Untersuchung eines Hybrid-Buck-Wandlers hinsichtlich seiner Anwendbarkeit für Maschine Learning Applikationen

Bachelorarbeit zur Erlangung des akademischen Grades: Bachelor of Engineering

Sergej Lamert - 00727245
Fakultät angewandte Informatik

21. März 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

NN Neuronales Netz

ML Maschine Learning

SDA Serial Data

SCL Serial Clock

I²C Inter Integrated Circuit

Danksagung

An dieser stelle möchte ich mich bei jedem Bedanken, der mich während dieser Arbeit unterstützt hat.

Ein großer Dank geht an Andreas Federl, welcher mich sowohl in organisatorischer als auch fachlicher Hinsicht beraten und unterstützt hat.

Ebenfalls bedanken möchte ich mich bei meinen Freunden und meiner Familie, die mich sowohl kreativ als auch mental bei der Arbeit unterstützt haben.

Abstract

Die Anwendung von Maschine Learning Algorithmen auf Daten, welche von Hybrid-Buck-Wandler stammen ist momentan nicht weit in der Industrie ausgeprägt. Aus diesem Grund beschäftigt sich diese Arbeit mit der Analyse des o. g. Wandler-Typen hinsichtlich seiner Anwendbarkeit für Maschine Learning Applikationen. Die Analyse beinhaltet die Generierung von Daten, eine Analyse bzgl. der Qualität der Daten sowie der maximalen Abtastrate. Abgeschlossen wird die Analyse mit einem Testfall in Form eines Neuronalen Netzes. Die Ergebnisse der Analyse zeigten, dass die Daten nur geringe Abweichungen von ihrem Mittelwert haben, sowie dass die maximale Abtastrate etwa Hz beträgt. Die Anwendung von Neuronalen Netzen anhand von verschiedene periodischen Signalen zeigte eine Genauigkeit von x. Zusammenfassend besteht für die in dieser Arbeit behandelten Hybrid-Buck-Wandler Potential für ML Applikationen verwendet zu werden, insofern es sich bei dem Leistungssignal um ein niederfrequentes Signal handelt, da die Daten sonst durch den Alias Effekt unbrauchbar gemacht werden.

1 Einleitung

DC-DC-Buck-Wandler, oder auch Tiefsetzsteller, sind heutzutage Bestandteil verschiedenster Applikationen. So stellen diese beispielsweise Spannung in Notebook-Prozessoren und Ladegeräten zu Verfügung oder regeln den Strom an Stepper-Motoren. Durch die große Anzahl an Anwendungsfällen, ist es besonders in der heutigen Zeit, in der Künstliche Intelligenz und die damit einhergehenden Machine Learning Algorithmen immer wichtiger und präsenter werden, zu evaluieren, inwieweit sich solche Wandler für Machine Learning Anwendungen eignen, wenn Daten über Strom, Spannung, Leistung und Temperatur des Wandlers über eine Digitale Schnittstelle ausgelesen werden können. Deshalb beschäftigt sich diesee Arbeit die Analyse eines Hybrid-Buck-Wandlers für Anwendungen im Bereich des Maschine Learning bzw. der künstlichen Intelligenz.

Dadurch bedingt, dass es sich hierbei um einen neu entwickelten Wandler handelt, welcher noch nicht auf dem Markt ist und sich beim Schreiben dieser Arbeit noch in der Testphase befindet, wird die Analyse in mehreren Schritten durchgeführt werden. Zuerst werden in dieser Arbeit allgemeine Testfälle durchgeführt, um die Qualität der Daten sicherzustellen und um einen Einblick in das Verhalten des Wandlers zu gewinnen. Neben der Qualität der Daten muss wird in dieser Arbeit ebenfalls die Quantität der Daten erfasst, dies bedeutet im speziellen, wie viele Daten mit diesem Wandler in einer Zeiteinheit generiert werden können. Des Weiteren wird im Zusammenhang mit dieser Arbeit Software generiert, mit dem die vorhandenen Daten ausgelesen, manipuliert und visualisiert werden können. Schlussendlich ist das Ziel dieser Arbeit, einen ersten Test bezüglich der Anwendbarkeit und Zuverlässigkeit der vom Buck-Wandler generierten Daten für Maschine Learning Applikationen, im speziellen dem der Neuronalen Netze, durchzuführen und zu bewerten.

2 Theoretischer Hintergrund

Dadurch bedingt, das diese Arbeit sich an vielen Konzepten aus der Signaltheorie, Elektrotechnik, Informatik und Mathematik bedient, ist es sinnvoll vorab einige Begriffe und Konzepte zu definieren und zu erläutern. Aus diesem Grund, werden im Folgenden das I²C Protokoll, das Nyquist-Frequenz Theorem, die Grundlagen von Neuronalen Netzen sowie die verwendete Software näher erläutert.

2.1 Hybrid-Buck-Wandler

Bei dem Buck-Wandler handelt es sich um einen DC-DC Wandler, das bedeutet, dieser Wandler nimmt eine Gleichstrom Eingangsspannung von 15 bis 40 Volt entgegen und setzt diese herunter auf eine Gleichstrom Ausgangsspannung von zwölf Volt. In Abb. 1 ist besagter Wandler zu erkennen. Unten rechts in der Abbildung ist der Eingang zu sehen, oben rechts der Ausgang und links mittig ist die Schnittstelle angebracht. Des Weiteren ist der hier verwendete Wandler auf einen Stromfluss von zehn Ampere limitiert, was bedeutet, dass die maximale Leistung am Ausgang des Wandlers 120 Watt beträgt. Es bedeutet ebenfalls, dass, wenn der Strom zehn Ampere übersteigt, die Ausgangsspannung gedrosselt wird. Für den weiteren Verlauf der Arbeit wird für den vorgestellten Hybrid DC-DC Wandler immer der simplere Begriff ”Wandler”verwendet. Der Wandler besitzt ebenfalls eine I²C Schnittstelle, aus welcher Ausgangsspannung, Eingangsspannung, Ausgangsstrom und Temperatur des Boards gemessen und ausgelesen werden können. Die Datenleitungen des I²C Busses sind bereits auf dem Wandler mit Pull-Up Widerständen ausgestattet, wodurch die Leitungen im unbenutzten Zustand auf fünf Volt gesetzt sind. Dadurch müssen keine externen Pull-Up Widerstände an die Leitungen angeschlossen werden.

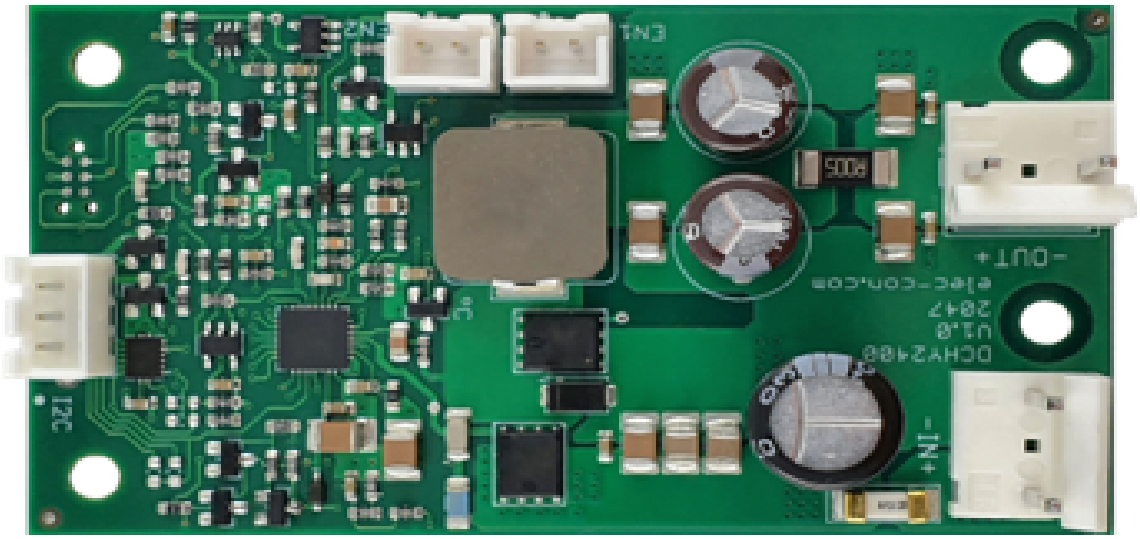


Abbildung 1: Verwendeter Hybrid DC-DC Wandler

2.2 I2C

I2C steht für Inter-Integrated Circuit und wurde von Philips Semiconductors im Jahr 1982 entwickelt. Es handelt sich dabei um einen seriellen Datenbus im Master-Slave Stil. Dieses Protokoll besitzt insgesamt vier Leitungen, von denen jeweils eine für die Versorgungsspannung und eine als Ground verwendet wird. Des Weiteren werden zwei Leitungen für den Datenaustausch verwendet, davon ist ein Kanal explizit für die Daten (Serial Data) und der andere Kanal für den Takt (Serial Clock) vorgesehen. Der Takt des I²C Protkolls ist dabei auf Werte zwischen 100 KHz (standard mode) und 3.2 MHz (high speed) spezifiziert. Abb. 2 sind die genannten Master und Slave Komponenten zu erkennen. Der Master gibt allen Slaves vor, was sie zu senden haben, und wie schnell diese es tun sollen. Des Weiteren ist zu erkennen, das die beiden Leitungen SDA und SCL an einem Pull-Up Widerstand angeschlossen sind. Dieser dient dazu, die Leitungen, wenn weder Master noch Slave senden, auf die Versorgungsspannung zu schalten, sodass keine undefinierten Zustände entstehen und die Leitung im unbenutzten Zustand eine logische eins besitzt, also aktiv ist. Die Kommunikation erfolgt durch Adressen, so besitzt jeder Sensor oder Mikrocontroller eine I2C Adresse, über die ein Master auf diese zugreifen kann. Das auslesen der Daten eines Slaves erfolgt per Registeradressen, so hat beispielsweise ein beliebiger I²C fähiger Sensor ein Register, in dem bestimmte Werte gespeichert sind. In dieser Arbeit handelt es sich bei dem I2C Slave um einen Mikrocontroller auf dem Buck-Wandler. Als Master können sowohl ein PICkit Serial Analyzer als auch ein Raspberry Pi fungieren.

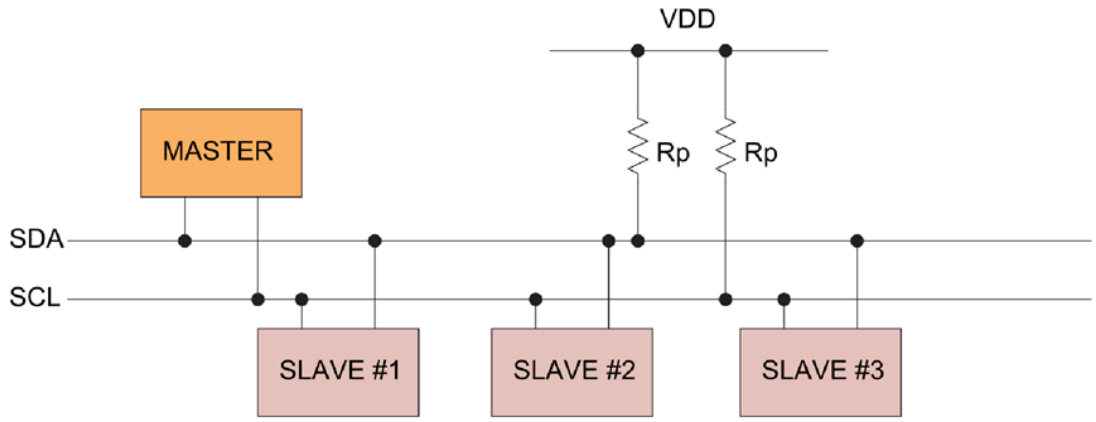


Abbildung 2: I2C Bus Beispiel

Die o. g. Kommunikation des I²C Protokolls erfolgt per Datenframes. Diese beinhalten Informationen bezüglich Start- und Stopbedingungen, Adressen sowie gesendete Daten. In Abb. x ist die Struktur eines I²C Datenframes visualisiert. Dieses beginnt immer mit einer Startbedingung. Eine Startbedingung entspricht einem Wechsel des SDA Signals von High auf Low, während SCL High ist. Sobald SDA auf Low gesetzt wurde, wird auch SCL auf Low gesetzt. Darauf folgt die Adresse des Gerätes, mit dem kommuniziert werden soll, welche zwischen sieben und zehn Bit lang ist. Nach der Adresse folgt ein einzelnes Bit des Senders um zu signalisieren, ob geschrieben oder gelesen werden soll. Danach kommt vom Empfänger ein acknowledge (ACK) oder not-acknowledged (NACK) zurück, je nachdem, ob die anfrage erfolgreich war oder nicht. Im Falle eine nicht erfolgreichen Anfrage (NACK) bricht der Master die Kommunikation ab. Sollte die Anfrage erfolgreich gewesen sein, folgt darauf hin ein oder mehrere Datenbytes mitsamt einem ACK bzw. NACK wenn die Daten erfolgreich empfangen wurden. Sobald die Kommunikation abgeschlossen ist, wird diese mit einer Stop-Condition beendet. Die Stop-Condition ist das Pendant zu der Start-Condition, da diese durch das setzen des SCL Buses auf High und dem darauffolgenden setzen des SDA Buses auf High durchgeführt wird.

