die wir auch in der Zwischenpräsentation gezeigt haben:

<https://streamable.com/ycxf3>

Alternativ: <https://www.youtube.com/watch?v=2tk0ydRsXgg>

6.5 Use-Cases von Straight bar Wägezellen

Die Art von uns verwendeter Wägezelle wird unter anderem in diesen Bereichen verwendet:

- Küchenwaagen
- Industrielle Gewichtsmessung, die an einem Punkt erfolgt

6.6 Alternative Bauformen von Wägezellen

Wägezellen werden in verschiedenen Bauformen hergestellt, hier werden diese aufgezählt und untersucht um die richtige Wägezelle bei Änderung der Projektanforderungen auswählen zu können

6.6.1 Disk-Wägezellen

Disk-Wägezellen haben eine runde Form und sind kompakter gebaut.



Abbildung 6.8: Disk-Wägezelle

Quelle: http://www.forsentek.com/prodetail_13.html

6.6.2 Wägesensoren

Wägesensoren besitzen nur einen Dehnmessstreifen anstatt herkömmliche, die vier besitzen. Man kann vier Wägesensoren in einer Wheatstone-Bridge-Formation zusammen schalten und wie eine Straight-Bar-Wägezelle betreiben. Dabei kann man die einzelnen Wägesensoren auf einer größeren Fläche betreiben und sie als Personenwaage, Fahrzeugwaage oder allgemein als Waage für große Objekte verwenden.



Abbildung 6.9: Wägesensor

Quelle:

<https://www.botshop.co.za/product/load-cell-sensor-resistance-strain-50kg/>

6.6.3 S-Typ-Wägezellen

Diese Wägezellen lassen sich, dank ihrer S-Form, in Spannungs- und Kompressionsbetrieb betreiben.



Abbildung 6.10: S-Typ-Wägezelle

Quelle: <https://www.coventryscales.co.uk/scale-type/load-cells/tension-s-type-load-cells/s-type-load-cell/>

6.6.4 Kompressions-Wägezellen

Diese Art von Wägezellen sind nur in Kompression belastbar.



Abbildung 6.11: Kompressions-Wägezelle

Quelle: [http://www.eilersen.com/compression-load-cell/product/
atex-compression-load-cell-dla/](http://www.eilersen.com/compression-load-cell/product/atex-compression-load-cell-dla/)

6.7 Alternative Gewichtssensoren

6.7.1 Force Sensitive Resistor (FSR)

Bei Kraftzunahme beziehungsweise Kompression des Sensors verringert sich der Abstand zwischen den Sensorfolien, die voneinander mit einer speziellen Tinte getrennt sind, und somit auch ihr Widerstand. Diese Sensoren sind sehr platzsparend, aber auch ungenauer als Wägezellen, weshalb sie nicht sehr gut für die Gewichtsmessung geeignet sind.



Abbildung 6.12: Force Sensitive Resistor

Quelle: <https://solarbotics.com/product/50803/>

6.7.2 Piezoelektrischer Sensor

Bei Krafteinwirkung produzieren diese Sensoren eine elektrische Ladung, die gemessen werden kann. Sie sind eher für dynamische Anwendungen geeignet, da Signale nur bei Kraftänderungen gemessen werden können, beziehungsweise die Ladung sehr schnell wieder gegen Null geht. Es gibt spezielle Varianten, die ihre Ladung bis zu einer Minute halten können, diese weisen jedoch einen hohen Drift auf.

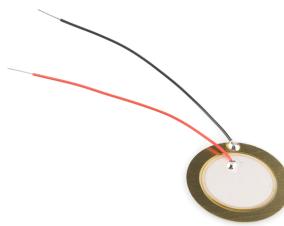


Abbildung 6.13: Piezoelektrischer Sensor

Quelle: <https://www.sparkfun.com/products/10293>

7 Drahtlose Verbindung zwischen den Komponenten mit dem LoRaWAN Protokoll

7.1 Aufsetzen eines Servers (LoRa-Gateway)

Unser LoRa-Gateway hat im Prinzip die Funktion eines WLAN-Routers. Alle Clients (LoRa-Nodes) melden sich an dem Gateway an und kommunizieren mit dem Gateway. Wie der Begriff „Gateway“ schon andeutet, ist dies ein Tor, das zum Internet führt (siehe folgende Abbildung 7.1).

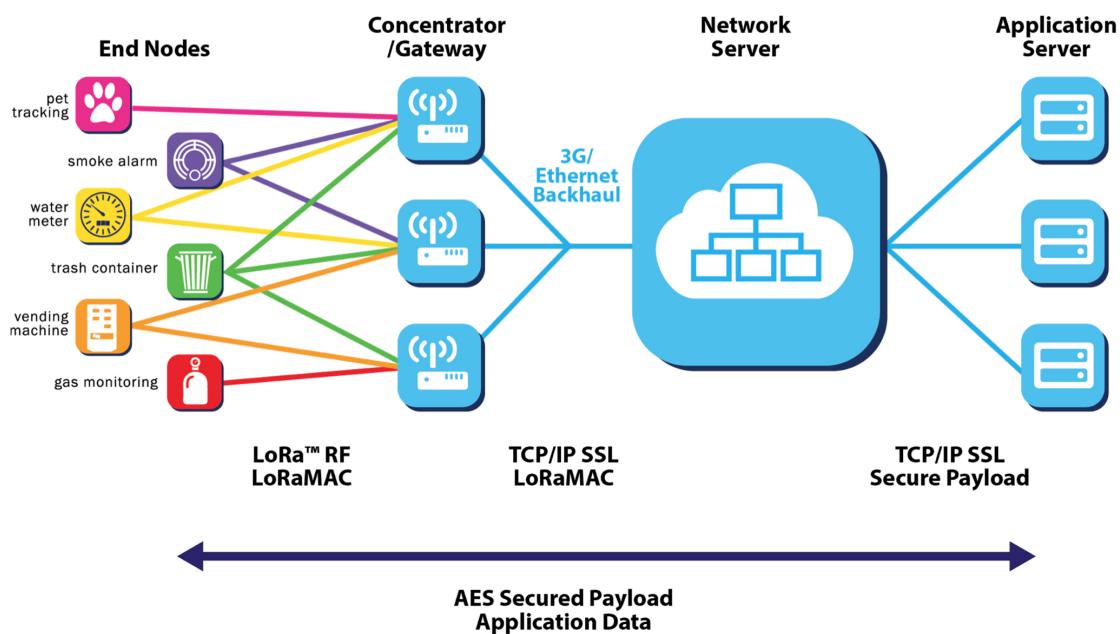


Abbildung 7.1: Übersicht eines Beispiel LoRa-Netzwerks

Quelle: <http://www.mstanley.co.uk/blog/wp-content/uploads/2015/11/Enabling-world-wide-mobility-for-the-IoT-image-2.jpg>

Realisiert wird das alles durch ein „iC880A-SPI Concentrator Board“, das mit Hilfe einer Antenne mit den LoRa-Nodes Daten austauscht. Diese Daten werden durch ein Raspberry Pi, der an das Serial Peripheral Interface (SPI) des Concentrator Boards angeschlossen ist, ins Internet versendet.

