Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Lennart Koch

Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Lennart Koch

Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Jürgen Vollmer
Zweitgutachter : Prof. Dr.-Ing. Karl-Ragmar Riemschneider

Abgegeben am 21. April 2010

Lennart Koch

Thema der Diplomarbeit

Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren

Stichworte

Klirrfaktor, ABS-Sensor, Harmonische, DFT, Rad-Unrundlauf, Unterabtastung Aufwandsminimierung, magnetischer Sensor, Drehzahlsensor

Kurzzusammenfassung

In dieser Arbeit wird das Thema „Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren“ behandelt. Diese Arbeit baut auf der Diplomarbeit „Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren“ von Herrn N. Jegenhorst auf. Zur Analyse des Problems wurden Messergebnisse, die mit einem bestehenden Mikrocontroller-basierten System gewonnen wurden, untersucht. Der Algorithmus des bestehenden Systems wurde mit Matlab durch eine Festkomma-Arithmetik mit begrenzter Wortbreite nachgebildet. Als Kontrolle dienten Fließkomma-Berechnungen. In einem weiteren Schritt wurde der verwendete Algorithmus in Hinsicht auf die Berechnungszeit optimiert. Zur Überprüfung der Funktion des Simulationsprogramms, sind die Simulationen ausgeführt worden. Bei der anschließenden Bewertung der Ergebnisse ist ein Unrundlauf des Encoderrades festgestellt worden, dessen Auswirkungen analysiert worden sind.

Lennart Koch

Title of the paper

Low complexity estimation the state of ABS-sensors based on harmonics

Keywords

Total Harmonic Distortion, ABS-sensor, harmonic, DFT, irregular rotating of Encoder, subsampling, magnetic sensor, rotation-speed-sensor

Abstract

The topic of this diploma thesis is “Low complexity estimation the state of ABS-sensors based on harmonics”. This thesis is based on the diploma thesis of Mr. N. Jegenhorst “Controllersystem for ABS-Sensors“. To analyze the problem it’s necessary to check up the measurements of the microcontroller based system first. In the next step the algorithm of the system were cloned with the fixpoint-toolbox in Matlab. The results checked with floating-point results and the algorithm was optimized with respect to the calculation time. For checking the result some simulations were executed. At least the results were analyzed and evaluated. In this way an irregular rotating of the Encoder was determined and the influence analyzed in this thesis.

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. Jürgen Vollmer, dafür bedanken, dass er es ermöglicht hat diese Diplomarbeit zu erstellen und mir zu jederzeit zur Seite stand und sich viel Zeit für meine Fragen genommen hat.

Ebenso bedanke ich mich bei Herrn Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider für seine Arbeit als Zweitgutachter und Projektverantwortlicher.

Weiterhin bedanke ich mich besonders bei Herrn Dipl.-Ing. Martin Krey, für seinen fachlichen Rat und die tatkräftige Unterstützung bei meiner Diplomarbeit. Außerdem danke ich dem gesamten Team für die gute Zusammenarbeit. Besonders erwähnen möchte ich Herrn Martin Stahl und Herrn Niels Jegenhorst, die die Aufnahme der Messwerte ermöglicht und mir freundlicherweise zur Verfügung gestellt haben. Darüber hinaus hatten beide immer ein offenes Ohr für meine Fragen.

Inhaltsverzeichnis

1. Einführung

Diese Arbeit beschäftigt sich mit dem Thema „Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren“. Sie ist ein Teil des Forschungsprojekts „Experimentelle digitale Signalverarbeitung und Zustandsbestimmung für ABS-Sensoren (ESZ-ABS)“, das derzeit mit Unterstützung der Industrie an der Hochschule für Angewandte Wissenschaften Hamburg durchgeführt und von Prof. Dr.-Ing. Riemschneider geleitet wird. Die Abkürzung ABS steht in dieser Arbeit für das Anti-Blockier-System eines Kraftfahrzeugs. Harmonische ist der Fachausdruck für „ganzzahlige Vielfache einer bestimmten Grundschwingung“ [17]. Wobei die erste Harmonische Schwingung der Grundschwingung selber entspricht.
Gerade aktuell geht die Meldung durch die Presse , dass der Automobilhersteller Toyota weltweit 437.000 Modelle vom Typ Prius zurückrufen muss, da diese Probleme mit dem ABS-System haben. Grund dafür ist ein Programmfehler in der Steuerelektronik des Autos. Der durch diese Panne entstandene Schaden ist für Toyota sehr groß, da zum einen das Image beschädigt wurde und zum anderen die Reparaturkosten für die betroffenen Modelle von Toyota getragen werden muss. Das ist ein Indiz für die Bedeutung der fehlerfreien Funktion des Systems. Da der Sensor ein Teil des Systems ist, gilt für ihn die gleiche Wichtigkeit.
Um eine korrekte Einbauposition des Sensors zu ermitteln, ist im Rahmen der Diplomarbeit von Herrn Niels Jegenhorst ein Radmessplatz, sowie ein Messsystem, das im folgenden als Demonstrator bezeichnet wird, entstanden, dessen Spezifikationen in Abschnitt 2.2 dargestellt sind. Die Messdaten, die als Grundlage für die Simulationen dienen, sind ebenfalls im Rahmen der Arbeit von Herrn Jegenhorst entstanden.
Das Ziel dieser Arbeit ist es den Aufwand des von Herrn Jegenhorst [7] entwickelten Algorithmus zu minimieren und weiter zu optimieren. Der Algorithmus steht im Fokus der Untersuchungen. Um diesen zu simulieren ist die Matlab-Version „R2008a“ als Entwicklungsumgebung verwendet worden.
Um eine Lösung des Problems zu erarbeiten werden in Kapitel 3 zunächst die Messdaten, die von Herrn Jegenhorst aufgenommen worden sind, analysiert und für die neuen Bestimmungen aufbereitet. Da der Mikrocontroller keine Gleitkomma-Operationen durchführen kann, muss im Kapitel 4 die Festkomma-Arithmetik[1] mit Matlab nachgebildet werden, um nächsten Kapitel ein Programm entwickeln zu können, das die Klirrfaktorberechnung nachbildet. In Kapitel 6 sind neue Methoden zur Berechnung der Wurzelfunktion erarbeitet und

[1] engl. Fixed-Point- oder Fixpoint-Arithmetik

programmiert worden und in Kapitel 7 sollen die verwendeten Berechnungsverfahren in die Programmiersprache „C“ übersetzt werden, um die Berechnungszeit eines Mikrocontrollers (MC) zu ermitteln. Als nächstes soll die Kirrfaktorberechnung im Rahmen von Kapitel 8 in ein Programm integriert werden, das eine Simulation durchführen und die Ergebnisse abspeichern und verwalten kann. Daraufhin sind in Kapitel 9 einige Simulationen mit dem neuen Programm durchgeführt, sowie mögliche Reduzierungen der Auflösung des Analog-Digital-Umsetzers sowie der Sinus- und Kosinus-Wertetabellen untersucht werden. Zum Schluss sollen die Simulationsergebnisse in Kapitel 10 diskutiert und anschließend bewertet werden.

2. Grundlagen

2.1. Funktion des Anti-Blockier-Systems

Das Anti-Blockier-System besteht im wesentlichen aus einem Sensor, einem Encoderrad und einer Kontrolleinheit, die für die Auswertung zuständig ist. Der ABS-Sensor gehört zu der Gruppe der magnetoresistiven Sensoren, dessen Funktionsprinzip auf dem magnetoresistiven Effekt basiert. Der magnetoresistive (MR) Effekt ist die Widerstandsänderung eines Materials in Abhängigkeit des magnetischen Feldes. Dieser Effekt ist 1856 von Thomson entdeckt worden. In einem ABS-Sensor sind 4 magnetoresistive Widerstände in Form einer Wheatstone'schen Messbrücke miteinander verschaltet Die beiden Widerstände R_T dienen zur Offsetkompensation des Sensors. Der Aufbau dieses ABS-Sensors, ist in Abbildung 2.1 dargestellt.

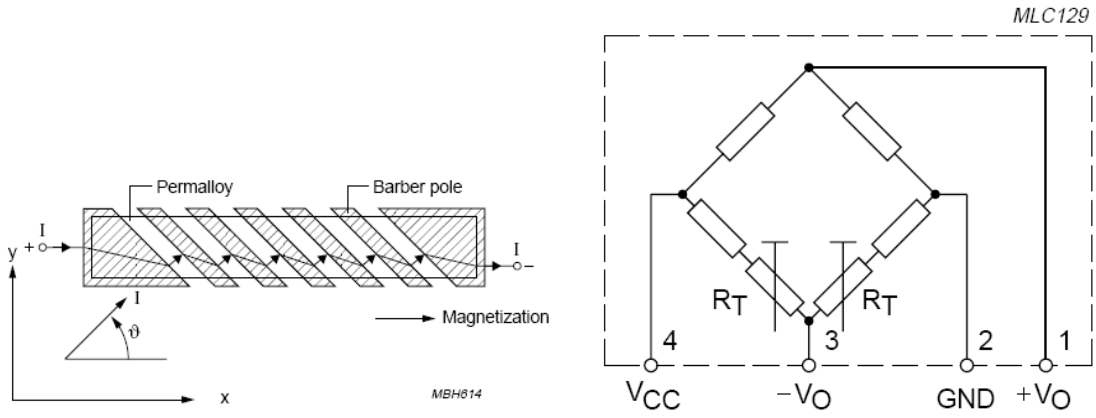

Abbildung 2.1.: Links: Aubau eines magnetoresistiven Widerstands; Rechts: Innerer Aufbau eines ABS-Sensors [16]

Magnetoresistive Sensoren besitzen ähnliche Eigenschaften, wie Hall-Sensoren. Allerdings ist es mit magnetoresistiven Sensoren möglich kleinere Änderungen der Feldstärke zu detektieren. Während magnetoresistive Sensoren eine Änderung der Ausgangspannung von 20 mV/kA/m aufweisen, liegt der Wert für Hall Sensoren typischerweise bei 0.4 mV/kA/m.

Der Widerstand besteht aus Permalloyschicht. Das ist eine weichmagnetische Nickel-Eisen-Legierung mit hoher magnetischer Leitfähigkeit. Die charakteristische Kennlinie ist in Abbildung 2.2 gestrichelt dargestellt Zur Linearisierung der Widerstände sind auf die Permalloyschicht dünne Metallbahnen sogenannte Barber-Pole aufgebracht.
In Abhängigkeit von der Intensität und der Richtung des Magnetfeldes ändern sich die Widerstände innerhalb der Messbrücke und somit auch das Ausgangssignal des Sensors. Die charakeristische Kennlinie eines Sensors ist ebenfalls in Abbildung 2.2 dargestellt.

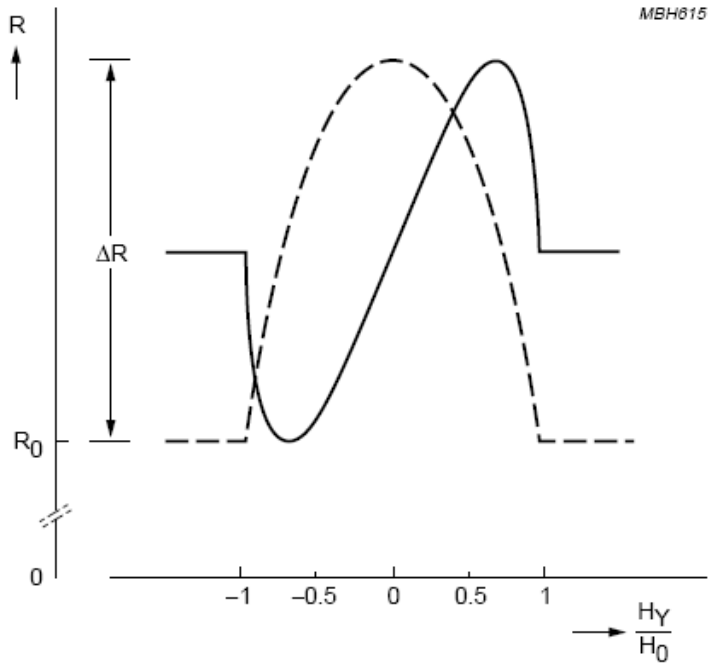

Abbildung 2.2.: gestrichelte Kennlinie: charakaristische Kennline eines Permalloy Widerstands; durchgezogene Linie: charakteristische Kennlinie eines ABS-Sensors [16]

Das Encoderrad ist am Radlager eines Autos befestigt und dreht sich daher mit derselben Drehzahl, wie das Rad eines Autos. Diese Frequenz wird in dieser Arbeit als Drehfrequenz f_D bezeichnet. Es wird zwischen zwei verschiedenen Arten von Encoderrädern unterschieden.

- **aktives Encoderrad**: Das aktive Encoderrad ist magnetisiert. Beim aktiven Encoderrad sind immer abwechselnd Nord- und Südpole nebeneinander angebracht, sodass sich die Richtung des Magnetfeldes, und somit auch das Ausgangssignal des Sensors kontinuierlich ändert Bei einem aktiven Encoderrad wird zur Stabilisierung des

Magnetfeldes nur ein kleiner Dauermagnet benötigt. Es besitzt eine glatte Oberfläche. Wird ein aktiver ABS-Sensor verwendet, können bereits Geschwindigkeiten von 0,1km/h ausgewertet werden. Ein Bild eines Radlagers sowie eine Skizze des Aufbaus eines aktiven Encoderrades sind in Abbildung 2.3 dargestellt.

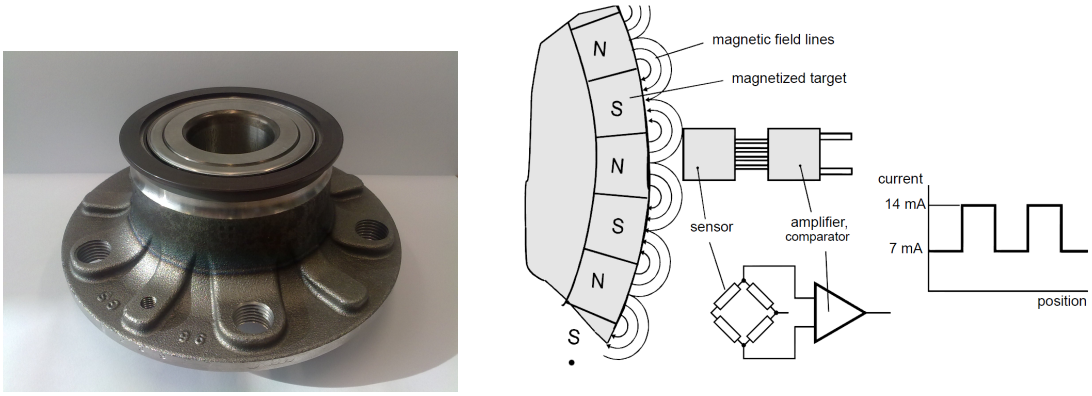

Abbildung 2.3.: Links: Bild eines Radlagers mit aktiven Encoderrad eines VW Golf V; Rechts: Skizze eine aktiven Encoderrades [14]

- **passives Encoderrad**: Das passive Encoderrad ist nicht magnetisiert. Es ist äußerlich durch einen ferromagnetischen zahnradförmigen Ring um das Radlager gekennzeichnet, der magnetisiert ist. Wenn ein passives Encoderrad verwendet werden soll, wird ein zusätzlicher Dauermagnet benötigt, um ein Stützfeld aufzubauen. Die Messreihen, die im Verlauf dieser Arbeit ausgewertet werden sollen, sind mit einem passivem Encoderrad aufgenommen worden. Ein Radlager mit passivem Encoderrad ist in Abbildung 2.4 dargestellt.

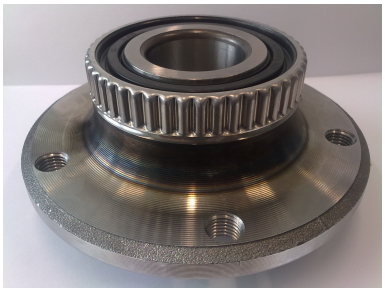

Abbildung 2.4.: Radlager mit passivem Encoderrad eines BMW 330i Touring

Die Entstehung des Ausgangssignals des Sensors ist in Abbildung 2.5 für ein passives Encoderrad dargestellt.

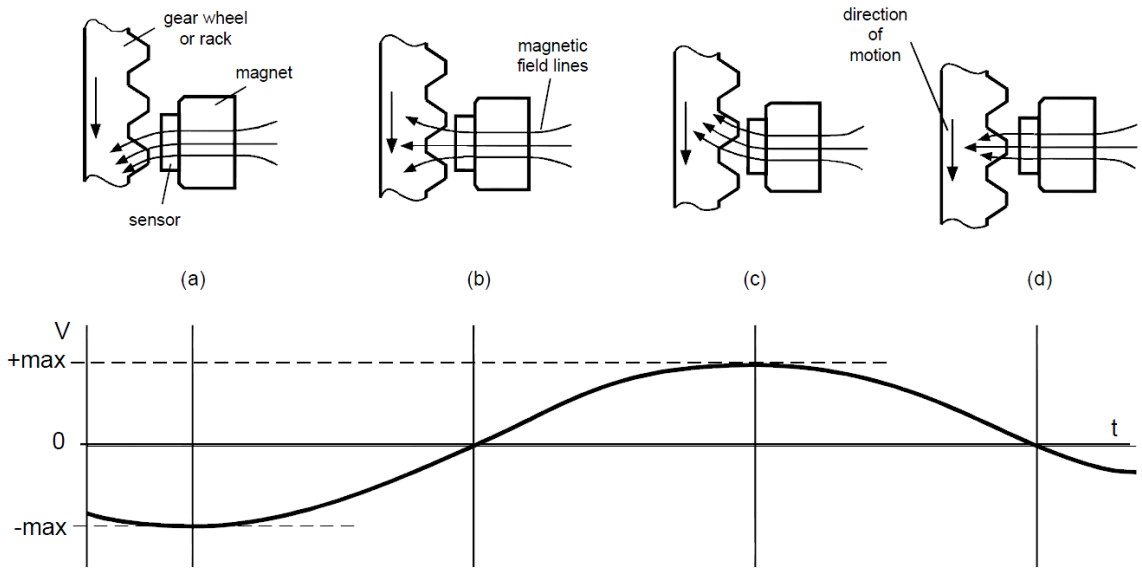

Abbildung 2.5.: Funktionsweise eines ABS-Sensors [14]

In der Abbildung 2.5 ist zu erkennen, dass pro vorbei laufenden Zahn eine Periode am Ausgang des Sensors detektiert werden kann. Die Frequenz dieses Signals wird in dieser Arbeit mit Zahnfrequenz f_Z bezeichnet. Die Zahnfrequenz ist um einen konstanten Faktor größer als die Drehfrequenz. Dieser Faktor ergibt sich aus der Anzahl der Zähne bzw. der Anzahl der Nord- und Südpole z.

$$z = \frac{f_Z}{f_d} \tag{2.1}$$

Damit das Anti-Blockier-System richtig funktioniert, darf sich der Sensor nicht zu nah, aber auch nicht zu weit vom Encoderrad entfernt sein. Die vom Hersteller vorgeschriebene Einbauposition soll durch die Schätzung des Klirrfaktors ermittelt werden. Der Klirrfaktor ist ein Maß für nichtlineare Verzerrungen eines Signals. Er ist als das Verhältnis der Leistung der Oberwellen zur Gesamtleistung des Signals ohne Gleichanteil definiert und soll in dieser Arbeit als prozentuale Größe geschätzt werden, um später Indikatorbits aus den Werten abzuleiten. Wenn sich die Position des Sensors im Laufe der Zeit verschiebt soll der Sensor eine Warnung an das Steuergerät senden. So kann gewährleistet werden, dass sich der Sensor immer in dem vom Hersteller vorgegebenen Abstand zum Encoderrad befindet. Befindet sich der Sensor zu dicht am Encoderrad, ist das Magnetfeld stärker als vom Hersteller vorgegeben. Die magnetisch-elektrische Kennlinie kann für diese magnetische Feldstärke nicht mehr als nahezu linear angesehen werden und das Ausgangssignal wird verzerrt. Dieser Fall ist in Abbildung 2.6 dargestellt.

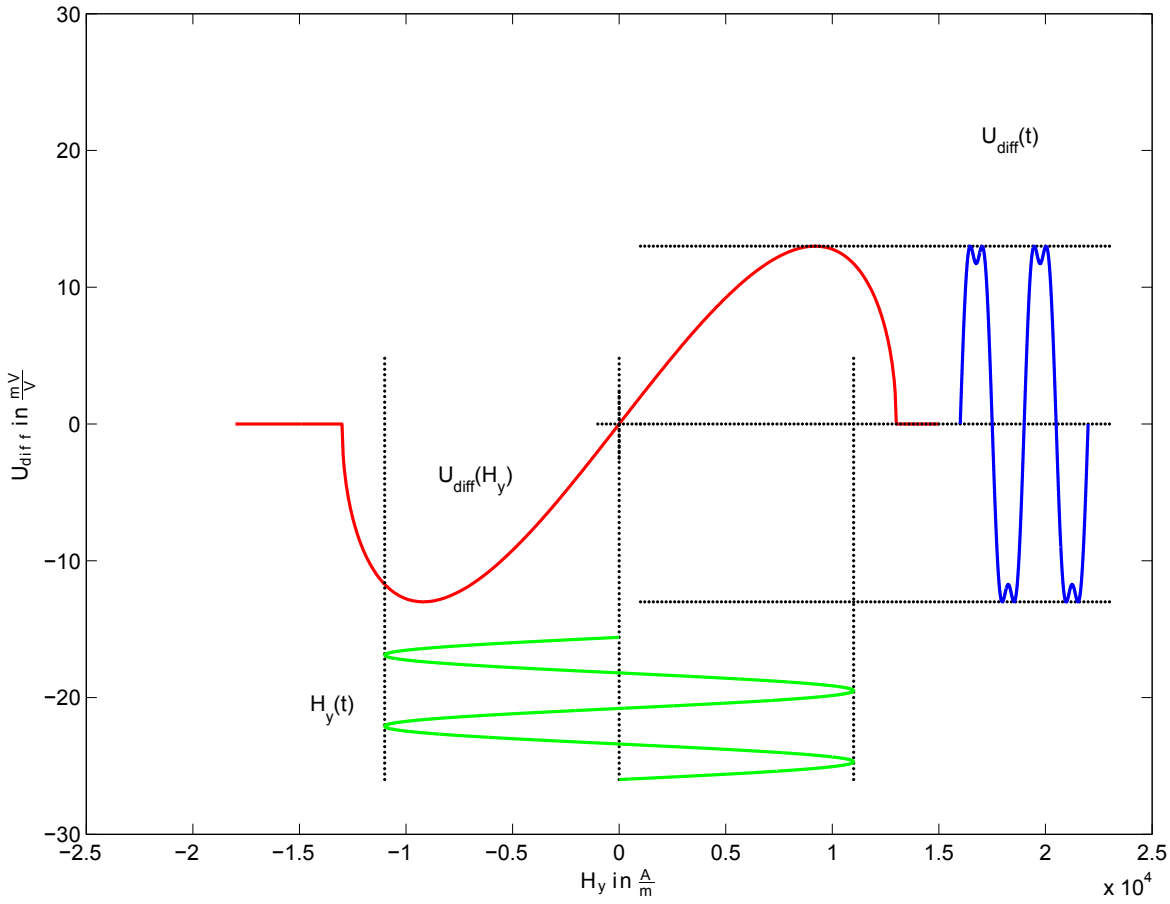

Abbildung 2.6.: Linearisierte magnetisch-elektrische Kennlinie des AMR-Sensorchips, mit einer Aussteuerung stärker als der näherungsweise lineare Bereich, nach der Theorie lt. Dibbern[7]

Ist der Sensor mit dem vorgegebenen Abstand zum Encoderrad eingebaut, ändert sich die Feldstärke hingegen nicht so stark und die Amplitude der Ausgangsspannung ist kleiner. Wie in Abbildung 2.7 zu sehen ist, kann die Kennlinie als nahezu linear angesehen werden und das Signal wird nicht oder nur leicht verzerrt.

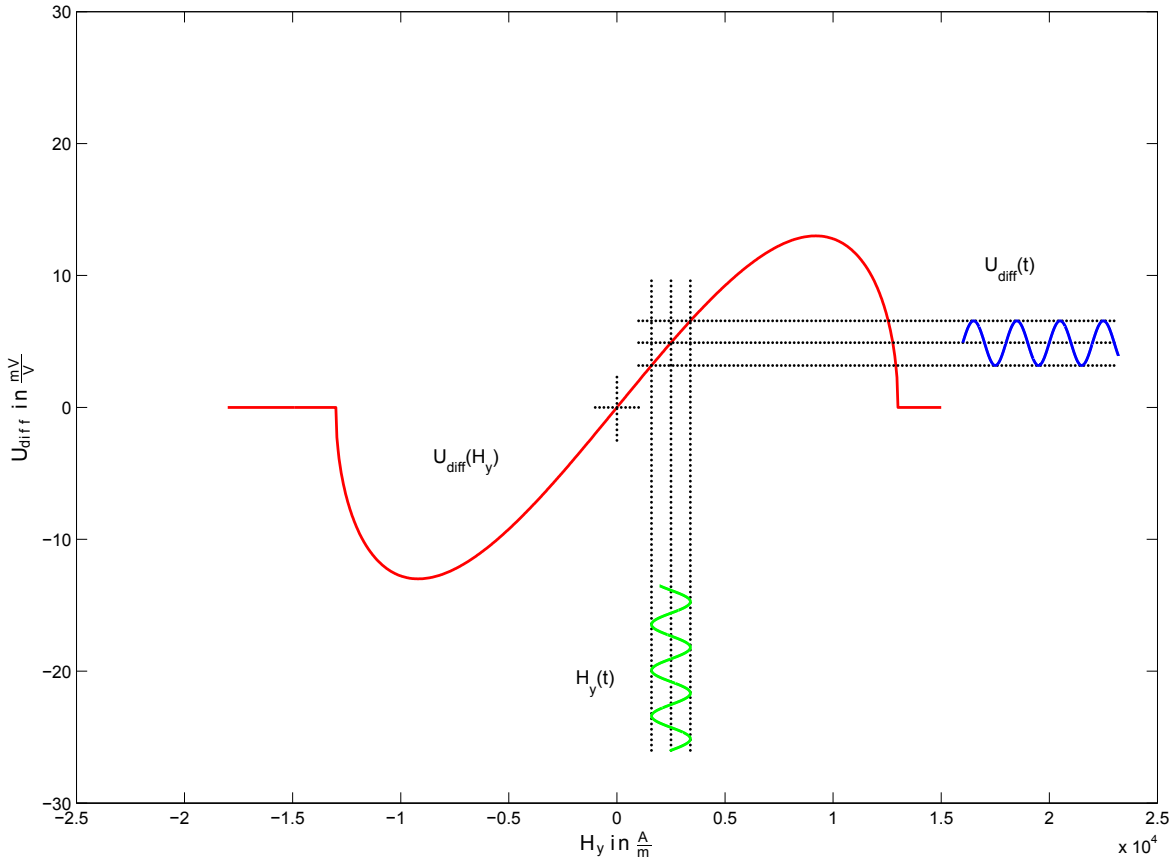

Abbildung 2.7.: Linearisierte magnetisch-elektrische Kennlinie des AMR-Sensorchip,mit einer Aussteuerung im näherungsweise linearen Bereich , nach der Theorie lt. Dibbern [7]

2.2. Begriffsdefinition und Spezifikation des Radmessplatzes

2.2.1. Begriffsdefinition

Der Radmessplatz ist im Rahmen der Diplomarbeit Jegenhorst [7] entstanden. Mit Hilfe des Messplatzes ist es möglich, die Veränderung des Klirrfaktors im Abhängigkeit von der Entfernung zwischen Encoderrad und Sensor zu untersuchen. Die Messdaten, die mit diesem Radmessplatz aufgenommen wurden, sind Voraussetzung für diese Arbeit. Die Schätzung der Harmonischen, sowie die Sensordiagnosefunktionen wurden mit Hilfe eines Mikrocontrollers der MSP430-Familie durchgeführt. Zur Kontrolle wurde das Ausgangssignal

parallel zur Auswertung auf der Mikrokontrollerplattform mit einem Speicheroszilloskop DPO4054 von Tektronix mit identischer Vorverstärkung aufgezeichnet.

2.2.2. Spezifikationen

Für den Radmessplatz wurden Spezifikationen festgelegt, diese sind im folgenden dargestellt:
Für alle Messungen, die dieser Arbeit zugrunde liegen, wird ein passives Encoderrad mit 50 Zähnen verwendet.
Die Messungen werden bei einer Zahnfrequenz f_z von 107 Hz durchgeführt. Daraus folgt, dass sich das Encoderrad mit einer Drehfrequenz $f_d = \frac{107}{50} = 2,14$Hz dreht. Daraus ergibt sich eine Zeit, die vergeht, während sich das Rad um eine Umdrehung weiter dreht, von $T_d = \frac{1}{f_D} = \frac{1}{2,14Hz} = 0,47s$.
Die Aufnahme der Messwerte erfolgt mit dem integrierten 12 Bit Analog-Digital-Umsetzer. Durch einen Vorverstärker ist gewährleistet, dass dieser immer ausreichend ausgesteuert ist. Nähere Informationen sind aus [7] zu entnehmen. Es werden 64 Abtastwerte pro Periode des Ausgangssignals des Sensors aufgenommen. Für eine Periodendauer von $T_z = \frac{1}{107Hz} = 9,3$ms ergibt sich eine erforderliche Abtastfrequenz von $T_A = 64 \cdot 107$Hz $= 6,848$kHz. Da diese Abtastzeit nicht realisierbar ist, wird eine Unterabtastung verwendet. Die Funktionsweise der Abtastung ist in Abbildung 2.8 dargestellt.

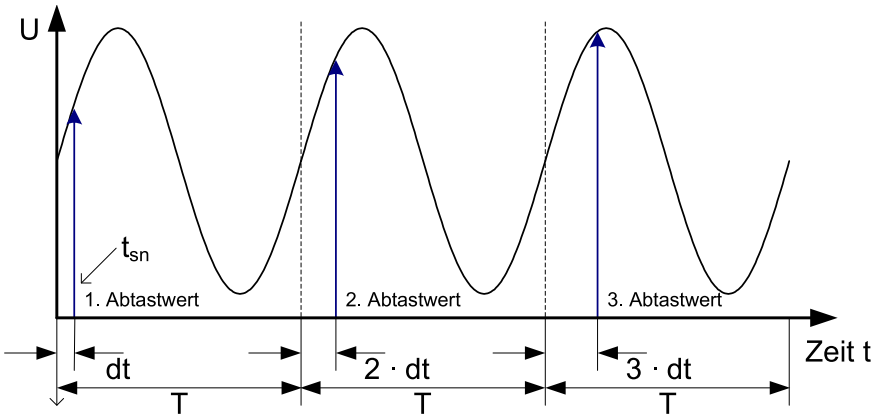

Abbildung 2.8.: Prinzip der sequenziellen Abtastung [7]

Nähere Informationen können aus [7] entnommen werden. Dabei wird alle 6-7 Perioden ein Wert aufgenommen. Daraus ergibt sich, dass insgesamt etwa 400 Perioden berücksichtigt werden. Es wird zudem angenommen, dass während der Aufnahme der Daten die Frequenz, sowie die Amplitude konstant sind.

Die Berechnung der diskreten Fouriertransformation wird mit einer Sinus- und einer Kosinus-Tabelle durchgeführt, die mit einer Auflösung von 10 Bit in dem Programmspeicher des Mikrocontrollers abgelegt sind. Der Klirrfaktor wird im Speicher des Mikrocontrollers als Look-Up-Tabelle mit 256 Werten und einer Wortbreite von 8 Bit zur Verfügung gestellt.
Das Ausgangssignal des Sensors wird mit dem Oszilloskop vier Sekunden lang aufgezeichnet. Die Aufzeichnung der Daten erfolgt mit einer Samplingrate von 250 k/Samples pro Sekunde. Daraus ergibt sich, dass in vier Sekunden 1 Million Samples aufgenommen worden sind, das entspricht einer Abtastfrequenz $f_s = 250$kHz. Die Auflösung des Oszilloskops beträgt 8 Bit. Davon sind allerdings nur sieben Bit verlässlich, da das letzte Bit vom Rauschen bestimmt wird.

3. Analyse und Aufbereitung der verfügbaren Messdaten

3.1. Verfügbare Messdaten

Im diesem Kapitel sollen die Messdaten, die von Herrn Jegenhorst im Rahmen seiner Diplomarbeit [7] aufgenommen wurden, analysiert und für die Verwendung in dieser Arbeit aufbereitet werden. Ziel dieses Kapitels ist es, die analysierten Daten zu reduzieren und durch fehlende Informationen zu ergänzen, um diese später in einer neuen Datenstruktur abspeichern zu können.
Die verwendeten Messdaten nehmen auf der Festplatte viel Speicherplatz ein, sodass sie in typischerweise acht Dateien abgelegt worden sind. Es bestehen zwei Datenstrukturen. In der ersten Datenstruktur mit dem Namen „measure_demo“ sind, die vom Demonstrator ermittelten Daten enthalten. In der zweiten Datenstruktur mit dem Namen „measure_scope“ sind die Kontrollmessungen mit dem Oszilloskop enthalten. In der ersten Datei ist noch eine weitere Struktur mit dem Namen „parameters“ abgelegt. In dieser Struktur ist angegeben, bei welcher Distanz eine Messung beginnen und enden soll und wie groß die Entfernung zwischen zwei Messungen sein darf. Außerdem sind noch die Umschaltpunkte des internen Vorverstärkers sowie die Abtastrate des Oszilloskops enthalten. Da diese Struktur für die Simulation sehr wichtige Daten enthält, ist sie in dieser Arbeit verwendet und zusammen mit der neuen Datenstruktur abgespeichert worden.
Aus der Datenstruktur „measure_demo“ sind das Eingangssignal „u_diff“, die Distanz zwischen Sensor und Encoderrad „distance“, der Verstärkungsfaktor des Vorverstärkers „gain“, die vom Mikrocontroller berechneten Beträge der ersten bis fünften Harmonischen „mag“ sowie der vom Mikrocontroller berechnete Klirrfaktor „hd_lut“ entnommen worden. Die anderen Variablen sind für die Simulation nicht relevant oder werden während der Simulationen neu berechnet.
Aus der Struktur „measure_scope“ sind die Distanz zwischen Sensor und Encoderrad „distance“ sowie ein Teil des gemessenen Signals „u_diff_y“ übernommen worden.

3.2. Aufbereitung der Oszilloskopdaten

Damit die Annäherung des Oszilloskopsignals durch eine Fourierreihe zu korrekten Ergebnissen führt, ist es wichtig, möglichst genau die Nulldurchgänge des Signals und damit die Periode zu erkennen. Dadurch soll erreicht werden, dass das herausgetrennte Signal möglichst aus ganzen Perioden besteht, um das Entstehen von Leck-Effekten zu vermeiden. Im Gegensatz zu den vorhanden Scripten aus [7] ist hier auf die Verwendung des Referenzsensors verzichtet worden, da die beiden Sensoren offensichtlich eine verschiedene Periodendauer ermitteln, und somit die letzte Periode des Signals unvollständig enthalten ist. In Abbildung 3.1 ist ein Ausschnitt des Oszilloskopsignals, das mit dem Programm „rmp_stepper_scope_record_analyze_tekdata“, welches in [7] enthalten ist, erzeugt worden.

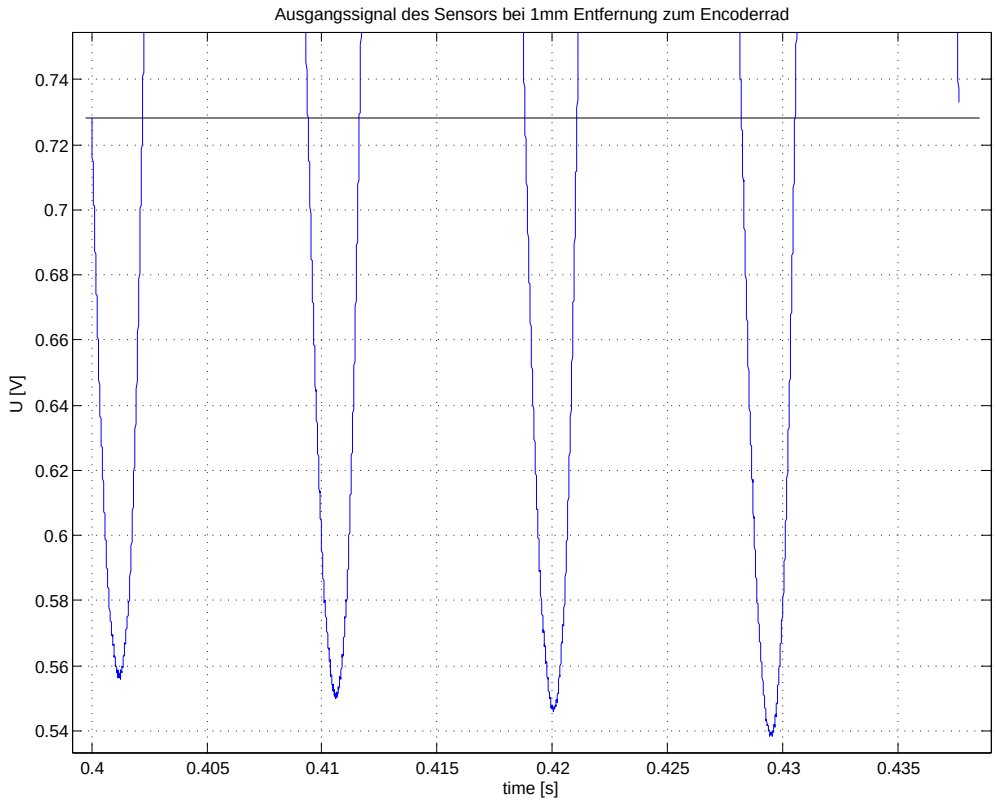

Abbildung 3.1.: Signal dessen Nulldurchgänge mit dem Referenzsensor detektiert worden sind

Dieses Problem soll in dieser Arbeit vermieden werden, deshalb ist es zuerst kopiert und dann gefiltert worden. Dann ist aus dem gemessenen Signal mit dem Befehl: `ref = sign(c1 - mean(c1))` das Referenzsignal erzeugt worden. Für die Erkennung der Nulldurchgänge ist das vorhandene Softwarekonzept übernommen worden. Die

Nulldurchgänge können durch diese Maßnahme, wie in Abbildung 3.2 zu sehen ist, besser erkannt werden.

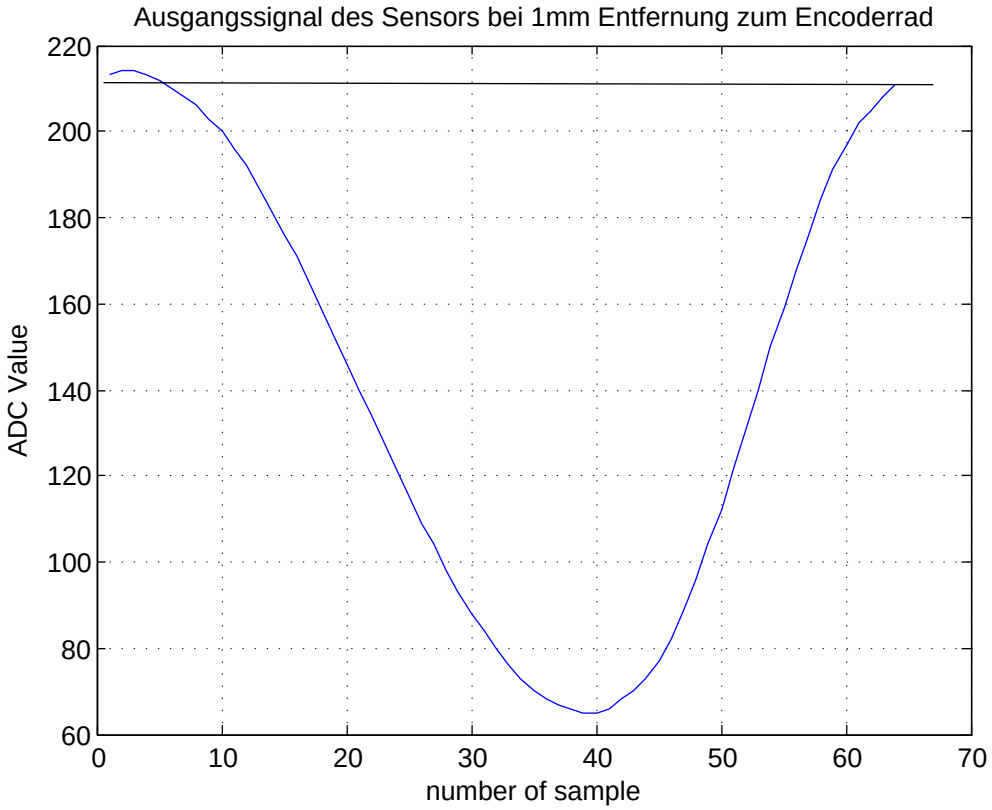

Abbildung 3.2.: Signal dessen dessen Nulldurchgänge mit dem selbst erstelltem Referenzsignal detektiert worden sind

3.3. Approximation des gemessen Signals

3.3.1. Idee zur Berechnung des approximierten Signals

Die gemessenen Signale, die meistens über mehrere Perioden aufgenommen worden sind, sollen für die Simulation auf eine Periode reduziert werden. Dafür ist eine Fouriertransformation über alle aufgenommen Perioden durchgeführt worden. Danach ist aus den Fourierkoeffizienten eine Fourierreihe aufgestellt worden, die sich dem gemessenen Signal möglichst gut annähern soll. Wenn zum Beispiel ein Signal über zwei Perioden aufgenommen worden ist und von diesem Signal das Spektrum berechnet wird, ist zu erwarten, dass jeder zweite Fourierkoeffizient Null ist. Wenn dass Signal zum Beispiel durch 10 Harmonische beschrieben worden ist, bedeutet das, dass die ersten 20 Fourierkoeffizienten berücksichtigt

werden müssen, da nur jeder zweite Fourierkoeffizient eine Harmonische des Signal darstellt. Die Approximation des gemessen Signals ist in zwei Teile aufgeteilt worden. Im ersten Schritt soll während der Aufbereitung der Messdaten eine Fouriertransformation durchgeführt und daraus die Fourierkoeffizienten bestimmt werden. Bei der Durchführung der Simulation soll dann das gemessene Signal unter Berücksichtigung der Fourierkoeffizienten durch eine Fourierreihe angenähert werden. Diese Aufteilung ist eingeführt worden, da die Anzahl der Samples pro Periode zu diesem Zeitpunkt noch nicht bekannt ist.

3.3.2. Mathematische Herleitung

In diesem Abschnitt ist die mathematische Herleitung zu dem Lösungsansatz aus dem vorherigen Kapitel dargestellt. Bevor ein Signal aus den Messdaten errechnet werden kann, müssen zunächst die komplexen Fourierkoeffizienten $\underline{S}_k$ mit Hilfe der diskreten Fouriertransformation (DFT) bestimmt werden. Diese kann für ein Signal s, das aus einer geraden Anzahl von Samples M besteht, mit folgender Formel berechnet werden:

$$\underline{S}_k = \frac{1}{M} \sum_{n=0}^{M-1} s[n] \cdot e^{-j\frac{2\pi kn}{M}} \tag{3.1}$$

Danach soll das Signal mit Hilfe der Fourierreihe mit einer bestimmten Anzahl von harmonischen Schwingungen approximiert werden, damit die Signale für eventuelle spätere Untersuchungen reproduzierbar sind. Die allgemeine Form einer komplexen Fourierreihe, die das zeitkontinuierliche Signal s mit $K = \frac{M}{2}$ Harmonischen annähert, lautet wie folgt:

$$s[n] = \sum_{k=-K}^{K-1} S_k \cdot e^{j\frac{2\pi kn}{2 \cdot K}} \tag{3.2}$$

Da es sich bei dem gemessenem Signal um ein reelles Signal handelt, gelten für das Spektrum folgende Symetrieeigenschaften:

$$\underline{S}_k = \underline{S}_{-k}{}^*$$

für periodische Signale sind außerdem folgende Aussagen möglich:

$$\underline{S}_{-k} = \underline{S}_{M-k}$$
$$\underline{S}_k = \underline{S}_{M-k}{}^*$$

Diese Beziehungen lassen sich einfach beweisen. Der Beweis kann im Anhang A.1 nachgelesen werden. Werden diese Eigenschaften in der Gleichung berücksichtigt, ergibt sich folgender Ausdruck:

$$s[n] = S_0 + \sum_{k=1}^{K-1} \left[\underline{S_{-k}} \cdot e^{-j\frac{2\pi kn}{2 \cdot K}} + \underline{S_k} \cdot e^{j\frac{2\pi kn}{2 \cdot K}} \right] \tag{3.3}$$

Werden $\underline{S_k}$ sowie $\underline{S_{-k}}$ durch folgende Ausdrücke ersetzt, ergibt sich folgender Ausdruck:

$$\underline{S_k} = |S_k| \cdot e^{-j\varphi_k}$$

$$\underline{S_{-k}} = |S_k| \cdot e^{j\varphi_k}$$

lässt sich die Gleichung 3.3 wie folgt umschreiben:

$$s[n] = S_0 + \sum_{k=1}^{K-1} \left[|\underline{S_k}| \cdot e^{j\varphi_k} \cdot e^{j\frac{2\pi kn}{2 \cdot K}} + |\underline{S_k}| \cdot e^{-j\varphi_k} \cdot e^{-j\frac{2\pi kn}{2 \cdot K}} \right] \tag{3.4}$$

mit $\cos(\omega T + \varphi) = \frac{1}{2}\left(e^{j\omega T+\varphi} + e^{-j\omega T+\varphi}\right)$ lässt sich die Formel weiter zusammenfassen. Wird zudem noch ein Parameter für die Amplitude der Harmonischen $A_k = 2 \cdot |S_k|$ eingeführt, von denen nur N für die Berechnung der Fourierreihe berücksichtigt werden soll, ergibt sich folgender Ausdruck:

$$s_{app} = X_0 + \sum_{k=1}^{N} A_k \cdot \cos(2\pi n + \varphi) \qquad \text{für} \qquad N \leq K - 1 \tag{3.5}$$

An dieser Stelle ist es wichtig, darauf zu achten, dass folgende Beziehung gilt: $N \leq K - 1$. So kann sichergestellt werden, dass die halbe Abtastfrequenz nicht enthalten ist. In der Formel 3.5 ist noch nicht berücksichtigt, über wie viele Perioden p das Signal aufgenommen wurde. Wenn das noch berücksichtigt wird, erhält man folgenden Ausdruck:

$$s = X_0 + \sum_{k=1}^{N} A_k \cdot \cos(\frac{2\pi kn}{p} + \varphi) \qquad \text{für} \qquad N < K \tag{3.6}$$

Das ist nötig, da zu Beginn davon ausgegangen wurde, dass die Fourierkoeffizienten über eine Periode bestimmt worden sind. Wenn allerdings die Fourierkoeffizienten über P Perioden bestimmt werden, sind im idealen Fall nur alle $n \cdot P$ Koeffizienten ungleich Null. Wenn N Harmonische Berechnet werden sollen, muss die Ober Grenze der Summe mit P multipliziert werden, sodass sich $N \cdot P$ ergibt.

$$s = X_0 + \sum_{k=1}^{N \cdot P} A_k \cdot \cos(\frac{2\pi kn}{P} + \varphi) \tag{3.7}$$

3.3.3. Abschätzung der Qualität der Ergebnisse

Um die Qualität der Approximation des Signals bewerten zu können, muss die Leistung der nicht berücksichtigten Oberschwingungen P_{ob} berechnet werden. Da die Fourierkoeffizienten bereits für die Berechnung des approximierten Signals ermittelt worden sind, können diese schon verwendet werden. Allgemein ergibt sich für K errechnete Amplituden der Oberschwingungen A_k und N für das Signal berücksichtigte Harmonische, folgende Formel:

$$P_{ob} = \frac{1}{2 \cdot (K-1)} \cdot \sum_{k=N+1}^{K-1} A_k^2 \tag{3.8}$$

Zur Kontrolle der Leistung der nicht berücksichtigten Schwingungen P_{harm} ist die Leistung der berücksichtigten Harmonischen bestimmt worden. Die Formel lässt sich für K ausgerechnete Amplituden A_k und N berücksichtige Harmonische wie folgt aufstellen:

$$P_{harm} = \frac{1}{2 \cdot (K-1)} \cdot \sum_{k=1}^{N} A_k^2 \tag{3.9}$$

Die Summe von P_{harm} und P_{ob} muss somit der Leistung des gleichanteilfreien Signals P_{ges} entsprechen.

3.4. Bereitstellung der Simulationsdaten

Für die Berechnung der im Verlauf des Kapitel dargestellten Parameter ist ein Matlab-Skript mit dem Namen „prepare_measurements“ entwickelt worden. Bei der Ausführung des Skripts wird der Benutzer aufgefordert, die Anzahl der Harmonischen anzugeben, mit denen das Ausgangssignal des Sensors approximiert werden soll, sowie die Anzahl der Perioden, über die aus den Oszilloskopdaten eine Fouriertransformation errechnet werden soll. Das Oszilloskopsignal wird ebenfalls auf die eingegebene Anzahl der Perioden reduziert und in der Datenstruktur abgelegt. Die maximal mögliche Eingabe ist an dieser Stelle 100. Die Bedeutung des Parameters mag zwar an dieser Stelle etwas schwierig zu verstehen sein, wird aber im Verlauf dieser Arbeit noch erläutert werden.
Die neu berechneten Variablen werden zusammen mit einigen alten Variablen in einer

Datenstruktur mit dem Namen „data“ gespeichert. Im Gegensatz zu der bisherigen Datenstruktur aus [7] sind jetzt Messergebnisse vom Radmessplatz als auch die mit dem Oszilloskop aufgezeichneten Daten in der Datenstruktur enthalten.
Die Datenstruktur wird in das Verzeichnis der jeweiligen Messreihe unter dem Namen „*Name der Messreihe*_for_simulation.mat“ abgespeichert. Für eine Messreihe mit dem Namen „2009_10_07_rmp_01“ lautet der Dateiname beispielsweise „2009_10_07_rmp_01_for_simulation.mat“. In der Tabelle 3.1 werden die in der Datenstruktur beschriebenen Variablen und deren Bedeutung erklärt.

Name der Komponenten	**Beschreibung**
measure_rmp	In dieser Datenstruktur sind die Daten enthalten, die mit dem Demonstrator aufgenommen wurden. Eine Beschreibung ist in Tabelle 3.2 zu finden
measure_scope	In dieser Datenstruktur sind die Daten enthalten, die zu Vergleichszwecken mit dem Oszilloskop aufgenommen wurden. Eine Beschreibung ist in Tabelle 3.3 zu finden
gain	In dieser Variable ist der Verstärkungsfaktor des Vorverstärkers abgelegt. Dieser ist in den Datenstrukturen „measure_rmp“ und „measure_scope“ noch nicht berücksichtigt und muss zur korrekten Darstellung der Signale aus den Messungen heraus gerechnet werden.
N	Anzahl der Koeffizienten, die nicht zu Null gesetzt werden.
shift_factor	Diese Variable enthält den Schiebefaktor, um die Verstärkerstufen der Hardware heraus zu rechnen. Dieser berechnet sich wie folgt: shift_factor = $\log_2(gain)$.
distance	Hier ist die Entfernung in Millimeter zwischen Sensor und Encoderrad bei der jeweiligen Messung enthalten.

Tabelle 3.1.: Beschreibung der ersten Ebene der Datenstruktur

Name der Komponenten	Beschreibung
u_diff	Die gemessene Differenzspannung der AMR Brücke über 2 Perioden mit 64 Werten pro Periode. Dieses Feld ist den vorhandenen Messreihen entnommen.
coefficients	Berechnung der Koeffizienten mit Hilfe der Matlab Funktion fft() über 2 Perioden. Der Gleichanteil des Signals wird durch den ersten Koeffizienten repräsentiert.
P_harm	In dieser Variablen wird die Leistung der zu Approximation des Signals berücksichtigten Harmonischen hinterlegt.
P_ob	Hier wird die Leistung der zu Approximation des Signals nicht berücksichtigten Harmonischen hinterlegt.
hd_lut	Klirrfaktor in Prozent, mit Hilfe der Wurzel-Tabelle berechnet. Dieser Wert wird aus der vorhandenen Datenstruktur übernommen.
mag	Beträge der ersten bis fünften Harmonischen zum Vergleich. Diese Daten sind ebenfalls aus der vorhandenen Datenstruktur übernommen worden.
periodes	Anzahl der Perioden, über die die Samples in dem Vektor u_diff enthalten sind.

Tabelle 3.2.: Beschreibung der Datenstruktur „measure_rmp“

Name der Komponenten	Beschreibung
u_diff	Die gemessene Differenzspannung der AMR Brücke, aufgenommen mit einer Abtastfrequenz von 250 kHz über eine einstellbare Anzahl von Perioden
coefficients	Berechnung der Koeffizienten mit Hilfe der Matlab Funktion fft() unter Verwendung von u_diff. Die Anzahl der Koeffizienten ist in der Variable N enthalten.
P_harm	In dieser Variablen wird die Leistung der zu Approximation des Signals berücksichtigten Harmonischen Schwingungen hinterlegt.
P_ob	Hier wird die Leistung der zu Approximation des Signals nicht berücksichtigten harmonischen Schwingungen hinterlegt.
periodes	Anzahl der Perioden, über die die Samples in dem Vektor u_diff enthalten sind.

Tabelle 3.3.: Beschreibung der Datenstruktur „measure_scope“

4. Nachbildung der Festkommaarithmetik in Matlab

4.1. Einführung

In diesem Kapitel soll ein Konzept vorgestellt werden, das einen Lösungsweg aufzeigt, die Festkommaberechnungen mit Matlab nachzubilden. Das ist erforderlich, da bei der Realisierung des Radmessplatzes ein Mikrocontroller der MSP430-Familie ausgewählt worden ist, der intern keine Gleitkommawerte verarbeiten kann.
Damit solche Software- oder auch Hardwarekomponenten mit Matlab simuliert oder auch nachgebildet werden können, ist in den Matlab-Versionen ab R14 die Fixed-Point-Toolbox enthalten. In dieser Arbeit wird mit der Version V2.3, die in der Matlab Version R2008a enthalten ist, gearbeitet. Mit ihr ist es unter anderem möglich, Festkomma-Datentypen mit Wortlängen von maximal 65.535 Bit zu definieren, Einstellungen zur Ausführung der Fixed-Point-Arithmetik lokal und global vorzunehmen, sowie logische und bitorientierte Operatoren zu verwenden. Eine kurze Beschreibung der Fixed-Point-Toolbox ist im Anhang B.1 angefügt.

4.2. Nachbildung des MSP430 Hardware Multiplizierers

Für die Berechnung des Klirrfaktors in dem Simulationsprogramm ist der Hardware-Multiplizierer vorgesehen worden. Bei dem Hardware-Multiplizierer handelt es sich um eine Peripherie-Einheit des MSP430, die sich außerhalb der CPU befindet. Er ist unter anderem in dem Mikrocontroller des Typs MSP430x1611 vorhanden.
Die Funktionsweise des Hardware-Multiplizierers kann in dem „Applikation-Report“ [2] nachgelesen werden. Die Wortbreiten der Register sind dem Blockschaltbild des Hardware-Multiplizierers entnommen worden, das in Abbildung 4.1 dargestellt ist.

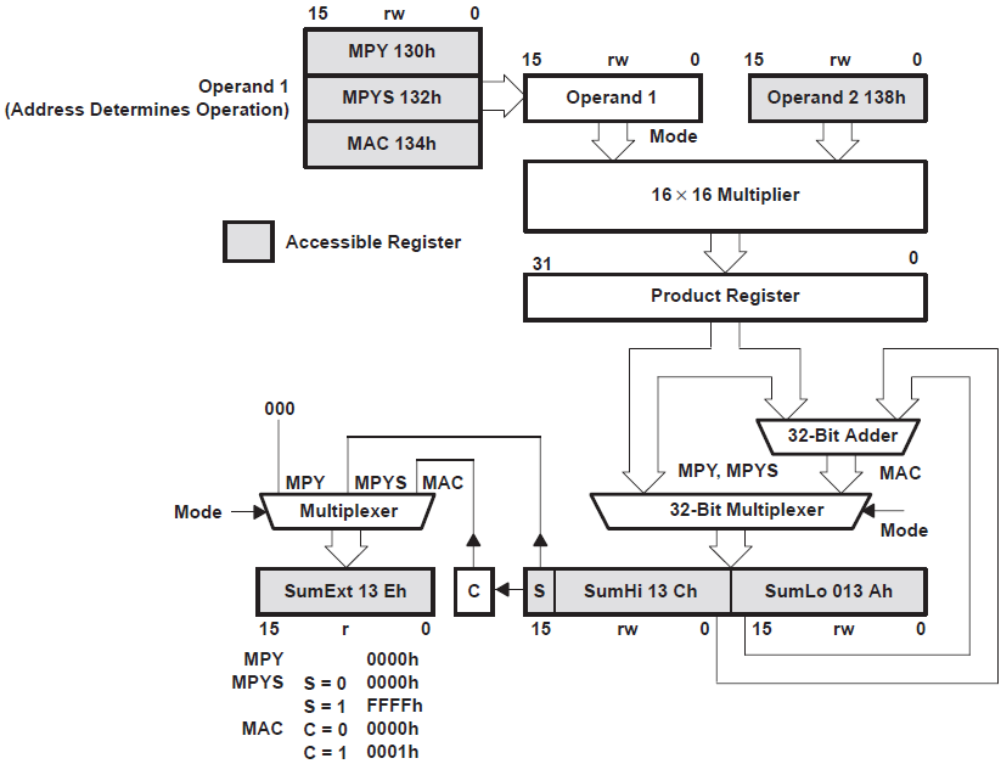

Abbildung 4.1.: Blockschaltbild des Hardware Multiplizierers des MSP430 [2]

Für die automatisierte Wortbreitenbegrenzung in Matlab eignet sich am besten das „fimath-Objekt“. Wie in der Abbildung 4.1 zu sehen ist, können mit dem Hardware-Multiplizierer zwei 16-Bit-Variablen multipliziert werden. Das Ergebnis der Multiplikation ist 32 Bit breit und wird im „Product Register“ zur Verfügung gestellt. Diese kann entweder mit dem Wert des Ergebnis-Registers addiert werden oder direkt in das Ausgaberegister geschrieben werden, das eine Länge von 32 Bit aufweist. Der Inhalt des „Product Register“ kann entweder in das Ausgaberegister geschrieben werden, oder es besteht die Möglichkeit eine Summe über mehrere Multiplikationsergebnisse mit dem internen Addierer zu berechnen. Wenn Nachkommastellen benötigt werden, soll die Berechnung dafür durch richtige Skalierung der Werte erfolgen. Den bestimmten Datentypen sollen jedoch keine feste Anzahl von Nachkommastellen zugeordnet werden. Deshalb wird der Parameter „FractionLength“ zu Null gesetzt. Der Overflow-Modus ist auf „wrap“ gesetzt worden, da das dem Überlaufverhalten einer Variablen in einem C-Programm entspricht. Falls ein Wert zu groß ist, um ihn in ein 16-Bit-Register zu schreiben, muss dieser geschoben werden. Dass entspricht immer einem Abrunden des Wertes; deshalb ist die Eigenschaft RoundMode auf „floor“ gesetzt. Aus den eben beschriebenen Bedingungen kann folgendes „fimath-Objekt“ erstellt werden:

```
F = fimath;
```

```
F.ProductMode           = 'KeepLSB';
F.ProductWordLength     = 32;
F.MaxProductWordLength  = 32;
F.ProductFractionLength = 0;
F.SumMode               = 'KeepLSB';
F.SumWordLength         = 32;
F.SumFractionLength     = 0;
F.OverflowMode          = 'wrap';
F.RoundMode             = 'floor';
F.CastBeforeSum         = false;
```

4.3. Nachbildung der verwendeten Datentypen

Damit in der Simulation die Standarddatentypen wie zum Beispiel char, long und short verwendet werden können, müssen dafür zunächst numerictype Objekte angelegt werden. Diese sind ganzzahlig also ohne Nachkommastellen definiert worden. In Tabelle 4.1 sind die nachgebildeten Datentypen dargestellt:

Datentyp	**Vorzeichen**	**Wortbreite**	**Nachkommastellen**
long	ja	32	0
long64	ja	64	0
short	ja	16	0
char	ja	8	0

Tabelle 4.1.: Übersicht der implementierten Datentypen

Vorzeichenlose Datentypen sind nicht nachgebildet, da diese in der Simulation keine Anwendung finden, da die Zwischenergebnisse vorzeichenbehaftet sind.
Außerdem sind noch drei spezielle Datentypen, die nicht in Programmiersprachen bekannt sind, definiert worden. Das ist zum einen der 24-Bit-Datentyp „bit24“, der zur Reduzierung der Wortbreiten der Register benötigt wird. Der zweite Datentyp ist der einzige vorzeichenlose Datentyp mit der Bezeichnung t_ADC, dessen Breite eingestellt werden kann. Er beinhaltet die simulierten Samplewerte, die auf der Plattform mit dem internen ADC erfasst werden. Der dritte Datentyp hat den Namen t_LUT. Von diesem Datentyp sind die Sinus- und Kosinus-Tabellen für die Berechnung der DFT abgelegt. Die Datentypen sind angelegt worden, um die Warnfunktion der Fixed-Point-Toolbox auszunutzen, die eventuelle Überläufe beim Füllen der Wertetabellen sowie bei der Berechnung der Samplewerte signalisiert. Eine Übersicht über die besonderen Datentypen bietet Tabelle 4.2:

Datentyp	Vorzeichen	Wortbreite	Nachkommastellen
bit24	ja	24	0
tADC	nein	einstellbar	0
tLUT	ja	einstellbar	0

Tabelle 4.2.: Übersicht der besonderen Datentypen

4.4. Definition der Datentypen in Matlab

Die Verwendung der Fixed-Point-Toolbox ist mit vielen Einstellungen verbunden, die in jeder Funktion, in der die Fixed-Point-Variablen verwendet werden sollen, vorgenommen werden müssen. Um die Datentypen und definierten arithmetischen Eigenschaften global definieren zu können, ist eine Funktion entwickelt worden, die Datentypen in einer Struktur zur Verfügung stellt. Der Aufbau der Struktur und die Bedeutung der einzelnen Strukturelemente ist in Tabelle 4.3 dargestellt:

Name der Komponenten	Beschreibung
Properties	In dieser Fixed-Point Struktur sind die Eigenschaften eines Fixed-Point-Objekts zur möglichst guten Nachbildung der Recheneinheit festgelegt.
long	numerictype Objekt für den Datentyp long
long64	numerictype Objekt für den Datentyp long64
bit24	numerictype Objekt für den Datentyp bit24
short	numerictype Objekt für den Datentyp short
char	numerictype Objekt für den Datentyp char
tADC	numerictype Objekt für die Nachbildung des Analog Digital Umsetzers
tLUT	numerictype Objekt Nachbildung des Real und Imaginärteils eines Zeigers für die DFT Berechnung
N_LUT	Wortbreite des Datentypes tLUT
N_ADC	Wortbreite des Datentypes tADC

Tabelle 4.3.: Beschreibung der Rückgabestruktur der Funktion „define_Types“

Für den Datentyp t_LUT wird intern zusätzlich ein Vorzeichenbit generiert, da im Gegensatz zu der Implementierung aus [7] eine ganze Periode eines Sinus bzw. eines Kosinus hinterlegt ist. Damit die Simulationsergebnisse mit den Messergebnissen des Radmessplatzes vergleichbar sind, muss das Vorzeichenbit für die negative Halbwellen hinzugefügt werden. Bei

einer Auflösung der Sinus- und Kosinus Tabelle von 10 Bit, wie sie bei der Demonstrator-Software verwendet worden ist, wird ein 11-Bit-Datentyp erzeugt, in dem 10 Bit für die Daten und ein Bit für das Vorzeichen verwendet wird.
Diese Implementierung ist gewählt worden, da in jedem beliebigen Programm einfach Fixed-Point-Datentypen erstellt werden können. Soll an einem Datentypen eine Änderung vorgenommen werden, muss der Quellcode nur in der Funktion define_Types geändert werden. Dadurch wird vermieden, dass in einer Funktion die Änderungen nicht eingepflegt werden und es deshalb zu fehlerhaften Ergebnissen kommt. Der wesentliche Grund ist allerdings, dass wenig Code geschrieben werden, muss um zum Beispiel eine Variable d vom Datentyp „short“ und mit Null initialisiert zu erstellen. In dem kleinen Beispielprogramm ist eine Lösung für dieses Problem dargestellt:

```
fix = define_Types;
d = fi(0,fix.short,fix.Properties);
```

Wenn die Datentypen t_ADC und t_LUT jeweils mit einer Auflösung von zehn Bit verwendet werden sollen, kann ein Beispielprogramm so aussehen:

```
fix = define_Types(10,10);
e = fi(0,fix.t_ADC,fix.Properties);
f = fi(0,fix.t_LUT,fix.Properties);
```

Mit diesen Variablen lassen sich alle arithmetischen Operationen, wie zum Beispiel die Multiplikation und die Addition wie bei Verwendung der Gleitkomma-Arithmetik ausführen.

5. Implementierung der Klirrfaktorberechnung

5.1. Einleitung

In diesem Kapitel sollen zwei verschiedene Verfahren zur Ermittlung des Klirrfaktors vorgestellt werden. Diese sollen dann unter Berücksichtigung der Ergebnisse des letzten Kapitels in einem Matlab Programm unter Verwendung von Variablen mit begrenzter Wortbreite berechnet werden. In diesem Zusammenhang müssen eventuelle Überläufe bzw. Leerläufe der Variablen beachtet werden. Das Ziel des Kapitels soll sein, dass das Programm den Klirrfaktor mit begrenzter Wortbreite und geringem Aufwand möglichst genau ermitteln kann.

5.2. Berechnung des Klirrfaktors

In diesem Abschnitt soll das Problem zunächst mathematisch untersucht werden und verschiedene Berechnungsmethoden ausgewählt und bewertet werden.
Die erste Berechnungsmethode ergibt sich unmittelbar aus der Definition des Klirrfaktors. Als Klirrfaktor wird „das Verhältnis der Oberwellen zur Grundwelle“ [9] bezeichnet. Das kann mathematisch durch folgende Formel ausgedrückt werden:

$$k = \sqrt{\frac{\sum_{k=2}^{\infty} |A_k|^2}{\sum_{n=1}^{\infty} |A_k|^2}} \tag{5.1}$$

Die obere Grenze der Summe ist nur von theoretischer Bedeutung, da diese nicht erreicht werden kann. In der Regel wird für ∞ eine endliche Zahl N eingesetzt. An dieser Stelle besteht die Gefahr, dass nicht alle relevanten Oberschwingungen berücksichtigt werden. Wenn die Zahl N wiederum sehr groß ist, kann die Berechnung bei Verwendung dieser Variante sehr viel Zeit in Anspruch nehmen. Im Rahmen der Arbeit Jegenhorst [7] ist abgeschätzt worden, dass fünf Harmonische zur Berechnung des Klirrfaktors ausreichend sind. In einem späteren Kapitel wird auf das nicht unerhebliche Auftreten der 7. und 8. Harmonischen eingegangen.

Mit der zweiten Variante wird versucht, die Abschätzung der relevanten Harmonischen zu umgehen. Durch Interpretation der Formel 5.1 ergibt sich, dass im Nenner des Bruches alle Harmonischen mit Ausnahme des Gleichanteils summiert werden. Aus den Grundlagen der Signal- und Systemtheorie ist zudem bekannt, dass die Summe der Betragsquadrate aller Fourierkoeffizienten, der Gesamtleistung des reellen Signals ohne Gleichanteil entspricht. Der entsprechende Nachweis ist im Anhang A.2 angefügt. Wird dieser Zusammenhang in der Gleichung 5.1 berücksichtigt, ergibt sich folgender Ausdruck:

$$THD = \sqrt{\frac{\sum_{n=2}^{\infty} A_k^2}{P_{ges}}} \tag{5.2}$$

Die Summe der Leistungen der Oberschwingungen P_{ob} kann auch als Differenz der Gesamtleistung P_{ges} und der Leistung der ersten Harmonischen P_1 geschrieben werden:

$$P_{ob} = P_{ges} - P_1 \tag{5.3}$$

Wird das in die Formel 5.2 eingesetzt, ergibt sich folgende alternative Formel zur Klirrfaktorberechnung:

$$THD = \sqrt{\frac{P_{ges} - P_1}{P_{ges}}} \tag{5.4}$$

Wenn die Leistung der ersten Harmonischen mit Hilfe der Amplitude A_k bestimmt werden soll, muss die Leistung der Kosinus-Schwingung berücksichtigt werden, die $\frac{1}{2}$ beträgt. Es ergibt sich dann folgender Ausdruck:

$$THD = \sqrt{\frac{P_{ges} - \frac{A_1^2}{2}}{P_{ges}}} = \sqrt{\frac{P_{ges} - 2 \cdot S_1}{P_{ges}}} \tag{5.5}$$

Beide Verfahren sollen in zwei unterschiedlichen Matlab-Funktionen implementiert werden.

Um die Verfahren besser auseinander zu halten, wird das erste Verfahren im Folgenden als „HD5-Methode“ (Gleichung 5.1) und das zweite Verfahren als „HDI-Methode“ (Gleichung 5.5) bezeichnet.

5.3. Implementierung der „HD5-Methode“

5.3.1. Implementierung der diskreten Fouriertransformation

Einleitung

Für die Anwendung der HD5-Methode ist es erforderlich die diskrete Fouriertransformation zu berechnen. Die Berechnung soll in einer extra Funktion ausgeführt und die benötigten Betragsquadrate der Harmonischen übergeben werden. Die Harmonischen sollen, wie auch der Klirrfaktor mit reduzierter Wortbreite ermittelt werden.
Im ersten Teil sollen zunächst die mathematischen Grundlagen erläutert werden, um im nächsten Teil die Skalierung der Werte zu beschreiben. Dieses ist erforderlich, damit die Beträge möglichst genau ermittelt werden können.

Mathematische Grundlagen

Die allgemeine Formel zur Berechnung der diskreten Fouriertransformation lautet:

$$\underline{S}_k = \frac{1}{M} \sum_{n=0}^{M-1} s[n] \cdot e^{-j\frac{2\pi kn}{M}} \tag{5.6}$$

Diese Formel kann auch als Matrix ausgedrückt werden. Im Gegensatz zur Gleichung 5.6 soll der Vorfaktor zu $\frac{1}{\sqrt{M}}$ gewählt worden. So wird erreicht, das der Vorfaktor auf die Fouriertransformation und auf die inverse Fouriertransformation gleichmäßig aufgeteilt wird. Wir die Matrix für die Leistungsberechnung quadriert, wird der dann ebenfalls der Faktor $\frac{1}{\sqrt{M}}$ berücksichtigt. Der Ausdruck sieht dann wie folgt aus:

$$\vec{S} = \frac{1}{\sqrt{M}} \cdot \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & e^{-j\frac{2\pi 1}{M}} & e^{-j\frac{2\pi 2}{M}} & \dots & e^{-j\frac{2\pi 1n}{M}} \\ 1 & e^{-j\frac{2\pi 2}{M}} & e^{-j\frac{2\pi 4}{M}} & \dots & e^{-j\frac{2\pi 2n}{M}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-j\frac{2\pi 1k}{M}} & e^{-j\frac{2\pi 2k}{M}} & \dots & e^{-j\frac{2\pi kn}{M}} \end{pmatrix} \cdot \vec{s} \tag{5.7}$$

Der Vorteil dieser Variante besteht darin, dass für eine Rücktransformation vom Frequenz- in den Zeitbereich die inverse Matrix F^{-1} gebildet werden muss.
Beim Aufstellen der Matrix stellt der Ausdruck $e^{-j\frac{2\pi kn}{M}}$ ein Problem dar, weil es mit der Fixed-Point-Toolbox nicht möglich ist mit komplexen Größen zu arbeiten. Aus diesem Grund ist die Matrix in Real- und Imaginärteil aufgeteilt, sodass sich folgende Matrix ergibt:

$$\vec{S} = \frac{1}{\sqrt{M}} \left[\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \cos(\frac{2\pi 1}{M}) & \cos(\frac{2\pi 2}{M}) & \dots & \cos(\frac{2\pi 1n}{M}) \\ 1 & \cos(\frac{2\pi 2}{M}) & \cos(\frac{2\pi 4}{M}) & \dots & \cos(\frac{2\pi 2n}{M}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(\frac{2\pi 1k}{M}) & \cos(\frac{2\pi 2k}{M}) & \dots & \cos(\frac{2\pi kn}{M}) \end{pmatrix} \cdot \vec{s} + \right.$$
$$\left. j \cdot \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \sin(\frac{2\pi 1}{M}) & \sin(\frac{2\pi 2}{M}) & \dots & \sin(\frac{2\pi 1n}{M}) \\ 1 & \sin(\frac{2\pi 2}{N}) & \sin(\frac{2\pi 4}{N}) & \dots & \sin(\frac{2\pi 2n}{M}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sin(\frac{2\pi 1k}{M}) & \sin(\frac{2\pi 2k}{M}) & \dots & \sin(\frac{2\pi kn}{M}) \end{pmatrix} \cdot \vec{s} \right] \quad (5.8)$$

Die Zahlenwerte des Sinus und Kosinus werden ohne den Vorfaktor $\frac{1}{\sqrt{M}}$ in einem Array des Datentyps t_LUT im Speicher abgelegt.
Für die Berechnung des Klirrfaktors sind nach der Formel 5.1 nur die Quadrate der Beträge der Harmonischen relevant, sodass Real- und Imaginärteil quadriert und dann addiert werden können. Auf das anschließende Radizieren ist deshalb verzichtet worden. Diese Rechenvorschrift kann auch mathematisch ausgedrückt werden:

$$|S|^2 = \Re(S)^2 + \Im(S)^2 \quad (5.9)$$

Ermittlung der Schiebefaktoren

Damit es während der Berechnung der DFT zu keinen Überläufen der Register kommt, werden vor dem Beginn der Berechnung die maximal möglichen Wortbreiten der Zwischenergebnisse ermittelt. Dieses Vorgehen ist für die Simulation notwendig, damit die Simulation für verschiedene Wortbreiten ausgeführt werden kann.
Bei der Berechnung der Schiebefaktoren soll das Ziel verfolgt werden, die zur Verfügung stehende Wortbreite des Zielregisters w_Z möglichst gut auszunutzen. Die Wortbreite des Ergebnisses kann mit der folgenden Formel für den schlimmsten anzunehmenden Fall (worst case), der beinhaltet, dass Werte dem größten darstellbaren Wert entsprechen, vorausgesagt werden. Bei der Multiplikation eines Samples, das in der Wortbreite w_{ADC} vorliegt, und eines Wertes aus der Look-Up Tabelle mit der Wortbreite w_{LUT} addieren sich diese. Zudem muss noch beachtet werden, dass M Multiplikationsergebnisse aufaddiert werden müssen. Für den Fall, dass in allen Registern der maximal darstellbare Wert abgelegt ist, entspricht das Ergebnis der Summe M-mal dem Wert. Das bedeutet, dass maximal $\log_2(M)$ zusätzliche Bit benötigt werden. Das kann mathematisch wie folgt ausgedrückt werden.

$$w_{max} \leq= w_{ADC} + w_{LUT} + \log_2(M) \tag{5.10}$$

Wenn die Anzahl der Samples N im Zweierkomplement dargestellt werden kann, ist Faktor $\log_2(M)$ Element der natürlichen Zahlen. Wenn dies nicht der Fall ist, muss der Wert aufgerundet werden. Das kann wiederum dazu führen, dass nicht die volle Wortbreite verwendet wird.
Der Schiebefaktor zur Berechnung der Real- und Imaginärteile einer Harmonischen kann mit den Informationen aus der Formel 5.10 einfach ausgerechnet werden:

$$shift_1 = w_{max} - w_Z \qquad \text{für} \qquad w_{max} > w_Z \tag{5.11}$$

Die Formel ist für den Fall, dass die Wortbreite der Ergebnisvariablen größer ist als die maximale Wortbreite nicht gültig, da vermieden werden soll, dass in den niederwertigsten Bits (LSB) Nullen hineingeschoben werden.
Wie in Gleichung 5.9 ersichtlich ist, müssen für die Real- und Imaginärteile in einem weiteren Schritt die Quadrate errechnet werden. Bei einer Multiplikation verdoppelt sich die Wortbreite im MC von 16 auf 32 Bit. Um nach der erneuten Multiplikation kein Ergebnis mit einer Wortbreite von 64 Bit zu erhalten, was auf einer MSP430-Plattform zu einer langen Rechenzeit führen würde, müssen die Zwischenergebnisse wieder auf eine Wortbreite von 16 Bit reduziert werden. Dabei soll so wenig Genauigkeit wie möglich verloren gehen. Deshalb sollen nur die tatsächlich verwendeten Bits herausgeschnitten und eventuell zu viel enthaltene LSB abgeschnitten werden. Die Abbildung 5.1 soll noch einmal schematisch die eben beschriebene Idee darstellen.

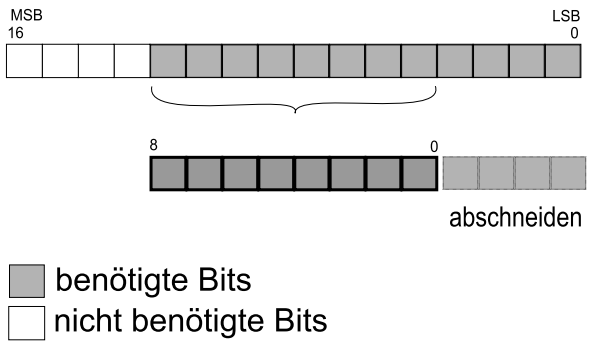

Abbildung 5.1.: Möglichkeit zur Reduzierung der Wortbreite mit geringem Genauigkeitsverlust

Zur Berechnung dieses Schiebefaktors muss zunächst die Anzahl der Bits ausgerechnet werden, die auf Grund der Größe des Ergebnisses nicht gesetzt sind. Dass kann mit Hilfe der Formel 5.11 durchgeführt werden. Als nächstes muss die Differenz zwischen der Registerwortbreite w_Z und der Wortbreite der Operandenregister w_{OP} aus Bild 4.1 gebildet werden.

Werden diese beiden Größen voneinander abgezogen, erhält man folgende Formel für den Schiebefaktor shift_q, der zur Reduzierung der Wortbreite auf 16 Bit erforderlich ist:

$$\text{shift}_q = w_Z - w_{max} - (w_Z - w_{OP}) = w_{OP} - w_{max} \tag{5.12}$$

Ist dieser Faktor negativ, entspricht das einem Schieben nach rechts. Wenn der Faktor positiv wird, soll auch hier nicht geschoben werden, um auch hier keine Folge von Nullen in den niederwertigen Bits zu erhalten. Als letztes ist nach der Quadrierung der Werte ein letztes Mal zu Schieben um zum einen die richtige Skalierung der Werte zu gewährleisten und zum anderen die Werte für die Berechnung der Summe der Gesamtleistung, sowie die Bildung der Leistung der Oberschwingungen vorzubereiten. Für die Berechnung des Klirrfaktors nach der ersten Variante spielt es keine Rolle, ob zu den Harmonischen ein konstanter Faktor multipliziert wird. Hier wird nur das Verhältnis zwischen der Summe der Oberschwingungen und der Gesamtleistung des Signals benötigt, sodass die konstanten Faktoren heraus gekürzt werden können. Es muss nur sichergestellt werden, dass es bei der Berechnung der Summen und bei der Übernahme des Wertes in eine Variable von einem anderen Datentyp zu keinen Überläufen kommen kann. Deshalb müssen in der Formel die Wortbreiten des Akkumulators w_{AKKU} und die Wortbreiten des Real- und Imaginärteils w_R berücksichtigt werden. Der Schiebefaktor s_2 wird daher durch folgende Formel beschrieben:

$$s_2 = w_R - w_{AKKU} \tag{5.13}$$

Datenflussdiagramm

In dem folgenden Datenflussdiagramm in Abbildung 5.2 ist der Algorithmus schematisch dargestellt. An einigen Stellen sind mehrere Wortbreiten angegeben. Diese sollen im Simulationsprogramm später eingestellt werden können.