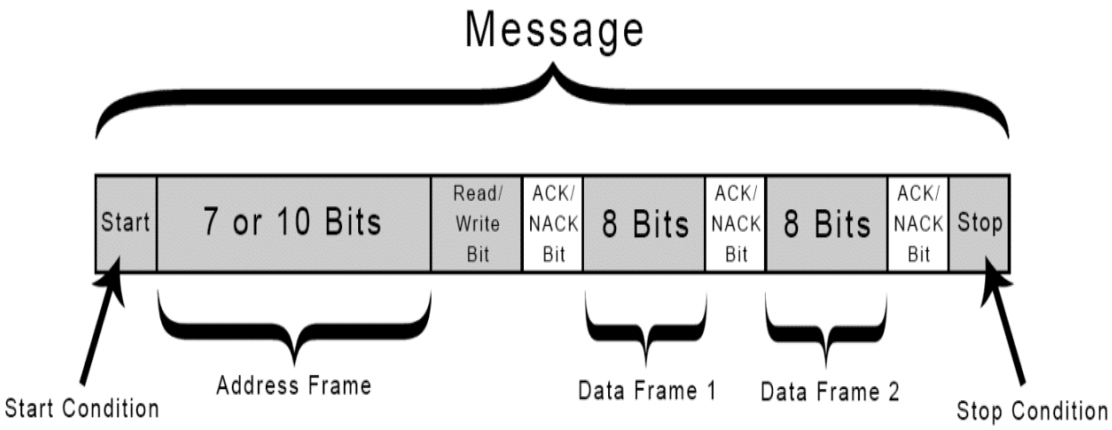


Abbildung 3: Struktur eines I²C Datenframes

2.3 Nyquist Frequenz Theorem

Das Nyquist Frequenz Theorem ist eine fundamentale Aussage über das Verhältnis von Signalfrequenz und Abtastfrequenz bei der Digitalisierung von Analogen Signalen. Für diesen Sachverhalt existiert die Gleichung Nr. X. Diese Gleichung sagt aus, das die maximale Signalfrequenz, welche ohne Verzerrungen, dem sog. Älias-Effekt", dargestellt werden kann, der Hälfte der Abtastrate entspricht.

$$f_{\text{nyquist}} = 1/2 * f_{\text{abtastrate}} \tag{1}$$

Um dies zu verdeutlichen, sind in Abb. X zwei Signale zu erkennen. Das ursprüngliche Signale (in blau aufgetragen) und das zweite Signal (rot aufgetragen), welches durch Abtastung generiert wurde. Bei dem ursprünglichen Signal handelt es sich um eine Sinuswelle mit einer Frequenz von 100 Hz. Die Abtastrate be ebenfalls 100 Hz. Sobald die Formel x auf diesen Sachverhalt angewendet wird, wird klar, dass die maximale Signalfrequenz bei 100 Hz * 1/2 = 50 Hz liegt, die tatsächliche Frequenz jedoch bei 100 Hz. Daraus resultiert der Alias Effekt, welcher im roten Signal zu erkennen ist.

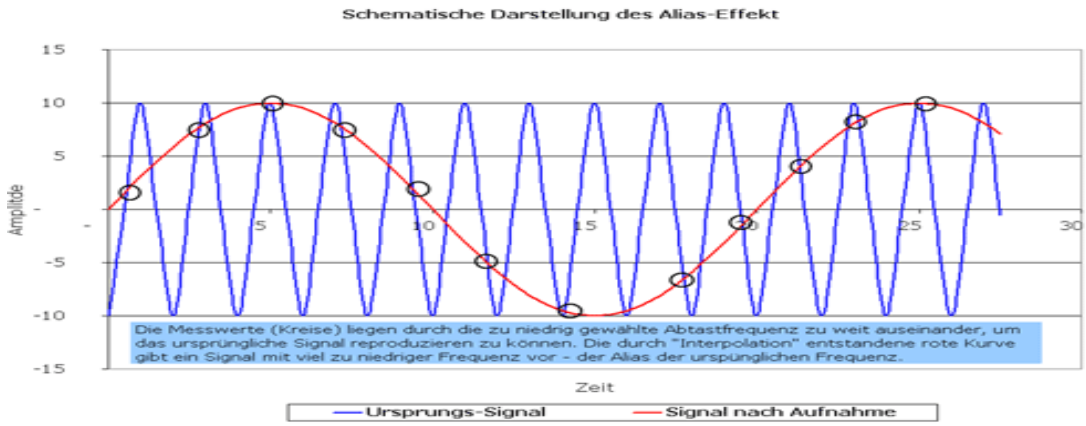
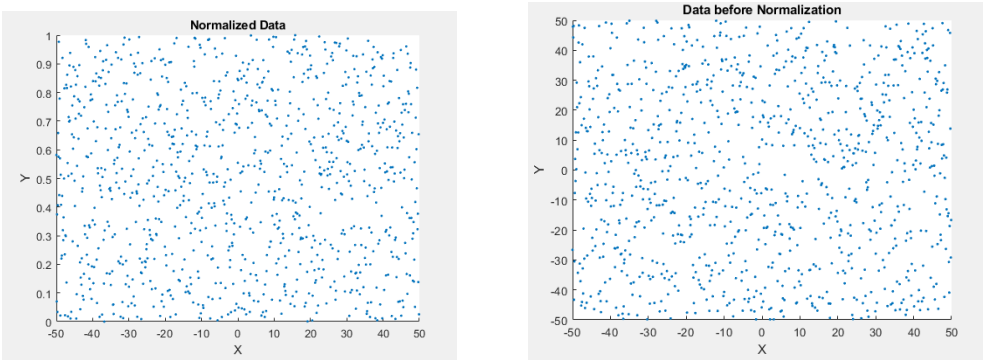


Abbildung 4: Beispiel des Alias Effekt

2.4 Normalisierung der Daten

Im bereich des Maschine Learning und Data Science ist es gängige Praxis vorhandene Daten, welche unterschiedliche Skalierungen besitzen zu Normalisieren. Normalisieren bedeutet, dass Daten insoweit verändert werden, dass sie in einem bestimmten Wertebereich dargestellt werden können. Diesbezüglich existieren mehrere Rechenmethoden, wie z. B. die z-Transformation oder feature scaling. Im Folgenden soll der Algorithmus des feature scaling näher erläutert werden.



(a) Normalisierte Daten (b) Daten vor der Normalisierung

Abbildung 5: Normalisierung von Daten

In Abb. x (b) ist ein Datensatz mit tausend zufällig generierten Werten zwischen -50 und +50 zu erkennen. Durch das anwenden der Gleichung x für Normalisierte Daten, wobei max(x) und min(x) jeweils die maximalen, bzw. minimalen Datenwerte des Datensatzes sind, wird der Datensatz auf Werte zwischen null und eins angepasst. Dies hat für Maschine Learning Anwendungen den Vorteil, dass die Trainingszeit verkürzt werden kann und die Chance verringert wird, in lokalen Minima der Kostenfunktion ßtecken zu bleiben”.

$$X' = \frac{X + max(X)}{max(X) - min(X)} \tag{2}$$

2.5 Neuronale Netze

Zur Durchführung einiger simpler Testfälle unter den Punkten 3.2.1 und 3.2.2 ist es notwendig, eines der Grundlegenden Konzepte von Machine Learning zu erläutern: das Neuronale Netz. Neuronale Netze gehören zur Maschine Learning Kategorie des ßupervised learning". Beim supervised learning werden die Parameter eines neuronalen Netzes anhand der Eingabedaten (Input) und bereits definierten Lösungen (Output) daraufhin optimiert, bei gegebenen Input eine passende Lösungsstrategie für den bereits definierten Output zu finden.

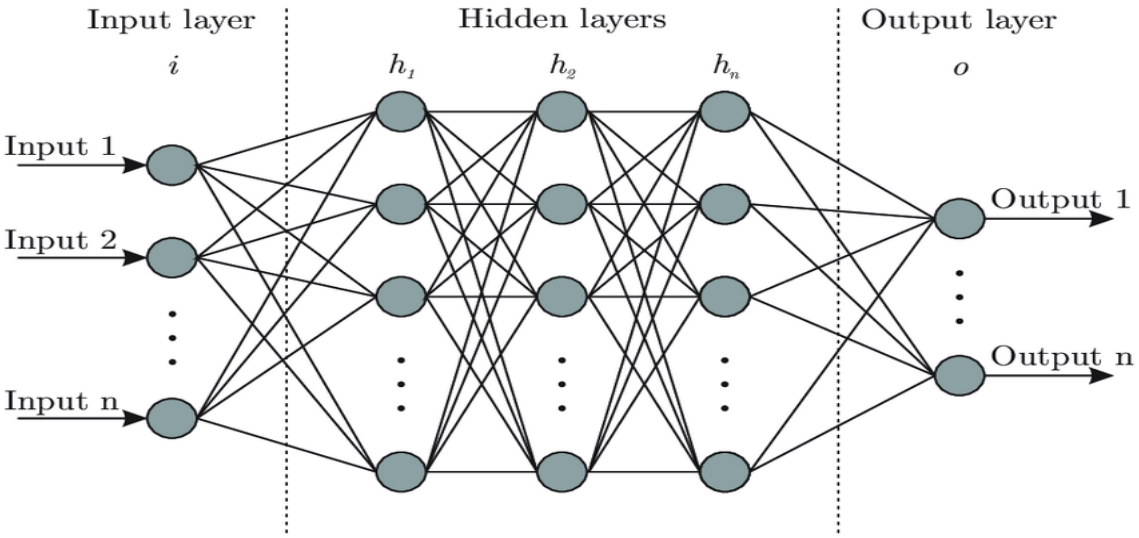


Abbildung 6: Beispiel eines Neuronalen Netzes:

In Abb. 2 erkennt man eine Abbildung der Architektur eines möglichen neuronalen Netzes. Dieses ist aufgeteilt in mehrere Schichten: Input layer,hidden layer und Output layer. Das Input layer nimmt die Daten auf, welche verarbeitet werden müssen. Diese Daten können von verschiedener Art sein, so können es z. B. Pixeldaten eins Bildes sein, oder der Zeitverlauf eines Signals. Im hidden layer werden die Daten durch diverse mathematische Operationen verarbeitet, um im Output layer Aussagen über die Input Daten machen zu können, d. h. um die Input Daten zu klassifizieren. ben den einzelnen Schichten ist zu erkennen, dass es mehrere Knoten gibt, welche untereinander durch Kanten vollvermascht sind. Die Knoten werden im Allgemeinen "Neuronen"genannt und sie besitzen einen numerischen Wert, welcher bestimmt, wie äktivßie sind. Die Kanten, die die Neuronen verbinden, werden "weightsöder Gewichtungen genannt. Die Gewichtungen sind dabei die Hauptparameter eines Neuronalen Netzes.

i Neuronalen Netzen wird zwischen zwei Algorithmen unterschieden, dem Feedforward Algorithmus, welcher bei gegebenen Input Daten und Gewichtungen einen bestimmten Output liefert. Und dem Backpropagation Algorithmus, welcher bei gegebenem Input und Output neue Gewichtungen für das Neuronale Netz berechnet. Im Folgenden werden beide Algorithmen näher erläutert.

