Bachelorthesis

Heiko Poppinga

Controller-Implementation und messtechnische Erprobung der Signalverarbeitung für die Diagnosefunktion von ABS-Sensoren

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Heiko Poppinga

Controller-Implementation und messtechnische Erprobung der Signalverarbeitung für die Diagnosefunktion von ABS-Sensoren

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.rer.nat Jochen Schneider

Abgegeben am 20. April 2011

Heiko Poppinga

Thema der Bachelorthesis

Controller-Implementation und messtechnische Erprobung der Signalverarbeitung für die Diagnosefunktion von ABS-Sensoren

Stichworte

ABS-Sensor, AMR-Effekt, Klirrfaktor, Sensor-Diagnose, Unterabtastung, Sequentielle Abtastung

Kurzzusammenfassung

Dieser Arbeit beschäftigt sich mit der Implementierung und Verifikation von verschiedenen Algorithmen zur Zustandserkennung von ABS-Sensoren. Diese Algorithmen wurden zunächst in Matlab evaluiert und anschliessend auf einer Experimentalplattform auf Basis eines TI MSP430 implementiert und ausgiebig erprobt. Die Besonderheit dieser Algorithmen ist, dass spezielle Zustände wie zum Beispiel die Frequenzverdopplung des Sensorsignals detektiert werden können.

Heiko Poppinga

Title of the paper

Controller implementation and testing of the signal processing algorithm for the diagnosis of ABS sensors

Keywords

ABS-sensor, AMR-effect, harmonic distortion, sensor diagnosis, subsampling, sequential sampling

Abstract

In this report, different algorithms for the diagnosis of ABS sensors were implemented and verified. These algorithms were first evaluated in Matlab and then implemented on an experimental platform based on an TI MSP430 and tested extensively. These algorithms are able to detect special sensor states such as doubling of the sensor-signal frequency.

Danksagung

Zunächst möchte ich mich bei Herrn Prof. Dr.-Ing Karl-Ragmar Riemschneider bedanken, für das engagierte Betreuen dieser Arbeit. Des Weiteren Danke ich Herrn Prof. Dr.rer.nat Jochen Schneider für die Übernahme der Zweitprüfung.

Ich danke auch allen Mitgliedern des ESZ-ABS-Teams für ihre Unterstüzung. Besonders möchte ich Herrn Dipl.-ing Martin Krey danken, der mir stets mit Rat und Tat zur Seite stand. Außerdem danke ich Herrn Kalin Ivanov für die Durchführungen und das außerplanmäßigen Vorziehen der für mich notwendigen Messungen am Radmessplatz.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einführung

Diese Arbeit im Rahmen des Forschungsprojekts „Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren“ (ESZ-ABS), betreut durch Prof. Dr. Karl-Ragmar Riemschneider und Dipl.-Ing. Martin Krey, entstanden. Das Ziel dieses Forschungsprojekts ist ein Chipentwurf eines ABS-Sensors mit integrierter Diagnosefunktion. Diese Diagnosefunktion soll durch Auswertung der Signalform eine Aussage über die Funktionstüchtigkeit des Sensors zulassen, um im Störfall das ABS-System zu deaktivieren und den Fahrer auf die Fehlfunktion hinzuweisen. Dieser Störfall, der detektiert werden soll, ist nicht der gänzliche Defekt des ABS-Sensors, bei dem keinerlei Signale mehr übertragen werden. Es handelt sich um das ungleich schwerer zu diagnostizierenden Fall, wenn der Sensor zwar Signale sendet, aber diese nicht zuverlässig sind.

Diese Arbeit beschäftigt sich mit der Weiterentwicklung und Erweiterung von Möglichkeiten zur Sensordiagnose, welche aufgezeigt und diskutiert werden. Dafür wurden Messungen mit einem ABS-Sensor auf AMR-Basis an verschiedenen Radnaben durchgeführt. Betrachtet wurde insbesonders ein Sonderfall, der dazu führt, dass der Controller des ABS-Sensors durch Auswertung des Sensorsignals eine doppelt so große Raddrehzahl wie die tatsächliche bestimmt. Dieser Fall ist besonders kritisch, da eine solche Fehlfunktion zu schweren Unfällen führen kann.