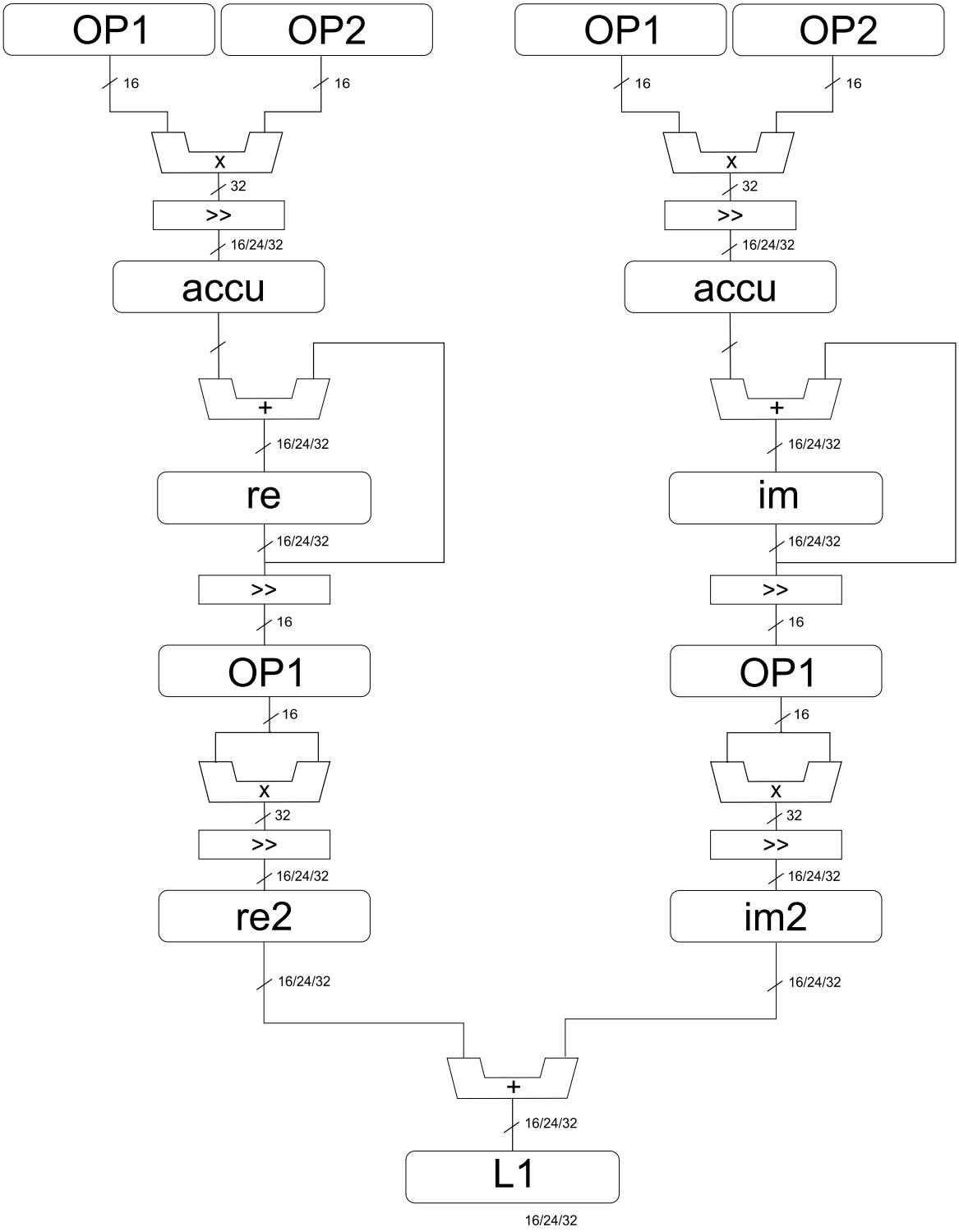

Abbildung 5.2.: Berechnung eines Fourierkoeffizienten der DFT

5.4. Implementierung der HDI-Methode

5.4.1. Ermittlung der Leistung des gleichanteilfreien Signals

Bei der HDI-Methode werden theoretisch unendlich viele Harmonische für die Berechnung des Klirrfaktors ermittelt. Die Abkürzung ergibt sich aus „**h**armonic-**d**istortion-**i**nfinite“(HDI). In der Literatur ist das Verfahren auch als „THD+N“ oder auch „THD+noise“ bekannt [10].
In ersten Schritt soll die Leistung des periodischen gleichanteilfreien Signals P_{ges} ermittelt werden. Die mittlere Leistung wird im zeitkontinuierlichen Bereich mathematisch durch die Formel

$$P_{ges} = \frac{1}{T} \cdot \int_0^T s_{\sim}(t)^2 dt \tag{5.14}$$

beschrieben werden. In der digitalen Signalverarbeitung wird eine bestimmte Abtastfrequenz verwendet, sodass das Integral bei Einhaltung des Abtasttheorems exakt durch eine Summe ersetzt werden kann und sich somit folgende Formel ergibt:

$$P_{ges} = \frac{1}{T} \cdot \sum_{n=0}^{\frac{T-\Delta T_a}{T_a}} s_{\sim}(n \cdot T_a)^2 \tag{5.15}$$

Mit T_a und $M = \frac{T_a}{T}$kann die Formel vereinfacht werden:

$$P_{ges} = \frac{1}{M} \cdot \sum_{n=0}^{M-1} s_{\sim}(n)^2 \tag{5.16}$$

Das gleichanteilfreie Signals kann auch als die Differenz aus dem gleichanteilbehafteten Signal s und dem Gleichanteil $s_{=}$ dargestellt werden:

$$s_{\sim} = s - s_{gl} \tag{5.17}$$

Wird das in der Formel 5.17 berücksichtigt, ergibt sich folgende Formel zur Berechnung der Leistung des Signals:

$$P_{ges} = \frac{1}{M} \cdot \sum_{n=0}^{M-1} (s(n) - s_{gl})^2 \tag{5.18}$$

Jetzt ergeben sich zwei Möglichkeiten, die Leistung zu errechnen:

1. Den Gleichanteil ausrechnen, von allen Samplewerten den Gleichanteil abziehen, das ermittelte Teilergebnis quadrieren und alle Teilergebnisse zum Gesamtergebnis aufaddieren.
2. Die Binomische Formel aus Gleichung 5.18 auflösen und weiter vereinfachen.

Um eine Entscheidung treffen zu können, welche Variante sich in der Praxis besser eignet, sollen beide Verfahren untersucht werden.

1. Möglichkeit: Gleichanteil von allen Samplewerten abziehen

Ermittlung des Gleichanteils Für die korrekte Bestimmung der Leistung der Harmonischen eines Signals, ist es wichtig, dass dieser vor der Berechnung bekannt ist, sodass er vor der Leistungsberechnung abgezogen werden kann.
Kann die Größe des Gleichanteil zum Beispiel aus schaltungstechnischen Aspekten bereits vorhergesagt werden, kann die Berechnung in diesem Fall auch entfallen. Für die Simulation ist der Gleichanteil berechnet worden, obwohl er ungefähr $U_{FS}/2$ beträgt. Durch die Berechnung wird erreicht, dass Abweichungen des Gleichanteils erkannt werden und das Simulationsergebis nicht verfälscht wird. Der Gleichanteil wird auch als arithmetischer Mittelwert eines Signals bezeichnet. Die Formel für den arithmetischen Mittelwert lautet:

$$s_{gl} = \frac{1}{M} \sum_{n=0}^{M-1} s(n) \tag{5.19}$$

Ermittlung der Schiebefaktoren bei der Berechnung des Gleichanteils In diesem Abschnitt soll der Schiebefaktor ermittelt werden, um den der Gleichanteil geschoben werden muss, um keine Überläufe der Variablen zu riskieren. Die Wortbreite hängt von der Auflösung des Analog-Digital-Umsetzers w_{ADC} sowie von der Anzahl der Samples pro Periode M ab. Die zu erwartende Wortbreite w_{gl} lässt sich durch folgende Formel ausdrücken:

$$w_{gl} = w_{ADC} + \log_2(M) \tag{5.20}$$

Wird dieser Faktor größer als die Wortbreite der Variablen w_v, muss ein Schiebefaktor $shift_g$ ausgerechnet werden, der durch die folgende Formel bestimmt wird:

$$shift_g = w_v - w_{gl} \tag{5.21}$$

Das Ergebnis muss nach der Bildung der Summe durch die Anzahl der Samples dividiert werden. Das ist durch ein Schieben des Registerinhalts nach rechts um $\log_2(M)$ Stellen realisiert worden.
In Abbildung 5.3 ist ein Blockschaltbild des Datenpfads dargestellt. In dem Schieberegister muss wie oben beschrieben um den Faktor $\log_2(M)$ geschoben werden. Die Wortbreite der Variablen kann in der Simulation auf die Werte 16 Bit, 24 Bit oder 32 Bit begrenzt werden. Im Blockschaltbild sind alle Werte angegeben worden.

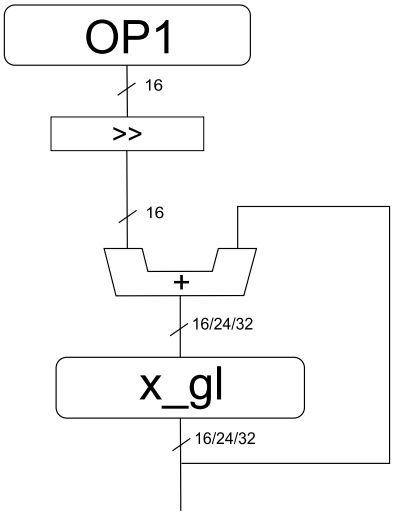

Abbildung 5.3.: Datenpfad zur Berechnung des Gleichanteils

Ermittlung der Schiebefaktoren bei der Ermittlung der Signalleistung Damit bei der Berechnung der Gesamtleistung die Variablen nicht überlaufen, müssen hier erneut Schiebefaktoren ermittelt werden. Auch hier muss die maximal zu erwartende Anzahl der Bits w_{max} vor der Berechnung der Schiebefaktoren ermittelt werden. Dafür lässt sich in diesem Fall folgende Formel aufstellen:

$$w_{max} = 2 \cdot w_{ADC} + \log_2(M) \tag{5.22}$$

Daraus lässt sich nach derselben Formel wie im vorherigen Punkt der Schiebefaktor $shift_{ges1}$ ausrechnen:

$$shift_{ges1} = w_v - w_{max} \qquad \text{für} \qquad shift_{ges1} \leq 0 \tag{5.23}$$

Das Schieben um diesen Faktor soll nur erfolgen, wenn dieser kleiner als Null ist. Das entspricht einem Schieben nach rechts. Somit ist gewährleistet, dass bei der Bildung der

Summe keine Überläufe entstehen und in die niederwertigen Bits (LSB) keine Folge von Nullen geschoben wird. Um eine richtige Skalierung der Leistung der ersten Harmonischen und der Leistung des Signals zu erreichen, sind bei unterschiedlichen Auflösungen des Analog-Digital-Umsetzers w_{ADC} und der Sinus- und Kosinus- Tabellen w_{LUT} weitere Multiplikationsfaktoren fak_{LUT} und fak_{ADC} eingeführt worden, die dafür sorgen, dass beide Operatoren die selbe Anzahl an gedachten Nachkommastellen aufweisen. Sie lassen sich durch folgende Formel bestimmen:

$$fak_{LUT} = w_{ADC} - (w_{LUT} - 1) \quad \text{für} \quad w_{ADC} \geq (w_{LUT} - 1) \quad \text{sonst} \quad 0 \tag{5.24}$$

$$fak_{ADC} = (w_{LUT} - 1) - w_{ADC} \quad \text{für} \quad w_{ADC} < (w_{LUT} - 1) \quad \text{sonst} \quad 0 \tag{5.25}$$

Dieses Vorgehen ist nötig, da die Berechnung der Signalleistung unabhängig von den Sinus- und Kosinus-Tabellen ist. Durch die Schiebefaktoren wird erreicht dass die Leistung der ersten Harmonischen sowie die Leistung des Signals richtig skaliert sind.
Bevor die Schiebeoperation durchgeführt werden kann, muss überprüft werden ob es dabei zu keinen Überläufen der Variable kommen kann. Das kann durch die vorausgesagte maximale Wortbreite unter Berücksichtigung des Schiebefaktors $shift_{ges1}$ geschehen. Dafür ist folgende Bedingung aufgestellt worden

$$w_v < w_{max} - shift_{ges1} + 2 \cdot fak_{ADC} \tag{5.26}$$

Wenn die Bedingung erfüllt ist, soll der Schiebebefehl um den Faktor $2 \cdot fak_{ADC}$ an dieser Stelle ausgeführt werden und der Schiebefaktor $shift_{ges2}$ ist Null. Wenn die Wortbreite nicht ausreicht, soll sich mit dem Schiebefaktor $shift_{ges2}$ gemerkt werden, dass dieser Faktor an dieser Stelle nicht berücksichtigt werden kann und daher bei der Berechnung der Leistung der ersten Harmonischen zusätzlich um den Faktor $shift_{ges2} = -2 \cdot fak_{ADC}$ geschoben werden muss.
Bei der Berechnung der Leistung der ersten Harmonischen müssen sowohl $shift_{ges1}$ als auch $shift_{ges2}$ berücksichtigt werden. Zur besseren Übersicht ist ein neuer Schiebefaktor eingeführt worden, der sich wie folgt zusammensetzt:

$$shift_{ges} = shift_{ges1} + shift_{ges2} \tag{5.27}$$

Das Blockschaltbild in Abbildung 5.4 zeigt auch hier wieder die Vorgehensweise bei der Berechnung auf.

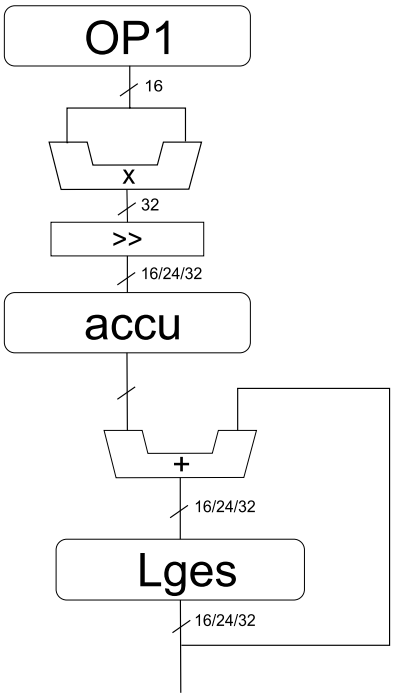

Abbildung 5.4.: Datenpfad zur Berechnung der Gesamtleistung des Signals

2. Möglichkeit: Entwicklung über binomische Formel

Auflösen der binomischen Formel und Interpretation der Ergebnisse In diesem Abschnitt soll der Lösungsansatz verfolgt werden, die binomische Formel in der Gleichung 5.18 aufzulösen. Wird die binomische Formel ausgeschrieben, ergibt sich folgende Gleichung:

$$P_{ges} = \frac{1}{M} \cdot \sum_{n=0}^{M-1} (s(n) - s_{gl})^2 = \frac{1}{M} \cdot \sum_{n=0}^{M-1} (s(n)^2 - 2 \cdot s(n) \cdot s_{gl} + s_{gl}^2) \tag{5.28}$$

Die Summe kann in mehrere Summen aufgeteilt werden, sodass sich folgender Ausdruck ergibt:

$$P_{ges} = \frac{1}{M} \left[\sum_{n=0}^{M-1} s^2(n) - 2 \sum_{n=0}^{M-1} s(n) \cdot s_{gl} + \sum_{n=0}^{M-1} s_{gl}^2 \right] \tag{5.29}$$

Der Gleichanteil ist immer konstant, deshalb ergibt die Summe

$$\sum_{n=0}^{M-1} s_{gl}^2$$

den Wert $M \cdot s_{gl}^2$. Wird das in der Gleichung 5.29 berücksichtigt, ergibt sich folgender Ausdruck:

$$P_{ges} = \frac{1}{M} \cdot \sum_{n=0}^{M-1} s^2(n) - 2 \cdot s_{gl} \cdot \frac{1}{M} \cdot \sum_{n=0}^{M-1} s(n) + \frac{1}{\not{M}} \cdot \not{M} \cdot s_{gl}^2 \tag{5.30}$$

In der Formel tritt der Term

$$s_{gl} = \frac{1}{M} \sum_{n=1}^{M-1} s(n) \tag{5.31}$$

auf. Dieser entspricht der Formel zur Berechnung des arithmetischen Mittelwerts des Signals aus Gleichung 5.19. Wird diese Erkenntnis in der Gleichung 5.30 berücksichtigt, erhält man folgenden Ausdruck:

$$P_{ges} = \frac{1}{M} \cdot \sum_{n=0}^{M-1} s^2(n) - 2 \cdot {s_{gl}}^2 + {s_{gl}}^2 = \frac{1}{M} \cdot \sum_{n=0}^{M-1} s^2(n) - {s_{gl}}^2 \tag{5.32}$$

Wenn diese Gleichung verwendet wird, bietet sich der Vorteil, dass die Leistung der Summe des Signals mit Gleichanteil berechnet werden kann. Der Nachteil besteht darin, dass der Gleichanteil quadriert werden muss und dadurch weitere Ungenauigkeiten in die Berechnung einfließen. Die Leistung des gleichanteilfreien Signals kann im Nachhinein durch die Subtraktion des Quadrats des Gleichanteils berechnet werden.

Ermittlung der Schiebefaktoren für den Gleichanteil Die Schiebefaktoren zur Berechnung des Gleichanteils können in dem Absatz 5.4.1 nachgelesen werden. Die Division durch N ist für dieses Verfahren in 2 Stufen aufgeteilt worden. An der ersten Stufe wird so weit geschoben, bis der Inhalt des Registers, in dem der Gleichanteil gespeichert ist, in eine 16 Bit Variable übernommen werden kann. Für den Schiebefaktor kann folgende Gleichung aufgestellt worden:

$$shift_g = 16 - w_{gl} - 1 \qquad \text{für} \qquad shift_g \leq 0 \tag{5.33}$$

Um das Ergebnis, das mit 32 Bit Genauigkeit vorliegt in eine Variable mit einer Wortbreite w_x zu übernehmen, muss folgender Schiebefaktor $shift_{gl2}$ verwendet werden:

$$shift_{gl2} = -shift_{ges1} + 2 \cdot shift_{gl} - (\log_2(M) + 2 \cdot shift_g) \tag{5.34}$$

Der letzte Term resultiert aus der Aufteilung der Division durch M. Bei der Berechnung des Quadrats verdoppelt sich die Wortbreite, deshalb muss um den Faktor $2 \cdot \log_2(M)$ nach rechts geschoben werden. Da vor der Quadrierung bereits um den Faktor $shift_g$ nach rechts geschoben worden ist, muss dieser von $\log_2(M)$ abgezogen[1] werden, sodass nur noch um den Faktor $2 * (\log_2(M) + shift_g)$ geschoben werden muss.

Ermittlung der Schiebefaktoren für die Leistungsberechnung Die Schiebefaktoren für die Leistungsberechnung sind mit der vorherigen Berechnung identisch. Die Berechnung der weiteren Schiebefaktoren kann in Abschnitt 5.4.1 nachgelesen werden.

5.4.2. Passende Skalierung der Leistung der 1. Harmonischen

Im Gegensatz zu der HD5-Methode ist es hier zwingend erforderlich, dass die Leistung der ersten Harmonischen und die Leistung des Signals mit demselben Skalierungsfaktor versehen werden, damit sich dieser heraus kürzt. Daher werden hier auch die Multiplikationsfaktoren verwendet, die gewährleisten sollen, dass bei der Multiplikation eines Wertes aus der Sinus- und Kosinus-Tabelle und eines Samples die höchstwertigen Bits (MSB) an der selben Position der 16 Bit Variable stehen. So wird an dieser Stelle die Multiplikation mit einem konstanten Faktor vermieden. Der Funktionsablauf ist danach identisch mit dem Funktionsablauf für die Berechnung der HD5-Methode. Allerdings muss der Schiebefaktor $shift_2$ anders errechnet werden, um eine richtige Skalierung der Ergebnisse zu erhalten.
In dem Schiebefaktor muss Faktor $shift_{ges}$, um den die Leistung des Signals zusätzlich dividiert worden ist, enthalten sein, um dort Überläufe zu vermeiden. Außerdem muss jeweils durch Addition einer Eins berücksichtigt werden, dass laut Formel 5.5 der Wert am Ende verdoppelt werden muss und ein weiteres Bit für das Vorzeichen des Analog-Digital-Umsetzers erforderlich ist. Es gibt sich daraus folgende Formel:

$$shift_2 = w_{max} - w_{AKKU} + shift_{ges} + 2 \tag{5.35}$$

[1] da der Faktor $shift_g$ negativ angesetzt worden ist entspricht das in diesem Fall einer Addition

6. Radizieren mit reduziertem Aufwand

6.1. Einführung und Spezifikationen

6.1.1. Einführung

In diesem Kapitel soll das Radizieren beschrieben werden, das bei der Berechnung des Klirrfaktor erforderlich ist. Das Radizieren stellt eine sehr rechenintensive Operation dar und kann für einen Entwicklungspunkt $x_0 = 0$ durch folgende Taylorreihe angenähert werden:

$$f(x) = 1 + \frac{1}{2} \cdot x - \frac{1}{8} \cdot x^2 + \frac{3}{48} \cdot x^3 - \frac{15}{2304} \cdot x^4 \ldots \tag{6.1}$$

Die Reihenentwicklung ist jedoch aufwendig zu berechnen. Dabei besteht die Gefahr, dass der Mikrocontroller den Funktionswert in der vorgegebenen Zeit nicht berechnen kann und durch einen leistungsfähigeren Mikrocontroller ersetzt werden muss. Eine andere Variante, die sich schneller ausführen lässt, ist die Ablage der Funktionswerte im Programmspeicher des Mikrocontrollers. Der Bedarf an Speicherplatz ist zwar dadurch größer, aber es muss nur der entsprechende Wert aus dem Speicher ausgelesen werden. Hierbei ist es möglich, als Feldindex das Ergebnis der Division zu benutzen und an der entsprechenden Speicherstelle den Funktionswert des bestimmten x-Wertes zu hinterlegen.

6.1.2. Die bisherige Ermittlung eines Funktionswertes

Dieses Verfahren ist im Rahmen der Diplomarbeit Jegenhorst [7] auch verwendet worden. Dort man sich für eine Wurzel-Tabelle mit 256 Zwischenwerten x, die linear skaliert sind, entschieden worden. Konkret ist vor der Division die Summe der Oberschwingungen um 10 Stellen nach links geschoben worden, damit das Ergebnis größer als Null ist. Das entspricht einem Multiplikationsfaktor von 1024. Die Wurzel-Tabelle ist mit den Werten der folgenden Funktion gefüllt worden:

$$THD_{\%} = 100 \cdot \sqrt{\frac{1}{1024} \cdot x} = \sqrt{\frac{100^2}{1024} \cdot x} \tag{6.2}$$

Wie Abbildung 6.1 zeigt, liegt der kleinste Wert der Tabelle bei 3%. Der interessante Bereich für diese Problemstellung sind die kleinen Klirrfaktorwerte, da der Sensor in diesem Fall richtig eingebaut ist. Daher ist diese Variante nicht sehr gut geeeignet.

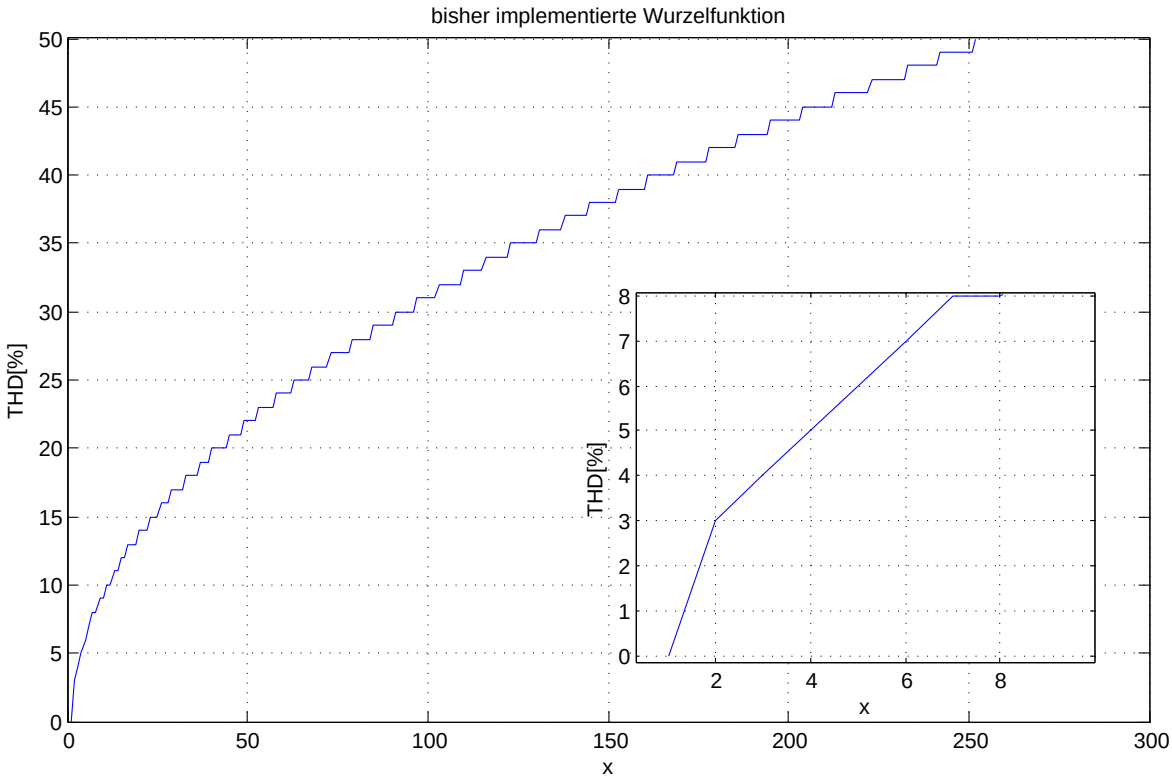

Abbildung 6.1.: In der Arbeit von Jegenhorst [7] verwendete Wurzelfunktion

6.1.3. Spezifikationen

Anhand der Messdaten, die in der Arbeit Jegenhorst [7] enthalten sind, wurden folgende neue Spezifikationen für die zukünftige Wurzel-Tabelle erarbeitet:

- Die neue Tabelle darf nicht mehr Werte als die alte Tabelle enthalten sein. Das bedeutet, dass nicht mehr als 256 Werte in der neuen Tabelle enthalten sein dürfen.
- Aus der Tabelle sollen die Ergebnisse als Prozentangabe entnommen werden können.

- Die Tabelle soll einen Wertebereich von 0 bis 31 % abdecken. Wobei der Klirrfaktor in diesem Berech mit einer Genauigkeit von $\pm 1\%$ ermittelt werden soll.

Im nächsten Abschnitt soll die Größe der Wurzel-Tabelle ermittelt werden, die erforderlich ist, um die Spezifikationen einzuhalten.

6.2. Analyse der neuen Spezifikationen

In diesem Abschnitt sollen die Auswirkungen der neuen Spezifikationen untersucht werden. Es soll außerdem berechnet werden, mit welcher Auflösung ein Ergebnis vorliegen muss, um das Ergebnis mit der geforderten Auflösung von 1% ermitteln zu können.
Damit die Lösung in dem Wertebereich von 0% bis 100% dargestellt werden kann, muss der Wurzelausdruck, dessen Ergebnis immer zwischen 0 und 1 liegt, mit 100 multipliziert werden.

$$THD_{\%} = 100 \cdot \sqrt{x} = \sqrt{100^2 \cdot x} \tag{6.3}$$

In einer weiteren Randbedingung ist festgehalten, dass die neue Wurzelfunktion bis 31% definiert sein soll. Wird dieser in die Formel 6.3 eingesetzt und die Gleichung nach x aufgelöst, erhält man das folgende Ergebnis für den größten x-Wert:

$$31\% = \sqrt{100^2 \cdot x_{max}} \Rightarrow x_{max} = \frac{31^2}{100^2} = 0,09 \tag{6.4}$$

Damit die Ergebnisse der Division die Position des Wertes in der Tabelle liefern, muss die minimale Schrittweite sowie der Minimal- und Maximalwert im Zweierkomplement dargestellt werden. Der Minimalwert stellt kein Problem dar, da dieser Null ist. Für den Maximalwert ist der nächst größere Wert relevant, der im Zweierkomplement dargestellt werden kann, $x_{max} = 0,125$ das entspricht $x_{max} = 2^{-3}$.
Als letztes muss die Anzahl der Zwischenwerte Z bestimmt werden. Um möglichst wenig Speicher für die Tabelle zu verwenden, soll die maximale Schrittweite Δx ermittelt werden. Diese muss kleiner als 1,5% sein, da bei diesem Wert die Entscheidungsgrenze für das Runden von Werten liegt. Das kann mathematisch mit folgender Formel beschrieben werden:

$$1,5\% < \sqrt{100^2 \cdot x_{min}} \Rightarrow x_{min} < \frac{1.5^2}{100^2} = 2,25 \cdot 10^{-4} \tag{6.5}$$

Von diesem Wert muss der Logarithmus dualis gebildet werden, um einen passenden Wert zu ermitteln, der im Zweierkomplement dargestellt werden kann:

$$ld(\Delta x) = \log_2(0,000225) = -12,11 \hookrightarrow -13 \tag{6.6}$$

Das ergibt eine maximale Schrittweite von $\Delta x = 2^{-13} = 0,000122$ um die Spezifikationen einzuhalten.

Der Wert stellt gleichzeitig auch den Schiebefaktor dar, um den die Summe der Oberschwingungen nach links geschoben werden muss, bevor die Division durchgeführt wird. Zudem ist jetzt gewährleistet, dass das Ergebnis der Division bei einem Klirrfaktor, der größer ist als 1% ungleich Null ist.

Aus den bisher in diesem Abschnitt gewonnen Ergebnissen kann die Anzahl der Werte ermittelt werden, die mit dem Verfahren aus [7] benötigt werden würden:

$$n_{max} = \frac{2^{-3}}{2^{-13}} = 2^{-3+13} = 2^{10} = 1024 \tag{6.7}$$

Das ergibt eine Wortbreite des Ergebnisses von $log_2(1024) = 10$ Bit. Der Verlauf dieser Kennlinie ist in der Abbildung 6.2 dargestellt.

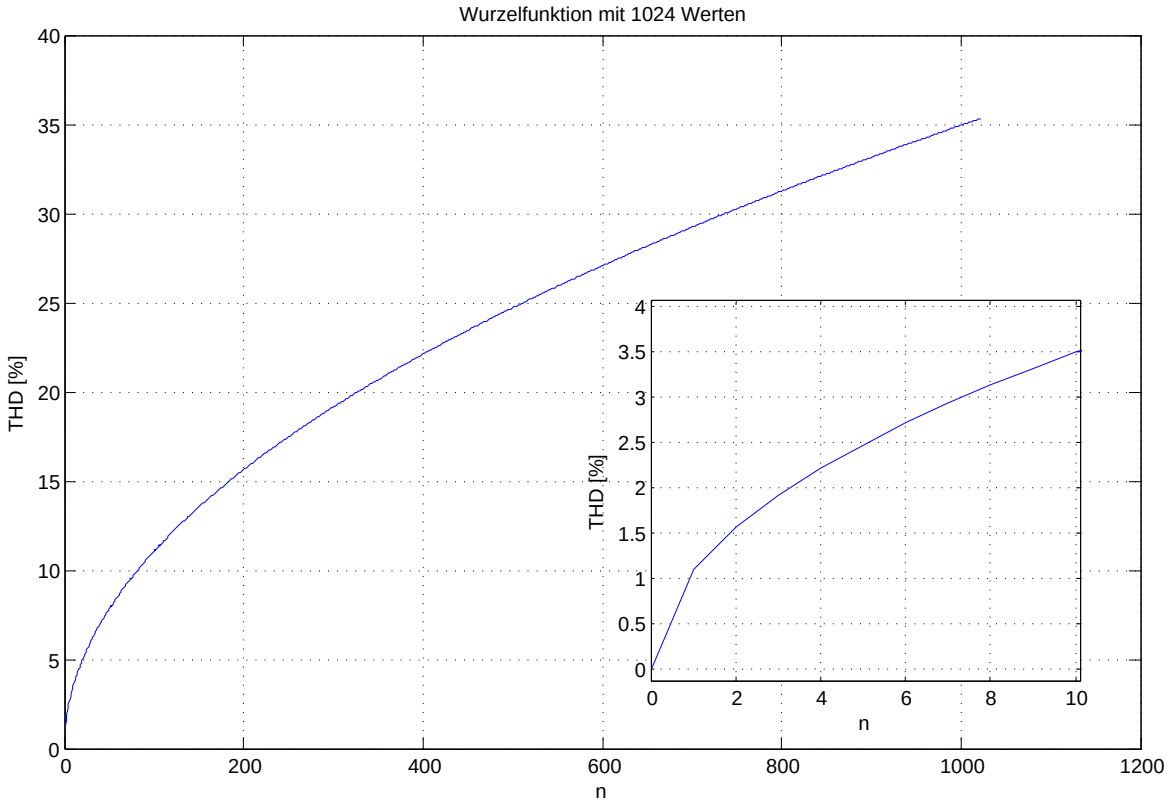

Abbildung 6.2.: Theoretisch erforderliche Wurzelfunktion um geforderte Genauigkeit zu erreichen

Da die Anzahl der Werte erhöht werden muss, was ein Widerspruch zu der aufgestellten Spezifikation darstellt, müssen Verfahren entwickelt werden um den Verlauf der Kennlinie anders zu codieren um weniger Zwischenwerte zu benötigen.

6.3. LUT-Methode mit wechselnden Tabellen

Um den Speicherbedarf zu reduzieren, ist der Ansatz verfolgt worden, die Tabelle in mehrere kleinere Tabellen zu unterteilen, wobei die Schrittweite Δx in jeder Tabelle variiert. Dafür wird zunächst die Steigung der Funktion zwischen zwei x-Werten ausgerechnet. Mathematisch handelt es sich dabei um eine numerische Differentiation. Wird die Steigung zwischen zwei Werten s durch die maximale Schrittweite Δx geteilt und von diesem Ergebnis der Logarithmus duales gebildet, erhält man folgende Formel für die möglichen Schiebefaktoren, um die das Divisionergebnis g geschoben werden kann, um eine Auflösung der Prozentskala von einem Prozent zu erreichen:

$$g = ld\left(\frac{\Delta x}{s}\right) \tag{6.8}$$

Da nur das Verschieben um natürliche Zahlen möglich ist, müssen die Ergebnisse abgerundet werden.

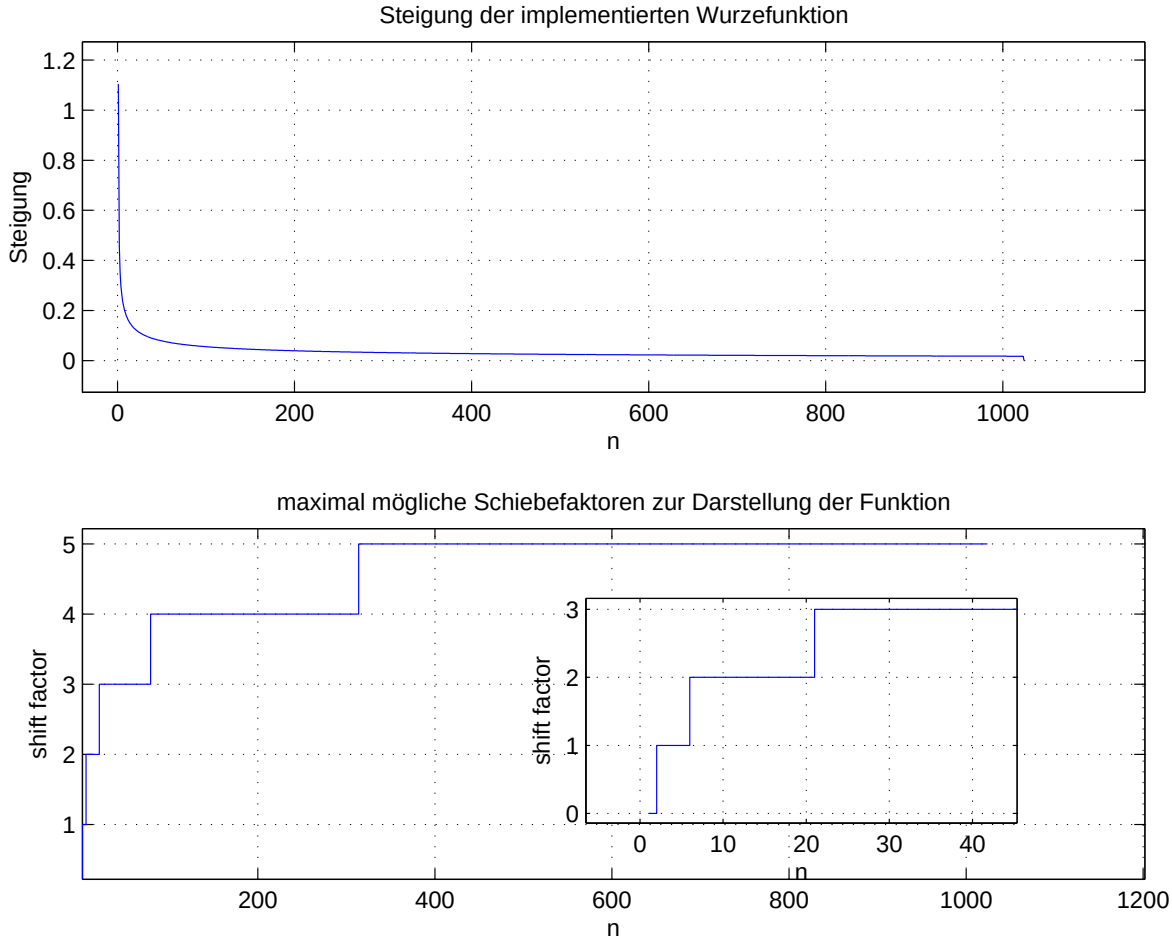

Abbildung 6.3.: Darstellung der 1. Ableitung und der möglichen Schiebefaktoren

Aus dem unteren Plot der Abbildung 6.3 lassen sich dann die Grenzen für die einzelnen Tabellen ermitteln. Diese sind in Tabelle 6.1 zusammengefasst.

unterer Wert	oberer Wert	Schiebefaktor
0	2	0
2	6	1
6	21	2
21	79	3
79	314	4
314	1024	5

Tabelle 6.1.: Einteilung der Grenzen der verschiedenen Wertetabellen

Anhand Tabelle 6.1 lässt sich feststellen, dass zunächst eine Tabelle, die acht Werte enthält, mit dem kleinsten ermittelten Δx realisiert werden muss. Dann kann der Eingangswert um 2 Stellen nach recht geschoben und die Schrittweite Δx um den Faktor vier vergrößert werden. Es ist zudem noch zu beachten, dass der Maximalwert der ersten Tabelle, um den entsprechenden Schiebefaktor nach links geschoben nicht immer Eins ist. Deshalb entstehen Wertebereiche, die von zwei oder mehreren Tabellen abgedeckt werden. Für diesen Fall sollen die Ergebnisse aus der Tabelle, die am feinsten aufgelöst ist, entnommen werden. Die Aufteilung der Wurzel-Tabellen ist der Tabelle 6.2 zu entnehmen:

Anzahl der Werte	Schiebefaktor nach rechts	Minimalwert	Maximalwert
8	0	0	7
8	2	7	$7+8\cdot4+4-1=35$
16	3	$7+1<<3=15$	$16\cdot8+8-1=135$
32	4	$15+1<<4=31$	$32\cdot16+16-1=527$
32	5	$31+1<<5=63$	$32\cdot32-1=1023$

Tabelle 6.2.: Übersicht über die erstellten Wurzel-Tabellen und der dazugehörigen Grenzen

In dem folgenden Beispiel wird die Wurzel von 700 ermittelt werden. Das entspricht binär „1010111100“.

1010111100 » 2	Schieben, da für Tabelle 1 zu groß
10101111 » 1	Schieben, da für Tabelle 2 zu groß
1010111 » 1	Schieben, da für Tabelle 3 zu groß
101011 » 1	Schieben, da für Tabelle 4 zu groß
10101	Es wird der „10101“ = 21. Wert aus Tabelle 5 verwendet

An dieser Stelle steht in der Tabelle der Wert 703. Diese Abweichung kommt zustande, da nicht mehr geklärt werden kann, ob beim Schieben, gesetzte Bits verloren gegangen sind.

In Tabelle 5 kann nur noch ein Bereich von „1010111111“ = 703 und „1010100000“ = 672, in dem das Ergebnis liegt, ermittelt werden. Da in diesem Bereich die Funktionswerte nur 1% auseinander liegen, wird die geforderte Genauigkeit eingehalten.
Bei dieser Methode liegt der große Vorteil darin, dass von ehemals 1024 Werten und 256 Werte aus der Tabelle in [7] nur noch 127 Werte im Speicher abgelegt werden müssen.
Ein Nachteil ist, dass der Wertebereich der verschiedenen Tabellen sich überschneiden kann. Das macht das Aufstellen der Tabellen komplizierter und es wird Speicherplatz verschenkt. Eine weitere Möglichkeit die Tabelle zu codieren, ohne dass doppelte Werte abgespeichert werden müssen, wird im nächsten Abschnitt diskutiert.

6.4. Methode mit PCM codierten Ausgabewerten

6.4.1. Grundidee des Verfahrens

Diese Methode basiert auf der Idee die Kennlinie abschnittsweise anzunähern. Die abschnittsweise Annäherung der Kennlinie wird beispielsweise bei der „Pulse Code Modulation“ (PCM) verwendet. Das Verfahren ist speziell für logarithmische Kennlinien geeignet, deren Steilheit für kleine x-Werte groß ist. Da die Funktion $\sqrt{x} = x^{0.5}$ eine ähnliche Form hat, kann das Verfahren für das Problem verwendet werden. Für die Verwendung dieses Verfahrens muss die Kennlinie in zwei gleich große Segmente aufgeteilt werden, wobei das erste Segment wiederum in zwei Segmente unterteilt wird. Dadurch entsteht eine logarithmische Einteilung der Segmente. Die Kennlinie wird innerhalb eines Segments durch eine Gerade mit bestimmter Steigung angenähert. Ein Segment wird wiederum in Intervalle aufgeteilt, die linear skaliert sind.

Eine vollständige Aufteilung der Kennlinie in einzelne Segmente, die wieder in einzelne Intervalle aufgeteilt sind, ist in Abbildung 6.4 dargestellt.

Um aus einem linear skalierten Wert ein PCM-Codewort zu ermitteln, ist eine Codierung erforderlich. Es wird dafür jedoch nur noch eine Tabelle geben, in der sich die Grenzen der einzelnen Segmente nicht mehr überschneiden.

6.4.2. Umcodierung des Divisionsergebnisses

Das Ergebnis der Division, die bei der Berechnung des Klirrfaktors ausgeführt werden muss, ist linear skaliert. Um das passende Ergebnis aus der Tabelle zu entnehmen, muss das Divisionsergebnis umcodiert werden. Der Aufbau des PCM-Codeworts kann der folgenden Abbildung entnommen werden:

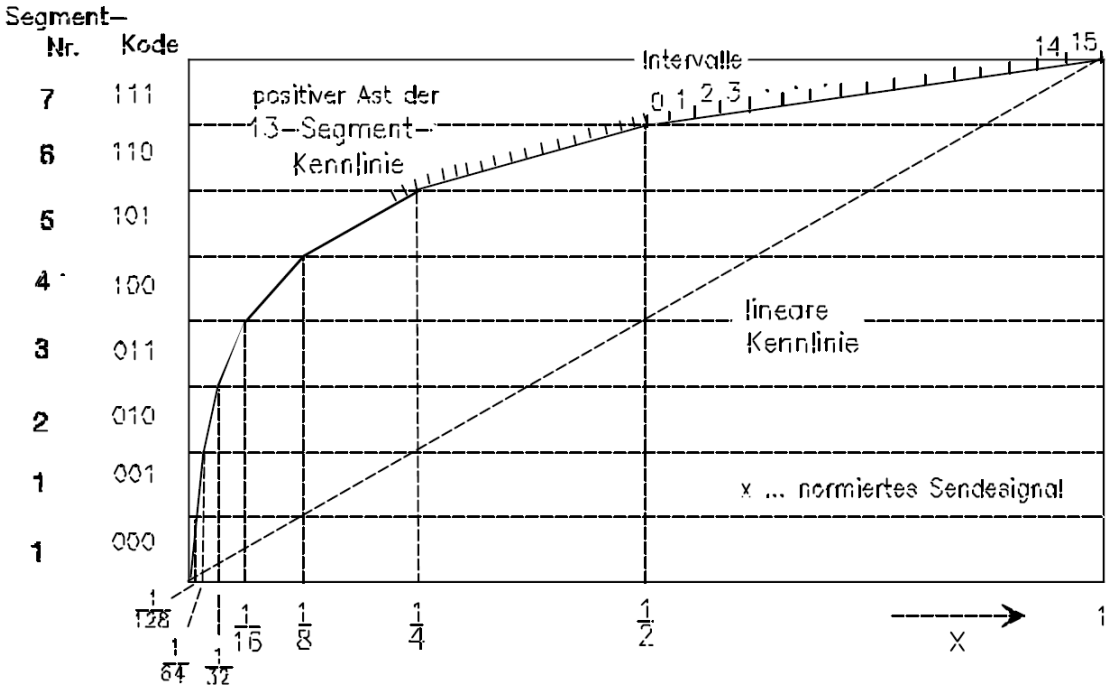

Abbildung 6.4.: Beispiel einer Segmentkennlinie [13]

Vorzeichen	**Bereichscode**	**Intervallcode**

Die Umcodierung kann am einfachsten realisiert werden, wenn vom höchstwertigsten Bit abwärts die Nullen detektiert werden und gleichzeitig ein Zähler, der mit der maximalen Anzahl der darstellbaren Segmente gefüllt ist, abwärts gezählt wird. Wenn an einer Position eine Eins detektiert wird, stellt der Zählerstand den Bereichscode dar. Abhängig von der Anzahl der Intervallen, stellen die nachfolgenden Bits den Intervallcode dar. In dem Beispiel aus Abbildung 6.4 ist ein Segment in 16 Intervalle unterteilt worden. Daher stellen dort die nächsten 4 Bit den Intervallcode dar.

In Tabelle 6.3 ist das Divisionsergebnis und das PCM-Codewort dargestellt:

Segment	Divisionsergebnis	PCM-Codewort
I	0000000wxyz	000wxyz
II	0000001wxyz	001wxyz
III	000001wxyz-	010wxyz
IV	00001wxyz–	011wxyz
V	0001wxyz—	100wxyz
VI	001wxyz—-	101wxyz
VII	01wxyz——	110wxyz
IIX	1wxyz———	111wxyz

Tabelle 6.3.: Umwandlung des linearen Datenworts in ein logarithmisches Codewort

6.4.3. Codierung der Wurzelfunktion in 8 Stufen

In diesem Abschnitt soll die Codierung der verwendeten Wurzelfunktion, die in Abbildung 6.2 dargestellt ist, beschrieben werden. Dazu muss die Kennlinie zunächst in einzelne Segmente unterteilt werden. Zur Ermittlung der Anzahl der Intervalle ist das Segment IIX betrachtet worden. Das Segment ist von x=512 bis x=1024 gebildet; das entspricht einem Funktionswert von 25% bzw. 35%. Da es in diesem Bereich zudem nicht auf das genaue Ergebnis ankommt, weil der Sensor bei einem Klirrfaktor des Ausgangssignals von 25% aus technischen Gründen nur noch ungenaue Ergebisse liefert, wurden 3 Bit für den Intervallcode vorgesehen. Es kommen daher zwei Werte in der Tabelle nicht vor. Es sind jedoch noch 7 Bit übrig, um die Funktion in weitere Segmente zu unterteilen, sodass für kleinere Klirrfaktorwerte das Ergebnis genauer dargestellt werden kann. Die Aufteilung der Segmente ist in Tabelle 6.4 dargestellt:

Segment	Bereich	min. Prozentwert	max. Prozentwert	Differenz (max.-min.)
I	0 - 7	0	3	3
II	8 - 15	3	4	1
III	16 - 31	4	6	2
IV	32 - 63	6	9	3
V	64 - 127	9	12	3
VI	128 - 255	13	18	5
VII	256 - 511	18	25	7
IIX	512 - 1023	25	35	10

Tabelle 6.4.: Gewählte Grenzen der Wurzel-Tabelle

Mit dieser Codierung wird erreicht, dass statt 72 nur noch 64 Tabellenwerte benötigt und in der Tabelle keine doppelten Werte mehr enthalten sind.

6.4.4. Codierung der Wurzelfunktion in 9 Stufen

Mit dieser Methode soll erreicht werden, dass kleine Klirrfaktoren genauer ermittelt werden können. Um die Notwendigkeit festzustellen soll zuerst eine Ergebnis in Abbildung 6.5 bei Codierung der Wurzelfunktion in 8 Stufen angesehen werden.

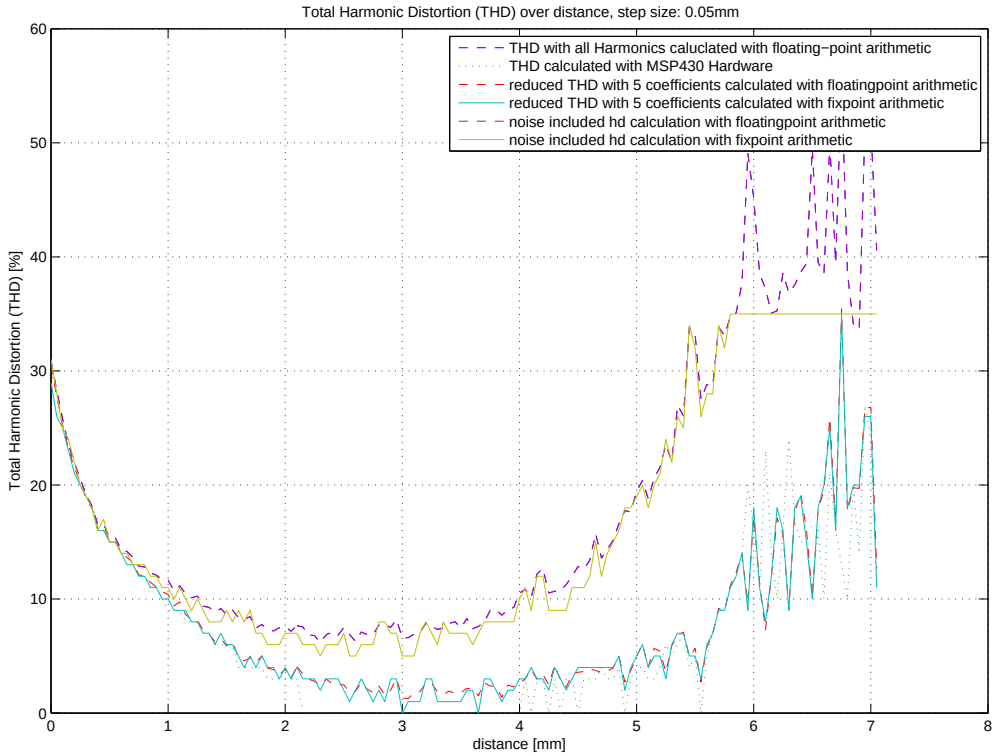

Abbildung 6.5.: Klirrfaktor über die Distanz bei Verwendung einer Wurzelfunktion, die in 8 Stufen codiert ist“

An Abbildung 6.5 fällt auf, dass bei kleinen Ergebnissen große Abweichungen zwischen der Festkomma- und der Gleitkommaberechnung auftreten. Das bedeutet, dass die Ergebnisse in dem Segment I noch zu ungenau sind. Um dem entgegen zu wirken, ist dieses Segment noch einmal halbiert worden, sodass sich die in Tabelle 6.5 dargestellten Grenzen ergeben:

Segment	**Bereich**	**min. Prozent-wert**	**max. Prozent-wert**	**Differenz (max.-min.)**
I	0 - 3	0	2	2
II	4 - 7	2	3	1
III	8 - 15	3	4	1
IV	16 - 31	4	6	2
V	32 - 63	6	9	3
VI	64 - 127	9	12	3
VII	128 - 255	13	18	5
IIX	256 - 511	18	25	7
IX	512 - 1023	25	35	10

Tabelle 6.5.: Gewählte Grenzen der Wurzel-Tabelle

Für die Codierung des Intervalls sind auch hier 3 Bit erforderlich. Für die Codierung der 9 Segmente, ist zudem noch ein weiteres Bit erforderlich, sodass die Wortbreite des Ergebnisses 11 Bit betragen muss. Daraus folgt, dass das Ergebnis der Division um 14 Stellen nach links geschoben werden muss.
Es werden zwar wieder 72 Byte für die Darstellung der Tabelle verwendet, die Ergebnisse im Bereich zwischen 0 und 3% stimmen jetzt jedoch besser mit dem ideal berechneten Klirrfaktor überein. Es besteht jetzt allerdings der Nachteil, dass ein 14 Bit anstatt eines 13 Bit Ergebnisses ohne Einfluss von Rauschen ermittelt werden muss, da ein Segment mehr in der Tabelle enthalten ist.

7. Testimplementierung des Algorithmus' auf dem Mikrocontroller

7.1. Vorgehensweise

In diesem Kapitel soll das Zeitverhalten der HDI-Methode analysiert werden. Zu diesem Zweck soll das Programm in der Programmiersprache C erneut geschrieben werden, da das Programm später auf einem Mikrocontroller der MSP430 Familie getestet werden soll. Dazu ist ein Evaluation-Board vom Typ „MSP430-169STK“ [12] der Firma Olimex verwendet werden, auf dem sich ein MSP430x169 befindet. Das Evaluation-Board ist in der folgenden Abbildung dargestellt:

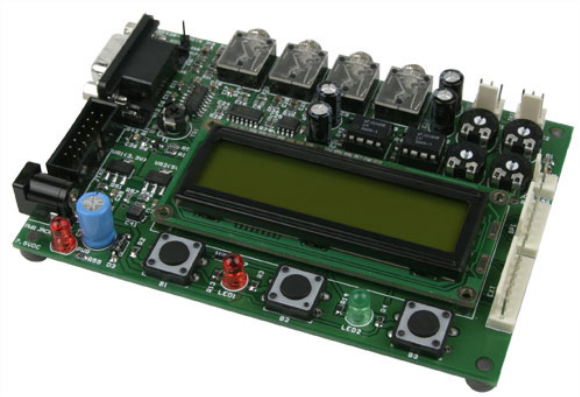

Abbildung 7.1.: Evaluation-Board der Firma Olimex [12]

Abschließend ist die Durchlaufzeit der HDI-Methode mit der Klirrfaktor Berechnung aus der Arbeit von Herrn Jegenhorst [7] vergleichen worden. Für diesen Zweck wurde die Berechnung aus dem Quellcode der Radmessplatz-Software heraus getrennt.

Es handelt sich dabei jedoch nur um eine Beispielanwendung. Die Samples des Eingangssignals sind zur Vereinfachung in einem Feld im Programmspeicher des Mikrocontrollers abgelegt. Wenn das Eingangssignal verändert werden soll, muss das Projekt neu kompiliert werden.

Für die Realisierung der Aufnahme der Werte soll ein 12-Bit-Analog- Digital-Umsetzer verwendet werden. Alle anderen Parameter sind aus [7] übernommen. Die Wurzelfunktion soll PCM-codiert in 9 Stufen werden. Deshalb muss zusätzlich eine Funktion entwickelt werden, die das Divisionsergebnis in ein PCM- Codewort codiert.

Für die Darstellung der Ergebnisse soll das LCD-Display verwendet werden, das fest auf dem Evaluation-Board vorhanden ist. Die erforderlichen Einstellungen sind dem Quellcode des Semesterprojekts mit dem Titel „Local-Positioning-System“ [4] entnommen worden. Gleiches gilt für die Einstellungen des Timers zur Ermittlung der Laufzeitdifferenz. Die Schiebefaktoren sind für das C-Programm im Vorfeld berechnet und als Konstanten im Quellcode eingesetzt worden.

7.2. Aufwandsschätzung der HDI-Methode

Als nächstes soll die Zeitersparnis theoretisch betrachtet und anschließend mit dem Evaluation-Board gemessen werden.
Am leichtesten lässt sich die Dauer der Berechnung der ersten Harmonischen voraussagen, da diese auch für den approximierten Klirrfaktor berechnet werden muss. Es ist für beide Verfahren eine ähnliche Vorgehensweise gewählt worden, sodass gesagt werden kann, dass die Berechnung der ersten Harmonischen $\frac{1}{5}$ der Berechnungszeit des HD5-Methode in Anspruch nimmt. Es sind somit nur noch die Anzahl der benötigten Operationen für den Gleichanteil sowie die Leistung des Signals zu ermitteln.

Für die Berechnung einer Harmonischen eines Signal mit N Samples sind $a_H = 2 \cdot N + 1$ Additionen und $m_H = 2 \cdot (N + 1)$ Multiplikationen erforderlich. Für die Berechnung des Gleichanteils sind noch einmal N Additionen erforderlich, da alle Samples aufaddiert werden müssen. Die anschließende Division durch N kann durch eine Schiebeoperation realisiert werden. Für die Berechnung der Leistung des Signals sind weiterhin N Subtraktionen des Gleichanteils und N Additionen für das Aufaddieren der Quadrate der Samples erforderlich. Außerdem werden noch N Multiplikationen für das Quadrieren der Samples benötigt. Werden alle Additionen a bzw. Multiplikationen m zusammengezählt, ergibt sich, dass zusätzlich $a = 3 \cdot N$ Additionen und $m = N$ Multiplikationen benötigt werden. Wie groß der Zeitunterschied zwischen einer Multiplikation und einer Addition genau ist, konnte leider nicht herausgefunden werden. Deshalb wird angenommen, dass eine Multiplikation viermal so viel Zeit in Anspruch nimmt wie eine Addition.
Daraus folgt dann, dass die Berechnungszeit der Leistung des Signals k-mal der Berechnungszeit einer Harmonischen entspricht.

$$k = \frac{4 \cdot m + a}{4 \cdot m_H + a_H} = \frac{4 \cdot N + 3 \cdot N}{4 \cdot [2 \cdot (N + 1)] + 2 \cdot N + 1} \approx \frac{7 \cdot N}{10 \cdot N} = 0,7 \qquad (7.1)$$

Aus diesen Ergebnissen lässt sich ein Faktor x ermitteln, um den die Berechnung nach der HDI-Methode schneller ist, als die mit der HD5-Methode:

$$x = \frac{5}{1+k} = \frac{5}{1.7} = 2,94 \tag{7.2}$$

7.3. Implementierung der ersten Berechnungsmögichkeit

7.3.1. Vergleich der Zwischenergebnisse

In diesem Abschnitt werden die Zwischenergebnisse der 1. Berechnungsmöglichkeit, die in Abschnitt 5.4.1 beschrieben ist, untersucht. Zu Beginn der Untersuchung sollen die mit Matlab ermittelten Zwischenergebnisse mit den Ergebnissen des MSP430-Mikrocontrollers verglichen werden. Die Berechnungsergebnisse des Mikrocontrollers werden mit Hilfe der Debug-Funktion ermittelt. Die Zwischergebnisse, aus denen sich später der Klirrfaktor errechnet, sind in Tabelle 7.3.1 dezimal dargestellt:

	Mit dem Mikrocontroller ermittelte Ergebnisse	Mit Matlab ermittelte Ergebnisse	mit Gleitkomma-Arithmetik berechnet
Gleichanteil	2219	2219	2219
Gesamtleistung	57041756	57041270	57041267
Leistung 1. Harmonische	56964756	56979925	56979924
ermittelter Klirrfaktor	3%	3%	3,72 %

Tabelle 7.1.: Vergleich der Ergebnisse des C-Programms und des Matlab Programms

An den Ergebnissen ist auffällig, dass die Berechnung des Gleichanteils sehr exakt möglich ist. Bei der Berechnung der Leistung der ersten Harmonischen sowie bei der Gesamtleistung treten Ungenauigkeiten auf, die vermutlich durch die Multiplikationen bedingt sind. Diese Vermutung kann durch die exakte Berechnung des Gleichanteils, für die keine Multiplikationen erforderlich ist, gefestigt werden. Die Ergebnisse der Matlab-Berechnung mit reduzierter Wortbreite liegen zudem näher am exakten Ergebnis als die Berechnungsergebnisse des Mikrocontrollers.

7.3.2. Vergleich der Durchlaufzeiten beider Verfahren

Im diesem Schritt soll die Durchlaufzeit beider Verfahren bestimmt werden. Um die Messung durchführen zu können, ist zuerst, wie in [7] praktiziert, in der Entwicklungsumgebung für den verwendeten Controller der MSP1232 angegeben worden, da dieser über keinen Hardware-Multiplizierer verfügt.
Für die Messung ist ein Timer verwendet worden, der alle 100μs einen Zähler um 1 erhöht. Der Mikrocontroller arbeitet mit einem Systemtakt von 8MHz. Es sind bei diesem Versuch folgende Zählerstände ermittelt worden:

Verfahren	**Zählerstand**
HD5 Methode	34553
HDI-Methode	11284

Daraus kann ein Faktor x errechnet werden, der aussagt, um wie viel die Berechnung der HDI-Methode schneller als die Berechnung mit der HD5-Methode ist.

$$x = \frac{34553}{11284} = 3,06 \tag{7.3}$$

Als Ergebnis kann festgehalten werden, dass sich der Klirrfaktor unter Verwendung der HDI-Methode, sowie geschickterem Schieben und neuer Wurzel-Tabelle ungefähr um den Faktor 3 schneller berechnen lässt, als mit der HD5-Methode. Wenn die Berechnung wie in diesem Programm implementiert wird, besteht der Nachteil, dass alle aufgenommenen Samples gespeichert werden müssen. Deshalb wird im nächsten Schritt die zweite vorgeschlagene Berechnungsmethode untersucht.

7.4. Implementierung der ersten Berechnungsmögichkeit

7.4.1. Vergleich der Zwischenergebnisse

Für diese Berechnungsmethode, die in Abschnitt 5.4.1 beschrieben ist, sollen zuerst die Zwischenergebnisse verglichen werden. In Tabelle 7.4.1 sind die ermittelten Teilergebnisse dezimal aufgelistet:

	Mit dem Mikrocontroller ermittelte Ergebnisse	Mit Fixed-Point-Arithmetik ermittelte Ergebnisse	mit Gleitkomma-Arithmetik berechnet
Gleichanteil	2219	2219	2219
Gesamtleistung	1786160	1783926	1782539
Leistung 1. Harmonische	1780148	1780623	1780074
ermittelter Klirrfaktor	4%	3%	3,72 %

Tabelle 7.2.: Vergleich der Ergebnisse des C-Programms und des Matlab Programms

Es ist für diese zweite Berechnungsmethode lediglich die Berechnung der Leistung des Signals verändert worden, sodass die Ergebnisse bis auf den Faktor $\frac{1}{L}$ vergleichbar sind.

7.4.2. Vergleich der Berechnungszeiten beider Verfahren

Für die zweite Berechnungsmethode wird eine etwas kürzere Durchlaufzeit erwartet, weil statt N Subtraktionen des Gleichanteils nur noch eine Multiplikation und eine Subtraktion nötigt sind. In der folgenden Tabelle sind die ermittelten Zeiten der beiden Verfahren dargestellt:

Verfahren	**Zählerstand**
HD5-Methode	34509
HDI-Methode	9363

Daraus kann ein Faktor x errechnet werden, der aussagt, um wie viel die Berechnung mit der HDI-Methode schneller ist, als die Berechnung mit der HD5-Methode.

$$x = \frac{34509}{9363} = 3,69 \tag{7.4}$$

Wird der Klirrfaktor nach dieser Methode berechnet, ist die Berechnung sogar um den Faktor 3,69 schneller. Zudem besteht noch der Vorteil, dass die Samples nicht mehr gespeichert werden müssen und der Gleichanteil trotzdem berechnet werden kann. Daher ist es in der Praxis empfehlenswert, diese Berechnungsmethode zu verwenden.

8. Beschreibung des Simulationsprogramms

8.1. Einleitung

In diesem Kapitel soll das Programm vorgestellt werden, mit dem die Ermittlung des Klirrfaktors der Radmessplatz-Software nachgebildet werden kann. Mit dem Programm soll erreicht werden, den Klirrfaktor in Abhängigkeit vom Luftspalt zwischen Sensor und Encoderrad darzustellen. Damit eine Simulation einfach durchgeführt werden kann und die Simulationsergebnisse, sowie die verwendeten Simulationsparameter und die erstellten Bilder gut dokumentiert und für die spätere Verwendung gesichert werden können, ist ein Konzept erstellt worden, nachdem die Daten automatisch gesichert werden. Damit korrekte Ergebnisse erreicht werden, müssen bei Verwendung des Simulationsprogramms einige Punkte beachtet werden. Jeder Anwender, der das Simulationsprogramm bedienen möchte, sollte sich vor der erstmaligen Simulation mit diesem Kapitel der Diplomarbeit beschäftigen. Dazu werden im nächsten Abschnitt die einzelnen Arbeitsschritte erläutert. Dabei ist es wichtig, dass die Schritte der Reihe nach auszuführen. In der Abbildung 8.1 ist der Ablauf des Programms grafisch dargestellt.

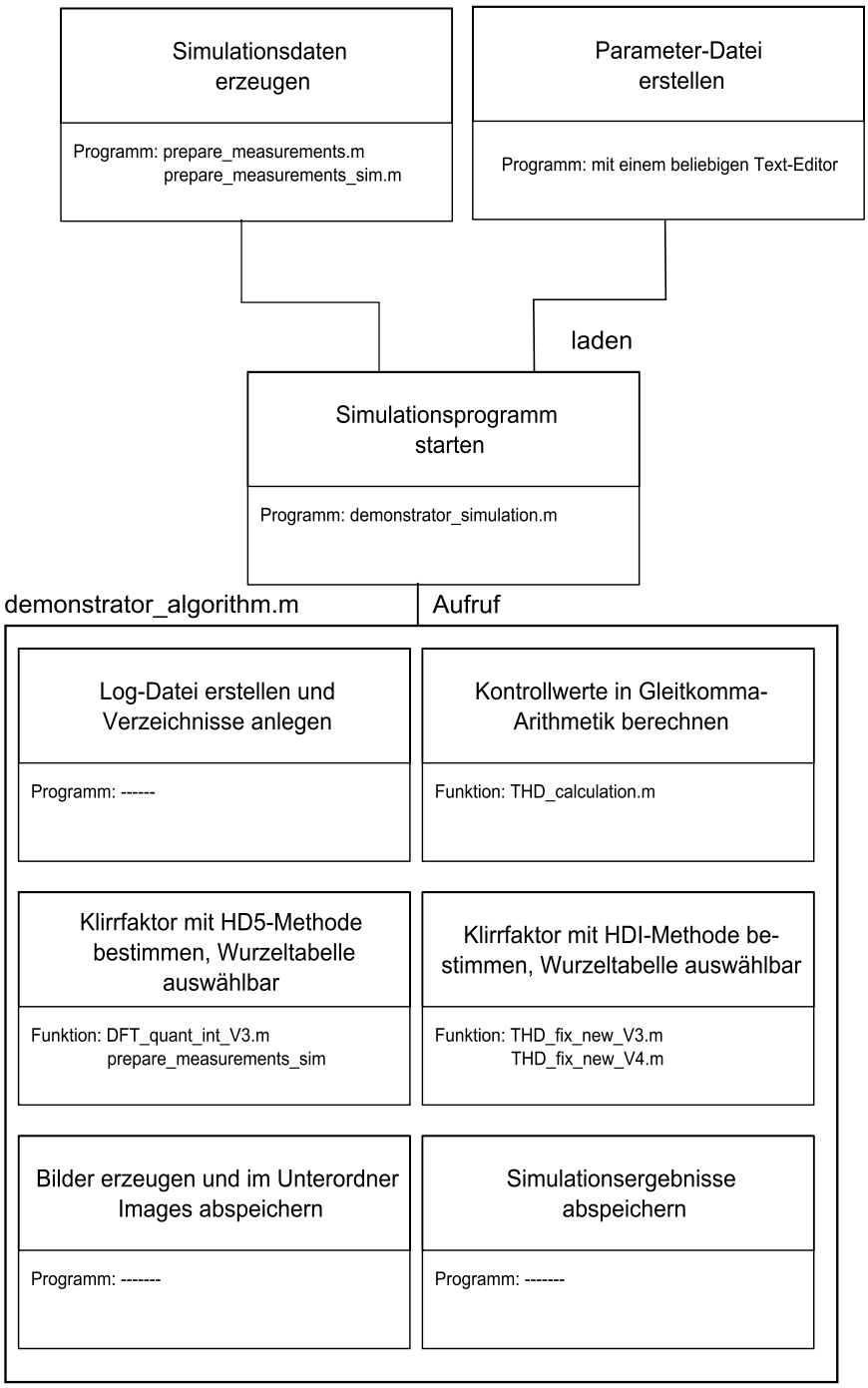

Abbildung 8.1.: Aufbau des Simulationsprogramms

8.2. Anleitung zum Durchführen einer Simulation

8.2.1. Aufbereiten der Messdaten

Für die Aufbereitung der Messdaten stehen zwei Matlab-Programme zur Verfügung. Das erste Programm trägt den Namen „prepare_measurements“ und ist in Kapitel 3.4 beschrieben. Das zweite Programm trägt den Namen „prepare_measurements_sim“ und verwendet statt der Messdaten des Radmessplatzes nachgebildete Messergebnisse. Die Beschreibung des Programms befindet sich in Abschnitt 10.3.4.

8.2.2. Simulationsparameter festlegen

Einstellen von Zahlenwerten

In diesem Abschnitt werden die Simulationsparameter vorgestellt, die als Ziffer angegeben werden müssen. Tabelle 8.1 soll einen Überblick über die einstellbaren Parameter, die als Ziffer angegeben werden müssen, geben:

Parameter	**Beschreibung**	**untere Grenze**	**obere Grenze**
Nb_LUT	Auflösung der Sinus- und Kosinus-Tabelle	1 Bit	15 Bit
Nb_ADC	Auflösung des Analog-Digital-Umsetzers in Bit	1 Bit	15 Bit
No	Anzahl der berücksichtigten Harmonischen des HD5-Methode	eingestellte Anzahl der Harmonischen des approximierten Signals	
N_Sample	Anzahl der Samplewerte pro Periode	16	nicht vorgesehen

Tabelle 8.1.: Grenzen der einstellbaren Parameter

Auswahl der Datenquelle

Mit dem Parameter use_data kann ausgewählt werden, ob die Daten, welche mit dem Oszilloskop aufgenommen wurden, oder ob die Daten die mit dem Demonstrator aufgenommen

wurden, zur Anwendung kommen sollen. Die Angabe wird hier in einer Zeichenkette gemacht. Durch die Angabe von 'demo' werden die Messdaten des Demonstrators ausgewählt. Durch die Angabe von 'scope' werden die Daten des Oszilloskops verwendet. Es besteht außerdem die Möglichkeit, keine Angabe zu machen. In diesem Fall wird als Default-Wert ('demo') verwendet.

Auswahl der Wurzel-Tabelle

Mit dem Parameter use_table kann eine Wurzel-Tabelle gewählt werden, in der die Wurzelfunktion gespeichert ist. Die Wurzel-Tabellen können mit einer Ziffer zwischen 1 und 4 ausgewählt werden. Der Default-Wert ist für diese Einstellung der Wert „2“. Eine Übersicht mit welcher Ziffer die gewünschte Tabelle ausgewählt werden kann, ist in Tabelle 8.2 dargestellt:

Ziffer	Wurzel-Tabelle
1	original Wurzel-Tabelle aus der Diplomarbeit von N.Jegenhorst
2	Methode mit PCM codierten Ausgabewerten (9 Stufen)
3	Methode mit PCM codierten Ausgabewerten (8 Stufen)
4	LUT-Methode mit unterschiedlichen Tabellen

Tabelle 8.2.: Übersicht der verschiedenen Wurzel-Tabellen

Auswahl des Typs einer Variablen

Zur Auswahl des Typs der Variablen, die zur Berechnung des Klirrfaktors benötigt werden, gibt es 3 Auswahlmöglichkeiten:

- **short**: Dieser Datentyp entspricht dem Typ „short“ in der Programmiersprache C. Es stehen 15 Datenbits sowie 1 Vorzeichenbit zur Verfügung.
- **24Bit**: Dieser Datentyp ist im Rahmen dieser Diplomarbeit eingefügt worden. Da für die Wurzel-Tabelle ein gültiges Ergebnis mit einer Wortbreite von 14 Bit benötigt wird (s. Kapitel 6.4.4), kann es bei dem Datentyp „short“ zu Problemen kommen. Da der Aufwand laut Aufgabenstellung in dieser Arbeit reduziert werden soll, ist dieser Datentyp eingeführt worden. Er hat eine Wortbreite von 24 Bit, die sich in ein Vorzeichenbit und 23 Datenbits aufteilen.
- **long**: Dieser Datentyp entspricht dem Datentyp „long“ in der Programmiersprache C. Für die Daten stehen hier 31 Bit zur Verfügung, und für das Vorzeichen ist ein Bit reserviert. Der Datentyp wird als Default-Wert angenommen.

Der Name dieses Parameters in der Simulation lautet „wordlength“. Die vorgestellten Datentypen können als Zeichenkette für den Parameter „wordlength“ angegeben werden. Wenn für diesen Parameter in der Parameter-Datei nichts angegeben wird, wird automatisch der Datentyp „long“ verwendet.

Auswahl der Berechnungsmöglichkeit für die HDI-Methode

In dieser Arbeit sind in Abschnitt 5.4 zwei Möglichkeiten für die Berechnung des Klirrfaktors vorgestellt worden. Bei der ersten Möglichkeit ist der Gleichanteil im Vorfeld berechnet, und von jedem aufgenommenen Zahlenwert subtrahiert worden. Bei der zweiten Möglichkeit wird der Gleichanteil parallel zur Berechnung der Signalleistung ermittelt. Wenn diese Berechnung abgeschlossen ist, wird erst der Gleichanteil abgezogen. Um die Berechnungsmethode frei wählen zu können, ist ein weiterer Parameter „noise_inc_var“ eingeführt worden. Wird für den Parameter keine Angabe gemacht oder eine 1 angegeben, wird die erste Berechnungsmöglichkeit ausgewählt. Mit einer 2 wird die zweite Möglichkeit ausgewählt.

Beispiel einer Parameter-Datei

Damit die Parameter verwendet werden können, muss eine sogenannte Parameter-Datei erstellt werden. In dieser müssen die in diesem Abschnitt vorgestellten Parameter eingetragen werden. Die Parameter-Datei kann nach folgendem Muster in einem beliebigen Editor erstellt und als Text-Datei abspeichert werden:

```
Nb_LUT = 10,                    'Aufloesung der Sinus- und Kosinus-Tabelle'
Nb_ADC = 12,                    'Aufloesung des ADC'
No = 5,                         'Anzahl der beruecksichtigten Koeffizienten
                                 für HD5-Methode'
N_Sample = 64,                  'Anzahl der Abtastwerte pro Periode'
use_data = 'demo',              'Datenquelle'
use_table = 2,                  'verwendete Wurzel-Tabelle (Doku DA)'
wordlength = 'long',            'Wortbreite der Register'
noise_inc_var = 2,              'Verwendung 2. Moeglichkeit der
Berechnung'
```

8.2.3. Verwalten der Simulationsergebnisse

Der Name der Messreihe

Die Ergebnisse der Simulationen werden in dem Order „D:/simulation_folder“ abgelegt. Für jede Messreihe wird ein Unterordner erstellt, dessen Name sich aus dem Namen der

Parameter-Datei, dem Datum und der aktuellen Uhrzeit zusammensetzt. Die Namensgebung einer Messreihe wird immer nach folgendem Muster vorgenommen: „*Name der Parameter-Datei*“_JJJJMMDDHHMM.
Für eine Messreihe, die zum Beispiel am 24.02.2010 um 12:45 aufgenommen und für die eine Parameter-Datei „test.txt“ verwendet wurde, wird ein Unterordner mit folgendem Namen erstellt: test_201002241245.

Erklärung der Log-Datei

Bei der Log-Datei handelt es sich um eine Textdatei, in der alle Informationen über die verwendeten Simulationsparameter, über das Eingangssignal sowie über das Ergebnis der Simulation festgehalten werden.
Das Aussehen der Log-Datei kann sich leicht verändern. Das ist davon abhängig, ob die aufgenommenen Daten des Radmessplatzes oder die Oszilloskopdaten verwendet wurden. Eine Log-Datei kann zum Beispiel wie folgt aussehen:

```
************************************************************
*             Simulation Radmessplatz                      *
************************************************************

Messung durchgeführt am : 18.04.2010 um 18:55 Uhr

verwendete Parameter
Name der Parameterdatei           : D:\SVN\rmp\subprojects
\koch_le\sw\matlab\parameter files\demo.txt
Name der Messreihe                : 2009_10_07_rmp_01_for_simulation.mat
verwendete Messdaten              : demo
Samples pro Periode               : 64
berücksichtige Koeffizienten      : 5
Auflösung Look-up Table für DFT   : 10 Bit
Auflösung Analog-Digital-Umsetzer : 12 Bit
verwendete Look-up Tabelle        : PCM Codierung 9 Stufen
Berechnungsmöglichkeit HDI-Methode: Berechnung Gesamtleistung mit Gleichanteil
Wortbreite der Variablen          : long


************************************************************
*            Eigenschaften des Eingangssignals             *
************************************************************

Es werden die Messwerte des Demonstrators verwendet.
Nähere Informationen zum verwendeten Abtastverfahren sind in der
Diplomarbeit Jegenhorst nachzulesen.
verwendete Parameter
Anzahl der berücksichtigten  Harmonischen               : 25


************************************************************

Ergebnis der Simulation:

Simulation erfolgreich durchgeführt! Daten können unter dem
Namen Part_1_NLUT=10_No=5_NADC=12_Samples=64.sim
erneut geöffnet werden
```

****

Sicherung der Simulationsergebnisse

Die Simulationsergebnisse werden zur späteren Auswertung in einer MAT-Datei abgespeichert. Der Name dieser Datei setzt sich auch hier aus der Nummer der Teilsimulation und den Simulationsparametern zusammen. In dieser Datei sind alle Ergebnisse sowie die verwendeten Eingabedateien wie die Eingangssignale oder die Sinus- und Kosinus-Tabellen hinterlegt. Eine genaue Auflistung der Variablennamen und eine Beschreibung, welche Daten darin enthalten sind, ist im Anhang C.1 zu finden.

Automatisch erstellte Plots

Nach jedem Durchlauf der Simulation werden automatisch einige Plots erstellt. Diese werden im Unterordner „Images“ zur Verfügung gestellt. Die Plots werden in den Formaten „*.pdf“, „*jpg“ und“*.fig“ gespeichert. Der Name der Plots setzt sich aus den Werten der Simulationsparameter, der Nummer der Teilsimulation sowie einer Endung, die die dargestellten Signale beschreibt, zusammen. Die Nummer der Teilsimulation ist bei den bisher behandelten einfachen Simulationen immer „Eins“. Der Wert ändert sich nur bei den Werten einer Simulationsreihe. Nähere Informationen zur Aufnahme einer Simulationsreihe sind Abschnitt 8.3 zu entnehmen. Nach einer Simulation werden Plots mit folgenden Endungen erzeugt:

- **Magnitudes_Real_and_Imaginary_Parts**: In diesem Bild sind die Beträge, Real- und Imaginärteile der ersten 5 Harmonischen dargestellt. In der oberen Reihe werden diese mit Gleitkomma-Arithmetik und in der zweiten Reihe mit Festkomma-Arithmetik berechnet.
- **THD_over_distance**: In diesem Plot wird der Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad dargestellt.
- **power_of_signals**: In diesem Plot ist die Leistung des Signals sowie der Leistung der ersten Harmonischen im Abhängigkeit von der Entfernung zwischen Encoderrad und dem Sensor dargestellt.
- **error_approximated_THD**: In diesem Plot ist der Fehler, der bei Festkomma-Berechnung des Klirrfaktors unter Berücksichtigung einer bestimmten Anzahl von Perioden dargestellt.
- **error_HDI_method**: In diesem Plot ist der Fehler, der bei Festkomma-Berechnung des HDI-Methode aufgetreten ist, dargestellt.