2.5.1 Feedforward Algorithmus

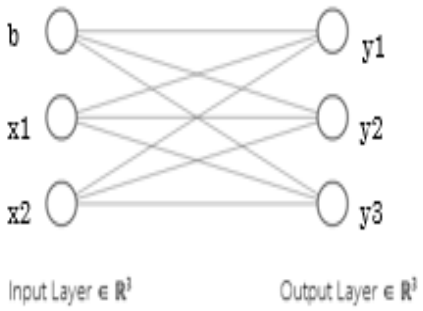
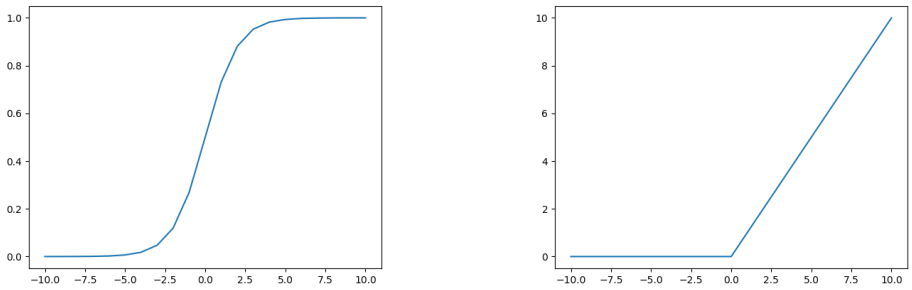


Abbildung 7: Konzept des Feedforward Algorithmus

Abb. X zeigt ein vereinfachtes Neuronales Netzt mit zwei Schichten. Das Input layer nimmt zwei Eingangswerte entgegen und verrechnet diese per Matrixmultiplikation mit den Gewichtungen, wie in Gleichung X dargestellt. Nach dieser Matrixmultiplikation entsteht ein neuer Vektor mit der gleichen Dimension wie der Output layer. Zu diesem Vektor wird noch der sog. ”Biasäddiert. Der Bias dient als feste Skalierung dazu, ein Neuron gezielt mehr, bzw. weniger Aktiv zu machen, das bedeutet mathematisch, dass der numerische Wert gezielt gesteigert bzw. verringert wird. Nachdem der Input mit den Gewichtungen und dem Bias verrechnet wurde, wird auf das Ergebnis eine Aktivierungsfunktion angewendet. Eine Aktivierungsfunktion dient dazu zu bestimmen, wie äktivëin Neuron ist. Es existieren mehrere Aktivierungsfunktionen. So zeigen Abbildungen 4 und 5 zwei Mögliche Funktionen.

$$A\left(\begin{bmatrix} w_{10} w_{11} \\ w_{20} w_{21} \\ w_{30} w_{31} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{3}$$



(a) Sigmoid (b) Rectified Linear Unit

Abbildung 8: zwei mögliche Aktivierungsfunktionen

Die Formel für die Sigmoid Funktion:

$$s(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

In Abb. 6 sind zwei existierende Aktivierungsfunktionen zu sehen, der Sigmoid die ReLu Funktion. Die ReLu Funktion gehört zu den beliebtesten Aktivierungsfunktionen, da diese Linearität gewährleistet. Wenn der Eingang der Funktion kleiner oder gleich 0 ist, dann ist der Ausgang immer 0. Für Werte größer als 0, ist der Output immer die Identitätsfunktion. Diese Funktion hat den Vorteil, dass sie negative Werte ignoriert, und positive Werte unverändert durchlässt. Des Weiteren ist zu erkennen, dass diese Funktion an der Stelle x = 0 nicht differenzierbar ist, da dort ein Knick vorzufinden ist.

Neben der ReLu Funktion ist aber auch der Sigmoid weitläufig bekannt. Dieser unterscheidet sich von der ReLu Funktion dadurch, dass diese nichtlinear (siehe Gleichung 3 ist und sowohl negative als auch positive Ausgangswerte erlaubt. Im allgemeinen ist die ReLu Funktion bevorzugt zu verwenden, da diese Linearität gewährleistet

$$y = \begin{cases} 0 & \text{if } x <= 0 \\ x & \text{if } x > 0 \end{cases}$$

2.5.2 Backpropagation Algorithmus

Der zweite, wichtig Algorithmus für ein Neuronales Netz ist der Backpropagation Algorithmus. Dieser sorgt dafür, dass die vorher erwähnten Gewichtungen an die Input-Output Datenpaare angepasst werden, sodass später korrekte Vorhersagen allein durch die Input Daten gemacht werden können. Dies funktioniert kon-

zeptionell so, dass der Fehler, welcher zwischen den und den berechneten Ausgangsdaten entsteht, von den Output Knoten wird, so dass jeden davorgehenden Knoten ebenfalls ein Fehler berechnet werden kann. Für diesen Algorithmus muss eine neue Funktion eingeführt werden: Die Verlustfunktion. Die Verlustfunktion, wie sie als Gleichung x dargestellt ist, ist eine Funktion in Abhängigkeit der Gewichtungen und sie berechnet den Fehler von den vorhergesagten Daten und den tatsächlichen Daten. Es existieren mehrere Möglichkeiten den Fehler zu berechnen, die Cross Entropy Methode ist jedoch mehr oder weniger standard. In der Gleichung wird zwei mal summiert. die erste Summe berechnet die Fehler in den einzelnen Output Neuronen. So ist K z . B. gleich 3, falls 3 Output Neuronen existieren. In diesem Fall wird für jedes Neuron der vorhergesagte Wert und der tatsächliche Wert verglichen. Die Zweite Summe summiert die berechnete Summe des Neuronen vergleichs über alle Datensätze, die in das Neuronale Netz eingespeist wurden. Darauffolgend wird das Ergebnis durch die Anzahl der Datensätze geteilt, um den durchschnittlichen Verlust zu berechnen

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^(i) log((h_\theta(x^i))_k) + (1 - y_k^i) log(1 - (h_\theta(x^i))_k \tag{5}$$

Dadurch, dass J eine Funktion in Abhängigkeit der Parameter Theta ist, besteht die Möglichkeit den Gradient Descent Algorithmus zu verwenden, um die Parameter auf die Eingangs- und Ausgangsdaten hin zu optimieren. In Abb. 7 ist eine mögliche Kostenfunktion in Abhängigkeit von 2 Parametern Theta eins und Theta zwei zu sehen. Das Ziel des Gradient Descent Algorithmus ist es sich, schrittweise, innerhalb mehrerer Iterationen, durch der Parameter an das absolute Minimum der Kostenfunktion an Dies bedeutet, dass die Partielle Ableitung der Funktion J in Abhängigkeit der einzelnen Parameter Theta durchgeführt werden muss:

$$J(\theta)' = \frac{\partial J}{\partial \Theta} \tag{6}$$

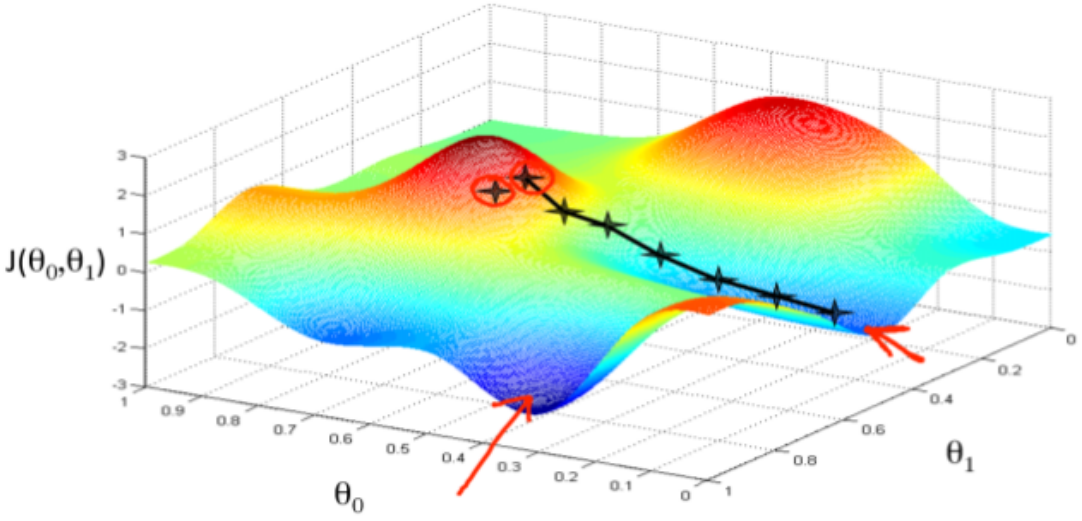


Abbildung 9: Beispiel des Gradient Descent Algorithmus an einer beliebigen Funktion

2.6 Verwendeten Platformen, Programmiersprachen und Bibliotheken

Zur Datenerhebung per I²C wurde zwei weitgehend unterschiedliche Platformen verwendet. Zum einen wurde ein PICkit Serial Analyzer genutzt, um die Daten von einem Windows Rechner per Programmiersprache C zu erheben. Dadurch, dass dieser Ansatz jedoch nur ein geringe Datenrate pro Sekunde erzielte, wurde stattdessen auf einen Raspberry Pi 4 gewechselt, welcher einer I²C Schnittstelle besitzt. Des Weiteren wurde für die Datenerfassung- und Verarbeitung die Sprache Python gewählt, da diese eine hohe Flexibilität bezüglich der Nutzbaren Bibliotheken mit sich bring. Das trainieren des Neuronalen Netzes erfolgt auf einem Windows Rechner mit der Maschine Learning Platoform Tensorflow, wobei Keras als High-Level API dient. Die graphische Visualisierung der Daten erfolgt ausschließlich per Pyplot library.

3 Methodik

3.1 Datenerfassung

Dieser Abschnitt beinhaltet verschiedene Ansätze, wie die Daten vom Wandler abgerufen werden können. Dies beinhaltet die Verwendung eines PICkit Serial Analysers sowie die eines Raspberry Pi.

3.1.1 Notwendigkeit eines Pegelwandlers

Dadurch bedingt, dass der verwendete Wandler einen Pegel von fünf Volt am I²C Bus besitzt, und ein Raspberry Pi jedoch nur 3.3 Volt, ist es notwendig einen Pegelwandler zwischen die beiden Platinen zu schalten, sodass die Möglichkeit von Hardware defekten ausgeschlossen werden kann. Der dabei verwendete Pegelwandler hat die Bezeichnung TXS0108E. Die Schaltung des Aufbaus ist in Abb. XX dargestellt.

3.1.2 PICkit Serial Analyzer

Das Aufzeichnen der Wandler-Daten kann innerhalb zweier Plattformen durchgeführt werden. Eine Plattform ist hierbei ein PICkit Serial Analyzer, der es ermöglicht, auf einem Windows Betriebssystem unter der Verwendung der Programmiersprache C Das Protokoll zu betreiben. Der genannte Serial Analyzer ist in Abb. X zu erkennen. Dieser besitzt, wie in der Markierung 6 zu sehen ist mehrere Anschlüsse, welche für verschiedene Protokolle wie I²C, UART und SPI verwendet werden können. Der Quellcode, welcher zur Datenerfassung per PICkit verwendet wurde, ist im Anhang zu finden. Der Ablauf des Programms beginnt mit der Definierung verschiedener Funktionen um aus Byte-Daten die tatsächlichen Werte zu berechnen. In der Main Funktion werden zu beginn alle relevante Programmparameter definiert und initialisiert. Dies schließt Adressen, Datenarrays sowie Konfigurationsparameter des PICkit mit ein. Anschließend erfolgt die Initialisierung des PICkit mit den vorher definierten Parametern. Abgeschlossen wird das Programm durch einen superloop, welche kontinuierlich Daten vom Wandler ausliest, auswertet und in eine Datei abspeichert. Evtl. noch die Codezeilen mit angeben

Legend:

1 – Status LEDs	3 – Lanyard Connection	5 – Pin 1 Marker
2 – Push Button	4 – USB Port Connection	6 – Communications Connector

Abbildung 10: Strukturierung der gemessenen Daten

3.1.3 Raspberry Pi

Die zweite Plattform, welche in den Folgenden Tests primär genutzt wird, ist ein Raspberry Pi. Dadurch, dass einem Raspberry Pi sowohl I²C Pins als auch entsprechende Software Libaries zur Verfügung stehen, ist es möglich auch darauf das I²C Protokoll zu nutzen.