ACHTUNG! Es ist sehr wichtig, dass alles korrekt angeschlossen ist und schon eine

Antenne an das Concentrator Board angeschlossen ist, da es sonst zu Schäden an den Bauteilen kommen kann.

Auf dem Wiki der GitHub-Seite von The Things Network Zurich (<https://github.com/ttn-zh/ic880a-gateway/wiki>) kann man nachlesen welche Bauteile man benötigt, wie man die Pins verbinden soll, das Betriebssystem aufsetzt und anschließend das Gateway im TheThingsNetwork registriert.

Nachdem man die Anweisungen zum Konfigurieren befolgt hat, sollte es auf der Webseite für Gateways von TheThingsNetwork zu sehen sein.

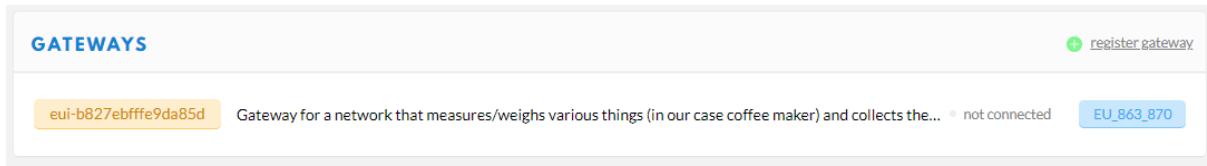


Abbildung 7.2: Übersicht der registrierten Gateways im TheThingsNetwork

Quelle: <https://console.thethingsnetwork.org/gateways>

7.2 Erstellen einer LoRa-Applikation zum Abrufen der Daten

Damit wir jetzt auch mit den verschlüsselten Daten, die wir empfangen, etwas anfangen können, müssen wir erstmal eine Applikation in der Konsole von TheThingsNetwork erstellen.

The screenshot shows the 'APPLICATIONS' section of the TheThingsNetwork console. It lists one application named 'lora_measure_network'. The application details are as follows:

- Name:** lora_measure_network
- Description:** Network that measures/weights various things (in our case coffee maker) and collects their status (f...)
- Handler:** ttn-handler-eu
- MAC Address:** 70 B3 D5 7E D0 00 C5 AE

A red box highlights the 'add application' button in the top right corner.

Abbildung 7.3: Übersicht der erstellten Applikationen im TheThingsNetwork

Quelle: <https://console.thethingsnetwork.org/applications>

The screenshot shows the 'ADD APPLICATION' form. The fields filled in are:

- Application ID:** test2414414 (highlighted by a red box)
- Description:** Just a test application. (highlighted by a red box)
- Application EUI:** EUI issued by The Things Network (empty field)
- Handler registration:** ttn-handler-eu (highlighted by a red box)

At the bottom right, there are 'Cancel' and 'Add application' buttons, with 'Add application' highlighted by a red box.

Abbildung 7.4: Formular zur Erstellung einer Applikation

Quelle: <https://console.thethingsnetwork.org/applications>

Application ID: Sollte einzigartig und eindeutig sein.

Description: Eine kurze, aber aussagekräftige Beschreibung der Applikation.

Handler registration: Sollte passend zum Standort gewählt werden.

Nachdem man nun eine Applikation erstellt hat, sollte man nun auch ein Gerät zu dieser Applikation hinzufügen.

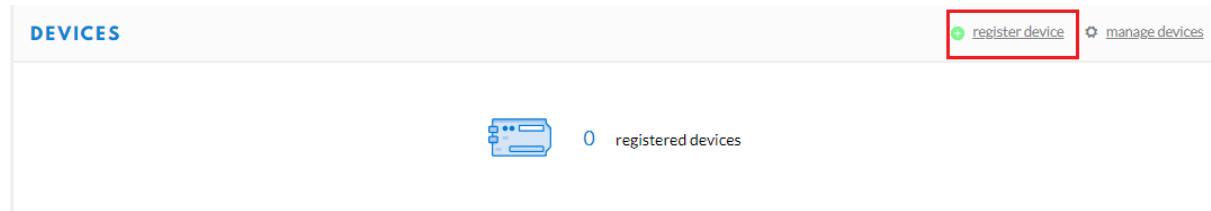


Abbildung 7.5: Übersicht der registrierten Geräte

A screenshot of a 'REGISTER DEVICE' form. It includes fields for 'Device ID' (containing 'test1'), 'Device EUI' (with a clear button), 'App Key' (with a generate button), and 'App EUI' (containing '70 B3 D5 7E D0 01 05 17'). At the bottom right, there are 'Cancel' and 'Register' buttons, with 'Register' being highlighted by a red box.

Abbildung 7.6: Formular zur Registrierung eines Geräts

Quelle: <https://console.thethingsnetwork.org/applications>

Device ID: Sollte einzigartig und am besten aussagekräftig gewählt werden, damit man diese ID später einem Gerät zuordnen kann.

Device EUI: Kann man selber bestimmen oder generieren lassen, indem man das Zufällig-Icon bzw. den Stift-Icon anklickt, um es umzuschalten. Dies wird später benötigt, um das Gerät richtig zu konfigurieren!

App Key: Kann man selber bestimmen oder generieren lassen, indem man das Zufällig-Icon bzw. den Stift-Icon anklickt um es umzuschalten. Dies wird später benötigt um das Gerät richtig zu konfigurieren!

App EUI: Es sollte schon eine vorgewählt sein. Nachdem wir ein Gerät erstellt haben, sollten wir die Aktivierungsmethode ändern:

Dazu gehen wir in *Application* → *application_id* → *devices* → *device_id* → *Settings*. Darin kann man die Activation Method von OTAA zu ABP ändern.

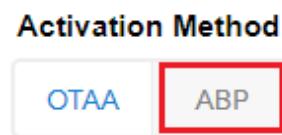


Abbildung 7.7: Aktivierungsmethoden: links Over-the-Air Activation (OTAA), rechts Activation By Personalization (ABP)

Quelle: <https://console.thethingsnetwork.org/applications>

Wenn wir die Änderungen speichern, sollten wir wieder auf die Seite des Geräts kommen. Scrollen wir ganz nach unten, dann sehen wir ein Codebeispiel.