- **error_demonstrator**: In diesem Plot ist der Fehler des mit dem Radmessplatz ermittelten Ergebnisses dargestellt.
- **error_different_implementations**: In diesem Plot wird der relative und der absolute Fehler zwischen der HDI-Methode und der HD5-Methode dargestellt. Dieser Fehler muss im idealen Fall Null sein.

8.3. Durchführung einer Reihe von Simulationen

Eine weitere wichtige Funktion des Simulationsprogramm ist die Erstellung einer Simulationsreihe. Wenn der Benutzer zum Beispiel wissen möchte, wie die Ergebnisse für einen Analog-Digital-Umsetzer mit einer Auflösung von 5,6,7 und 12 Bit aussehen, kann er folgende Parameter-Datei erstellen:

```
Nb_LUT = 10,
Nb_ADC = [5 6 7 12],
No = 5,
N_Sample = 64,
use_data = 'demo',
use_table = 2,
wordlength = 'long'
```

Das Simulationsprogramm führt dann vier Simulationen aus. Die Bilder werden für jede Simulation erstellt, jedoch nicht mehr angezeigt, da der Arbeitsspeicher schnell belegt sein würde. Für jede Simulationsreihe werden die relevanten Daten in einer eigenen Datei gespeichert, sodass die einzelnen Simulationen auch verwendet werden können. Der Dateiname für die erste Simulation beginnt mit „Part 1“, für die zweite Simulation dann mit „Part 2". Für weitere Simulationen wird der Index weiter hochgezählt.

9. Ergebnisse der Simulation

9.1. Einleitung

In diesem Kapitel ist das Simulationsprogramm zuerst auf die Funktionalität geprüft worden. Verwendet werden hierfür, die in [7] festgelegten Parameter. Danach ist die Wortbreite der Variablen, in denen die Zwischenergebnisse abgelegt und aufaddiert werden, reduziert worden. Unter Verwendung dieser Ergebnisse wurde eine umfangreiche Simulationsreihe durchgeführt, um Parameter zu bestimmen, bei denen das Simulationsprogramm noch korrekte Ergebnisse ermitteln kann. In dieser Arbeit wird die Messreihe 2009_10_07_rmp_01 verwendet. Im folgenden sind einige Simulationen durchgeführt worden. Die Ergebnisse dieser Simulationen sind vollständig auf der Daten-CD [8] enthalten.

9.2. Simulation mit den verwendeten Parametern des Radmessplatzes

Als erstes soll der Radmessplatz mit den dort verwendeten Parametern simuliert werden. Um die Ergebnisse besser vergleichen zu können, werden die Ergebnisse, dem Demonstrator ermittelt worden sind, in dem selben Plot dargestellt. Die sonstigen Parameter der Simulation können der folgenden Parameter-Datei entnommen werden:

```
************************************************************
*             Simulation Radmessplatz                      *
************************************************************

Messung durchgeführt am : 18.04.2010 um 18:55 Uhr

verwendete Parameter
Name der Parameterdatei           : D:\SVN\rmp\subprojects\koch_le\
sw\matlab\parameter files\demo.txt
Name der Messreihe                : 2009_10_07_rmp_01_for_simulation.mat
verwendete Messdaten              : demo
Samples pro Periode               : 64
berücksichtige Koeffizienten      : 5
Auflösung Look-up Table für DFT   : 10 Bit
Auflösung Analog-Digital-Umsetzer : 12 Bit
verwendete Look-up Tabelle        : PCM Codierung 9 Stufen
Berechnungsmöglichkeit HDI-Methode: Berechnung Gesamtleistung mit Gleichanteil
Wortbreite der Variablen          : long
```

```
************************************************************
*            Eigenschaften des Eingangssignals             *
************************************************************

Es werden die Messwerte des Demonstrators verwendet.
Nährere Informationen zum verwendeten Abtastverfahren sind in der
Diplomarbeit Jegenhorst nachzulesen.
verwendete Parameter
Anzahl der berücksichtigiten  Harmonischen               : 25


************************************************************

Ergebnis der Simulation:

Simulation erfolgreich durchgeführt! Daten können unter dem
 Namen Part_1_NLUT=10_No=5_NADC=12_Samples=64.sim
erneut geöffnet werden
************************************************************
```

9.2.1. Diskussion des Klirrfaktors in Abhängigkeit zu der Entfernung zwischen Sensor und Encoderrrad

In diesem Abschnitt wird der Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad näher betrachtet .

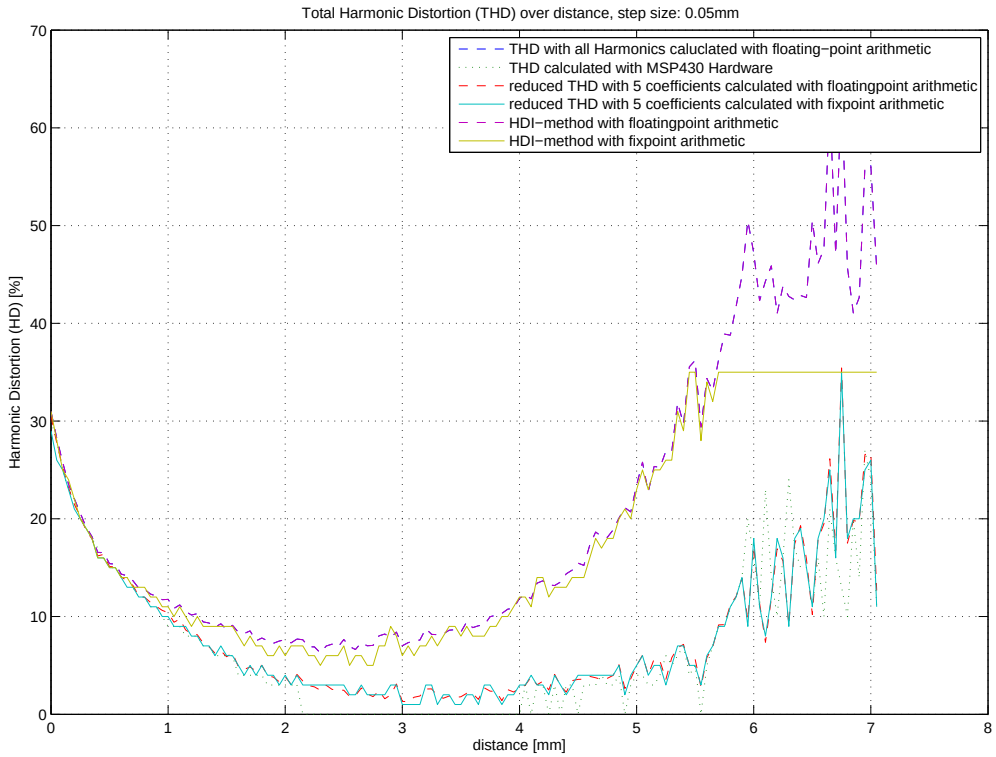

Abbildung 9.1.: Darstellung des Klirrfaktors in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad, für die Parameter des Radmessplatzes [8]

In Abbildung 9.1 ist zu erkennen, dass die Ergebnisse der HD5-Methode, mit den gemessen Werten übereinstimmen. Bei Verwendung der HDI-Methode weichen die Ergebnisse der Festkomma- und Gleitkomma-Berechnung voneinander ab. Um das Problem zu untersuchen, wird die Formel zur Berechnung betrachtet:

$$THD = \sqrt{\frac{P_{ges} - P_1}{P_{ges}}} = \sqrt{1 - \frac{P_1}{P_{ges}}} \tag{9.1}$$

An dieser Formel lässt sich erkennen, dass der Klirrfaktor kleiner wird, wenn die Leistung der ersten Harmonischen zu groß geschätzt wird. Wenn mehr Harmonische berücksichtigt werden, weichen die Kennlinien ab einer Entfernung von ungefähr 0,5mm voneinander ab. Bei der Berechnung durch den Mikrocontroller besteht weiterhin das Problem, dass das Ergebnis bei sehr kleinen Werten Null wird. Das Problem kommt zum einen durch die ungünstig gewählte Wurzel-Tabelle zur Ermittlung der Wurzelfunktionen und zum anderen durch unnötig starke Reduzierung der Teilergebnisse, zustande.
Das mit Festkomma-Arithmetik berechnete Ergebnis mit der HDI-Methode, stimmt erwartungsgemäß mit der Berechnung des Klirrfaktors unter Berücksichtigung aller Koeffizienten überein. Das kann durch den Nachweis erklärt werden, der im A.2 angefügt ist.

Die Fixpoint-Implementationen nähern sich an die exakt berechneten Kennlinien gut an. Zur Untersuchung, der tatsächlich erreichen Genauigkeit, soll im nächsten Abschnitt untersucht werden. Wird die Entfernung größer als 0,5 Millimeter, vergrößern sich die ermittelten Ergebnisse der HD5 Methode und der HDI-Methode. Dieses Problem soll, im nachfolgenden Kapitel untersucht werden.

9.2.2. Analyse der Abweichungen bei Verwendung von Fixed-Point-Arithmetik

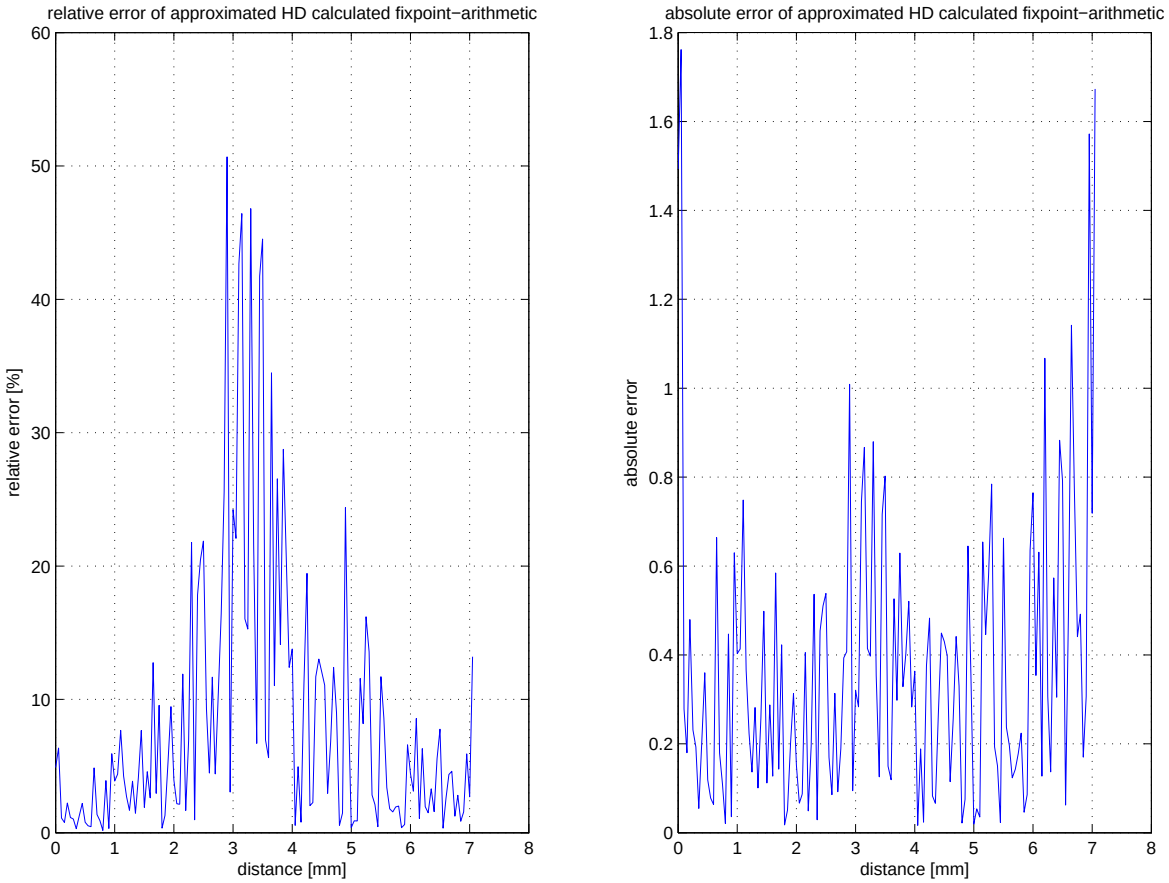

Abbildung 9.2.: Relative und absolute Abweichungen der HD5-Methode [8]

Wie Abbildung 9.2 zeigt, liegt dieser Implementierung die maximale Abweichung des Festkomma-Ergebnisses bei ca. 1.8%, allerdings ist an dieser Stelle der Klirrfaktor sehr groß. Für diesen Bereich ist nicht möglich das Radizieren mit einer Genauigkeit von 1% durchzuführen. Die übrigen Werte können in diesem Fall sicher mit einer Genauigkeit von

1,5% bestimmt werden. Das stimmt mit der zuvor festgelegten Genauigkeit der Wurzel-Tabelle überein.

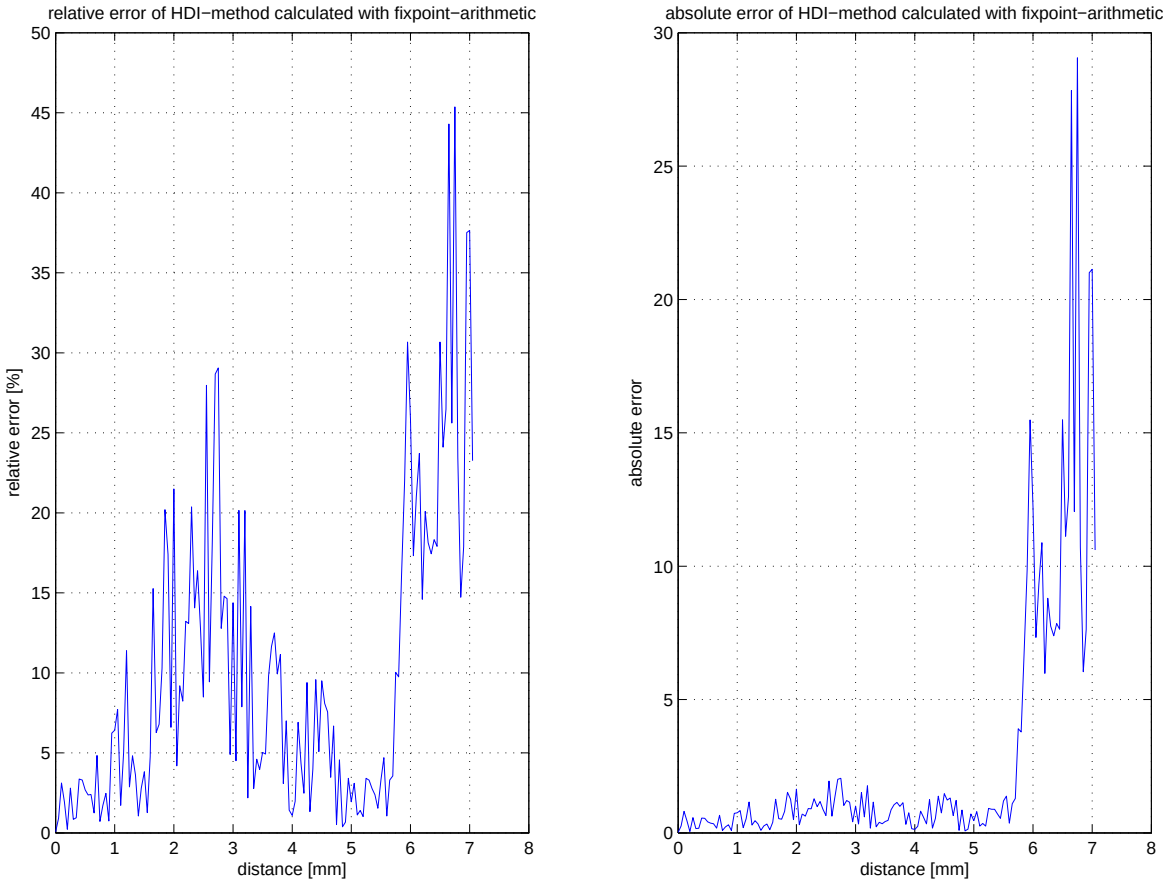

Abbildung 9.3.: Relative und absolute Abweichungen bei Verwendung der HDI-Methode [8]

In dem Fall, der in Abbildung 9.3 dargestellt ist, kann festgehalten werden, dass das Ergebnis, das zwischen 0 und 5,5 Millimeter berechnet wurde, um maximal 2% abweicht. Damit lässt sich eine Zustanserkennung eines ABS-Sensors durchführen. Da der Klirrfaktor (wie in Bild 9.1) bei dieser Implementierung für große Entfernungen sehr stark ansteigt, wird der Maximalwert, der in der Wurzel-Tabelle abgelegt ist, erreicht. Das hat zur Folge, dass der absolute und somit auch der relative Fehler größer wird.
In einem weiteren Plot ist die Abweichung zwischen den beiden Berechnungsverfahren dargestellt. Idealerweise sollte dieser Fehler Null sein. In Abbildung 9.4 ist bereits zu erkennen, dass für Entfernungen von mehr als 0,5 Millimeter der Fehler groß ist. Daraus lässt sich schließen, dass ein Problem vorliegt, das im folgenden Kapitel näher untersucht wird.

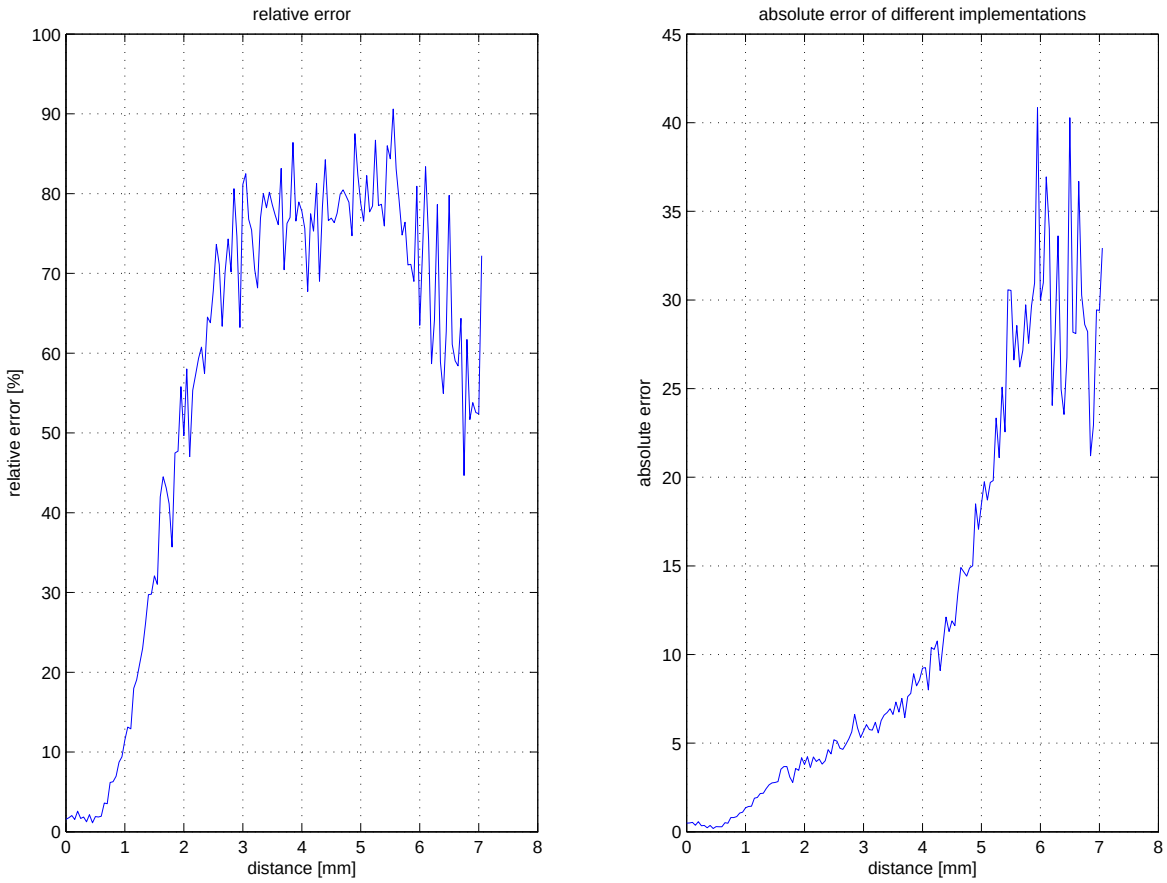

Abbildung 9.4.: Darstellung der Abweichung zwischen den beiden verwendeten Berechnungsverfahren [8]

9.3. Reduzierung der Wortbreite der Variablen

9.3.1. Reduzierung der Wortbreite auf 16 Bit

In diesem Versuch sollen die Variablen zum Speichern der Ergebnisse und für die Additionen nur noch 16 Bit aufweisen. Vor den Multiplikation wird die Wortbreite nach wie vor auf 16 Bit begrenzt.

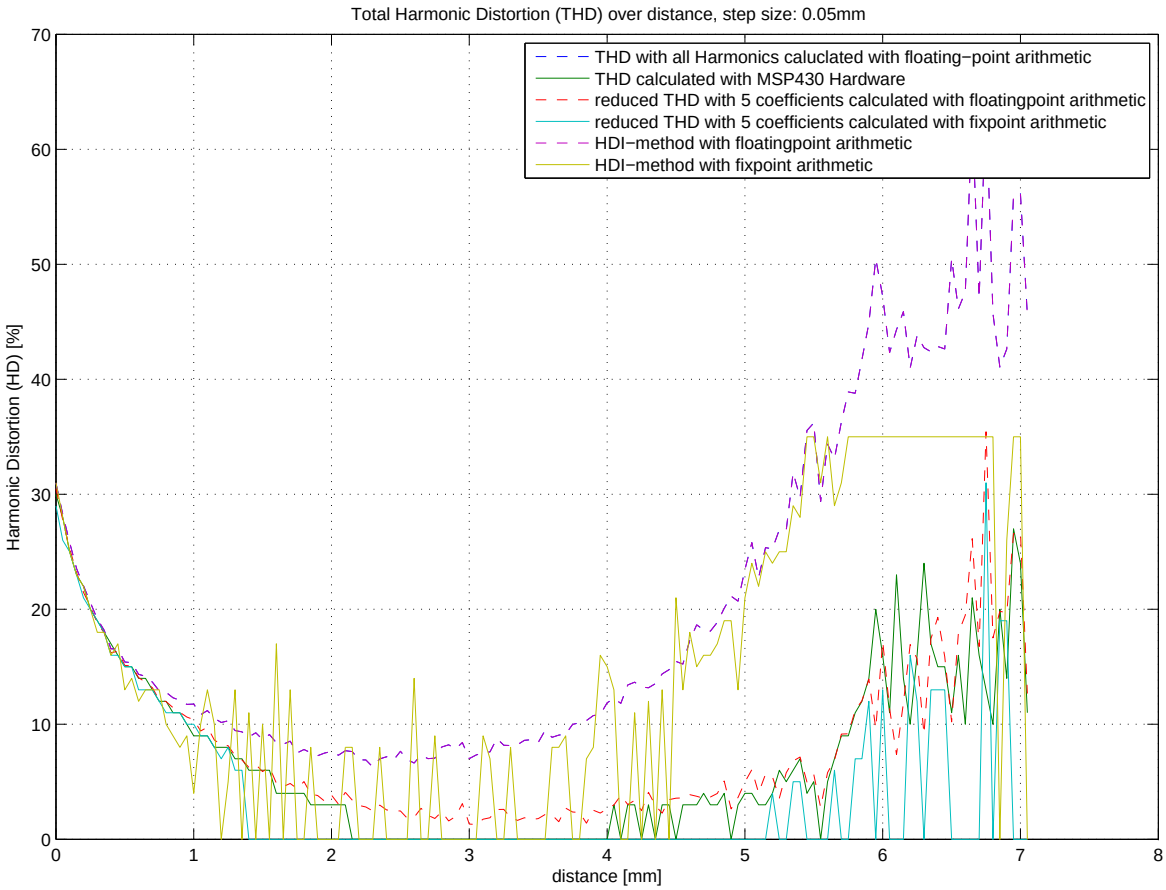

Abbildung 9.5.: Darstellung des Klirrfaktors in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad [8]

Anhand der Abbildung 9.5 lässt sich erkennen, dass der Klirrfaktor nicht mehr zuverlässig ermittelt werden kann. Bei der HD5-Methode (türkis) ist es wohl bei der Berechnung der zweiten bis fünften Harmonischen zu sehr kleinen Ergebnissen gekommen, sodass diese durch Quantisierung Null werden. Deshalb ist der Klirrfaktor bei diesem Berechnungsverfahren an den entsprechenden Stellen Null. Dieses Problem besteht zwar bei Verwendung der HDI-Methode (beiger Verlauf) Berechnung nicht. Stattdessen können die Gesamtleistung sowie die Leistung der ersten Harmonischen des Signals nur mit unzureichender Genauigkeit berechnet werden, da bei geringer Wortbreite NOB die Signal-zu-Rauschleistung SNR kleiner sein muss. Für ein sinusförmiges Signal kann die Signal-zu-Rauschleistung durch folgende Formel bestimmt werden:

$$SNR = NOB \cdot 6,02 + 1,76 \tag{9.2}$$

Im schlimmsten Fall kommt es wegen Genauigkeitsproblemen zu dem Fall, das die erste

Harmonische des Signals größer als die Gesamtleistung ist, kommen. Für den Fall wird die Berechnung abgebrochenen und der Klirrfaktor ist Null.
In Abschnitt 6.4.4 ist beschrieben, dass für die Einhaltung der Spezifikationen der Wurzel-Tabelle 14 Bit, die nicht durch Rauschen beeinflusst sind, ermittelt werden müssen. Das ist in diesem Fall auch nicht mehr gegeben. Eine genauere Analyse der Fehler erübrigt sich für diesen Fall, da die Qualität der Ergebnisse unzureichend ist.

9.3.2. Reduzierung der Wortbreite auf 24 Bit

Im nächsten Schritt soll die Wortbreite auf 24 Bit reduziert werden. Es gibt zwar keinen Standarddatentypen, in dieser Wortbreite. Die Ergebnisse dieser Simulation können für eine spätere Beschreibung mit VHDL von Bedeutung sein, da pro Register 8 Bit eingespart werden können.

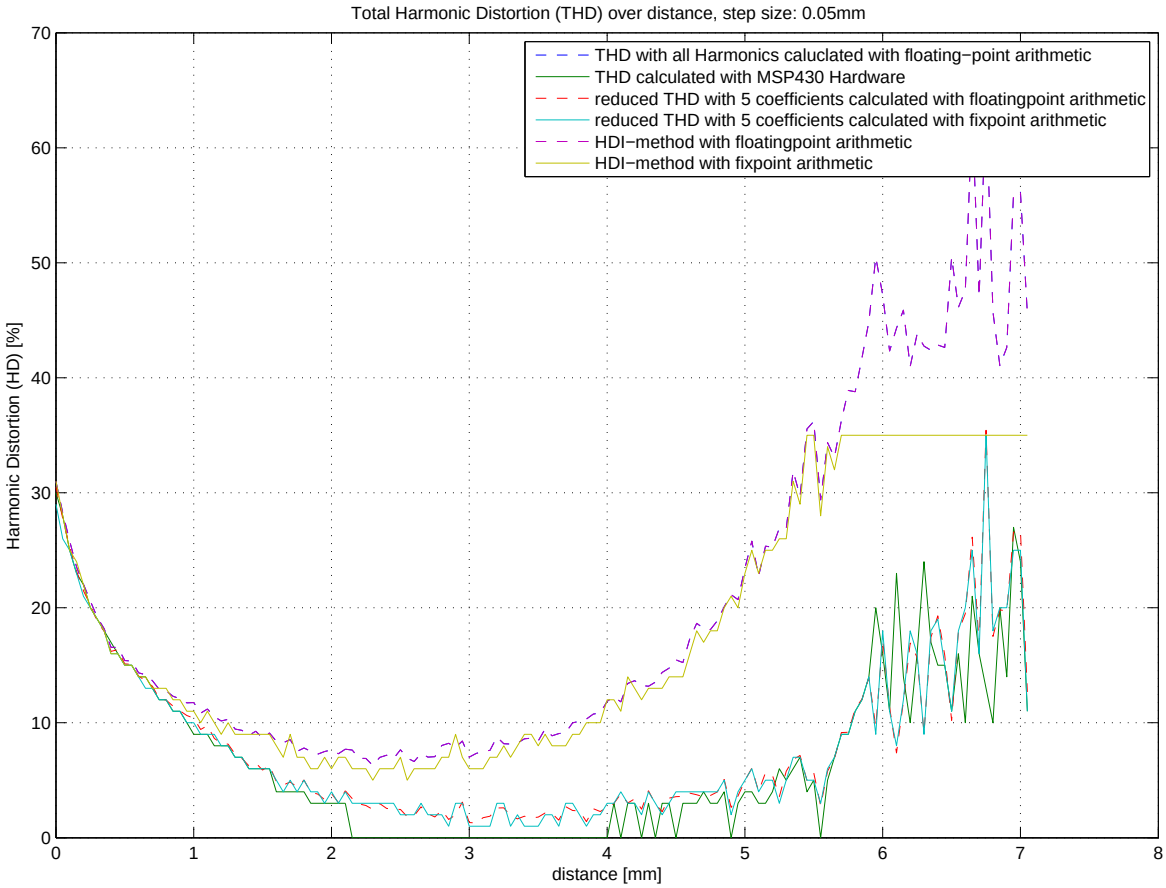

Abbildung 9.6.: Darstellung des Klirrfaktors in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad [8]

In Abbildung 9.6 ist dargestellt, dass Ergebnisse mit denen der ersten Simulation mit einer Wortbreite von 32 Bit, vergleichbar sind. Zur genaueren Bewertung der Ergebnisse werden in den Abbildung 9.7 und 9.8 die Abweichungen der Fixed-Point-Implementierung dargestellt.

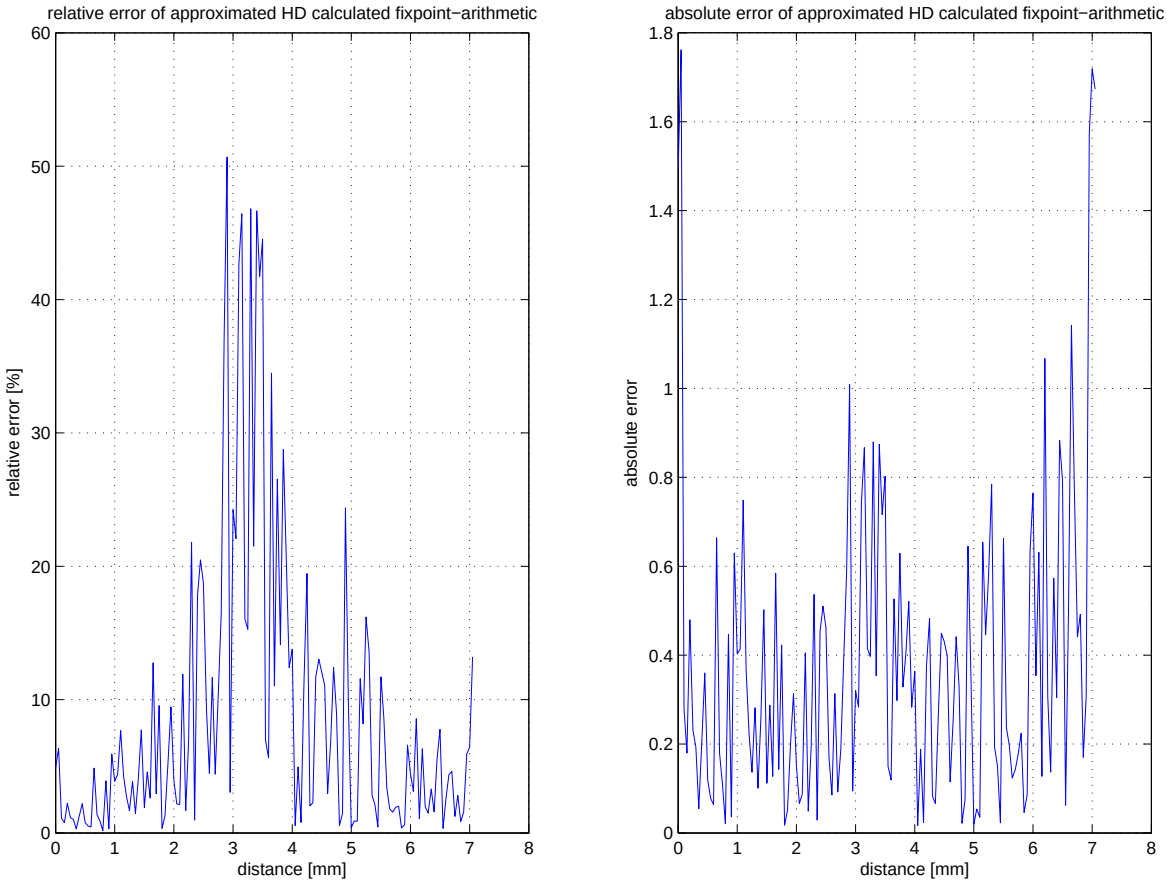

Abbildung 9.7.: Darstellung der absoluten und relativen Abweichungen der HD5-Methode bei Berechnung mit einer Wortbreite von 24 Bit [8]

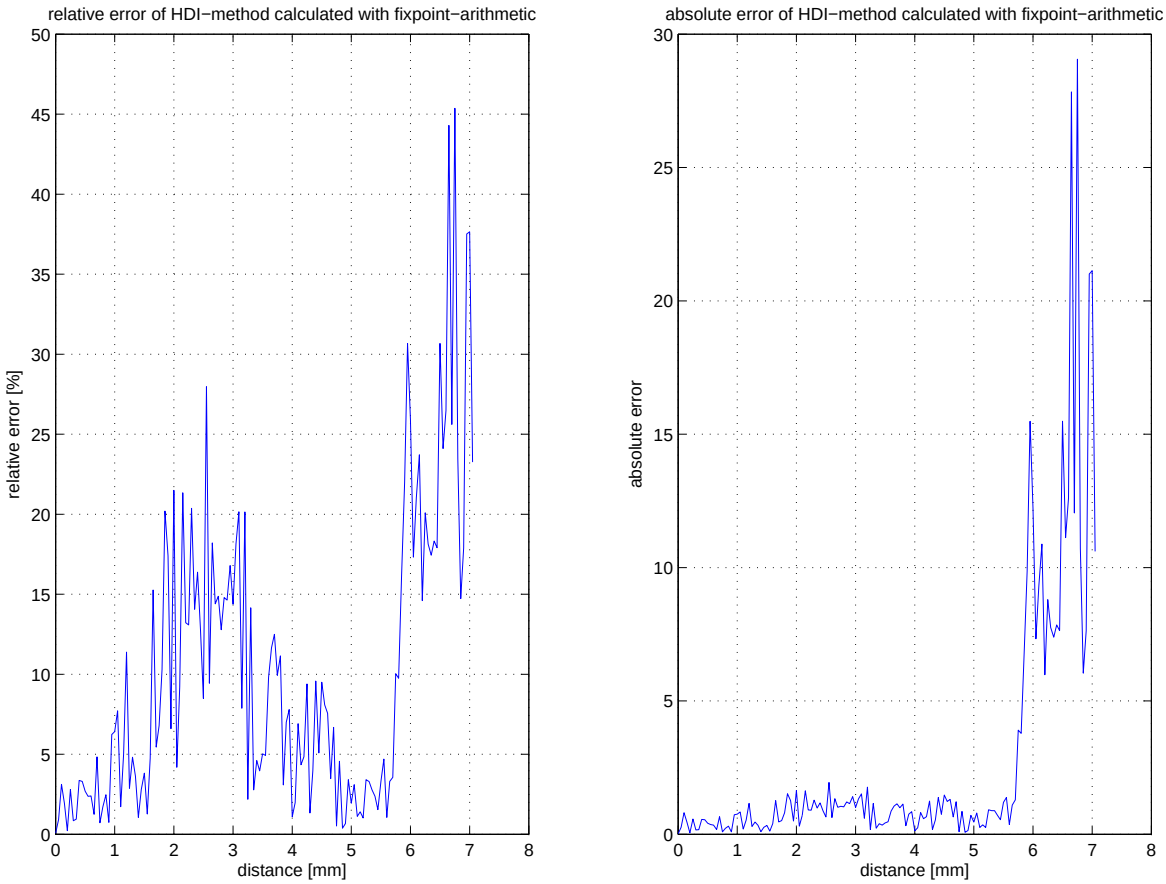

Abbildung 9.8.: Darstellung der absoluten und relativen Abweichungen der HDI-Methode bei Berechnung mit einer Wortbreite von 24 Bit [8]

9.3.3. Bewertung der Simulationsergebnisse

Eine genaue Aussage, wie stark die Wortbreite begrenzt werden kann, ist nach Meinung des Autors nach dem jetzigen Stand des Projektes noch unseriös, da erst genaue Schwellen festgelegt werden müssen, ab welcher Größe des Klirrfaktors für die richtige Funktion des Anti-Blockier-System garantiert werden kann. Erst wenn dieses festlegt ist, ist eine verbindliche Aussage über die Wortbreite möglich.
Wird jedoch davon ausgegangen, dass die jetzigen Spezifikationen an die Auflösung der Ergebnisse, die in Abschnitt 6.1.3 dargestellt sind, endgültig erreicht werden müssen, ist eine Reduzierung der Wortbreite auf 16 Bit nicht zulässig. Es ist jedoch ausreichend die Wortbreite der Register in denen die Werte gespeichert werden auf 24 Bit zu begrenzen. Es kann dann ein 24-Bit-Addierer und ein 16x16-Bit Multiplizierer verwendt werden, dessen Ergebnis auf 24 reduziert werden muss.

9.4. Reduzierung der Eingangswerte

9.4.1. Versuchsbeschreibung

Als nächstes soll eine Simulationsreihe durchgeführt werden, bei der die Auflösung der Sinus- und Kosinus-Tabellen sowie die Auflösung des Analog-Digital-Umsetzer verkleinert wird. Die Sinus- und Kosinus-Tabelle soll eine Wortbreite von 6-, 7-, 8-, 9- und 10-Bit aufweisen. Die Auflösung des Analog-Digital-Umsetzers(ADC) soll eine Auflösung von 6-, 8-, 10- und 12-Bit annehmen. Die Wortbreite der Variablen soll auf 24-Bit begrenzt werden, da sie in dem vorherigen Abschnitt als kleinste sinnvolle Wortbreite experimentell bestimmt worden ist. Weiterhin sind folgende Parameter verwendet worden:

- Anzahl der berücksichtigten Koeffizienten für reduzierten Klirrfaktor: 5
- Wurzel-Tabelle: in 9 Stufen als PCM-Codewort codiert
- Datenquelle: Messwerte des Demonstrators
- Anzahl der Abtastwerte pro Periode: 64
- Es werden beide Berechnungsmögichkeiten der HDI-Methode verwendet.

9.4.2. Ergebnisse des Versuchs

Da diese Simulation sehr umfangreich ist, sind die Ergebnisse dieser Simulation im Anhang in Punkt D.1 dargestellt. Beispielhaft sind in Abbildung 9.9 und 9.10 für beide Berechnungsmethoden die die Ergebnisse unter Verwendung eines 6 Bit ADC und einer Sinus- und Kosinus-Tabelle mit einer Wortbreite von 10 Bit dargestellt:

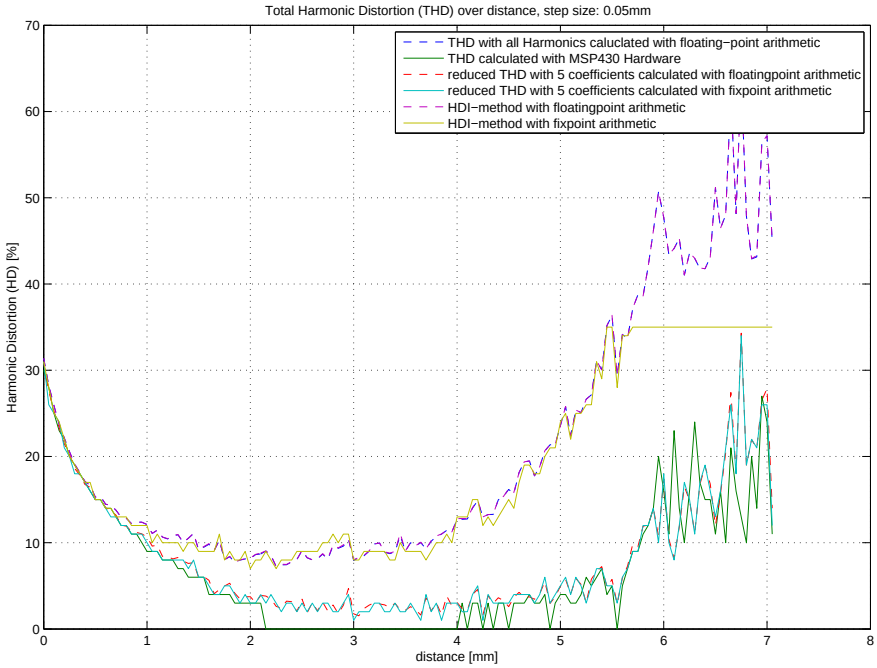

Abbildung 9.9.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung der ersten Berechnungsmöglichkeit [8]

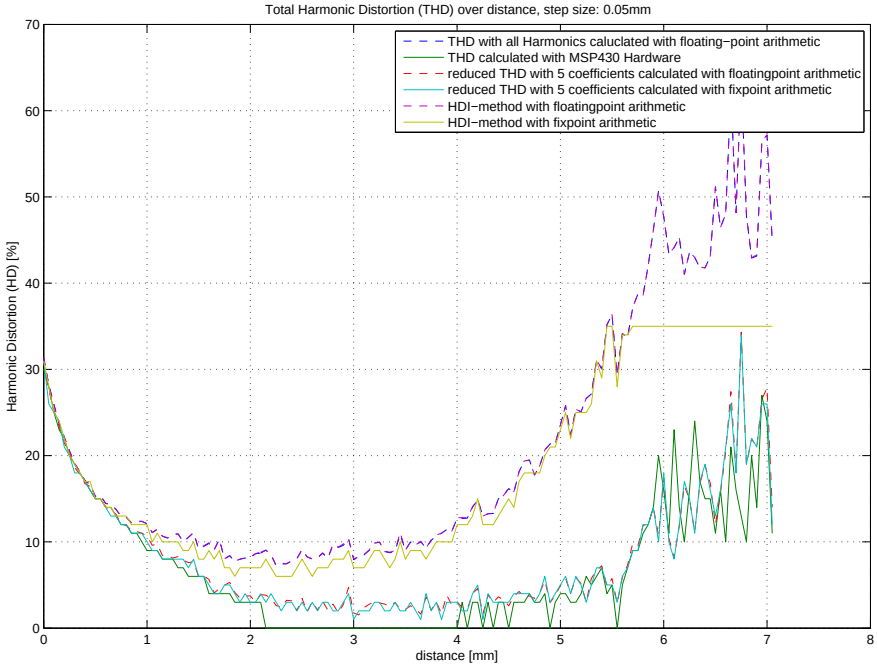

Abbildung 9.10.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung der zweiten Berechnungsmöglichkeit [8]

9.4.3. Auswertung der Versuchs

Bei der Auswertung des Versuchs wurde festgestellt, dass eine Reduzierung der Sinus- und Kosinus-Wertetabellen, gerade für die Ergebnisse, die mit der HDI-Methode ermittelt worden sind, nicht unproblematisch ist. Wenn eine Sinus- und Kosinus-Wertetabelle mit einer Auflösung von 6-Bit und ein Analog-Digital-Umsetzer mit einer Auflösung von 12-Bit verwendet werden, sind die Ergebnisse, wie in Abbildung D.8 bzw. D.24 dargestellt ist, nicht zufriedenstellend. Deshalb ist es eher zu empfehlen, etwas mehr Speicherplatz zu belegen um eine Wertetabelle mit einer Auflösung von 10-Bit abzuspeichern und dafür die Wortbreite des Analog-Digital-Umsetzers auf 8-Bit zu reduzieren.

9.5. Erhöhung der Abtastfrequenz

9.5.1. Versuchsbeschreibung

In einem weiteren Versuch soll die Abtastfrequenz erhöht werden, sodass sich die Anzahl der Samples pro Periode erhöht. Die Wortbreite der Variablen beträgt immer noch 24-Bit. Aus dem letzten Versuch ist eine sinnvolle Auflösung der Sinus- und Kosinus-Tabelle von 10-Bit ermittelt worden. Die Auflösung des ADC soll zwischen 2- und 6-Bit variieren, ebenso die Abtastrate in Zweierpotenzen von $2^7 = 128$ bis $2^9 = 512$.

9.5.2. Ergebnisse des Versuchs

Die Ergebnisse dieses Versuch sind in Anhang D.2 dargestellt. Beispielhaft sind in Abbildung 9.11 und 9.12 für beide Berechnungsmethoden die die Ergebnisse unter Verwendung eines 6 Bit ADC und Ermittlung von 512 Samples pro Periode dargestellt.

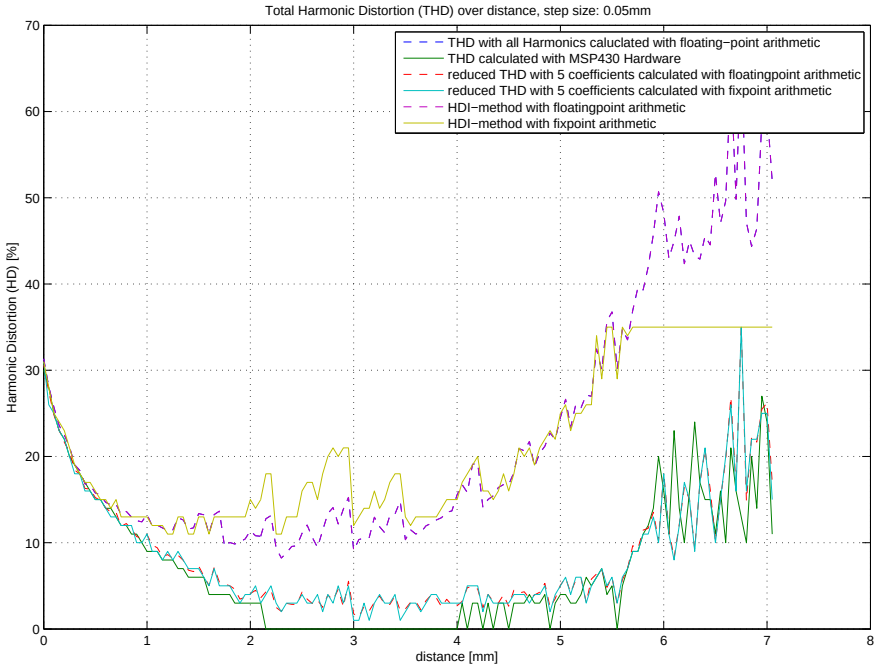

Abbildung 9.11.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

[8]

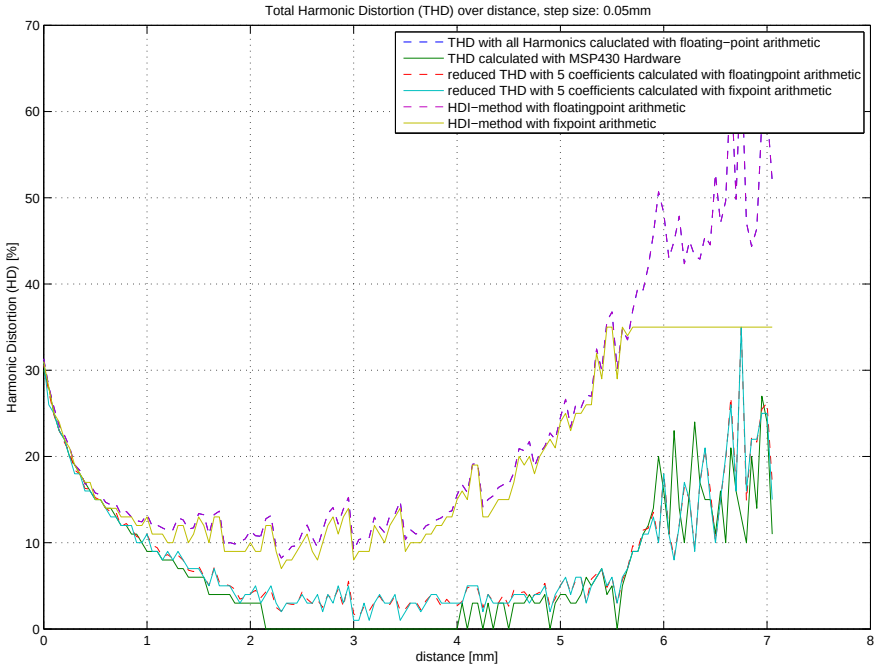

Abbildung 9.12.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode

9.5.3. Auswertung des Versuchs

Bei diesem Versuch wurde festgestellt, dass die Ergebnisse bei Erhöhung der Abtastfrequenz und Berechnung mit begrenzter Wortbreite besser mit den Ergebnissen, die zur Kontrolle mit Gleitkomma-Arithmetik bestimmt worden sind, übereinstimmen. Wenn die Wortbreite Analog-Digital-Umsetzers kleiner als vier Bit wird, sind die Ergebnisse fehlerhaft.

9.6. Vor- und Nachteile der HDI-Methode

In diesem Abschnitt sollen die Vor- und Nachteile der HDI-Methode diskutiert werden. Zu den Vorteilen zählen folgende Punkte:

- Ein großer Vorteil dieses Verfahrens ist, dass die Berechnung weniger Zeit in Anspruch nimmt. Die Messergebnisse können in Kapitel 7 nachgelesen werden.
- Weiterhin ist festzuhalten, dass der Parameter „Anzahl der berücksichtigten Koeffizienten“ für dieses Verfahren keine Bedeutung mehr hat. Es besteht keine Gefahr mehr, dass nicht genügend Harmonische zur Berechnung des Klirrfaktors berücksichtigt werden.

- Ein weiterer Vorteil besteht darin, dass nur noch zwei Sinus- und Kosinus-Tabellen benötigt werden. Das Auswahlverfahren zur Ermittlung des passenden Wertes, das in [7] verwendet wurde, ist nicht mehr erforderlich.

Dem gegenüber stehen folgende Nachteile:

- Ein großer Nachteil dieses Verfahrens ist, dass im Zähler der Gleichung 5.4 eine Differenz gebildet werden muss. Das kann dazu führen, dass die Differenz negativ wird. Das kann ebenfalls auftreten, wenn eine der Quantisierungsfehler des Leistung der 1. Harmonischen, durch die Verwendung einer Sinus- und Kosinus-Tabelle mit geringer Wortbreite, nicht zu vernachlässigen ist. Theoretisch tritt dieser Fall nie ein, da die Leistung der ersten Harmonischen niemals größer werden darf, als die Gesamtleistung des Signals. Wenn jedoch die Sinus- und Kosinus-Tabelle oder die Wortbreite der Variablen reduziert wird, tritt dieser Fall ein (s. Abschnitt9.3.1). Um keinen Absturz des Simulationsprogramms zu verursachen, ist dieser Fall abgefangen, und der Klirrfaktor zu Null gesetzt worden.
- Bei diesem Verfahren ist außerdem der Gleichanteil störend, der dem Ausgangssignal des Sensors überlagert ist. Dieser muss entweder durch eine gute Schätzung oder durch genaue Berechnung ermittelt werden und dann von dem digitalen Eingangssignal abgezogen werden.

10. Bewertung der Ergebnisse

10.1. Abweichungen der Ergebnisse bei Verwendung der HDI-Methode

Anhand der Beobachtungen der Simulation aus Abschnitt 9.2 ist festgestellt worden, dass der Klirrfaktor, der unter Berücksichtigung der ersten fünf harmonischen Schwingungen errechnet wurde, ab 0,5 Millimeter erheblich von dem Klirrfaktor, der mit der HDI-Methode berechnet ist, abweicht.
Eine weitere Simulation, die die aufgezeichneten Daten des Oszilloskops verwendet, zeigt, dass die Approximation des Klirrfaktor durch die HD5-Methode erlaubt ist. Die Ergebnisse sind in Abbildung 10.1 dargestellt.

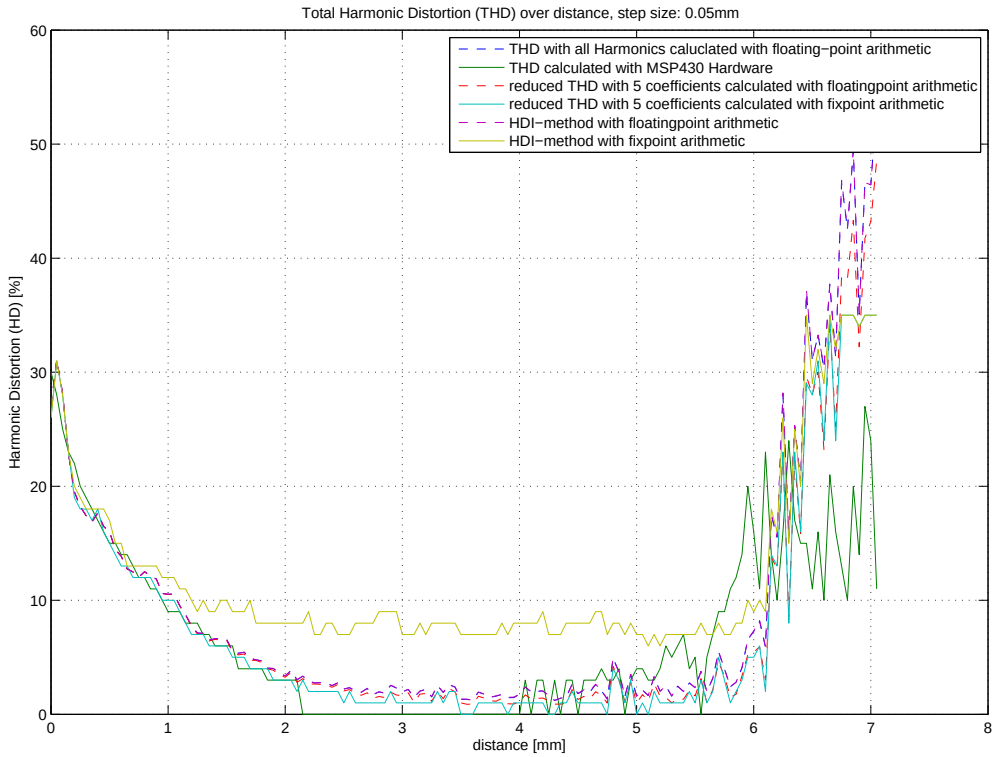

Abbildung 10.1.: Klirrfaktor über die Distanz bei Verwendung von Messdaten des Oszilloskops [8]

Aufgrund dieser Erkenntnis besteht die Vermutung, dass die Abweichung durch die Unterabtastung entsteht, da die Oszilloskopdaten mit einer Samplefrequenz f_s von 250kHz aufgenommen worden sind. Zudem ist dem ADC und dem Oszilloskop derselbe Vorverstärker vorgeschaltet, sodass dieser ebenso als mögliche Fehlerquelle ausgeschlossen werden kann. In diesem Kapitel soll daher die Ursache des Problems erforscht und Lösungen diskutiert werden.

10.2. Untersuchung möglicher Ursachen des Fehlers

10.2.1. HD5-Methode um Harmonische erweitern

In dem ersten Versuch soll geklärt werden, wie viele Harmonische berücksichtigt werden müssen, damit das Ergebnis der HD5-Methode mit den Ergebnissen der HDI-Methode nahezu übereinstimmt. Dazu ist zuerst eine Simulation mit den gleichen Parametern, wie sie in Abschnitt 9.2 verwendet worden, durchgeführt. Der einzige Unterschied ist, dass statt 5

Harmonische jetzt 12 Harmonische berücksichtigt werden. Der Klirrfaktor über die Größe des Luftspalts zwischen Sensor und Encoderrad ist in Abbildung 10.2 dargestellt.

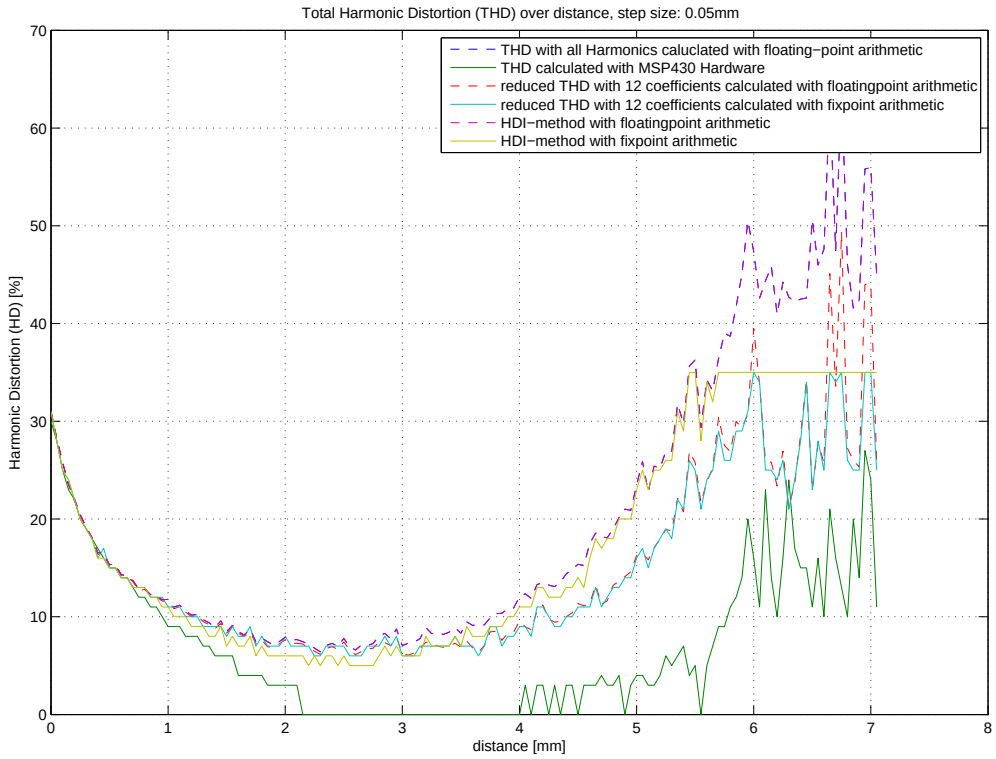

Abbildung 10.2.: Klirrfaktor über die Distanz mit Messdaten des Demonstrators [8]

Bei Betrachtung der Simulationsergebnisse in Abbildung 10.2 ist festzustellen, dass die Klirrfaktorberechnung mit 12 Oberschwingungen mit den Ergebnissen der HDI-Methode-Berechnung wesentlich besser übereinstimmt. Es ist also anzunehmen, dass dem unterabgetasteten Signal Harmonische überlagert sind. Deshalb soll als nächstes eine Analyse des unterabgetastetem Ausgangssignal durchgeführt werden.

10.2.2. Bewertung der Qualität des Eingangssignals

Um diesen Fehler zu untersuchen, wurde zunächst das abgetastete Signal des Demonstrators für verschiedene Entfernungen zum Sensor verglichen. Die erzielten Ergebnisse sind in Abbildung 10.3 und 10.4 dargestellt, die mit dem Matlab-Script „zufaellig_nichtzufaellig.m“ erzeugt worden sind.

Bei Betrachtung der Signale im Zeitbereich ist bereits festzustellen, dass dem Signal bei größeren Entfernungen eine Oberschwingung überlagert ist, die nicht zu vernachlässigen

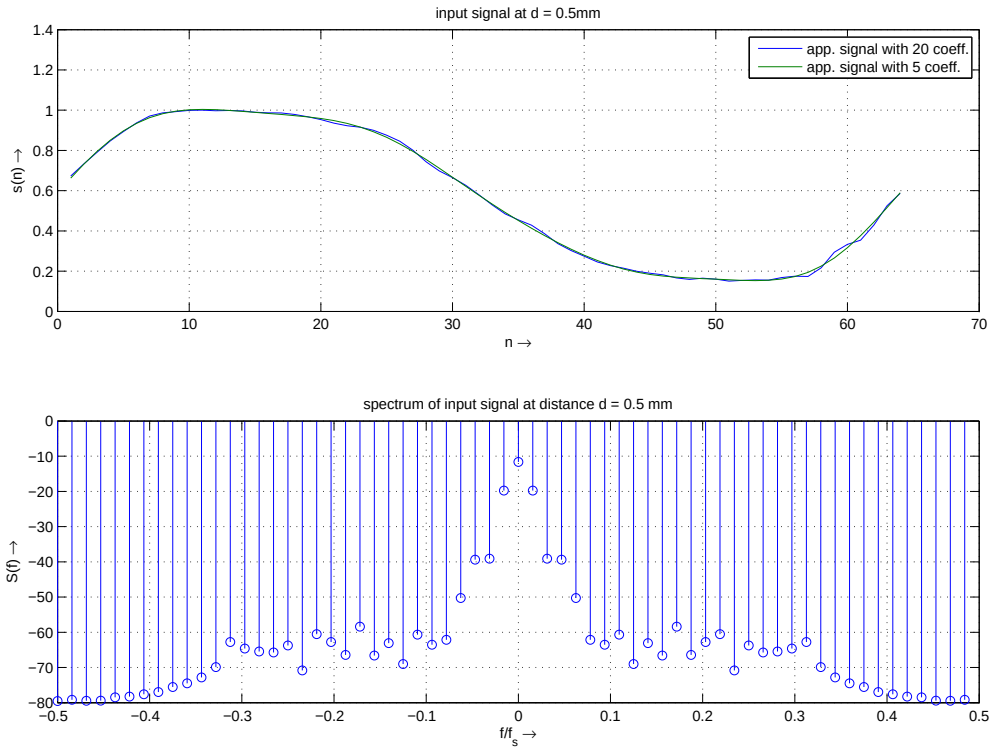

Abbildung 10.3.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 0,5mm

ist. Werden die im unteren Teil des Bildes abgebildeten Spektren untersucht, ist festzustellen, dass Spektralanteile der 7 und 8 Harmonischen an den Stellen $\frac{f}{f_s} = \frac{7}{64} = 0,109$ bzw. bei $\frac{f}{f_s} = \frac{8}{64} = 0,125$ enthalten sind.
Dadurch lassen sich die Abweichungen der Verläufe der Klirrfaktoren erklären. Jetzt stellt sich allerdings die Frage, warum die 7. und 8. Harmonische auftreten. Um diese Frage beantworten zu können, muss das Verfahren zur Datenaufnahme des Demonstrators genauer betrachtet werden. Vor der Durchführung dieser Untersuchung ist es erforderlich, die parallel aufgenommenen Oszilloskopdaten genauer zu analysieren.

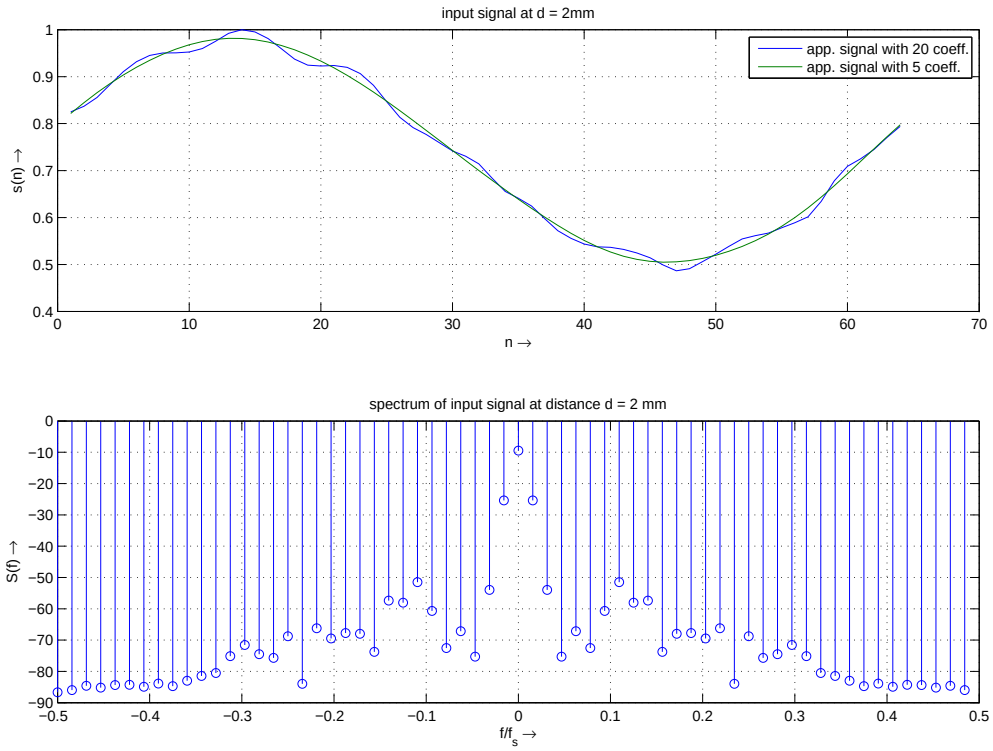

Abbildung 10.4.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 2mm

10.3. Analyse der Amplitude des Sensorsignals über die Zeit

10.3.1. Untersuchung jeder einzelnen Periode des Oszilloskopsignals

In diesem Schritt sollen die Aufzeichnungen des Oszilloskops überprüft werden. Dazu werden die einzelnen Nulldurchgänge detektiert, um anschließend für jede Periode eine Fouriertransformation durchzuführen. Aus diesen Ergebnissen soll dann der Klirrfaktor berechnet und im Zeitbereich dargestellt werden. Zusätzlich ist der Betrag der ersten bis zur fünften Harmonischen sowie der berechnete Klirrfaktor jeder Periode zusammen mit dem Klirrfaktor, der durch den Mikrocontroller bestimmt worden ist, über den Drehwinkel des Encoderrades φ aufgetragen. Dabei wird davon ausgegangen, dass der Drehwinkel φ bei der ersten Periode immer Null ist. Damit die Anzahl der Diagramme überschaubar bleibt, ist die Analyse nur für ein Intervall von 0 - 7 Millimeter in 1 Millimeter Schritten ausgeführt worden. Zudem ist am Radmessplatz eine Einrichtung vorhanden, um zu erreichen, dass das Encoderrad ausgerichtet ist. Das wird im Folgenden als Beseitigung des Unrundlaufs bezeichnet.

Die Simulationen sind jeweils einmal mit und ohne einmal diese Einrichtung durchgeführt worden. Das Simulationsprogramm trägt den Namen rmp_scope_analyze.m. Die Plots sind im Anhang E zu finden. Beispielhaft sind in Abbildung 10.5 und 10.6 die erzielten Ergebnisse bei einer Distanz zwischen Sensor und Encoderrad von einem Millimeter dargestellt:

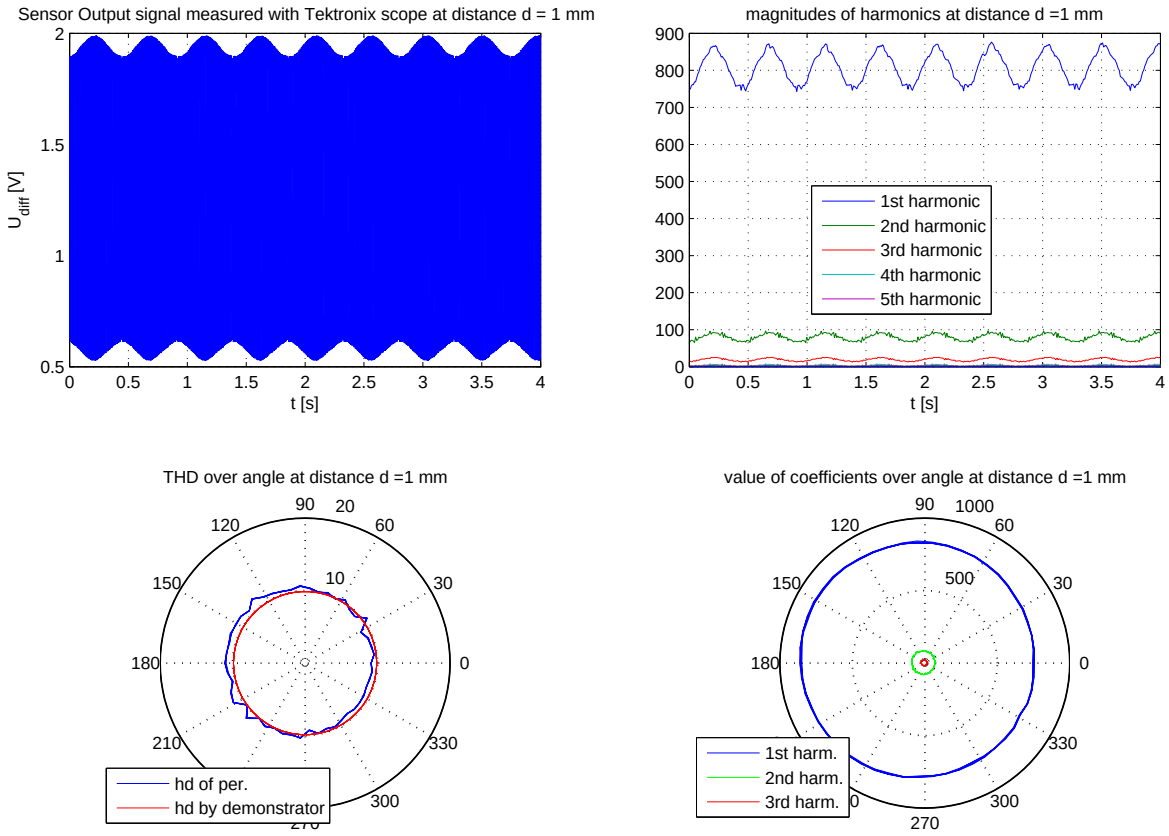

Abbildung 10.5.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 1mm [8]

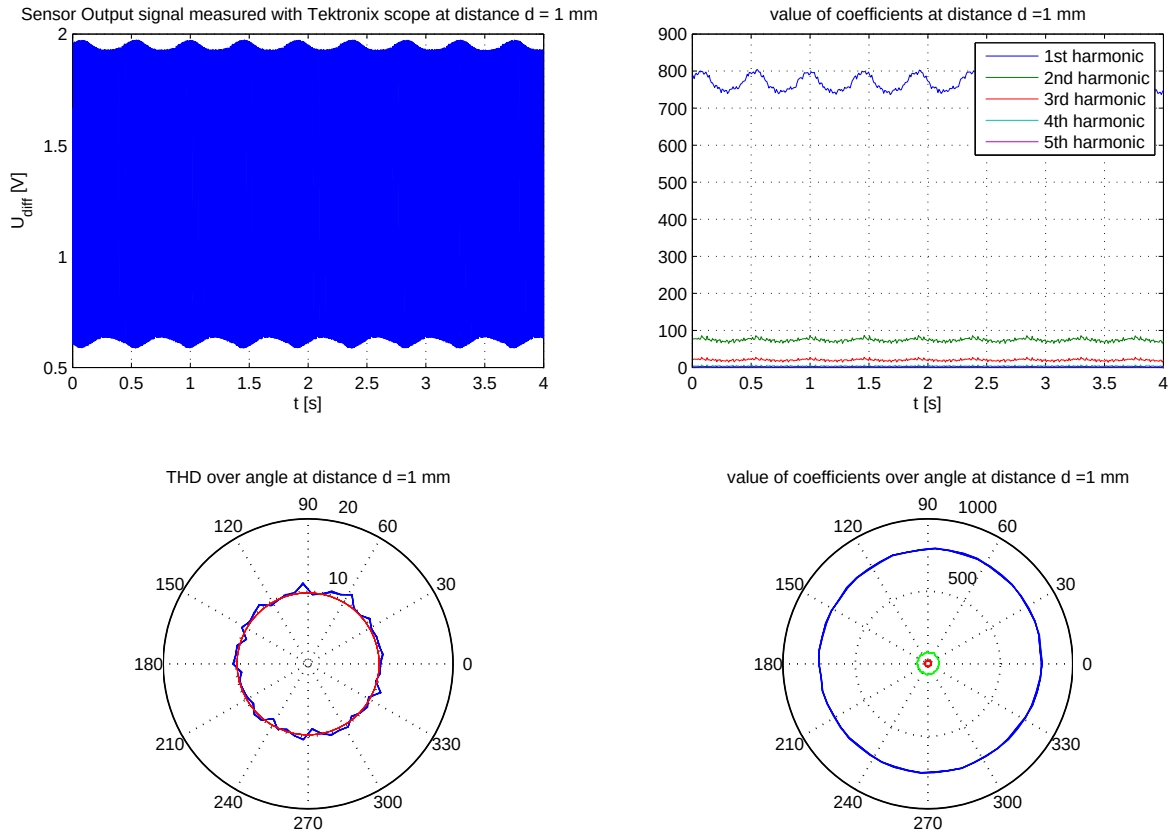

Abbildung 10.6.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 1mm [8]

Es wird festgestellt, dass sich der Betrag der ersten Harmonischen periodisch verändert. Die Periodendauer beträgt ungefähr 0,5 Sekunden. Da ebenfalls die Pulsfrequenz von 107Hz und die Anzahl der Zähne des Encoderrades von 50 bekannt ist, kann die Drehfrequenz f_D des Encoderrades errechnet werden:

$$f_D = \frac{107Hz}{50} = 2,14Hz \tag{10.1}$$

Wird jetzt noch die Periodendauer bestimmt, ergibt sich folgender Wert:

$$T_D = \frac{1}{f_D} = \frac{1}{2,14Hz} = 0,47s \tag{10.2}$$

Dieser Wert ist als Periodendauer in dem Plot „value of coefficients“ wiederzufinden. Daraus lässt sich schließen, dass die Überlagerung der niederfrequenten Schwingung etwas mit

dem sichtbaren Unrundlauf des Encoderrades zu tun hat. Da zudem die ermittelten Klirrfaktoren immer einem bestimmten Zahn des Encoderrades zugeordnet werden können, stützt das zusätzlich die Aussage. Weiterhin ist festzustellen, dass die Amplitude des Signals bei Entfernungen kleiner als 0,5mm weniger stark schwankt, als die Amplitude der Signale bei größeren Entfernungen. Diese Feststellung deckt sich eher mit den Abweichungen aus Abbildung 9.2. Bei kleinen Distanzen stimmt das Ergebnis der HD5-Methode und der HDI-Methode gut überein. Bei einem Millimeter ist eine kleine Abweichung zu erkennen. Je weiter sich der Sensor von dem Encoderrad entfernt, desto stärker sind die Schwankungen der Amplitude des Signals, und die Ergebnisse der beiden Berechnungsverfahren weichen stärker voneinander ab.

10.3.2. Erklärungsversuche der 7. und 8. Harmonischen

Das Ausgangssignal des Sensors wurde mit einem Unterabtastverfahren aufgenommen. Die Unterabtastung ist nur bei Signalen erlaubt, die über die gesamte Zeit T_g periodisch sind. Wie auf den Abbildungen in Kapitel E zu sehen ist, ist dies nicht der Fall, da die Amplitude des Signals sich mit der Zeit ändert. Wie in Punkt 2.2.2 beschrieben, ist der unrunde Lauf des Encoderrrades durch die Drehfrequenz, die $1/50$ der Pulsfrequenz entspricht, gegeben. Der Demonstrator nimmt nur alle 6 - 7 Perioden ein neuen Samplewert auf. Bei 64 Samples, die pro Periode aufgenommen werden, ergibt sich daraus, dass im Mittel $6.5 \cdot 64 = 416$ Perioden berücksichtigt worden sind. Daraus lässt sich schließen, dass sich das Rad eines Autos während der Aufnahme einer Periode um 8 Umdrehungen gedreht haben muss.
Während einer Radumdrehung werden im Mittel $\frac{50}{6.5} = 7,69$ Samples aufgenommen. Werden in diese Formel die minimale und maximale Anzahl der Perioden, in denen ein Wert aufgenommen wird, eingesetzt, ergeben sich folgende Werte:

$$S_{min} = \frac{50}{7} = 7,14 \tag{10.3}$$

$$S_{max} = \frac{50}{6} = 8,33 \tag{10.4}$$

Das bedeutet, dass der 8. bzw. 9. Wert bereits eine Radumdrehung später aufgenommen wird.
Dadurch ergibt sich, dass die Amplitude der ersten Periode, aus der ein Wert abgetastet wurde, ungefähr der Amplitude des 8. bzw. 9. Wertes entspricht. Das wird durch Abbildung 10.7 verdeutlicht:

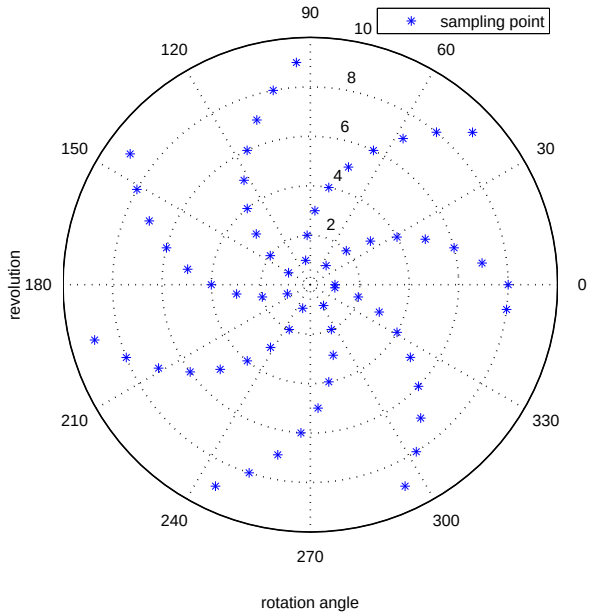

Abbildung 10.7.: Abtastpunkte über den Drehwinkel des Encoderrades aufgetragen bei Aufnahme eines Wertes alle 7 Perioden

Also ergibt sich hieraus eine hochfrequente Oberschwingung, deren Periodendauer $1/8$ bzw. $1/9$ der Periodendauer der Grundschwingung entspricht. Werden diese Erkenntnisse in den Frequenzbereich übertragen, ergibt sich daraus eine Oberschwingung von $\frac{64}{9} \cdot f_0 \approx 7 \cdot f_0$ bzw. $\frac{64}{8} \cdot f_0 = 8 \cdot f_0$.

10.3.3. Simulation der Unterabtastung bei unrund laufendem Encoderrad

Um die Veränderung des Signals, die durch die Unterabtastung entsteht, genauer Untersuchen zu können, wird die Unterabtastung simuliert. Dazu ist das Ausgangssignal des Sensors bei einer Entfernung von 1mm nachgebildet worden. Die verwendeten Werte sind annähernd aus der Abbildung 10.5 ermittelt und in die folgende Formel eingesetzt worden.

$$s = A_0 + \hat{A} \cdot \cos(2\pi t) = 1.25 + \left[0.7 + 0.025 \cdot \cos\left(\frac{2\pi t}{50}\right)\right] \cdot \cos(2\pi t) \tag{10.5}$$

Das nachgebildete Signal ist in Abbildung 10.8 dargestellt. Da in dem Signal nur die Grundschwingung enthalten ist, kann genau überprüft werden, welche Harmonische durch die Unterabtastung in dem Signal auftreten.