Es ist ebenfalls wichtig erwähnen, dass die Struktur, in der die Daten innerhalb der Software abgefragt werden relevant ist, da diese sowohl die Abtastrate, als auch die Qualität der Daten beeinflussen kann. So zeigt die Variante 1 des Codes im Anhang eine mögliche Struktur. In dieser werden die einzelnen Parameter separat nacheinander in 2 Byte Blöcken abgefragt. Dies hat den Vorteil, das die Daten spezifisch abgefragt werden . Die Nachteile dieses Verfahrens sind jedoch, ein zusätzlicher Overhead, da durch mehrmaliges Abfragen jedes mal die Adresse des Mikrocontrollers mit angegeben werden muss, was den Datenfluss verlangsamt. Das bedeutet, dass die Adressbits, welche in Abb. 3 dargestellt sind, mit jeder Abfrage mitgeschickt werden müssen. Ein weiterer Nachteil ist, dass die Daten nicht vom selben Zeitabschnitt stammen. So werden beispielsweise, während die Daten aus dem Temperatur Register abgefragt werden, bereits die Daten in den Spannungs- und Stromregistern aktualisiert. ie Abfragestruktur, welche zu bevorzugen ist, ist im Anhang als Variante 2 dargestellt. Diese zeigt, dass alle 8 Bytes des Datenregisters auf einmal ausgelesen werden. Dies hat den Vorteil, dass alle Daten zur gleichen Zeiteinheit abgefragt werden und verhindert

zugleich den o. g. Effekt des Overhead, da alle vier Datenregister mit einmaligem versenden der Adresse abgerufen werden.

3.2 Speicherung und Strukturierung der Daten

Die Speicherung der erfassten Daten erfolgt während der Programmlaufzeit. Dieser Umstand bietet sowohl Vor- als auch Nachteile. Ein Nachteil dieser Strategie ist, dass das speichern der Daten nach jeder Abfrage Rechenzeit kostet, was die Abtastrate des Systems verlangsamt. Ein Vorteil dieser Methodik ist jedoch, dass Daten sofort gespeichert werden, und somit nicht die Möglichkeit besteht, dass die Daten bei einem Programmabsturz verloren gehen. Ein weiterer Vorteil besteht darin, dass die Daten nicht im RAM des Raspberry Pi gespeichert werden müssen, da sie mit jeder neuen Datenabfrage auf der Festplatte gespeichert werden, und die Daten im RAM sofort von neuen Daten überschrieben werden. Zusammengefasst kann gesagt werden, dass eine geringe Beeinträchtigung der Abtastrate in kauf genommen wird, für eine geringere Auslastung des RAM und mehr Sicherheit bei der Datenerfassung.

In Abb. x ist der Ausschnitt eines Datensatzes zu erkennen. Die Spalten repräsentieren jeweils eine Kategorie von Daten, wie z. B. die Temperatur. Die einzelnen Zeilen beinhalten jeweils einen kompletten Datensatz, welcher in der Zeit aufgenommen wurde, der im Zeitstempel eingetragen ist. Der Zeitstempel hat das Format Stunden:Minuten:Sekunden:Millisekunden.

[Zeit]	[Temp.]	[I_OUT]	[V_OUT]	[V_IN]	[POWER_OUT]
14:45:54:808062	49.59935897435897	0.06510416666666667	12.041015625	23.486328125	0.7839202880859376
14:45:54:816166	49.59935897435897	0.06510416666666667	11.982421875	23.486328125	0.7801055908203125
14:45:54:821332	49.84975961538461	0.048828125	12.041015625	23.486328125	0.5879402160644531
14:45:54:828651	49.59935897435897	0.06510416666666667	12.041015625	23.61328125	0.7839202880859376
14:45:54:835517	49.59935897435897	0.06510416666666667	11.982421875	23.5498046875	0.7801055908203125
14:45:54:841584	49.84975961538461	0.06510416666666667	12.01171875	23.486328125	0.7820129394531251

Abbildung 11: Strukturierung der gemessenen Daten

3.3 Methoden der Datenverarbeitung

In diesem Abschnitt werden die einzelnen Methoden der Datenverarbeitung innerhalb der Software dargestellt. Dies beinhaltet das Laden der Daten, das preprocessing sowie die Visualisierung. Das Laden der Daten erfolgt in der Funktion ünr_data", welche im Anhang in der Sektion x gefunden innerhalb der Zeilen 70 und 117 gefunden werden kann. In dieser Funktion werden die Daten, welche in einer Textdatei gespeichert sind, in eine Matrix gespeichert, welche durch die Library "Numpy"generiert wird. Nachdem die Daten geladen wurden, muss im nächsten Schritt der Zeitverlauf aus den Zeitstempeln generiert werden. Dies ist

deshalb notwendig, da die Daten nicht mit dem Zeitformat ”%H:%M:%S:%f”visualisiert werden Aus diesen Zeitstempel werden Zeitdaten in Sekunden generiert. Sobald die Zeitstempel formatiert wurden, gibt die Funktion in Zeile 117 eine Matrix mit den gemessenen Daten und ein Array mit dem Zeitverlauf aus. Was ebenfalls zu erwähnen ist, ist die For-Schleife am Ende der unr_data Funktion. Diese Schleife dient dazu Fehler in einem rudimentären Ausmaß zu erkennen. Wie in dem Code zu erkennen ist, werden Datenwerte, welche kleiner als null und größer als Hundert sind als Fehler gehandhabt und können entweder nur zu einem Fehlerzähler hinzu-addiert werden oder direkt dadurch kompensiert werden, indem anstelle des Fehlerhaften Werts der vorher kommende Wert verwendet wird.

Dadurch bedingt, dass alle Signale rauschen, existiert die Möglichkeit einen Software-gesteuerten Tiefpassfilter über das Signal zu legen. Diese Funktion ist zwischen den Zeilen 120 und 125 in der Funktion äpply_filterßu erkennen. Diese Funktion kann über die Library ßcipy”genutzt werden.

Eine weitere wichtige Funktion ist ”calc_meta”. Diese Funktion dient dazu Metadaten des vorhandenen Datensatzes zu berechnen. Zu den Metadaten gehören: Mittelwert, Länge des Datensatzes, Standardabweichung, numerische Minimum- und Maximum-Werte. Dadurch, dass als Input-Parameter die gesamte Datenmatrix übergeben wird (data), ist die Ausgabe entsprechend eine Matrix in der zu jedem Datensatz (Strom, Spannung, Temperatur) ein Array mit den Metadaten gehört.

Alle bisher genannten Funktionen können genutzt werden, um die Daten abzurufen und zu visualisieren, wie in den Zeilen 158 bis 168 aufgezeigt ist. Es werden Daten aus jeder Datei in einem Verzeichnis geladen, Metadaten berechnet sowie ein bestimmtes Signal visualisiert werden.

3.4 Bestimmung der Abtastrate

Die Abtastrate des Systems ist durch mehrere Faktoren begrenzt.Der

Die statische Datenerfassung dient dem Zweck, die Funktionalität der Wandler unter einer konstanten Last zu überprüfen. Es wurden zwei simple Testfälle aufgebaut. Bei beiden handelt es sich um Parallelschaltungen von 4,7 Ohm Widerständen. Der erste Fall schaltet zwei davon parallel, der zweite drei. Durch das Anwenden des Ohmschen Gesetzes, kann ein theoretischer Stromfluss und somit auch die theoretische Leistungsaufnahme berechnet werden. Der Wandler wird durch ein Netzteil mit 24 Volt Ausgangsspannung betrieben. Diese 24 Volt werden von dem Wandler auf 12 V herabgesetzt. Somit kann der Strom berechnet werden, der durch die Widerstände fließt. Bei zwei parallel-geschaltenen Widerständen ergibt das einen theoretischen Stromfluss von 5.1 A wie und bei

drei parallel geschaltenen Widerständen 7.65 A wie in den Gleichungen 3 und 4 zu sehen ist.

$$\frac{U}{R} = I \tag{7}$$

$$\frac{12V}{4.7} * 2 = 5.10A \tag{8}$$

$$12V * 4.7Ohm * 3 = 7.65A \tag{9}$$

Da die statische Datenerfassung über eine konstante Last durchgeführt wird, sind hohe Abtastraten nicht zwingend erforderlich. Aus diesem Grund ist die Nutzung eines PICkit, welcher eine geringe Abtastrate besitzt nicht problematisch. Es wird jedoch trotzdem ein Raspberry Pi zur Datenerfassung verwendet, da dieser durch seine höhere Abtastrate eine feinere Ansicht der Daten ermöglicht, wodurch das Rauschen der Daten besser dargestellt werden kann.

3.6 Dynamische Datenerfassung

Das Ziel der dynamischen Datenerfassung ist es, zu evaluieren, wie der Wandler sich unter Lasten verhält, die sich mit der Zeit ändern. Sowie mögliche Abweichungen zwischen den einzelnen Wandlern selbst. Der Testaufbau beinhaltet den Hybridwandler, einen Stereoverstärker, welcher per Wandler mit Strom versorgt wird und in den ein Audiosignal eingespeist wird. In Abb. 5 ist die Schaltung des Testaufbaus abgebildet. Es ist zu erkennen, das am Eingang des Buck-Wandlers 24 V eingespeist werden und das die am Ausgang liegenden 12 V an den Stereoverstärker angeschlossen sind. Des Weiteren ist ersichtlich, dass nur ein Kanal des Stereoverstärkers genutzt wird. Dieser Kanal wird mit einem Audiosignal aus einem externen Rechner versorgt. Als Signale werden hierbei Sinussignale mit variierender Frequenz verwendet, welche mit dem Programm Audacity generiert werden. Das Ausgangssignal fällt über einen Lastwiderstand ab. Obwohl es ein Stereoverstärker ist, wurde auf einen Lautsprecher verzichtet, da durch einen Lastwiderstand die Reproduzierbarkeit gewährleistet wird.

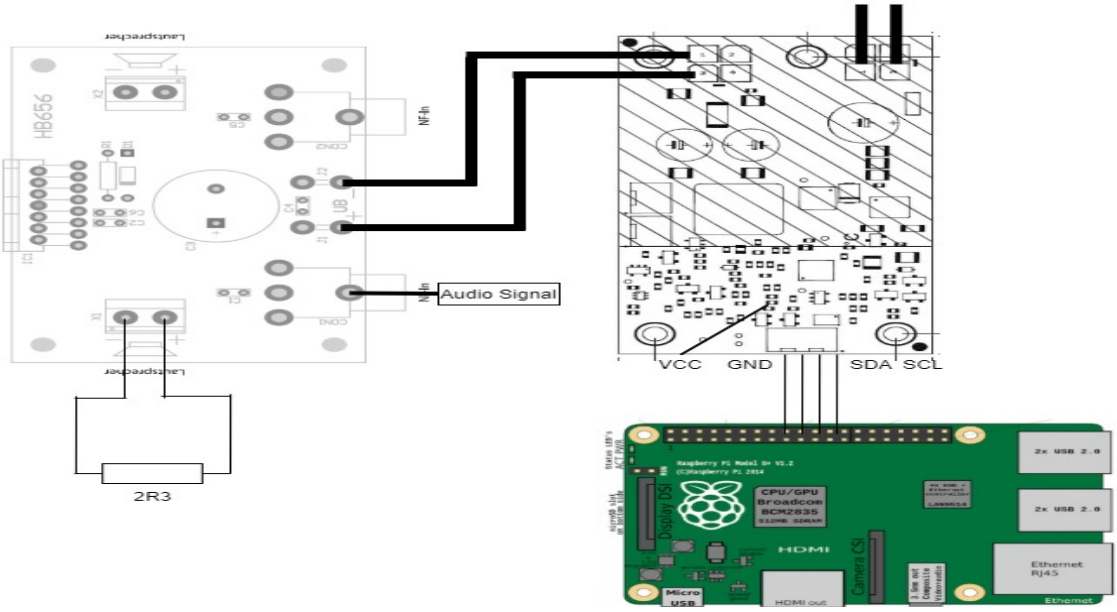


Abbildung 12: Versuchsaufbau der dynamischen Datenerfassung

Der erste Testschritt ist dabei die Überprüfung, wie hoch die maximale Frequenz eines Signals sein kann, sodass interpretierbare Daten erzeugt werden können. Die maximale Frequenz kann durch praktische Tests empirisch bestimmt werden. Der Umfang dieser Tests besteht darin, die Daten von mehreren Wandlern über einen Zeitraum T zu messen und anschließend die Anzahl der gemessenen Samples durch die gemessene Zeit zu teilen, um so die durchschnittliche Sample-Rate pro Sekunde zu erhalten.