The image shows a code editor window titled 'EXAMPLE CODE'. It contains three lines of C-style code defining device addresses and keys. The code is color-coded: 'const' is blue, 'char' is green, and the variable names and their values are in black. The values themselves are in yellow-green.

```
1 const char *devAddr = "26011C0D";
2 const char *nwkSKey = "FEDC44AB0EED5686F0A43523612ECE23";
3 const char *appSKey = "DAF11CBE2CE9E1345BA1715CE3407D25";
```

Abbildung 7.8: Beispielcode

Quelle: <https://console.thethingsnetwork.org/applications>

Diesen Codeausschnitt sollten wir entweder irgendwo speichern oder uns einfach

merken, dass er auf der Seite des Geräts ist, da wir ihn später zum Konfigurieren des Nodes wieder brauchen werden.

7.3 Aufsetzen eines Clients (LoRa-Node)

Bevor wir die LoRa-Nodes mit Strom versorgen, sollten wir erstmal eine Antenne an den RFH (Radio Frequency High Band – PIN 23) Anschluss befestigen.

Unser LoRa-Node ist über die serielle Schnittstelle UART (PIN 6 + 7 am RN2483) an den Mikrocontroller Atmega644PA (PD0 bzw. PIN 14 + PD1 bzw. PIN 15) angeschlossen und kommuniziert so mit dem Mikrochip. Die Baudrate für die UART Verbindung beträgt standardmäßig 57600 bps, kann aber auch mit einer gewissen Character Folge geändert werden. (Siehe folgende Abbildung 7.9)

1.4 UART INTERFACE

All of the RN2483 module's settings and commands are transmitted over UART using the ASCII interface.

All commands need to be terminated with <CR><LF> and any replies they generate will also be terminated by the same sequence.

The default settings for the UART interface are 57600 bps, 8 bits, no parity, 1 Stop bit, no flow control. The baud rate can be changed by triggering the auto-baud detection sequence of the module. To do this, the host system needs to transmit to the module a break condition followed by a 0x55 character at the new baud rate. The auto-baud detection mechanism can also be triggered during Sleep to wake the module up before the predetermined time has expired.

Note: A break condition is signaled to the module by keeping the UART_RX pin low for longer than the time to transmit a complete character. For example, at the default baud rate of 57600 bps keeping the UART_RX pin low for 938 µs is a valid break condition, whereas at 9600 bps this would be interpreted as a 0x00 character. Thus, the break condition needs to be long enough to still be interpreted as such at the baud rate that is currently in use.

Abbildung 7.9: Spezifikationen für das UART-Interface des RN2483

Quelle: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>

Kommen wir nun zur Initialisierung des RN2483, wenn dieser nun am Microchip hängt und soweit ansprechbar ist:

Folgende Methoden haben wir von der Drotek RN2483 Library (<https://github.com/drotek/RN2483>) für Arduino genommen und für den AVR angepasst und umgeschrieben. Außerdem haben wir einen Teil von der UART-Library von unserem Beitreuer Ulrich Straus übernommen.

Zu aller erst setzen wir den RN2483 zurück, indem wir per UART die Befehle „**sys factoryRESET**“ und „**sys reset**“ senden und jeweils mit einem Break- und New-Line-Character bestätigen. (ASCII: Break = 0x0D; New-Line = 0x0A).

Als nächstes füttern wir den RN2483 mit den Daten, die wir beim Erstellen des Geräts im Applikations-Server erhalten haben. Dazu werden folgende Befehle benötigt:

- **mac set devaddr <address>**
- **mac set nwkskey <nwksesskey>**
- **mac set appskey <appSesskey>**

Also mit dem Beispielcode von Abbildung 7.8 wäre dann:

<address> = 26011C0D;

<nwksesskey> = FEDC44AB0EED5686F0A43523612ECE23;

<appSesskey> = DAF11CBE2CE9E1345BA1715CE3407D25

Nun müssen wir noch den SpreadingFactor (kurz: SF) festlegen und anschließend die Konfiguration speichern und dem Netzwerk beitreten.

Dies wird wie folgt gemacht:

- **mac set dr 5: wobei 5 = SF7/125 kHz**
- **mac save**
- **mac join abp**

Nach jedem Befehl benutzen wir derzeit noch eine Delay-Funktion um dem RN2483 Chip Zeit zum Bearbeiten und Antworten zu lassen. Die aufgerundeten Delay-Zeiten bzw. Berechnungszeiten sieht man im folgenden Diagramm.

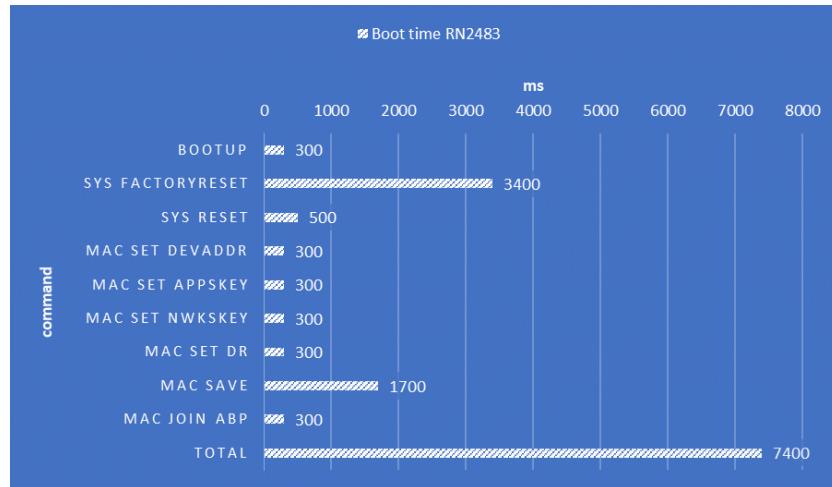


Abbildung 7.10: RN2483 Boot Time unterteilt in die einzelnen Funktionen und der benötigten Zeit in ms

Jetzt ist der RN2483 fertig konfiguriert und wir können mit dem Befehl „**mac tx <type> <portno> <data>**“ Nachrichten versenden. Dabei ist:

<type> = cnf (confirmed) oder uncnf (unconfirmed);

<portno> = Portnummer zwischen einschließlich 1 und 223;

<data> = Die Nachricht als Hexadezimal Wert.

Die vollständige Kommandoliste mit Syntax und Rückgabewerten findet man im RN2483 LoRa Technology Module Command Reference User's Guide: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>

Wir benötigen aus dieser Liste nur die System Commands aus Kapitel 2.3 und die MAC Commands aus Kapitel 2.4

7.4 Spreading Factor

Spreading Factor (kurz SF) ist, wie der Begriff teilweise schon verrät, ein Faktor, der angibt wie weit sich eine versendete Nachricht ausbreitet, also was für eine Reichweite sie hat. Umso höher dieser Faktor, desto höher die Reichweite. Jedoch leidet die Geschwindigkeit der Übertragung darunter. Man kann sich merken: kleinerer SF = höhere Bitrate, nicht so hohe Reichweite. Hoher SF = niedrigere Bitrate, höhere Reichweite. In der folgenden Tabelle ist dies sehr gut erkennbar.