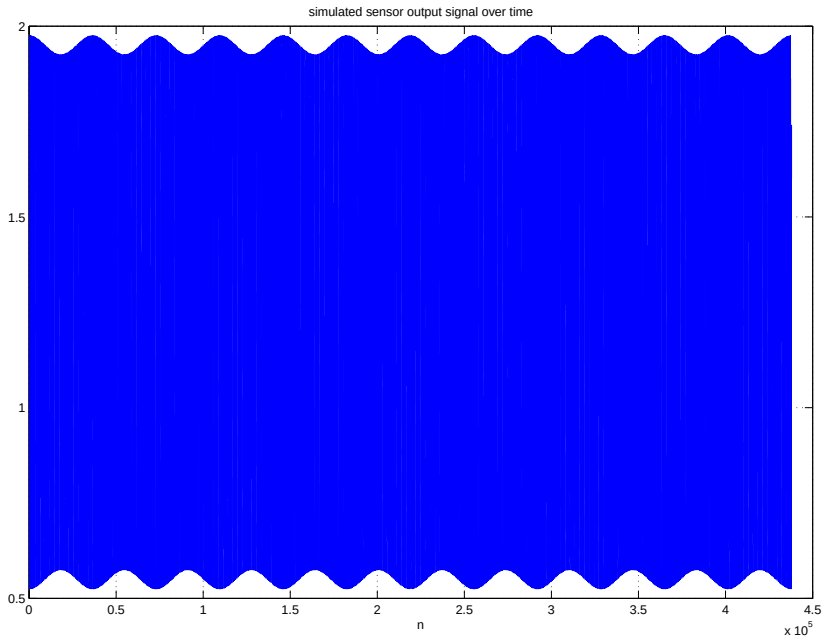

Abbildung 10.8.: Nachgebildetes Ausgangssignal des Sensors bei 1mm Entfernung zum Encoderrad

Die Messung der Periodendauer kann bei dieser Simulation entfallen, da diese beim Erstellen des Signalvektors vorgegeben wird. In diesem Beispiel ist die Abtastfrequenz der Oszilloskopdaten an den Takt des Timers von $12,8\mu$s [7] des MSP430 angepasst worden. Die Anzahl der Samples pro Periode können mit folgender Formel bestimmt werden:

$$N = \frac{T}{T_A} = \frac{f_A}{f} = \frac{1}{f \cdot T_A} \tag{10.6}$$

Werden die Werte in diese Formel eingesetzt, ergibt sich für eine Zahnfrequenz $f_Z = 107$Hz die Anzahl der Samples pro Periode zu $N = 730$.
Wird auch hier alle 6 - 7 Perioden ein Wert abgetastet, entsteht daraus folgendes Signal:

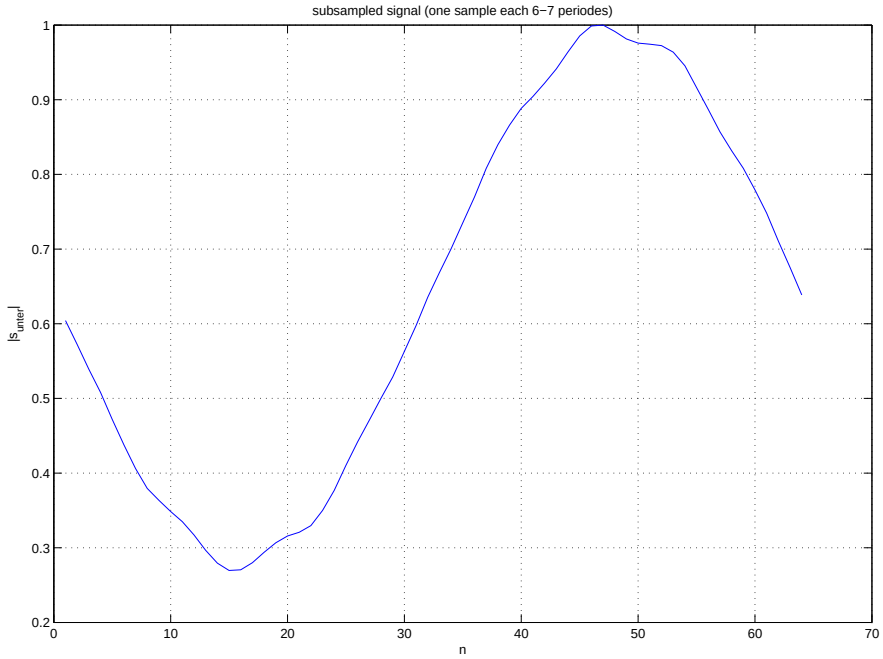

Abbildung 10.9.: Nachgebildetes unterabgetastetes Ausgangssignal des Sensors

Der Abbildung 10.10 ist zu entnehmen, dass dem Signal hochfrequente Störungen überlagert sind. Das wird beim Betrachten des Spektrums in Abbildung 10.10 des Signals deutlicher:

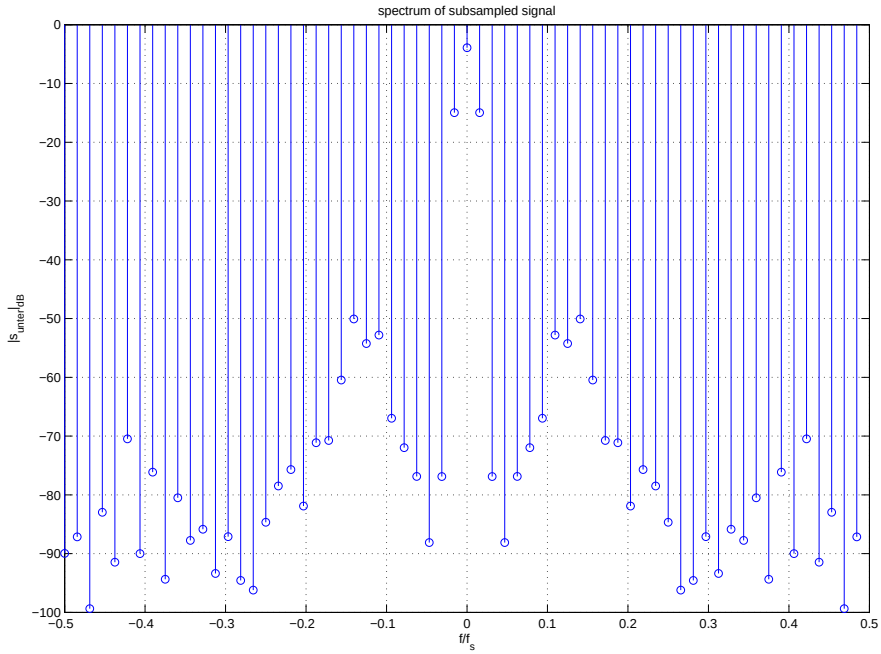

Abbildung 10.10.: Spektrum des nachgebildeten unterabgetasteten Ausgangssignal des Sensors

Die Auswertung des Spektrums des Signals ergibt, dass, wie vorher schon vermutet, Spektralanteile mit 7-, 8-, 9- und 10-facher Grundschwingung hinzugekommen sind.
Mit diesen gewonnenen Erkenntnissen soll versucht werden, die Unterabtastung nicht mit dem simuliertem Signal, sondern mit dem aufgezeichneten Signal des Oszilloskops nachzubilden. Zuvor sollen noch die genauen Abtastpunkte untersucht werden, um so der Entstehung der 7. bis 10. Harmonischen auf die Spur zu kommen.

10.3.4. Nachbildung der Messergebnisse des Demonstrators

Im Rahmen dieses Versuchs ist ein neues Matlab-Programm mit dem Namen „prepare_measurements_sim.m“ erstellt worden. Mit diesem Programm werden die Messdaten des Demonstrators mit der oben vorgestellten Funktion aus den Oszilloskopdaten gewonnen. Da nicht genügend Perioden aufgezeichnet wurden, wird davon ausgegangen, dass das Signal sich alle 400 Perioden wiederholt. Die Bestimmung der Periodendauer wird mit Hilfe der Nulldurchgangserkennung realisiert. Jeder zweite Nulldurchgang beschreibt die Grenzen einer Periode. Als Vorlage dient das in Kapitel 3 erläuterte Programm. Es müssen hier zusätzlich Angaben gemacht werden, wie viele Perioden minimal und maximal verstreichen sollen, bis ein neuer Wert aufgenommen werden soll. Der genaue Wert wird für jeden Abtastpunkt zufällig aus dem angegebenen Bereich bestimmt. Außerdem ist es möglich, die

Samples der Reihe nach aufzunehmen oder die aktuelle Position zufällig zu ermitteln. Diese Funktion wird für einen weiteren Versuch benötigt.
Unter Verwendung der neu erstellten Daten soll die Simulation aus Abschnitt 9.2 erneut durchgeführt werden. Ziel ist es, die Kurvenverläufe dieser Simulation mit den neuen Daten annähern zu können. Die Ergebnisse dieses Versuchs sind in Abbildung 10.11 dargestellt.

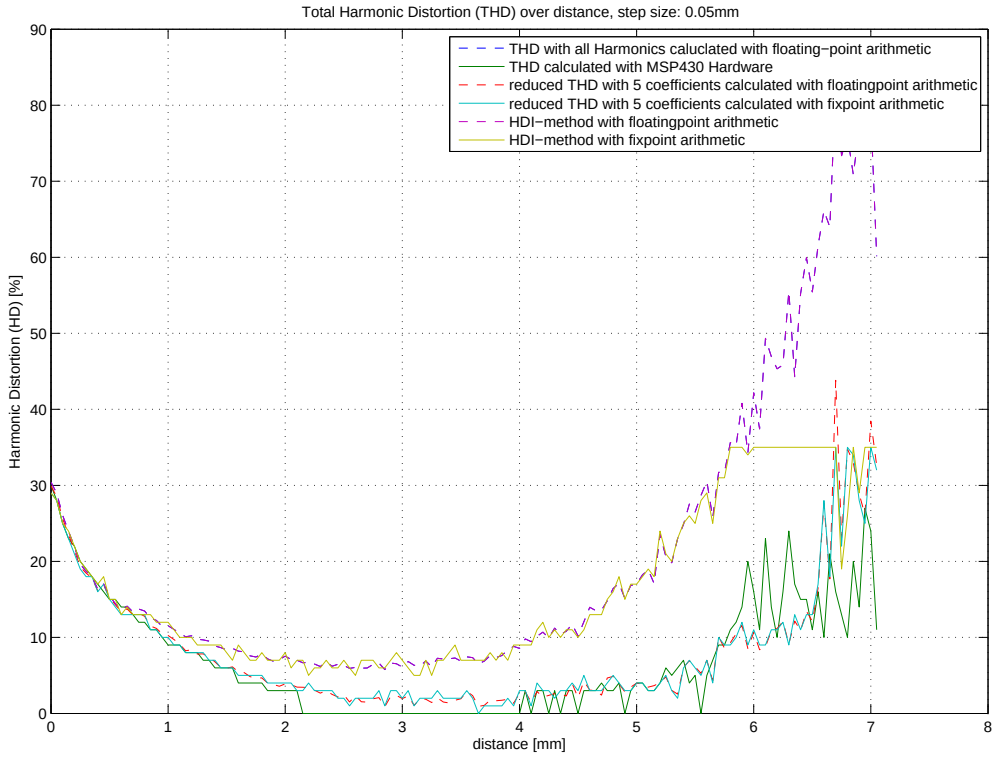

Abbildung 10.11.: Klirrfaktor über die Distanz mit nachgebildeten Messdaten des Demonstrators [8]

10.4. Lösungsvorschläge zur Angleichung der Ergebnisse beider Methoden

10.4.1. Erarbeitete Lösungen

Nachdem das Problem untersucht und die Ursache identifiziert ist, sollen in diesem Abschnitt Lösungsvorschläge diskutiert werden, mit denen es möglich ist, korrekte Ergebnisse unabhängig vom Unrundlauf des Encoderrades zu erzeugen. Aus der zuvor durchgeführten Signalanalyse und dem Studium der Arbeit Jegenhorst [7] sind folgende Lösungsvorschläge erarbeitet worden:

- Das Problem durch zufällige Wahl der Abtastwerte zu unterdrücken
- Die Abtastfrequenz deutlich erhöhen, sodass die Unterabtastung nicht mehr benötigt wird
- Die Samples öfter als alle 6 -7 Perioden aufnehmen
- Das Encoderrad zentrieren
- zufällige Aufnahmereihenfolge der Abtastwerte

10.4.2. Zufällige Wahl der Abtastwerte

Beschreibung der Idee

Die zufällige Wahl der Abtastwerte basiert auf der Idee, die Leistung der 7. und 8. Harmonischen auf alle Harmonischen aufzuteilen. Dieses Verfahren ist in der Arbeit von Herrn Jegenhorst ohne Kenntnis des Problems erwähnt, jedoch nicht realisiert worden. Dem in Kapitel 3 bzw. 10.3.4 beschriebenen Programm ist zu diesem Zweck ein weitere Auswahlmöglichkeit hinzugefügt worden. Wird die zufällige Reihenfolge der Abtastwerte gewählt, wird die Reihenfolge durch einen Zufallsgenerator bestimmt. Mit den neu erstellten Daten ist dann eine Simulation durchgeführt worden.

Ergebnisse bei Realisierung dieser Lösung

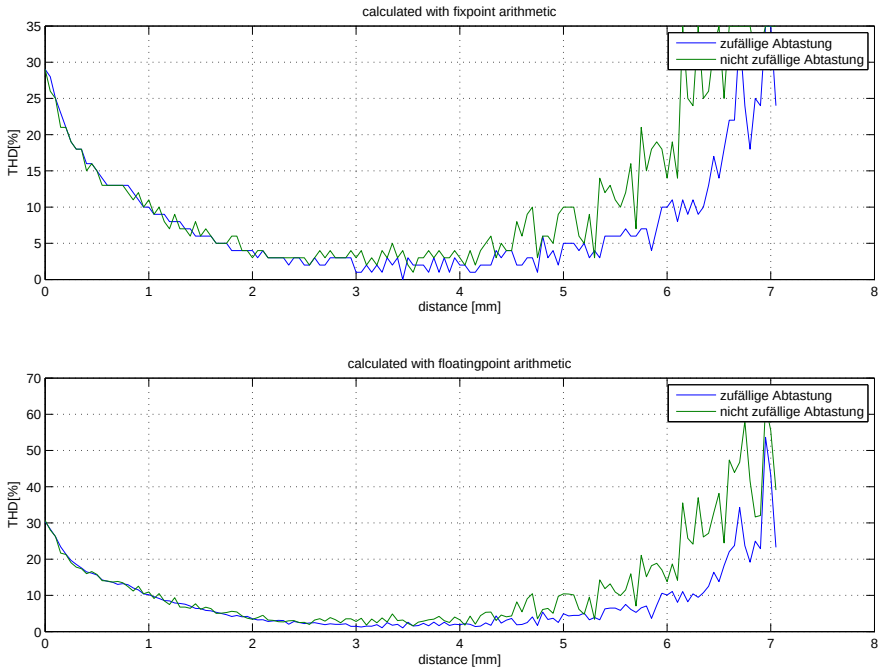

Abbildung 10.12.: Vergleich des HD5-Methodes bei Verwendung von Messdaten in zufälliger und nicht zufälliger Abtastreihenfolge

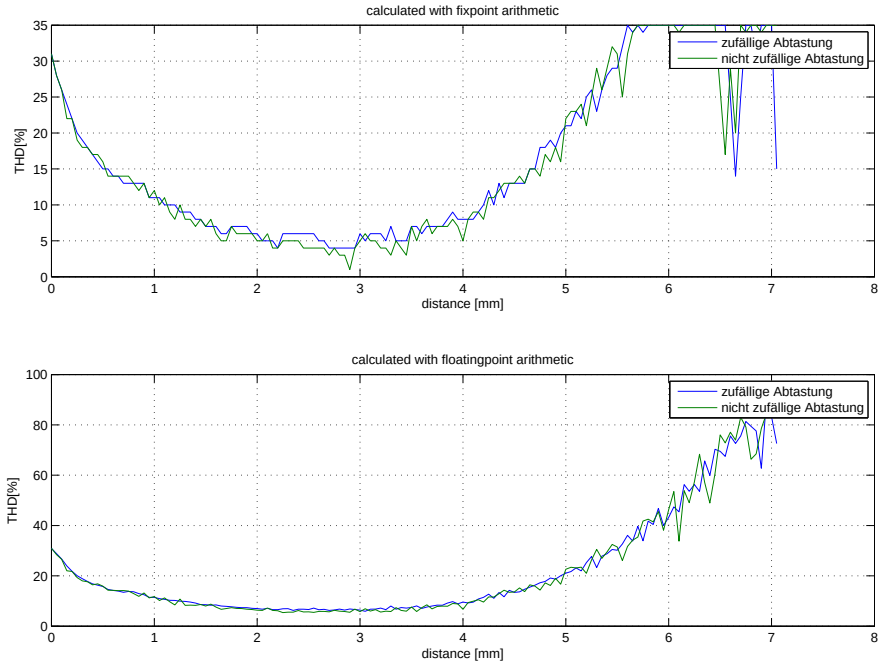

Abbildung 10.13.: Vergleich des Ergebnisses bei Verwendung der HDI-Methode und Samples, die in zufälliger und nicht zufälliger Abtastreihenfolge aufgenommen sind

Bewertung der Ergebnisse

Aus den Ergebnissen der Simulation ist ersichtlich, dass die zufällige Abtastreihenfolge des Signals keine Verbesserung ist. Bei dem HDI-Methode-Berechnungsverfahren tritt eine Verbesserung des Ergebnisses erst ab 4 bis 4,5 mm Entfernung zwischen Sensor und Encoderrad auf. In diesem Bereich wird durch die zufällige Abtastung ein kleinerer Klirrfaktor errechnet.
Abschließend ist festzuhalten, dass die zufällige Abtastung der Werte zu keiner wesentlichen Verbesserung führt, sodass der dadurch bedingte Mehraufwand nicht gerechtfertigt ist und daher als Lösung des Problems nicht in Frage kommt.

10.4.3. Verringerung des Grads der Unterabtastung

Beschreibung des Lösungsvorschlags

Der Demonstrator arbeitet momentan mit einer 6-fachen Unterabtastung. Wird diese zum Beispiel in eine einfache oder zweifache Unterabtastung umgewandelt, werden öfter Samples aufgenommen und für die Aufnahme einer Periode weniger Radumdrehungen benötigt.

Es ist jedoch zu befürchten, dass nicht die 7. und 8. Harmonische auftreten, sondern eine andere Harmonische, die auch die Berechnung nach der HD5-Methode verfälschen kann. Um den Lösungsvorschlag zu bewerten, sind Simulationen mit ein-, zwei-, drei-, und vierfacher Unterabtastung durchgeführt worden.

Ergebnisse des Versuchs

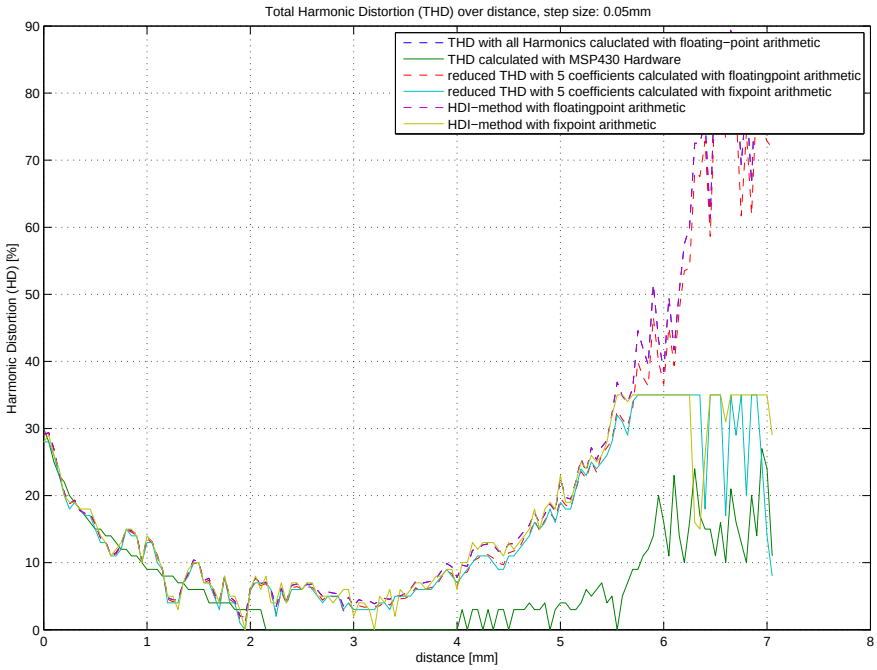

Abbildung 10.14.: Klirrfaktor bei Aufnahme eines Samples pro Periode [8]

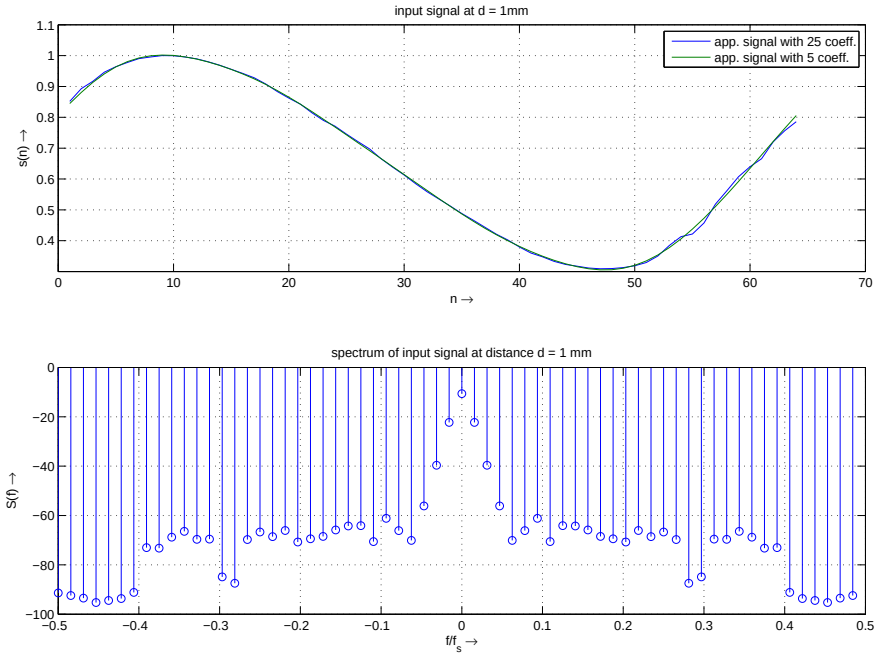

Abbildung 10.15.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 1mm bei Aufnahme eines Samples in jeder Periode

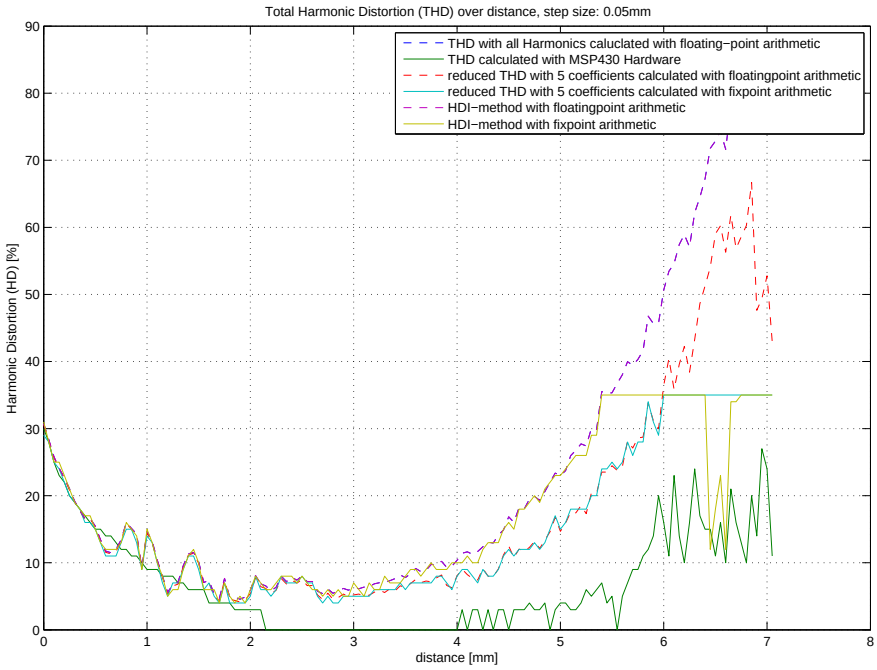

Abbildung 10.16.: Klirrfaktor bei Aufnahme eines Samples alle zwei Perioden [8]

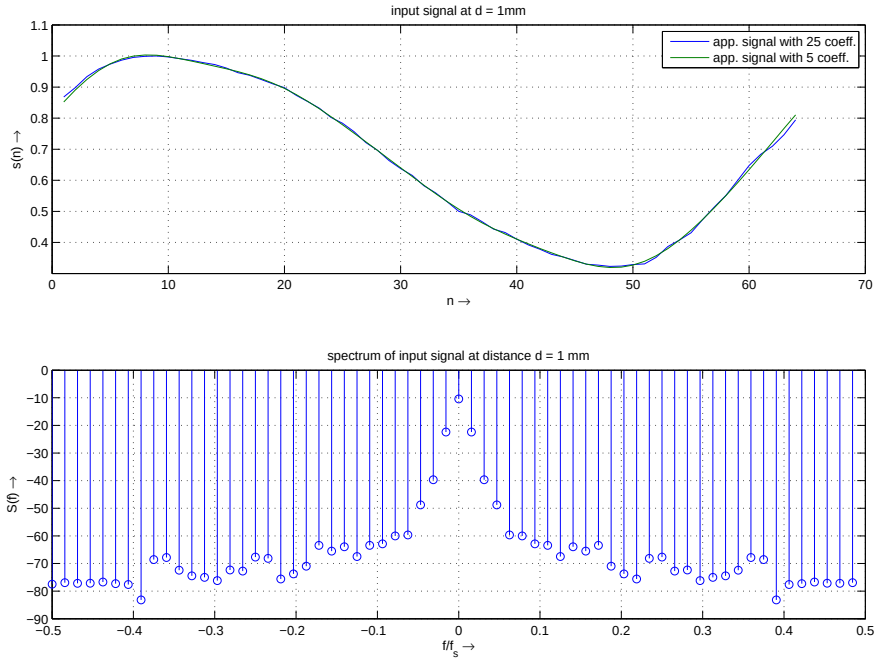

Abbildung 10.17.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 1mm bei Aufnahme eines Samples alle 2 Perioden

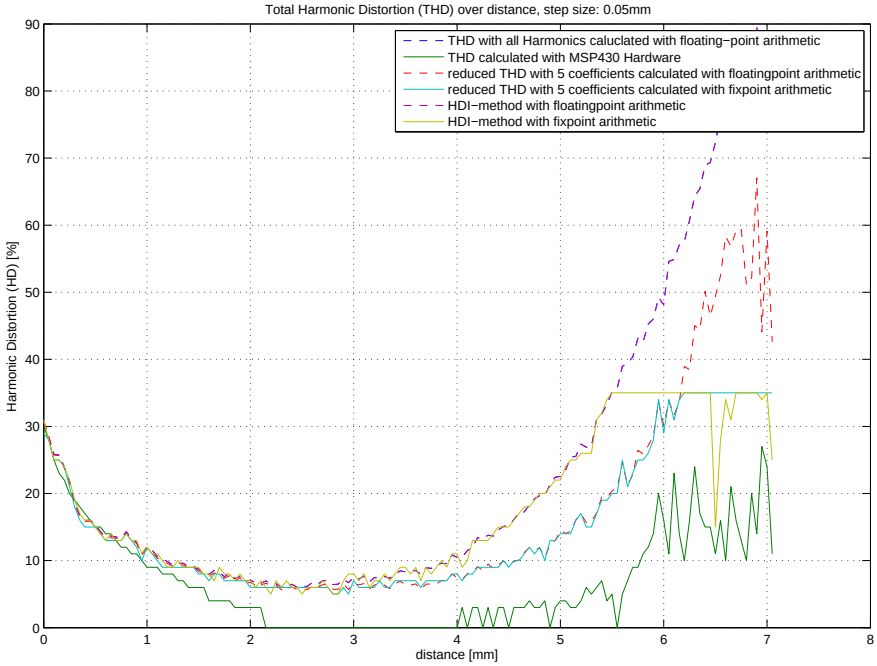

Abbildung 10.18.: Klirrfaktor bei Aufnahme eines Samples alle drei Perioden [8]

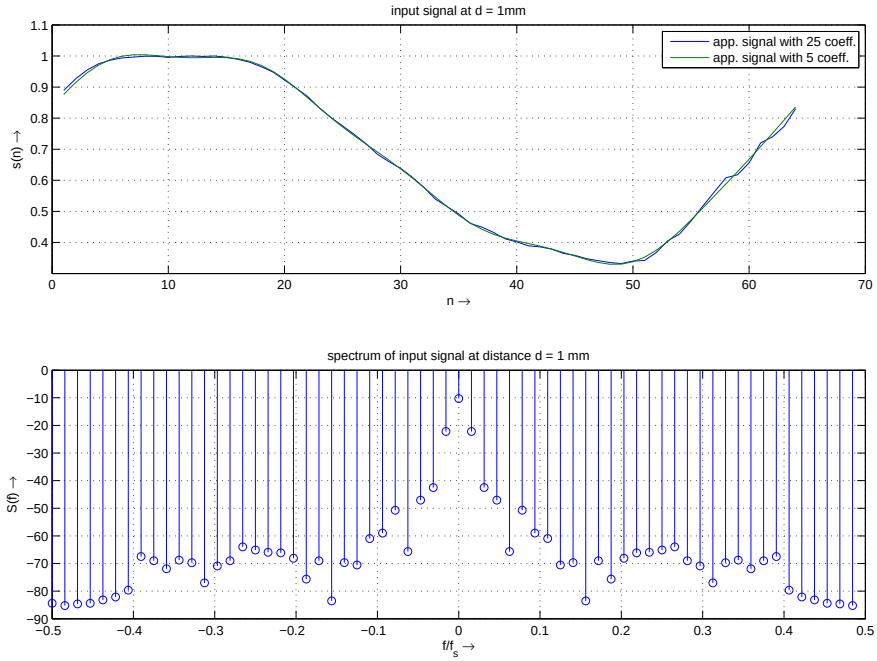

Abbildung 10.19.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 1mm bei Aufnahme eines Samples alle 3 Perioden

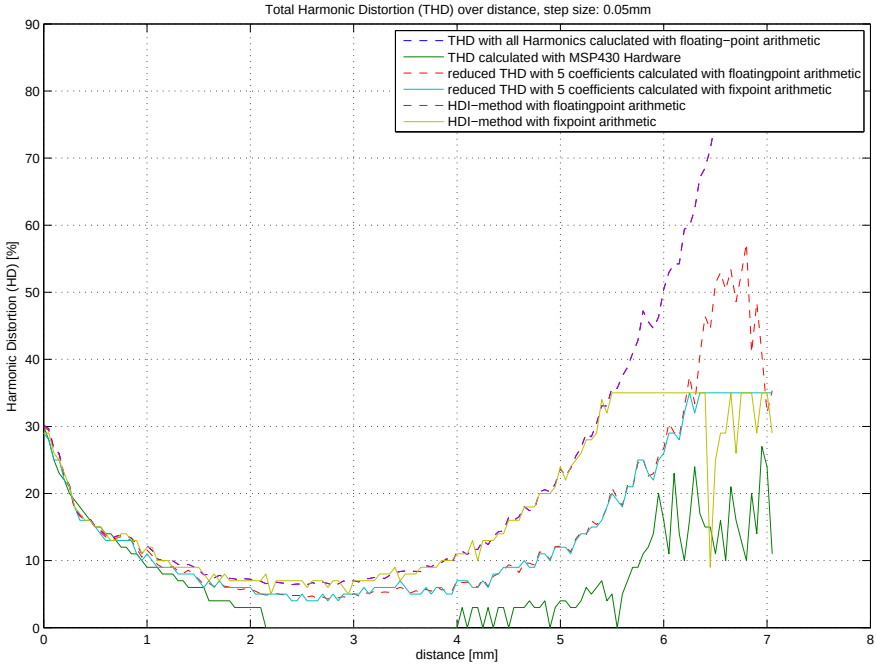

Abbildung 10.20.: Klirrfaktor bei Aufnahme eines Samples alle vier Perioden [8]

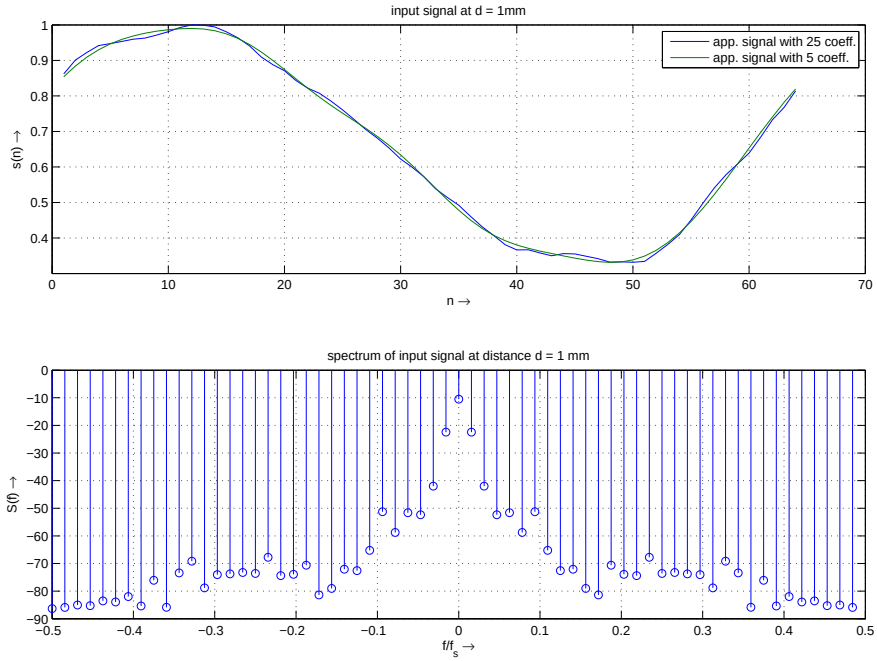

Abbildung 10.21.: Eingangsignal bei einer Entfernung zwischen Sensor und Encoderrad von 1mm bei Aufnahme eines Samples alle 4 Perioden

Bewertung der Ergebnisse

Allgemein kann festgehalten werden, dass bei kleinem Grad der Unterabtastung die Lösungen beider Berechnungsverfahren gut übereinstimmen. Allerdings nähern sich die Ergebnisse nach der HD5-Methode an die Ergebnisse der HDI-Methode an. Je größer der Grad der Unterabtastung ist, desto mehr weichen die Ergebnisse der beiden verwendeten Berechnungsverfahren voneinander ab. Von dieser Lösung ist daher auch abzuraten, da die recht guten Ergebnisse der HD5-Methode verfälscht werden würden.

10.4.4. Den Unrundlauf des Encoderrades beseitigen

Beschreibung des Lösungsansatzes

Um dieses Problem zu analysieren, ist parallel zu dieser Arbeit eine Vorrichtung entworfen worden, mit der es möglich ist, den Unrundlauf des Encoderrades zu minimieren. Danach wurde eine weitere Messung am Radmessplatz durchgeführt, um mit diesen Daten erneut die Simulation mit der identischen Parameter-Datei auszuführen. Fur diesen Versuch stehen Messdaten mit dem Namen 2010_02_10_rmp_03 zur Verfügung

Darstellung der Ergebnisse

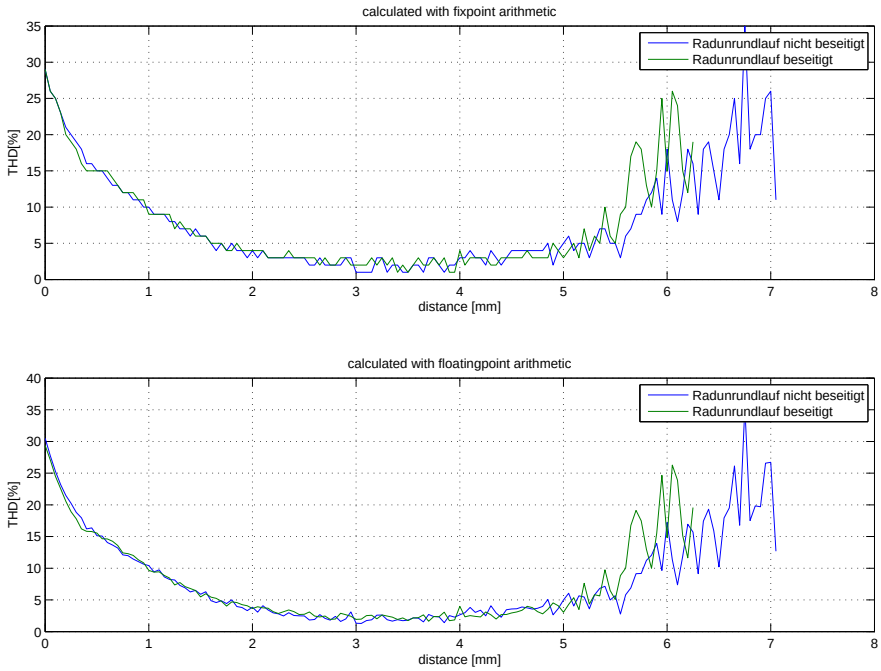

Abbildung 10.22.: Vergleich der Ergebnisse beider Simulation für das HD5-Methode

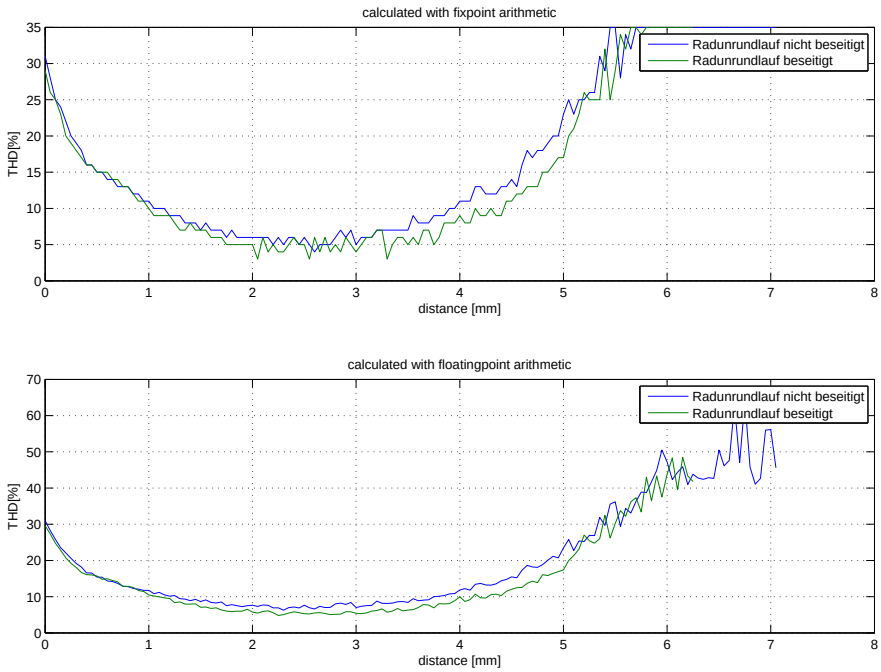

Abbildung 10.23.: Vergleich der Ergebnisse bei Verwendung der HDI-Methode

Bewertung der Ergebnisse

Bei dieser Simulation fällt auf, dass der Radunrundlauf entscheidend in die Ergebnisse bei Verwendung der HDI-Methode einfließt. Durch die Reduzierung des Unrundlaufs näheren sich die Ergebnisse besser an die der HD5-Methode an. Wenn das Encoderrad zentriert wird, fallen die Änderungen der Amplitude geringer aus, sodass diese Lösung in Zukunft verwendet werden könnte. Falls jedoch ein eingebautes Encoderrad zum Beispiel durch einen Steinschlag verbogen wird, müsste es ausgetauscht werden.

10.4.5. Die Unterabtastung vermeiden

Beschreibung der Lösung

Wie an den Simulationsergebnissen, die unter Verwendung der aufgezeichneten Daten des Oszilloskops, das mit einer Abtastrate von 250kHz arbeitet, zu sehen ist, wird dieser Lösungsvorschlag zum Erfolg führen. Methode ist jedoch beim jetzigen Stand des Projektes nicht realisierbar. Weiterhin besteht der Nachteil, dass für die Berechnung des Klirrfaktors nur ein Zahn des Encoderrades berücksichtigt wird. Somit ist diese Variante auch nicht sehr empfehlenswert, da keine Mittelung über mehrere Perioden enthalten ist. Es können zwar

mehrere Berechnungen durchgeführt und in einem weiteren Schritt der Mittelwert der Teilergebnisse berechnet werden. Dieser dient dann als Endergebnis. Dadurch wird jedoch der Aufwand der Berechnung wieder vergrößert.

Darstellung der Ergebnisse

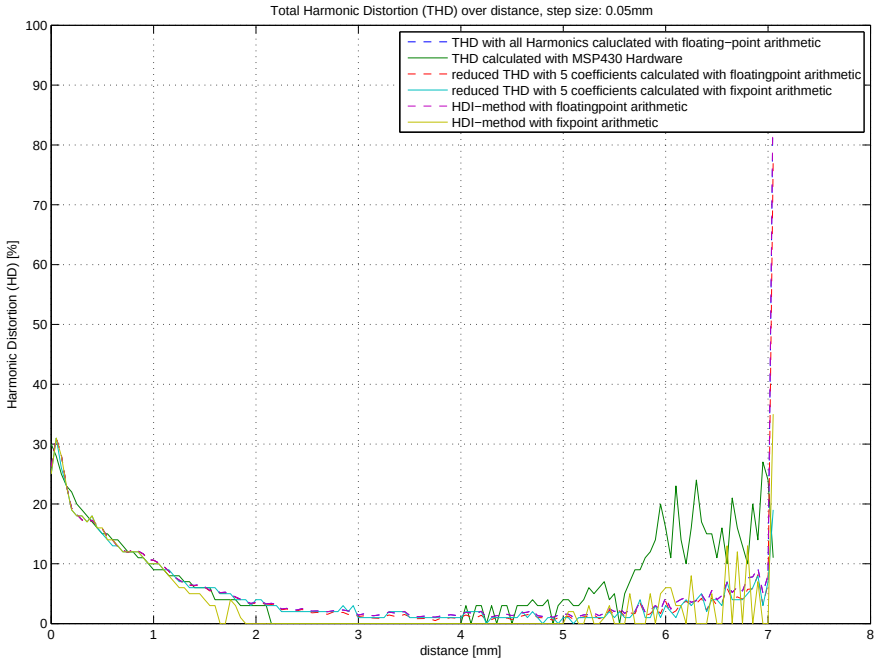

Abbildung 10.24.: Vergleich der Ergebnisse bei Verwendung der HDI-Methode [8]

Bewertung des Ergebnisses

Abschließend kann festgehalten werden, dass dieser Lösungsvorschlag der sicherste ist. Es muss nicht auf den Unrundlauf den Encoderrades geachtet werden. Daher sollte langfristig daran gearbeitet werden, dieses Verfahren zu realisieren, da die anderen Lösungsvorschläge entweder zu keinem guten Ergebnis führen oder sich nicht durchsetzen lassen.

11. Fazit und Ausblick

11.1. Fazit

Bei dieser Arbeit war die Hauptaufgabe, den Aufwand der Schätzung der Harmonischen zu minimieren und zu optimieren. Diese Aufgabe kann als erfüllt angesehen werden. Der Sensor, der zum Abschluss des Forschungsprojekts entwickelt werden soll, darf nur wenige Euro kosten. Daher ist es nötig den Aufwand zu reduzieren und so das System zu verkleinern, um später die Produktionskosten zu senken. Bei diesem Vorgehen darf jedoch nicht die Genauigkeit der ermittelten Klirrfaktoren gefährdet werden.
Dafür ist die HDI-Methode eingeführt worden, mit der der Klirrfaktor zur Zustandsbestimmung von ABS-Sensoren 3,7-mal schneller, als mit der bisher verwendeten HD5-Methode bestimmt werden kann. Die Berechnung der Wurzelfunktion konnte ebenfalls effizienter als in der Arbeit von Herrn Jegenhorst gestaltet worden. Die Messungen, die in dieser Arbeit enthalten sind, wurden ausgewertet und daraufhin neue Spezifikationen für die Genauigkeit der Wurzel-Tabelle aufgestellt worden. Diese haben ergeben, dass die Anzahl der Werte um den Faktor 4 vergrößert werden musste und das Ergebnis eine Wortbreite von 14 Bit aufweisen muss. Durch effizientere Codierung ist die Tabelle von 1024 Werten auf 72 bzw. 64 Werte reduziert worden. Das entspricht einer Reduzierung der Werte um den Faktor 16. Das bedeutet, dass nur noch 6,25% des bisherigen Speicherplatzes benötigt werden.
Damit die bessere Genauigkeit der Wurzel-Tabelle ausgenutzt werden kann, ist die bisher verwendete HD5-Berechnungsmethode in dieser Arbeit optimiert worden. Statt konstante Schiebefaktoren zu verwenden, sind Formeln entwickelt worden, um die Schiebefaktoren in Abhängigkeit von den maximalen benötigten Bits zu berechnen.
Beim Vergleich der Ergebnisse der HDI-Methode mit denjenigen der HD5-Methode ist festgestellt worden, dass die Ergebnisse beider Verfahren ab einem Abstand zwischen Sensor und Encoderrad ab 0,5mm voneinander abweichen. Da dieses Problem völlig unerwartet aufgetreten ist, wurde in dieser Arbeit eine Analyse des Problems durchgeführt.
Um das Problem für den späteren Einsatz der HDI-Methode vermeiden zu können, sind Lösungsvorschläge erarbeitet worden, von denen der Lösungsvorschlag „Den Unrundlauf des Encoderrades zu vermeiden“ oder „Die Unterabtastung zu vermeiden“ als erfolgsversprechend bewertet wurden. Diese konnten jedoch im Rahmen dieser Arbeit nicht genauer untersucht werden, da die technischen Voraussetzungen beim jetzigen Stand des Projekts nicht gegeben sind.
Eine konkrete Aussage darüber, welche Wortbreiten die verwendeten Variablen aufweisen

müssen und welche Auflösung der Analog-Digital-Umsetzers und der Sinus-und Kosinus-Tabellen notwendig sind, konnten in dieser Arbeit ebenfalls nicht getroffen werden, da in Zusammenarbeit mit der Industrie erst festgelegt werden muss, wie genau der Klirrfaktor zu bestimmen ist.

11.2. Ausblick

Während der Bearbeitung der Problemstellung sind einige Fragen und Ideen entstanden, die im Rahmen weiterer Abschlussarbeiten beantwortet bzw. weiter verfolgt werden können.

- Die Änderungen, die im Rahmen dieser Arbeit an de HD5-Methode vorgenommen wurden, müssen in die vorhandene Software integriert werden.
- Um die HDI-Methode weiter untersuchen zu können, muss die vorhandene Software in das Projekt von Herrn Jegenhorst integriert werden. Dazu kann auf die Vorlagen, die zur Ermittlung der Berechnungszeit erstellt worden sind, zurückgegriffen werden.
- Weiterhin ist es interessant zu untersuchen, ob der Unrundlauf des Encoderrades mit dem Präzisionsmessplatz, der im Rahmen einer weiteren Arbeit entstanden ist, geringer als mit dem Prototypen des Radmessplatzes ausfällt. Dazu müssen mit dem neuen Messplatz Daten aufgenommen und unter Verwendung dieser Messergebnisse die Simulationen aus dieser Arbeit erneut ausgeführt werden.
- Dem Unrundlauf des Encoderrades muss durch Maßnamen entgegengewirkt werden. Für die Vermeidung der Unterabtastung ist es erforderlich einen schnellen Analog-Digital-Umsetzer zu verwenden und die Berechnungsvorschrift mit VHDL zu beschreiben, um die Berechnung auf einem FPGA durchzuführen.

Literaturverzeichnis

[1] BARTSCH, Hans-Joachim: *Taschenbuch Mathematischer Formeln*. Fachbuchverlag Leipzig, 2007. – ISBN 3-8273-7067-1

[2] BIERL, Lutz: MSP430 Hardware Multiplier Application Report. (1999). – URL `http://focus.ti.com/lit/an/slaa042/slaa042.pdf`

[3] BOLL, R. ; OVERSHOTT, K. J.: Magnetic Sensors. In: GÖPEL, W. (Hrsg.) ; HESSE, J. (Hrsg.) ; ZEMEL, J. N. (Hrsg.): *Sensors a Comprehensive Survey* Bd. 5. Weinheim : VCH, 1989. – insb. Kap. 9 von U. Dibbern. – ISBN 3-527-26771-9

[4] BRANDT, Philip ; PETERS, Garlef ; KOCH, Lennart: *Local Positioning System*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2009

[5] DRESCHHOFF, Jan-Heiner: *FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2010

[6] ENGEL, Björn ; DALAN, Marco: *Prius-Rückruf: Das steckt hinter dem Brems-Debakel bei Toyota*. 09.03.2010. – URL `http://www.welt.de/motor/article6329398/Das-steckt-hinter-dem-Brems-Debakel-bei-Toyota.html`

[7] JEGENHORST, Niels: *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2009

[8] KOCH, Lennart: *Anhänge und Quelltexte der Diplomarbeit (bei Prof. Vollmer hinterlegt)*. 2010

[9] LIPINSKI, Klaus ; LACKNER, Hans ; LAUÉ, Oliver P.: *ITWissen-Das große Onlinelexikon*. 19.03.2010. – URL `http://www.itwissen.info/definition/lexikon/Klirrfaktor-THD-total-harmonic-distortion.html`

[10] LIPINSKI, Klaus ; LACKNER, Hans ; LAUÉ, Oliver P.: *ITWissen-Das große Onlinelexikon - THD+N (total harmonic distortion plus noise)*. 19.03.2010. – URL `http://www.itwissen.info/definition/lexikon/total-harmonic-distortion-plus-noise-THD-plusN.html`

[11] MAHTOUF, Abdelkhalek: *Messungen und Signalanalyse an einem magnetischen Sensor*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Dezember 2008

[12] N.N: *Beschreibung des Evaluation-Boards vom Typ MSP430-169STK*. 16.04.2010. – URL `http://www.olimex.com/dev/msp-169stk.html`

[13] REDEMANN, P. ; NEUNER, F.: *Lehrbrief Kommunikationstechnik - Grundlagen der PCM-Technik*. 2004. – URL `http://telecom.htwm.de/telecom/praktikum/pcm/lehrbrief-pcm2004.pdf`

[14] SCHMEISSER, Fritz ; DIETMAYER, Klaus: Rotational Speed Sensors KMI15/16. (1999)

[15] SCHOERMER, Christian: *Konstruktion und Automatisierung eines Radmessplatzes für ABS-Sensoren unterschiedlicher Hersteller*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2010

[16] SEMICONDUCTORS, Philips: Magnetoresistive sensors for magnetic field measurement. (2000)

[17] SENGPIEL, Dipl.-Ing. E.: *Tontechnik Rechner - sengpielaudio*. 02.02.2010. – URL `http://www.sengpielaudio.com/Rechner-harmonische.htm`

[18] SIEBENMORGEN, Frank: *Ansteuerelektronik und Mikrocontrollersteuerung eines Kreuzspulenmessplatzes für ABS-Sensoren*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Juni 2009

[19] STAHL, Martin: *Controllersystem zur Verstärkung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion*, Hochschule für Angewandte Wissenschaften Hamburg, Bachlorthesis, 2010

[20] STURM, Mathias: *Mikrokcntrollertechnik am Beispiel der MSP 430-Familie*. Carl Hanser Verlag, 2006. – ISBN 3-446-21800-9

[21] WERNER, Martin: *Digitale Signalverarbeitung mit MATLAB 4. durchgesehene Auflage*. Vieweg+Teubner, 2009. – ISBN 978-3-8348-0457-0

[22] WIESINGER, Johannes: *Aktive Rad-Drehzahlsensoren von BOSCH*. 11.04.2010. – URL `http://www.kfztech.de/kfztechnik/elo/sensoren/drehzahlsensor.htm`

A. mathematische Beweise

A.1. Symmetrie eines Spektrums

In diesem Abschnitt soll die Symmetrieeigenschaft eines Spektrums bewiesen werden. Dazu wird in die allgemein gültige Gleichung 3.1 für k der Ausdruck $M-k$ eingesetzt. Man erhält dann folgende Gleichung:

$$\underline{X}_{M-k} = \frac{1}{M}\sum_{n=0}^{M} s[n] \cdot e^{-j\frac{2\pi[M-k]n}{M}} \tag{A.1}$$

werden die Gleichung aufgelöst, ergibt sich folgender Ausdruck:

$$\underline{X}_{M-k} = \frac{1}{M}\sum_{n=0}^{M} s[n] \cdot \underbrace{e^{-j\frac{2\pi Mn}{M}}}_{=1} \cdot e^{j\frac{2\pi kn}{M}} \tag{A.2}$$

Die Gleichung lässt sich jetzt wieder vereinfachen, sodass sich für $\underline{X}_{M-k}$ folgender Ausdruck ergibt:

$$\underline{X}_{M-k} = \frac{1}{M}\sum_{n=0}^{M} s[n] \cdot e^{j\frac{2\pi kn}{M}} = M_k \tag{A.3}$$

A.2. Äquivalenz der Summe der Quadrate und der Summe der Beträge der Fourierkoeffizenten

In diesem Abschnitt soll bewiesen werden, dass das Ergebnis der Summe aller Samples eines Signals und die Summe der Beträge aller Fourierkoeffizienten gleich ist. Das kann mathematisch wie folgt ausgedrückt werden:

$$\sum_{n=0}^{N-1} s_n^2 = \frac{1}{K} \cdot \sum_{k=0}^{K-1} |X_k|^2 \tag{A.4}$$

die Fourierkoeffizienten X_k können mit Hilfe der Fouriertransformationen aus den Samples des Signals s_n bestimmt werden. Die Formel dafür lautet:

$$X_k = \sum_{n=0}^{N-1} \left[s_n \cdot e^{j \cdot \frac{2\pi kn}{N}} \right]^2 \tag{A.5}$$

Wird Gleichung A.5 in Gleichung A.4 eingesetzt, erhält man folgende Gleichung:

$$\sum_{n=0}^{N-1} s_n^2 = \frac{1}{K} \cdot \sum_{k=0}^{K-1} \left| \sum_{n=0}^{N-1} \left[s_n \cdot e^{j \cdot \frac{2\pi kn}{N}} \right]^2 \right| \tag{A.6}$$

Diese Formel kann zu folgendem Ausdruck umgeformt werden:

$$0 = \frac{1}{K} \cdot \sum_{k=0}^{K-1} \left| \sum_{n=0}^{N-1} \left[s_n \cdot e^{j \cdot \frac{2\pi kn}{N}} \right]^2 \right| - \sum_{n=0}^{N-1} s_n^2 \tag{A.7}$$

Die Rechenregeln für Summen erlauben es, dass die beiden Summen des ersten Term vertauscht werden dürfen. Dann kann die Formel weiter zusammengefasst werden, und es ergibt sich folgender Ausdruck:

$$0 = \sum_{n=0}^{N-1} s_n^2 \left[\frac{1}{K} \cdot \sum_{k=0}^{K-1} \left| e^{j \cdot \frac{2\pi kn}{N}} \right|^2 - 1 \right] \tag{A.8}$$

Damit diese Bedingung erfüllt werden kann, muss der Ausdruck in der Klammer zu Null gesetzt werden:

$$0 = \frac{1}{K} \cdot \sum_{k=0}^{K-1} \left| e^{j \cdot \frac{2\pi kn}{N}} \right|^2 - 1 \tag{A.9}$$

oder anders ausgedrückt

$$1 = \frac{1}{K} \cdot \sum_{k=0}^{K-1} \left| e^{j \cdot \frac{2\pi kn}{N}} \right|^2 \tag{A.10}$$

Da es sich bei dem Ausdruck $e^{j \cdot \frac{2\pi kn}{N}}$ um einen komplexen Zeiger handelt, dessen Betrag immer 1 ist, darf das Gleichheitszeichen geschrieben werden und die Behauptung ist somit wahr.

B. Wissenswertes zur Festkommaberechnung mit Matlab

B.1. Grundlagen der Matlab Fixpoint Toolbox

B.1.1. Aufbau einer Fixpoint-Variablen

Eine Fixpoint-Variable besteht aus einer Datenstruktur, die folgende Strukturelemente enthält:

- bin: Der Wert wird binär als String zur Verfügung gestellt.
- data: Gleitkommawert des Fixpoint Wertes wird in einer numerischen Variablen zur Verfügung gestellt.
- dec: Der Dezimalwert des Wertes wird als Zeichenkette zur Verfügung gestellt
- double: Gleitkommawert wird in einer „double“ Variable zur Verfügung gestellt.
- hex: Der Wert wird hexadezimal in einer Zeichenkette zur Verfügung gestellt.
- int: Der Inhalt der Variablen wird als Integer Zahl zur Verfügung gestellt. Alternativ kann dieses Ergebnis auch durch das Benutzen der folgenden Funktionen erreicht werden: int8, int16, int32, int64, uint8, uint16, uint32 und uint64.
- oct: Der Wert wird oktal in einer Zeichenkette zur Verfügung gestellt.

Um eine Fixpoint-Variable zu erstellen, steht in Matlab der Befehl fi() zur Verfügung. Um zum Beispiel der Variable a ohne Veränderung der Eigenschaften einen neuen Wert zuzuweisen, muss folgende Anweisung geschrieben werden: a(:) = 48.
Wird zum Beispiel in die Matlab Kommandozeile der Befehl a = fi(32) eingegeben, wird eine Fixpoint-Variable a definiert und mit dem Wert 32 initialisiert. Es werden zudem noch folgende Eigenschaften über die Variable a ausgegeben:

```
 DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 16
        FractionLength: 9
```

Die Wortbreite sowie die Anzahl der Nachkommastellen sind durch Matlab festgelegt, da keine weiteren Eigenschaften angegeben worden sind. Diese besagen, dass die Variable `a` eine Wortbreite von 16 Bit aufweist, Vorzeichen behaftet ist, und 9 Bit für die Darstellungen der Nachkommastellen verwendet werden. Daraus lässt sich schließen, dass 6 Bit für die Darstellung der ganzen Zahl übrig bleiben. Das bedeutet, dass der Wertebereich von 0 bis 63 definiert ist. Wenn in einer nächsten Operation der Wert von `a` um 32 erhöht werden soll, ist es nicht mehr darstellbar. Es wird dann folgendes Ergebnis ausgegeben:

```
  a(:)=a + 32

a =

   63.9980

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 16
        FractionLength: 9
```

Das ist für das weitere Vorgehen keine gute Möglichkeit. Wie dieses Problem umgangen werden kann, wird im nächsten Abschnitt erläutert.

B.1.2. Ein numerictype-Objekt definieren

Ein numerictype-Objekt enthält Informationen über den Aufbau eines Fixpoint-Objektes. Diese können nur vor dem Erstellen einer Variablen festgelegt werden, und sind danach nicht mehr veränderbar. Das numerictype-Objekt besteht ebenfalls aus einer Struktur, und es können dort folgende Eigenschaften festgelegt werden:

Eigenschaft	Beschreibung
Bias	dient zur Normalisierung des Exponenten bei Gleitkommazahlen. Ist nur für die Mantisse Exponent Darstellung erforderlich.
DataType	Vordefinierte Matlab Datentypen. Zur Auswahl stehen „boolean“, „double“, „Fixed“, „ScaledDouble“ und „single“. Die Datentypen „Fixed“ und „ScaledDouble“ sind keine Build-In Datentypen, bei denen weitere Eigenschaften angegeben werden können.
DataTypeMode	Datentyp und Skalierung. Dort stehen die Build-In Datentypen „boolean“, „double“ und „single“ zur Verfügung.
FixedExponent	Exponent der Mantisse Exponent Darstellung
FractionLength Scaling	Anzahl der Bits, die für die Darstellung der Nachkommastellen reserviert ist. Es gibt drei mögliche Optionen: BinaryPoint gibt an, dass die Variable über „wordlength“ und „fractionlength“ definiert ist, SlopeBias gibt an, dass die Variable über Slope und Bias festgelegt ist.
Signed / Signedness	Angabe, die ob Variable ein Vorzeichen aufweisen soll. Angabe bei Signed: true oder false. Angabe bei Signedness: Signed, Unsigned, auto.
Slope	Matisse der Mantisse-Exponent Darstellung.
WordLength	Wortbreite der Variablen in Bit.

Tabelle B.1.: Beschreibung eines numerictype-Objekts

Um zum Beispiel ein numerictype-Objekt mit dem Namen b zu entwerfen, in dem eine Wortbreite von 16 Bit ohne Vorzeichen definiert ist und 5 Bit für die Nachkommastellen zur Verfügung stehen sollen, kann wie folgt geschrieben werden:

```
b = numerictype( 'Singed', false, 'Scaling', 'BinaryPoint',...
'wordlength',16,'fractionLength', 5)
```

Dieses Beispiel kann auch in der Exponentialform dargestellt werden. Dann kann das numerictype Objekt so aussehen:

```
b = numerictype('DataType','Fixed', 'Signed', false, 'Scaling',...
    'SlopeBias', 'wordlength',16,'FixedExponent', -5)
```

Um eine Variable a mit einem Wert von 32.4375 von Typ b zu definieren, kann dies wie folgt angegeben werden:

```
a = fi(32.453,b);
```

Für den ersten Fall wird folgendes Ergebnis ausgegeben:

```
a =

   32.4375

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 16
        FractionLength: 5
```

Die Abweichung kommt zustande da 0.453 nicht mit 5 Bit dargestellt werden kann. Deshalb wir das Ergebnis abgerundet.

Wird die Variable in Exponentialform dargestellt, wird folgendes Ergebnis ausgegeben:

```
a =

   32.4375

          DataTypeMode: Fixed-point: slope and bias scaling
            Signedness: Unsigned
            WordLength: 16
                 Slope: 2^-5
                  Bias: 0
```

B.1.3. Aufbau eines fimath-Objekts

Ein fimath-Objekt definiert die Eigenschaften arithmetischer Operationen, wie die Addition, Subtraktion, Division und Multiplikation. Sie müssen ebenfalls bei der Definition einer neuen Fixpoint-Variablen angegeben werden. Die Einstellmöglichkeiten können in der Matlab Hilfe nachgelesen werden.

Die arithmetischen Eigenschaften müssen bei der Definition einer neuen Variablen mit den numerictype-Eigenschaften angegeben werden. Das kann zum Beispiel für ein fimath-Objekt mit dem Namen `p` und dem numerictype-Objekt `b` wie folgt aussehen:

```
x = fi(32,b,p);
```

B.2. Die Schiebe Operationen

In der Fixpoint-Toolbox sind für das Scheiben nach rechts die Funktionen bitsra, bitsrl sowie für das Schieben nach links die Funktionen bitsla, bitsll vorgesehen. Diese sind nicht verwendet worden, weil sie nur als Matlab-Code vorliegen und die Simulation sehr viel mehr Zeit in Anspruch nehmen würde.
Stattdessen ist die Funktion bitshift() verwendet worden, da diese im kompilierter Form vorliegt. Allerdings muss beachtet werden, dass ein positiver Schiebefaktor einem Schieben nach links und ein negativer Schiebefaktor einem Schieben nach rechts entspricht.
Durch das Verwenden dieser Funktion lässt sich das Ausführen des Simulationsprogramms beschleunigen.

C. Beschreibung der Datenstruktur

C.1. Beschreibung der abgespeicherten Simulationsergebnisse

Variablenname	Beschreibung
DFT_imag	Sinus-Wertetabelle mit Auflösung von „NLUT“ Bit.
DFT_real	Kosinus-Wertetabelle mit Auflösung von „NLUT“ Bit.
NADC	Auflösung des simulierten Analog-Digital-Umsetzers für die aktuelle Simulation.
NLUT	Auflösung der Sinus- und Kosinus Wertetabellen für die aktuelle Simulation.
NO	Anzahl der berücksichtigten Koeffizienten für die Berechnungsmethode aus der Arbeit von Herrn Jegenhorst.
N_harm	Anzahl der Harmonischen, mit denen das Signal approximiert worden ist.
P1	Errechnete Leistung der ersten Harmonischen mit der HDI-Methode.
Pges	Errechnete Leistung des Signals mit der HDI-Methode.
S	Anzahl der Samples pro Periode.
THD5_mat	In Gleitkomma-Arithmetik bestimmtes Ergebnis mit der HD5-Methode
THD_calc	In Fixpoint-Arithmetik bestimmtes Ergebnis des HD5-Methode
THD_lut	Aus der Arbeit von Herrn Jegenhorst übernommene Klirrfaktor-Ergebnisse die mit dem Mikrocontroller ermittelt worden sind.
THD_mat	In Gleitkomma-Arithmetik bestimmter Klirrfaktor unter Berücksichtigung aller im Signal enthalten Harmonischen.
THD_new	Mit Fixpoint-Arithmetik bestimmtes Ergebnis mit der HDI-Methode.
THD_new_mat	In Gleitkomma-Arithmetik bestimmtes Ergebnis mit der HDI-Methode
distance	Vektor in dem für alle Messungen die Entfernungen zwischen Sensor und Encoderrad abgelegt sind.

Variablenname	Beschreibung
fixp	Datenstrukur, in der die Fixpoint-Datentypen und Einstellungen abgelegt sind. Eine Beschreibung ist in Kapitel 4 zu finden.
im_calc	Feld, in dem für alle Messpunkte der Imaginärteil aller Harmonischer für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte sind in Festkomma-Arithmetik bestimmt worden.
im_mat	Feld, in dem für alle Messpunkte der Imaginärteil aller Harmonischen für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte sind in Gleitkomma-Arithmetik bestimmt worden.
mag_calc	Feld, in dem für alle Messpunkte die Beträge aller Harmonischen für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte sind in Festkomma-Arithmetik bestimmt worden.
mag_mat	Feld, in dem für alle Messpunkte die Beträge aller Harmonischer für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte in Gleitkomma-Arithmetik bestimmt worden.
parameters	Aus den Messdaten aus [7] entnommene Datenstruktur, in der Messbereich, die Abtastfrequenz des Oszilloskops, die Umschaltstufen des internen Vorverstärkers und die Entfernung zwischen zwei Messpunkten angegeben ist.
re_calc	Feld, in dem für alle Messpunkte der Realteil aller Harmonischen für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte sind in Festkomma-Arithmetik berechnet worden.
re_mat	Feld, in dem für alle Messpunkte der Realteil aller Harmonischen für das Verfahren aus der Arbeit von Herrn Jegenhorst abgelegt ist. Diese Werte sind ist in Gleitkomma-Arithmetik berechnet worden.
s_fix	Feld in dem für alle Messpunkte die Samples für eine Periode gespeichert sind.

Tabelle C.1.: Beschreibung der bei jeder Simulation abgespeicherten Ergebnisse

D. Simulationsergebnisse

D.1. Analyse des Klirrfaktors über die Distanz für verschiedene Parameter

D.1.1. Verwendung der ersten Berechnungsmöglichkeit

In diesem Versuch soll die Simulation unter Verwendung von verschieden Wortbreiten für den Analog-Digital-Umsetzer sowie der Sinus- und Kosinus-Wertetabellen durchgeführt werden.