Dadurch, dass die Datenerfassung auf einem Raspberry Pi abläuft, ist es möglich, die Daten nach jeder Abfrage sofort in eine Textdatei abzuspeichern. Es werden Daten zur Zeit, Temperatur, Ausgangsstrom, Ausgangsspannung und Eingangsspannung aufgezeichnet und tabellarisch abgespeichert.

Die Nyquist-Frequenz stammt aus der Signaltheorie und setzt die maximale Abtastrate in Verhältnis mit der maximalen Frequenz, die ohne Verzerrungen dargestellt werden kann. Die Gleichung besagt, dass die maximale Frequenz, die ohne Verzerrungen dargestellt werden soll, mit einer Abtastrate abgetastet werden soll, die mindestens doppelt so hoch ist, wie die abgetastete Frequenz. Sobald die Abtastrate experimentell bestimmt werden kann, kann das Nyquist-Frequenz theorem darauf angewendet werden, um so die Frequenz zu bestimmen, die maximal mit dem Wandler abgetastet werden kann.

3.6.1 Echtzeitmonitoring

Das Ziel des Echtzeitmonitoring ist es, in laufenden Betrieb des Wandlers Daten zu Strom, Spannung und Temperatur zu erheben, diese Daten zur Verarbeiten

und Aussagen darüber zu treffen, welche Last gerade an dem Wandler angeschlossen ist. Eine Last ist in diesem Fall ein Audiosignal, welches durch einen Stereo Verstärker, welcher an den Wandler angeschlossen ist, verstärkt wird. Zur Klassifizierung, welche Last, bzw. welches Signal gerade am Stereoverstärker angeschlossen ist, wird ein neuronales Netz verwendet, welches per Tensorflow API generiert wird. Die gemessenen Signale werden dann vom Raspberry Pi, zwecks Verminderung der Rechenleistung, per Socket an einen Rechner geschickt, welcher dann die Daten per Neuronales Netz interpretiert und eine entsprechende Klassifizierung berechnet. Als Testfälle wurden Sinussignal mit verschiedenen Frequenzen verwendet, um die allgemeine Anwendbarkeit von Deep-Learning Algorithmen zu verifizieren.

3.6.2 Vergleichsmonitoring mehrerer Wandler im laufenden Betrieb

Das Ziel des Vergleichsmonitoring ist es, eine Software Struktur aufzubauen, welche es ermöglicht, mehrere Wandler gleichzeitig auf einem Raspberry Pi per I²C zu betreiben und die Daten in Echtzeit zu visualisieren und in Textdateien abzuspeichern.

wie in Abb. X zu sehen ist, werden für diesen Testaufbau zwei unterschiedliche Wandler parallel zu einem Netzteil geschaltet, welches 24 V Versorgungsspannung liefert. Die jeweiligen Ausgänge der Wandler werden dann an einen Lastwiderstand angeschlossen. Des Weiteren werden die Ground, SDA und SCL Verbindungen beider Wandler verbunden.

Dadurch, dass die verwendeten Wandler unterschiedlich Adressen besitzen, müssen keine Ergänzungen an der Schaltung durchgeführt werden, es reicht einzig die Datenleitungen (SDA, SCL) zusammenzuschalten. Während der Messung der Wandler, werden die Daten von jedem Messpunkt in Echtzeit auf ein Koordinatensystem per PyPlot übertragen.

3.7 Anwendbarkeitstest - Neuronales Netz

Das Ziel des Anwenbarkeitstest ist es zu evaluieren, ob und in welchem Ausmaß sich die vom Wandler generierten Daten sich für Anwendungsfälle im Bereich des Maschine Learning, im speziellen dem der Neuronalen Netze, eignen. Hierfür wird ein simpler Testfall generiert, in dem die Daten mehrerer Signale mit verschiedenen Frequenzen als Input-Daten dienen und der Output entsprechend eine Klassifizierung der Signalfrequenz herausgeben soll.

Als Input-Daten wird der zeitliche Verläufe der Leistung eingespeist. wie in Abb. x zu sehen ist, bedeutet dass, das t0 - t50 Samples einen Datensatz ver-

wendet werden. Es ist deshalb bedeutsam die Daten so zu strukturieren, da damit wird, dass das Neuronale Netz die Daten interpretieren kann, da es sich um zeitlich aufeinanderfolgende Datensamples handelt. In einem Negativbeispiel, in dem nur 1 einziges Sample genommen werden ein NN Probleme damit dieses einzige Sample zu Klassifizieren, da es Teil von verschiedenen Signalen sein kann

Als Netzwerk Architektur wird ein klassisches vollvermaschtes Neuronales Netz mit zwei Hidden Layern verwendet. Die Anzahl der Neuronen in den Hidden layern muss jedoch praktisch bestimmt werden, damit die besten Resultate erzielt werden können.

Die entsprechenden Codezeilen sind im Anhang zu zwischen den Zeilen 1 und 50 zu finden. Darin ist zu erkennen, dass die Trainingsdaten zuerst zufällig gemischt werden, um darauffolgend Normalisiert zu werden. Die in dem Code verwendete Normalisierung ist die ”min-max-Normalisierung.Änschließend wird das Model des Neuronalen Netzes initialisiert. Sobald das Model erstellt wurde, wird es per Befehl ”model.compile”mit den gegebenen Trainingsdaten trainiert.

Für diese Art von Test ist es unabkömmlich das Nyquist-Frequenz Theorem anzuwenden. Es ist deshalb so wichtig, da die Daten, die in eine Neuronales Netz eingespeist werden genau einer Kategorie zugewiesen können müssen. Das bedeutet, dass Falls ein Signal mit einer zu niedrigen Abtastrate digitalisiert wird, und sich somit Verzerrungen ergeben, ist nicht mehr gewährleistet, dass die Klassifizierung der ursprünglichen Signalfrequenz erreicht werden kann. Aus diesem Grund müssen für den Anwendbarkeitstest nur Signale mit einer Frequenz verwendet werden, welche nicht durch den Alias-Effekt verzerrt werden.

4 Ergebnisse

4.1 Ergebnisse der statischen Tests

Im Folgenden werden die Ergebnisse der statischen Tests mit Bezug auf Temperatur- und Leistungsverlauf mehrerer Wandler dargestellt und näher erläutert. Der Versuchsaufbau und die Testumgebung waren bei allen drei Wandlern nahezu identisch. In Abb. 10 ist der Temperaturverlauf [in ℃] von drei unterschiedlichen Wandlern über einen Zeitverlauf von 3600 Sekunden aufgetragen. Es ist zu erkennen, dass der Wandler mit der Bezeichnung 'Board_10' eine wesentlich höhere Einschwingtemperatur besitzt als die anderen zwei. Dies lässt sich darauf zurückführen, dass besagter Wandler einen Hardware defekt besitzt. Des Weiteren ist zu erkennen, dass die beiden verbleibenden Wandler trotz identischer Hardware und gleichen Testbedingungen unterschiedliche Temperaturen vorweisen. Somit kann behauptet werden, das unter gleichen Bedingungen Abweichende Resultate herauskommen

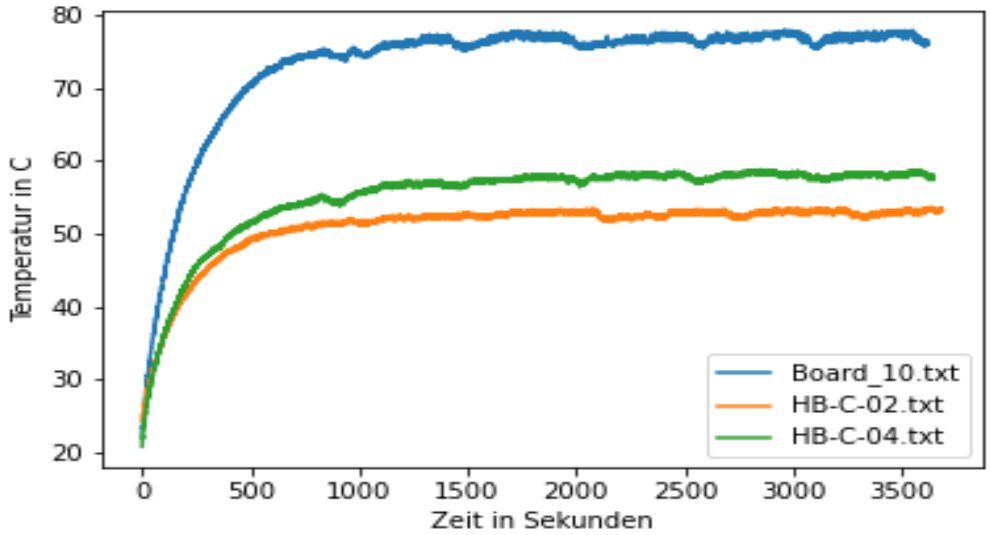


Abbildung 13: Temperaturvergleich mehrerer Wandler

In Abb. x ist der Leistungsverlauf [in Watt] der drei Wandler geplotet gegen die [Zeit in Sekunden] zu sehen. Aus diesem Graphen ist sofort ersichtlich, dass alle Signale rauschen. Wie man Tabelle X entnehmen kann, beträgt die Standardabweichung vom Mittelwert bei allen Signalen Un 300 Milliwatt. Die theoretische Leistung, welche bei einer Spannung von 12 V und zwei Parallel geschalteten Widerständen (siehe 3.1) generiert werden müsste, beträgt 61.2 Watt:

$$[H]12V * 5.1A = 61.2W \tag{10}$$

Somit liegt der Wandler mit der Bezeichnung ”HB-C-02äm nächsten zum theoretisch berechneten Wert. Diese praktische Abweichung kann durch einen Leistungsabfall in den Leitungskabeln und Ungenauigkeiten in der Hardware zumindest teilweise er werden.
Aufgrund dieser Resultate ist es für Maschine Learning Applikationen sinnvoll auf diese Daten einen Frequenzfilter anzuwenden, um das Rauschen zu eliminieren, da ansonsten Inkonsistenzen bei den K.I Applikationen vorkommen können, da die Daten unter gleichen Bedingungen variieren.