DataRate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000
8..15	RFU	

Abbildung 7.11: Tabelle mit verschiedenen Spreading Factors und die zugehörige Bitrate

Quelle: <https://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png>

Derzeit sind unsere Payloads (Nachrichten) durchschnittlich nur 2 Bytes lang, da wir nur den Füllstand übermitteln. Beim SF7 könnten wir 647 Nachrichten am Tag sen-

den, wenn wir unter dem 30 Sekunden Sendelimit bleiben wollen.

Die genaue Berechnung und weitere Infos zum Spreading Factor findet man auf folgender Google Docs Tabelle: <https://docs.google.com/spreadsheets/d/1QvcKsGeTTPpr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I/edit>

7.5 Security / Datensicherheit

Jedes gute System sollte vor Angreifern geschützt sein. Und das ist auch bei LoRaWAN von TheThingsNetwork der Fall. Die Nachrichten werden vor dem Versenden verschlüsselt und werden beim Empfänger mit Hilfe des App-Session-Key entschlüsselt. Da die Nachrichten sowieso abgefangen werden können, weil ein Radio-Protokoll benutzt wird, kann der Angreifer diese Nachricht nicht lesen, da sie verschlüsselt ist. Er könnte aber diese Nachricht erneut senden (Replay-Angriff). Als Maßnahme, um gegen diese Art von Angriff zu schützen, wird ein Frame Counter eingesetzt der jedes Mal, wenn eine Nachricht versendet wird sich um 1 erhöht. So werden alle alten Nachrichten ignoriert und man ist vor Replay-Angriffen geschützt.

Weitere Infos hierzu gibt es auf der TheThingsNetwork Seite zu Security: <https://www.thethingsnetwork.org/docs lorawan/security.html>

Folgende Abbildung zeigt auch wie die Komponenten (LoRa-Nodes und LoRa-Gateway) mit den Daten umgehen.

FIGURE 1-1: SIMPLE LoRa™ TECHNOLOGY NETWORK DIAGRAM

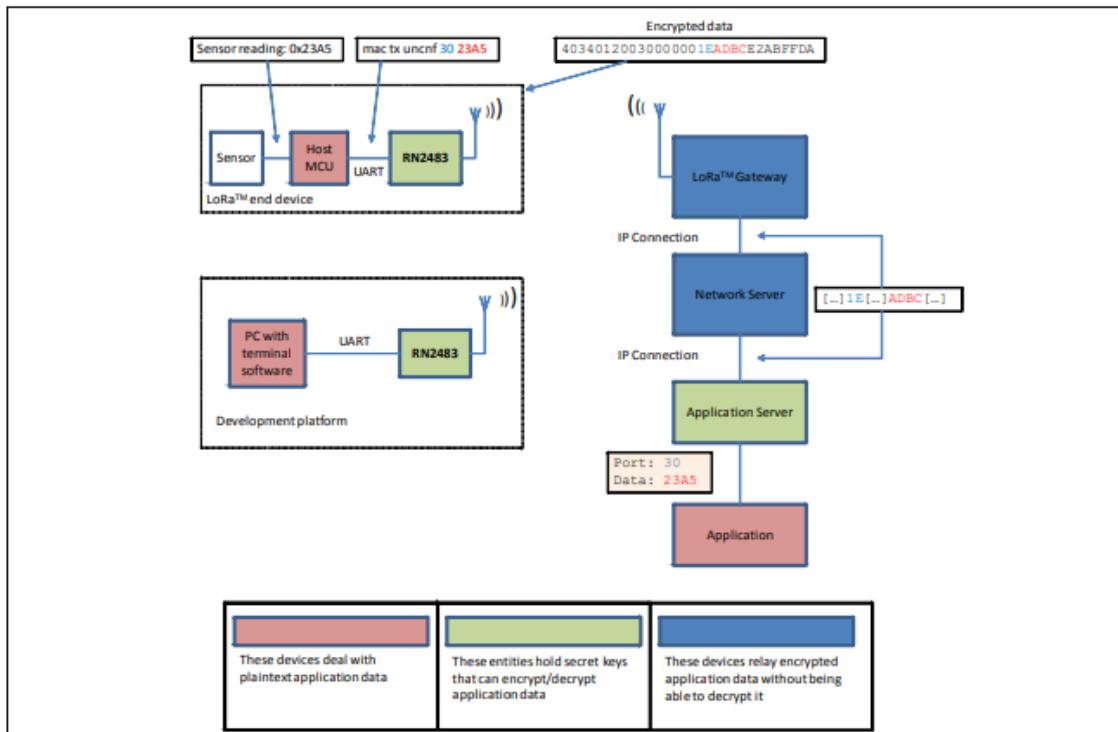


Abbildung 7.12: Verschlüsselung und Datensicherheit bei der Kommunikation zwischen verschiedenen LoRa-Geräten

Quelle: <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>, Kapitel 1.1, Seite 1

7.6 RN2483- und UART-Methoden und ihre Funktionalität

7.6.1 RN2483.c

- **RN2483_init()**: Initialisiert / Konfiguriert das RN2483 Modul wie im Codebeispiel 8.1 zu sehen ist.
- **RN2483_sendData(char *s)**: Konvertiert mit sendHex2(s) das gewünschte Char-Array in Hexadezimal ASCII Code und sendet es mit der UART-Schnittstelle an das Modul, welches anschließend die Nachricht über LoRaWAN an das Gateway schickt.
- **RN2483_sendCmd(char *cmd)**: Sendet ein Kommando über die UART-Schnittstelle an das Modul.
- **RN2483_prepareMessage(int fillRatio)**: Konvertiert ein Integer Wert in Hexadezimal ASCII Code und sendet es über die UART-Schnittstelle an das Modul,

welches anschließend die Nachricht über LoRaWAN an das Gateway schickt.

- **delayFunction(int number):** Führt einen Delay aus, da bei uns maximal nur 850ms am Stück mit der Delay Funktion gewartet werden kann. Bei Parameter-Wert 1 wird 3400ms gewartet, bei 2 wird 500ms gewartet, bei 3 wird 300ms gewartet und bei 4 wird 1700ms gewartet.

7.6.2 uart.c

- **init_uart():** Initialisiert die UART-Schnittstelle, d.h. setzt die Baudrate, aktiviert die RX und TX Pins und setzt den asynchronen UART Modus mit 8 Data bits, no parity, 1 stop bit.
- **uart_putc(unsigned char c):** Sendet einen Buchstaben über UART, wenn die Schnittstelle bereit ist.
- **sendString(char tempStringChar[]):** Sendet ein Char-Array über UART, indem es für jeden Char die uart_putc() Methode aufruft.
- **sendHex2(char *s):** Konvertiert das gegebene Char-Array in Hexadezimal ASCII Code und sendet es über die sendString() Methode.