6 Bit Look-up Tabelle

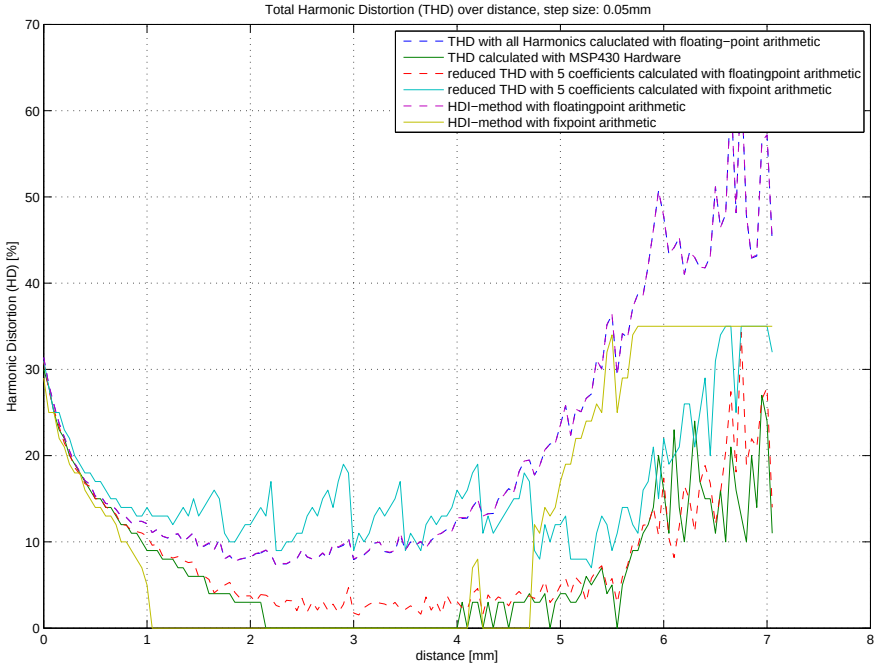

Abbildung D.1.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

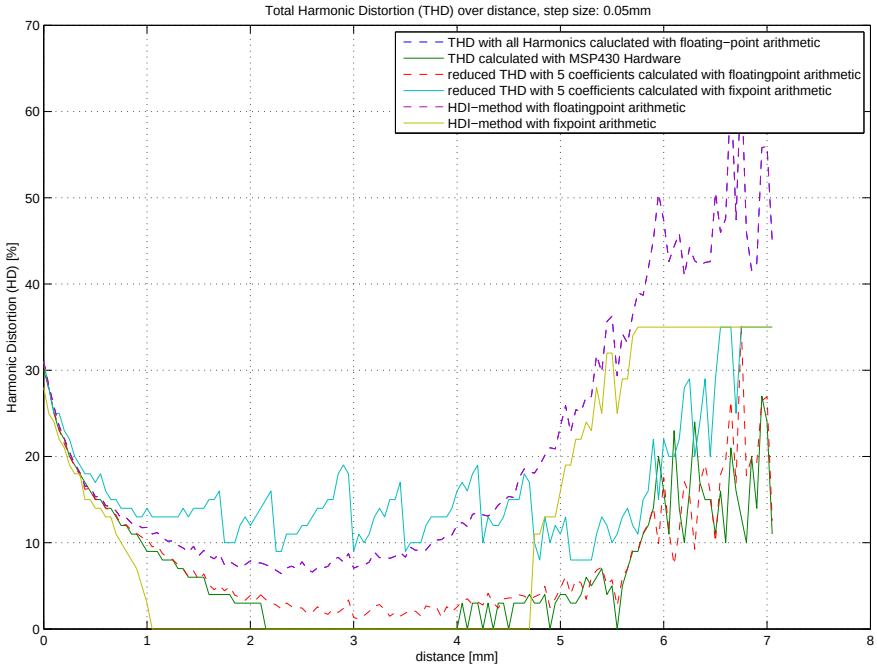

Abbildung D.2.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

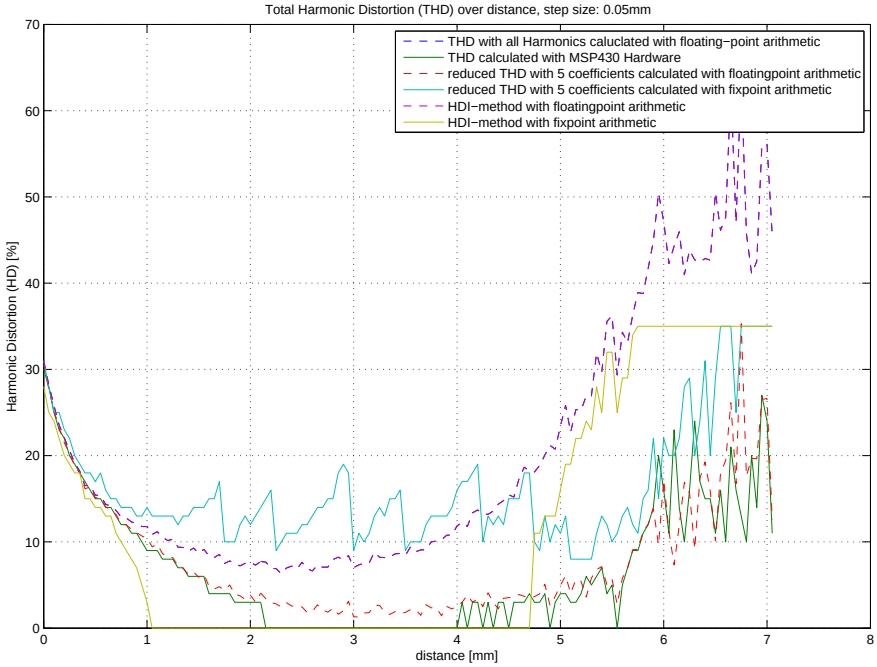

Abbildung D.3.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

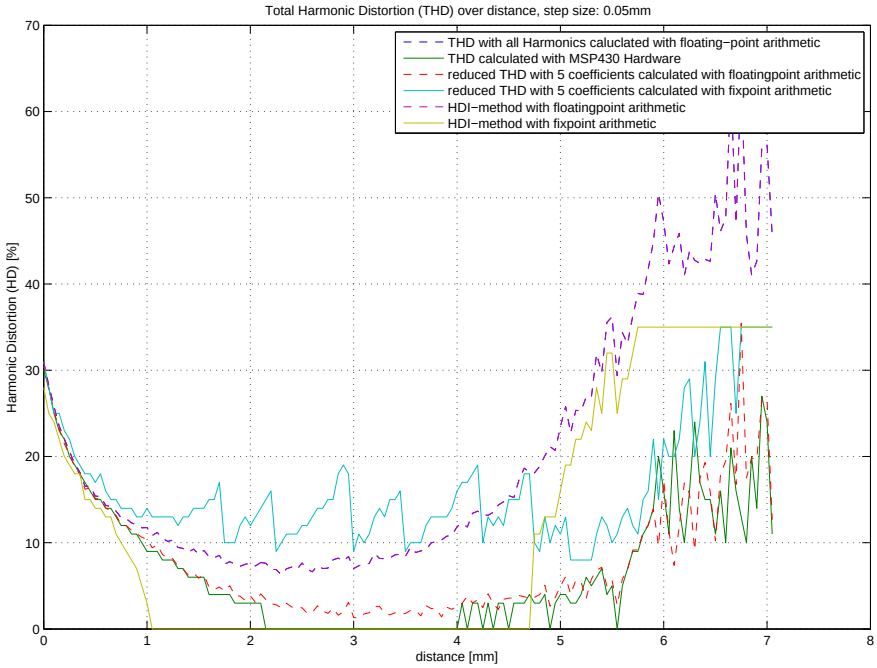

Abbildung D.4.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

8 Bit Look-up Tabelle

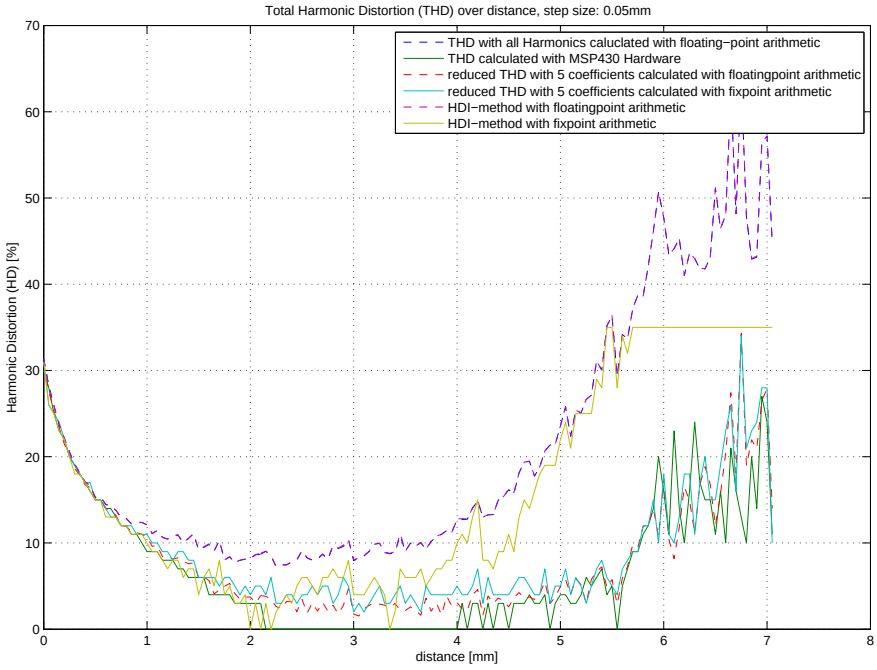

Abbildung D.5.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

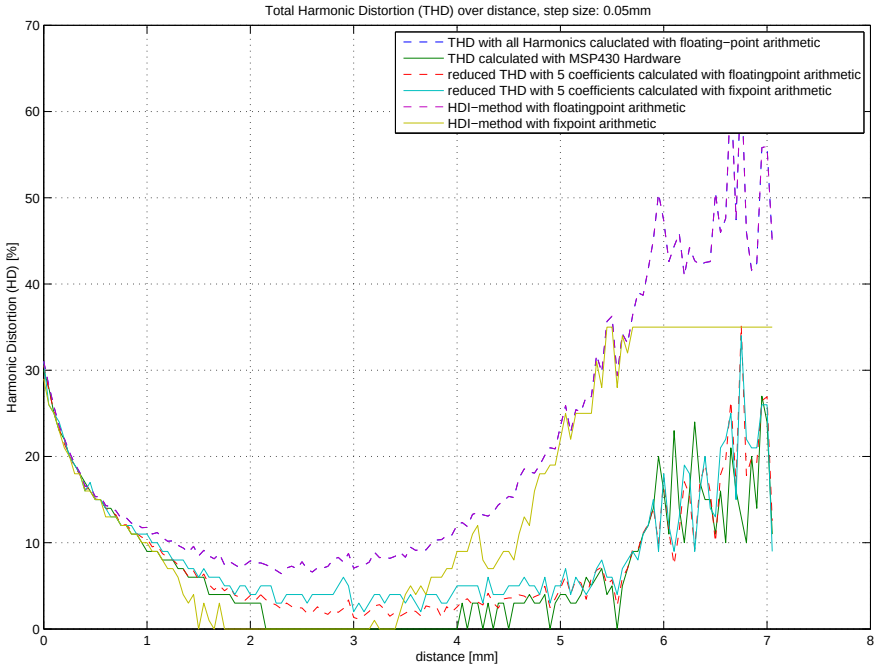

Abbildung D.6.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

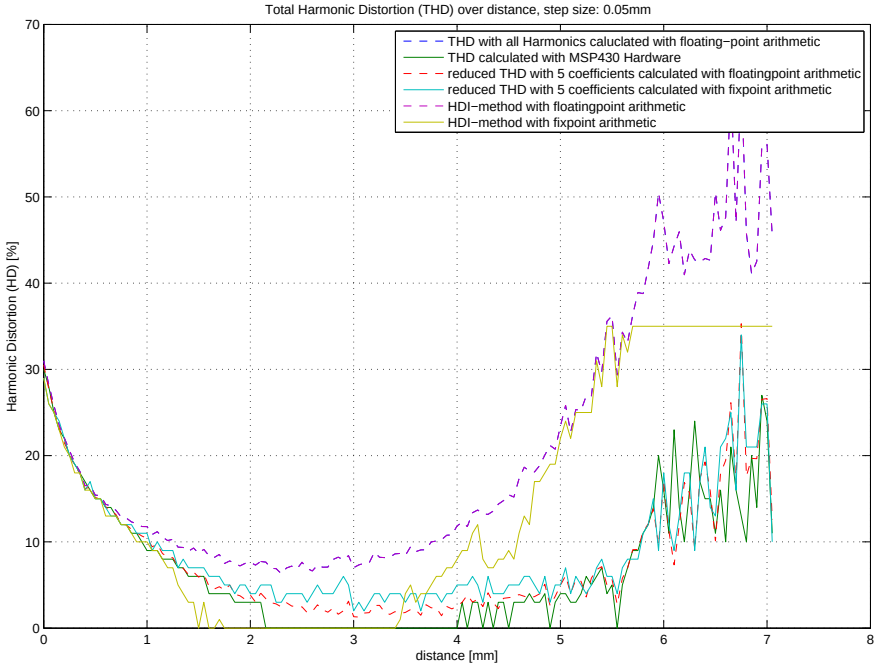

Abbildung D.7.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

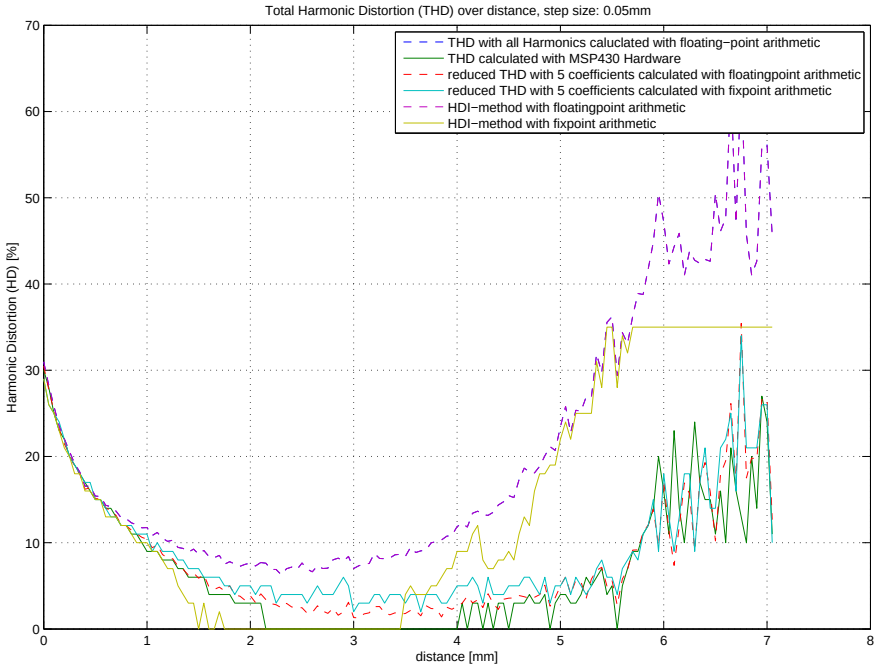

Abbildung D.8.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

9 Bit Look-up Tabelle

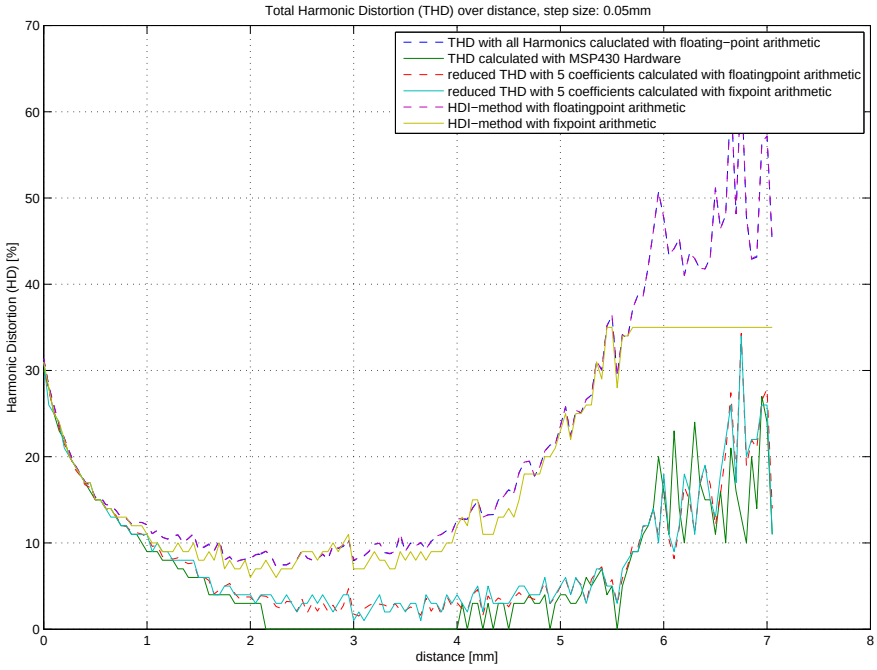

Abbildung D.9.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

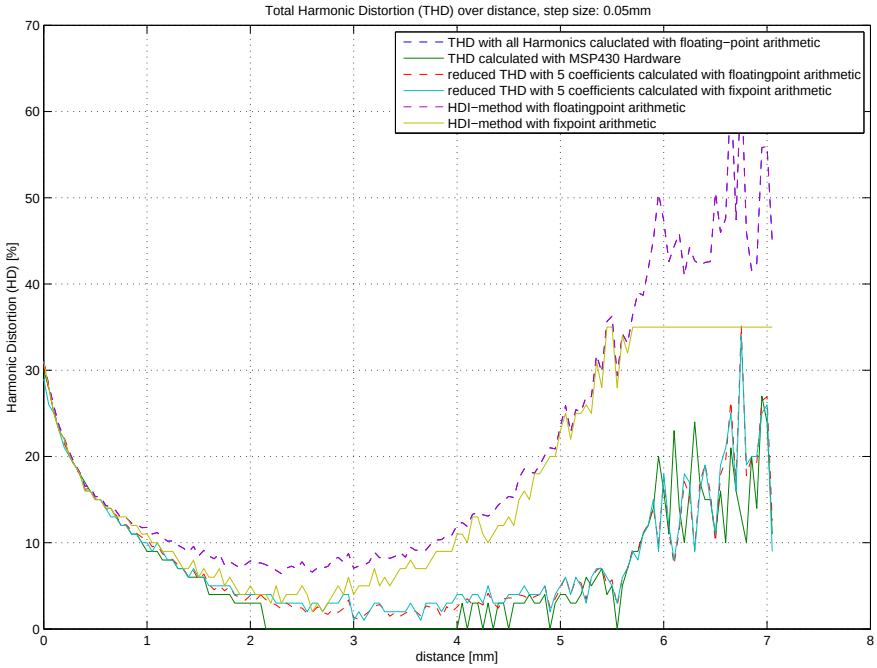

Abbildung D.10.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

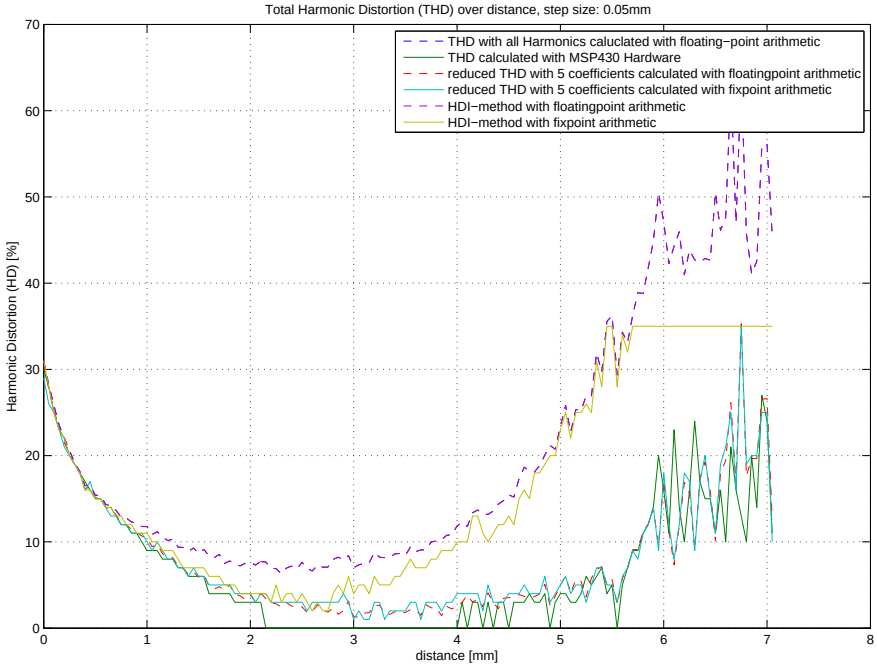

Abbildung D.11.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

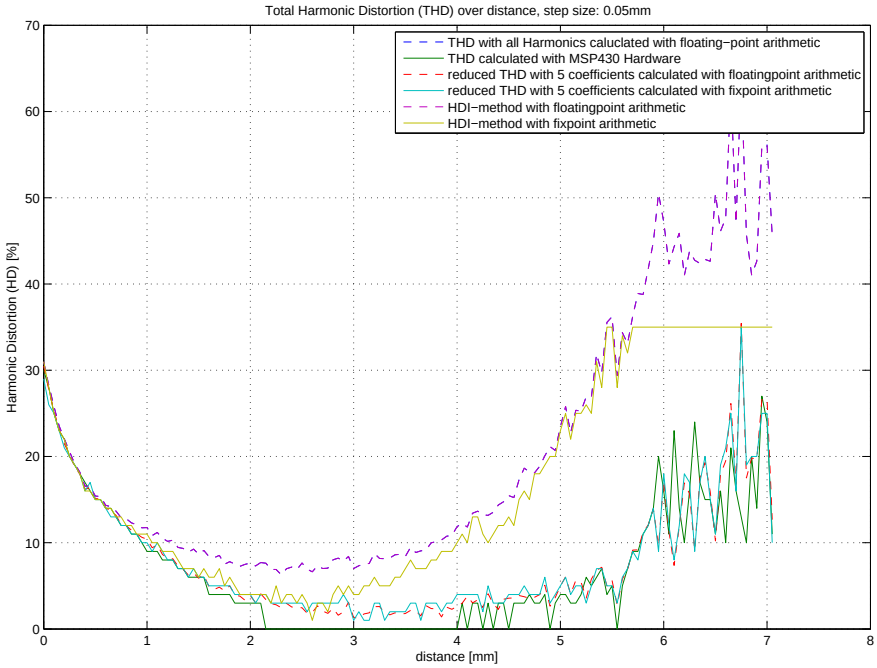

Abbildung D.12.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

10 Bit Look-up Tabelle

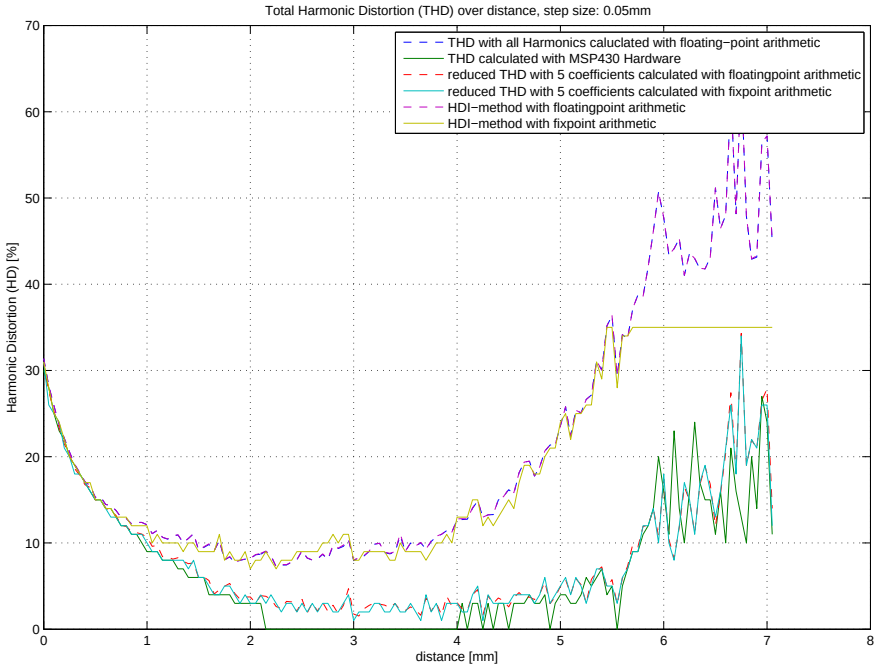

Abbildung D.13.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

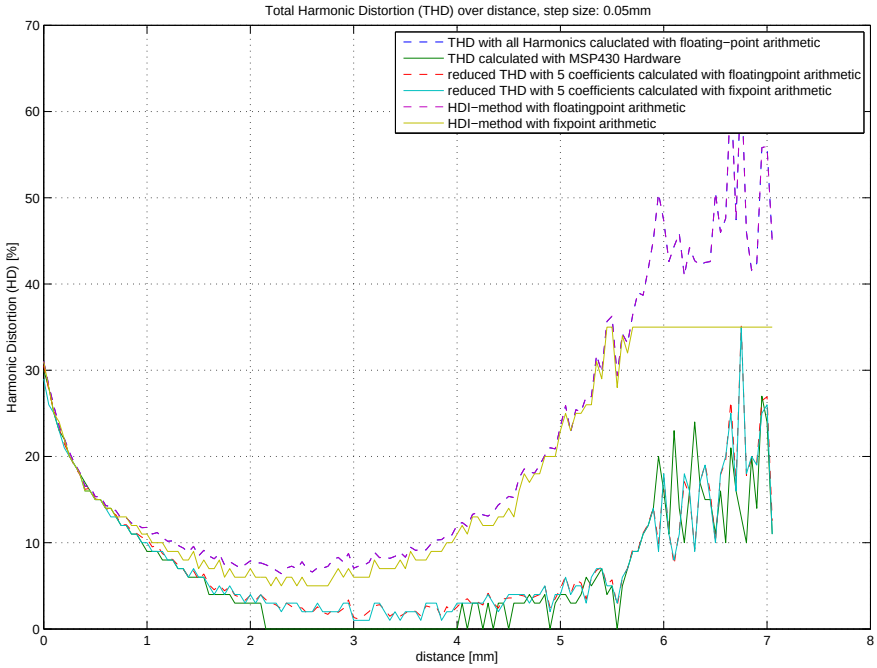

Abbildung D.14.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

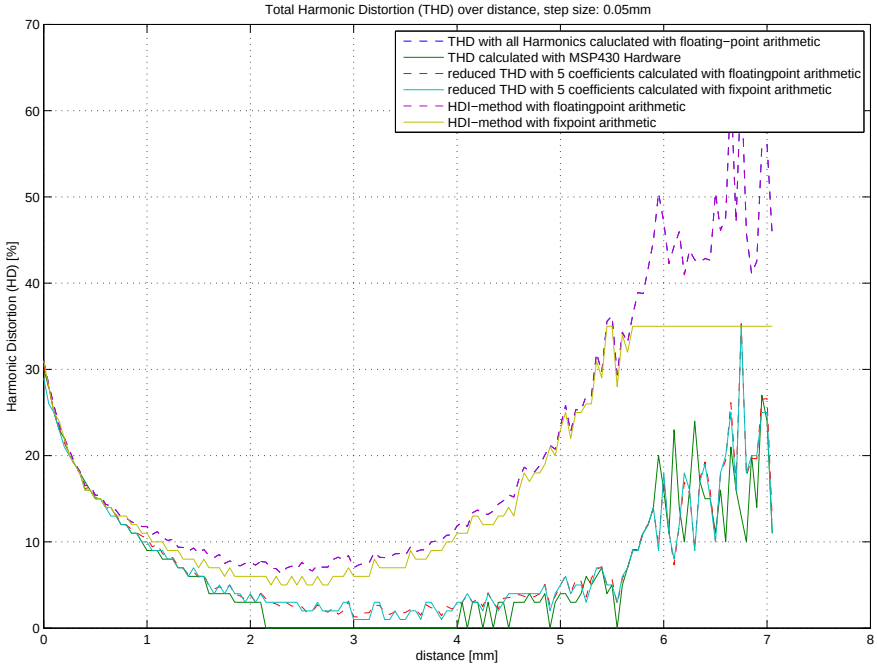

Abbildung D.15.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC

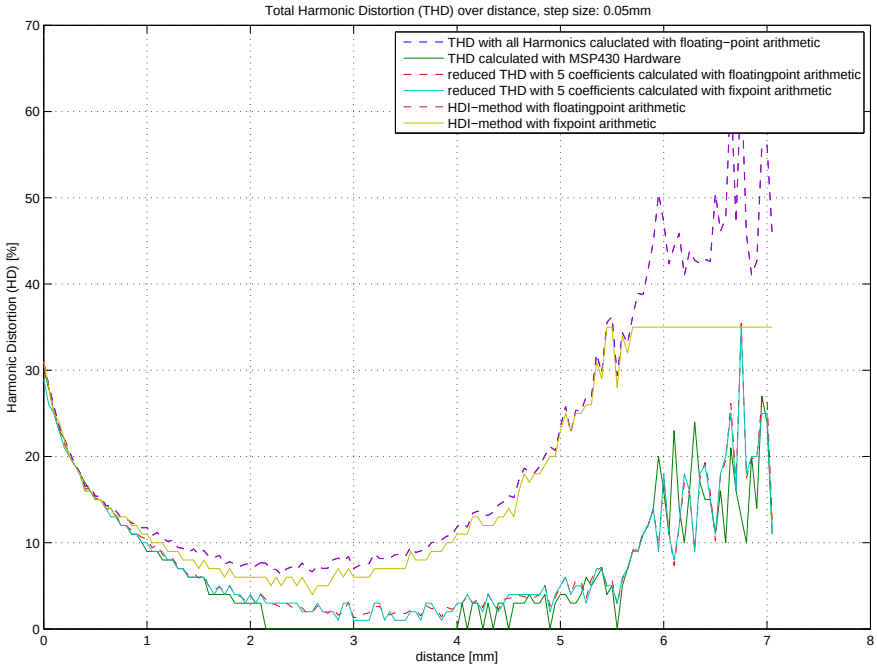

Abbildung D.16.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

D.1.2. Verwendung der zweiten Berechnungsmöglichkeit

In diesem Versuch soll die Simulation unter Verwendung von verschieden Wortbreiten für den Analog-Digital-Umsetzer sowie der Sinus- und Kosinus-Wertetabellen durchgeführt werden.

6 Bit Look-up Tabelle

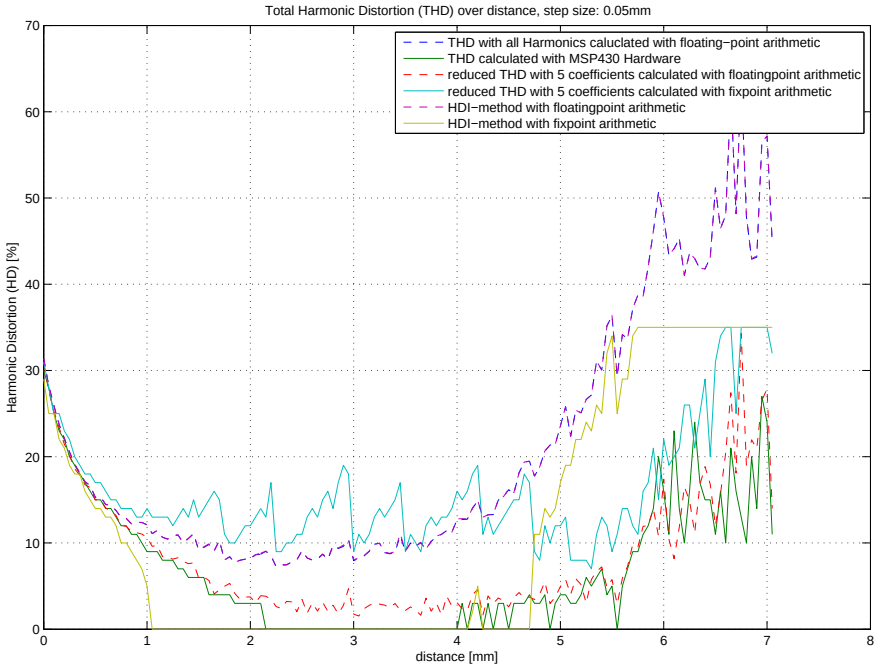

Abbildung D.17.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

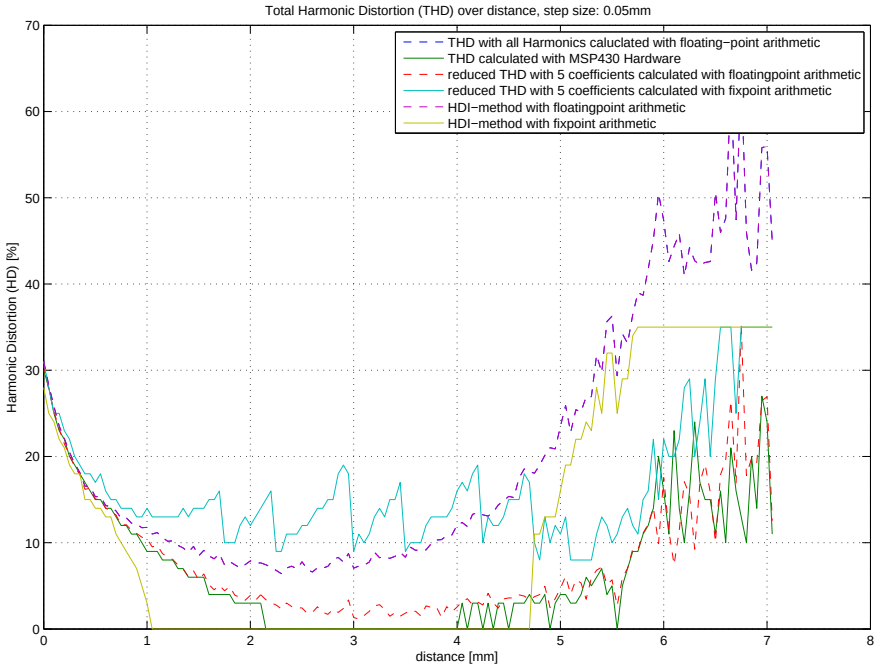

Abbildung D.18.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

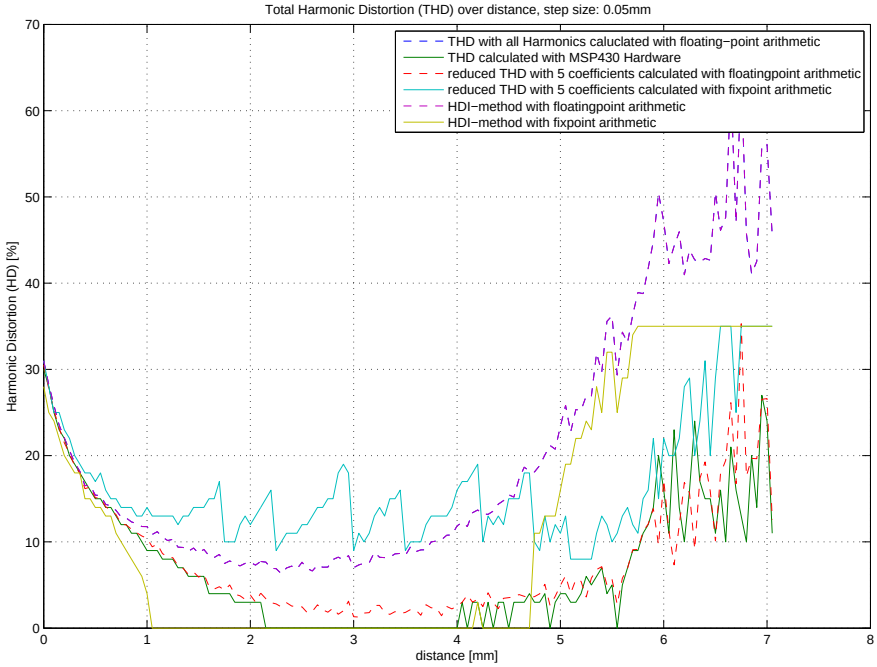

Abbildung D.19.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

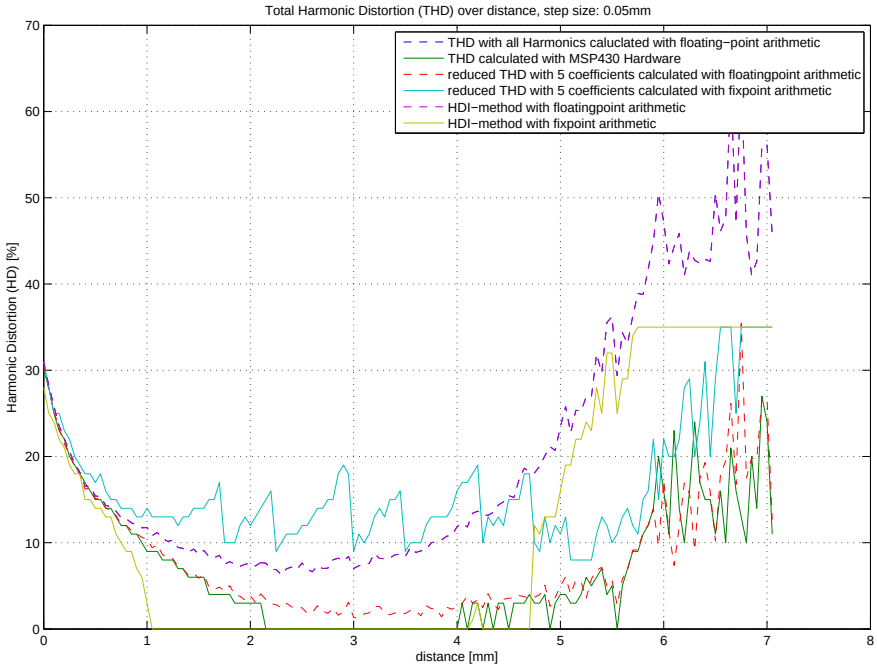

Abbildung D.20.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

8 Bit Look-up Tabelle

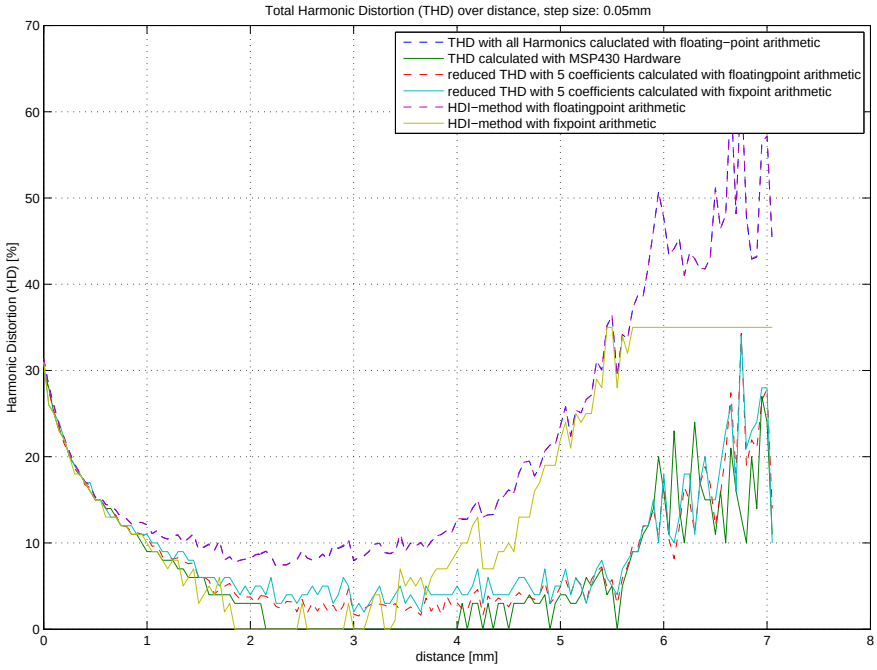

Abbildung D.21.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

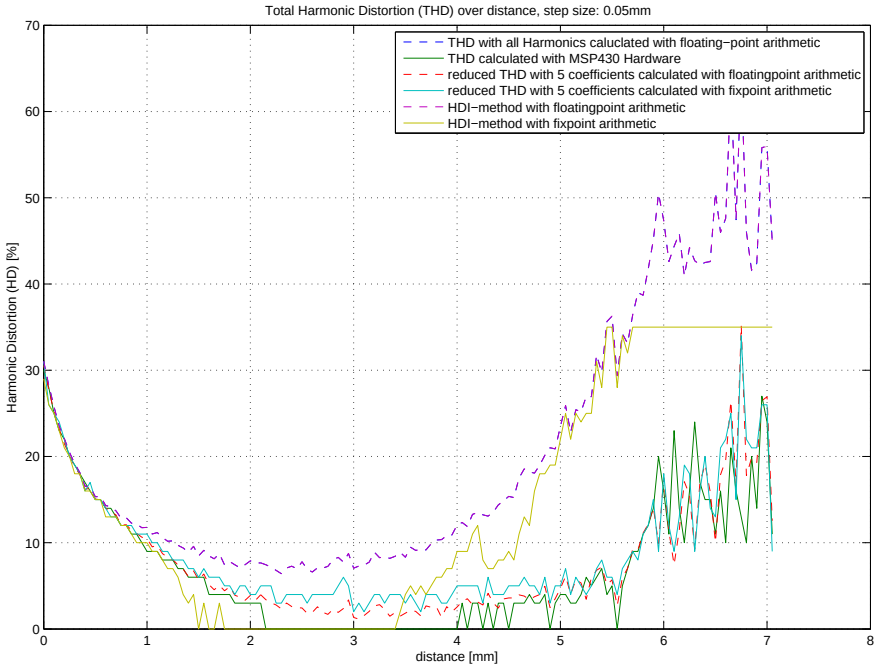

Abbildung D.22.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

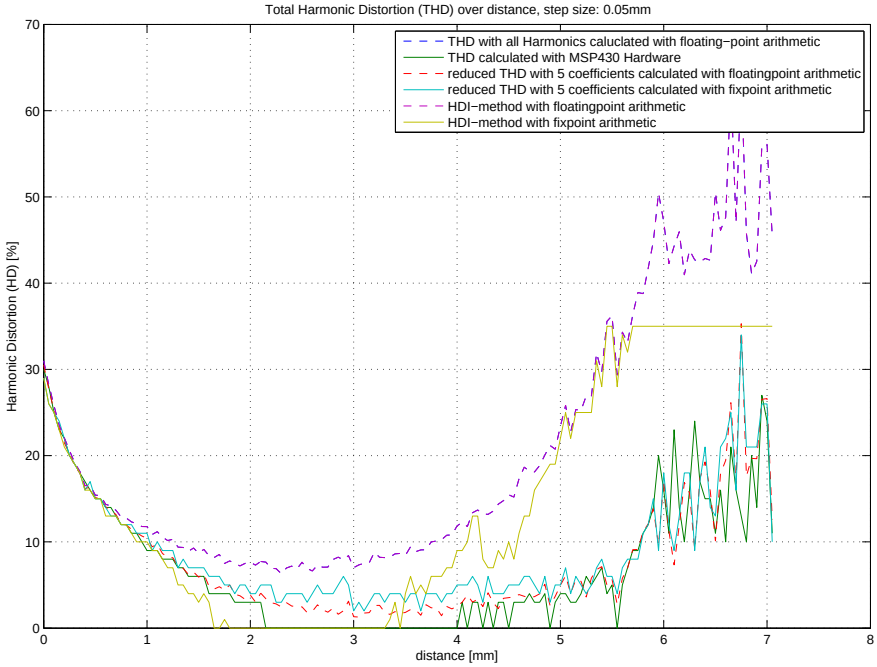

Abbildung D.23.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

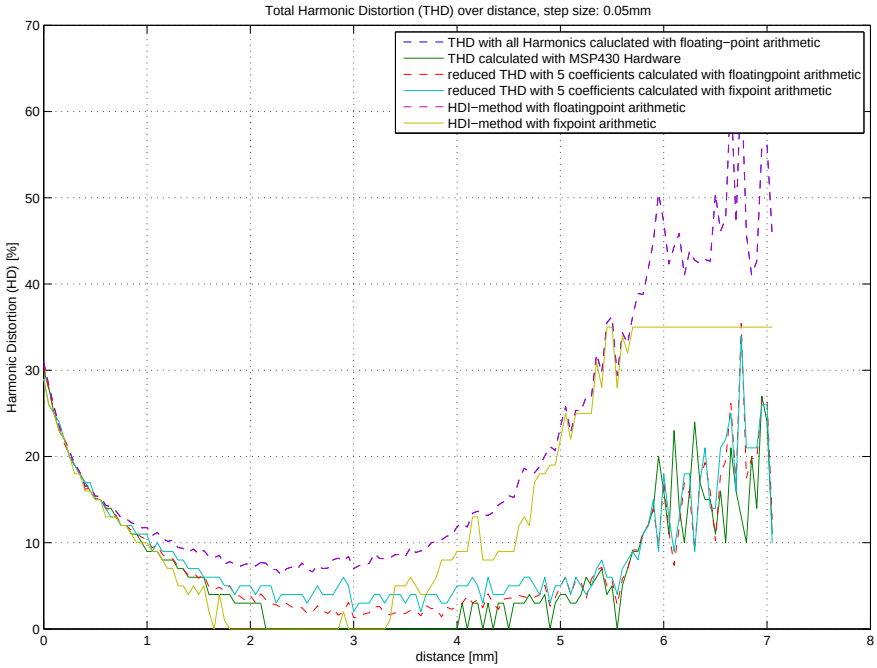

Abbildung D.24.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

9 Bit Look-up Tabelle

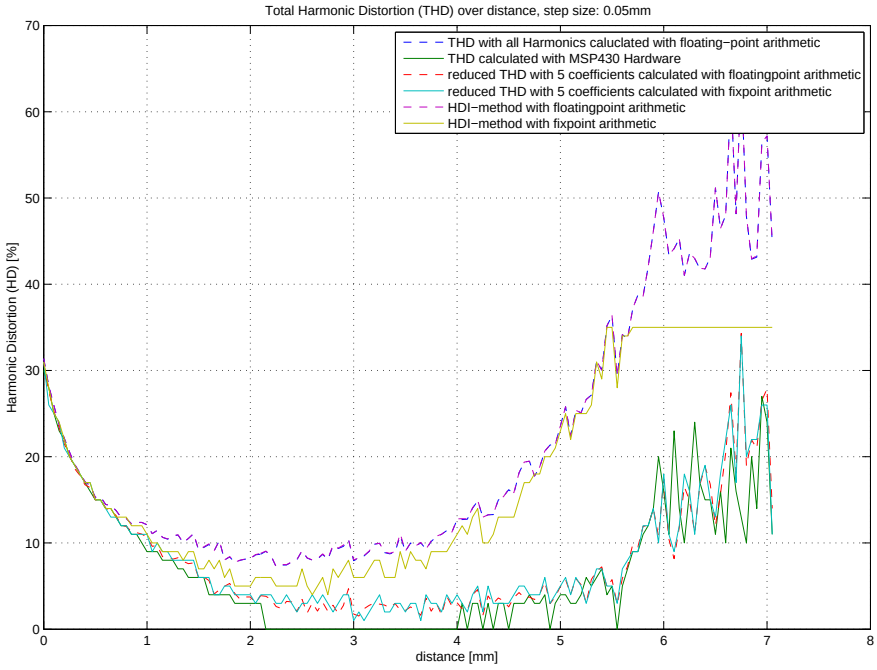

Abbildung D.25.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

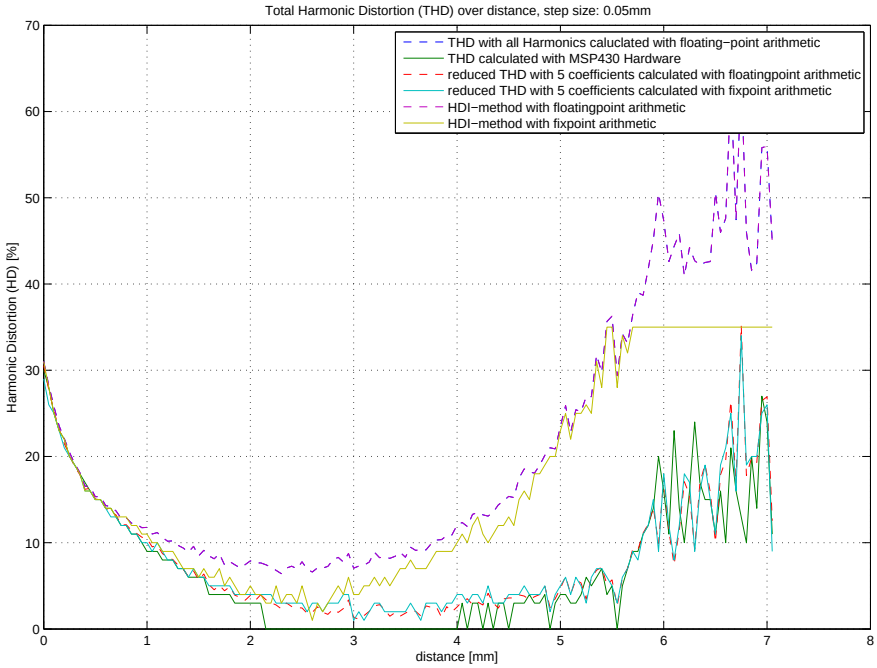

Abbildung D.26.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

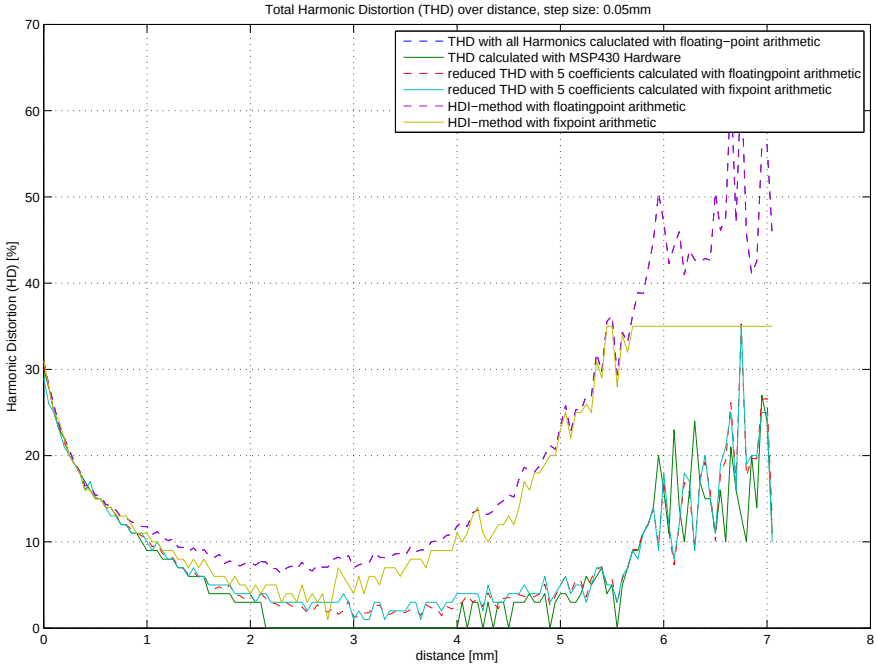

Abbildung D.27.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

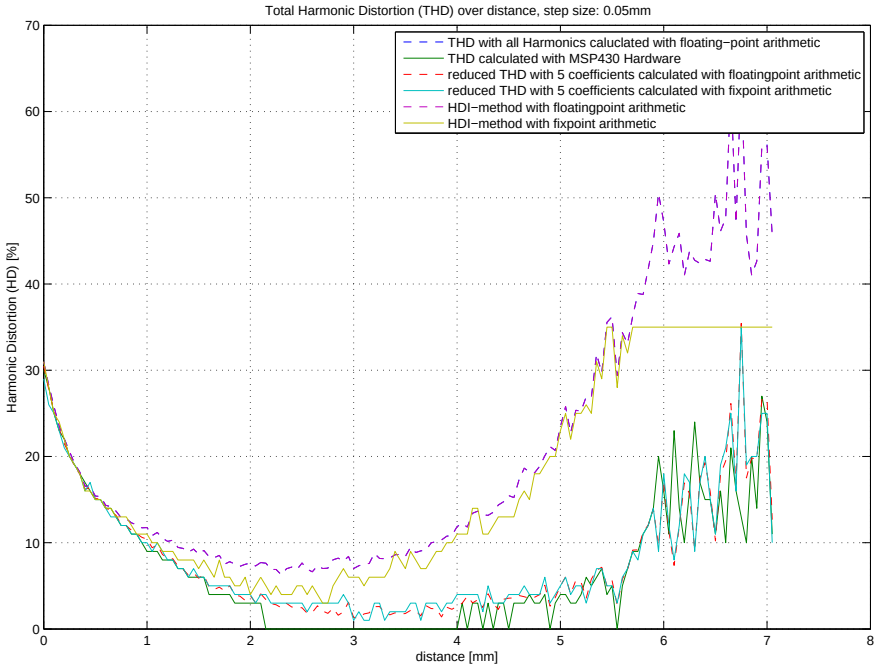

Abbildung D.28.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

10 Bit Look-up Tabelle

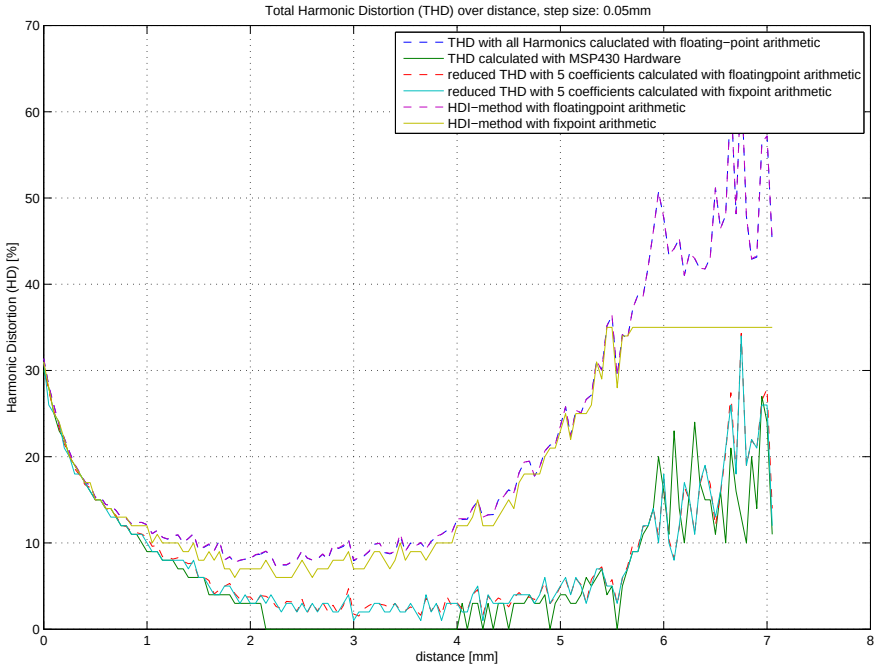

Abbildung D.29.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 6 Bit ADC [8]

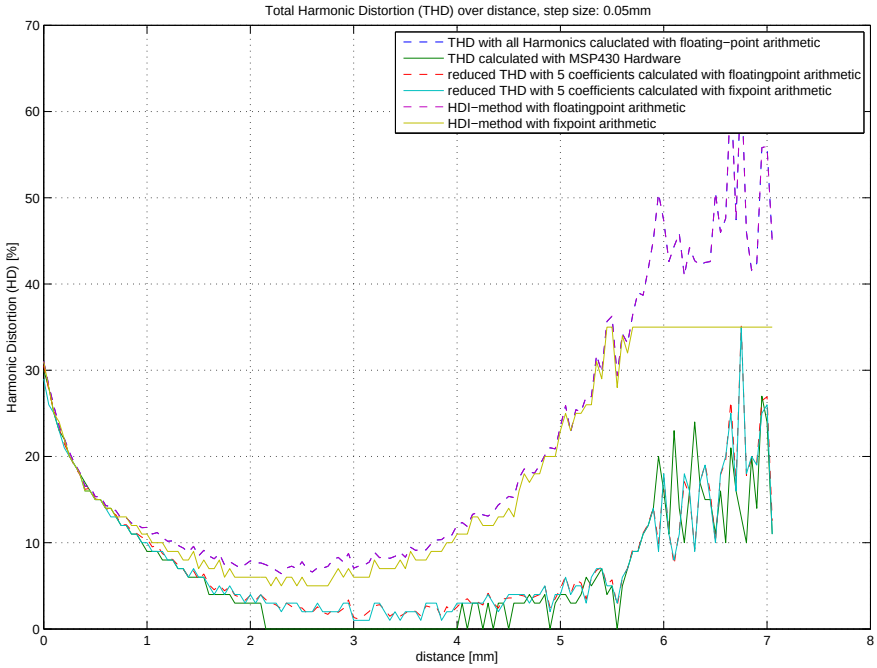

Abbildung D.30.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 8 Bit ADC [8]

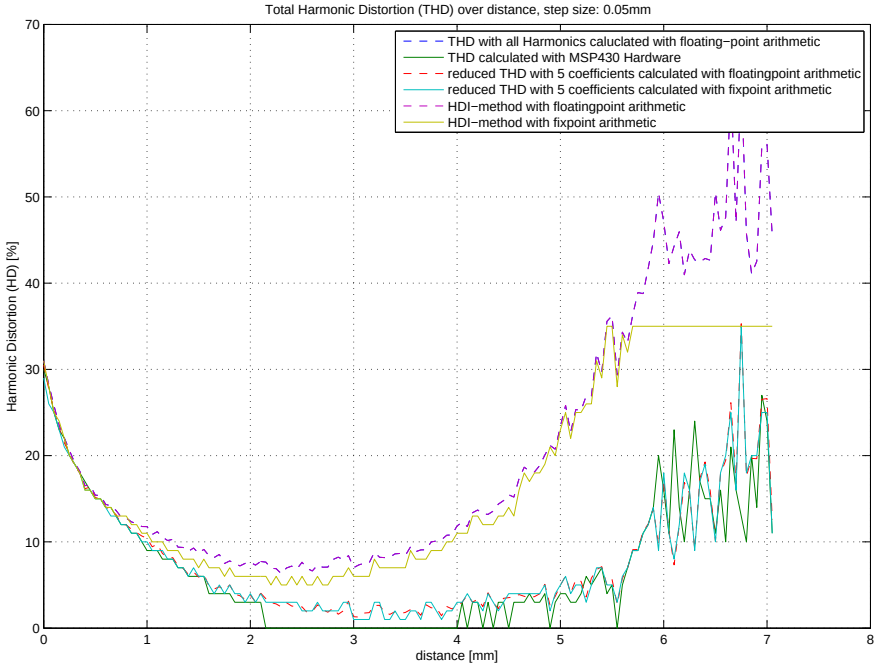

Abbildung D.31.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 10 Bit ADC [8]

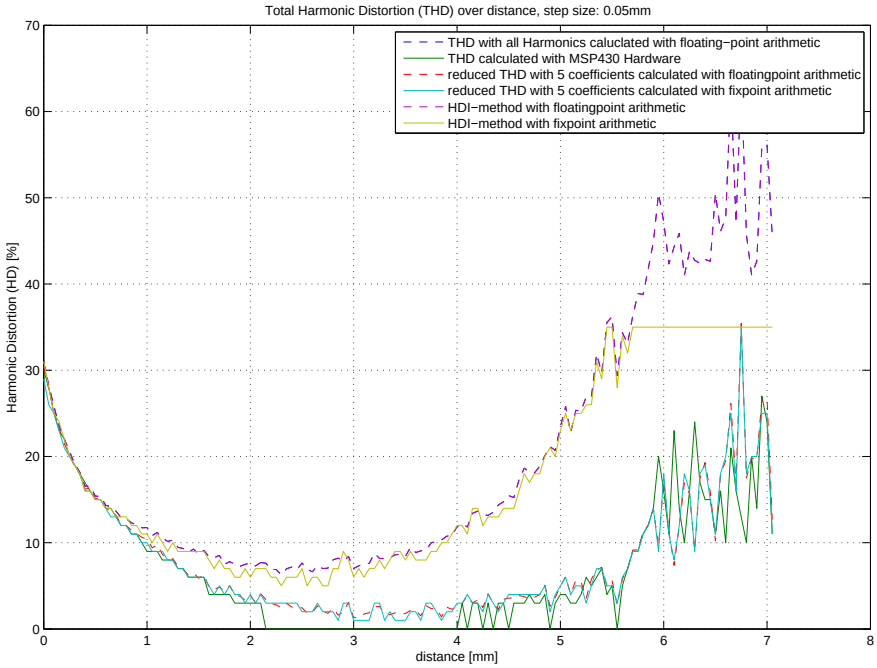

Abbildung D.32.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad bei Verwendung eines 12 Bit ADC [8]

D.2. Erhöhen der Abtastfrequenz

D.2.1. Verwendung der 1. Berechnungsmöglichkeit

In diesem Abschnitt befinden sich die Ergebnisse des in Abschnitt 9.5 beschriebenen Versuchs.

Verwendung eines 3 Bit Analog-Digital-Umsetzers

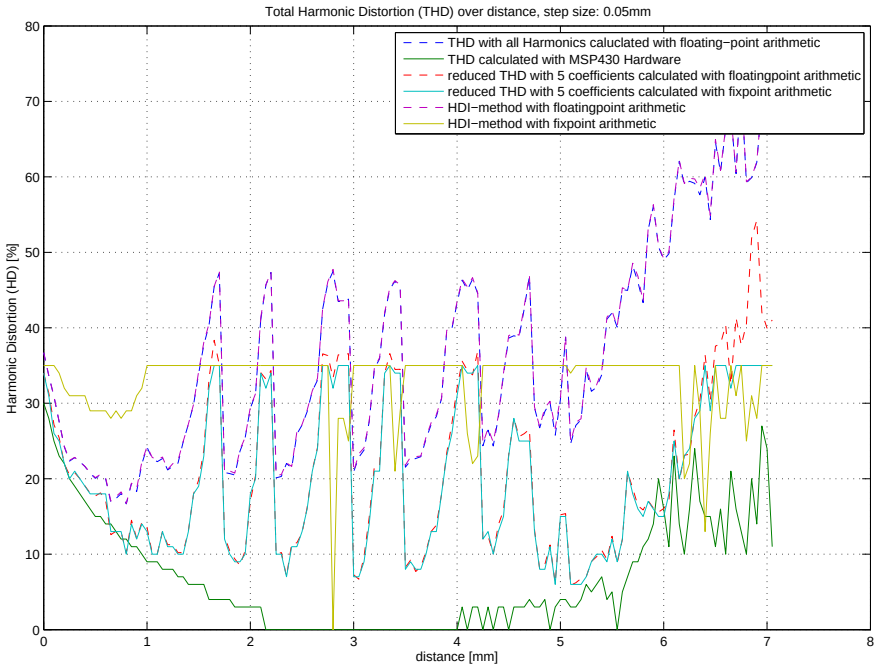

Abbildung D.33.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

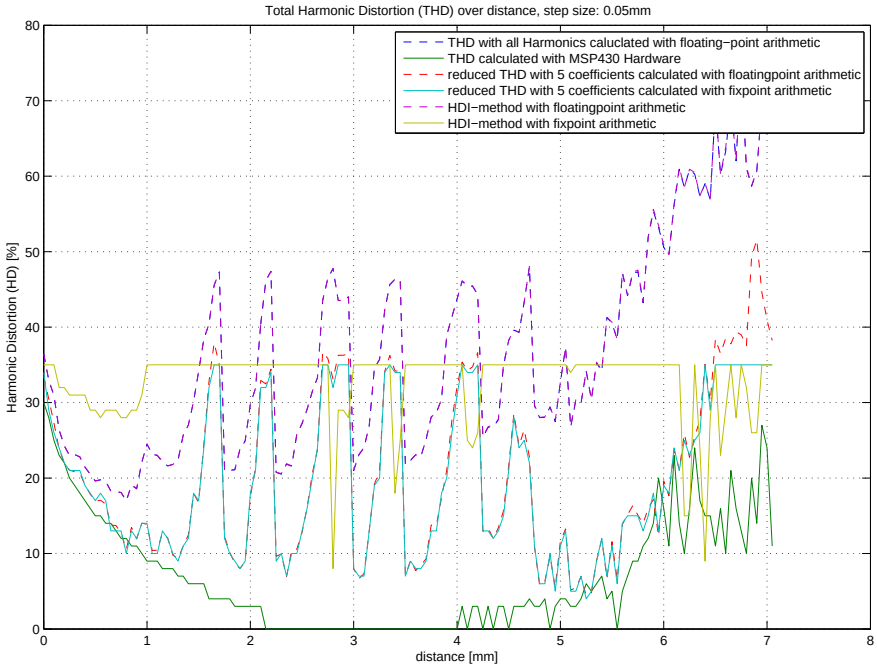

Abbildung D.34.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

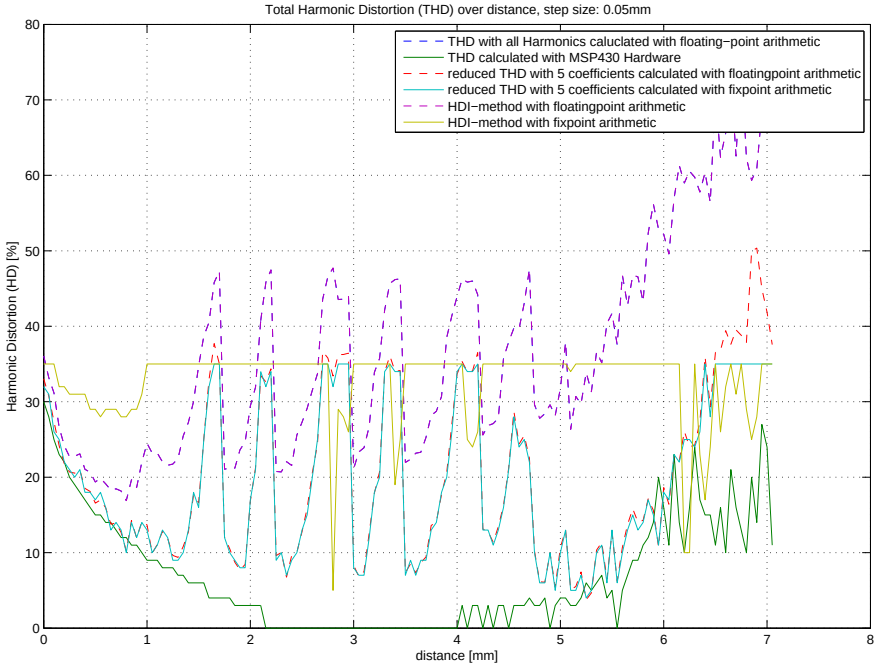

Abbildung D.35.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

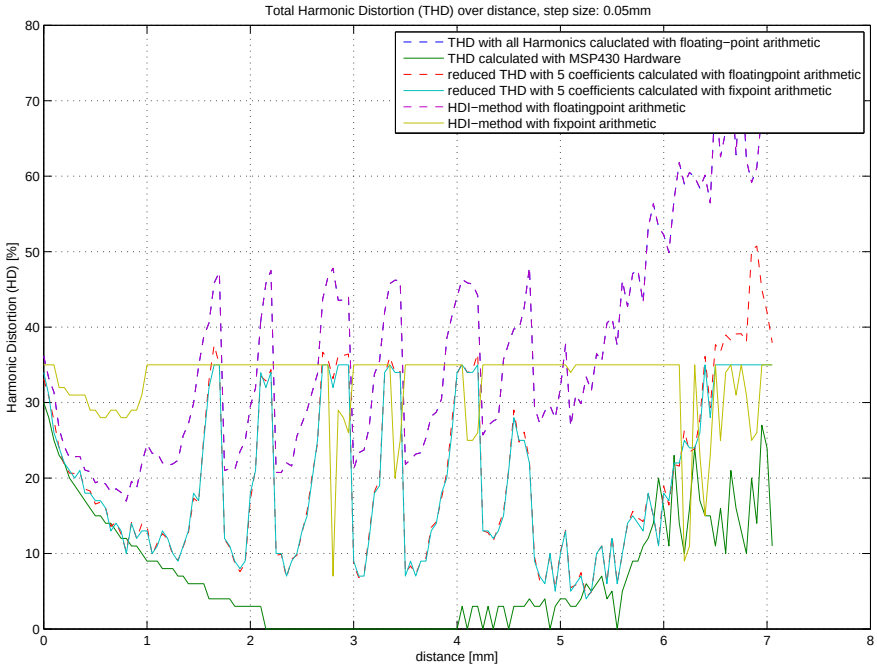

Abbildung D.36.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 4 Bit Analog-Digital-Umsetzers

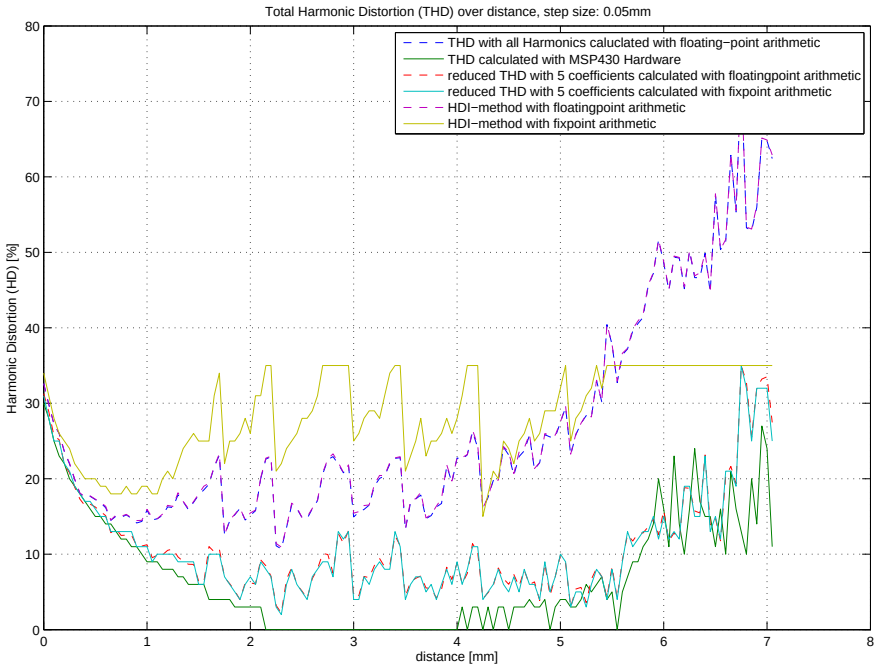

Abbildung D.37.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

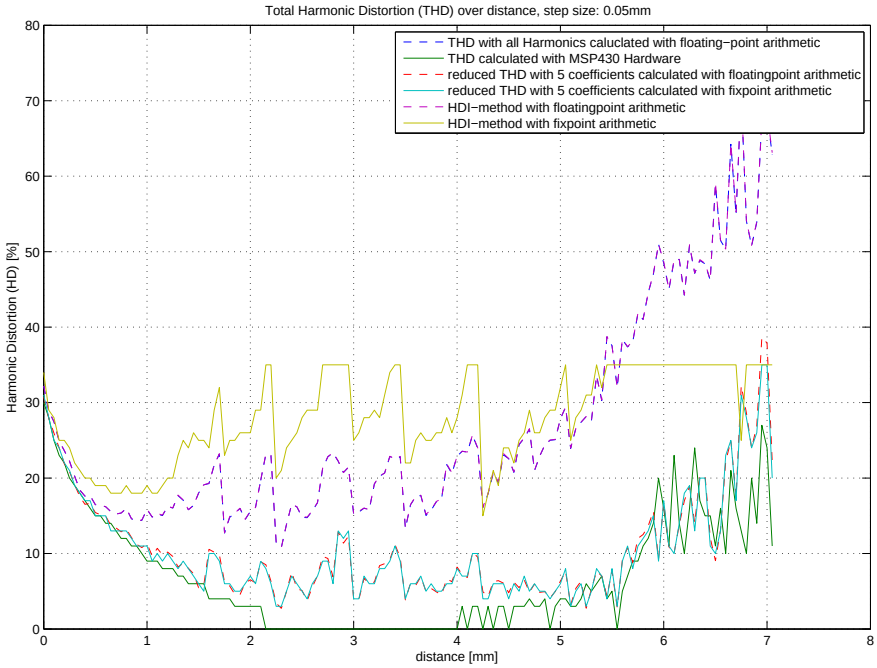

Abbildung D.38.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

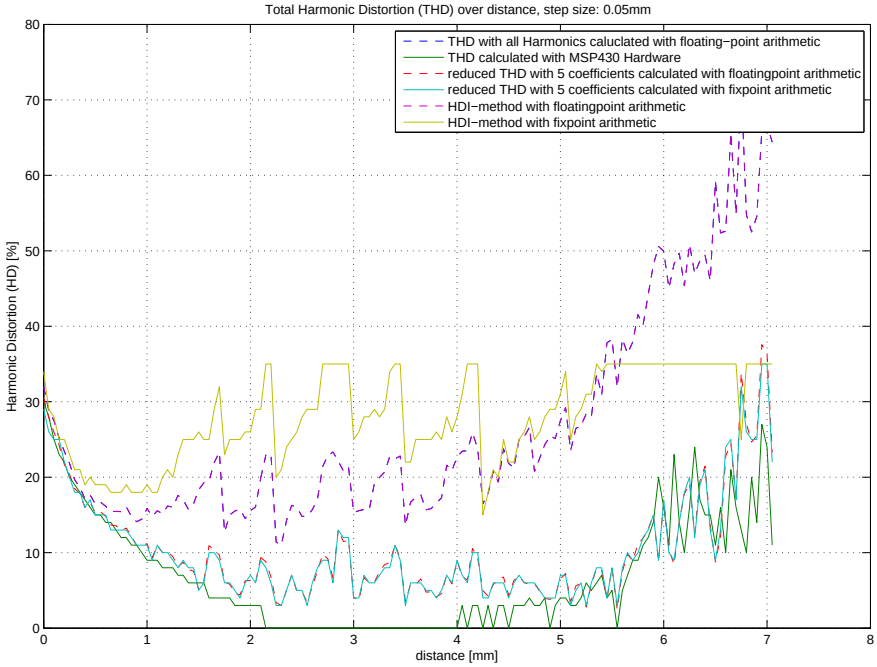

Abbildung D.39.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

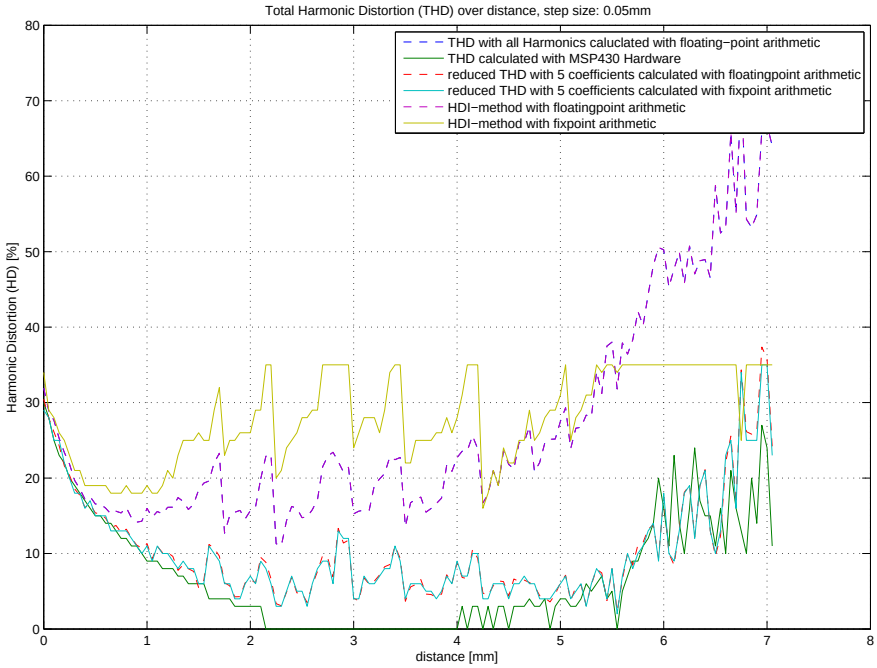

Abbildung D.40.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 5 Bit Analog-Digital-Umsetzers

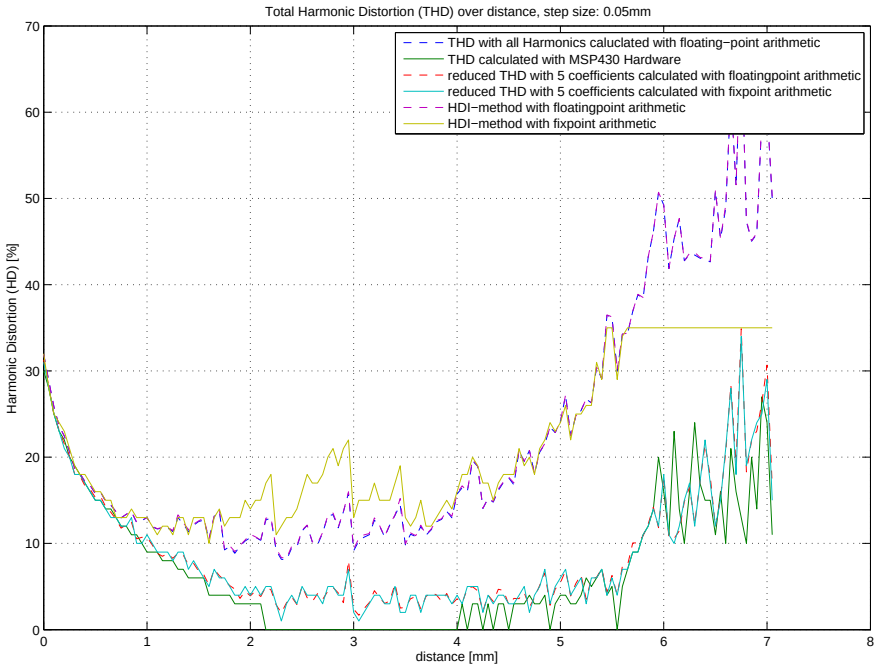

Abbildung D.41.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

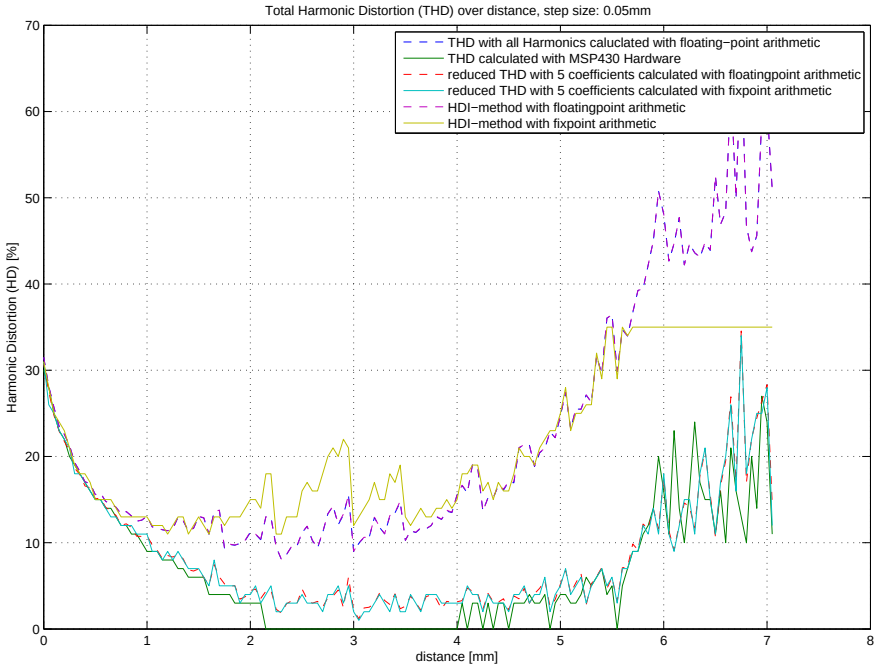

Abbildung D.42.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

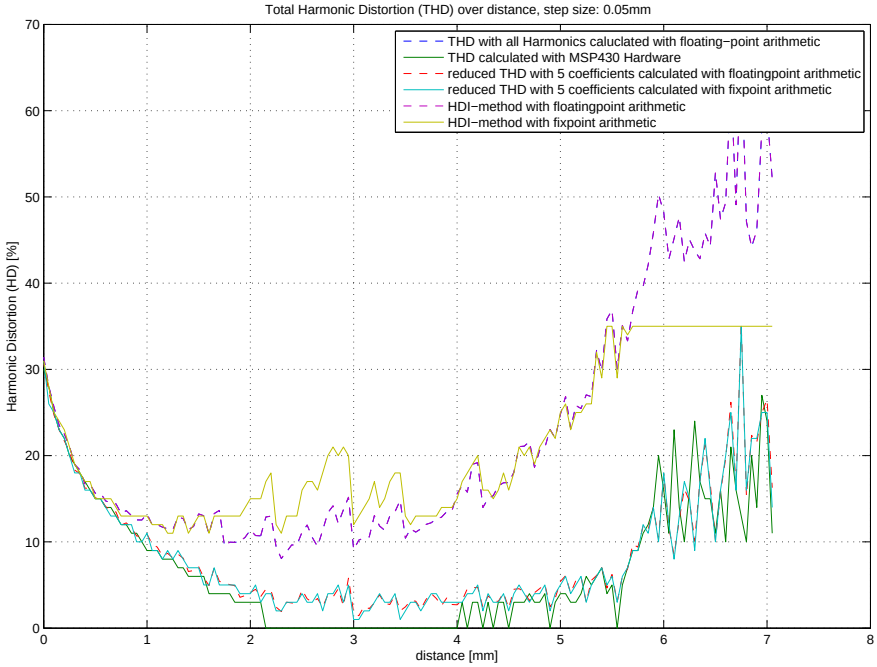

Abbildung D.43.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

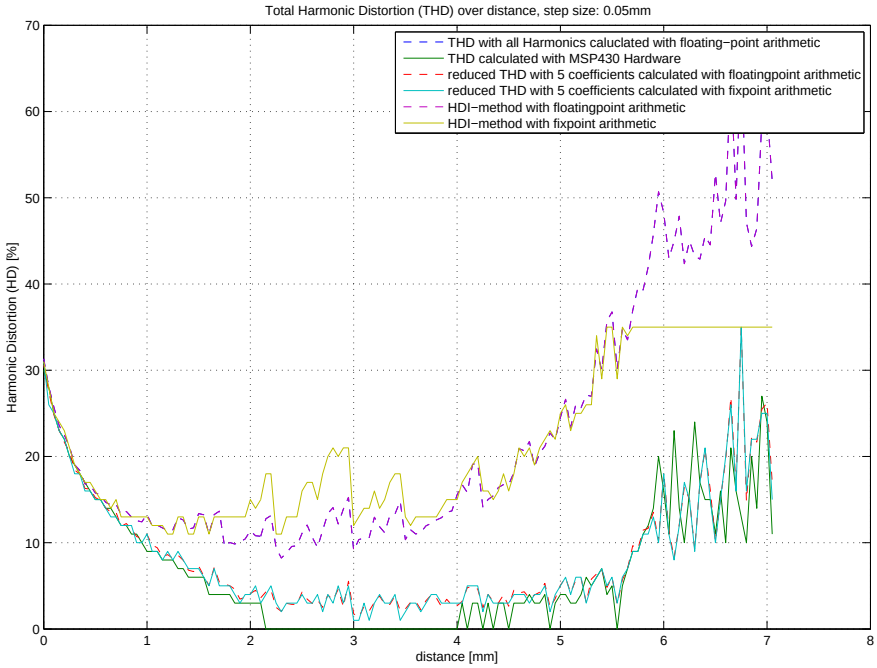

Abbildung D.44.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 6 Bit Analog-Digital-Umsetzers

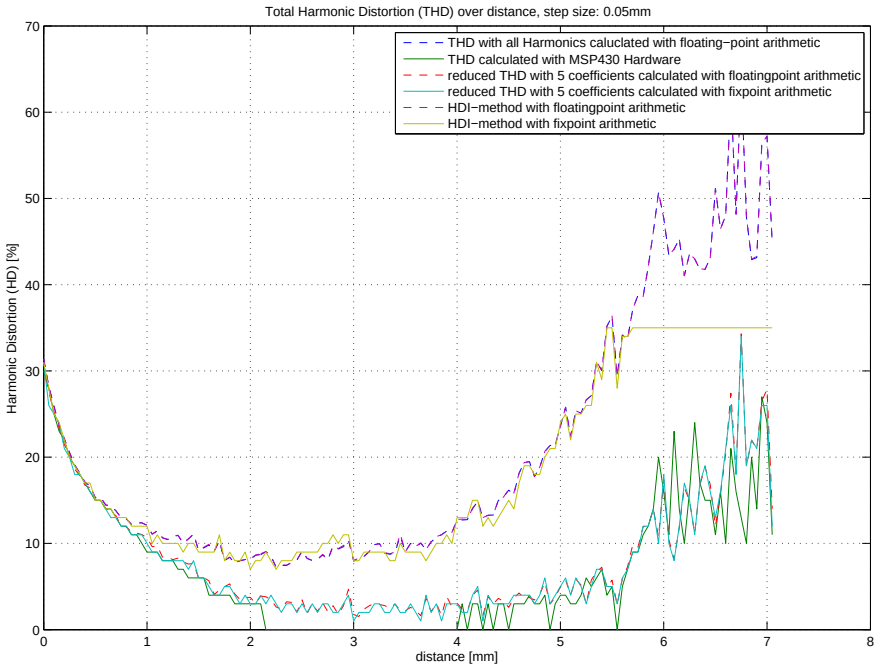

Abbildung D.45.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

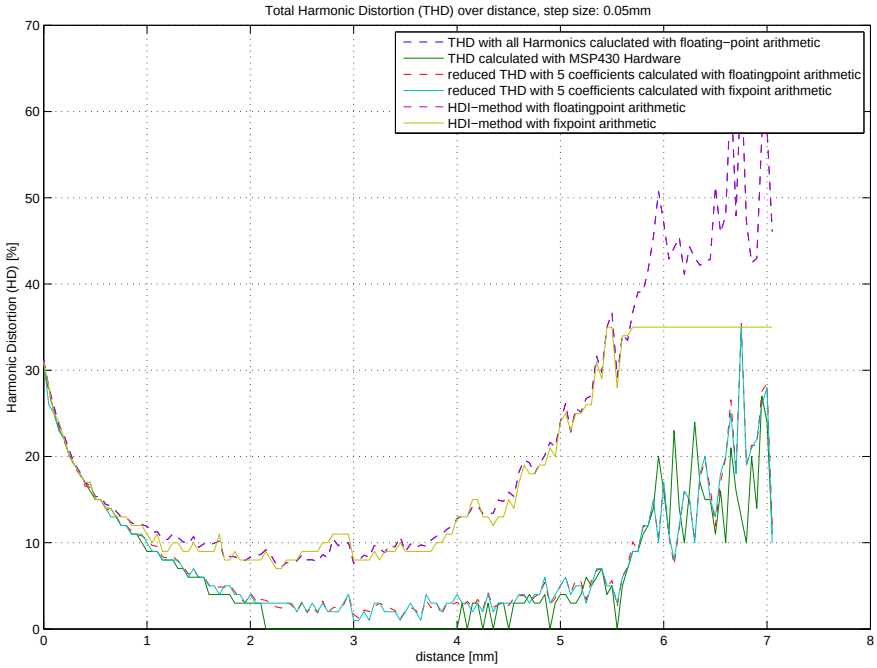

Abbildung D.46.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

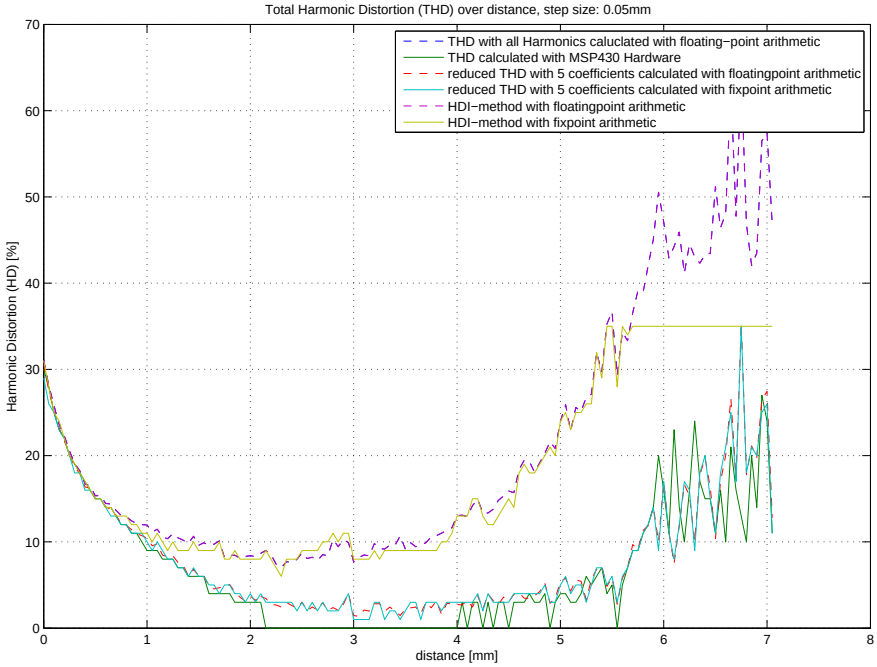

Abbildung D.47.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

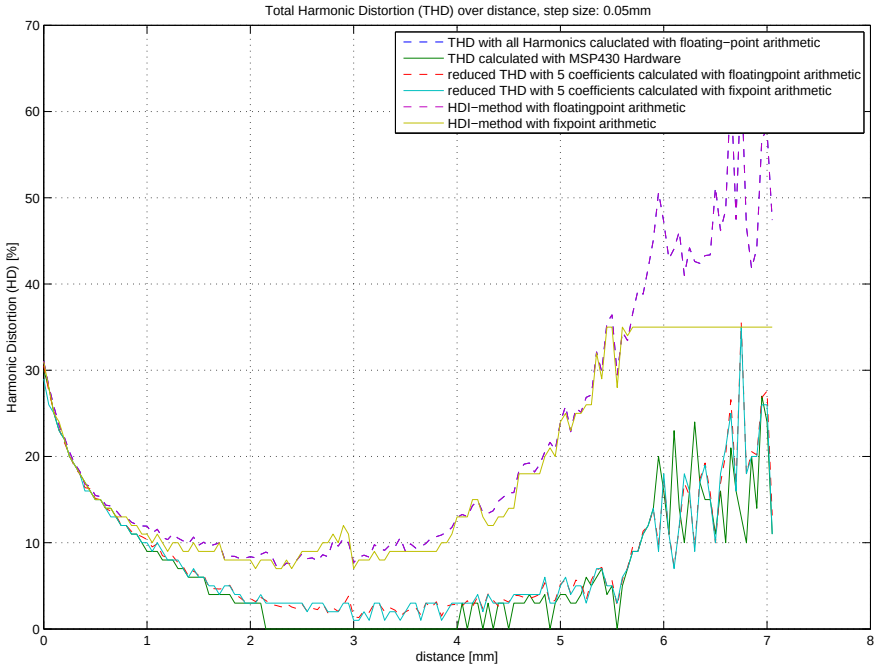

Abbildung D.48.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

D.2.2. Verwendung der 2. Berechnungsmöglichkeit

In diesem Abschnitt befinden sich die Ergebnisse des in Abschnitt 9.5 beschriebenen Versuchs.