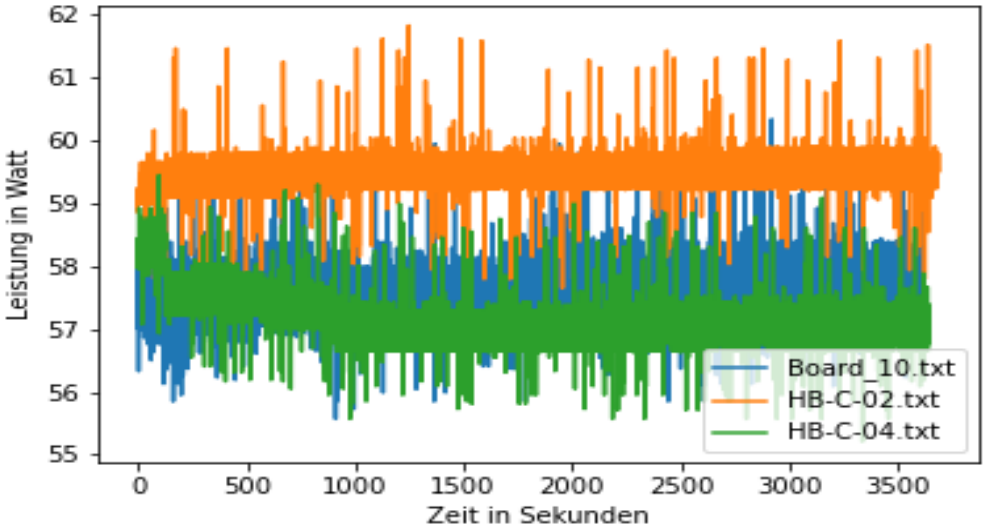


Abbildung 14: Leistungsvergleich mehrerer Wandler

In Abb. x ist der Leistungsverlauf zu erkennen, auf den ein Tiefpassfilter angewendet wurde. Bei dem Tiefpassfilter handelt es sich um Blablabla details. Dieser Datensatz besitzt nun kein Rauschen mehr, die einzelnen Signale sind jedoch trotzdem noch unterscheidbar.

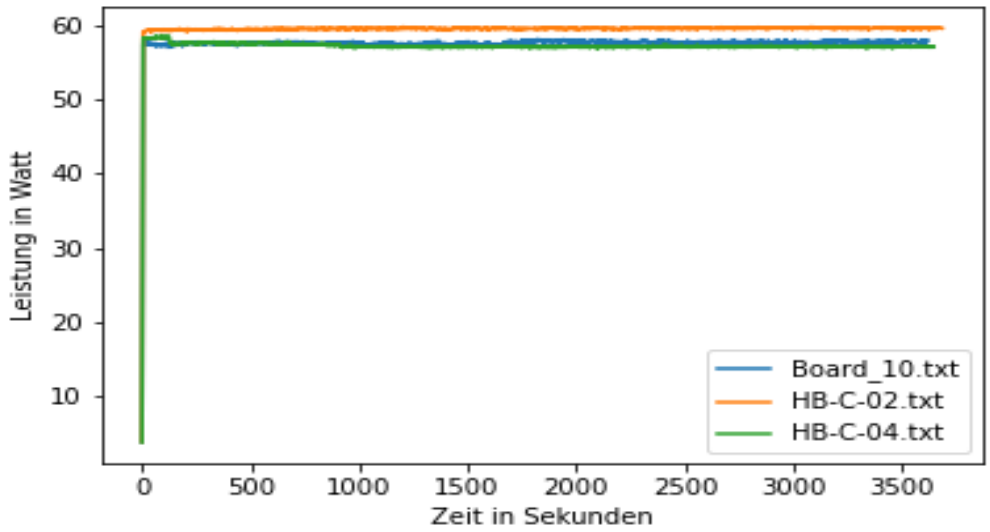


Abbildung 15: Gefilterter Leistungsverlauf der drei Wandler

4.2 Ergebnisse der dynamischen Tests

Zur dynamischen Datenerfassung ist zu sagen, dass das I²C Protokoll sich limitierend darauf ausgewirkt hat. Dadurch, dass stabile Programmausführung nur bei einer I²C Taktrate von 70 KHz gewährleistet war, war die Konsequenz daraus, dass die Abtastrate ebenfalls reduziert wurde. Die Ermittlung der Abtastrate erfolgte durch Messungen mehrerer Wandler, die mit verschiedenen Signalen gespeist wurden.Wie in Abb. 13 zu erkennen ist, wurden zwei verschiedene Wandler mit jeweils 6 verschiedenen Signalen verwendet. Zur Bestimmung der jeweiligen Abtastrate wurde zu durch Zeitstempel das Zeitdelta bestimmt, d. h. der Zeitraum in dem die Messungen durchgeführt wurden. Anschließend wurde die Anzahl der aufgenommenen Samples durch das Zeitdelta geteilt, um so auf die durchschnittliche Abtastrate zu kommen. Darauffolgend wurde der Mittelwert von allen Wandler-Signal Paaren berechnet, welcher 110 Hz beträgt. Wenn auf dieses Ergebnis die oben genannten Konzepte der Nyquist Frequenz angewendet werden, so zeigt sich, dass die maximale Signalfrequenz, welche ohne Verzerrungen durch den Wandler abgetastet werden kann, bei 55 Hz liegt. Aus diesem Grund wurden im nachfolgenden Test mit einem Neuronalen Netz nur Signalfrequenzen verwendet, welche einen geringere Frequenz als 55 Hz besitzen.

$Abtastrate$	$Datensatz$
334	$HB\_C\_02\_200Hz$
333	$HB\_C\_02\_20Hz$
339	$HB\_C\_02\_30Hz$
335	$HB\_C\_02\_5Hz$
336	$HB\_C\_02\_Cash$
333	$HB\_C\_02\_Lose$
335	$HB\_C\_02\_SoS$
330	$HB\_C\_07\_200Hz$
340	$HB\_C\_07\_20Hz$
335	$HB\_C\_07\_30Hz$
332	$HB\_C\_07\_5Hz$
335	$HB\_C\_07\_Cash$
339	$HB\_C\_07\_Lose$
334	$HB\_C\_07\_SoS$
338	$HB\_C\_08\_200Hz$
336	$HB\_C\_08\_20Hz$
339	$HB\_C\_08\_30Hz$
334	$HB\_C\_08\_5Hz$
335	$HB\_C\_08\_Cash$
339	$HB\_C\_08\_Lose$
337	$HB\_C\_08\_SoS$
334	$HB\_C\_15\_200Hz$
335	$HB\_C\_15\_20Hz$
332	$HB\_C\_15\_30Hz$
337	$HB\_C\_15\_5Hz$
335	$HB\_C\_15\_Cash$
335	$HB\_C\_15\_Lose$
332	$HB\_C\_15\_SoS$
335	$Durchschnitt$

In Abb. 4 ist das generierte Signal zu sehen, welches in den Verstärker eingespeist wird. Dabei handelt es sich in diesem Beispiel um ein 10 Hz Sinussignal, welches in Audacity generiert wurde und als Signalinput für den Stereo-Verstärker dient. Dieses Signal kann mit folgender Formel beschrieben werden:

$$f(x) = sin(2\pi 10x) \tag{11}$$

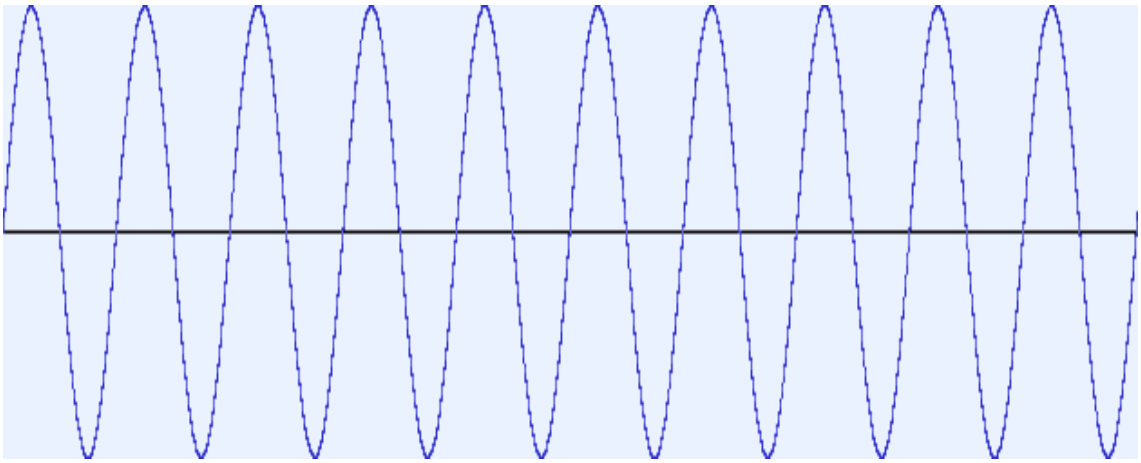


Abbildung 16: In Audacity generiertes 10 Hz Sinussignal

Dadurch, dass der Verstärker sowohl positive als auch negative Signale verstärkt, ergibt sich für die reine Leistungsaufnahme des Wandlers eine theoretische Leistungskurve wie sie in Abb. 11 dargestellt ist, nämlich der Betrag der o. g. Sinusfunktion:

$$|f(x)| = |sin(2 * pi * x * 10)| \tag{12}$$

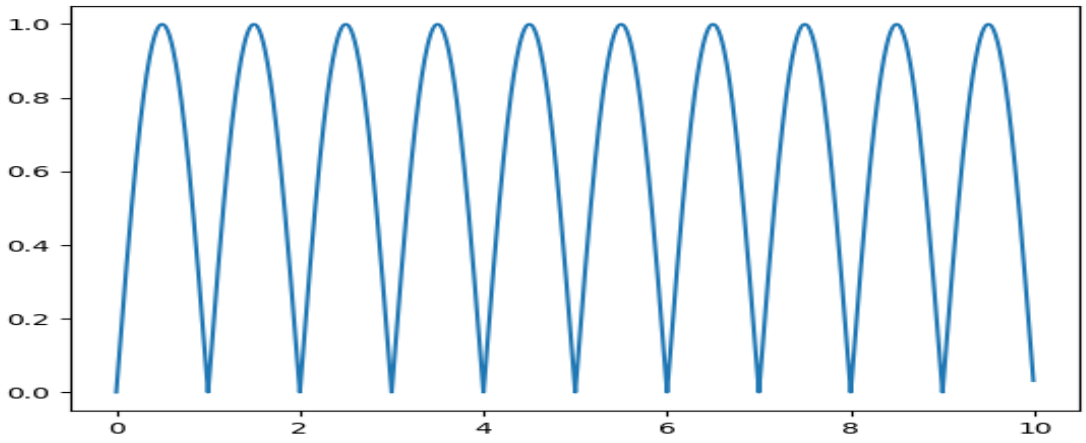


Abbildung 17: Effektive Frequenz eines 10 Hz Signals ist 20 Hz, weil, weil es für den Verstärker egal ist ob + oder - Signal

Dadurch, dass es vorher ein Sinussignal mit einer Frequenz von 10 Hz war, ist es nun der Absolutwert des selben Sinussignals mit einer effektiven Frequenz von 20 Hz.

Das Messen der Daten erfolgte auf einem Raspberry Pi per Protokoll und einer eingestellten Taktrate von 70 KHz, da diese Stabi Die Datenerhebung und Verarbeitung erfolgte durch die Programmiersprache Python.

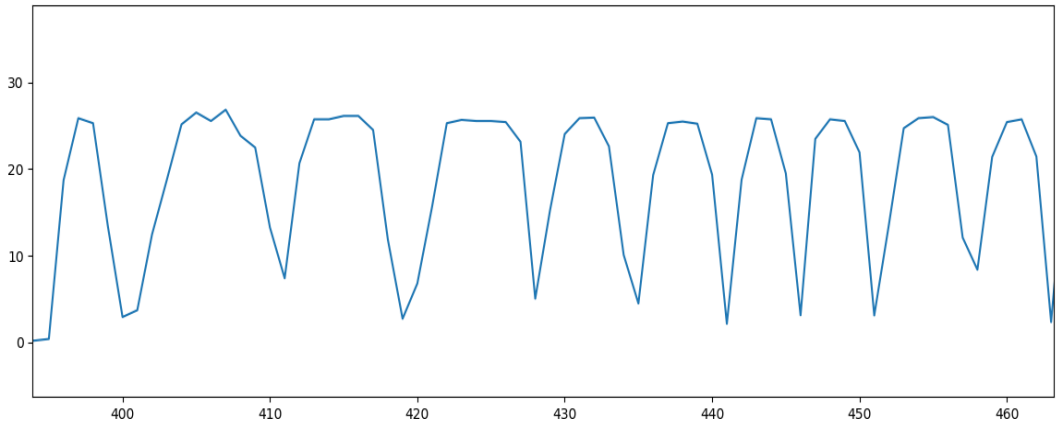


Abbildung 18: Tatsächlich gemessene Leistung des Wandlers

In Abb. 6 ist der ungefilterte Leistungsverlauf des Wandler dargestellt. Es ist ersichtlich, dass das umgeklappte 20 Hz Sinussignal in Abb. 5, dem gemessenen Sinussignal, entspricht. Der unsaubere Verlauf der Leistung ist auf Rauschen innerhalb der Hardware und auf die geringe Abtastrate von 110 Hz zurückzuführen.