1.1. Anisotroper magnetoresitiver Sensor

Moderne ABS-Sensoren nutzen den anisotrop magnetoresistiven Effekt, welcher vor über 150 Jahren erstmals beschrieben wurde. Dieser Effekt bewirkt, dass ein Metall in Abhängigkeit von Stärke und Richtung eines äußeren Magnetfeldes seinen elektrischen Widerstand ändert. Um diesen Effekt auszunutzen ordnet man vier dieser Metalle, also Widerstände, in einer Wheatstonschen Brücke an (siehe Abbildung 1.1). Diese Messbrücke befindet sich auf einem Permanentmagneten, der ein sogenanntes Stützfeld erzeugt. Eine Skizze eines Sensors mit Messbrücke, Permanentmagneten und Controllereinheit ist in Abbildung 1.2 zu sehen. Ein Zahnrad, welches aus einem ferromagnetischen Material besteht, im Folgenden als passives Encoderrad bezeichnet, beeinflusst die Ausrichtung der Magnetfeldlinien, sodass

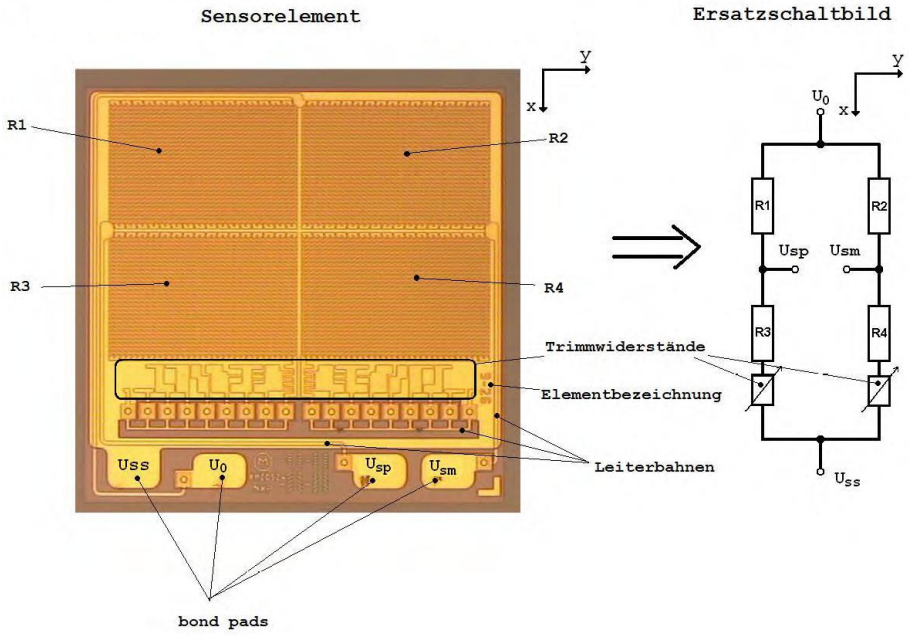


Abbildung 1.1.: AMR-Messbrücke, entnommen aus [11]

sich auch das Ausgangssignal der Messbrücke verändert. Auf eine andere Weise funktionieren aktive Encoderräder. Diese besitzen abwechselnd magnetische Nord- und Südpole, wodurch ein ähnlicher Effekt erzielt wird, wie auch beim passiven Encoderrad. Beispiele für ein aktives und ein passives Encoderrad sind die Golf V- bzw. Golf IV-Radnaben, die unter 1.3 abgebildet sind.

Der Sensor besitzt eine nichtlineare Kennlinie, allerdings hat sie, wie in 1.5 zu sehen, einen deutlichen linearen Bereich, in dem der Sensor folglich genutzt werden muss.