Verwendung eines 3 Bit Analog-Digital-Umsetzers

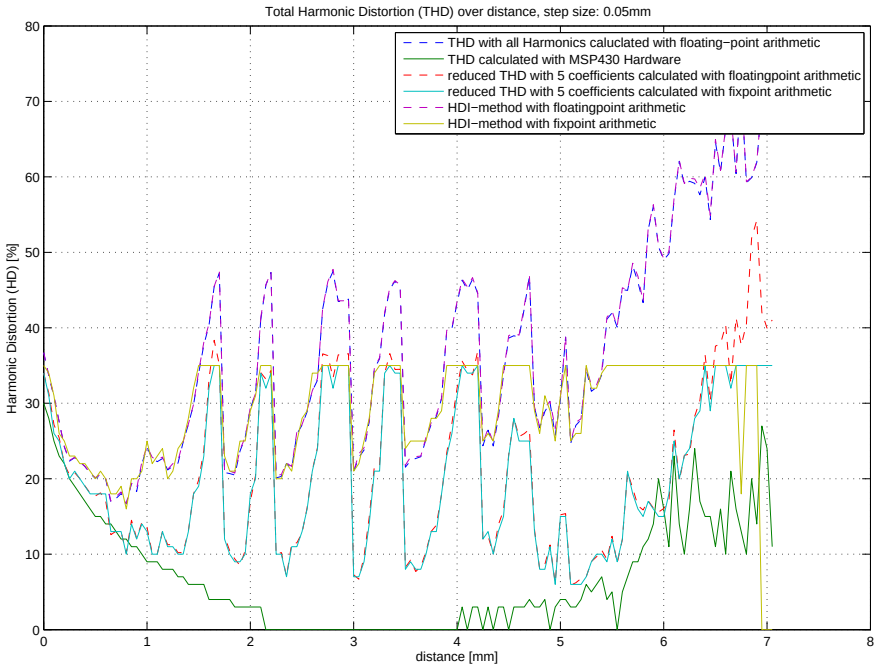

Abbildung D.49.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

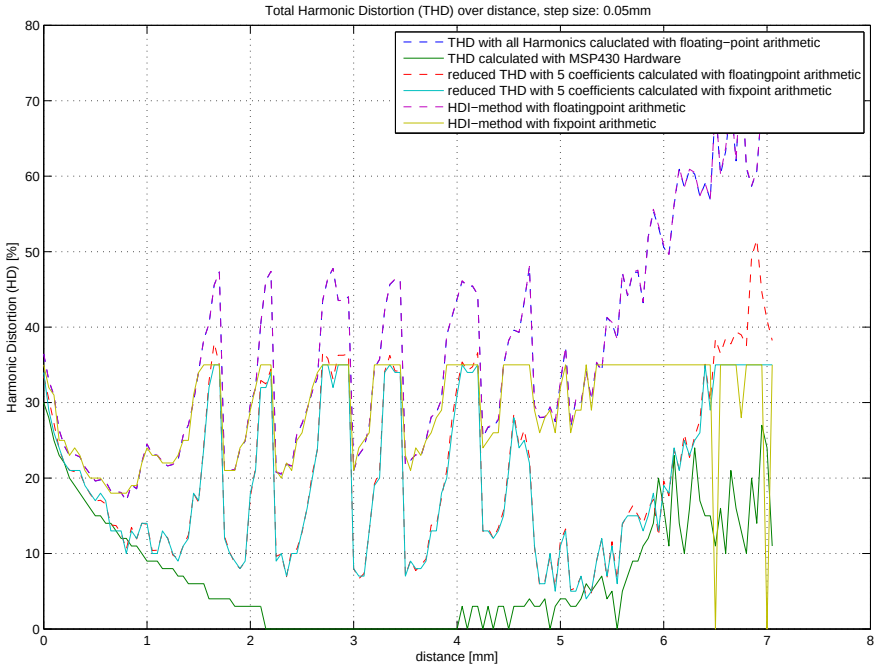

Abbildung D.50.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

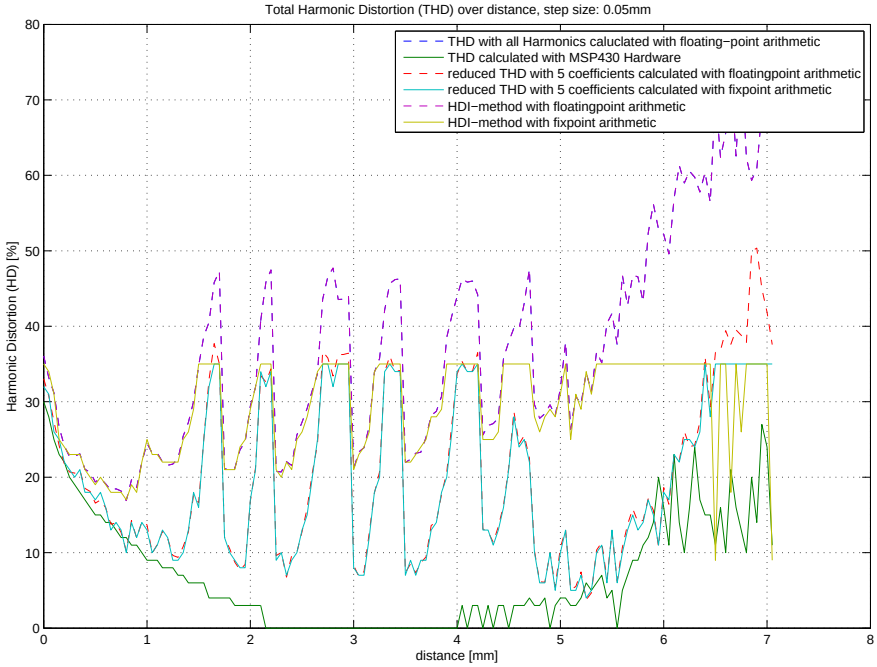

Abbildung D.51.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

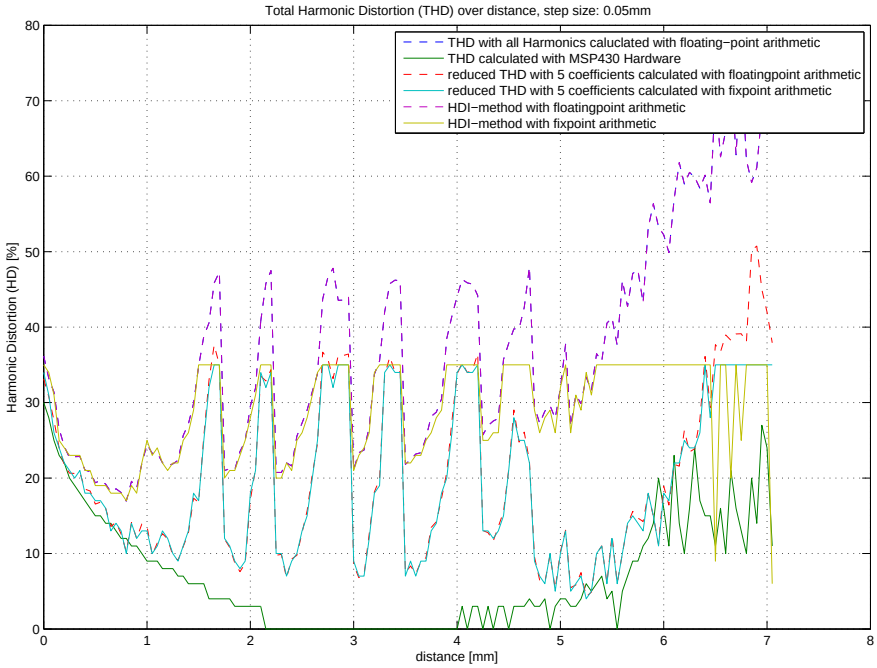

Abbildung D.52.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 4 Bit Analog-Digital-Umsetzers

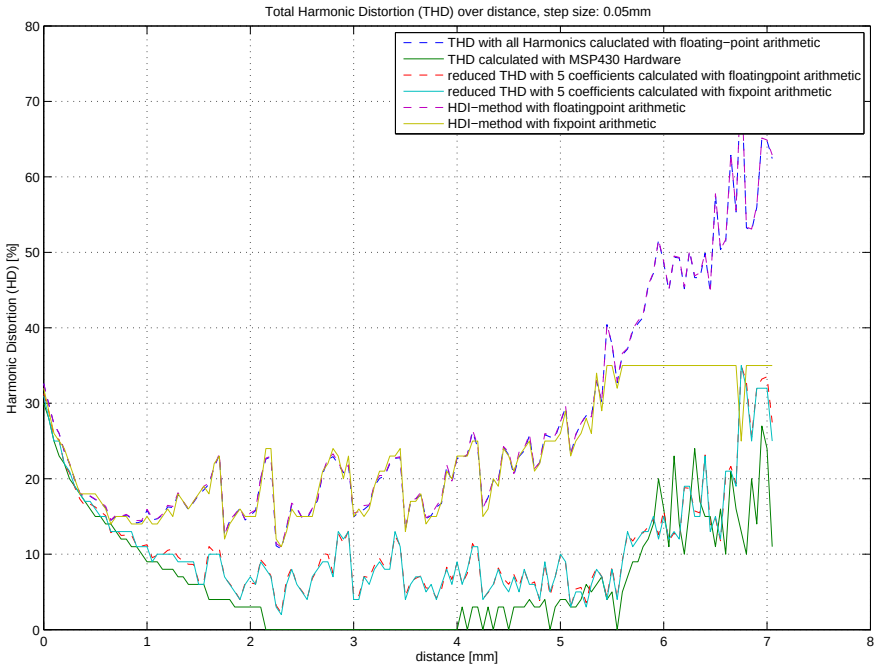

Abbildung D.53.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

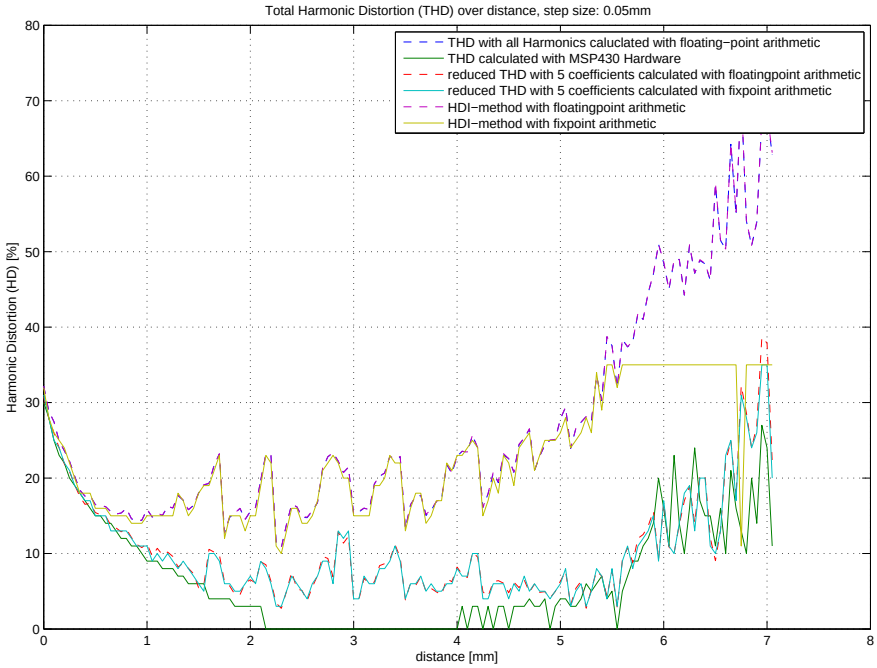

Abbildung D.54.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

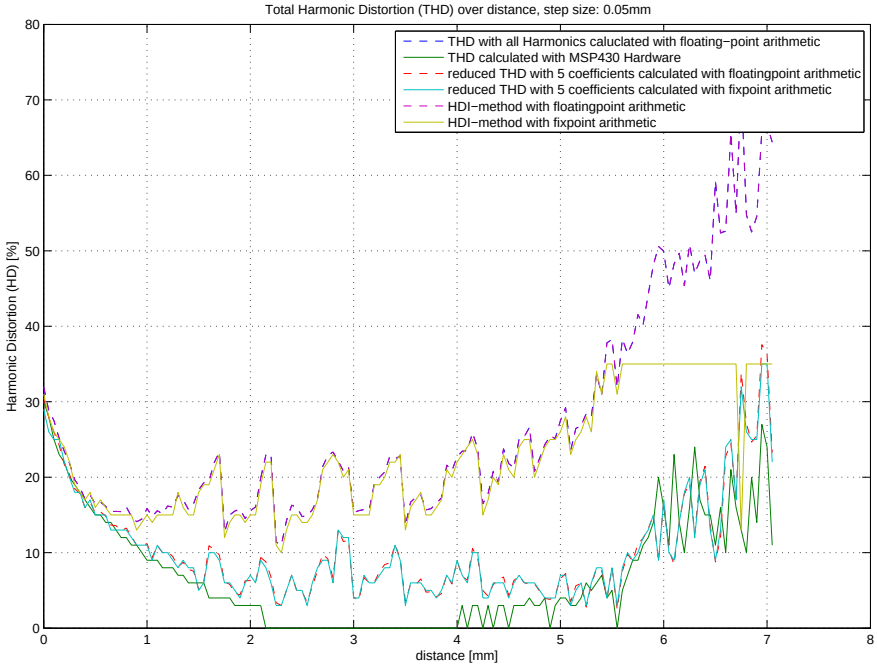

Abbildung D.55.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

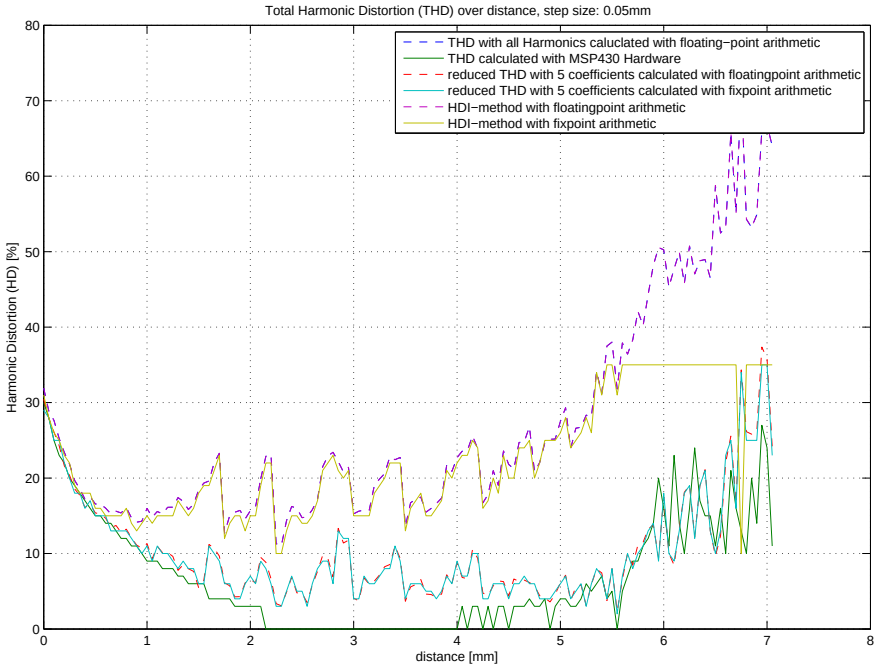

Abbildung D.56.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 5 Bit Analog-Digital-Umsetzers

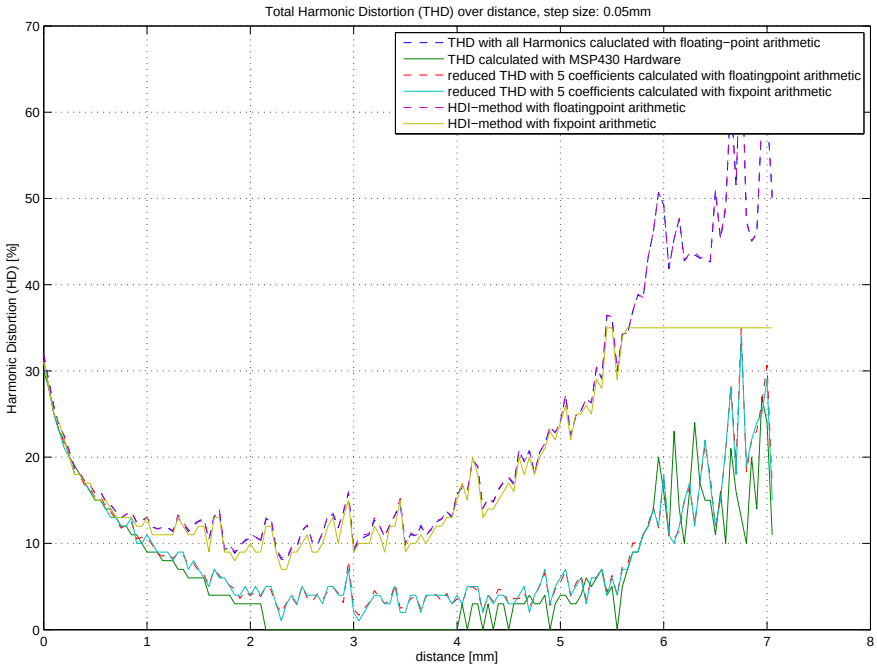

Abbildung D.57.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

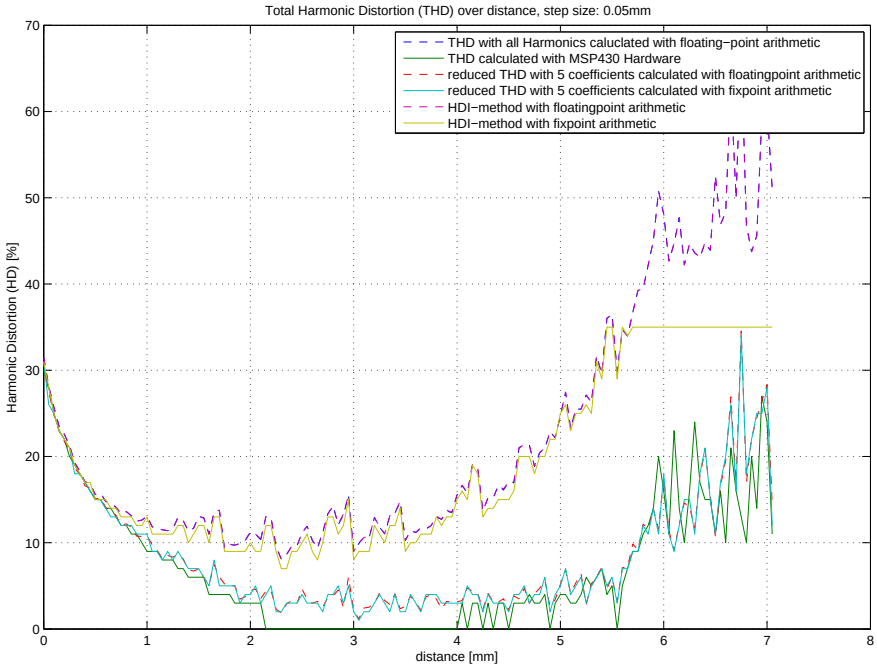

Abbildung D.58.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

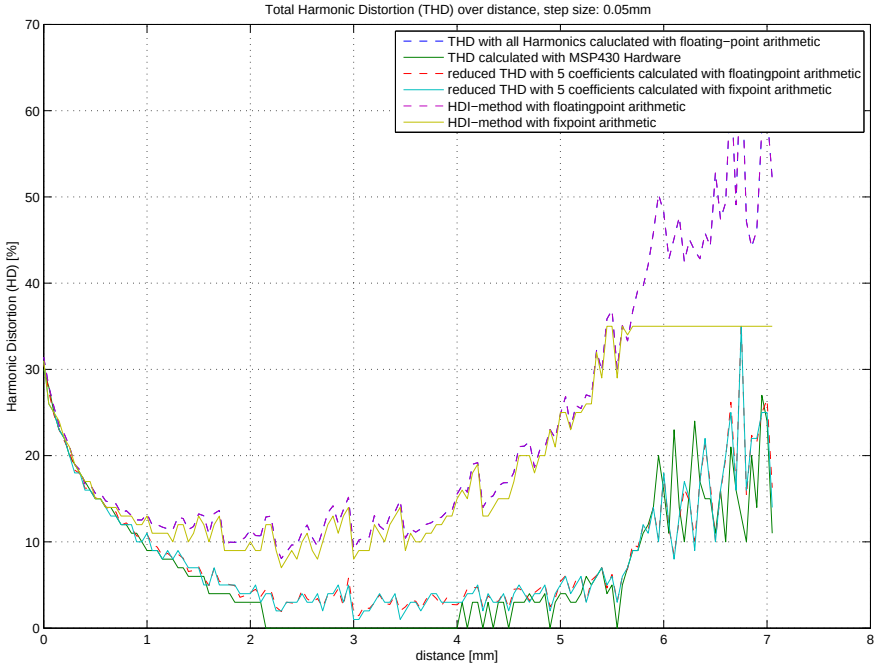

Abbildung D.59.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

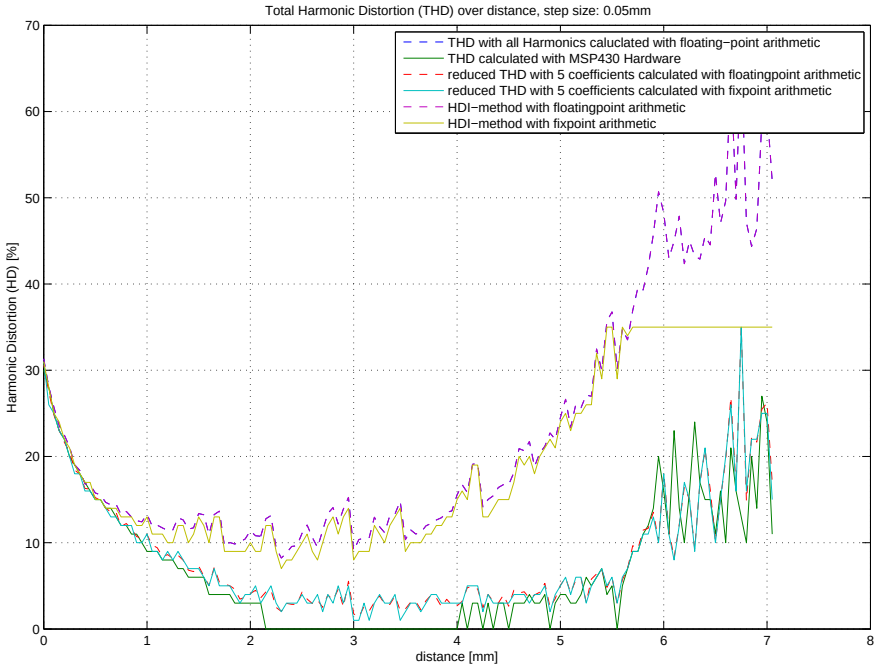

Abbildung D.60.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

Verwendung eines 6 Bit Analog-Digital-Umsetzers

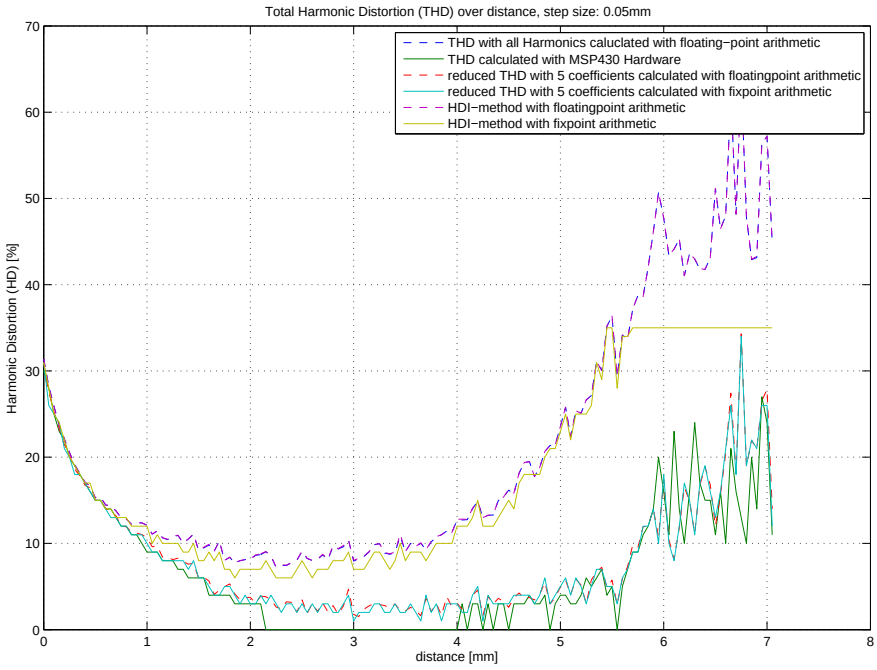

Abbildung D.61.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 64 Samples pro Periode [8]

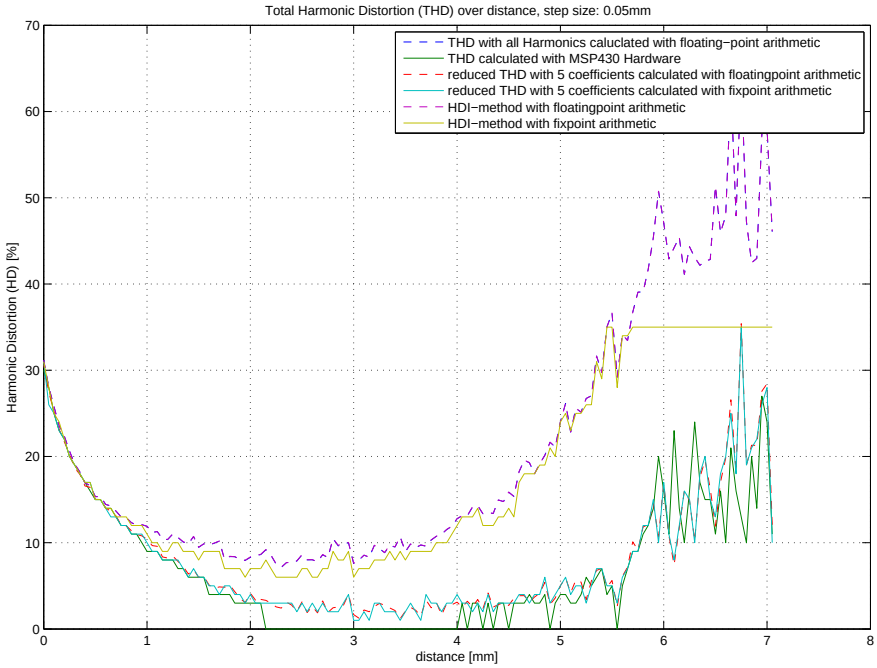

Abbildung D.62.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 128 Samples pro Periode [8]

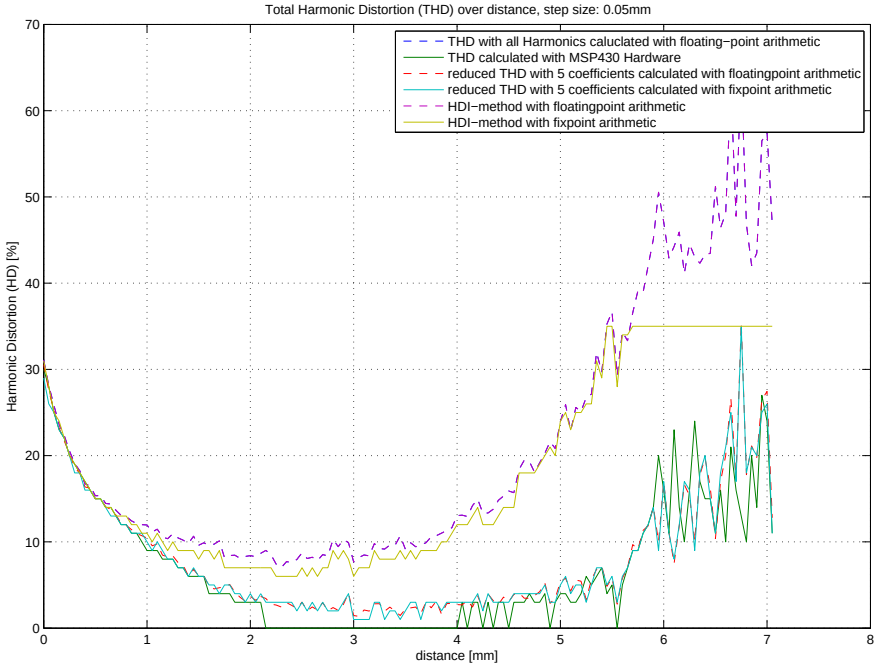

Abbildung D.63.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 256 Samples pro Periode [8]

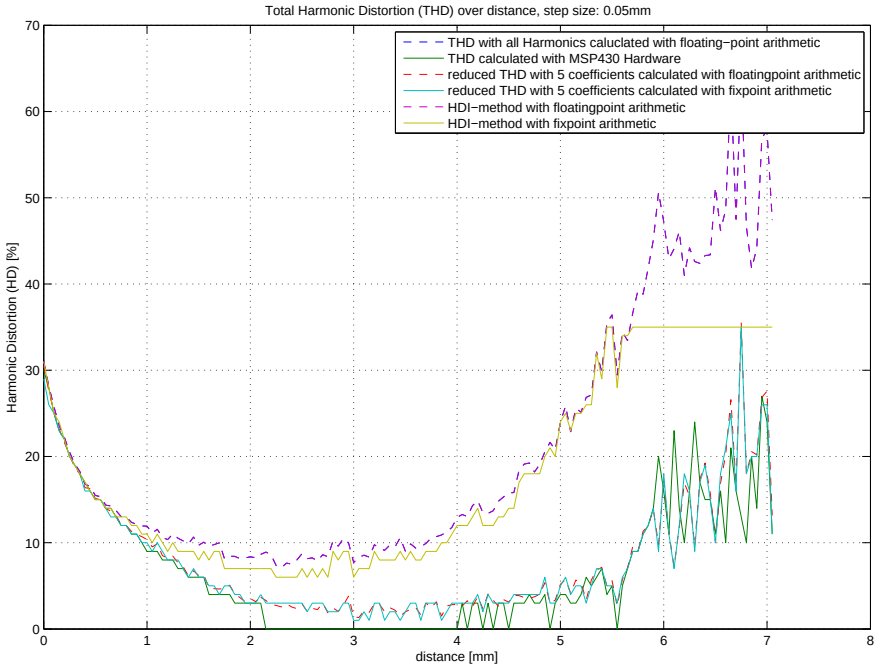

Abbildung D.64.: Klirrfaktor in Abhängigkeit von der Entfernung zwischen Sensor und Encoderrad unter Berücksichtigung von 512 Samples pro Periode [8]

E. Signalanalyse des Unrundlaufs des Encoderrades

E.1. keine Beseitigung des Unrundlaufs

Für die Plots ist hier die Messreihe 2009_10_07_rmp_01 verwendet worden

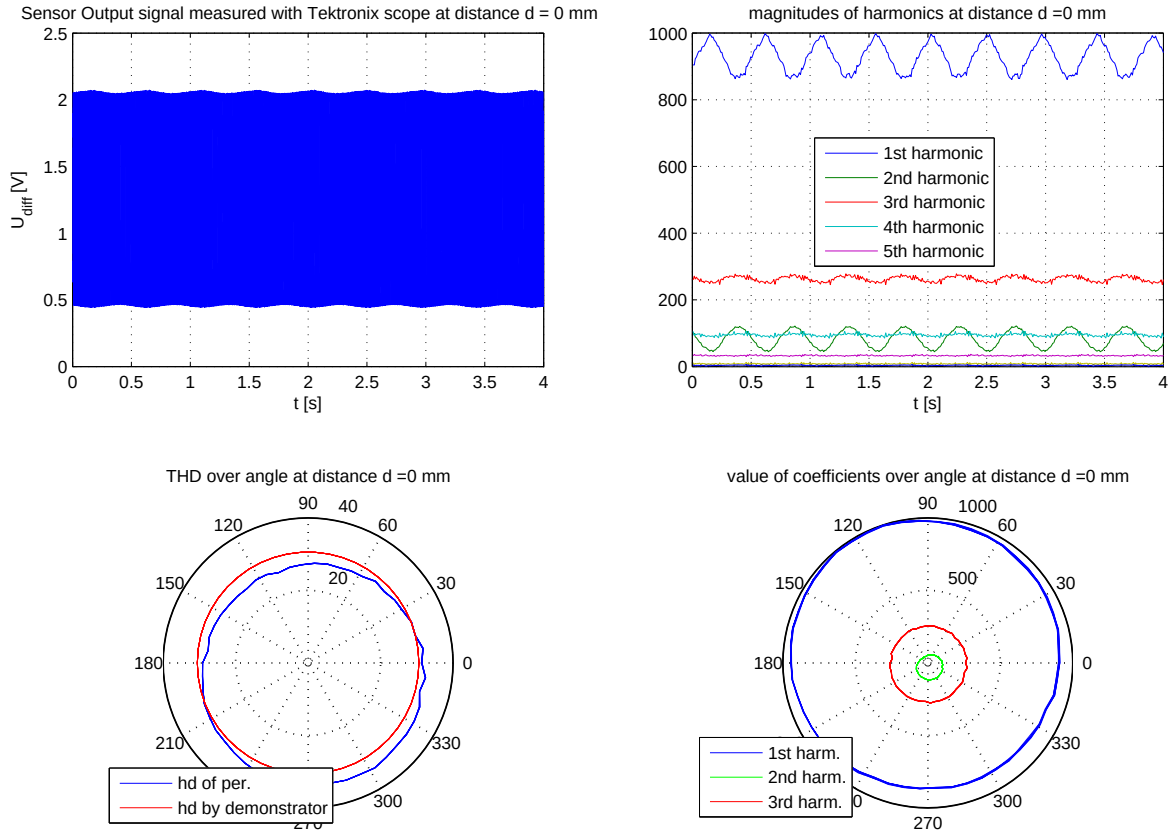

Abbildung E.1.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei sehr kleiner Entfernung zum Sensor [8]

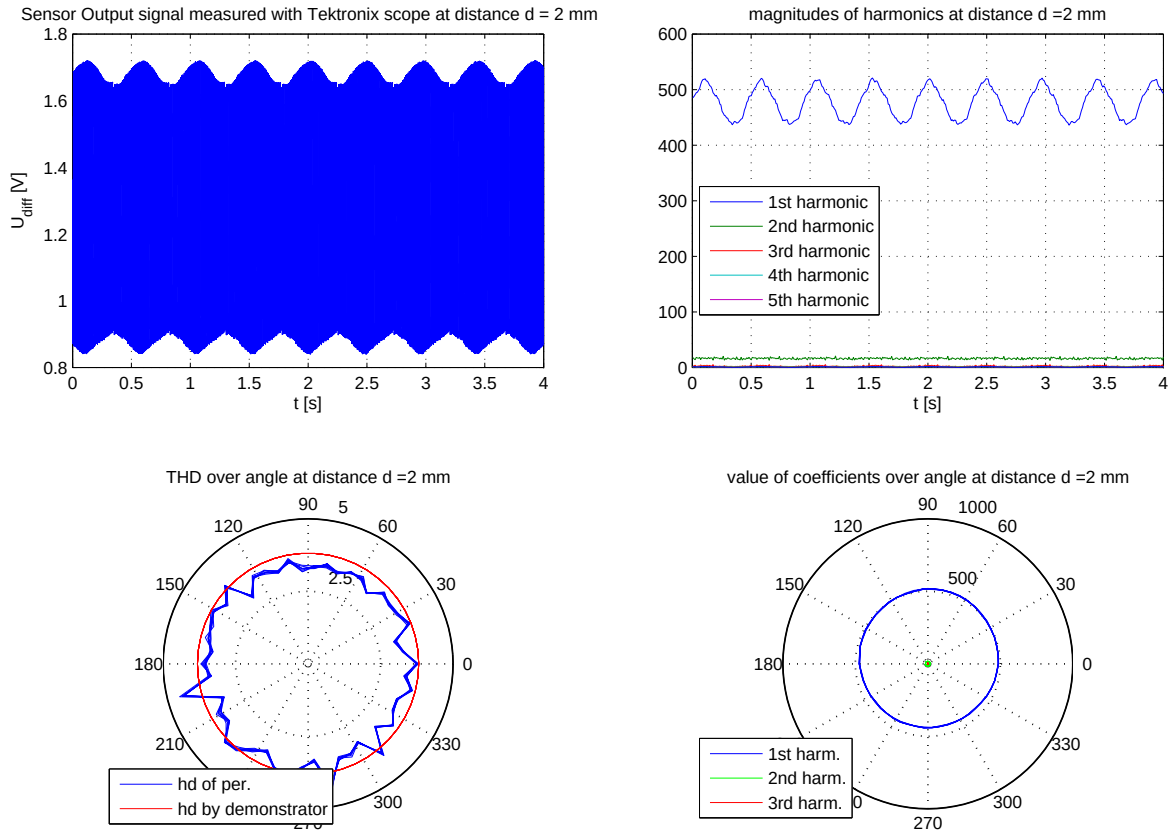

Abbildung E.2.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 2mm [8]

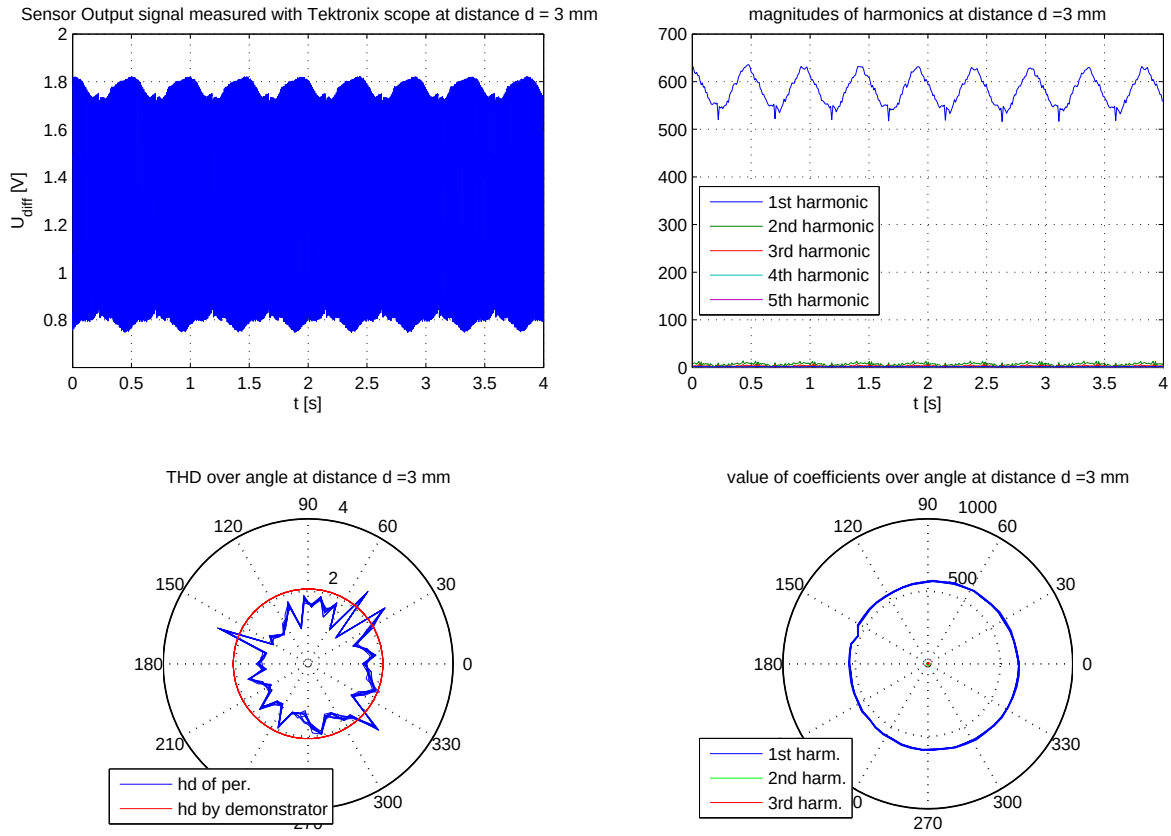

Abbildung E.3.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 3mm [8]

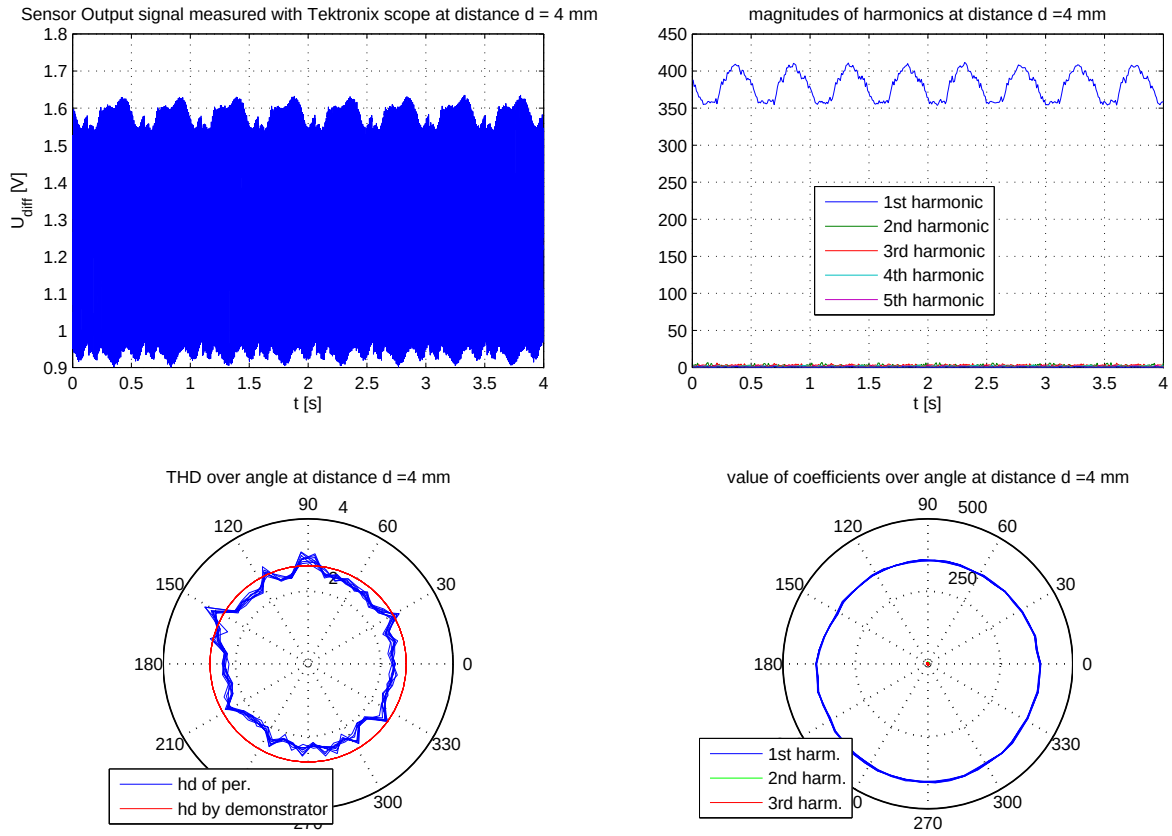

Abbildung E.4.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei seiner Entfernung zum Sensor von 4mm

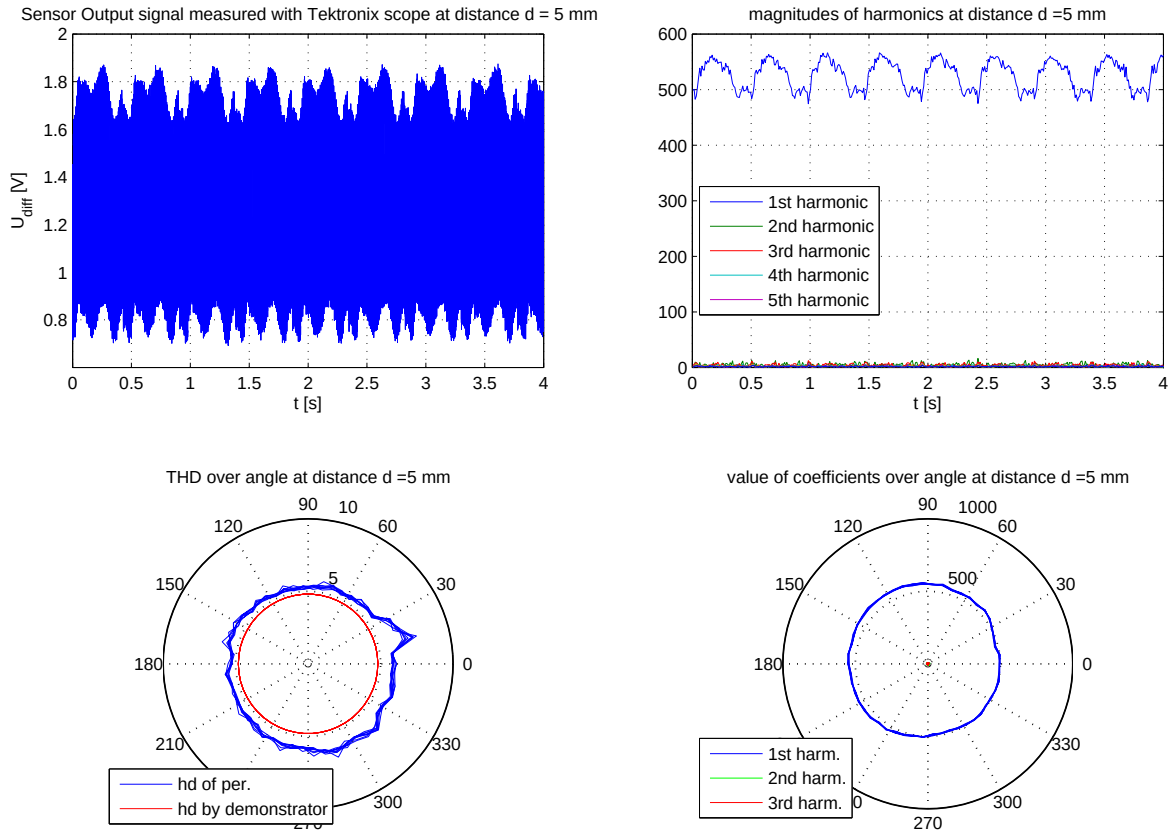

Abbildung E.5.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei seiner Entfernung zum Sensor von 5mm [8]

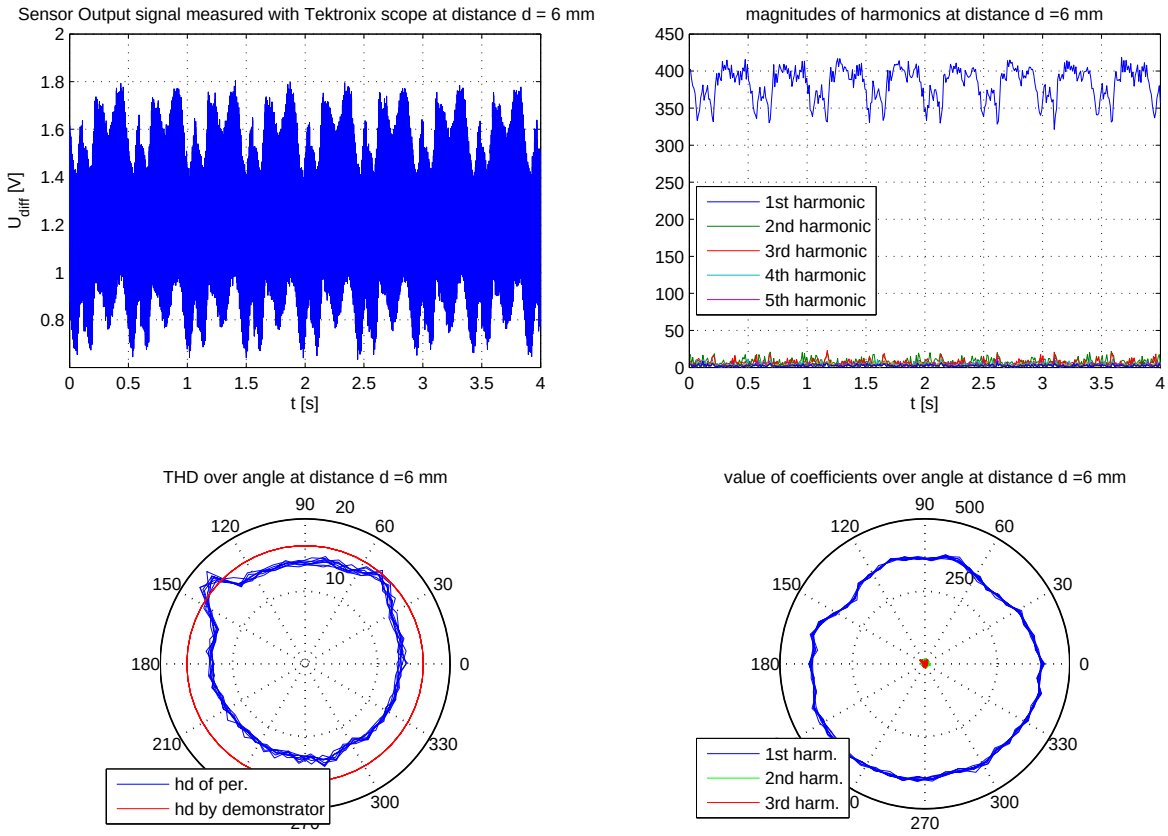

Abbildung E.6.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei seiner Entfernung zum Sensor von 6mm [8]

E.2. Beseitigung des Unrundlaufs

Für die Plots ist hier die Messreihe 2010_02_10_rmp_03 verwendet worden.

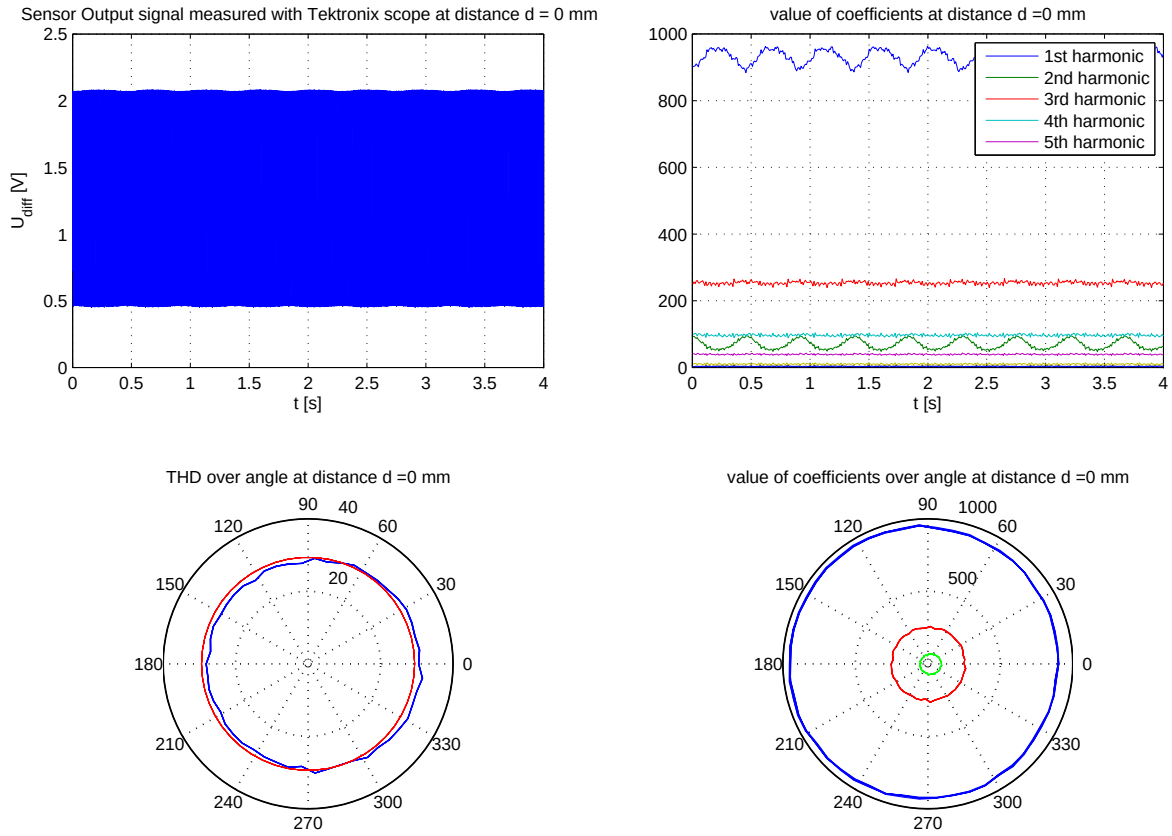

Abbildung E.7.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei sehr kleiner Entfernung zum Sensor [8]

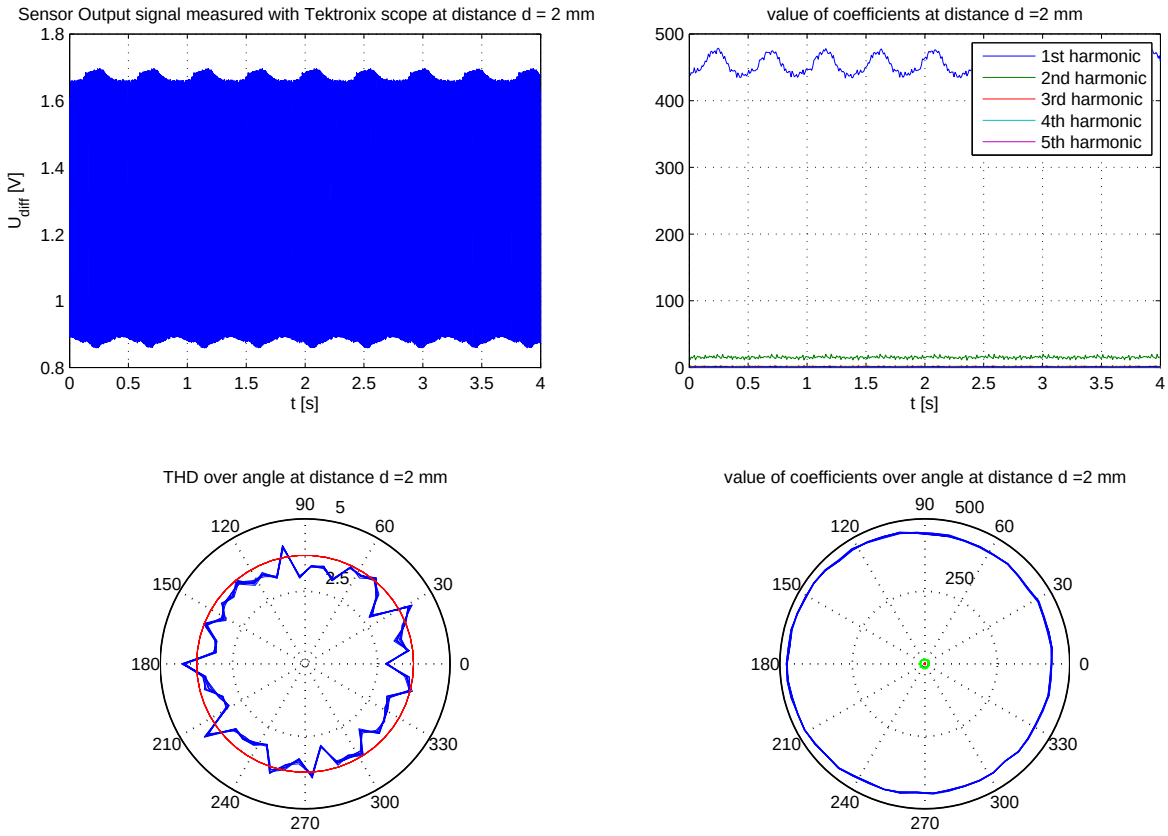

Abbildung E.8.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 2mm [8]

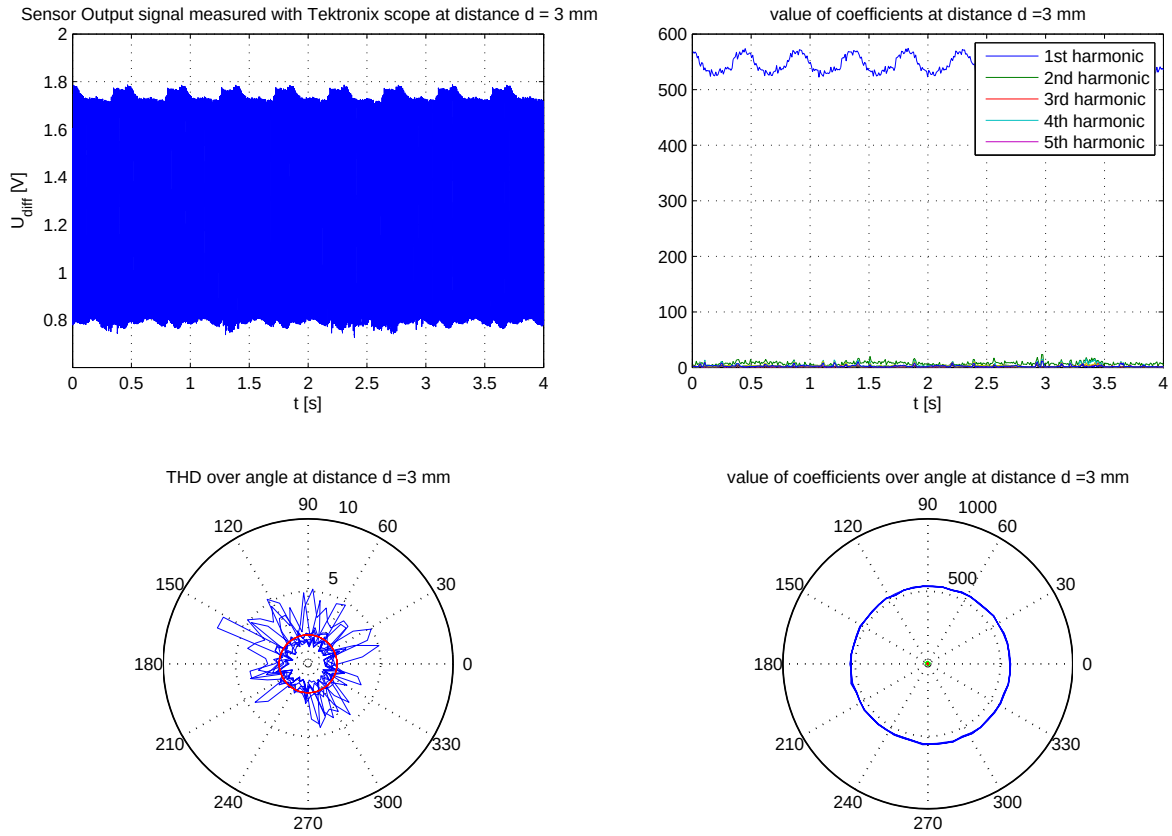

Abbildung E.9.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 3mm [8]

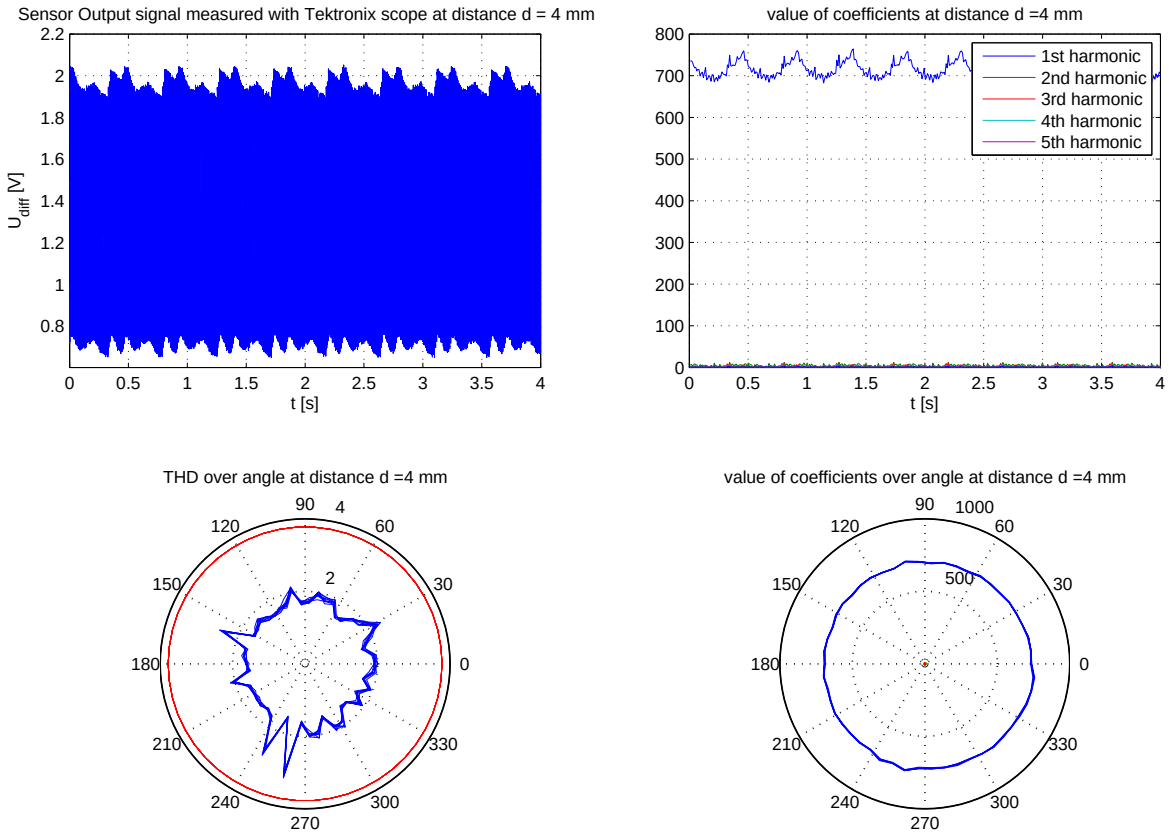

Abbildung E.10.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 4mm [8]

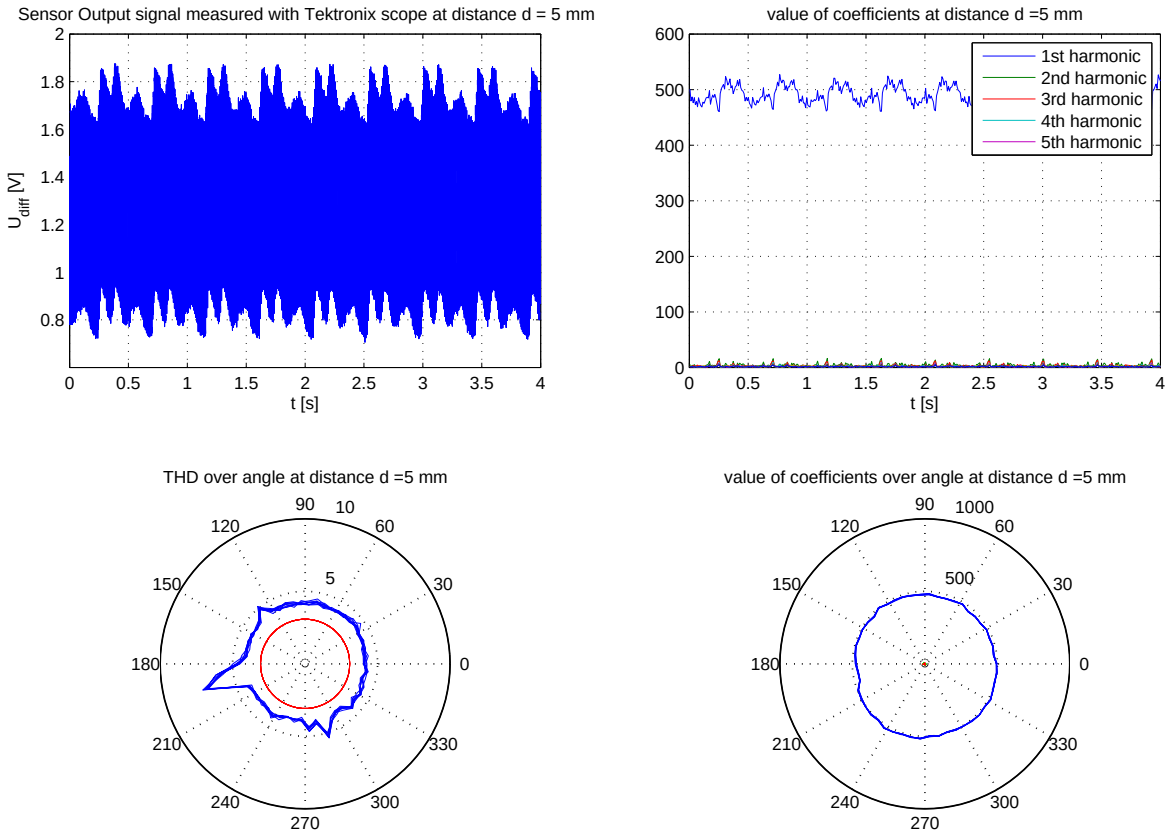

Abbildung E.11.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 5mm [8]

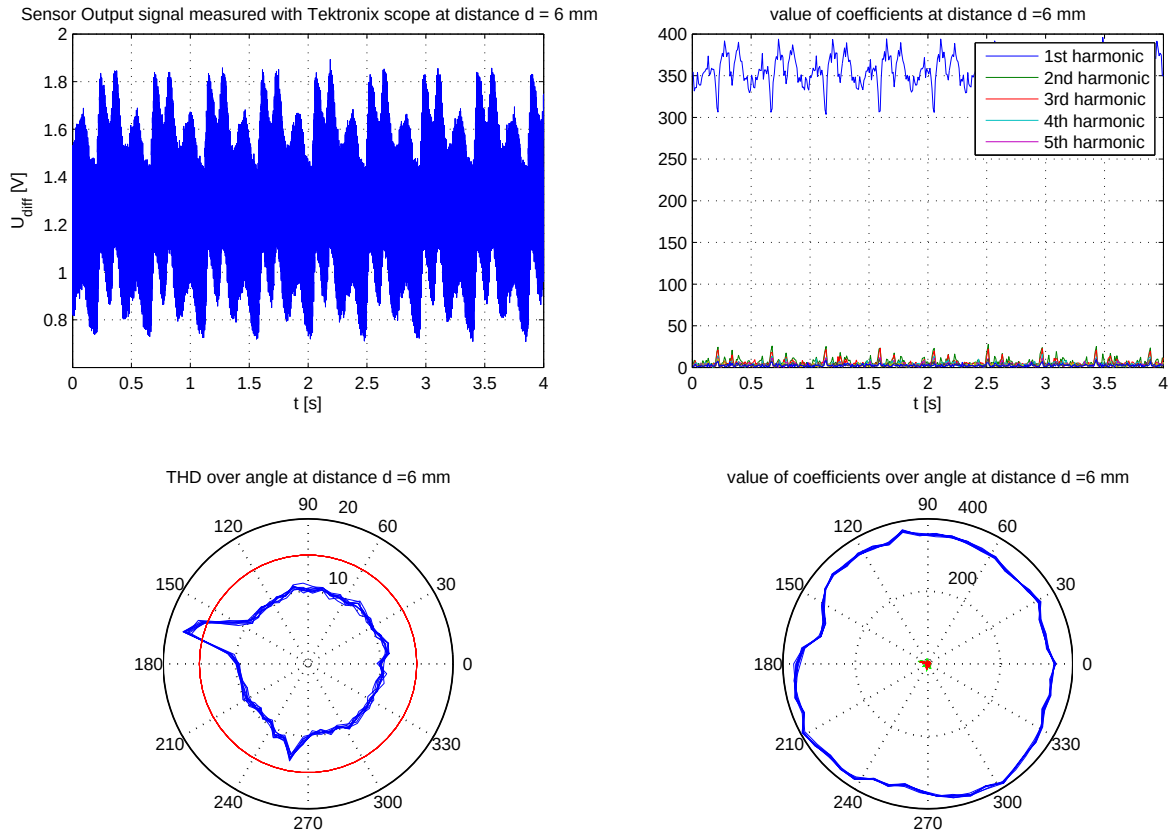

Abbildung E.12.: Ergebnis der Fouriertransformation jeder einzelnen Schwingung bei einer Entfernung zum Sensor von 6mm [8]

F. Quellcode

F.1. Matlab Quellcodes

F.1.1. Programme

prepare_measurements

Listing F.1:

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Berechnet aus den Daten des Radmessplatzen eine angegebene Anzahl von
% Koeffizienten aus Scopedaten und Daten vom Radmessplatz
%
% Erstellung einer neuen Datenstruktur (Dokumentation in Diplomarbeit)
%
% Version 1.1
%
%
% Datei:          prepare_measurements.m
%
% erstellt von: Lennart Koch
% erstellt am:  08.01.2010
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Arbeitsplatz bereinigen
clear all;
close all;
clc;

%% Vezeichnis mit Messdaten auswählen

DirName = uigetdir(pwd,'Choose directory with data-files');
cd(DirName);
list_of_files   = ls('*.rmp.mat');
number_of_files = size(list_of_files, 1);

load(list_of_files(1,:));           % Erstes Datenpaket laden

%% Umgebungsvariablen festlegen und auswerten

stepsize    = parameters.stepsize;      % Schrittweite in mm
startpoint  = parameters.startpoint;    % Anfangspunkt in mm
endpoint    = parameters.endpoint;      % Endpunkt in mm
fs          = parameters.samplerate;    % sampling frequenz von Scope

%Vektor mit den Entfernungen aller Messpunkte erzeugen
distance = startpoint : stepsize : endpoint;

% Anzahl der Messpunkte ermitteln
LM = length(distance); %Länge der Messreihe

if isempty(measure_demo) || isempty(measure_scope)
     error('In erstem Datenpaket muss mindestens eine Messung enthalten sein');
else
    %Anzahl der Samples Signalvektor des Radmessplatzes bestimmen
    L = length(measure_demo(1,1).u_diff);
```

```
53   end
54
55
56  % Eingabe der Anzahl der berücksichtigenten Perioden für Scopedaten
57  % und Eingabe der berücksichtigen Koeffizienten
58
59   N = input('Bitte␣Anzahl␣der␣Koeffizienten␣mit␣denen␣Signal␣approximiert␣werden␣soll,␣eingeben:␣');
60  while ( N > L/2 -2)
61      N = input('Fehler␣bei␣der␣Eingabe!!␣Bitte␣Anzahl␣der␣Koeffizienten␣erneut␣eingeben:␣');
62  end
63
64   periodes = input('Bitte␣Anzahl␣der␣Perioden,␣über␣die␣FFT␣berechnet␣werden␣soll,␣␣eingeben:␣');
65   while ( periodes > 100) %für mehr Perioden wird die Datenstruktur zu groß
66      N = input('Fehler␣bei␣der␣Eingabe!!␣Bitte␣Anzahl␣der␣Koeffizienten␣erneut␣eingeben:␣');
67   end
68
69   % Filter für Referenzsignal berechnen
70   [n,Wn] = cheb2ord([0.1  1]/125,[0.03  10]/125,3,40);
71   [b,a] = cheby2(n,40,Wn);
72   freqz(b,a,512,250);
73
74
75  %% Datenstrukturen erstellen
76
77  data(1,LM) = struct('measure_rmp', {0},...      % Daten Radmessplatz
78                      'measure_scope', {0},...    % Daten Scope
79                      'N',{N},...                 % Anzahl der berücksichtigten Koeffizienten
80                      'gain', {1},...             % Verstärungsfaktor des Hardware Vorvertärkers
81                      'shift_factor', {1},...     % ld(gain) Schibefaktor in Bit
82                      'distance', {1},...         % Entfernung Encoderrad zum Sensor
83                      'periodes', {periodes}...   % Anzahl der berücksichtigten Perioden
84                  );
85
86  for measure = 1:LM
87
88        data(1,measure).measure_rmp = ...
89                      struct ('u_diff', {0},...       % gemessenes Differzsignal
90                              'coefficients', {0},... % Fourier Koeffizienten
91                              'P_harm', {0},...       % Leistung der berücksichtigen Harmonischen
92                              'P_ob', {0},...         % Leistung der nicht berücksichtigen Harm.
93                              'hd_lut', {0},...       % THD berechnet  mit MSP Hardware
94                              'mag', {0}...           % Beträge der harmonischen (MSP Hardware)
95                              );
96
97      data(1,measure).measure_scope = ...
98                      struct ('u_diff',{0},...        % gemessenes Differzsignal(Ausschnitt)
99                              'coefficients' ,{0},... % Fourier Koeffizienten
100                             'P_harm' ,{0},...       % Leistung der berücksichtigen Harmonischen
101                             'P_ob', {0},...         % Leistung der nicht berücksichtigen Harm.
102                             'periodes',{periodes}...% Anzahl der berücksichtigten Perioden
103                             );
104 end
105
106 %Eingelesene Daten entfernen, da erneut 1. Teil der Messreihe geladen wird
107   clear measure_scope
108   clear measure_demo
109   clear parameters
110
111
112   %% Messdaten verarbeiten
113
114   k = 1;                                % Zähler für aktuelle Messung Radmessplatz
115   l = 1;                                % Zahler für aktuelle Messung Scope
116
117
118 for i=1:number_of_files                  % Dateien durchlaufen
119
120      disp(['loading␣' list_of_files(i,:)]);
121      load(list_of_files(i,:));           % jeweilige Datei laden
122      disp('prepare␣rmp␣data␣for␣simulation');
123     % Verarbeitung Daten Radmessplatz und der ersten Ebene in Datenstruktur
124
125      for j=1:length(measure_demo)
126          % Anzahl der Harmonischen im Signal
127          data(1,k).N                   = N;
128          % Verstärkung Vorverstärker
129          data(1,k).gain                = measure_demo(j).gain;
130          data(1,k).shift_factor        = log2(measure_demo(j).gain);
131          %Entfernung Ecoderrad Sensor
132          data(1,k).distance            = measure_demo(j).distance;
133
134          % Diffrenzsignal in neue Datenstruktur übernehmen
135          data(1,k).measure_rmp.u_diff  = measure_demo(j).u_diff;
136          data(1,k).measure_rmp.mag     = measure_demo(j).mag(:,1) / measure_demo(j).gain;
137          data(1,k).measure_rmp.hd_lut  = measure_demo(j).hd_lut;
```

```
138
139         % Wert der Samples ausrechnen(Umrechnung Jegenhorst)
140         Samples  = round(measure_demo(j).u_diff .* (2^12/2.5));
141
142         % Spektrum des Signals errechnen
143         S_sample = fft(Samples);
144
145         if L == 128 %In u_diff sind zwei Perioden enthalten
146             data(1,k).measure_rmp.periodes = 2;
147
148             % Fourierkoeffizienten abspeichern
149             data(1,k).measure_rmp.coefficients = S_sample(1:2*N+1);
150
151                 % Leistung der berücksichtigten Schwingungen ( Verwendung 2 Variante)
152             data(1,1).measure_rmp.P_harm  = 2 * 2.5/((L -1) *2^12) * ...
153                                             (sum(abs(S_sample(2:2 * N+1)).^2));
154
155             % Leistung nicht berücksichtigte Schwingungen (Verwendung 2 Variante)
156             data(1,1).measure_rmp.P_ob  =  2 * 2.5/((L -1) *2^12) *...
157                                             sum(abs(S_sample(2* N+ 2:fix(L/2)-2)).^2);
158         elseif L == 64 %In u_diff ist eine Periode enthalten
159
160             data(1,k).measure_rmp.periodes = 1;
161
162             % Fourierkoeffizienten abspeichern
163             data(1,k).measure_rmp.coefficients = S_sample(1:N+1);
164
165             % Leistung der berücksichtigten Schwingungen ( Verwendung 2 Variante)
166             data(1,1).measure_rmp.P_harm  = 2 * 2.5/(L *2^12) * ...
167                                             (sum(abs(S_sample(2:N+1)).^2));
168
169             % Leistung nicht berücksichtigte Schwingungen (Verwendung 2 Variante)
170             data(1,1).measure_rmp.P_ob  =  2 * 2.5/(L *2^12) *...
171                                            sum(abs(S_sample(N+ 2:fix(L/2)-2)).^2);
172          end
173
174
175
176         k = k + 1;
177         clear('Samples');
178     end
179
180
181     % Verarbeitung Daten Oszilloskop
182     disp('prepare␣scope␣data␣for␣simulation');
183     for j=1:length(measure_scope)
184         % Messdaten laden
185         c1   = measure_scope(j).u_diff_y';       % Differenzsignal Channel 1
186
187         cf1 = filter(b,a,c1); % Eingangssignal filtern
188         ref=cf1-mean(cf1);
189
190         % Gleichanteil von Signal abziehen
191         c1w = c1 - mean(c1);
192
193         % Referenzsignal(Augang externer Sensor laden
194         ref_sig = ref - mean(ref);
195
196         % Zähler zu 0 setzen
197         counter = 0;
198
199          %Nulldurchgänge Refernenzsignal erkennen
200         for n=1:(length(ref_sig)-1)
201             if (sign(ref_sig(n)) ~= sign(ref_sig(n+1))); %wenn Nulldurchgang
202                 counter = counter+1;
203                 if counter == 2*20 -1
204                     n_start = n;
205                 end
206                 if counter ==  2*20 -1 + 2* periodes % ab 20. Periode(Jegenhorst)
207                     n_end = n;
208                 end
209             end
210         end
211
212         %Faktor aus Analyse der Schwingungen ermittelt da Signal zu Groß wird wg Radeiern
213         data(1,1).measure_scope.u_diff = c1(n_start:n_end) + 0.1;
214
215         % Anzahl der Samples des Ausschnitts bestimmen
216         L_sig = length(data(1,1).measure_scope.u_diff);
217
218         % Wert der Samples ausrechnen(Umrechnung Jegenhorst)
219         Samples = round(data(1,1).measure_scope.u_diff * (2^12/2.5));
220
221         % FFT von Signal ausrechnen
222         S_app = fft(Samples);
```