8 Application Server

8.1 Visualisierung der empfangenen Daten mit Angular CLI

Das Front-End ist die Ansicht, auf welche der Kunde blickt, um die Füllstände aller Behälter zu sehen. Es ist minimalistisch gehalten, sodass keine Überladung entsteht und das menschliche Auge die relevante Information schneller erkennt.

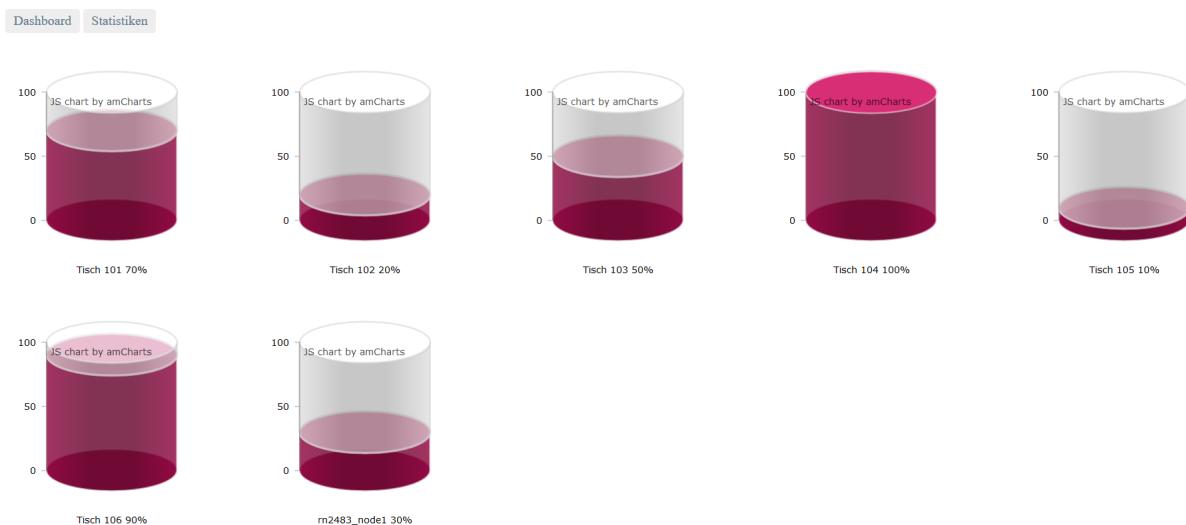


Abbildung 8.1: Übersicht der Füllstände der einzelnen Kaffeemaschinen auf der Webseite

Wie auf dem Bild zu sehen ist, gibt es zwei Seiten, auf die man blicken kann. Auf der Haupt- / Startseite (Dashboard) kann man die Zylinder mit Ihren Inhalten sehen. Jeder aktive Node wird hier mit ihren aktuellen Füllständen angezeigt. Die zweite Seite (Statistiken) zeigt auf einem größeren Diagramm die historischen Werte bis zum aktuellen Zeitpunkt an. Die Historischen Werte zeigen jedoch nur die Daten des Vortags bis zum neuesten Stand an.

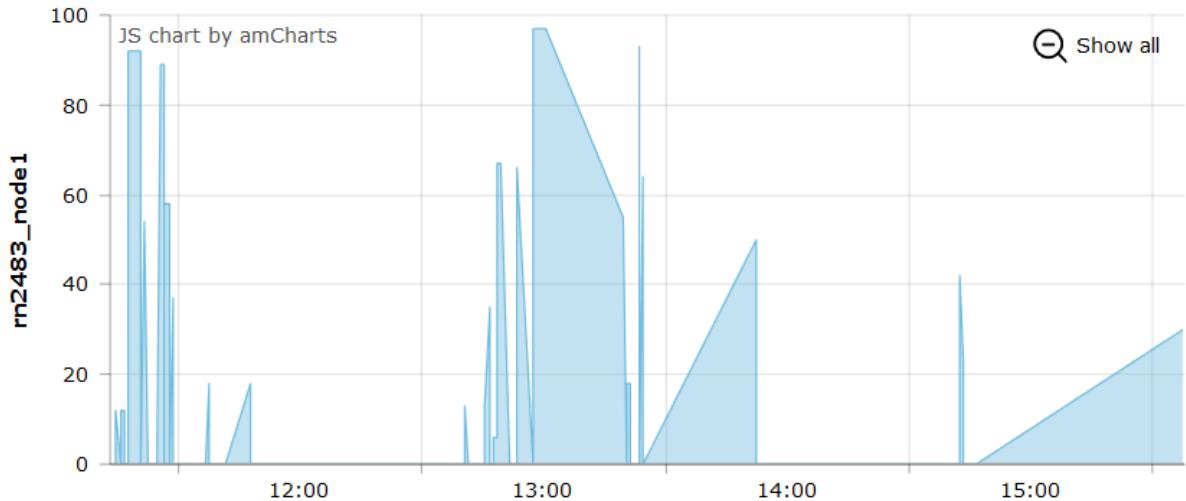


Abbildung 8.2: Füllstand-Verlauf einer Kaffeemaschine von 12:00 Uhr bis 15:00 Uhr

Die Visualisierung ist mithilfe des Front-End-Frameworks Angular CLI geschrieben worden. Die Programmiersprache von Angular ist TypeScript, eine Weiterentwicklung von JavaScript, mit einigen Verbesserungen wie die Typisierung von Variablen. Weiterhin wurden die Zylinder-Behälter mit der „AmCharts“ Angular-Bibliothek implementiert. Diese Bibliothek bietet die Möglichkeit diese Diagramme mit Hilfe einer „create“ Funktion zu erstellen und einer „update“ Funktion mit neuen Daten zu versorgen und damit das Diagramm in Echtzeit zu verändern. Darüber hinaus wurden die Statistikdiagramme mit Hilfe der gleichen Bibliothek erstellt. Diese Diagramme bieten die gleichen Funktionen wie die Zylinderdiagramme.

8.2 NodeJS Server für die Verarbeitung der Datenpakete

Der Server ist in der Programmiersprache NodeJS geschrieben. Darin wurde ein Express Server implementiert. Dieser Express Server nutzt einen Websocket, um die Verbindung zwischen Server und Client herzustellen. Diese Verbindung wird von dem NPM (Node Package Manager) Modul „ttn“ genutzt. Hierbei ist ttn eine Bibliothek des Networkservers „The Things Network“. Durch das ttn Modul bekommt der Server die Nachrichten vom Network Server und damit kann das Paket verarbeitet und per Websocket an den Client weitergeleitet werden. Während der Weiterverarbeitung erfolgt auch die Speicherung einkommender Daten.