Wird der Sensor ausserhalb des linearen Bereichs angesteuert, so kommt es zu nichtlinearen Verzerrungen des Ausgangssignals. Die Illustration 1.4 zeigt ein passives Encoderrad, welches sich vor einem AMR-Messbrücke - dem ABS-Sensorkopf - dreht. Bewegt sich ein Zahn und eine Lücke an dem Sensorkopf vorbei, so ergibt sich als Ausgangssignal eine Periode eines Sinus, vorausgesetzt der Sensor wird im linearen Bereich betrieben. Die Controllereinheit, die unter der Sensorkopf angebracht ist, verarbeitet das analoge Ausgangssignal der Messbrücke und gibt ein digitales Ausgangssignal aus. Dieses digitale Ausgangssignal ist ein Stromprotokoll, welches in [1] spezifiziert ist (Siehe: 1.4). Das Protokoll beginnt mit dem 'Speed pulse'. Dieser wird ausgegeben, sobald der in der Steuereinheit befindliche Smart Comparator einen Nulldurchgang im Ausgangssignal der Messbrücke erkannt hat. Jeder Nulldurchgang, verursacht im Fall von passiven Encoderrädern durch Zahn bzw. Lücke, im Fall von aktiven Encoderrädern durch Nord- bzw. Südpol wird „increment“ genannt. Ein

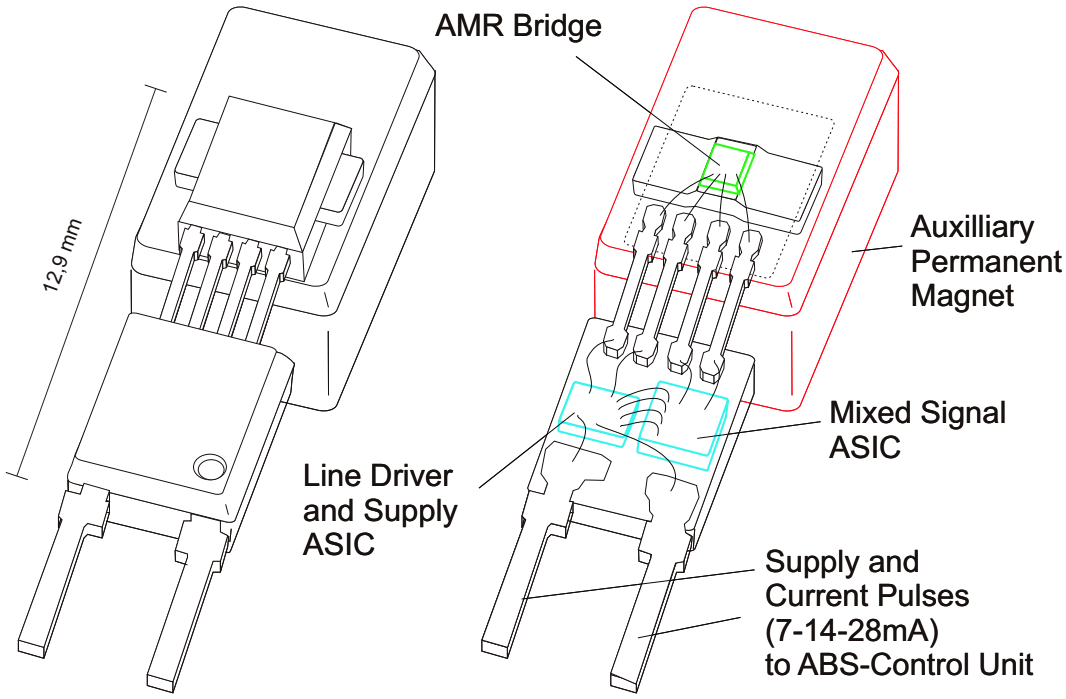


Abbildung 1.2.: AMR-Sensor mit Messbrücke, permanten Magneten und Controllereinheit

passives Encoderrad mit 50 Zähnen sowie ein aktives Encoderrad mit 50 Polpaaren besitzt folglich 100 „increments“. Nähere Erläuterungen zum AMR-Effekt sind zum Beispiel in [2] im Kapitel 9 von U. Dibbern nachlesbar.