```
223
224         % Koeffizienten abspeichen, idealierweise ist nur jeder "periodes" Wert
225         % +1 da erster Wert Gleichanteil
226         data(1,1).measure_scope.coefficients = S_app(1:N*periodes +1).';
227
228
229         %Leistung der berücksichtigten Schwingungen ( Verwendung 2 Variante)
230         data(1,1).measure_scope.P_harm  = 2 * 2.5/((L_sig -1) *2^12) * (sum(abs(S_app(2:N*periodes+1)).^2));
231
232         %Leistung nicht berücksichtigte Schwingungen (Verwendung 2 Variante)
233         data(1,1).measure_scope.P_ob  =  2 * 2.5/((L_sig -1) *2^12) *...
234                                          sum(abs(S_app(N * periodes + 2:fix(L_sig/2)-2)).^2);
235
236
237         l = l +1;
238         clear('Samples');
239     end
240     clear measure_scope                         % clean up workspace
241     clear measure_demo
242 end
243
244 %% Neue Datenreihe abspeichern
245
246
247 % Datenreihe unter anderem Namen abspeichern
248
249 old_name = list_of_files(1,:); %Dateinamen in String kopieren
250 s = findstr(old_name,'_part'); %nach _part suchen
251 file = strcat(old_name(1:s-1), '_for_simulation.mat'); %neuen  Dateinamen erstellen
252
253 save(file, 'data', 'parameters');
```

prepare_measurements_sim

Listing F.2:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Das Programm bildet die Unterabtastung aus den Oszilloskopdaten nach.
% Dafür werden die Messdaten des Oszilloskops verwendet.
%
% Eingaben
% -Anzahl der Harmonischen, mit denen das Singal approximiert werden soll
% -Anzahl der Perioden des Oszilloskopsignals über die FFT Berechnet werden
%  soll
% -Aufnamhereihenfolge der Werte zufällig oder der Reihe nach
% -maximale Anzahl an Perioden, die Verstreichen sollen, bis Abtastung
%  erfolgt
% -minimale Anzahl an Perioden, die Verstreichen sollen, bis Abtastung
%  erfolgt
%
% Erstellung einer neuen Datenstruktur (Dokumentation in Diplomarbeit)
%
% Version 1.1
%
%
% Datei:          prepare_measurements_sim.m
%
% erstellt von: Lennart Koch
% erstellt am:  25.01.2010
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Arbeitsplatz bereinigen
clear all;
close all;
clc;

%% Vezeichnis mit Messdaten auswählen

DirName = uigetdir(pwd,'Choose␣directory␣with␣data-files');
cd(DirName);
list_of_files    = ls('*.rmp.mat');
number_of_files = size(list_of_files, 1);

load(list_of_files(1,:));            % Erstes Datenpaket laden

%% Umgebungsvariablen festlegen und auswerten

stepsize    = parameters.stepsize;      % Schrittweite in mm
startpoint  = parameters.startpoint;    % Anfangspunkt in mm
endpoint    = parameters.endpoint;      % Endpunkt in mm
fs          = parameters.samplerate;    % sampling frequenz von Scope

%Vektor mit den Entfernungen aller Messpunkte erzeugen
distance = startpoint : stepsize : endpoint;

% Anzahl der Messpunkte ermitteln
LM = length(distance); %Länge der Messreihe

if isempty(measure_demo) || isempty(measure_scope)
            error('In␣erstem␣Datenpaket␣muss␣mindestens␣eine␣Messung␣enthalten␣sein');
else
   %Anzahl der Samples Signalvektor des Radmessplatzes bestimmen
   L = length(measure_demo(1,1).u_diff);
 end



% Eingabe der oben beschriebenen Parameter
 N = input('Bitte␣Anzahl␣der␣Koeffizienten␣mit␣denen␣Signal␣approximiert␣werden␣soll,␣eingeben:␣');
while ( N > L/2 -2)
     N = input('Fehler␣bei␣der␣Eingabe!!␣Bitte␣Anzahl␣der␣Koeffizienten␣erneut␣eingeben:␣');
end

 periodes = input('Bitte␣Anzahl␣der␣Perioden,␣über␣die␣FFT␣berechnet␣werden␣soll,␣␣eingeben:␣');
 while ( periodes > 100) %für mehr Perioden werden dei Messreihen zu groß
    periodes = input('Fehler␣bei␣der␣Eingabe!!␣Bitte␣Anzahl␣der␣Koeffizienten␣erneut␣eingeben:␣');
 end

 random = input('Soll␣die␣Aufnahme␣der␣Samples␣der␣Reihe␣nach␣oder␣zufäälig␣erfolgen␣(0:␣der␣Reihe␣nach;␣1:␣zufällig:␣');
 while (random ~=1 && random ~= 0)
```

```
80        random = input('Fehler bitte erneut eingeben (0: der Reihe nach; 1: zufällig: ');
81    end
82
83    tm_max = input('Bitte maximale Anzahl Perioden angeben, die ein Wert aufgenommen werden soll (Hardware 7 Perioden): ');
84     while ( tm_max < 1) %für mehr Perioden werden dei Messreihen zu groß
85       tm_max = input('Fehler: maximal bei jeder Periode möglich');
86     end
87
88    tm_min = input('Bitte minimale Anzahl Perioden angeben, die ein Wert aufgenommen werden soll (Hardware 6 Perioden): ');
89     while ( tm_min < 1) %für mehr Perioden werden dei Messreihen zu groß
90       tm_min = input('Fehler: maximal bei jeder Periode möglich');
91    end
92
93
94    % Filter für Referenzsignal berechnen
95    [n,Wn] = cheb2ord(1/125,2.5/125,3,80);
96    [b,a] = cheby2(n,80,Wn);
97
98    freqz(b,a,512,250);
99
100
101   %% Datenstrukturen erstellen
102   simulation_parameters = struct( 'zufall', {random},... %Zufällige oder n. zufällige Aufnahmereihenfolge
103                                   'min_tw', {tm_min},... %min. Anzahl Perioden, bis Wert aufgenommen
104                                   'max_tw', {tm_max}...  %max. Anzahl Perioden, bis Wert aufgenommen
105                                 );
106
107
108   data(1,LM) = struct('measure_rmp', {0},...      % Daten Radmessplatz
109                       'measure_scope', {0},...    % Daten Scope
110                       'N' ,{0},...                % Anzahl der berücksichtigten Koeffizienten
111                       'gain' , {0},...            % Verstärungsfaktor des Hardware Vorvertärkers
112                       'shift_factor', {0},...     % ld(gain) Schibefaktor in Bit
113                       'distance', {0},...         % Entfernung Encoderrad zum Sensor
114                       'periodes', {periodes}...   % Anzahl der berücksichtigten Perioden
115                  );
116
117   for measure = 1:LM
118
119        data(1,measure).measure_rmp = struct ('u_diff', {0},...      % gemessenes Differenzsignal
120                                              'coefficients', {0},...% Fourier Koeffizienten
121                                              'P_harm', {0},...      % Leistung der berücksichtigen Harmonischen
122                                              'P_ob', {0},...        % Leistung der nicht berücksichtigen Harm.
123                                              'hd_lut', {0},...      % THD berechnet  mit MSP Hardware
124                                              'mag', {0}...          % Beträge der Harmonischen berechnet mit MSP Hardware)
125                                              );
126
127       data(1,measure).measure_scope = struct ('u_diff',{0},...       % gemessenes Differzsignal(Ausschnitt)
128                                               'coefficients' ,{0},...% Fourier Koeffizienten
129                                               'P_harm' ,{0},...      % Leistung der berücksichtigen Harmonischen
130                                               'P_ob', {0},...        % Leistung der nicht berücksichtigen Harm.
131                                               'periodes', {periodes}...% Anzahl der berücksichtigten Perioden
132                                            );
133   end
134
135   %Eingelesene Daten entfernen
136
137    clear measure_scope
138    clear measure_demo
139    clear parameters
140
141
142    %% Messdaten verarbeiten
143
144    k = 1;       % Aktuelle Messreihe Radmessplatz
145    l = 1;       % Aktuelle Messreihe Scope
146
147
148   for i=1:number_of_files                  % Dateien durchlaufen
149
150       disp(['loading ' list_of_files(i,:)]);
151       load(list_of_files(i,:));            % jeweilige Datei laden
152
153      % Verarbeitung Daten ersten Ebene in Datenstruktur
154
155       for j=1:length(measure_demo)
156
157           data(1,k).N                    = N;
158           data(1,k).gain                 = measure_demo(j).gain;
159           data(1,k).shift_factor         = log2(measure_demo(j).gain);
160           data(1,k).distance             = measure_demo(j).distance;
161
162           data(1,k).measure_rmp.hd_lut   = measure_demo(j).hd_lut;
163           k = k + 1;
164       end
```

```
165
166
167     % Verarbeitung Daten Oszilloskop
168     disp('prepare scope data for simulation');
169     for j=1:length(measure_scope)
170         % Messdaten laden
171         c1    = measure_scope(j).u_diff_y';       % Differenzsignal Channel
172
173         cf1 = filter(b,a,c1); % Eingangssignal filtern
174         ref=cf1-mean(cf1);
175
176         % Referenzsignal(Augang externer Sensor laden
177         ref = ref - mean(ref);
178
179         % Zähler zu 0 setzen
180         counter = 0;
181
182          %Referenzsignal auswerten, damit immer ganze Perioden
183          %entnommen werden
184         for n=1:(length(ref)-1)
185             if (sign(ref(n)) ~= sign(ref(n+1)));
186                 counter = counter+1;
187                 if counter == 2*20 -1
188                     n_start = n;
189                 end
190                 if counter ==  2*20 -1 + 2* periodes
191                     n_end = n;
192                 end
193                 if counter ==  2*20 -1 + 2*(400 + 1)
194                     n_end_rmp_sim = n; %Ende Signalvektor für Radmessplatz Simulation 400 Perioden
195                 end
196
197             end
198         end
199
200         %Faktor aus Analyse der Schwingungen ermittelt da Signal zu Groß wird wg Radeiern
201         data(1,1).measure_scope.u_diff = c1(n_start:n_end) + 0.1;
202
203         %Referenzsignal auf gleiche Weise anpassen
204         ref_sig = ref(n_start:n_end);
205
206
207         % Anzahl der Samples des Ausschnitts bestimmen
208         L_sig = length(data(1,1).measure_scope.u_diff);
209
210         % Wert der Samples ausrechnen(Umrechnung Jegenhorst)
211         Samples = round(data(1,1).measure_scope.u_diff * (2^12/2.5));
212
213         % FFT von Signal ausrechnen
214         S_app = fft(Samples);
215
216         % Koeffizienten abspeichen, idealierweise ist nur jeder "periodes" Wert
217         % != 0, da N Koeffizienten berücksichtigt werden sollen N*periodes +1,
218         % +1 da erster Wert Gleichanteil
219         data(1,1).measure_scope.coefficients = S_app(1:N*periodes +1).';
220
221
222         %Leistung der berücksichtigten Schwingungen ( Verwendung 2 Variante)
223         data(1,1).measure_scope.P_harm  = 2 * 2.5/((L_sig -1)*2^12) * ...
224                                                 (sum(abs(S_app(2:N*periodes+1)).^2));
225
226         %Leistung nicht berücksichtigte Schwingungen (Verwendung 2 Variante)
227         data(1,1).measure_scope.P_ob  =  2 * 2.5/((L_sig -1) *2^12) * ...
228                                                 sum(abs(S_app(N * periodes + 2:fix(L_sig/2)-2)).^2);
229
230         %% Nachbildung der Unterabtastung des Radmessplatzes
231
232
233         % Um Timerticks von 12,8us(Jegenhorst) zu Simulieren muss dieses
234         % Signal( fs = 250 kHZ) um den Faktor 3 reduziert werden
235         s = downsample(c1(n_start:n_end_rmp_sim) + 0.1, 3); % Signal mit 400 Perioden
236
237         % Ensprechend das Referenzsignal auch
238         ref_sig_sim  = downsample(ref(n_start:n_end_rmp_sim),3);
239
240         counter = -1;       % Zähler gür die Nulldurchgäge
241         per = 0;            % Periodenzähler
242         n_start = 1;
243
244     % Signal über viele Perioden auswerten, Periodendauer und Position der
245     % Nulldurchgänge (lezter Wert einer Periode) speichern
246         for n=1:(length(ref_sig_sim)-1000)
247             if (sign(ref_sig_sim(n)) ~= sign(ref_sig_sim(n+1)));
248                 counter = counter+1;
249                 if counter ==  2
```

```
250                     n_end = n;
251                     per = per +1;
252                     T(per) = n_end - n_start;  % Ermittlung der Periodendauer einer Periode
253                     nulld(per) = n_end;          %Ermittlung der Position eines Nulldurchgangs
254                     counter = 0;
255                     n_start = n+1;
256                 end
257             end
258         end
259
260         M = 128;  %Länge des Signals, das 2 Perioden beinhaltet
261
262
263         per_count = 0; % Periodenzähler zum entnehmen der Samples
264
265
266         % Siglavektor für Unterabgetastetes Signal erzeugen
267         data(1,1).measure_rmp.u_diff = zeros(1,M);
268
269
270         if random == 1
271             [y x] = sort(rand(1,64)); % y interessiert nicht (Zufallswerte)
272         elseif random == 0
273             x = 1:64;% Reihenfolge der Aufnahme der Samples (hier der Reihe nach
274         end
275
276
277
278         for sample = 0:M-1
279             %Zufällige Auswahl zwischen minimal und maximalgrenze
280             t_um = round(tm_min + (tm_max-tm_min).*rand(1,1));
281             %Position des verwendeten Samples bestimmen( wenn per_count zu
282             %groß, beginne bei 1. Periode erneut mit dem Zählen
283             % die 2 stammt von /(M/2)
284             pos = fix(T(mod(per_count,per-1)+1)* 2 *  x(mod(sample,M/2)+1) /M);
285             if sample < 64
286                 % 1. Periode aufnehmen
287                 data(1,1).measure_rmp.u_diff(x(mod(sample,M/2)+1)) = ...
288                             s(nulld(mod(per_count,per-1)+1) + pos);
289             else
290                 % 2. Periode aufnehmen
291                 data(1,1).measure_rmp.u_diff(64 + x(mod(sample,M/2)+1)) =...
292                             s(nulld(mod(per_count,per-1)+1) + pos);
293             end
294             per_count = per_count + t_um; % Nummer der Periode um tum erhöhen
295         end
296
297         Samples  = round(data(1,1).measure_rmp.u_diff .* (2^12/2.5));
298
299          % Spektrum des Signals errechnen
300          S_sample = fft(Samples);
301
302          data(1,1).measure_rmp.periodes = 2;
303
304          % Fourierkoeffizienten abspeichern
305          data(1,1).measure_rmp.coefficients = S_sample(1:2*N+1);
306
307          % Leistung der berücksichtigten Schwingungen ( Verwendung 2 Variante)
308          data(1,1).measure_rmp.P_harm  = 2 * 2.5/((L-1) *2^12) *...
309                                          (sum(abs(S_sample(2:2 * N+1)).^2));
310
311          % Leistung nicht berücksichtigte Schwingungen (Verwendung 2 Variante)
312          data(1,1).measure_rmp.P_ob  =  2 * 2.5/((L -1) *2^12) * ...
313                                        sum(abs(S_sample(2* N+ 2:fix(L/2)-2)).^2);
314
315         l = l +1;
316         clear('Samples', 'T', 'nulld'); % Größen Löschen, da die Anzahl der Perioden variiert
317     end
318     clear measure_scope                         % clean up workspace
319     clear measure_demo
320 end
321
322 %% Neue Datenreihe abspeichern
323
324 % Datenreihe unter anderem Namen abspeichern
325 old_name = list_of_files(1,:);                                  %Dateinamen in String kopieren
326  s = findstr(old_name,'_part');                                 %nach _part suchen
327  file = strcat(old_name(1:s-1), '_rmp_sim_for_simulation.mat'); %neuen  Dateinamen erstellen
328
329 save(file, 'data', 'parameters', 'simulation_parameters');
```

demonstrator_simulation

Listing F.3:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Programm zum durchführen einer Simulation
%
% Eingaben
% -Parameter Datei (Eingabeaufforderung)
%
%
% Version 1.0
%
%
% Datei:          demonstrator_simulation.m
%
% erstellt von: Lennart Koch
% erstellt am:  25.01.2010
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% Arbeitsplatz bereinigen
close all;
clear all;

% Ausgabefenster bereinigen
clc;

% Parameterdatei laden
    pwd;
%     Datei in  Auswahl Dialogbox auswählen und dann laden
    [FileName,PathName] = uigetfile('*txt','Bitte eine Messreihe auswählen');

    if isequal(FileName, 0)
        error('No file selected');
    else
        disp(['selected file: ' FileName]);
    end

   FileNameComp=fullfile(PathName, FileName);

   % Name der Messung = Dateiname ohne Endung

   name = FileName(1:end-4);

   demonstrator_algorithm(name, FileNameComp); %Simulation ausführen
```

analyse_input_signal

Listing F.4:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Analyse des aufgenommenem Ausgangssginal des Sensors mit dem Demontrator
% werden soll. Darstellung der Ergebnisse als Kreisdiagramm.
% Eingabe:
% Schrittweite, in der Analyse des Eingangssignal durchgeführt wird [mm]
%
%
% Version 1.0
%
% Datei:          analyse_input_signal.m
%
% erstellt von: Lennart Koch
% erstellt am:  16.12.2009
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% Arbeitsplatz bereinigen
clear all;
close all;
clc; %Ausgabefenster löschen

%% Datei auswählen und laden
cd('D:/simulation_folder');
pwd;
    %Datei in  Auswahl Dialogbox auswählen und dann laden
    [FileName,PathName] = uigetfile('*.sim.mat','Bitte eine Messreihe auswählen');

    if isequal(FileName, 0)
        error('No file selected');
    else
        disp(['selected file: ' FileName]);
    end

    FileNameComp=fullfile(PathName, FileName);

    load(FileNameComp);%gespeicherte Variablen laden


LM = size(s_fix,1);
L = size(s_fix,2); % Länge des Eingangssignals
n = 1:L;

k = -1/2:1/L:1/2-1/L; % Vektor zur Darstellung der Frequenzachse

d_a = input('Bitte die Distanz zwischen zwei Messpunkten , an denen das Eingangssignal 
    dargestellt werden soll , eingeben [mm]: ');
while (d_a <parameters.stepsize) || (d_a > 5) %ab 5mm keine sinnvolle Analyse mehr möglich
    tm_max = input('Fehler: maximal bei jeder Periode möglich');
end

m = round(d_a/ parameters.stepsize);


```

```
59 for i = 1:m:LM                                          % für alle aufgen. Messdaten mit
       Schrittw. N
60     s= s_fix.double(i,:)/ max(s_fix.double(i,:));       % normiertes Signal erzeugen
61     S1 = fft(s) / length(s_fix);                        % Spektrum Eingangssignal
62     S1_app = S1;
63     S1_app(7:end-5) = 0;                                % 5 Harmonische berücksichtigen
64     s1_app = ifft(S1_app) * length(s_fix) ;             % app. Signal im zeitbereich bestimmen
65
66     %Dartellung
67     h = figure('Name', ['Analyse des Eingangssignals bei ' num2str(distance(i)) 'mm']);
68     orient landscape;
69     subplot(2,1,1)
70     plot(n,s, n, s1_app);
71     ylabel('s(n) \rightarrow');
72     xlabel('n \rightarrow');
73     legend(['app. signal with ' num2str(N_harm) ' coeff.'], 'app. signal with 5 coeff.');
74     title(['input signal at d = ' num2str(distance(i)) 'mm']);
75     grid;
76
77     subplot(2,1,2)
78     stem(k, db(fftshift(S1)));
79     ylabel('S(f) \rightarrow');
80     xlabel('f/f_s \rightarrow');
81     title(['spectrum of input signal at distance d = ' num2str(distance(i)) ' mm'] );
82     grid;
83     %Speichern zusammen mit den anderen Plots dieser Simulation
84     filename = [PathName '/Images/input_signal_' num2str(distance(i)) 'mm'];
85     filename = strrep(filename,'.',',');
86       saveas(h, filename, 'pdf');
87       saveas(h, filename, 'fig');
88       saveas(h, filename, 'jpg');
89 end
```

zufaellig_nichtzufaellig

Listing F.5:

```
% Vergelich zufällige Abtastreihenfolge, nicht zufällige Abtstreihenfolge

close all;
clear all;

% Ausgabefenster bereinigen
clc;

ANALYSE = 1;        %1: Vergleich zufällige, nicht zufällige Abtastreihenfolge
                    %2: Vergelich mit und ohne Radunrundlauf

%% Einlesen der Ergebnisse

% Messreihe zufällige Reihenfolge laden
     pwd;
%     Datei in  Auswahl Dialogbox auswählen und dann laden
if ANALYSE == 1
    [FileName PathName] = uigetfile('*sim.mat','Bitte Simulationsergebnisse mit zufälliger Abtastreihenfolge laden');
elseif ANALYSE == 2
    [FileName PathName] = uigetfile('*sim.mat','Simulationsdaten mit beseitigtem Unrundlauf laden');
end

    if isequal(FileName, 0)
        error('No file selected');
    else
        disp(['selected file: ' FileName]);
    end

    FileNameComp=fullfile(PathName, FileName);

    load(FileNameComp);
    % Werte sichern

    dist1 = distance
    THD1_new_fix = THD_new.double;
    THD1_new_float = THD_new_mat;

    THD1_app_fix = THD_calc.double;
    THD1_app_float = THD5_mat;


    % Messreihe nicht zufällige Reihenfolge laden
     pwd;
%     Datei in  Auswahl Dialogbox auswählen und dann laden
     if ANALYSE == 1
        [FileName PathName] = uigetfile('*sim.mat','Bitte Simulationsergebnisse mit nicht zufälliger Abtastreihenfolge laden');
     elseif ANALYSE == 2
        [FileName PathName] = uigetfile('*sim.mat','Simulationsdaten mit Unrund laufendem Rad laden');
     end

    if isequal(FileName, 0)
        error('No file selected');
    else
        disp(['selected file: ' FileName]);
    end
```

```
56
57      FileNameComp=fullfile(PathName, FileName);
58
59      load(FileNameComp);
60
61      THD2_new_fix = THD_new.double;
62      THD2_new_float = THD_new_mat;
63
64      THD2_app_fix = THD_calc.double;
65      THD2_app_float = THD5_mat;
66
67      %% Ausgabe der Ergebnisse
68
69  if ANALYSE ==  1
70      leg_txt1 = 'zufällige Abtastung';
71      leg_txt2 = 'nicht zufällige Abtastung';
72  elseif ANALYSE == 2
73      leg_txt1 = 'Radunrundlauf nicht beseitigt';
74      leg_txt2 = 'Radunrundlauf beseitigt';
75  end
76
77      %------------------ Abweichung NIHD Verfahren darstellen ----------------
78
79
80      h = figure('Name','HDI.method hd');
81      set(h,'PaperPositionMode','manual');
82      set(h, 'PaperUnits', 'centimeters');
83      set(h, 'PaperType', 'A4');
84      orient landscape
85
86      subplot(2,1,1)
87      plot(distance, THD2_new_fix, dist1, THD1_new_fix);
88      grid;
89      %legend(, );
90      legend(leg_txt1, leg_txt2);
91      title('calculated with fixpoint arithmetic');
92      xlabel('distance [mm]');
93      ylabel('THD[%]');
94
95      subplot(2,1,2)
96      plot(distance, THD2_new_float, dist1, THD1_new_float);
97      grid;
98      legend(leg_txt1, leg_txt2);
99      title('calculated with floatingpoint arithmetic');
100     xlabel('distance [mm]');
101     ylabel('THD[%]');
102
103     saveas(h, 'noise_included_HD', 'fig');
104     saveas(h, 'noise_included_HD', 'pdf');
105     saveas(h, 'noise_included_HD', 'jpg');
106
107     %------------------ Abweichung hd5 Verfahren darstellen ----------------
108
109     h = figure('Name','HD5');
110     set(h,'PaperPositionMode','manual');
111     set(h, 'PaperUnits', 'centimeters');
112     set(h, 'PaperType', 'A4');
113     orient landscape
114
115     subplot(2,1,1)
116     plot(distance, THD2_app_fix, dist1, THD1_app_fix);
117     grid;
118     legend(leg_txt1, leg_txt2);
```

```
119         title('calculated␣with␣fixpoint␣arithmetic');
120         xlabel('distance␣[mm]');
121         ylabel('THD[%]');
122
123         subplot(2,1,2)
124         plot(distance, THD2_app_float, dist1, THD1_app_float);
125         grid;
126         legend(leg_txt1, leg_txt2);
127         title('calculated␣with␣floatingpoint␣arithmetic');
128         xlabel('distance␣[mm]');
129         ylabel('THD[%]');
130
131           saveas(h, 'approximated_THD', 'fig');
132         saveas(h, 'approximated_THD', 'pdf');
133         saveas(h, 'approximated_THD', 'jpg');
```

drehung

Listing F.6:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Ermittlunng der Position des Encoderrades, an der ein Wert entnommen
% werden soll. Darstellung der Ergebnisse als Kreisdiagramm.
%
%
% Version 1.0
%
% Datei:          drehung.m
%
% erstellt von: Lennart Koch
% erstellt am:  08.02.2010
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%

N = 64;             %Anzahl der Samples
phi = zeros(1,N);  %Winkel jeden Abtastpunkt
r= zeros(1,N);      % Anzahl der Umdrehungen

Z = 50;             %Zähne des Encoderrades
tw = 7;             %Anzahl der Pulse die verstreichen, bis nächster Wert abgetastet wird


for i = 1:N
    phi(i) = 2*pi/Z * (i-1) * tw;    % Drehwinkel für jedes Sample ausrechnen
    r(i)= fix(phi(i)/(2*pi)) +1;     % Radumdrehung
end

polar(phi,r,'*');
xlabel('rotation angle')
ylabel('revolution');
grid;
legend('sampling point');
```

unterabtast

Listing F.7:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Das Programm soll das Auftreten der 7 und 8 Harmonischen zeigen.
% Dazu wird die Unterabtastung des Radmessplatzes auf ein bekanntes Signal,
% dessen idealer Klirrfator 0 ist, angewendet
%
%
% Version 1.1
%
%
% Datei:        unterabtast.m
%
% erstellt von: Lennart Koch
% erstellt am:  17.03.2010
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clear all;


Z = 50; % Anzahl der Zähne Encoderrad

S = 730; % Samples pro Periode des generierten Signals

wT = 0:2*pi/S:600*(2* pi -1/S); %Erstellung Zeitvektor

%Definition des Eingangssignals
s = 1.25 + cos(wT) .* (0.7 + 0.025 *cos(wT/50)); % Simulation des Sensorsignals

h= figure('Name', 'simulated␣Output␣signal␣from␣sensor');
orient landscape;
plot(s);   % Ausgabe Ausgangssignal Sensor
grid;
xlabel('n');
title('simulated␣sensor␣output␣signal␣over␣time');
% Bild speichern
saveas(h, 'signal', 'pdf');
saveas(h, 'signal', 'fig');
saveas(h, 'signal', 'jpg');

% Nulldurchgangserkennung

ref = sign(s - mean(s));
counter = 0;
per = 1;

% Periodendauer messen
  for n=1:(length(ref)-1)
    if (sign(ref(n)) ~= sign(ref(n+1)));
        counter = counter+1;
        if counter == 1
            n_start = n;
        end
        if counter == 2 * per + 1
            n_end = n;
            nulld(per) = n_end;% Nulldurchgangserkennung
```

```
60              n_start = n;
61              per = per +1;
62          end
63      end
64    end
65    % Aufnahme des Signals noch der ersten Periode mit 64 Abtastpunkten pro
66    % Periode
67
68    N = 64;
69
70    % Anzahl der Perioden, die verstreichen soll, ohne dass 1 Samplewerte
71    % entnommen wird (Hardware ca. 3 - 4) -> Umsetzzeit in Perioden
72
73
74    per_count = 1; % Periodenzähler zum entnehmen der Samples
75
76    s_unter = zeros(1,N);
77    % Simulation über 2 Perioden
78    for sample = 1:N
79        t_um = round(6 + (7-6) * rand(1));
80        %Position des verwendeten Samples bestimmen
81        pos = fix(S *sample /N);
82        per_count = per_count + t_um;
83       s_unter(sample) = s(nulld(per_count) + pos);
84    end
85
86   % Skallierung des Signal zwischen 0 und 1
87    s_unter = s_unter / max(s_unter);
88   % Darstellung Signal und Spektrum von unterabgetastem Signal
89
90  h = figure('Name', 'subsampled signal over one periode');
91  orient landscape;
92  plot(s_unter);   % Ausgabe Ausgangssignal Sensor
93  grid;
94  xlabel('n');
95  ylabel('|s_{unter}|');
96  title('subsampled signal (one sample each 6-7 periodes)');
97  % Bild speichern
98  saveas(h, 'subsampled_signal', 'pdf');
99  saveas(h, 'subsampled_signal', 'fig');
100 saveas(h, 'subsampled_signal', 'jpg');
101
102
103 h = figure('Name','spectrum of subsampled signal');
104 orient landscape;
105 df = 1/length(s_unter);
106 f = -0.5 : df : 0.5 - df;
107 stem(f, db(df * abs(fftshift(fft(s_unter)))));   % Spektrum unterabgetastetes Signal
108 grid;
109 xlabel('f/f_s');
110 ylabel('|s_{unter}|_{dB}');
111 title('spectrum of subsampled signal');
112 % Bild speichern
113 saveas(h, 'spectrum_subsampled_signal', 'pdf');
114 saveas(h, 'spectrum_subsampled_signal', 'fig');
115 saveas(h, 'spectrum_subsampled_signal', 'jpg');
```

rmp_scope_analyze

Listing F.8:

```
1  %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2  %
3  % Änderung:      Lennnart Koch, 04.01.2010
4  %
5  % Beschreibung: Analyse der Rohdaten ohne Filter
6  %                - FFT von einer Periode und Koeffizienten plotten
7  %
8  %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
9
10 clear all
11 close all
12 clc
13
14 disp('-- Start rmp_stepper_scope_record_analyze_tekdata.m --');
15
16 %% ---- Datein auswählen, Variablen vorbereiten ------------------------------
17 % get directory
18 DirName = uigetdir(pwd,'Choose directory with data-files');
19 cd(DirName);
20 list_of_files = ls('*.rmp.mat');
21 number_of_files = size(list_of_files, 1);
22
23 load(list_of_files(1,:));           % eine Datei laden zum Lesen Parameter
24 filename = list_of_files(1,:);
25
26 stepsize     = parameters.stepsize;      % Schrittweite in mm
27 startpoint   = parameters.startpoint;    % Anfangspunkt in mm
28 endpoint     = parameters.endpoint;      % Endpunkt in mm
29 fs           = parameters.samplerate;    % sampling frequenz
30 amp_setting = parameters.amp_setting;   % Verstärkerumschaltung
31
32 abstand = startpoint : stepsize : endpoint;
33 abstand = abstand';
34
35 meas_range  = ((endpoint - startpoint) / stepsize)+1;
36
37 cnt = 0;   % Zählvariable für bisher gelesene Messreihen
38
39 Z = 50;    % Anzahl der Zähne des Encoderrades
40
41
42 %% ---- Dateien durchlaufen --------------------------------------------------
43
44 disp('---------- START EINLESEN ----------');
45 data_inc = 1;
46
47 for i=1:number_of_files                % datafile seperated in n parts
48
49                                        % load one datafile
50     disp(['loading ' list_of_files(i,:)]);
51     load(list_of_files(i,:));
52
53                                        % Strukturen auslesen
54     for j = 1:length(measure_scope)
55
56         if mod(cnt, 20) == 0
57             disp(['calculating ' num2str(cnt) ' from ' num2str(meas_range)]);
58
59             ref  = measure_scope(j).ref_y';     % Refenzsensor     Channel 4
```

```
60              c1    = measure_scope(j).u_diff_y';  % Differenzsignal Channel 1
61
62
63              num_samples = length(c1);           % number of samples
64              t_max    = num_samples/fs;          % time range of data
65
66
67              t = 0:1/fs:t_max- 1/fs;
68
69  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Eigene Arbeit %%%%%%%%%%%%%%%%%%%%%%%%%%
70              h = figure('Name', ['signal distance d =' num2str(abstand(cnt+1)) ' mm']);
71              orient landscape;
72              subplot(2,2,1);
73              plot(t, c1);
74              grid;
75              title(['Sensor Output signal measured with Tektronix scope at distance d = '
                    num2str(abstand(cnt+1)) ' mm']);
76              xlabel('t [s]');
77              ylabel('U_{diff} [V]');
78
79              %% Auszählen der Radfrequenz
80              %  = Raddrehzahl * Zähne des Zahnrades an der Referenz.
81              %  Signal muss AC gekoppelt sein !
82              ref=ref-mean(ref);
83              counter=0;
84              per_count = 1;  % Periodenzähler
85              n_old = 1; %Speichern des Anfangs der Periode
86              for n=1:(length(ref)-1)
87                  if (sign(ref(n)) ~= sign(ref(n+1)));
88                      counter = counter+1;
89                      if counter == 2;
90                          if n_old ~= 1
91                              sig = c1(n_old:n-1);%Eine Periode des Singal speichern
92                              C1 = abs(fft(sig));% Spektrum errechnen
93                              Koeff(per_count,:) = C1(2:9); %Fourierkoeffizienten speichern
94
95                              klirrf(per_count) = THD(C1); % THD errechnen
96                              per_count = per_count +1;    % Periodenzähler inkremetieren
97                          else
98                              t_start= n / fs;%Startzeitpunkt errechnen für Zeitsignal
99                          end
100                         counter = 0;
101                         n_old = n;
102                     end
103                 end
104             end
105
106             phi = zeros(1,per_count-2); % Drehwinkel errechnen
107             for m =1:per_count-1
108                 phi(m) = 2*pi/Z * (m-1);
109             end
110
111             %Mit Mikrocontroller ermittelter Klirrfaktor
112             hd = measure_demo(j).hd_abs * ones(1,length(phi));
113
114             subplot(2,2,2);
115             t_vec = linspace(t_start, t_max, length(Koeff)); %Zeitachse für Darstellung im
                    Zeitbereich
116             plot(t_vec, Koeff);
117             legend('1st harmonic', '2nd harmonic', '3rd harmonic' ,'4th harmonic','5th
                    harmonic','location', 'Best');
118             grid;
119             xlabel('t [s]');
```

```
            title(['magnitudes of harmonics at distance d = ' num2str(abstand(cnt+1)) ' mm']);

            % Kreisplots hinzufügen
             subplot(2,2,4);
             polar(phi, Koeff(:,1)','b');
             p21 = gca; %nötig, damit richte Darstellung der Legende
             hold all;
             polar(phi, Koeff(:,2)','g');
             hold all;
             polar(phi, Koeff(:,3)','r');
             legend(p21,'1st harm.', '2nd harm.', '3rd harm.', 'location', 'Best');
             grid;
             title(['value of coefficients over angle at distance d = ' num2str(abstand(cnt+1)) ' mm']);

             subplot(2,2,3);
             polar(phi, klirrf,'b');
             p31 = gca; %nötig, damit richte Darstellung der Legende
             hold all;
             polar(phi, hd,'r');
             legend(p31,'hd of per.','hd by demonstrator','location', 'Best');
             grid;
             title(['THD over angle at distance d = ' num2str(abstand(cnt+1)) ' mm']);

             saveas(h, ['analyzed_d=' num2str(abstand(cnt+1)) 'mm'], 'epsc');
             saveas(h, ['analyzed_d=' num2str(abstand(cnt+1)) 'mm'], 'pdf');
             saveas(h, ['analyzed_d=' num2str(abstand(cnt+1)) 'mm'], 'fig');
             saveas(h, ['analyzed_d=' num2str(abstand(cnt+1)) 'mm'], 'jpg');
            clear Koeff;
            clear test;
            clear klirrf;
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        end
        cnt = cnt +1; %Zähler hochzählen
    end % Ende Schleife zum Einlesen der Datenstrukturen aus File

    %cleanup workspace
    clear measure_scope
    clear measure_demo
%
 end % Ende Schleife zum Einlesen der Files

disp('--- End rmp_stepper_scope_record_analyze_tekdata.m ---');
```

Wurzeltab

Listing F.9:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Mit diesm Skript soll das changing LUT Verfahren erprobt werden, sowie
% die Wurzelfunktion nach dem Jegenhorst Verfahren mit der erfordelichen
% Anzalh an Zwischenwerten dargestellt werden
%

% Version 1.0
%
%
% Datei:          Wurzeltab.m
%
% erstellt von: Lennart Koch
% erstellt am:  19.11.2009
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;

% -------------------- andere Variante ----------------------------------

% Funktion Stufe 1 maximale Schrittweite um im unteren Bereich 1% Auflösung
% zu gewährleisten
x_anders = 0:1:1024-1;

Func = sqrt(100^2*2^-3/2^10 * x_anders);    %Wurzelfunktion mit 1% Auflösung

% Darstellung der Wurzelfunktion
figure(9)
plot(x_anders, Func);
grid;
xlabel('n')
ylabel('THD [%]');
title('Wurzelfunktion mit 1024 Werten');


% Ableitung Bestimmen um Steigung der Funktion zu ermitteln
len_x = length(x_anders);
valdiff = zeros(1, len_x);
for i = 1:len_x -1
    %Steigung der Funktion zwischen zwei Punkten
    valdiff(i) = Func(i + 1) - Func(i);
end
% Schiebfaktoren ausrechnen um verschiedene Stufen zu realisieren
shift_fac = fix(log2(1./ valdiff));% mögliche Schiebefaktoren errechnen

figure(10)
subplot(2,1,1)
plot(valdiff);
xlabel('n')
ylabel('Steigung');
title('Steigung der implementierten Wurzelfunktion');
grid;
subplot(2,1,2)
stairs(shift_fac);
```

```
60 grid
61 xlabel('log_2(n)')
62 ylabel('shift␣factor');
63 title('maximal␣mögliche␣Schiebefaktoren␣zur␣Darstellung␣der␣Funktion');
64
65 % Testen der 4 ermittelten Stufen
66
67 % Implementierung 1 Stufe: 8 Werte
68 x_imp1 = 0:1:8-1;
69 Func_imp1 = sqrt(100^2*2^-3/2^10 * x_imp1);
70
71 % Implementierung 2. Stufe: 8 Werte
72 % start = 7
73 %wertpos 111(7) >> 2 = 1 % Position an der vorheriger max.Wert vorkommt
74 x_imp2 = 7:4:8*4 +4 -1; %wg Scheibefaktor 2
75 Func_imp2 = sqrt(100^2*2^-3/2^10 * x_imp2);
76
77 % Implemtierung 3. Stufe: 16 Werte
78 % start = 7 + 1 >> 3  = 15
79 %wertpos 10 0100 (35) >> 3 = 100= 4 % Position an der vorheriger max.Wert vorkommt
80 x_imp3 = 15:8:16*8+8-1;
81
82 Func_imp3 = sqrt(100^2 * 2^-3/2^10* x_imp3);
83
84 %Impelementierung 4. Stufe = 16 Werte
85 % start = 15 + 1 << 4 = 31
86 %wertpos= 1000 0011 >> 4 = 100= 4 % Position an der vorheriger max.Wert vorkommt
87 x_imp4 = 31:16:16*32+31-1;
88 Func_imp4 = sqrt(100^2 * 2^-3/2^10 * x_imp4);
89
90
91 %Implementierung 5. Stufe = 32 Werte
92 % start = 31 + 1 << 5 = 63
93 %wertpos= 1 0000 0111 >> 5 = 100= 8 % Position an der vorheriger max.Wert vorkommt
94 x_imp5 = 63:32:32*31+ 63 -1;
95 Func_imp5 = sqrt(100^2 * 2^-3/2^10 * x_imp5);
96 error = find(Func_imp5 > 31); % maximal Wert 31%
97 Func_imp5(error) = 31;
```

Wurzeltab_PCM_8Stufen

Listing F.10:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Skript zur Berechnung einer Wurzeltabelle mit PCM Codierung mit 8 Stufen.
% Es sind 10 Bit für das Ergebnis erforderlich
%
%
% Version 1.1
%
%
% Datei:        Wurzeltab_PCM_8Stufen.m
%
% erstellt von: Lennart Koch
% erstellt am:  16.12.2009
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all;
close all;

% Funktion Stufe 1 maximale Schrittweite um im unteren Bereich 1% Auflösung zu gewährleisten
%x_anders = 0:2^-13:2^-3 - 2^-13; %Länge 1024 entspricht 20 Bit Codierung
x_anders = 0:1:1023; %Länge 1024 entspricht 10 Bit Ergebnis

Func = sqrt(100^2 *2^-3/2^10 * x_anders); % Originalfunktion

% Ausgabe Originalfunktion
figure(1)
plot(x_anders, Func);
grid;
xlabel('n')
ylabel('THD [%]');
title('Wurzelfunktion mit 1024 Werten');

xu = zeros(1,8);
xo = zeros(1,8);


%% Ober- und Untergerenzen der Segmente festlegen
% 1 Stufe Untergrenze = 0;
 xu(1) = 0;

% 1 Stufe Obergrenze
xo(1) = 1024/128 -1;

% 2. Stufe Untergrenze
xu(2) = 1024/128;

% 2. Stufe  Obergrenze
xo(2) = 1024/64-1;

% 3. Stufe Untergrenze
xu(3) = 1024/64;

% 3. Stufe  Obergrenze
xo(3) = 1024/32-1;
```

```
59
60 % 4. Stufe Untergrenze
61 xu(4) = 1024/32;
62
63 % 3. Stufe  Obergrenze
64 xo(4) = 1024/16-1;
65
66 % 5. Stufe Untergrenze
67 xu(5) = 1024/16;
68
69 % 5. Stufe  Obergrenze
70 xo(5) = 1024/8-1;
71
72 % 6. Stufe Untergrenze
73 xu(6) = 1024/8;
74
75 % 6. Stufe  Obergrenze
76 xo(6) = 1024/4-1;
77
78 % 7. Stufe Untergrenze
79 xu(7) = 1024/4;
80
81 % 7. Stufe  Obergrenze
82 xo(7) = 1024/2-1;
83
84 % 8. Stufe Untergrenze
85 xu(8) = 1024/2;
86
87 % 8. Stufe Obergrenze
88 xo(8) = 1024-1;
89
90 Tabgr_o = sqrt(100^2 *0.125/1024 *xo); %Funktionswerte der Obergrenzen
91 Tabgr_u = sqrt(100^2 *0.125/1024 *xu); %Funktionswerte der Untergrenzn
92 Tabelle = zeros(8,8);
93
94 %Neue Wurzeltabelle mit Daten füllen
95 for i = 1:9
96     Tabelle(i,:) = round(linspace(Tabgr_u(i), Tabgr_o(i),8));
97 end
98
99 %Wurzeltabelle zur Verwendung in Simulationsprogramm speichern
100 save('SQRT_LUT_PCM_8STUFEN', 'Tabelle');
101
102 %grafische Darstellung der Wurzelfunktion
103 figure(3)
104 plot(xo, Tabgr_o, xu, Tabgr_u);
105 grid;
```

Wurzeltab_PCM_9Stufen

Listing F.11:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktionen:
% Skript zur Berechnung einer Wurzeltabelle mit PCM Codierung mit 9 Stufen.
% Es sind 11 Bit für das Ergebnis erforderlich
%
%
% Version 1.0
%
%
% Datei:          Wurzeltab_PCM_9Stufen.m
%
% erstellt von: Lennart Koch
% erstellt am:  16.12.2009
%
% Änderungen:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
close all;


% Funktion Stufe 1 maximale Schrittweite um im unteren Bereich 1% Auflösung zu
    gewährleisten
%x_anders = 0:2^-13:2^-3 - 2^-13; %Länge 1024 entspricht 20 Bit Codierung
x_anders = 0:1:1023; %Länge 1024 entspricht 10 Bit Codierung

Func = sqrt(100^2 *2^-3/2^10 * x_anders); % Originalfunktion

% Ausgabe Originalfunktion
figure(1)
plot(x_anders, Func);
grid;
xlabel('n')
ylabel('THD␣[%]');
title('Wurzelfunktion␣mit␣1024␣Werten');


xu = zeros(1,9);
xo = zeros(1,9);


%% Ober- und Untergerenzen der Segmente festlegen

% 1 Stufe Untergrenze = 0;
 xu(1) = 0;

% 1 Stufe Obergrenze
xo(1) = 1024/256 -1;

% 2 Stufe Obergrenze = 1024/256;
 xu(2) = 1024/256;

% 2 Stufe Obergrenze
xo(2) = 1024/128 -1;

% 3. Stufe Untergrenze
xu(3) = 1024/128;

```

```
59 % 3. Stufe Obergrenze
60 xo(3) = 1024/64-1;
61
62 % 4. Stufe Untergrenze
63 xu(4) = 1024/64;
64
65 % 4. Stufe Obergrenze
66 xo(4) = 1024/32-1;
67
68 % 5. Stufe Untergrenze
69 xu(5) = 1024/32;
70
71 % 5. Stufe Obergrenze
72 xo(5) = 1024/16-1;
73
74 % 6. Stufe Untergrenze
75 xu(6) = 1024/16;
76
77 % 6. Stufe Obergrenze
78 xo(6) = 1024/8-1;
79
80 % 7. Stufe Untergrenze
81 xu(7) = 1024/8;
82
83 % 7. Stufe  Obergrenze
84 xo(7) = 1024/4-1;
85
86 % 8. Stufe Untergrenze
87 xu(8) = 1024/4;
88
89 % 9. Stufe  Obergrenze
90 xo(8) = 1024/2-1;
91
92 % 8. Stufe Untergrenze
93 xu(9) = 1024/2;
94
95 % 9. Stufe Obergrenze
96 xo(9) = 1024-1;
97
98 Tabgr_o = sqrt(100^2 *0.125/1024 *xo);  %Funktionswerte der Obergrenzen
99
100 Tabgr_u = sqrt(100^2 *0.125/1024 *xu);  %Funktionswerte der Untergrenzen
101
102 Tabelle = zeros(9,8);
103
104 for i = 1:9
105     Tabelle(i,:) = round(linspace(Tabgr_u(i), Tabgr_o(i),8));
106 end
107
108 %Wurzeltabelle zur Verwendung in Simulationsprogramm speichern
109 save('SQRT_LUT_PCM_9STUFEN', 'Tabelle');
110
111 %grafische Darstellung der Wurzelfunktion
112 figure(3)
113 plot(xo, Tabgr_o, xu, Tabgr_u);
114 grid;
```

F.1.2. Funktionen

calc_function

Listing F.12:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktion zur Berechnung einer approximiereten Funktion mit variabler
% Anzahl von Samplewerten
%
%
%
% Version 1.0
%
% Datei:              calc_function.m
%
% erstellt von: Lennart Koch
% erstellt am:  20.11.2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [s_fix gain] = calc_function(data, prop, S, option)
% erzeugen einer Signalmatrix für Funktion demontrator_alogrithm in
% Fixpoint Arithmetik
    % Eingabe:
    % data : Datenstruktur
    % prop : Fixpoint Eigenschaften und Datentypen
    % S    : Anzahl der Samplewerte
    %
    % Ausgabe:
    % s_fix: Samples im Fixpoint-Datentyp tADC, variable Wortbreite


    wT = (0:2*pi:(S -1) * 2* pi) / S; %Erstellung Zeitvektor
    LM = length(data);               % Anzahl der Messungen in der Messreihe
    gain = zeros(1,LM);
    L = length(data(1,1).measure_rmp.u_diff);


    Nb = prop.tADC.WordLength; % immer volle Aussteuerung

    diff_bit = 12 - Nb;          %Fourierkoeffizienten sind mit 12 Bit ausgesteuert
    diff_faktor = 2^diff_bit;    % wenn weniger Bit Divisionsfaktor ausrechnen um
                                 % Überlauf zu vermeiden


    switch(option)
        case 'demo'% Berechnung für Demonstrator Signal durchführen
            disp('----- calc_function: You`ve selecteded option "demo" -----');
            s = zeros(LM,S);
            for i = 1:LM
                %Gleichanteil vor Berechnung dees Signals in Vektor
                %übernehmen
                s(i,:) = data(1,i).measure_rmp.coefficients(1);
                for n = 1:length(data(1,i).measure_rmp.coefficients) - 1
                    %Phasenwinkel bestimmen
                    phi = atan2(imag(data(1,i).measure_rmp.coefficients(n+1)),...
                                real(data(1,i).measure_rmp.coefficients(n+1)));
                    %approximiertes Signal berechnen
                    s(i,:) = s(i,:) + 2 * abs(data(1,i).measure_rmp.coefficients(n+1)) *...
                             cos(1/data(1,i).measure_rmp.periodes * n * wT + phi);
                end
```

```
56                  gain(i) = data(1,i).gain;  %Verstärkungsfaktor in Vektor kopieren
57              end
58              s = round(s/L);
59              s = s / diff_faktor;
60
61              s_fix(:) = fi(s, prop.tADC,prop.Properties); %Festkomma Variable
62
63          case 'scope'
64              disp('---- calc_function: You've selecteded option "scope" ----');
65              s_scope = zeros(LM,S);
66              periodes = data(1,1).measure_scope.periodes;
67
68              for i = 1:LM
69                  L_scope = length(data(1,i).measure_scope.u_diff);
70                  %Gleichanteil vor Berechnung dees Signals in Vektor
71                  %übernehmen
72                  s_scope(i,:) = data(1,i).measure_scope.coefficients(1);
73                  for n = 1:length(data(1,i).measure_scope.coefficients) - 1
74                      %Phasenwinkel bestimmen
75                      phi = atan2(imag(data(1,i).measure_scope.coefficients(n+1)),...
76                                  real(data(1,i).measure_scope.coefficients(n+1)));
77                      %approximiertes Signal berechnen
78                      s_scope(i,:) = s_scope(i,:) + ...
79                          2 * abs(data(1,i).measure_scope.coefficients(n+1)) * cos(1 /
                                periodes * n * wT + phi);
80                      gain(i) = data(1,i).gain; % Verstärkungsfaktor in Vektor kopieren
81                  end
82                  s_scope(i,:) = round(s_scope(i,:) / L_scope);
83              end
84
85              s_scope = s_scope / diff_faktor;
86              s_fix(:) = fi(s_scope, prop.tADC,prop.Properties);%Festkomma Variable
87          otherwise
88              error('You have selected wrong option');
89      end
90  end
```

THD_fix_int

Listing F.13:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Funktion berechnet den THD (Total Harmonic Distortion)
%Berechnung der Wurzelfunktion mit Hilfe einer Tabelle
%
%
%
% Version 1.0
%
% Datei:                THD_fix_int.m
%
% erstellt von: Lennart Koch
% erstellt am:   11.02.2010
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function erg = THD_fix_int(abs_K, prop, wordlength, lut)
%Funktion berechnet den THD (Total Harmonic Distortion)
%Berechnung der Wurzelfunktion mit Hilfe einer Tabelle
%
%
%Formel :
%           N                   N
%factor =  sum  abs_K  / sum  abs_K )
%          k = 2                k = 1
%
%
%Eingabe
%   abs_K       : Vektor mit den Absolutwerten der Koeffizienten
%   prop        : Fixpoint Eigenschaften
%   wordlength  : verwendete Wortbreite der Register: long, short, 24Bit
%   lut:        : 1:   original Wurzetabelle von N. Jegenhorst
%                 2:   PCM codiert (9 Stufen)
%                 3:   PCM codiert (8 Stufen)
%                 4:   eigenes Verfahren
%
% Ausgabe
%   THD in [%]



if nargin ~= 4
    error('myApp:argChk', 'Wrong number of input arguments')
end


 % Vereinbarung der Fixpoint Variablen
 factor = fi(0,prop.long, prop.Properties); % geändert 11.02.2010

    switch (wordlength)
        case 'short'   % getestet, funktioniert nicht gut
            sum_ges = fi(0,prop.long, prop.Properties);
            sum_ob = fi(0,prop.long, prop.Properties);  %32 Bit Wortlänge
        case '24Bit'   % getestet, funktioniert nicht gut
            sum_ges = fi(0,prop.long, prop.Properties);
            sum_ob = fi(0,prop.long, prop.Properties);  %32 Bit Wortlänge
        case 'long'    % getestet, funktioniert gut
            sum_ob = fi(0,prop.long64, prop.Properties); %64 Bit Wortlänge
            sum_ges = fi(0,prop.long, prop.Properties); %64 Bit Wortlänge
        otherwise %Wenn ungültige Wortlänge gewählt
            error('THD_fix_int: You have selected wrong wordlength!');
    end
```

```
60
61      L = length(abs_K);
62
63      for i = 2:L
64          sum_ob(:) = sum_ob + abs_K(i); %Oberschwingungen addieren
65      end
66
67          %Betrag der Amplituden mit der Gesamtschwingung
68
69      sum_ges(:) = sum_ob + abs_K(1); % Summe der ersten 5 Harmonischen
70
71
72  % Ausgabe der Ergebnisse zur Kontrolle der Funktion
73            disp(['THD_fix: sum_ob: ' sum_ob.bin]); %Testausgabe
74            disp(['THD_fix: sum_gs: ' sum_ges.bin]);
75            disp(['THD_fix: factor: ' factor.bin]);
76
77
78        if lut == 1
79           ob_shift = 10;
80           sum_ob(:) = bitshift(sum_ob ,ob_shift); %Schieben um 10 Stellen -> Doku
81           factor(:) = divide(prop.long, sum_ob, sum_ges);
82
83           %Erstellen der Tabelle
84           x = 0: 10000 / 1024: (2^8 -1) * 10000 / 1024;   %256 Werte in Tabelle enthalten
85           Wurzel = round(sqrt(x));
86
87           if factor > length(Wurzel);   % Wenn sehr großer Klirrfaktor
88               erg = Wurzel(end);
89           else
90               erg = Wurzel(factor.int +1); %Wert aus Tabelle ermitteln
91           end
92
93        elseif lut == 2
94           ob_shift = 14;       % Schiebefaktor der Oberschwingungen
95           shiftdiff = ob_shift-14; % Shiffaktordifferenz = ob_shift-14
96           Spalte = 0;
97           Zeile = 0;
98
99           sum_ob(:) = bitshift(sum_ob ,ob_shift); %Schieben um 14 Stellen nach links
100          factor(:) = divide(prop.long, sum_ob, sum_ges);
101
102          for bit = 20:29- shiftdiff
103              if factor.bin(bit) == '1'  % wenn Bit gesetzt ist
104                  Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
105                  Zeile = 32- bit -2 -  shiftdiff;
106                  if Zeile > 8   % Für Große THD bleibt Zeile Konstant
107                      Zeile = 8;
108                      Spalte = 7;
109                  end
110              break;
111              end
112          end
113
114          %wenn Zeile und Spalte weiterhin 0: THD sehr klein
115
116          if (Spalte == 0) && Zeile == 0
117              Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
118          end
119
120          % ermittelte Position in der Tabelle anzeigen
121          disp(['Zeile: ' num2str(Zeile)]);
122          disp(['Spalte: ' num2str(Spalte)]);
```

```
123         load('SQRT_LUT_PCM_9STUFEN.mat');
124         erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
125
126     elseif lut == 3
127         ob_shift = 13;      % Schiebefaktor der Oberschwingungen
128         shiftdiff = ob_shift-13; % Shiffaktordifferenz = ob_shift-13
129         Spalte = 0;
130         Zeile = 0;
131
132         sum_ob(:) = bitshift(sum_ob ,ob_shift);
133         factor(:) = divide(prop.long, sum_ob, sum_ges);
134
135         for bit = 20:29- shiftdiff
136             if factor.bin(bit) == '1'  % wenn Bit gesetzt ist
137                 Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
138                 Zeile = 32- bit -2 -  shiftdiff;
139                 if Zeile > 7   % Für sehr großen Klirrfaktor konstanter Wert
140                     Zeile = 7;
141                     Spalte = 7;
142                 end
143                 break;
144             end
145         end
146
147         %wenn Zeile und Spalte weiterhin 0: THD sehr klein
148         if (Spalte == 0) && Zeile == 0
149             Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
150         end
151
152         % ermittelte Position in der Tabelle anzeigen
153         disp(['Zeile: ' num2str(Zeile)]);
154         disp(['Spalte: ' num2str(Spalte)]);
155         load('SQRT_LUT_PCM_8STUFEN.mat');
156         erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
157
158     elseif lut == 4
159          ob_shift = 13;      % Schiebefaktor der Oberschwingungen
160          Wurzel = sqrt_table;      % Wurzeltabellen  erzeugen
161          sum_ob(:) = bitshift(sum_ob ,ob_shift);
162          factor(:) = divide(prop.long, sum_ob, sum_ges);
163
164          if factor >= 7
165             factor(:) = bitshift(factor, -2);  %Ergebnis schieben und in anderer Tabelle
166             if factor >= 7                  % wenn factor immer noch groß
167                 factor(:) = bitshift(factor, -1); %nochmals um ein Bit schieben und inb
                        anderer Tablle suchen
168                 if factor >= 16                  % wenn immer noch groß
169                     factor(:) = bitshift(factor, -1); % nochmals um ein  Bit schieben und
                            in anderer Tabelle nachsehen
170                     if factor >= 32                  % wenn immer noch groß
171                         factor(:) = bitshift(factor, -1); % nochmals um ein  Bit schieben
                                und in anderer Tabelle nachsehen
172                         if factor >= 32                  %wenn immer noch zu groß ->
                                sehr großer THD
173                             factor(:) = 31;              % Maximalwert verwenden
174                             erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle 5
175                         else
176                             erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle 5
177                         end
178                     else
179                         erg = Wurzel.Table4(factor.int); %Ergebnis aus Tabelle 4
180                     end
181                 else
```

```
182                     erg = Wurzel.Table3(factor.int); %Ergebnis aus Tabelle 3
183                 end
184             else
185                 erg = Wurzel.Table2(factor.int); %Ergebnis aus Tabelle 2
186             end
187         else
188             erg = Wurzel.Table1(factor.int +1); %Ergebnis aus Tabelle 1
189         end
190     else
191         error('wrong look-up_table!') % wenn falsche Ziffer angegeben
192     end
193 end
```

DFT_quant_intV3

Listing F.14:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Funktion berechnet den THD (Total Harmonic Distortion)
%Berechnung der Wurzelfunktion mit Hilfe einer Tabelle
%
%
%
% Version 1.0
%
% Datei:              DFT_quant_intV3.m
%
% erstellt von: Lennart Koch
% erstellt am:  02.11.09
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 function [re im absolut] = DFT_quant_int_V3(x_t,prop, Real, Imag, wordlength, N_K)
%DFT_quant_int berechnet eine reduzierte DFT des Signals
% über 1 Periode x_t .Die Auflösung der Samples und der LUT sind in den
% Datentypen tADC und tLUT eingestellt
%
% Eingabe:
%   x_t:        Zahlenfoge des Signals im Zeitbereich
%   N_K:        Anzahl der Koeffizienten die erechnet werden sollen
%   prop:       Struktur mit Benutzerdefinierten Fixpoint Dateitypen(Ergebnis
%               von define_Types.m)
%   Real:       Wertetabelle für den Realteil
%   Imag:       Wertetabelle für den Imaginärteil
%   wordlength: Variablentyp der verwendeten Register 'short', 'long',
%
%
% Ausgabe:
%   re      :   Realteil der Koeffizienten
%   im      :   Imaginärteil der Koeffizienten
%   absolut:    Quadrate der Beträge der Koeffizienten
%            |              |
% Formel: X = 1/sqrt(N)* MAT * x
%            |              |
%             Wird in der Funktion berechnet


N = length(x_t);

 w_LUT = Real.WordLength; % Wortlänge LUT Sinus und Kosinus
 w_ADC = x_t.Wordlength;  % Wortlänge ADC Samples

    if nargin == 5
       K = N; % wenn keine Anzahl der Koeffizienten angegeben ist, volle DFT berechnen
    elseif nargin == 6
       K =  N_K;  %Übergebene Anzahl Fourier Koeffizienten berechnen
    else
       error('myApp:argChk', 'Wrong␣number␣of␣input␣arguments');
    end

%-----------------------------------------------------------------------------
% Es wird in der Simaulation davon ausgangen, dass für jede Multiplikation
% der Hardware Multipizierer des MSP430 verwendet wird, dieser ist z.B im
% 1611 integriert

    accu = fi(0,prop.long, prop.Properties);    % 32 Bit Akku
    OP1 = fi(0,prop.short, prop.Properties);    % Operator1 16 Bit
```

```
60      OP2 = fi(0,prop.short, prop.Properties);     % Operator2 16 Bit
61
62      accu_wl = accu.WordLength;          % maximale Wortlänge von Produkt
63      op_wl = OP1.WordLength;
64
65  %-------------------------------------------------------------------------
66
67  switch (wordlength)
68      case 'short'   % getestet, funktioniert gut
69         re = fi(zeros(1,K),prop.short, prop.Properties);
70         im = fi(zeros(1,K),prop.short, prop.Properties);
71         re2 = fi(zeros(1,K),prop.short, prop.Properties);
72         im2 = fi(zeros(1,K),prop.short, prop.Properties);
73         absolut = fi(zeros(1,K),prop.short, prop.Properties);
74      case '24Bit'   % getestet, funktioniert gut
75         re = fi(zeros(1,K),prop.bit24, prop.Properties);
76         im = fi(zeros(1,K),prop.bit24, prop.Properties);
77         re2 = fi(zeros(1,K),prop.bit24, prop.Properties);
78         im2 = fi(zeros(1,K),prop.bit24, prop.Properties);
79         absolut = fi(zeros(1,K),prop.bit24, prop.Properties);
80      case 'long'    % getestet, funktioniert gut
81         re = fi(zeros(1,K),prop.long, prop.Properties);
82         im = fi(zeros(1,K),prop.long, prop.Properties);
83         re2 = fi(zeros(1,K),prop.long, prop.Properties);
84         im2 = fi(zeros(1,K),prop.long, prop.Properties);
85         absolut = fi(zeros(1,K),prop.long, prop.Properties);
86      otherwise       % Wenn ungültige Wortlänge gewählt
87          error('DFT_quant_int_V2:␣You␣have␣selected␣wrong␣wordlength!');
88  end
89
90         %Schiebefaktor ausrechnen
91
92         s = ceil(log2(N));
93
94         l_harm = (w_LUT -1) + w_ADC  + s; % für 64 Werte: 12 +10 + 6 = 26
95
96
97          if re.wordlength < l_harm
98              shift_re = re.wordlength -l_harm;
99          else
100             shift_re = 0;
101         end
102         shift_test = OP1.wordlength -l_harm -shift_re;  %Begründung in Doku
103         shift_quad = re2.wordlength - accu_wl;          % Damit Summenbildung Gewährleistet
104        for k = 1:K
105            for n = 1:N
106               % ------------------ Realanteil berechnen ------------------
107               OP1(:) = x_t.int(n);
108               OP2(:) = Real(k,n);
109               accu(:) = OP1 .* OP2; % Erzeugung doppeltes VZ
110               if shift_re ~= 0
111                   accu(:) = bitshift(accu,shift_re +1);
112                   accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
113               end
114               re(k) = re(k) + accu; % 1 Wert zwischenspeichern
115
116               % ------------------Imaginäranteil berechnen -------------------
117                OP1(:) = x_t.int(n);
118                OP2(:) = Imag(k,n);
119                accu(:) = OP1 .* OP2; % Erzeugung doppeltes VZ
120                if shift_re ~= 0
121                    accu(:) = bitshift(accu,shift_re +1);
122                accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
```

```
123                 end
124                 im(k) = im(k) + accu; % 1 Wert zwischenspeichern
125             end
126
127             re(k) = bitshift(re(k), (shift_test +1));
128             OP1(:)= quantize(prop.qu16, re.int(k) / 2); % statt bitshift
129             accu(:) = OP1 .* OP1; % Doppeltes Vorzeichen entfernen
130             accu(:) = bitshift(accu, shift_quad +1);
131             re2(k) = quantize(prop.qu32, accu.int / 2); %LSB runden
132
133             im(k) = bitshift(im(k), (shift_test +1));
134             OP1(:) = quantize(prop.qu16, im.int(k) / 2); %%LSB runden
135             accu(:) = OP1 .* OP1;
136             accu(:) = bitshift(accu, shift_quad +1);
137             im2(k) = quantize(prop.qu32, accu.int / 2); % %LSB runden
138             absolut(k) = re2(k) + im2(k); %Absolutwert^2 berechnen
139             absolut(k) =  bitshift(absolut(k), -1); % wegen Addition
140         end
141 end
```

THD_fix_newV3

Listing F.15:

```
1  function [erg L1, Lges] = THD_fix_newV3(x_t, prop,Real, Imag, wordlength, lut)
2  %[erg L1, Lges] = THD_fix_newV3(x_t, prop,Real, Imag, wordlength, lut)
3  %Funktion berechnet den THD (Total Harmonic Distortion), Eingangswerte 32
4  %Bit Worwortbreite. Verwendung von Variablen mit einstellbarer Wortbreite
5  %Berechnung der Wurzelfunktion mit Hilfe einer Tabelle
6  %
7  %Formel:
8  %             /----------------
9  %            /  Uges^2 - U1^2
10 % THD=   _  / ----------------
11 %         \/         Uges^2
12 %
13 % THD[%] = sqrt(factor * 100^2 *2^-3/2^10) => 13 Bit Scheiben der
14 % Oberschwingungen erforderlich
15 %
16 %Eingabe
17 %   x_t  : Eingangssignal
18 %   prop : Fixpoint Eigenschaften
19 %   Real : Kosinustabelle zur Berechnung des Realteils der ersten Harmonischen
20 %   Imag : Kosinustabelle zur Berechnung des Imaginärteils der ersten Harmonischen
21 %   wordlength: verwendete Wortbreite
22 %   lut  :  1: original Tabelle des Radmessplatzes
23 %           2: PCM codiert (9 Stufen)
24 %           3: PCM codiert (8 Stufen)
25 %           4: selbst erarbeites Verfahren
26 %
27 % Ausgabe
28 %   erg : THD in [%]
29 %   L1  : Leistung von 1 Harmonischer Schwingung
30 %   Lges: Gesamtleistung des Signals
31 %
32 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
33 % erstellt  : 15.03.2010
34 % bearbeitet: 15.03.2010
35 % Version   : 1.0
36 % Status:    funktioniert
37 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38 % Probleme:
39 %
40 % Fehler bei Verwendung der eingestellten Auflösung für LUT
41
42 if nargin ~= 6
43     error('myApp:argChk', 'Wrong␣number␣of␣input␣arguments')
44 end
45
46 %-------------------------------------------------------------------------------
47 % Es wird in der Simaulation davon ausgangen, dass für jede Multiplikation
48 % der Hardware Multipizierer des MSP430 verwendet wird, dieser ist z.B im
49 % 1611 integriert
50
51     accu = fi(0,prop.long, prop.Properties);     % 32 Bit Akku
52     OP1 = fi(0,prop.short, prop.Properties);     % Operator1 16 Bit
53     OP2 = fi(0,prop.short, prop.Properties);     % Operator2 16 Bit
54
55     accu_wl = accu.WordLength;          % maximale Wortlänge von Produkt
56     op_wl = OP1.WordLength;
57
58 %-------------------------------------------------------------------------------
59
```

```
60  % Vereinbarung der Fixpoint Variablen
61  switch (wordlength)
62      case 'short'   % getestet, funktioniert recht gut
63          factor = fi(0,prop.long, prop.Properties);
64          sum_ob = fi(0,prop.long, prop.Properties);  %32 Bit Wortlänge
65          % ------------ Variablen 1. Oberschwingung ----------------
66          im1 = fi(0,prop.short, prop.Properties);
67          re1 = fi(0,prop.short, prop.Properties);
68          abs_im1 = fi(0,prop.short, prop.Properties);
69          abs_re1 = fi(0,prop.short, prop.Properties);
70          % ------------------ Gleichanteil ------------------------
71          x_gl = fi(0,prop.short, prop.Properties);
72          % ------------------- Leistungen -------------------------
73          L1 = fi(0,prop.short, prop.Properties); % Leistung 1 Oberschwingung
74          Lges = fi(0,prop.short, prop.Properties); % Leistung Gesamtsignal
75      case '24Bit'
76          factor = fi(0,prop.long, prop.Properties);
77          sum_ob = fi(0,prop.long64, prop.Properties);  %32 Bit Wortlänge
78          % ------------ Variablen 1. Oberschwingung ----------------
79          im1 = fi(0,prop.bit24, prop.Properties);
80          re1 = fi(0,prop.bit24, prop.Properties);
81          abs_im1 = fi(0,prop.bit24, prop.Properties);
82          abs_re1 = fi(0,prop.bit24, prop.Properties);
83          % ------------------ Gleichanteil ------------------------
84          x_gl = fi(0,prop.bit24, prop.Properties);
85          % ------------------- Leistungen -------------------------
86          L1 = fi(0,prop.bit24, prop.Properties); % Leistung 1 Oberschwingung
87          Lges = fi(0,prop.bit24, prop.Properties); % Leistung Gesamtsignal
88      case 'long'    % getestet, funktioniert gut
89          factor = fi(0,prop.long, prop.Properties);
90          sum_ob = fi(0,prop.long64, prop.Properties); %64 Bit Wortlänge
91          %sum_ges = fi(0,prop.long, prop.Properties); %64 Bit Wortlänge
92          % ------------ Variablen 1. Oberschwingung ----------------
93          im1 = fi(0,prop.long, prop.Properties);
94          re1 = fi(0,prop.long, prop.Properties);
95          abs_im1 = fi(0,prop.long, prop.Properties);
96          abs_re1 = fi(0,prop.long, prop.Properties);
97          % ------------------ Gleichanteil ------------------------
98          x_gl = fi(0,prop.long, prop.Properties);
99          % ------------------- Leistungen -------------------------
100         L1 = fi(0,prop.long, prop.Properties); % Leistung 1 Oberschwingung
101         Lges = fi(0,prop.long, prop.Properties); % Leistung Gesamtsignal
102     otherwise %Wenn ungültige Wortlänge gewählt
103         error('THD_fix_new: You have selected wrong wordlength!');
104 end
105
106     % Eingestellte Wortbreiten analysieren
107      N = length(x_t);
108
109      w_LUT = Real.WordLength; % Wortlänge LUT Sinus und Kosinus
110      w_ADC = x_t.Wordlength;  % Wortlänge ADC Samples
111
112
113     %Ausgleichfaktoren für unterschiedliche Wortbreite ADC und LUT
114     if w_ADC >= (w_LUT-1)
115         fak_LUT = w_ADC - (w_LUT -1);
116         fak_ADC = 0;
117     elseif w_ADC < (w_LUT-1)
118          fak_LUT = 0;
119          fak_ADC = (w_LUT -1) - w_ADC;
120     end
121
122     %% Schiebefaktoren ermitteln
```

```
123     % Scheibefaktor bestimmen um /L zu realisieren
124
125     s = ceil(log2(N));
126
127     % Für Gleichanteil
128
129     l_max = w_ADC + s;
130     if x_gl.wordlength <= l_max % = da MSB vorzeichen
131         shift_gl = x_gl.wordlength -l_max -1;
132     else
133         shift_gl = 0;
134     end
135
136     %Für Gesamtleistung des Signals
137     %Für Addition der Samples
138     w_max_add = 2 * w_ADC +2 + s;
139
140     if Lges.wordlength <  w_max_add
141         shift_Lg_add = Lges.wordlength -w_max_add;
142     else
143         shift_Lg_add = 0;
144     end
145
146     %Prüfen ob ausreichende Wortbreite für richtige Skallierung
147     % ist so erforderlich, da w_max_add 0 sein kann.
148     if Lges.wordlength <  (w_max_add - shift_Lg_add +2 * fak_ADC) %Wenn Wortbreite für
            Skallierung nicht ausreichend ist
149         shift_Lg_skall = -2 * fak_ADC;
150     else
151         shift_Lg_skall = 0;
152     end
153
154     shift_Lg = shift_Lg_add + shift_Lg_skall;
155
156
157
158     %% Gleichanteil berechnen
159     for n = 1:N
160         OP1(:) = x_t(n);
161         x_gl(:) = x_gl + bitshift(OP1,shift_gl);
162     end
163     x_gl(:) = bitshift(x_gl,-s - shift_gl +1); %Division durch N
164     x_gl(:) = quantize(prop.qu16, x_gl.int / 2); % statt bitshift
165
166
167     %% Leistung Signal ohne Gleichanteil bestimmen
168     % wenn maximal 16 Bit ADC verwendet wird, ist Gleichanteil max 15 Bit lang
169     x0 = int16(x_t - x_gl); % bis hier einwandfrei für 32 Bit Register
170
171     for n = 1:N
172         OP1(:) = x0(n); % da Signal nur im positiven Berech => Gleichanteil Umax/2
173         accu(:) = OP1 .* OP1;
174          if shift_Lg_add ~= 0
175         accu(:) = bitshift(accu,shift_Lg_add +1);
176         accu(:) = quantize(prop.qu32, accu.int / 2); % LSB runden
177          end
178         Lges(:) = Lges + accu;
179     end
180
181     %Skallierung der Leeistung des Signals
182     Lges(:) = bitshift (Lges, 2*fak_ADC + shift_Lg_skall);
183
184
```

```
185
186
187     %% Leistung 1. Oberschwingung bestimmen( aus DFT_quant_int_V3)
188
189     l_harm = (w_LUT -1) + fak_LUT + w_ADC + fak_ADC + s; % für 64 Werte: 12 +12 + 6 = 28
190
191
192     if re1.wordlength < l_harm
193       shift_re1 = re1.wordlength -l_harm;
194     else
195       shift_re1 = 0;
196     end
197
198     shift_test = OP1.wordlength -l_harm -shift_re1;
199     shift_quad = -(accu.wordlength -l_harm -2  - shift_Lg);
200     if shift_quad > 0
201         Lges(:) = bitshift(Lges,-shift_quad);
202         shift_quad = 0;
203     end
204
205       for n = 1:N
206             % ------------------- Realanteil berechnen -------------------
207             OP1(:) = x_t.int(n);
208             OP1(:) = bitshift(OP1,fak_ADC);
209             OP2(:) = Real(n);
210             OP2(:) = bitshift(OP2, fak_LUT);
211             accu(:) = OP1 .* OP2;
212             if shift_re1 ~= 0
213                 accu(:) = bitshift(accu,shift_re1 +1);
214                 accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
215             end
216             re1(:) = re1 + accu; % 1 Wert zwischenspeichern
217
218             %-----------------------Imaginäranteil berechnen -----------------
219             OP1(:) = x_t.int(n);
220             OP1(:) = bitshift(OP1,fak_ADC);
221             OP2(:) = Imag(n);
222             OP2(:) = bitshift(OP2, fak_LUT);
223             accu(:) = OP1 .* OP2; % Erzeugung doppeltes VZ
224              if shift_re1 ~= 0
225             accu(:) = bitshift(accu,shift_re1 +1);
226             accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
227              end
228             im1(:) = im1  + accu; % 1 Wert zwischenspeichern
229       end
230      re1(:) = bitshift(re1, (shift_test +1));%Auf 16 Bit reduzierten
231      OP1(:) = quantize(prop.qu16, re1.int / 2); % statt bitshift
232      accu(:) = OP1 .* OP1; % Doppeltes Vorzeichen entfernen
233      accu(:) = bitshift(accu, shift_quad + 1);
234      abs_re1(:) = quantize(prop.qu32, accu.int / 2); %%LSB runden
235      im1(:) = bitshift(im1, (shift_test +1)); %Auf 16 Bit reduzierten
236      OP1(:) = quantize(prop.qu16, im1.int / 2); % %LSB runden
237      accu(:) = OP1 .* OP1;
238      accu(:) = bitshift(accu, shift_quad + 1);
239      abs_im1(:) = quantize(prop.qu32, accu.int / 2); % %LSB runden
240      L1(:) = abs_re1 +  abs_im1;         %Leistung 1. Harmonische berechnen
241      L1(:) =  bitshift(L1, -1);          % wegen Addition
242
243
244      %% Division durchführen
245     if Lges > L1
246         sum_ob(:)  = Lges - L1;
```

```
247     else                %sonst ist der THD so klein, dass er nicht mehr Darstellbar ist
            also 0
248         sum_ob(:) = 0;
249     end
250
251
252     %% Ergebnis aus Tabellen 1 -4 ermitteln
253     % Ausgabe Übersicht wichtigste Werte, dient nur zur Kontrolle
254         disp(['Gleichanteil:    ' num2str(x_gl.double)]); %Testausgabe
255         disp(['Gesamtleisung:   ' num2str(Lges.double)]); %Testausgabe
256         disp(['Leistung 1 Harm.:' num2str(L1.double)]); %Testausgabe
257         disp(['THD_fix: sum_ob: ' sum_ob.bin]); %Testausgabe
258         disp(['THD_fix: sum_gs: ' Lges.bin]);
259         disp(['THD_fix: factor: ' factor.bin]);
260
261
262         if lut == 1
263
264             ob_shift = 10;
265
266             sum_ob(:) = bitshift(sum_ob ,ob_shift);
267             factor(:) = divide(prop.long, sum_ob, Lges);
268
269             %Erstellen der Tabelle
270             x = 0: 10000 / 1024: (2^8 -1) * 10000 / 1024;   %256 Werte in Tabelle
                    enthalten
271             Wurzel = round(sqrt(x));
272
273             if factor > length(Wurzel);   % Wenn sehr großer Klirrfaktor
274                erg = Wurzel(end);
275             else
276                 erg = Wurzel(factor.int +1); %Wert aus Tabelle ermitteln
277             end
278
279
280
281         elseif lut == 2
282             %Definition der Variablen für Ermittlung der Tabellen- Funktion
283
284            ob_shift = 14;       % Schiebefaktor der Oberschwingungen
285            shiftdiff = ob_shift-14; % Shiffaktordifferenz = ob_shift-13
286            Spalte = 0;
287            Zeile = 0;
288
289            sum_ob(:) = bitshift(sum_ob ,ob_shift);
290            factor(:) = divide(prop.long, sum_ob, Lges);
291
292             for bit = 20:29- shiftdiff
293                if factor.bin(bit) == '1'  % wenn Bit gesetzt ist
294                     Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
295                     Zeile = 32- bit -2 -  shiftdiff;
296                     if Zeile > 8   % Für Große THD bleibt Zeile Konstant
297                         Zeile = 8;
298                         Spalte = 7;
299                    end
300                    break;
301                end
302             end
303            %wenn Zeile und Spalte weiterhin 0: THD sehr klein
304
305            if (Spalte == 0) && Zeile == 0
306                Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
307            end
```

```
308
309             % ermittelte Position in der Tabelle anzeigen
310             disp(['Zeile: ' num2str(Zeile)]);
311             disp(['Spalte: ' num2str(Spalte)]);
312             load('SQRT_LUT_PCM_9STUFEN.mat');
313             erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
314
315         elseif lut == 3
316             %Definition der Variablen für Ermittlung der Tabellen- Funktion
317
318             ob_shift = 13;      % Schiebefaktor der Oberschwingungen
319             shiftdiff = ob_shift-13; % Shiffaktordifferenz = ob_shift-13
320             Spalte = 0;
321             Zeile = 0;
322
323             sum_ob(:) = bitshift(sum_ob ,ob_shift);
324             factor(:) = divide(prop.long, sum_ob, Lges);
325
326              for bit = 20:29- shiftdiff
327                 if factor.bin(bit) == '1' % wenn Bit gesetzt ist
328                     Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
329                     Zeile = 32- bit -2 -  shiftdiff;
330                     if Zeile > 7   % Für sehr großen Klirrfaktor konstanter Wert
331                         Zeile = 7;
332                         Spalte = 7;
333                     end
334                     break;
335                 end
336              end
337             if (Spalte == 0) && Zeile == 0
338                 Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
339             end
340
341             disp(['Zeile: ' num2str(Zeile)]);
342             disp(['Spalte: ' num2str(Spalte)]);
343             load('SQRT_LUT_PCM_8STUFEN.mat');
344             erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
345         elseif lut == 4
346
347               ob_shift = 13;      % Schiebefaktor der Oberschwingungen
348               Wurzel = sqrt_table;      % Wurzeltabellen  erzeugen
349
350               sum_ob(:) = bitshift(sum_ob ,ob_shift);
351               factor(:) = divide(prop.long, sum_ob, Lges);
352
353               if factor >= 7
354                 factor(:) = bitshift(factor, -2);  %Ergebnis schieben und in anderer
                        Tabelle
355                 if factor >= 7                    % wenn factor immer noch groß
356                     factor(:) = bitshift(factor, -1); %nochmals um ein Bit schieben und inb
                            anderer Tablle suchen
357                     if factor >= 16                   % wenn immer noch groß
358                         factor(:) = bitshift(factor, -1); % nochmals um ein  Bit schieben
                                und in anderer Tabelle nachsehen
359                         if factor >= 32                 % wenn immer noch groß
360                             factor(:) = bitshift(factor, -1); % nochmals um ein  Bit
                                    schieben und in anderer Tabelle nachsehen
361                             if factor >= 32                   %wenn immer noch zu groß
                                    -> sehr großer THD
362                                 factor(:) = 31;               % Maximalwert verwenden
363                                 erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle
                                        5
364                             else
```

```
365                                     erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle
                                            5
366                                 end
367                             else
368                                 erg = Wurzel.Table4(factor.int); %Ergebnis aus Tabelle 4
369                             end
370                         else
371                             erg = Wurzel.Table3(factor.int); %Ergebnis aus Tabelle 3
372                         end
373                     else
374                         erg = Wurzel.Table2(factor.int); %Ergebnis aus Tabelle 2
375                     end
376                 else
377                     erg = Wurzel.Table1(factor.int +1); %Ergebnis aus Tabelle 1
378                   end
379             else
380                 error('wrong look-up table!')
381             end
382 end
```

THD_fix_newV4

Listing F.16: ../bla.m

```
function [erg L1, Lges] = THD_fix_newV4(x_t, prop,Real, Imag, wordlength, lut)
%
%[erg L1, Lges] = THD_fix_newV4(x_t, prop,Real, Imag, wordlength, lut)
%
%Funktion berechnet den THD mit Hilfe des noise-included hd- Verfahrens.
%Die Wortbreiten der Variablen können eingestellt werden. Diese Version
%arbeitet mit tatsächlichen C Implementierung.
%Berechnung der Wurzelfunktion mit Hilfe einer Tabelle
%
%Formel:
%              /-----------------------
%             /   Uges^2 - U1^2
% THD=    _  / ---------------------
%          \/         Uges^2
%
% THD[%] = sqrt(factor * 100^2 *2^-3/2^10) => 13 Bit Scheiben der
% Oberschwingungen erforderlich
%
%Eingabe
%   x_t  : Eingangssignal
%   prop : Fixpoint Eigenschaften
%   Real : Kosinustabelle zur Berechnung des Realteils der ersten Harmonischen
%   Imag : Kosinustabelle zur Berechnung des Imaginärteils der ersten Harmonischen
%   wordlength: verwendete Wortbreite
%   lut  :  1: original Tabelle des Radmessplatzes
%           2: PCM codiert (9 Stufen)
%           3: PCM codiert (8 Stufen)
%           4: selbst erarbeites Verfahren
%
% Ausgabe
%   erg : THD in [%]
%   L1  : Leistung von 1 Harmonischer Schwingung
%   Lges: Gesamtleistung des Signals
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% erstellt  : 15.03.2010
% bearbeitet: 15.03.2010
% Version   : 0.9
% Status:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin ~= 6
    error('myApp:argChk', 'Wrong number of input arguments')
end

%---------------------------------------------------------------------------
% Es wird in der Simaulation davon ausgangen, dass für jede Multiplikation
% der Hardware Multipizierer des MSP430 verwendet wird, dieser ist z.B im
% 1611 integriert

    accu = fi(0,prop.long, prop.Properties);     % 32 Bit Akku
    OP1 = fi(0,prop.short, prop.Properties);     % Operator1 16 Bit
    OP2 = fi(0,prop.short, prop.Properties);     % Operator2 16 Bit

    accu_wl = accu.WordLength;          % maximale Wortlänge von Produkt
    op_wl = OP1.WordLength;

%---------------------------------------------------------------------------