4.3 Ergebnis des Anwendbarkeitstest - Neuronales Netz

Als Architektur des Neuronalen Netzes wurde ein Netz mit einem Input layer, zwei Hidden layern, sowie einem Output layer gewählt. Das Input layer hat eine Größe von 50 Neuronen, das bedeutet, dass in einem Datensatz 50 Samples enthalten sind. Die Anzahl der Hidden layer sowie deren Neuronen-Anzahl wurden durch empirische Tests bestimmt. Die Anzahl der Output Neuronen entspricht der Anzahl der möglichen Klassifizierungen, in diesem Fall 3 verschiedene Frequenzen: 10 Hz, 20 Hz, 30 Hz. Als Aktivierungsfunktion für alle Neuronen, außer die Output Neuronen, wurde ReLu ausgewählt. Der verwendete Optimierungsalgorithmus ist Gradient Descent mit Cross Entropy als Loss-Function.

Abb. 13 zeigt die Genauigkeit des Neuronalen Netzes in der Vorhersage der verschiedenen Frequenzen anhand der Trainingsdaten. Es ist aus dem Graphen ersichtlich, dass die Genauigkeit über 1000 Trainingsepochen 99% beträgt.

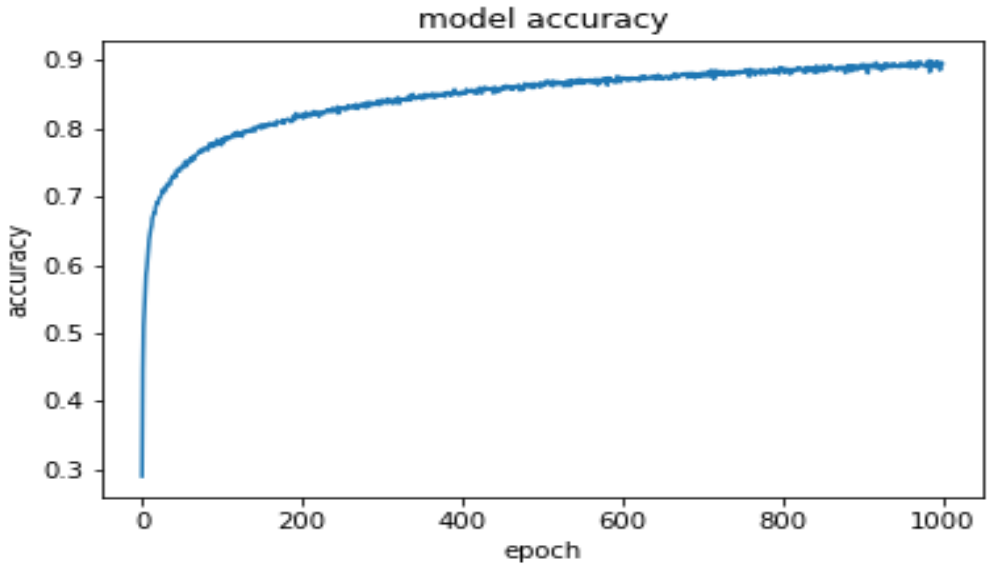


Abbildung 19: Genauigkeit der Anwendbarkeitstests

5 Fazit und Ausblick

Die Ergebnisse dieser Arbeit haben gezeigt, das alle verwendeten Wandler in ihrem Leistungssignal rauschen und auch untereinander unter gleichen Bedingungen leicht abweichende Ergebnisse hinsichtlich Strom, Spannung und Leistung liefern.

Des Weiteren konnte gezeigt werden, dass die Wandler eine durchschnittliche stabile Abtastrate von 110 Hz erreichen können, und dass die Daten die dabei generiert werden für Maschine Learning Applikationen verwendet werden können. Aus Basis dieser Erkenntnisse können nun weitere, komplexere Tests durchgeführt werden. Diese Tests können auf Signale mit komplexeren Verläufen und höheren Frequenzen angewendet werden. Ebenfalls können Praxisnähere Tests durchgeführt werden, das bedeutet, dass Hardwarekomponenten verwendet werden und mit dem Wandler mit Strom versorgt werden können, wie z. B. HDD, Raspberry Pi, etc.

6 Literaturverzeichnis

https://arxiv.org/abs/1710.05941 Quelle: https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o$_{f}ig1_{3}21259051https://www.fairaudio.de/lexikon/al$
$effekt-aliasing/https://www.microchip.com/Developmenttools/ProductDetails/DV164122$
$//www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-I2C-$
$Message-Frame-and-Bit-2.pnghttps://www.ti.com/lit/an/slva704/slva704.pdf?ts=$
$1614862288694ref_{u}rl = httpshttps://www.audacityteam.org/https://cdn-$
$reichelt.de/documents/datenblatt/A300/ADAFRUIT_{3}95_{E}NG_{T}DS.pdf$

7 Anhang

Code für PICKIT:

```
1  using System;
2  using System.IO;
3  using System.Text;
4  using System.Threading;
5  using System.Threading.Tasks;
6  using PICkitS;
7
8  namespace Bachelor
9  {
10     class Program
11     {
12         static double Calc_Temp(byte TEMP_HIGH, byte TEMP_LOW)
13         {
14             int New_High = Convert.ToInt32(TEMP_HIGH);
15             int New_Low = Convert.ToInt32(TEMP_LOW);
16             int New_High = Convert.ToInt16(TEMP_HIGH);
17             int New_Low = Convert.ToInt16(TEMP_LOW);
18             New_High = New_High << 8;
19             double Temp = New_High + New_Low;
20             Temp = ((Temp / 1024 * 5) - 0.4) / 0.0195;
21             return Temp;
22
23         }
24
25         static double Calc_I_Out(byte I_Out_High, byte I_Out_Low)
26         {
27             int New_I_Out_High = Convert.ToInt32(I_Out_High);
28             int New_I_Out_Low = Convert.ToInt32(I_Out_Low);
29             int New_I_Out_High = Convert.ToInt16(I_Out_High);
30             int New_I_Out_Low = Convert.ToInt16(I_Out_Low);
31
32             New_I_Out_High = New_I_Out_High << 8;
33             double Amps = New_I_Out_High + New_I_Out_Low;
34             Amps = Amps / 1024 * 5 / 0.3;
35             return Amps;
36
37
38         }
39
40         static double Calc_V_Out(byte V_Out_High, byte V_Out_Low)
41         {
42             int New_V_Out_High = Convert.ToInt32(V_Out_High);
43             int New_V_Out_Low = Convert.ToInt32(V_Out_Low);
```

```
44              int New_V_Out_High = Convert.ToInt16(V_Out_High);
45              int New_V_Out_Low = Convert.ToInt16(V_Out_Low);
46
47              New_V_Out_High = New_V_Out_High << 8;
48              double Volts = New_V_Out_High + New_V_Out_Low;
49              Volts = Volts / 1024 * 5 * 6;
50              return Volts;
51          }
52
53
54          static double Calc_V_In(byte V_In_High, byte V_In_Low)
55          {
56              int New_V_In_High = Convert.ToInt32(V_In_High);
57              int New_V_In_Low = Convert.ToInt32(V_In_Low);
58              int New_V_In_High = Convert.ToInt16(V_In_High);
59              int New_V_In_Low = Convert.ToInt16(V_In_Low);
60
61              New_V_In_High = New_V_In_High << 8;
62              double Volts = New_V_In_High + New_V_In_Low;
63              Volts = Volts / 1024 * 5 * 13;
64              return Volts;
65          }
66
67
68
69
70          static void Main(string[] args)
71          {
72              /*Slave Adress of the Device */
73              byte DEV_ADDR_WRITE = 0x10;
74
75
76              /*Number of Bytes that should be read*/
77              byte NUM_BYTES = 0x08;
78
79              /*Registers that I want to read*/
80              byte V_IN_ADDR = 0x20;
81              byte I_OUT_ADDR = 0x22;
82              byte V_OUT_ADDR = 0x24;
83              byte TEMP_ADDR = 0x26;
84
85              /*Array where the data gets saved*/
86              byte[] Data_Array = new byte[NUM_BYTES];
87              Data_Array[0] = 0x20;
88
89              string Script_View = "";
90
```

```
91              /*Set Clock speed to 100kHz */
92              int Set_Clock = 100;
93              double Voltage = 3.3;
94
95              /*Timeout after 2 seconds*/
96              int Set_TimeOut = 10000;
97
98              string now = DateTime.Now.ToString("h:mm:ss tt");
99              string path = @"C:\Users\lamer\Downloads\Test.txt";
100             /*Set Clock speed to 100kHz  and voltage to 3.3 V*/
101             int Set_Clock = 400;
102             double Voltage = 3.3;
103
104             /*Timeout after 10 seconds*/
105             int Set_TimeOut = 100;
106
107             string now = DateTime.Now.ToString("h:mm:ss tt");
108             string path = @"C:\Users\lamer\Downloads\
                    Board_2_Test_3_Parallel.txt";
109
110             byte High = 0x01;
111             byte Low = 0x54;
112             double x = Calc_Temp(High, Low);
113             Console.WriteLine(x);
114             int new_high = Convert.ToInt32(High) << 8;
115             int net = new_high + Convert.ToInt32(Low);
116             Console.WriteLine(new_high);
117             Console.WriteLine(Low);
118             Console.WriteLine(net);
119
120             /*Initialisiert den PICKIT, falls mehrere Analyzer
                    verwendet werden,
121             dann die Funktion  Initialize_PICkitSerial mit dem USB
                    Index verwenden
122             Hier ist der USB-Index == 0*/
123             if (Device.Initialize_PICkitSerial(0))
124             {
125                 Console.WriteLine("Device Initialization successful");
126             }
127             else
128             {
129                 Console.WriteLine("Device Initialization failed \n");
130                 return;
131             }
132             /*PicKit f r I2C Master konfigurieren */
133             if (I2CM.Configure_PICkitSerial_For_I2CMaster())
134             {
```

```
135                 Console.WriteLine("I2C Master Initialization successful
                        ");
136             }
137             else
138             {
139                 Console.WriteLine("I2C Master Initialization failed \n
                        ");
140                 return;
141             }
142
143             if (I2CM.Set_I2C_Bit_Rate(Set_Clock))
144             {
145                 Console.WriteLine("Clock Set successfully to: " +
                        Convert.ToString(Set_Clock));
146             }
147             else
148             {
149                 Console.WriteLine("Setting Clock failed \n");
150                 return;
151             }
152
153             I2CM.Set_Read_Wait_Time(Set_TimeOut);
154
155             Console.WriteLine("Timeout set to: " + Convert.ToString(
                    Set_TimeOut) + " milliseconds\n");
156
157             //Netzteil hat interne Pull-Ups, daher keine PicKit pullups
                    anmachen
158             if (I2CM.Set_Pullup_State(false))
159             {
160                 Console.WriteLine("Pullups Enabled");
161             }
162             else
163             {
164                 Console.WriteLine("Pullups could not be enabled");
165             }
166
167             while (1)
168             {
169                 I2CM.Read(DEV_ADDR_WRITE, V_IN_ADDR, NUM_BYTES, ref
                        Data_Array, ref Script_View);
170                 double V_IN = Calc_V_In(Data_Array[0], Data_Array[1]);
171                 double I_OUT = Calc_I_Out(Data_Array[2], Data_Array[3])
                        ;
172                 double V_OUT = Calc_V_Out(Data_Array[4], Data_Array[5])
                        ;
173                 double Temp = Calc_Temp(Data_Array[6], Data_Array[7]);
```

```
174
175                 now = DateTime.Now.ToString("h:mm:ss");
176
177                 string Temp1 = Temp.ToString("0.00");
178                 string V_OUT1 = V_OUT.ToString("0.00");
179                 string V_IN1 = V_IN.ToString("0.00");
180                 string I_OUT1 = I_OUT.ToString("0.00");
181
182                 using (StreamWriter w = File.AppendText(path))
183                 {
184                     w.WriteLine(now + " " + Temp1 + " " + V_OUT1 + " "
                            + V_IN1 + " " + I_OUT1);
185                 }
186
187
188
189
190
191
192             }
193         }
194     }
```