8.3 The Things Network

Das TTN Modul (SDK) ist die Schnittstelle zwischen dem Express Server und dem Network Server. Die Verbindung wird automatisch hergestellt. Damit jedoch eine Verbindung hergestellt werden kann, müssen im Voraus zwei Werte vorhanden sein. Der

erste ist die Application ID. Diese ID beschreibt das Projekt, welches vorher bei der TheThingsNetwork-Webseite festgelegt wurde. In diesem Fall ist die Application ID: lora_measure_network. Der zweite Wert ist der accessKey. Dieser ist für die Authentifizierung des SDK's mit dem Network Server erforderlich. Der Key muss vorher über die Webseite generiert werden. Die Nachrichten, zwischen Network Server und SDK, sind in einem bestimmten Format:

```
{  
    app_id: 'lora_measure_network',  
    dev_id: 'rn2483_node1',  
    hardware_serial: '0004A30B001C6A30',  
    port: 1,  
    counter: 0,  
    payload_raw: <Buffer 33 30>,  
    metadata: { time: '2018-07-05T18:04:13.482082016Z' },  
    message: '30'  
}
```

Listing 8.1: Payload

- **app_id:** Application ID
- **dev_id:** Das Gerät, welches die Nachricht gesendet hat
- **hardware_serial:** Hardwarenummer des Nodes
- **counter:** Messagecounter der bei jeder Nachricht um 1 inkrementiert
- **payload_raw:** Die Nachricht wird vom Network Server als Hexadezimal
- **metadata:** In diesem Objekt ist ein Zeitstempel enthalten. Dieser Zeitstempel ist von dem Zeitpunkt, als die Nachricht im Network Server angekommen ist.
- **message:** Das ist die Payload, welche vom payload_raw geparsst wurde

8.4 Inaktivität der Behälter

Sobald ein Gerät inaktiv wird (d.h. eine Stunde ohne Nachrichtenempfang), ist es für die Nutzer in der UI nicht mehr möglich, das Gerät zu sehen. Diese Inaktivität würde bei einem Thermobehälter die Abkühlung bedeuten. Das Limit basiert auf dem Resultat einer Gesprächsrunde innerhalb der Entwickler. Sobald jedoch das Gerät wieder sendet, wird es in der UI eingeblendet. Dieses Limit hat den Hintergrund, dass ein Gerät unerwartet vom Netz genommen wird. Wenn dieser Fall eintreten würde, könnte man in der Visualisierung nicht erkennen, ob das Gerät überhaupt noch existiert.

8.5 Voraussetzungen und Nutzung des Node-Servers

Unterstützte Betriebssystem

- Windows
- Linux
- Unix

Benötigte Software

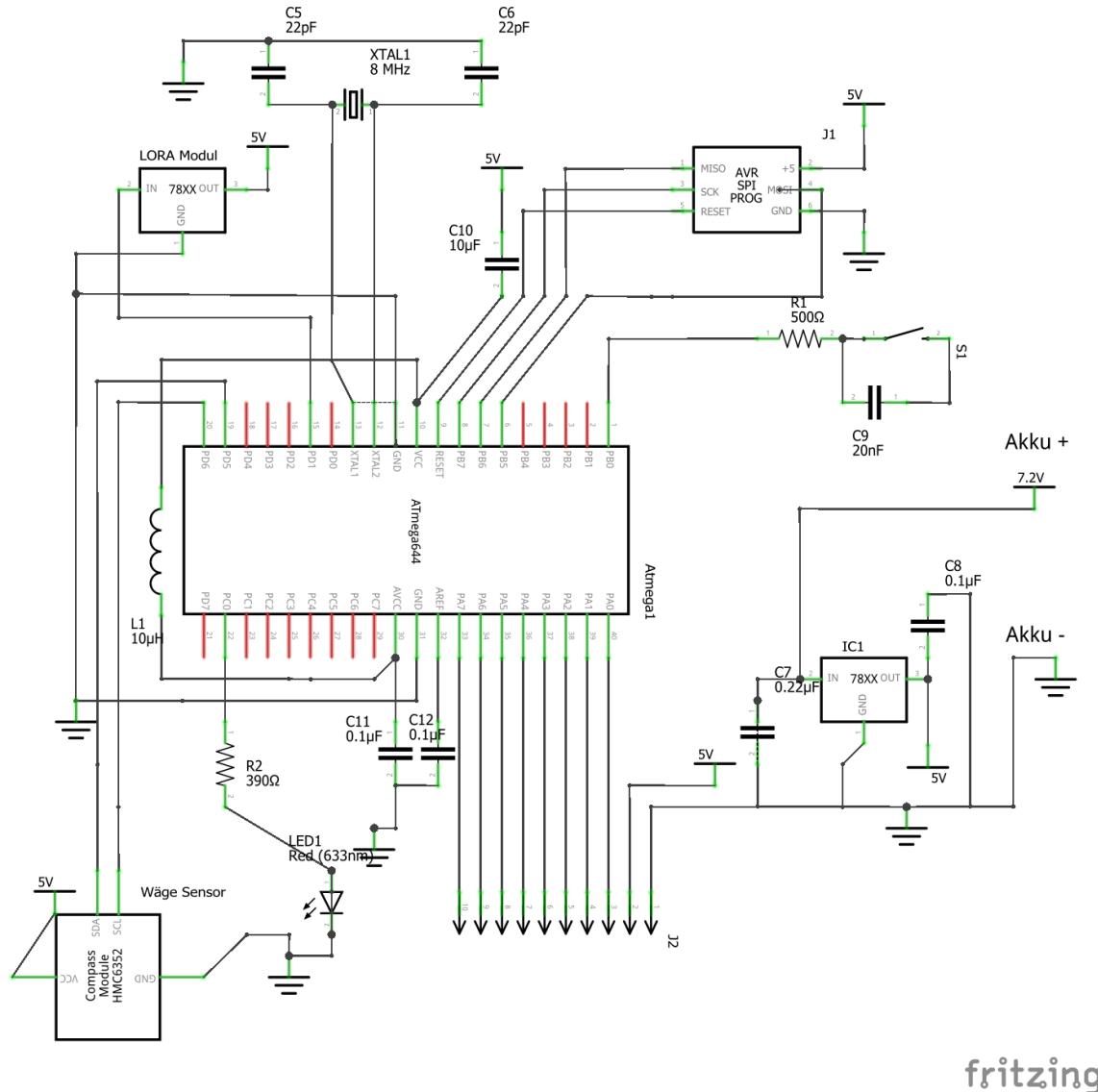
- NodeJS (inkl. NPM)
- Angular

Nutzung

- Starte zwei Kommandozeilenprogramme
 - Konsole 1 im Ordner „server“: node server.js
 - Konsole 2 im Ordner „client“: ng serve -open