```

```
60  % Vereinbarung der Fixpoint Variablen
61  switch (wordlength)
62      case 'short'   % getestet, funktioniert recht gut
63          factor = fi(0,prop.long, prop.Properties);
64          sum_ob = fi(0,prop.long, prop.Properties);  %32 Bit Wortlänge
65          % ------------ Variablen 1. Oberschwingung ----------------
66          im1 = fi(0,prop.short, prop.Properties);
67          re1 = fi(0,prop.short, prop.Properties);
68          abs_im1 = fi(0,prop.short, prop.Properties);
69          abs_re1 = fi(0,prop.short, prop.Properties);
70          % ------------------ Gleichanteil ------------------------
71          x_gl = fi(0,prop.short, prop.Properties);
72          % ------------------- Leistungen -------------------------
73          L1 = fi(0,prop.short, prop.Properties); % Leistung 1 Oberschwingung
74          Lges = fi(0,prop.short, prop.Properties); % Leistung Gesamtsignal
75      case '24Bit'
76          factor = fi(0,prop.long, prop.Properties);
77          sum_ob = fi(0,prop.long64, prop.Properties);  %32 Bit Wortlänge
78          % ------------ Variablen 1. Oberschwingung ----------------
79          im1 = fi(0,prop.bit24, prop.Properties);
80          re1 = fi(0,prop.bit24, prop.Properties);
81          abs_im1 = fi(0,prop.bit24, prop.Properties);
82          abs_re1 = fi(0,prop.bit24, prop.Properties);
83          % ------------------ Gleichanteil ------------------------
84          x_gl = fi(0,prop.bit24, prop.Properties);
85          % ------------------- Leistungen -------------------------
86          L1 = fi(0,prop.bit24, prop.Properties); % Leistung 1 Oberschwingung
87          Lges = fi(0,prop.bit24, prop.Properties); % Leistung Gesamtsignal
88      case 'long'    % getestet, funktioniert gut
89          factor = fi(0,prop.long, prop.Properties);
90          sum_ob = fi(0,prop.long64, prop.Properties); %64 Bit Wortlänge
91          %sum_ges = fi(0,prop.long, prop.Properties); %64 Bit Wortlänge
92          % ------------ Variablen 1. Oberschwingung ----------------
93          im1 = fi(0,prop.long, prop.Properties);
94          re1 = fi(0,prop.long, prop.Properties);
95          abs_im1 = fi(0,prop.long, prop.Properties);
96          abs_re1 = fi(0,prop.long, prop.Properties);
97          % ------------------ Gleichanteil ------------------------
98          x_gl = fi(0,prop.long, prop.Properties);
99          % ------------------- Leistungen -------------------------
100         L1 = fi(0,prop.long, prop.Properties); % Leistung 1 Oberschwingung
101         Lges = fi(0,prop.long, prop.Properties); % Leistung Gesamtsignal
102     otherwise %Wenn ungültige Wortlänge gewählt
103         error('THD_fix_new: You have selected wrong wordlength!');
104 end
105
106     %% Schiebefaktoren ermitteln
107
108     N = length(x_t);
109
110      w_LUT = Real.WordLength; % Wortlänge LUT Sinus und Kosinus
111      w_ADC = x_t.Wordlength;  % Wortlänge ADC Samples
112
113
114     if w_ADC >= (w_LUT-1)
115         fak_LUT = w_ADC - (w_LUT -1);
116         fak_ADC = 0;
117     elseif w_ADC < (w_LUT-1)
118          fak_LUT = 0;
119          fak_ADC = (w_LUT -1) - w_ADC;
120     end
121
122
```

```
123     % Scheibefaktor bestimmen um /L zu realisieren
124
125     s = ceil(log2(N));
126
127     %Für Gesamtleistung des Signals
128     %Für Addition der Samples
129     w_max_add = 2 * w_ADC + s;
130
131     if Lges.wordlength <  w_max_add
132         shift_Lg_add = Lges.wordlength -w_max_add;
133     else
134         shift_Lg_add = 0;
135     end
136
137     %Prüfen ob ausreichende Wortbreite für richtige Skallierung
138     % ist so erforderlich, da w_max_add 0 sein kann.
139     if Lges.wordlength <  (w_max_add - shift_Lg_add +2 * fak_ADC) %Wenn Wortbreite für
            Skallierung nicht ausreichend ist
140         shift_Lg_skall = -2 * fak_ADC;
141     else
142         shift_Lg_skall = 0;
143     end
144
145     shift_Lg = shift_Lg_add + shift_Lg_skall;
146
147
148
149
150     % Für Gleichanteil
151
152     l_max = w_ADC + s;
153     if x_gl.wordlength <= l_max % da MSB vorzeichen
154         shift_gl = x_gl.wordlength -l_max -1;
155     else
156         shift_gl = 0;
157     end
158
159     % Division durch N aufteilen 1. Schritt so weit,dass Ergebnis 16 Bit
160     shift_gl1 = op_wl - l_max -1;
161     if shift_gl1 > 0
162        shift_gl1 = 0;       %Nach links schieben vermeiden
163     end
164
165     %Schiebefaktor  quadrieren
166     shift_quad_gl = -2* shift_gl - (s +  2*shift_gl1)  + shift_Lg_add;
167
168     %                *Faktor gl        /s                  /Faktor Lg
169
170
171
172     %% Gesamtleistung des Signals ohne Gleicheinteil bestimmen (2. Lösung)
173     for n = 1:N
174         % Gleichnatei berechnen
175         OP1(:) = int16(x_t(n));
176         x_gl(:) = x_gl + bitshift(OP1,shift_gl);
177
178
179         % Gesamtleistung des Signals mit Gleichanteil
180         accu(:) = OP1 .* OP1;
181         if shift_Lg_add ~= 0
182             accu(:) = bitshift(accu,shift_Lg_add +1);
183             accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
184         end
```

```
185         Lges(:) = Lges + accu;
186     end
187
188     %Lges(:) = bitshift(Lges, -s); % * 1/N
189
190     x_gl(:) = bitshift(x_gl, shift_gl1);  %*1/N Teil 1
191     OP1(:) = x_gl.int;
192     accu(:) = OP1 .* OP1;
193     if shift_quad_gl ~= 0
194         accu(:) = bitshift(accu, shift_quad_gl +1);
195         accu(:) = quantize(prop.qu32, accu.int / 2);% statt bitshift
196     end
197
198     %Lges(:) = Lges - accu;
199     Lges(:) = bitshift (Lges - accu, 2*fak_ADC + shift_Lg_skall);
200
201      %% Leistung 1. Oberschwingung bestimmen( aus DFT_quant_int_V3)
202
203     l_harm = (w_LUT -1) + fak_LUT + w_ADC + fak_ADC + s; % für 64 Werte: 12 +10 + 6 = 26
204
205
206     if re1.wordlength < l_harm
207       shift_re1 = re1.wordlength -l_harm;
208     else
209       shift_re1 = 0;
210     end
211
212     shift_test = OP1.wordlength -l_harm -shift_re1;
213     %shift_quad = -(accu.wordlength -l_harm -3  - (w_ADC- (w_LUT-1)) - shift_Lg);
214     shift_quad = -(accu.wordlength -l_harm -2  - shift_Lg);
215     if shift_quad > 0
216         Lges(:) = bitshift(Lges,-shift_quad);
217         shift_quad = 0;
218     end
219
220       for n = 1:N
221             % ------------------ Realanteil berechnen ------------------
222             OP1(:) = int16(x_t.int(n));
223             OP1(:) = bitshift(OP1,fak_ADC);
224             OP2(:) = int16(Real(n));
225             OP2(:) = bitshift(OP2, fak_LUT);
226             accu(:) = OP1 .* OP2; % Erzeugung doppeltes VZ
227             if shift_re1 ~= 0
228                 accu(:) = bitshift(accu,shift_re1 +1);
229                 accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
230             end
231             re1(:) = re1 + accu; % 1 Wert zwischenspeichern
232
233             %Imaginäranteil berechnen
234             OP1(:) = int16(x_t.int(n));
235             OP1(:) = bitshift(OP1,fak_ADC);
236             OP2(:) = int16(Imag(n));
237             OP2(:) = bitshift(OP2, fak_LUT);
238             accu(:) = OP1 .* OP2; % Erzeugung doppeltes VZ
239              if shift_re1 ~= 0
240             accu(:) = bitshift(accu,shift_re1 +1);
241             accu(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
242              end
243             im1(:) = im1  + accu; % 1 Wert zwischenspeichern
244        end
245       re1(:) = bitshift(re1, (shift_test +1));
246       OP1(:) = quantize(prop.qu16, re1.int / 2); % statt bitshift
247       accu(:) = OP1 .* OP1; % Doppeltes Vorzeichen entfernen
```

```
248       accu(:) = bitshift(accu, shift_quad + 1);
249       abs_re1(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
250       im1(:) = bitshift(im1, (shift_test +1));
251       OP1(:) = quantize(prop.qu16, im1.int / 2); % statt bitshift
252       accu(:) = OP1 .* OP1;
253       accu(:) = bitshift(accu, shift_quad + 1);
254       abs_im1(:) = quantize(prop.qu32, accu.int / 2); % statt bitshift
255       L1(:) = abs_re1 +  abs_im1; %Absolutwert^2 berechnen
256       L1(:) =  bitshift(L1, -1); % wegen Addition
257
258       %% Division durchführen
259      if Lges > L1
260          sum_ob(:)  = Lges - L1;
261      else
262          sum_ob(:) = 0;
263      end
264      %sonst ist der THD so klein, dass er nicht mehr Darstellbar ist also 0
265
266
267      %% Ergebnis aus Tabellen 1 -4 ermitteln
268      % Ausgabe Übersicht wichtigste Werte, dient nur zur Kontrolle
269           disp(['Gleichanteil:    ' num2str(x_gl.double)]); %Testausgabe
270           disp(['Gesamtleisung:   ' num2str(Lges.double)]); %Testausgabe
271           disp(['Leistung 1 Harm.:' num2str(L1.double)]); %Testausgabe
272           disp(['THD_fix: sum_ob: ' sum_ob.bin]); %Testausgabe
273           disp(['THD_fix: sum_gs: ' Lges.bin]);
274           disp(['THD_fix: factor: ' factor.bin]);
275
276
277           if lut == 1
278
279               ob_shift = 10;
280
281               sum_ob(:) = bitshift(sum_ob ,ob_shift);
282               factor(:) = divide(prop.long, sum_ob, Lges);
283
284               %Erstellen der Tabelle
285               x = 0: 10000 / 1024: (2^8 -1) * 10000 / 1024;   %256 Werte in Tabelle
                      enthalten
286               Wurzel = round(sqrt(x));
287
288               if factor > length(Wurzel);   % Wenn sehr großer Klirrfaktor
289                  erg = Wurzel(end);
290               else
291                   erg = Wurzel(factor.int +1); %Wert aus Tabelle ermitteln
292               end
293
294
295
296           elseif lut == 2
297               %Definition der Variablen für Ermittlung der Tabellen- Funktion
298
299              ob_shift = 14;       % Schiebefaktor der Oberschwingungen
300              shiftdiff = ob_shift-14; % Shiffaktordifferenz = ob_shift-13
301              Spalte = 0;
302              Zeile = 0;
303
304              sum_ob(:) = bitshift(sum_ob ,ob_shift);
305              factor(:) = divide(prop.long, sum_ob, Lges);
306
307               for bit = 20:29- shiftdiff
308                  if factor.bin(bit) == '1'  % wenn Bit gesetzt ist
309                      Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
```

```
310                     Zeile = 32- bit -2 -  shiftdiff;
311                     if Zeile > 8   % Für Große THD bleibt Zeile Konstant
312                         Zeile = 8;
313                         Spalte = 7;
314                     end
315                     break;
316                 end
317              end
318             %wenn Zeile und Spalte weiterhin 0: THD sehr klein
319
320             if (Spalte == 0) && Zeile == 0
321                 Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
322             end
323
324             % ermittelte Position in der Tabelle anzeigen
325             disp(['Zeile: ' num2str(Zeile)]);
326             disp(['Spalte: ' num2str(Spalte)]);
327             load('SQRT_LUT_PCM_9STUFEN.mat');
328             erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
329
330         elseif lut == 3
331             %Definition der Variablen für Ermittlung der Tabellen- Funktion
332
333             ob_shift = 13;       % Schiebefaktor der Oberschwingungen
334             shiftdiff = ob_shift-13; % Shiffaktordifferenz = ob_shift-13
335             Spalte = 0;
336             Zeile = 0;
337
338             sum_ob(:) = bitshift(sum_ob ,ob_shift);
339             factor(:) = divide(prop.long, sum_ob, Lges);
340
341              for bit = 20:29- shiftdiff
342                 if factor.bin(bit) == '1'  % wenn Bit gesetzt ist
343                     Spalte = bin2num(prop.qu32, factor.bin(bit+1:bit +3));
344                     Zeile = 32- bit -2 -  shiftdiff;
345                     if Zeile > 7   % Für sehr großen Klirrfaktor konstanter Wert
346                         Zeile = 7;
347                         Spalte = 7;
348                     end
349                     break;
350                 end
351              end
352             if (Spalte == 0) && Zeile == 0
353                 Spalte = bin2num(prop.qu16, factor.bin(end-2- shiftdiff:end-shiftdiff));
354             end
355
356             disp(['Zeile: ' num2str(Zeile)]);
357             disp(['Spalte: ' num2str(Spalte)]);
358             load('SQRT_LUT_PCM_8STUFEN.mat');
359             erg = Tabelle(Zeile+1,Spalte+1); %Ausgabe des Ergebnisses
360         elseif lut == 4
361
362             ob_shift = 13;       % Schiebefaktor der Oberschwingungen
363             Wurzel = sqrt_table;       % Wurzeltabellen  erzeugen
364
365             sum_ob(:) = bitshift(sum_ob ,ob_shift);
366             factor(:) = divide(prop.long, sum_ob, Lges);
367
368             if factor >= 7
369                 factor(:) = bitshift(factor, -2);  %Ergebnis schieben und in anderer
                        Tabelle
370                 if factor >= 7                     % wenn factor immer noch groß
```

```
371                     factor(:) = bitshift(factor, -1); %nochmals um ein Bit schieben und inb
                            anderer Tablle suchen
372                     if factor >= 16                   % wenn immer noch groß
373                         factor(:) = bitshift(factor, -1); % nochmals um ein  Bit schieben
                                und in anderer Tabelle nachsehen
374                         if factor >= 32                   % wenn immer noch groß
375                             factor(:) = bitshift(factor, -1); % nochmals um ein  Bit
                                    schieben und in anderer Tabelle nachsehen
376                             if factor >= 32                   %wenn immer noch zu groß
                                    -> sehr großer THD
377                                 factor(:) = 31;               % Maximalwert verwenden
378                                 erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle
                                        5
379                             else
380                                 erg = Wurzel.Table5(factor.int); %Ergebnis aus Tabelle
                                        5
381                             end
382                         else
383                             erg = Wurzel.Table4(factor.int); %Ergebnis aus Tabelle 4
384                         end
385                     else
386                         erg = Wurzel.Table3(factor.int); %Ergebnis aus Tabelle 3
387                     end
388                 else
389                     erg = Wurzel.Table2(factor.int); %Ergebnis aus Tabelle 2
390                 end
391             else
392                 erg = Wurzel.Table1(factor.int +1); %Ergebnis aus Tabelle 1
393             end
394         else
395             error('wrong␣look-up_table!')
396         end
397 end
```

fehler

Listing F.17:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Version 1.0
%
% Datei:                fehler.m
%
% erstellt von: Lennart Koch
% erstellt am:  22.10.2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ rel_fehler, abs_fehler ] = fehler( x_a , x_r)
%function [ rel_fehler, abs_fehler ] = fehler( x_a , x_r)
%Berechnung vom relativen- und absoluten Fehler von zwei übergebenen
%Werten oder Vektoren
%Eingabe
    % x_r: idealer Wert
    % x_a: realer Wert
% Ausgabe
    %rel_feher in %
    %abs_fehler



 abs_fehler = abs(x_a - x_r);

 rel_fehler = abs(abs_fehler ./ x_r) .*100; % in Prozent

end
```

sqrt_table

Listing F.18:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Version 1.0
%
% Datei:                 sqrt_table.m
%
% erstellt von: Lennart Koch
% erstellt am:  22.10.2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ sqrt_tab ] = sqrt_table()
% Funktion erstellt Wurzeltabelle in 4 Auflösungen,
% Tabellen werden in Struktur bereitgestellt
    %   Rückgabewert:
    %   sqrt_tab : Struktur bestehend aus den 5 Wurzeltabellen

    % Implementierung 1 Stufe: 8 Werte
    x_imp1 = 0:1:8-1;
    Func_imp1 = sqrt(100^2*2^-3/2^10 * x_imp1);

    % Implementierung 2. Stufe: 8 Werte
    % start = 7
    %wertpos 111(7) >> 2 = 1 % Position an der vorheriger max.Wert vorkommt
    x_imp2 = 7:4:8*4 +4 -1; %wg Scheibefaktor 2
    Func_imp2 = sqrt(100^2*2^-3/2^10 * x_imp2);



    % Implemtierung 3. Stufe: 16 Werte
    % start = 7 + 1 >> 3  = 15
    %wertpos 10 0100 (35) >> 3 = 100= 4 % Position an der vorheriger max.Wert vorkommt
    x_imp3 = 15:8:16*8+8-1;
    Func_imp3 = sqrt(100^2 * 2^-3/2^10* x_imp3);

    %Impelementierung 4. Stufe = 16 Werte
    % start = 15 + 1 << 4 = 31
    %wertpos= 1000 0011 >> 4 = 100= 4 % Position an der vorheriger max.Wert vorkommt
    x_imp4 = 31:16:16*32+31-1;
    Func_imp4 = sqrt(100^2 * 2^-3/2^10 * x_imp4);

    %Implementierung 5. Stufe = 32 Werte
    % start = 31 + 1 << 5 = 63
    %wertpos= 1 0000 0111 >> 5 = 100= 8 % Position an der vorheriger max.Wert vorkommt
    x_imp5 = 63:32:32*31+ 63 -1;
    Func_imp5 = sqrt(100^2 * 2^-3/2^10 * x_imp5);
    error = find(Func_imp5 > 31); % maximal Wert 31%
    Func_imp5(error) = 31;

    sqrt_tab = struct('Table1', Func_imp1, 'Table2', Func_imp2, 'Table3', Func_imp3,...
                      'Table4', Func_imp4, 'Table5', Func_imp5);

end
```

THD

Listing F.19:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Version 1.0
%
% Datei:                THD.m
%
% erstellt von: Lennart Koch
% erstellt am:  21.09.2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function erg = THD( S_f, No )
%Funktion berechnet den THD (Total Harmonic Distortion) von einem
%übergebenen Spektrum in %
%Formel :
%          N                N
% sqrt( sum   |S_f|^2   / sum   |S_f|^2   )
%         k = 2             k = 1
%
%
%   S_f: Spektrum eines Signals
%   No : Anzahl der Harmonischen Schwingungen

maxTHDlength = fix((length(S_f) / 2) -2);

if nargin == 1
    No = maxTHDlength;
elseif nargin ~= 2
    error('myApp:argChk', 'Wrong␣number␣of␣input␣arguments')
end

% Prüfen ob THD wie gewünscht berechnet werden kann

 %Letzte zwei Schwingungen sollen nicht mehr berücksichtigt werden


    if maxTHDlength >= No
        % Betrag der Amplituden der Oberschwingungen

          sum_ob = sum(abs(S_f(3:No + 1).^2));

        %Betrag der Amplituden mit der Grundschwingung

          sum_gs = sum_ob + abs(S_f(2).^2);

        % Berechnung THD

        erg = sqrt (sum_ob /  sum_gs) * 100;
    else
        error('Berechnung␣konnte␣nicht␣ausgeführt␣werden,␣da␣zu␣wenig␣Spektrallinien␣im␣
            Spektrum␣enthalten␣sind');

    end
end
```

THD_calculation

Listing F.20:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Version 1.0
%
% Datei:              THD_calculation.m
%
% erstellt von: Lennart Koch
% erstellt am:  8.12.2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [THD_app THD_full  THD_new  re, im, betrag] = THD_calculation(s, V, No, LM)
% Berechnung des THD sowie Real- und Imaginärteil mit Matlab ohne Fixpoint-
% Toolbox.
% Simulation der Fälle
% - Aufnahme der Daten mit demonstrator, Auswertung mit Matlab
% - Aufnahme der Daten mit Oszilloskop, Auswertung der Daten mit Matlab
%
        % Eingabewerte:
    % S        : quantisiertes Eingangssignal
    % V        : Verstärkerstufen des Vorverstärkers
    % N        : Anzahl der verwendeten Koeffizienten für s
    % No       : Anzahl der zu berechneten Koeffizienten (Ergebnis für
    %             THD_app)
        %
        % Ausgabe
        % THD_app  : HD5-Methode
        % THD_full: HD undlich
        % THD_new  : HDI-Methode
        % re       : Realteil der Harmonischen
        % im       : Imaginärteil der Harmonischen
        % betrag   : Betrag der Harmonischen

% Neue Berechnungsmethode hinzugefügt


    %% Felder initialisieren

    % Länge der Messreihe bestimmen

    L  = size(s, 2);
    THD_full = zeros(1,LM);  % THD von allen Koeffizienten
    THD_new = zeros(1,LM);   % THD neue Berechnungsmethode
    THD_app = zeros(1,LM);   % THD "No" Koeffizienten
    re = zeros(No, LM);      % Realteil
    im = zeros(No, LM);      % Imaginärteil
    betrag = zeros(No, LM);  % Beträge

    %% Felder mit Werten füllen

    for i = 1:LM
        S = fft(s(i,:)); % FFT von quantisiertem Signal berechnen

        %Leistung Signal ausrechnen
        sig = s(i,:) - S(1)/L;
        Psig = sum(sig .* sig) / (L);  %S(1) = Gleichanteil
        P1 = 2* (abs(S(2))/L)^2 ;      % Leistung 1. Harmonische
        %THD nach anderer Berechnungsmethode ermitteln
        THD_new(i) = 100 * sqrt((Psig -P1)/ Psig);

```

```
60          % THD nach gewohnter Rechenmethode ermitteln
61          re(:,i) = real(S(2:No+1)) ./ V(i);  % Realteil bestimmen
62          im(:,i) = imag(S(2:No+1)) ./ V(i);  %Imaginärteil bestimmen
63          betrag(:,i) = abs(S(2:No+1)) ./V(i);%Betrag bestimmen
64          THD_full(i) = THD(S);
65          THD_app(i) = THD(S,No);
66      end
67
68  end
```

define_Types

Listing F.21:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Erstellen von Fixpoint- Datentypen und quantizer Objekten um bin2num,
% hex2num, num2hex und num2bin zu verwenden
%
% Version 1.0
%
%
% Datei:                define_Types.m
%
%
% erstellt von: Lennart Koch
% erstellt am:  01.11.2009
% geändert:     18.11.2009
%
% quantizer Objekte q16 und q32 eingefügt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function erg = define_Types(N_ADC, N_LUT)
%[F,long,long64,short,tADC,tLUT,tTHD] = define_Types(N_THD,N_ADC,N_LUT)
%   N_ADC:     Wortbreite für Samples vom ADC [Bit]
%   N_LUT:     Wertbreite für Sinus und Cosinus Werte in LUT [Bit]
%Ausgabe
%   Datenstrukur, die alle Typen enthält

    warning on  fi:overflow     %Ausgabe einer Warnung bei Overflow
    warning off fi:underflow    %Augabe einer Warnung bei Underflow

    fipref('DataTypeOverride', 'ForceOff', 'LoggingMode','on'); % Loggingmode einschaten

    %Wenn nicht alle Eigabeparameter eingegen worden sind
    if nargin == 0
        N_ADC = 15;
        N_LUT = 16;
    elseif nargin ~=2
        error('wrong␣number␣of␣input␣arguments');
    end

    % Verwendete fimath Eigenschaften
    F = fimath;
    F.ProductMode              = 'KeepLSB';
    F.ProductWordLength        = 32;
    F.MaxProductWordLength     = 32;
    F.ProductFractionLength    = 0;
    F.SumMode                  = 'KeepLSB';
    F.SumWordLength            = 32;
    F.SumFractionLength        = 0;
    F.OverflowMode             = 'wrap';
    F.RoundMode                = 'floor';
    F.CastBeforeSum            = false;

    % Definition Datentypen
    long   = numerictype('Signed',true,'WordLength',32,'FractionLength',0, 'Scaling' , 'BinaryPoint'); %32 Bit mit Vorzeichen ohne Kommastellen
    long64 = numerictype('Signed',true,'WordLength',64,'FractionLength',0, 'Scaling' , 'BinaryPoint'); %64 Bit mit Vorzeichen ohne Kommastellen
    short  = numerictype('Signed',true,'WordLength',16,'FractionLength',0, 'Scaling' , 'BinaryPoint'); %16 Bit mit Vorzeichen ohne Kommastellen
    bit24  = numerictype('Signed',true,'WordLength',24,'FractionLength',0, 'Scaling' , 'BinaryPoint'); %16 Bit mit Vorzeichen ohne Kommastellen
```

```
56      char    = numerictype('Signed',true,'WordLength', 8,'FractionLength',0, 'Scaling' , '
            BinaryPoint'); %8 Bit mit Vorzeichen ohne Kommastellen
57      tADC    = numerictype('Signed',false,'WordLength',N_ADC,'FractionLength',0, 'Scaling' ,
            'BinaryPoint');
58      tLUT    = numerictype('Signed',true,'WordLength',N_LUT +1,'FractionLength',0, 'Scaling'
            , 'BinaryPoint'); % + 1 Bit fürs Vorzeichen entspricht jetzt Angabe in C-Quellcode
59
60      % Quantizer Objekte definieren um LSB beim Schieben zu runden
61
62      q16 = quantizer('datamode','fixed','format',[16,0],'overflowmode',...
63      'wrap','roundmode','convergent');
64      q32 = quantizer('datamode','fixed','format',[32,0],'overflowmode',...
65      'wrap','roundmode','convergent');
66
67      % Datentypen in Struktur ablegen zum besseren Handling
68      erg = struct('Properties', {F} , 'long', {long}, 'long64', {long64},...
69          'bit24', {bit24}, 'short', {short},'char', {char}, 'tADC', {tADC}, 'tLUT', {tLUT
                },...
70          'N_LUT', {N_LUT}, 'N_ADC', {N_ADC}, 'qu16' , {q16} , 'qu32', {q32});
71  end
```

demonstrator_algorithm

Listing F.22:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation des Radmessplatzen und Vergleich mit berechneten Werten von
% Demonstrator
%
% Verwendung von Integer Werten
%
% Version 1.1
%
%
% Datei:              demonstrator algorithm.m
%
%
% erstellt von: Lennart Koch
% erstellt am:  17.11.2009
%
% v1.1: Neue THD Berechnungsmethode hinzugefügt THD_fix_new
% v1.2: 2. Möglichkeit NIHD- Berechnung
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 function demonstrator_algorithm (name, ParameterFile)
% demonstrator_algorithm (name, ParameterFile)
% Funktion simuliert die Demonstrator Hardware von Herrn Jegenhorst und gibt
% berechneten THD aus. Auüerdem wird der Name der Simulationsreihe
% Festgelegt und die Ergebnisse der Simulation selbstständig abgespeichert.

% Simulation der Fälle
% - Aufnahme der Daten mit demonstrator, Auswertung mit eigener Hardware
% - Aufnahme der Daten mit Oszilloskop, Auswertung mit eigener Hardware
%
%Vergleich mit den Fällen (THD_calculation)
% - Aufnahme der Daten mit demonstrator, Auswertung mit Matlab
% - Aufnahme der Daten mit Oszilloskop, Auswertung der Daten mit Matlab
%
    %Eingabedaten
    % Name:          : Name der Parameter Datei
    %Parameter File : Pfad einer Parameter Datei


    %% Übergabeparameter aus Datei lesen, wenn icht vorhanden mit default Werten füllen

    s = fileread(ParameterFile);

    wh = what;%aktuellen Pfad ermitteln

    disp(wh.path);

   % String in Variablen umwandeln
   eval(s);


   %Abfragen, ob alle Variablen vorhanden sind

   if exist('Nb_LUT', 'var') == 0
       Nb_LUT = 10;     % Default value
   end

   if exist('Nb_ADC', 'var') == 0
       Nb_ADC = 12;     % Default value
   end
```

```
60
61
62      if exist('N_Sample', 'var') == 0
63         N_Sample = 64;      % Default value
64      end
65
66      if exist('No', 'var') == 0
67         No = 5;      % Default value
68      end
69
70      if exist('wordlength', 'var') == 0
71         wordlength = 'long';      % Default value
72      end
73
74      if exist('use_data', 'var') == 0
75         use_data = 'demo';      % Default value
76      end
77
78      if exist('use_table', 'var') == 0
79         use_table = 2;      % Default value
80      end
81
82       if exist('noise_inc_var', 'var') == 0
83         noise_inc_var = 1;      % Default value
84      end
85
86
87
88
89
90      %% Datei mit Messdaten laden
91
92     % Aufbereitete Daten aus Datei laden
93       pwd;
94 %      Datei in  Auswahl Dialogbox auswälen und dann laden
95       [FileName,PathName] = uigetfile('*_for_simulation.mat','Bitte␣eine␣Messreihe␣auswählen
             ');
96
97      if isequal(FileName, 0)
98          error('No␣file␣selected');
99      else
100         disp(['selected␣file:␣' FileName]);
101     end
102
103     FileNameComp=fullfile(PathName, FileName);
104
105     load(FileNameComp);
106     %% Pfade festlegen
107
108     date_time = clock; % date_time = [year month day hour minute seconds]
109
110     % Verzeichnis für neue Simulationen
111
112     save_folder = 'D:/simulation_folder';
113
114     lfdnr = [num2str(date_time(1),'%d')  num2str(date_time(2),'%02d') num2str(date_time(3),
            '%02d')...
115         num2str(date_time(4),'%02d') num2str(date_time(5),'%02d')];
116
117     foldername = [lfdnr '_' name];
118
119
120     mkdir(save_folder, foldername);
```

```
121
122     pathname = [save_folder '/' foldername];
123     %Nummerindex für Simulationsdateien setzt sich aus Datum zusammen
124
125      LM = length(data);
126     N_harm = data(1,1).N;
127     part_number = 1;
128
129
130
131      %% Überprüfung der eingegebenen Daten
132
133     if min(Nb_LUT) < 1 || max(Nb_LUT) > 16    %Erlaubter Bereich: 1..16 Bit
134         error('Parameter␣Nb_LUT␣nur␣zwischen␣1␣und␣16␣Bit␣Wortbreite␣erlaubt');
135     end
136
137     if min(Nb_ADC < 1) || max(Nb_ADC > 16)    %Erlaubter Bereich: 1..16 Bit
138        error('Parameter␣Nb_ADC␣nur␣zwischen␣1␣und␣16␣Bit␣Wortbreite␣erlaubt');
139     end
140
141     if min(No) < 2 || max(No) > N_harm    %Erlaubter Bereich: 1..N_harm
142       error('Parameter␣Nb_LUT␣nur␣zwischen␣1␣und␣16␣Bit␣Wortbreite␣erlaubt');
143     end
144
145     if min(N_Sample) < 16
146          error('Parameter␣N_Sample␣zu␣klein');
147     end
148
149      if strcmp(use_data ,'demo')== 0 && strcmp(use_data ,'scope') == 0   %Erlaubter Bereich
             : 1..N_harm
150        error('datasource␣not␣avalueable');
151      end
152
153
154      if strcmp(wordlength ,'long')== 0 && strcmp(wordlength ,'short')== 0 &&  strcmp(
             wordlength ,'24Bit') == 0   %Erlaubter Bereich: 1..N_harm
155        error('wordlength␣option␣not␣avalueable');
156     end
157
158
159     if use_table < 1 || use_table > 4    %Erlaubter Bereich: 1..4 Bit
160        error('sqrt␣Table␣not␣avalueable');
161     end
162
163     if noise_inc_var < 1 || noise_inc_var > 2    %Erlaubter Bereich: 1..4 Bit
164        error('calculation␣option␣not␣avalueable');
165     end
166
167     %Überprüfen, ob mehrere Simulationsreihen durchgeführt werden sollen
168
169     if length(Nb_LUT) > 1 || length(Nb_ADC) > 1 || length(No) > 1 || length(N_Sample) > 1
170         visible = 'off';      % Es werden keine Plots angezeigt
171     else
172         visible = 'on';
173     end
174
175
176     %% Variablen festlegen
177
178
179     % Initalisierung der nicht Fixpoint Vektoren, nur Abhängigkeit von LM
180     distance = zeros(1,LM);          %Vektor für Distanz
181     THD_lut = zeros(1,LM);           % gemessen an Radmessplatz
```

```
182
183       for i = 1:LM
184           distance(i) = data(1,i).distance;                % Distanz einlesen
185           THD_lut(i) = data(1,i).measure_rmp.hd_lut;       % THD mit aus Lookup Table einlesen
186       end
187
188      for ns = 1:length(N_Sample) % Simulation für mehrere Samples
189          S = N_Sample(ns);
190
191          for n_lut = 1:length(Nb_LUT)
192              NLUT = Nb_LUT(n_lut);
193
194              %Berechnung LUT SInus und Kosinusfunktionen ( Real und Imaginärteil eines
                     Zeigers
195              wT = 0:2*pi/N_Sample(ns):(N_Sample(ns)-1)* 2* pi / N_Sample(ns);
196
197              sin_fix = ((2^(NLUT) -1) * (-sin(wT)));   %geändert 26.11.2009
198              cos_fix = ((2^(NLUT) -1) *   cos(wT));    % geändert 26.11.2009
199
200
201              for n_adc = 1:length(Nb_ADC)
202                  NADC = Nb_ADC(n_adc);
203                  fixp = define_Types(NADC, NLUT); % Datentypen erstellen
204
205                  %Sinus Cosinus FIxpoint Variablen erstellen
206                  sinx = fi(sin_fix, fixp.tLUT, fixp.Properties); % etwas andere Skallierung
                         als in C- Code
207                  cosx = fi(cos_fix, fixp.tLUT, fixp.Properties);   % etwas andere
                         Skallierung als in C- Code
208
209
210                  switch (use_data)
211                      case 'scope'
212                          disp('---------- you have selected option "scope" ----------');
213                          [s_fix gain_vec]= calc_function(data,fixp,N_Sample(ns), 'scope');
214                      case 'demo'
215                          disp('---------- you have selected option "demo" ----------');
216                          [s_fix gain_vec]= calc_function(data,fixp,N_Sample(ns), 'demo');
217                      otherwise
218                          % Fehlermeldung log Datei
219                          fprintf(fid, 'Diese Daten sind nicht verfügbar, bitte andere option
                                 wählen\n');
220                          error('This option is not available!');
221                  end
222
223
224
225  %% Log Datei erstellen
226
227      %File- Pointer erstellen
228      fid = fopen([pathname '/' foldername '_simulation_parmeters.txt'], 'wt');
229      fprintf(fid, '************************************************************\n');
230      fprintf(fid, '*             Simulation Radmessplatz                      *\n');
231      fprintf(fid, '************************************************************\n\n');
232
233      fprintf(fid, 'Messung durchgeführt am: %02d.%02d.%d um %02d:%02d Uhr\n\n', date_time
             (3),...
234                      date_time(2), date_time(1), date_time(4) , date_time(5));
235
236      fprintf(fid, 'verwendete Parameter\n');
237      fprintf(fid, 'Name der Parameterdatei          : %s\n', ParameterFile);
```

```
238     fprintf(fid, 'Name␣der␣Messreihe␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣%s\n', FileName);
239     fprintf(fid, 'verwendete␣Messdaten␣␣␣␣␣␣␣␣␣␣␣␣␣:␣%s\n', use_data);
240     fprintf(fid, 'Samples␣pro␣Periode␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣%d\n', N_Sample);
241     fprintf(fid, 'berücksichtige␣Koeffizienten␣␣␣␣␣␣:␣%d\n', No);
242     fprintf(fid, 'Auflösung␣Look-up␣Table␣für␣DFT␣␣␣:␣%d␣Bit\n' ,Nb_LUT);
243     fprintf(fid, 'Auflösung␣Analog-Digital-Umsetzer␣:␣%d␣Bit\n', Nb_ADC);
244     fprintf(fid, 'verwendete␣Look-up␣Tabelle␣␣␣␣␣␣␣:␣');
245     if use_table == 1
246         fprintf(fid, 'original␣Look-up␣Tabelle␣des␣Radmessplatzes\n');
247     elseif use_table == 2
248         fprintf(fid, 'PCM␣Codierung␣9␣Stufen\n');
249     elseif use_table == 3
250         fprintf(fid, 'PCM␣Codierung␣8␣Stufen\n');
251     elseif use_table == 4
252         fprintf(fid, 'verwendung␣mehrerer␣Tabellen␣72␣Werte\n');
253     end
254     fprintf(fid, 'Berechnungsmöglichkeit␣HDI-Methode:␣');
255     if noise_inc_var == 1
256         fprintf(fid, 'Berechnung␣Gesamtleistung␣ohne␣Gleichanteil\n');
257     elseif noise_inc_var == 2
258         fprintf(fid, 'Berechnung␣Gesamtleistung␣mit␣Gleichanteil\n');
259     end
260     fprintf(fid, 'Wortbreite␣der␣Variablen␣␣␣␣␣␣␣␣␣:␣%s\n\n\n', wordlength)
261
262     fprintf(fid, '************************************************************\n');
263     fprintf(fid, '*␣␣␣␣␣␣␣␣␣␣␣Eigenschaften␣des␣Eingangssignals␣␣␣␣␣␣␣␣␣␣␣*\n');
264     fprintf(fid, '************************************************************\n\n');
265
266     if strcmp(use_data ,'scope')
267         fprintf(fid, 'Es␣wird␣das␣aufgezeichnete␣Signal␣des␣Oszilloskops␣verwendet\n');
268         fprintf(fid, 'verwendete␣Parameter\n');
269
270         fprintf(fid, 'Abtastfrequenz␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣%d␣Hz\n', parameters.samplerate
                );
271         fprintf(fid, 'Anzahl␣der␣Harmonischen␣␣␣␣␣␣␣␣␣␣␣:␣%d\n', N_harm );
272         fprintf(fid, 'Perioden␣über␣die␣FFT␣berechnet␣wird:␣%d\n', data(1,1).measure_scope.
                periodes );
273
274     elseif (strcmp(use_data ,'demo')) && (isempty(findstr(FileName, '_rmp_sim'))) == false
                %Radmessplatz Simulation
275         fprintf(fid, ['Es␣wird␣eine␣nachgebildete␣Unterabtastung␣aus␣den␣Messergebnissen␣
                des\n' ...
276                       'Oszilloskops␣verwendet.␣Dazu␣sind␣untenstehende␣Parameter␣verwendet␣
                           worden\n\n']);
277
278         fprintf(fid, 'verwendete␣Parameter\n');
279
280         fprintf(fid, 'Anzahl␣der␣berücksichtigten␣Harmonischen␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣%d\n',
                data(1,1).N );
281         fprintf(fid, 'Aufnamereihenfolge␣der␣Messdaten␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣');
282         if simulation_parameters.zufall == 1
283             fprintf(fid, 'zufällig\n');
284         else
285             fprintf(fid, 'der␣Reihe␣nach\n');
286         end
287         fprintf(fid, 'minimal␣verstreihende␣Perioden,␣bis␣Wert␣aufgenommen␣wird:␣%d\n',
                simulation_parameters.min_tw);
288         fprintf(fid, 'maximal␣verstreihende␣Perioden,␣bis␣Wert␣aufgenommen␣wird:␣%d\n',
                simulation_parameters.max_tw);
289
290     else                                                          %Original
            Daten des Radmessplatzes
291        fprintf(fid, 'Es␣werden␣die␣Messwerte␣des␣Demonstrators␣verwendet.\n');
```

```
292         fprintf(fid, ['Nähere Informationen zum verwendeten Abtastverfahren sind in der\n'
                ...
293                       'Diplomarbeit Jegenhorst nachzulesen.\n']);
294
295         fprintf(fid, 'verwendete Parameter\n');
296         fprintf(fid, 'Anzahl der berücksichtigiten  Harmonischen                : %d\n',
                data(1,1).N );
297
298
299     end
300
301
302
303     fprintf(fid, '\n\n************************************************************\n\n');
304     fprintf(fid, 'Ergebnis der Simulation: \n\n');
305
306
307
308                  %% Durchführung der Simulation
309
310                 for coeff = 1:length(No)
311                     NO = No(coeff);
312
313                     % Felder deren Grösse von No abhängig ist,
314                     % initalisieren
315                     re_calc = fi(zeros(NO,LM),fixp.long,fixp.Properties);
316                     im_calc = fi(zeros(NO,LM),fixp.long,fixp.Properties);
317                     mag_calc = fi(zeros(NO,LM),fixp.long,fixp.Properties);
318                     THD_calc = fi(zeros(1,LM),fixp.char,fixp.Properties);
319                     THD_new = fi(zeros(1,LM),fixp.char,fixp.Properties);
320                     Pges = fi(zeros(1,LM),fixp.long,fixp.Properties);
321                     P1 = fi(zeros(1,LM),fixp.long,fixp.Properties);
322
323
324                      %DFT Matrix berechnen
325                      DFT_real = fi(zeros(NO,S), fixp.tLUT, fixp.Properties); %16 Bit für
                             Realteil geändert 19.11
326                      DFT_imag =  fi(zeros(NO,S), fixp.tLUT, fixp.Properties); %16 Bit für
                             Imaginärteil geändert 19.11
327
328
329                      for k = 1:NO          %für k = 1..K -> Gleichanteil nicht berechnen
330                         for n = 0:S-1   %für alle Samples von x_t
331                             kn = mod(k*n,S); %Produkt k * n berechnen, mod da Peridische
                                    Funktion und sin(k*n) = sin(k*n + N)
332                             DFT_real(k,n+1) = cosx(kn +1);        % Realantei des Zeigers
                                    ausrechnen
333                             DFT_imag(k,n+1) = sinx(kn +1);        % Imaginärteil des Zeigers
                                    bestimmen
334                         end
335                      end
336
337                      % THD über alle Schwingungen THD nur von
338                        % berücksichtigten Schwingungen sowie Real- und
339                        % Imaginärteile mit Matlab berechnen
340
341                      [THD5_mat, THD_mat, THD_new_mat , re_mat, im_mat, mag_mat] =
                             THD_calculation(s_fix.double, gain_vec, NO, LM);
342
343                         for i = 1:LM
344                             disp(['------------------ calculting ' num2str(i) ' of '
                                    num2str(LM) ' ------------------']);
345                            %------------------ THD Berechnung aus DFT ----------------
```

```
346                     [re_calc(:,i) im_calc(:,i) mag_calc(:,i)] = DFT_quant_int_V3(
                            s_fix(i,:),fixp, DFT_real, DFT_imag, wordlength, No);
                            % DFT errechnen
347                     THD_calc(i) = THD_fix_int( mag_calc(:,i),fixp, wordlength,
                            use_table); % THD errrechnen
348                     mag_calc(:,i) = bitshift( mag_calc(:,i),-(data(1,i).
                            shift_factor));
349                     % Shift_fac bei Signalaufnahme als analoger Verstärker
                            hinzugefügt
350                     re_calc(:,i) = bitshift(re_calc(:,i), -(data(1,i).shift_factor)
                            );
351                     im_calc(:,i) = bitshift(im_calc(:,i), -(data(1,i).shift_factor)
                            );
352
353                     %------------------HDI-Methode ------------
354                     if noise_inc_var == 1
355                         [THD_new(i) P1(i) Pges(i)] = THD_fix_newV3(s_fix(i,:),fixp,
                                DFT_real(1,:),DFT_imag(1,:),wordlength, use_table);
356                     elseif noise_inc_var == 2
357                         [THD_new(i) P1(i) Pges(i)] = THD_fix_newV4(s_fix(i,:),fixp,
                                DFT_real(1,:),DFT_imag(1,:),wordlength, use_table);
358                     end
359
360                     % Shift_fac bei Signalaufnahme als analoger Verstärker
                            hinzugefügt
361                      Pges(i) = bitshift(Pges(i), -(data(1,i).shift_factor));
362                      P1(i) = bitshift(P1(i), -(data(1,i).shift_factor));
363                 end
364
365                 dest_file = ['Part_' num2str(part_number) '_NLUT=' num2str(NLUT) '
                        _No=' num2str(NO)...
366                              '_NADC=' num2str(NADC) '_Samples=' num2str(S)]; %
                                    Dateiname für Matlabfiles
367
368                  part_number = part_number +1;
369                  fprintf(fid, 'Simulation erfolgreich durchgeführt! Daten können
                        unter dem Namen %s.sim erneut geöffnet werden\n', dest_file);
370
371             %% Ergebnisse Plotten und auf der Festplatte sichern
372              % Bildschirmgröße bestimmmen um Grafiken im
373              % Vollbildmodus anzuzeigen
374              scrsz = get(0,'ScreenSize');
375
376
377
378             mkdir(pathname, 'Images'); %Vezeichnis für Bilder erstellen
379
380             impath = [pathname '/Images/'];
381             disp(impath);
382             cd(impath);
383
384                % Darstellung Betrag, Real- und Imaginärteil von Hardware
385                % berechnet und simuliert
386                h = figure('Name',['Parameters: NLUT=' num2str(NLUT) ',NADC='
                        num2str(NADC) 'Koeff=' num2str(NO)...
387                     ',Samples=' num2str(S) 'stepsize=' num2str(distance(2)-
                            distance(1)) 'mm'], 'Position', scrsz, 'visible', visible
                            );
388                subplot(2,3,1);
389                plot(distance, mag_mat);
390                grid on;
391                ylabel('Magnitudes of harmonics');
392                xlabel('distance [mm]');
```

```
393                     title('Magntudes calculated with Matlab');
394                     subplot(2,3,2);
395                     plot(distance, re_mat);
396                     grid on;
397                     ylabel('Real Parts of harmonics');
398                     xlabel('distance [mm]');
399                     title('Real Parts calculated with Matlab');
400                     subplot(2,3,3);
401                     plot(distance, im_mat);
402                     grid on;
403                     ylabel('Imaginary Parts of harmonics');
404                     xlabel('distance [mm]');
405                     title('Imaginary Parts calculated with Matlab');
406
407
408                     subplot(2,3,4);
409                     plot(distance,mag_calc);
410                     grid on;
411                     ylabel('Magnitudes of harmonics');
412                     xlabel('distance [mm]');
413                     title('simulated Magnitudes');
414                     subplot(2,3,5);
415                     plot(distance,re_calc);
416                     grid on;
417                     ylabel('Real Parts of harmonics');
418                     xlabel('distance [mm]');
419                     title('simulated Real Parts');
420                     subplot(2,3,6);
421                     plot(distance,im_calc);
422                     grid on;
423                     ylabel('Imaginary Parts of harmonics');
424                     xlabel('distance [mm]');
425                     title('simulated Imaginary Parts');
426
427                     %Plot Speichern
428                     set(h,'PaperPositionMode','manual');
429                     set(h, 'PaperUnits', 'centimeters');
430                     set(h, 'PaperType', 'A4');
431                     set(h, 'PaperOrientation', 'landscape');
432
433                     filename = [dest_file '_Magnitudes_Real_and_Imaginary_Parts'];
434                     orient landscape
435                     saveas(h, filename, 'fig');
436                     saveas(h, filename, 'pdf');
437                     system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
438                     saveas(h, filename, 'jpg');
439
440                     % Darstellung Leistung des Gesamtsignals und der 1 Harmonischen
                            Schwingung
441                      h = figure('Name','Power of Signal and Power of first harmonic','
                            visible', visible);%, 'Position', scrsz);
442                      plot(distance, Pges,'-', distance, P1,'--');
443                      grid on;
444                      ylabel('Power(P)');
445                      xlabel('distance [mm]');
446                      legend('Power of Signal', 'Power of first harmonic');
447                      title(['Power of Signal, step size: ' num2str(distance(2)-distance
                            (1)) 'mm']);
448
449                      %Plot Speichern
450                     set(h,'PaperPositionMode','manual');
451                     set(h, 'PaperUnits', 'centimeters');
452                     set(h, 'PaperType', 'A4');
```

```
453                 set(h, 'PaperOrientation', 'landscape');
454                 orient landscape
455
456                 filename = [dest_file '_power_of_signals'];
457                 saveas(h, filename, 'fig');
458                 saveas(h, filename, 'pdf');
459                 system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
460                 saveas(h, filename, 'jpg');
461
462                 % Darstellung THD über Distanz
463                 h = figure('Name',['Parameters: NLUT=' num2str(NLUT) ',NADC='
                        num2str(NADC) 'Koeff=' num2str(NO) ...
464                      ',Samples=' num2str(S)], 'Position', scrsz, 'visible', visible
                           );
465                 plot(distance, THD_mat,'--', distance, THD_lut ,'-', distance,
                        THD5_mat,'--', distance, THD_calc, '-', distance, THD_new_mat,
                        '--', distance, THD_new ,'-');
466                 grid on;
467                 ylabel('Harmonic Distortion (HD) [%]');
468                 xlabel('distance [mm]');
469                 legend('THD with all Harmonics caluclated with floating-point
                        arithmetic',...
470                      'THD calculated with MSP430 Hardware',...
471                      ['reduced THD with ' num2str(NO) ' coefficients calculated with
                           floatingpoint arithmetic'], ...
472                      ['reduced THD with ' num2str(NO) ' coefficients calculated with
                           fixpoint arithmetic'],...
473                      'HDI-method with floatingpoint arithmetic',...
474                      'HDI-method with fixpoint arithmetic');
475                 title(['Total Harmonic Distortion (THD) over distance, step size: '
                         num2str(distance(2)-distance(1)) 'mm']);
476
477                 %Plot speichern
478                 set(h,'PaperPositionMode','manual');
479                 set(h, 'PaperUnits', 'centimeters');
480                 set(h, 'PaperType', 'A4');
481                 orient landscape
482
483                 filename = [dest_file '_THD_over_distance'];
484                 saveas(h, filename, 'fig');
485                 saveas(h, filename, 'pdf');
486                 system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
487                 saveas(h, filename, 'jpg');
488
489
490                 % Darstellung der Fehler von THD5_mat, THD_calc
491                 [THD5_relerr THD5_abserr] = fehler(THD_calc.double, THD5_mat);
492
493                 h=figure('Name','relative and absolute error of approximated THD
                        with floating point arithmetic and approximated THD with
                        fixpoint aritmetic', 'Position', scrsz ,'visible', visible);
494                 subplot(1,2,1)
495                 plot(distance,THD5_relerr);
496                 grid on;
497                 ylabel('relative error [%]');
498                 xlabel('distance [mm]');
499                 title('relative error of approximated HD calculated fixpoint-
                        arithmetic');
500                 subplot(1,2,2)
501                 plot(distance,THD5_abserr);
502                 grid on;
503                 ylabel('absolute error');
504                 xlabel('distance [mm]');
```

```
505                     title('absolute error of approximated HD calculated fixpoint-
                            arithmetic');
506
507
508                     set(h,'PaperPositionMode','manual');
509                     set(h, 'PaperUnits', 'centimeters');
510                     set(h, 'PaperType', 'A4');
511                     orient landscape
512
513                     filename = [dest_file '_error_approximated_HD'];
514                     saveas(h, filename, 'fig');
515                     saveas(h, filename, 'pdf');
516                     system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
517                     saveas(h, filename, 'jpg');
518
519
520                     % Darstellung der Fehler von THD5_mat, THD_lut
521                     [THDlut_relerr THDlut_abserr] = fehler(THD_lut, THD5_mat);
522
523                     h=figure('Name','relative and absolute error of hd calculated by
                            demonstrator' ,'Position', scrsz, 'visible', visible);
524                     subplot(1,2,1)
525                     plot(distance,THDlut_relerr);
526                     grid on;
527                     ylabel('relative error[%]');
528                     xlabel('distance [mm]');
529                     title('relative error of hd calculated by demonstrator');
530                     subplot(1,2,2)
531                     plot(distance,THDlut_abserr);
532                     grid on;
533                     ylabel('absolute error');
534                     xlabel('distance [mm]');
535                     title('absolute error of hd calculated by demonstrator');
536
537                     set(h,'PaperPositionMode','manual');
538                     set(h, 'PaperUnits', 'centimeters');
539                     set(h, 'PaperType', 'A4');
540                     orient landscape
541
542                     filename = [dest_file '_error_THD_demonstrator'];
543                     saveas(h, filename, 'fig');
544                     saveas(h, filename, 'pdf');
545                     system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
546                     saveas(h, filename, 'jpg');
547
548
549                     [THDcalc_relerr THDcalc_abserr] = fehler(THD_new.double, THD_mat);
550                     h=figure('Name','relative and Abosolute error of HDI-method' ,'
                            Position', scrsz, 'visible', visible);
551                     subplot(1,2,1)
552                     plot(distance,THDcalc_relerr);
553                     grid on;
554                     ylabel('relative error [%]');
555                     xlabel('distance [mm]');
556                     title('relative error of HDI-method calculated with fixpoint-
                            arithmetic');
557                     subplot(1,2,2)
558                     plot(distance,THDcalc_abserr);
559                     grid on;
560                     ylabel('absolute error');
561                     xlabel('distance [mm]');
562                     title('absolute error of HDI-method calculated with fixpoint-
                            arithmetic');
```

```
563
564                             set(h,'PaperPositionMode','manual');
565                             set(h, 'PaperUnits', 'centimeters');
566                             set(h, 'PaperType', 'A4');
567                             orient landscape
568
569                             filename = [dest_file '_error_HDI_method'];
570                             saveas(h, filename, 'fig');
571                             saveas(h, filename, 'pdf');
572                             system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
573                             saveas(h, filename, 'jpg');
574
575                             [THDcalc_relerr THDcalc_abserr] = fehler(THD5_mat, THD_mat);
576                             h=figure('Name','relative and absolute error of different
                                    implementations' ,'Position', scrsz, 'visible', visible);
577                             subplot(1,2,1)
578                             plot(distance,THDcalc_relerr);
579                             grid on;
580                             ylabel('relative error [%]');
581                             xlabel('distance [mm]');
582                             title('relative error');
583                             subplot(1,2,2)
584                             plot(distance,THDcalc_abserr);
585                             grid on;
586                             ylabel('absolute error');
587                             xlabel('distance [mm]');
588                             title('absolute error of different implementations');
589
590                             set(h,'PaperPositionMode','manual');
591                             set(h, 'PaperUnits', 'centimeters');
592                             set(h, 'PaperType', 'A4');
593                             orient landscape
594
595                             filename = [dest_file '_error_different_implementations'];
596                             saveas(h, filename, 'fig');
597                             saveas(h, filename, 'pdf');
598                             system(['pdfcrop ' filename '.pdf ' filename '_cropped.pdf']);
599                             saveas(h, filename, 'jpg');
600                             cd(wh.path);          %wieder in Programnmverzeichnis wechseln
601
602
603 %% Daten auf Fesplatte sichern
604
605  save([pathname '/' dest_file '.sim.mat'], 'distance', 's_fix' ,  ...
606                                  'THD_lut', 're_calc', 'im_calc', 'mag_calc', 'mag_mat', '
                                        THD_calc', 'THD_new', 'DFT_real', 'DFT_imag' ,...
607                                 'THD_mat', 'THD5_mat','THD_new_mat', 're_mat', 'im_mat', '
                                        NO', 'NADC', 'NLUT', 'S', 'fixp' ,...
608                                 'P1', 'Pges', 'N_harm', 'parameters');
609                     end
610                 end
611
612             end
613
614         end
615         % Schreiben der Log-Datei abschließen
616         fprintf(fid, '************************************************************\n\n');
617         fclose(fid);
618 end
```

F.2. C Quellcodes

F.2.1. main.c

Listing F.23:

```
1  /*
2  ================================================================================
3  |       Projekt:                Diplomarbeit, Gesamtleistungsverfahren mit neuer
       Wurzeltabelle
4  |       Erstellt von:   Lennart Koch
5  |       Erstellt am:    09.02.2010
6  |       Geändert am:    31.03.2010
7  |       Hardware:               MSP430f169 auf Olimex eval board
8  |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
       v1.021
9  |       Beschreibung:   Ermittlung der Rechenzeit approximierte hd Berechnung, noise
       included
10 |                                       hd-Berechnung
11 |
12 |       Funktion:
13 |       Datei:                  main.c
14 ================================================================================
15 */
16
17 //#define EXTERN                      // globale Variablen hier definieren
18 #define LUT_SIN_DEF                  // SIN_LUT hier initialisieren
19 #define SIGNAL                       // Feld mit Samples hier initialisieren
20 #include "header_main.h"
21 #include "table_sin_red.h"
22 #include "globales.h"
23
24
25 /* Simulationsparameter:
26 * N_ADC = 12 Bit
27 * N_LUT = 10 Bit
28 * Samples = 64;
29 *
30 */
31
32
33 int main(void)
34 {
35         int i;
36         int j;
37
38         P3DIR = 0xFF;                                                    //
               P3 als Ausgang
39         LED1_ON;
                   // Start signalisieren
40         LED2_OFF;
41         init_TimerB(800);                                                //
               Timer einstellen
42         InitLCD();
                   // LCD Einstellungen festlegen
43         showString("B1->Jegenhorst\nB2->noise inc hd");
44         _EINT();                                                  // interrupt enable
45         while ( 1 ) {
46
47                 char tasteGedrueckt = taste();                    // Abfrage ob Taste
                       gedrückt
```

```
48                  if (tasteGedrueckt == 11) {                                     // Wenn
                        Taste B1 auf Eval Board
49                      TBCCTL0 &= ~CCIE;                                                   //
                            CCR0 interrupt disabled
50                      z =0;
                                // Zählvariable Z zurücksetzen
51                      LED1_OFF;
                                // Start signalisieren
52                      LED2_ON;
53
54                      TBR = 0x0000;                                                       //
                            Timerzähler auf 0 setzen
55                      TBCCTL0 = CCIE;                                                     //
                            CCR0 interrupt enabled
56                      /** Jegenhorst Implementierung **/
57
58                      for (j = 0; j<10;j++){
59                              cleanup();
60                              for (i = 0; i < 64; i++)                        //
                                    Simulation ADC Interrupt Routine
61                              {
62                                      g_adc.u_diff_b= sig[i];
63                                      g_calc.lut_pos += LUT_POS_INC;
64                                      calc_dft_sample();
65                              }
66                              calc_dft_abs();
67                              calc_hd();
68                      }
69                      TBCCTL0 &= ~CCIE;                                                   //
                            CCR0 interrupt disabled
70                      LED1_ON;
                                // Ende signalisieren
71                      LED2_OFF;
72                      showStringAndInt("Jegenhorst\nz=␣",z);  // Ausgabe Zählerstand
73
74                  }
75                  else if (tasteGedrueckt == 21) {                        // wenn Taste B2
76                      TBCCTL0 &= ~CCIE;                                                   //
                            CCR0 interrupt disabled
77                      z =0;
78
79                      LED1_OFF;
                                // Start signalisieren
80                      LED2_ON;
81                      TBR = 0x0000;                                                       //
                            Timerzähler auf 0 setzen
82                      TBCCTL0 = CCIE;                                                     //
                            CCR0 interrupt enabled
83                      /** HDI-Methode **/
84
85                      for (j = 0; j<10;j++){
86                              cleanup();
87                              for (i = 0; i < 64; i++)                        //
                                    Simulation ADC Interrupt Routine
88                              {
89                                      g_adc.u_diff_b= sig[i];
90                                      calc_HD_noise_sampling();
91                              }
92                              calc_HD_noise_afer();
93                              decode_LUT();                                       //
                                    Ergebnis aus Tabelle ermitteln
94                      }
```

```
95                              TBCCTL0 &= ~CCIE;                                       // CCR0
                                    interrupt disabled
96                              LED1_ON;
                                    // Ende signalisieren
97                              LED2_OFF;
98                              showStringAndInt("THD␣noise␣inc.\nz=␣",z); // Ergebnis ausgeben
99
100                         }
101             }// Ende Wählschleife
102             return 0;
103
104     }
105
106
107     interrupt (TIMERB0_VECTOR) Timer_B(void) {
108             z ++;
109     }|
```

F.2.2. hdnoi.c

Listing F.24:

```
1  /*
2  ================================================================================================
3  |       Projekt:                Diplomarbeit
4  |       Erstellt von:   Lennart Koch
5  |       Erstellt am:    30.06.2009
6  |       Geändert am:    04.03.2010
7  |       Hardware:               MSP430f169 auf Olimex eval board
8  |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
       v1.021
9  |       Beschreibung:   Implementierung noise-included hd Verfahren für Controller s. Doku
10 |
11 |       Funktion:               OK
12 |       Datei:                  hdnoi.c
13 ================================================================================================
14 */
15
16 #include "header_main.h"
17 #include "table_sin_red.h"
18 #include "globales.h"
19
20 void calc_HD_noise_sampling(void){
21         int16_t j;
22         // Berechnung von Gleichanteil und 1 Harmonischer während der Datenaufnahme
23         // Gleichanteil;
24
25         // max. Wortbreite überprüfen
26         // w_ADC * ld(64) = 12 + 6 = 18 Bit also 32 Bit Register ausreichend
27         THD_calc.x_gl += (int32_t) g_adc.u_diff_b;      //Summe aller Samples
28
29         // Leistung Gesamtsignal mit Gleichanteil
30         // max Wortbreite = 2 * w_ADC + ld(64) = 24 + 6 = 30 => 32 Bit ausreichend
31
32         THD_calc.x0_quad = (int32_t) g_adc.u_diff_b * g_adc.u_diff_b;
33         THD_calc.l_ges += THD_calc.x0_quad;
34
35         //1. Harmonische, Real und Imaginärteil
36
37         // max.Wortbreite überprüfen
38         // w_ADC + w_LUT +ld(64) = 12 + 10 + 6 = 28 Bit => 32 Bit Register ist ausreichend
39
40         // Realteil errechnen
41         THD_calc.prod = (int32_t) g_adc.u_diff_b * table_sin[(THD_calc.lut_pos + IDX_PI_H)
               % 32];
42         j = THD_calc.lut_pos >> 4;
                   // Periode in 4 Teile unterteilen für cos
43         if ((j == 0) || (j == 3))
                   // wenn zwischen 0 und pi/2 oder zwischen 3pi/2 und pi
44                 THD_calc.real += THD_calc.prod;                                         //
                       Kosinus ist positiv
45         else
46                 THD_calc.real -= THD_calc.prod;                                         //
                       Kosinus ist negativ
47
48         // Imaginärteil errechnen
49         THD_calc.prod = (int32_t) g_adc.u_diff_b * table_sin[THD_calc.lut_pos % 32];
50         if ((j>> 1) == 1)
                   // Periode in 2 Teile aufteilen für Sinus Bestimmung
51                 THD_calc.imag += THD_calc.prod;                                         //
                       da - sin(x) gerechnet werden muss
```

```
52          else
53                  THD_calc.imag -= THD_calc.prod;                                 //
                        da -sin(x) gerechnet werden muss
54
55          THD_calc.lut_pos++;
                            //Position der LUT inkrementieren
56  }
57
58  void calc_HD_noise_afer(void){
59          THD_calc.x_gl >>= 3;
                        // 12 +6 Bit - 15 Bit damit für Quadrierung 15 Bit lang
60          THD_calc.l_ges >>= SHIFT_N;
                        // Division Leistung  /N
61
62          THD_calc.x0 = (int16_t) THD_calc.x_gl;
63          THD_calc.x0_quad = (int32_t) THD_calc.x0 * THD_calc.x0;
64
65          THD_calc.x0_quad >>= 6;
                        // 2 * (SHIFT_N - 3)
66
67          THD_calc.l_ges -= THD_calc.x0_quad;                                     //
                Subtraktion des Quadrads des Gleichanteils
68
69          //Leistung der 1. Harmonischen aus DFT Ergebnis errechnen
70          THD_calc.real = THD_calc.real >> SHIFT_QUAD;                    // Realteil auf
                untere 16 Bit schieben
71          THD_calc.x0 = (int16_t) THD_calc.real;
72          THD_calc.real_quad = (int32_t)THD_calc.x0 * THD_calc.x0;// Realteil quadrieren
73          THD_calc.imag = THD_calc.imag >> SHIFT_QUAD;                    // Imaginärteil auf
                 untere 16 Bit schieben
74          THD_calc.x0 = (int16_t) THD_calc.imag;
75          THD_calc.imag_quad = (int32_t)THD_calc.x0 * THD_calc.x0;// Imaginärteil quadrieren
76          THD_calc.l1 = THD_calc.real_quad + THD_calc.imag_quad;
77          THD_calc.l1 = THD_calc.l1 >> (SHIFT_N +1);                              // Division
                 /N+ 1 wg. Addition
78
79          //Leistung aller Oberschwingungen errechnen
80          if (THD_calc.l_ges >  THD_calc.l1){
81                  THD_calc.P_ob = THD_calc.l_ges - THD_calc.l1;           // Leistung
                        Oberschwingungen ermitteln
82                  THD_calc.factor = (uint64_t) (THD_calc.P_ob << 14) / THD_calc.l_ges;
83          }
84          else {
                                    // wenn Leist. 1. Harmonische größer Signalleistung
85                  THD_calc.factor = 0 ;
                            // Klirrfaktor = 0
86                  THD_calc.P_ob = 0;
                                    // Leistung Oberwellen = 0
87          }
88
89  }
```

F.2.3. sqrt.c

Listing F.25:

```
1  /*
2  ================================================================================
3  |       Projekt:                Diplomarbeit, C Implementierung
4  |       Erstellt von:   Lennart Koch
5  |       Erstellt am:    11.02.2010
6  |       Geändert am:    11.02.2010
7  |       Hardware:               MSP430f169 auf Olimex Evaluation board
8  |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
       v1.021
9  |       Beschreibung:   Verkürzte LUT Ermittlung der Wurzel eines Wertes
10 |
11 |       Funktion:               OK
12 |       Datei:                  table_sqrt.h
13 ================================================================================
14 */
15 #include "table_sqrt.h"
16 #include "header_main.h"
17 #include "globales.h"
18
19
20
21 void decode_LUT(void){
22         int spalte = 0;                                 // Spalte in der Tabelle
23         int zeile = 0;                                  // Zeile in der Tabelle
24         int temp  = MAX_POSIBLE;
25         uint32_t mask_zeile = 0x80000000; // max mögl. Bit
26         uint32_t mask_spalte = 0x70000000; //Maskierung folgende 3 Bit
27
28         do {
29                 if ((THD_calc.factor & mask_zeile) == mask_zeile){      // wenn bit gesetzt
                       ist
30                         if (temp > MAX_ZEILE){
                                        // wenn ermittelter Klirrfaktor > darstellbarer
                                    Bereich
31                                      zeile = MAX_ZEILE;
                                                        // Maximalwert auswählen
32                                      spalte = MAX_SPALTE;
33                         }
34                         else{
35                                      zeile = temp;
                                                //Zeile der Tabelle festlegen
36                                      spalte = (THD_calc.factor & mask_spalte)>> (temp -1); //
                                            Spalte der Tabelle festlegen
37                         }
38                 }
39                 mask_zeile = mask_zeile >> 1;                           // wenn maskiertes
                       Bit nicht gesetzt, Masken um eine Stelle nach links schieben
40                 mask_spalte = mask_spalte >> 1;
41                 temp--;
                        // Zähler dekrementieren
42         } while( zeile == 0);                                                   //
43         THD_calc.hd =  sqrt_LUT[zeile][spalte];                 // Klirrfaktor anhand der
               ermittelten Zeile und Spalte festlegen
44 }
```

F.2.4. cleanup.c

Listing F.26:

```
1  ///*
2  ================================================================================
3  |       Projekt:                Diplomarbeit
4  |       Erstellt von:   Niels Jegenhorst, Lennart Koch
5  |       Erstellt am:    30.06.2009
6  |       Geändert am:    04.03.2010
7  |       Hardware:               MSP430f169 auf Olimex eval board
8  |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
       v1.021
9  |       Beschreibung:   Initialisieren aller globalen Variablen, Vorlage von Niels
10 Jegenhorst, reduziert und fehlende Informationen ergänzt
11 |
12 |       Funktion:               OK
13 |       Datei:                  cleanup.c
14 ================================================================================
15 */
16
17 #include "header_main.h"
18 #include "globales.h"
19
20 //--- Funktion cleanup ---
21 void cleanup(){
22
23         g_adc.u_diff_b  = 0x0000;
24
25         for(g_calc.i=0; g_calc.i < HARMONICS; g_calc.i++) {
26                 g_calc.h_abs[g_calc.i]  = 0x00000000;
27                 g_calc.im[g_calc.i]             = 0x00000000;
28                 g_calc.re[g_calc.i]             = 0x00000000;
29         }
30         g_calc.factor           = 0x00000000;
31         g_calc.sum_harmonic = 0x00000000;
32         g_calc.sum_total        = 0x00000000;
33         g_calc.hd                       = 0x00;
34         g_calc.lut_pos          = 0x00;
35         g_calc.omega_hx         = 0x0000;
36         g_calc.omega_hx_cos = 0x0000;
37         g_calc.omega_inc        = - OMEGA_H1;
38         g_calc.i                        = 0x00;
39         g_calc.finish           = FALSE;
40
41         // ab hier eigene Arbeit
42         THD_calc.imag = 0;
43         THD_calc.real = 0;
44         THD_calc.imag_quad = 0;
45         THD_calc.real_quad = 0;
46         THD_calc.x_gl = 0;
47         THD_calc.lut_pos = 0;
48         THD_calc.prod = 0;
49         THD_calc.l_ges = 0;
50         THD_calc.l1 = 0;
51         THD_calc.x0 = 0;
52         THD_calc.x0_quad = 0;
53 }
54
55 //-------------------------|
```

F.2.5. Headerfiles

header_main.h

Listing F.27:

```
1  /*
2  ============================================================================================
3  |       Projekt:                Diplomarbeit, Signalcontroller
4  |       Erstellt von:   Lennart Koch
5  |       Erstellt am:    06.05.2010
6  |       Geändert am:    08.04.2010
7  |       Hardware:               MSP430f169 auf Olimex eval board
8  |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
       v1.021
9  |       Beschreibung:   Main - Header, Quelle Jegenhorst, LPS Projekt
10 |
11 |       Funktion:               OK
12 |       Datei:                  header_main.h
13 ============================================================================================
14 */
15
16 #ifndef HEADER_MAIN_H_
17 #define HEADER_MAIN_H_
18
19 /*#ifndef EXTERN
20 #define EXTERN extern
21 #endif*/
22
23 #include <msp430x16x.h>          // Für Compiler msp430x1232 ausgewählt um HW Multi
24 // zu deaktivieren!
25 #include <signal.h>
26 #include "table_sin_red.h"
27
28 //--- Definitionen ------------------------------------------------------------
29 #define FALSE                   0
30 #define TRUE                    !FALSE
31
32 #define N1      64          // Anzahl der Samples
33 #define N2      N1>>1               // N/2
34 #define N4      N1>>2               // N/4
35 #define SHIFT_N 6
36 #define SHIFT_QUAD 12           // Schiebefaktor für Quadratbildung
37
38
39 #define SAMPLE_P_P              64                                      // Samples pro
       Periode
40 #define SAMPLE_P_P_DIV  6                                       // Verschiebung für
       SAMPLE_P_P: log2(64)
41 #define HARMONICS               5                                       // Anzahl der
       Schwingungen
42
43 #define OMEGA_H1                16                                      // 1024 * fh1 / fs
       = 1024 / SAMPLE_P_P
44 #define LUT_POS_INC             16                                      // Inkrementierung
       in der LUT
45
46 //LCD commands
47 #define         DISP_ON                 0x0c            //LCD control constants
48 #define         DISP_OFF                0x08            //
49 #define         CLR_DISP                0x01            //
```

```
50 #define          CUR_HOME                0x02            //
51 #define          ENTRY_INC               0x06            //
52 #define          DD_RAM_ADDR             0x80            //
53 #define          DD_RAM_ADDR2            0xc0            //
54 #define          DD_RAM_ADDR3            0x28            //
55 #define          CG_RAM_ADDR             0x40            //
56
57 #define          LCD_Data        P4OUT
58 #define          LCD_LIGHT_ON    P4OUT |= BIT0
59 #define          LCD_LIGHT_OFF   P4OUT &= ~BIT0
60
61
62 // Taster
63 #define          B1              BIT5&P1IN       //B1 - P1.5
64 #define          B2              BIT6&P1IN       //B2 - P1.6
65 #define          B3              BIT7&P1IN       //B3 - P1.7
66
67 #define          _100us                  7               //7 cycles *12 + 20 = 104 / 104*1
       us = 104us
68
69 #define          E_HIGH          P4OUT |= BIT1
70 #define          E_LOW           P4OUT &= ~BIT1
71 #define          RS_HIGH         P4OUT |= BIT3
72 #define          RS_LOW          P4OUT &= ~BIT3
73
74 // LEDs
75 #define      LED1_ON         P3OUT &= ~BIT6
76 #define      LED1_OFF        P3OUT |= BIT6
77 #define      LED1_TOGGLE     P3OUT ^= BIT6
78 #define      LED2_ON         P3OUT &= ~BIT7
79 #define      LED2_OFF        P3OUT |= BIT7
80 #define      LED2_TOGGLE     P3OUT ^= BIT7
81
82
83
84 //--- Prototypen ------------------------------------------------------------
85
86 void decode_LUT(void);
87 void calc_HD_noise(void);
88 void calc_HD_noise_sampling(void);
89 void calc_HD_noise_afer(void);
90 void calc_dft_sample(void);
91 void calc_dft_abs(void);
92 void calc_hd(void);
93 void cleanup(void);
94 void Delay (unsigned int a);
95 void Delayx100us(unsigned int b);
96 void _E(void); // LCD
97 void init_TimerB (unsigned int cycles);
98 char taste( void );
99
100 void SEND_CHAR ( unsigned char); // LCD
101 void SEND_CMD ( unsigned char); // LCD
102 void InitLCD( void ); // LCD
103 void _E( void ); // LCD
104 void showString( char* ); // LCD
105 void showStringAndFloat( char* , float ); // LCD
106 void showStringAndInt( char* , unsigned int ); // LCD
107
108
109
110
111 #endif /* HEADER_MAIN_H_ */
```

```
112 //-------------------------------------------------------------------------|
```

table_sqrt.h

Listing F.28:

```
1  /*
2   ============================================================================================
3   |       Projekt:                Diplomarbeit, C Implementierung
4   |       Erstellt von:   Lennart Koch
5   |       Erstellt am:    11.02.2010
6   |       Geändert am:    11.02.2010
7   |       Hardware:               MSP430f169 auf Olimex eval board
8   |       Tools:                  MSPGCC - v.20060502; Eclipse - ganymede-SR2; USBEXpress -
        v1.021
9   |       Beschreibung:   Verkürzte LUT Ermittlung der Wurzel eines Wertes
10  |
11  |       Funktion:               OK
12  |       Datei:                  table_sqrt.h
13   ============================================================================================
14  */
15
16  #ifndef TABLE_SQRT_H_
17  #define TABLE_SQRT_H_
18
19  #ifndef EXTERN
20          #define EXTERN
21  #endif
22
23  #define MAX_ZEILE 8             // Anzahl der Zeilen in Matrix
24  #define MAX_SPALTE 7    // Anzahl der Spalten der Matrix
25  #define MAX_POSIBLE 29  // Maximal mögliche Anzahl der Spalten bei 32 Bit Wortbreite = 32
        -3
26
27  const unsigned char sqrt_LUT[9][8] ={
28                  {0,     0,      1,      1,      1,      1,      2,      2},
29                  {2,     2,      2,      3,      3,      3,      3,      3},
30                  {3,     3,      3,      4,      4,      4,      4,      4},
31                  {4,     5,      5,      5,      5,      6,      6,      6},
32                  {6,     7,      7,      7,      8,      8,      8,      9},
33                  {9,     9,      10,     10,     11,     11,     12,     12},
34              {13, 13,    14,     15,     15,     16,     17,     18},
35              {18, 19,    20,     21,     22,     23,     24,     25},
36              {25, 26,    28,     29,     31,     32,     34,     35},
37  };
38
39  #endif
```

globales.h

Listing F.29:

```
1
2  #ifndef GLOBALES_H_
3  #define GLOBALES_H_
4
5  #ifndef EXTERN
6          #define EXTERN extern
7          #endif
8
9
10                 //Ausgangssingal des Sensors
11 #ifdef SIGNAL
12 EXTERN const int16_t sig[64] = {2262,   2454,   2643,   2825,   2999,   3165,   3320,
       3464,   3595,   3713,   3817,
13                                                                  3905,   3978,   4033,
                                                                        4072,   4093,   4095,
                                                                        4079,   4045,
                                                                        3994,   3925,   3839,
14                                                                  3739,   3624,   3497,
                                                                        3359,   3212,   3057,
                                                                        2896,   2731,
                                                                        2563,   2393,   2223,
15                                                                  2054,   1887,   1722,
                                                                        1561,   1404,   1253,
                                                                        1107,   970,
                                                                        840,    721,    613,
16                                                                  518,    438,    374,
                                                                        327,    299,    291,
                                                                        304,    339,
                                                                        394,    471,    568,
17                                                                  684,    817,    967,
                                                                        1130,   1304,   1487,
                                                                        1677,   1871,
                                                                        2067
18 };
19 #endif
20
21 volatile int z;
22
23 /********************************** Aus Jegenhorst Arbeit
       **********************************/
24
25 struct {                                                                    // Berechungen:
26                 int16_t  omega_inc;                                       // Omega welches
                       inkrementiert wird
27                 uint16_t omega_hx;                                        // Omega des
                       jeweiligen Samples / Harmonischen
28                 uint16_t omega_hx_cos;                                  // Omega wie Omega_hx, nur
                       für Cosinus
29                 uint16_t omega_hx_half;                                 // Omega aus Omega_hx, zum
                       Spiegeln der LUTs
30                 uint16_t omega_hx_half_cos;                             // Omega wie Omega_hx_half,
                        nur für Cosinus
31                 int32_t  re[HARMONICS];                                 // Realteil
32                 int32_t  im[HARMONICS];                                 // Imaginärteil
33                 int32_t  factor;                                          // universeller
                       Faktor
34                 int32_t  h_abs[HARMONICS];                              // Betrag der Harmonischen
35                 uint32_t sum_harmonic;                                  // Summe der
                       Oberschwingungen
```

```
36                  uint32_t sum_total;                                     // Summe der
                        Gesamtschwingung
37                  uint8_t  hd;                                            // Klirrfaktor HD
38                  uint16_t lut_pos;                                       // Position in der
                        LUT (Cosinus / Sinus)
39                  uint8_t  i;                                                 //
                        Zählvariable
40                  uint8_t  x;                                                 //
                        Zählvariable
41                  uint8_t  finish;                                        // Flag
42          } volatile g_calc;
43
44
45          struct {                                                            // DAC:
46                      int16_t  u_diff_b;                                      //
                            Differenz Signal ohne Offset
47                  } volatile g_adc;
48
49  #endif
50
51  /*****************************************************************************/
52
53
54  struct{
55          int32_t x_gl;                                                   // Gleichanteil
56
57          int16_t x0;                                                         // Sample
                ohne Gleichanteil
58          int32_t x0_quad;                                                // Quadrat des
                Signals
59
60          int32_t imag;                                                   // Imaginärteil 1
                Harmonische
61          int32_t prod;                                                   // Produkt Signal x
                 Wurzeltabelle
62          int32_t real;                                                   // Realteil 1
                Harmonische
63          int32_t imag_quad;                                          // Quadrat Imaginärteil 1
                Harmonische
64          int32_t real_quad;                                              // Quadrat Realteil
                 1 Harmonische
65
66          int32_t l_ges;                                                  // Leistung des
                Gesamtsignals
67          int32_t l1;                                                         // Leistung
                 der ersten harmonischen Schwingung
68          int32_t P_ob;                                                   // Leistung der
                Oberschwingungen
69          int32_t factor;                                                 // Ergebis der
                Division vor dem Wurzelziehen
70
71          uint16_t lut_pos;                                               // Position in der
                LUT (Cosinus / Sinus)
72          uint8_t  hd;                                                    // Klirrfaktor in
                HD in %
73          } volatile THD_calc;
```

Weitere Dateien sind aus anderen Projekten übernommen wurden, sind auf [8] enthalten

Tabellenverzeichnis

Abbildungsverzeichnis

Index

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 21. April 2010 | _____

Ort, Datum | Unterschrift