1. Variante der Datenabfrage:

```
1  def rec_data(device_adress, data_to_send, dcount):
2    temp = bus.read_i2c_block_data(device_adress, TEMP_ADDR,2)
3    i_out = bus.read_i2c_block_data(device_adress, I_OUT_ADDR,2)
4    v_out = bus.read_i2c_block_data(device_adress, V_OUT_ADDR,2)
5    v_in = bus.read_i2c_block_data(device_adress, V_IN_ADDR,2)
6
7    temp_high = temp[0]
8    temp_low= temp[1]
9
10   i_high = i_out[0]
11   i_low = i_out[1]
12
13   v_out_high = v_out[0]
14   v_out_low = v_out[1]
15
16   v_in_high = v_in[0]
17   v_in_low = v_in[1]
```

2. Variante der Datenabfrage: [language = Python] from smbus2 import SMBus import numpy as np import time import matplotlib.pyplot as plt from soundplayer import SoundPlayer from datetime import datetime import sys

DEV$_A DDR1 = 0x08 V_I N_A DDR = 0x20 I_O UT_A DDR = 0x22 V_O UT_A DDR = 0x24 TEMP_A DDR = 0x26$

V$_I N_H IST = [] I_O UT_H IST = [] V_O UT_H IST = [] TEMP_H IST = [] POW_H IST = [] data_t o_s end = np.empty((50)) dcount = 0$

def calc$_t emp(val_h igh, val_l ow) : val = (val_h igh << 8) + val_l ow Temp = ((val/1024 * 5) - 0.4)/0.0195 return Temp$

def calc$_{io} ut(val_h igh, val_l ow) : val = (val_h igh << 8) + val_l ow Amps = (val/1024 * 5)/0.3 return Amps$

def calc$_{vo} ut(val_h igh, val_l ow) : val = (val_h igh << 8) + val_l ow Volt = (val/1024 * 5) * 6 return Volt$

def calc$_{vi} n(val_h igh, val_l ow) : val = (val_h igh << 8) + val_l ow Volt = (val/1024 * 5) * 13 return Volt$

def rec$_d ata(device_a dress, data_t o_s end, dcount)$:

Data = bus.read$_i 2c_b lock_d ata(device_a dress, V_I N_A DDR, 8)$

temp$_h igh = Data[6] temp_l ow = Data[7]$

i$_h igh = Data[2] i_l ow = Data[3]$

v$_o ut_h igh = Data[4] v_o ut_l ow = Data[5]$

v$_i n_h igh = Data[0] v_i n_l ow = Data[1]$

new$_r ow = [] current_t ime = datetime.now().strftime("new_r ow.append(current_t ime) TEMP = calc_t emp(temp_h igh, temp_l ow)$

new$_row.append(TEMP)$
I$_OUT = calc_iout(i_high, i_low)new_row.append(I_OUT)$
V$_OUT = calc_vout(v_out_high, v_out_low)new_row.append(V_OUT)$
V$_IN = calc_vin(v_in_high, v_in_low)new_row.append(V_IN)$
new$_row.append(V_OUT * I_OUT)$
return new$_row$
bus = SMBus(1) while(1): try: Data1 = rec$_data(DEV_ADDR1)withopen("Test.txt",'a')asf : f.write(str(Data1) + "")$
except KeyboardInterrupt: bus.close() sys.exit() plt.show()

Code für die Datenverarbeitung

```
1  import numpy as np
2  import math
3  import matplotlib.pyplot as plt
4  import mpld3
5  import scipy.signal as scys
6  import enum
7  from datetime import datetime
8  import os
9
10 def get_measure_time(t1, t2, sample_cnt):
11     fmt = '%H:%M:%S:%f'
12     t1 = t1#[:-4]
13     t2 = t2#[:-4]
14     tstamp1 = datetime.strptime(t1,fmt)
15     tstamp2 = datetime.strptime(t2,fmt)
16
17     if tstamp1 > tstamp2:
18         td = tstamp1 - tstamp2
19     else:
20         td = tstamp2 - tstamp1
21     td_secs = int(round(td.total_seconds()))
22     print(td_secs)
23     time = []
24     print(sample_cnt/td_secs)
25     for i in range(0, sample_cnt):
26         x = i * td_secs / sample_cnt
27         time.append(x)
28
29     with open("Metadata.txt","a") as f:
30             f.write(str(sample_cnt/td_secs) + "\n")
31     return time
32
33 #Enumeration, hat Atm keine Funktionalit t
34 class labels(enum.Enum):
35         Temperatur = 1
36         Ausgangsspannung = 2
37         Eingangsspannung = 3
38         Ausgangsstrom = 4
39         Leistung = 5
40 verz = [" in  Celsius ", " in Volt", " in Volt", " in Ampere", " in
       Watt"]
41
42 #Metadaten f r alle Parameter (Spannung,Strom,Temp) berechnen und in
       einem 2D Array speichern
```

```
43 def calc_meta(data):
44         daten = np.empty((4, 5))
45         for i in range(0,4):
46             Mittel = np.mean(data[i])
47             StandAbw = np.std(data[i])
48             len_data = len(data[:][0])
49             minimum = np.min(data[i])
50             maximum = np.max(data[i])
51             daten[i][0] = Mittel
52             daten[i][1] = StandAbw
53             daten[i][2] = len_data
54             daten[i][3] = minimum
55             daten[i][4] = maximum
56             print("Mittelwert: ", Mittel, "Standardabweichung: ",
                   StandAbw,
57                 "L nge: " , len_data, "Minimum: ", minimum, "Maximum:
                       ", maximum)
58         return daten
59
60 #Metadaten z. B. nur f r die Temperatur von einem Board berechnen
61 def calc_meta_single(data):
62     Mittel = np.mean(data)
63     StandAbw = np.std(data)
64     len_data = len(data)
65     minimum = np.min(data)
66     maximum = np.max(data)
67     return Mittel, StandAbw, len_data, minimum, maximum
68
69 #Entpacken der Daten aus der Textfile und aufteilen in Zeitverlauf und
       Daten
70 def unr_data(path):
71         count_samples = 0
72         count_errors = 0
73         data = np.genfromtxt(path, dtype ='str')
74         Daten = np.empty((5, len(data)))
75
76        # with open("Metadata.txt" ,"a") as f:
77          #    f.write(path+" ")
78         Time = data[:,0]
79
80         #Kleiner Loop um Fehler beim Format zu beseitigen
81         #z. B. 08:14:54.56666 -> 08:14:54:56666
82         #der Punkt ist ein Fehler der mir beim Daten generieren
               passiert ist
83         for idx,element in enumerate(Time):
84             element = np.char.replace(element, '.', ':')
85             Time[idx] = element
```

```
86          Time = get_measure_time(str(Time[0]), str(Time[len(data) -1]),
                len(data))
87
88          Temp = data[:,1]
89          Temp = Temp.astype(np.float32)
90
91          V_OUT = data[:,2]
92          V_OUT = V_OUT.astype(np.float32)
93
94          V_IN = data[:,3]
95          V_IN = V_IN.astype(np.float32)
96
97          I_OUT = data[:,4]
98          I_OUT = I_OUT.astype(np.float32)
99
100         POW = V_OUT * I_OUT
101
102         count_samples += len(data)
103
104         Daten[0] = Temp
105         Daten[1] = V_OUT
106         Daten[2] = V_IN
107         Daten[3] = I_OUT
108         Daten[4] = POW
109
110         for i in range(0, 4):
111             for j in range(0, len(data)):
112                 if(Daten[i][j] < 0 or Daten[i][j] > 100):
113                     count_errors+=1
114                     # Daten[i][j] = Daten[i][j-1]
115         print(count_errors)
116
117         return Daten, Time
118
119 #Funktion zum Filtern von hohen Frequenzen
120 def apply_filter(data):
121         n = 15
122         b = [1.0 / n] * n
123         a = 1
124         filtered = scys.lfilter(b,a, data)
125         return filtered
126
127 #Berechnen der Leistung aus Daten
128 def Calc_Power(Voltage, Current):
129         return Voltage * Current
130
131
```

```
132 #War eine spielerei, hat at the moment keine Verwendung
133 def Conf_Plot(Data = None, Time = None, x = None, y = None, Filter =
        None, Title = None):
134         if(Filter == True):
135             Data = apply_filter(Data)
136 
137         y = labels[y].value
138         fig = plt.figure()
139         ax1 = fig.add_subplot(211)
140         if(x == None):
141             ax1.set_ylabel(labels(y).name + verz[y-1])
142             ax1.set_xlabel("Zeit in Sekunden")
143             ax1.set_title(Title)
144             y_mean = [np.mean(Data[y-1])] * len(Data[y-1])
145             ax1.plot(Time[:], Data[y-1])#, label= labels(y).name + verz
                    [y] + vs + Zeit in Sekunden)
146         else:
147             x = labels[x].value
148             ax1.set_ylabel(labels(y).name + verz[y-1])
149             ax1.set_xlabel(labels(x).name + verz[x-1])
150             ax1.set_Title(Title)
151             ax1.plot(Data[x-1], Data[y-1])#, label= labels(y).name +
                    verz[y-1] + vs + labels(x).name + verz[x-1].name)
152         ax1.legend(loc ='best')
153         plt.savefig(labels(y).name)
154         plt.plot()
155         plt.show()
156 
157 directory = 'C:\\Users\\lamer\\Downloads\\Bachelor\\Data\\CSV\\'
158 for filename in os.listdir(directory):
159 
160     Data, Time = unr_data(directory+filename)
161     x = calc_meta_single(Data[4])
162     print(filename,": ", x[0], x[1])
163 
164     plt.plot(Time,Data[0], label = filename)
165     plt.xlabel('Zeit in Sekunden')
166     plt.ylabel('Temperatur in C')
167     plt.legend(loc = 2)
168     plt.show()
```

Code für das Neuronale Netz.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pydot
import graphviz
import numpy as np
import math
import matplotlib.pyplot as plt
import mpld3
import scipy.signal as scys
import enum
from datetime import datetime

config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
config.log_device_placement = True
sess= tf.compat.v1.Session(config=config)

Training_Data = np.concatenate((_5Hz, _20Hz, _30Hz,  __5Hz, __20Hz,
    __30Hz,___5Hz, ___20Hz, ___30Hz), axis =0)
np.random.shuffle(Training_Data)
Solution = Training_Data[:, Training_Data.shape[1]-1]
Training_Data = Training_Data[:,:-1]
Input_Parameter = Training_Data.shape[1]
Output_Layer_Size = 3
Hidden_Layer_Size = int(Input_Parameter*(2/3)) + Output_Layer_Size
print(Hidden_Layer_Size)



for i in range(0, Training_Data.shape[1]-2):
    max1 = max(Training_Data[:,i])
    min1 = min(Training_Data[:,i])
    Training_Data[:,i] = ( (Training_Data[:,i] - min1) / (max1-min1))


with sess:
    model = tf.keras.Sequential(
        [
            layers.Dense(80, input_dim = Input_Parameter,  activation =
                "relu", name = "Start"),
            layers.Dense(40, activation = "relu", name = "hidden1"),
            layers.Dense(3, activation ="softmax", name = "Output"),
        ]
     )
    model.compile(optimizer = "adam", loss=tf.keras.losses.
```

```
            SparseCategoricalCrossentropy(from_logits=True),
45                          metrics=['accuracy'])
46      history = model.fit(Training_Data, Solution, epochs = 300)
```