9 Schaltplan und Verlöten der Platine

Der Schaltplan wurde mit dem Programm Fritzing (<http://fritzing.org/home>) erstellt.



fritzing

Abbildung 9.1: Schaltplan, erstellt mit Fritzing

Als Mikrocontroller haben wir im Schaltplan einen ATMega644 statt einem ATMega644PA verwendet, da diese fast gleich sind. Ausgehend vom Mikrocontroller werden an den Pins XTAL1 und XTAL2 der Quarz angeschlossen. An Pin PA0 bis PA7 werden die Pins für das LCD-Display angeschlossen. An PC0 wird eine rote LED und davor der Widerstand R2 mit 390 Ohm angeschlossen. Der Wäge Sensor wird mit den Pins PD5 und PD6 angeschlossen. Hierbei wurde für den Sensor einfach ein Platzhalter mit vier Pins genommen, da kein passendes Teil für unsere Wägezelle gefunden wurde. Das LoRa-Modul wird an PD1 angeschlossen und es wurde ebenso eine Platzhalterkomponente mit drei Pins verwendet. Zudem haben wir einen Taster auf PD0 und an PD5 – PD7 einen ISP-Anschluss angelötet. Die Platine und dessen Bauteile werden mit zwei in Reihe geschalteten Akkus versorgt die jeweils ca. 3,6V liefern. Dazwischen geschaltet ist ein DC-DC Wandler, der den Stromkreis mit konstanten 5V versorgt. Dann haben wir anhand unseres Schaltplans unsere Platine verlötet.

Unsere Platine sieht im Moment so aus:

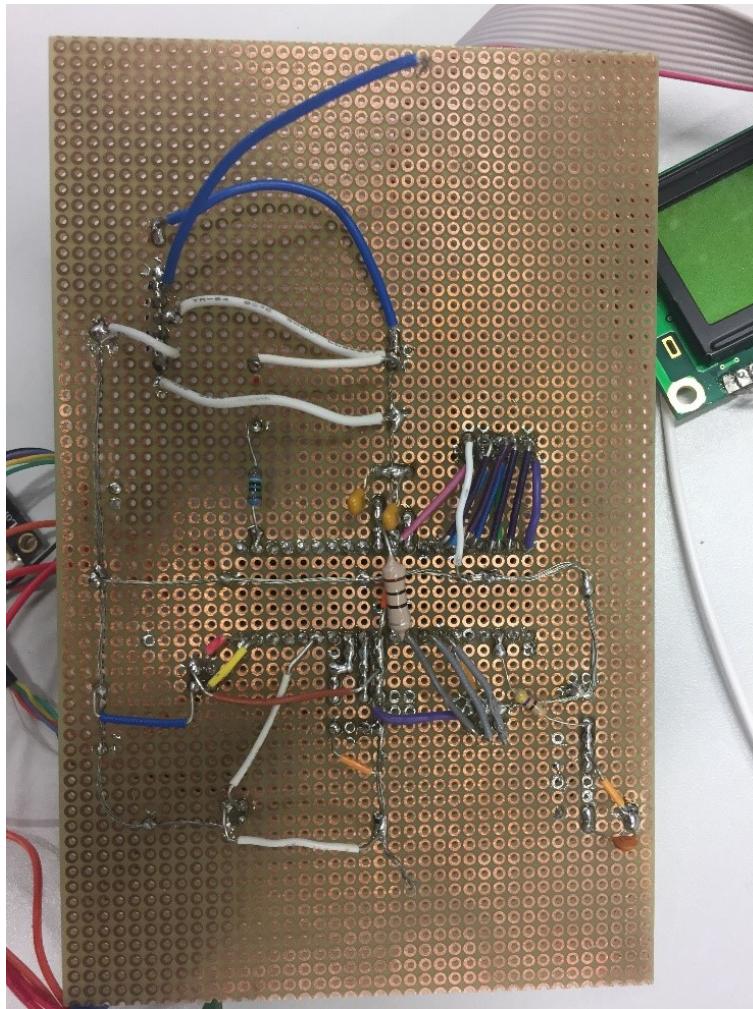


Abbildung 9.2: Rückseite der verlötenen Platine unseres ersten Prototyps

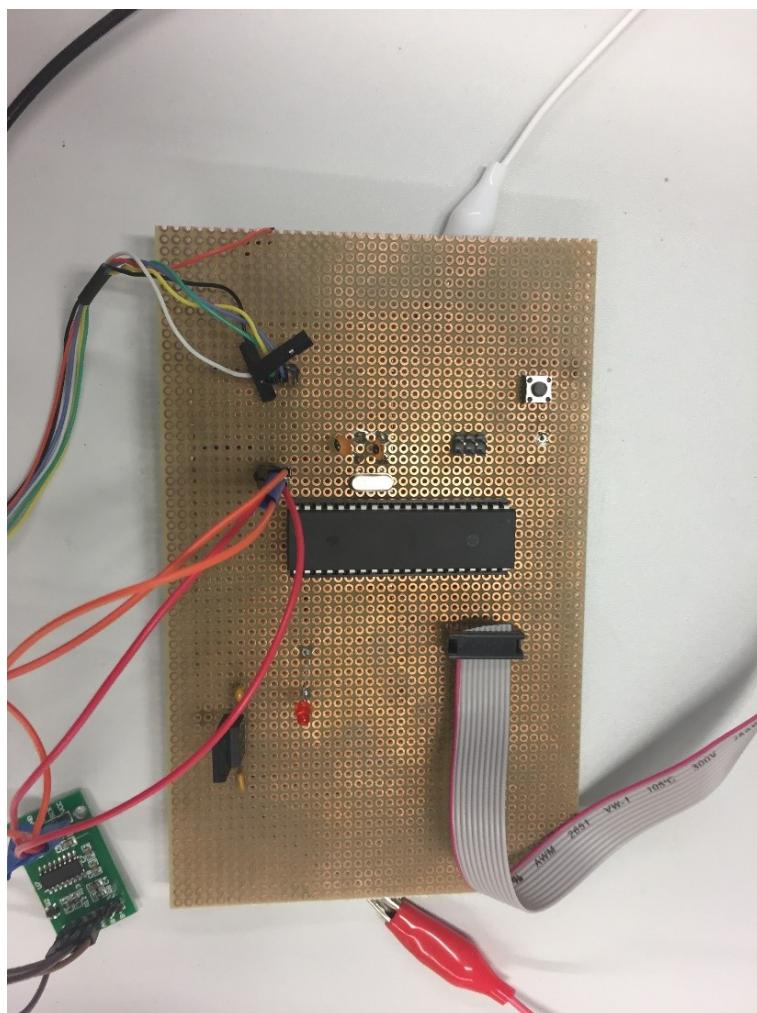


Abbildung 9.3: Vorderseite der verlötenen Platine unseres ersten Prototyps

Dem Taster wurde eine Tar-Funktion zugewiesen, mit der das Gewicht genullt werden kann. Softwareseitig ist es auch möglich dem Taster andere Funktionen zuzuweisen, wie beispielsweise das An- und Ausschalten des Gerätes oder mit Verbund eines Displays zwischen verschiedenen Anzeigemodi zu Wechseln.