(a) Golf V-Radnabe - akives Encoderrad

(b) Golf IV-Radnabe - passives Encoderrad

Abbildung 1.3.: Radnaben

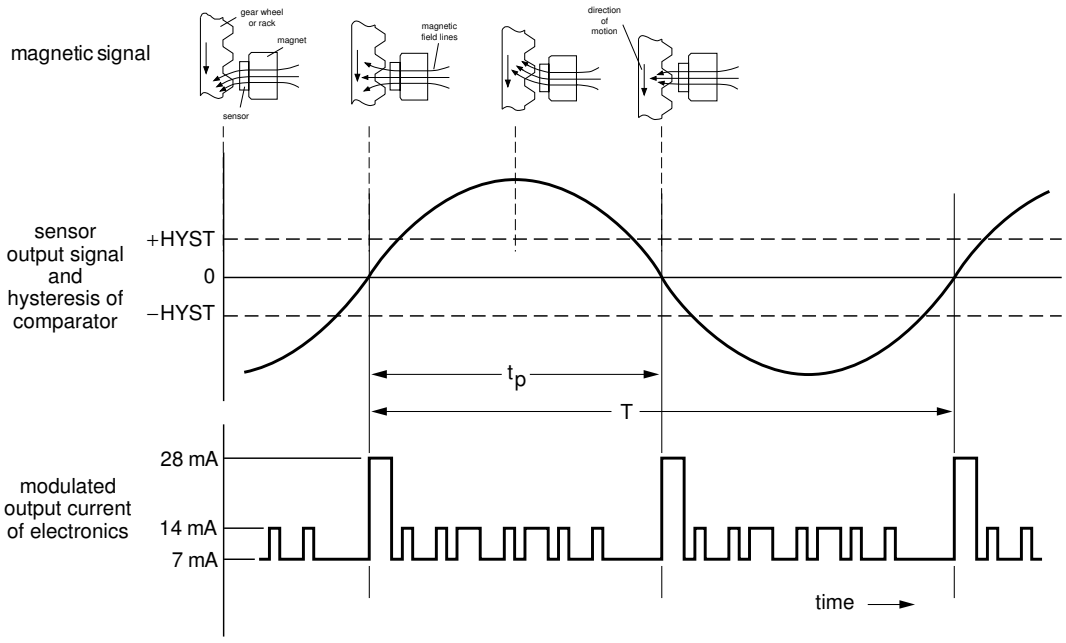


Abbildung 1.4.: Sensor vor passivem Encoderrad

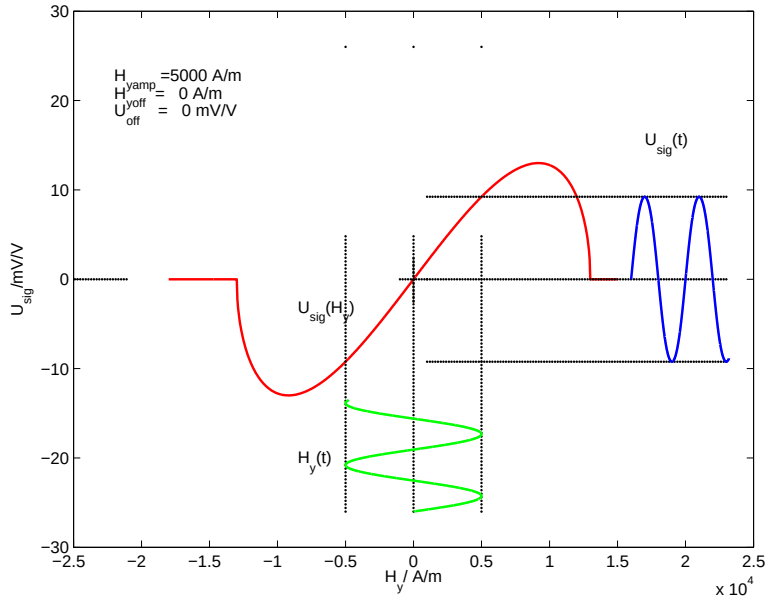


Abbildung 1.5.: Sensorkennlinie mit magnetischem Eingangssignal(grün) und elektrischem Ausgangssignal(blau)

1.2. Experimentalplattform

Die in dieser Arbeit entwickelten Funktionen zur Sensordiagnose wurden auf einer Experimentalplattform implementiert, die im Rahmen der Diplomarbeit *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren* [6] entwickelt wurde. Des Weiteren wurde eine Regelplatine zur Regelung von Offset und Verstärkung in der Arbeit [13] entwickelt. Die Plattform bildet die Grundfunktionen des Controllers eines KMI22 ABS-Sensor nach. Diese umfassen:

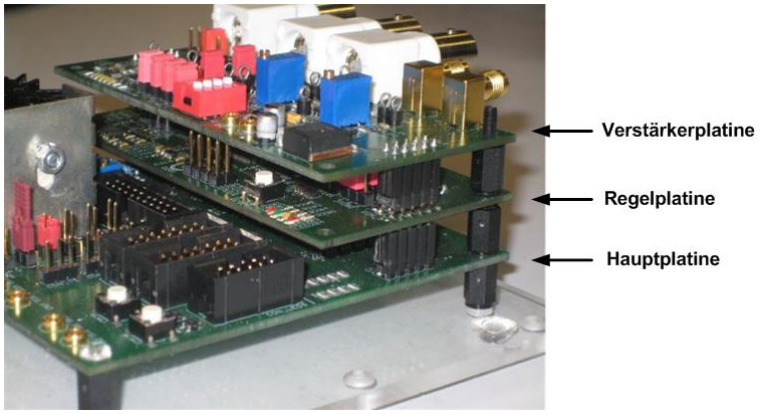


Abbildung 1.6.: Experimentalplattform bildet Funktionen des KMI22 ABS-Sensors nach

- Digitales Strom-Ausgangssignal
- Digitale Offsetkompensation
- Drehrichtungserkennung
- Digitales Ausgabeprotokoll
- Stillstandserkennung

Zusätzlich wurde als Sensordiagnosefunktion eine Analyse der harmonischen Schwingungen des Sensorsignals implementiert.

2. Analyse, Vorarbeiten und Problemstellung

2.1. Sensordiagnosefunktion - Klirrfaktorberechnung

In einer Voruntersuchung [10] wurde festgestellt, dass die Verzerrung des Differenzsignals der AMR-Messbrücke Aufschlüsse über die Funktionstüchtigkeit des Sensors geben können. Als Indikator für die Funktionstüchtigkeit des Sensors wird auf der Experimentalplattform ein Messwert berechnet und angezeigt, der dem Klirrfaktor *k* ähnelt. „Der Klirrfaktor k ist das Verhältnis des Oberwelleneffektivwert zum Gesamteffektivwert “ [9, S.366]. Im Folgenden bezeichnet $\bar{U}$ den Gesamteffektivwert und $\bar{U}_h$ den Effektivwert der Harmonischen mit dem Index h. In [3] wird dies wie folgt beschrieben:

$$k = \frac{\sqrt{\bar{U}^2 - \bar{U}_1}}{\bar{U}} \tag{2.1}$$

Anders formuliert in [9, S. 366]:

$$k_{\%} = 100\% \cdot \sqrt{\frac{\sum_{h=2}^{\infty} \bar{U}_h^2}{\sum_{h=1}^{\infty} \bar{U}_h^2}} \tag{2.2}$$

Das Verfahren Harmonic-Distortion-5 (HD5) zur Ermittlung des Klirrfaktors ist in der Vorarbeit [6] implementiert worden (siehe 2.1.2). Eine andere Methode, der Harmonic-Distortion-Infinite-Algortihmus (HDI) wurde in der Arbeit [8] entwickelt, und daraufhin im Rahmen dieser Arbeit auf der Experimentalplattform implementiert (siehe 2.1.3).

2.1.1. Abtastung

Um die Verzerrung des Sensorsignals analysieren zu können, muss es zunächst abgetastet werden. Da laut [6] eine periodische Abtastung aufgrund begrenzter Rechenleistung des MSP430-Mikrocontroller nicht möglich ist, wurde eine abgewandelte Form einer sequentielle

Abtastung realisiert. Mit einer sequentiellen Abtastung kann eine hohe zeitliche Auflösung Δt eines Signals erreicht werden, obwohl mit einer niedrigen Frequenz abgetastet wird.

Eine sequentielle Abtastung meint, dass aus einem frequenzunveränderlichen Signal pro Periode T ein Abtastwert entnommen wird. Das Abtastintervall ist folglich:

$$T_a = T + \Delta t \tag{2.3}$$

Nach S Abtastwerten ist eine Periode des Signals aufgenommen. Für S gilt:

$$S = \frac{T}{\Delta t} \tag{2.4}$$

Voraussetzung für dieses Verfahren ist folglich neben der konstanten Frequenz auch eine hinreichend gleichbleibende Signalform und Amplitude. Mathematisch ausgedrückt bedeutet das:

$$x(t) = x(t + kT) \qquad \text{für} \quad k = -\infty \leq k \leq +\infty \tag{2.5}$$

In der Praxis besitzt das Sensorsignal der AMR-Messbrücke allerdings selten eine konstante Frequenz. Denn durch das Beschleunigen und Abbremsen während einer Autofahrt verändert sich die Raddrehzahl permanent. Das Problem wurde gelöst, indem für jede Periode des Sensorsignals aus der ein Abtastwert entnommen werden soll, ein individueller Abtastzeitpunkt t_{sn} berechnet wird. Periode meint in diesem Fall die Dauer zwischen zwei aufsteigenden Nulldurchgängen des Sensorsignals. Dieser Abtastzeitpunkt wird berechnet indem die Dauer der vorherigen Periode T_{n-1} auch als Dauer für die aktuelle Signalperiode T_n angenommen wird. Daraufhin werden diese beiden Zeiten miteinander verglichen. Bei einer signifikanten Abweichung wird der Abtastwert verworfen. Der Toleranzbereich, in dem ein Wert akzeptiert wird, wurde auf $\pm 3,125\%$ festgelegt. Auf diese Weise können aus den S Abtastwerten zwei Halbwellen des Sensorsignals zusammengesetzt werden. Da sich das Encoderrad innerhalb einer Signalperiode um einen ganz bestimmten Winkel dreht, ist die Abtastung nicht mehr im Bezug auf die Zeit, sondern im Bezug auf den Drehwinkel des Encoderrades näherungsweise äquidistant. Diese Drehwinkelsynchrone Abtastung wird auch als *Order Tracking* bezeichnet.[4]

Die Voraussetzung für diese Art der Abtastung ist, dass die Signalform und die Amplitude bei jeder Encoderinkrementierung gleich bleibt. Dies drückt die Gleichung 2.5 in einer abgewandelten Form aus:

$$x(\alpha) = x(\alpha + k\Theta) \qquad \text{für} \quad k = -\infty \leq k \leq +\infty \tag{2.6}$$

Dabei entspricht Θ der Winkeldifferenz einer Encoderinkrementierung, also dem Vorbeibe-

wegen eines Zahn-Lücke-Paares des Encoderrades am Sensor. Es gilt in Bogenmaß:

$$\Theta = \frac{2\pi}{M} \quad \text{mit} \quad \text{M = Anzahl der Zähne} \tag{2.7}$$

Die Abbildung 2.1 zeigt schematisch eine solche sequentielle Abtastung eines drehwinkelsynchronen Signals. Allerdings ist zu beachten, dass nur Signalperioden, aus denen ein Abtastwert entnommen wird dargestellt sind. Aufgrund des Designs der realisierten Abtastung vergehen mehrere Signalperioden zwischen einzelnen Abtastpunkten. Siehe hierfür Abschnitt 4.1.