10 Ausblick

10.1 LoRaWAN Modul Code verbessern

- Firmware vom **RN2483** updaten
- Gebrauch von der sleep Funktion „sys sleep <length>“ machen um Strom zu sparen
- Beim Ausschalten den Frame-Counter mit „mac get upctr“ auslesen und in den EEPROM vom Atmega644PA speichern. Beim Einschalten den Frame Counter aus dem EEPROM auslesen und mit „mac set upctr <fCntUp>“ wieder neu setzen, damit man nicht jedes Mal den Frame Counter in der Konsole von The-ThingsNetwork zurücksetzen muss.
- Empfang von Nachrichten oder Befehlen vom Gateway ermöglichen und durch UART-Receive diese Nachrichten bzw. Befehle im Mikrochip auslesen.
- Delays mit Interrupts ersetzen, so dass das Programm weiterarbeiten kann, während es auf ein Ereignis wartet.
- Verschiedene Profile oder Modis passend zu den Spreading Factors programmieren z.B.: niedrige Reichweite + hohe Bandbreite, hohe Reichweite + niedrige Bandbreite, etc.

10.2 Front-End Lizenzkosten vermeiden

Das Frontend hat das Problem, dass es die ÄmCharts "Bibliothek nutzt. Die Bibliothek ist für Open-Source und nicht kommerzielle Projekte frei verfügbar. Die Lizenz (Stand 06.07.2018) kostet für ein Jahr 4590 €. Darin sind die Diagramme in den Seiten Dashboard und Statistik enthalten. In Zukunft sollen selbstgeschriebene Diagramme verwendet werden, um Lizenzkosten zu vermeiden.

10.3 Erstellen eines wasserfesten Gehäuses

In naher Zukunft ist es außerdem noch wichtig, dass sich die Platine und die nötige Hardware in einem Gehäuse befinden, da es sonst nicht im Kommerziellen nutzbar wäre. Zudem sollte das Gehäuse wasserfest sein, da sonst Kaffee oder ähnliches hindurchkommen und etwas beschädigen könnte.

Dadurch wird unser Produkt deutlich kompakter und nutzvoller für Hotels. Unser erster Ansatz wäre ein Prototyp mit einem 3D-Drucker zu erstellen.

10.4 Kommerzialisierung und mögliches Geschäftsmodell

Die nächsten nötigen Schritte zur Kommerzialisierung sind:

1. Kundengespräche:

Wir müssen Gespräche mit unseren potenziellen Kunden, beispielsweise Hotels oder Restaurants führen, damit wir abschätzen können, wie hoch die Nachfrage sein könnte. Hierbei bekommen wir direktes Feedback zu unseren Prototypen, sodass darauf reagiert werden kann. Außerdem müssen wir Kundengruppen erschließen, damit wir eingrenzen können, in welchen Branchen das Produkt nutzbar wäre.

2. Optimierung des Prototyps:

Wir müssen ein wasserdichtes Gehäuse erstellen, damit die Technik verpackt werden kann. Zudem muss das Design des Gehäuses angepasst werden, sodass es in der Gastronomie verwendbar ist. Des Weiteren werden wir den Prototypen verbessern, indem wir einen Sleep-Mode einführen, damit die Laufzeit verlängert wird, den Taster verbessern und unnötige Komponenten entfernen.

3. Erstellen von Business-Plan, Strategie und Dokumente für Unternehmensgründung:

Für eine erfolgreiche Start-Up-Gründung benötigen wir einen Business-Plan, damit wir einen seriösen roten Faden haben und wir damit Investoren überzeugen können. Zudem müssen wir uns Strategien überlegen und die nächsten Schritte planen. Dazu benötigen wir noch weitere Dokumente und Berechnungen für die Unternehmensgründung.

Mögliche Geschäftsmodelle:

Um den Aufwand für Gastronomiebetreiber möglichst klein zu halten, sollten Geräte- und Softwarewartung möglichst outsourced werden. Gastronomiebetreiber sollten nach diesem Modell monatlich oder jährlich einen Festbetrag bezahlen um Installation und Wartung weiter leiten zu können. So könnte zum Beispiel die Software auf einem Cloudserver laufen und Geräte regelmäßig ausgetauscht, geupdatet und gewartet werden.

Literatur

- Electronics, Drotek. *DataLink LoRa RN2483*. <https://drotek.com/shop/en/lora/763-data-link-lora-rn2483.html>.
- Electronics, Mouser. „Analog-to-Digital Converter“. https://www.mouser.com/ds/2/813/hx711_english-1022875.pdf.
- Inc., Microchip Technology. 2015-2017a. *Low-Power Long Range LoRa Technology Transceiver Module*. <http://ww1.microchip.com/downloads/en/DeviceDoc/50002346C.pdf>.
- . 2015-2017b. *RN2483 LoRa Technology Module Command Reference User's Guide*. <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784F.pdf>.
- Kaiser, Reto. *ic880a Gateway*. <https://github.com/ttn-zh/ic880a-gateway/wiki>.
- LoRa(WAN) airtime calculator*. <https://docs.google.com/spreadsheets/d/1QvcKsGeTPPr9icj4XkKXq4r2zTc2j0gsHLrnplzM3I/edit#gid=0>.
- mstanley. *LoRa WAN*. <http://www.mstanley.co.uk/blog/wp-content/uploads/2015/11/Enabling-world-wide-mobility-for-the-IoT-image-2.jpg>.
- Network, The Things. *LoRaWAN Security*. <https://www.thethingsnetwork.org/docs lorawan/security.html>.
- Sparkfun. *Gewichtsmessung*. <https://learn.sparkfun.com/tutorials/getting-started-with-load-cells>.
- Wikipedia contributors. 2018a. *Force-sensing resistor – Wikipedia, The Free Encyclopedia*. [Online; accessed 13-August-2018]. https://en.wikipedia.org/w/index.php?title=Force-sensing_resistor&oldid=846591724.
- . 2018b. *Load cell – Wikipedia, The Free Encyclopedia*. [Online; accessed 13-August-2018]. https://en.wikipedia.org/w/index.php?title=Load_cell&oldid=851413016.
- . 2018c. *Piezoelectric sensor – Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Piezoelectric_sensor&oldid=839026087. [Online; accessed 13-August-2018].
- Witekio. *LoRa WAN spreading factor*. [https://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png](http://witekio.com/wp-content/uploads/2018/01/lora-wan-spreading-factor.png).