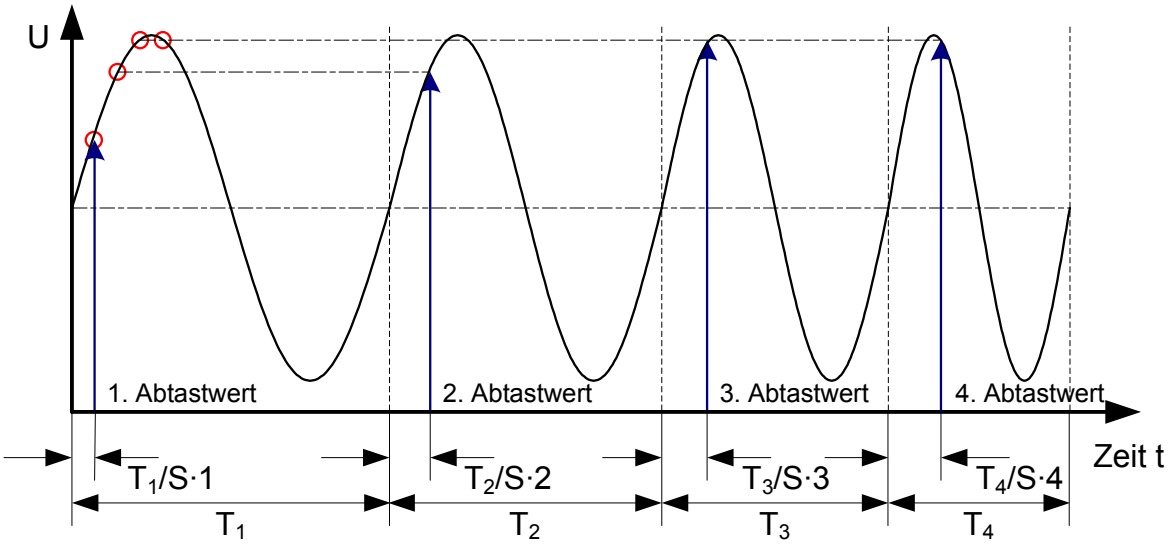


Abbildung 2.1.: Sequentielle Abtastung eines nicht periodischen Signals, entnommen aus [6]

Diese Form der Abtastung führt dazu, dass sich jede spektrale Analyse dieser Abtastwerte nicht auf Frequenzen bezogen ist. Stattdessen erhält man durch eine Diskrete Fourier Transformation (DFT) ein Ordnungsspektrum des Encoderrades. Diese Ordnung bezieht sich auf die Periodizität des Sensorsignals mit dem Drehwinkel des Encoderrades. Eine Schwingung der 1. Ordnung ist periodisch mit einem Drehwinkel von 2π, die der 2. Ordnung mit $\frac{2\pi}{2}$ und die der n-ten Ordnung mit $\frac{2\pi}{n}$. [7]

Da nur ein Ausschnitt Θ einer Encoderradumdrehung aus den Abtastwerten zusammengesetzt wird und Gleichung 2.7 gilt, erhält man durch eine DFT nicht das Vollständige Ordnungsspektrum, sondern die spektralen Anteile der Schwingungen der M-ten Ordnung und deren Vielfache.

In den folgenden Kapiteln werden die Abtastwerte zur Vereinfachung so behandelt, als wären sie Produkt einer zeitlich äquidistanten Abtastung. Es wird also davon ausgegangen, dass

das Encoderrad während eines Abtastintervalls eine konstante Drehzahl besitzt. Dies stellt eigentlich einen Ausnahmefall dar. Auf diese Weise können die Schwingungen der M-ten Ordnung als 1. Harmonische und die der $n \cdot M$-ten Ordnung als n-te Harmonische bezeichnet werden. Auch die Begrifflichkeit des *Klirrfaktor* k, welcher sich auf eine feste Grundfrequenz bezieht, wird beibehalten, obwohl es meist gar keine feste Grundfrequenz gibt.

2.1.2. Harmonic Distortion 5 - HD5

Der Name $HD5$ rührt daher, dass das Verfahren zur Bestimmtung des Klirrfaktors *k* eine DFT nutzt, die fünf Harmonische berechnet. Mit Hilfe der DFT lässt sich eine Abtastfolge x[k] der endlichen Länge N transformieren.

$$X[n] = \sum_{k=0}^{N-1} x[k] \cdot e^{-j2\pi nk/N} \qquad 0 \leq n \leq N-1 \tag{2.8}$$

und Umformung in die trigonometrische Form erhält man:

$$X[n] = \sum_{k=0}^{N-1} x[k](cos(\omega_n k) - jsin(\omega_n k)) \tag{2.9}$$

mit

$$\omega_n = 2\pi \frac{n}{N}$$

Da für die Klirrfaktor-Berechnung nur die harmonischen Anteile von Interesse sind, lässt sich die Formel 2.9 weiter umstellen und vereinfachen. Im Folgenden ist f_{h1} die Frequenz der ersten harmonischen Schwingung und n_h ist der ensprechende Index für diese Frequenz.

$$n_h = f_{h1} \cdot \Delta t \cdot N \tag{2.10}$$

mit dem Abtastinterval $\Delta t = \frac{1}{f_s}$. f_s ist die Abtastfrequenz. Ist *S* die Anzahl der Abtastungen pro Periode der ersten Harmonischen, also der Grundschwingung, so vereinfacht sich die Formel 2.10 weiter mit $f_s = S \cdot f_{h1}$:

$$n_h = \frac{N}{S} \tag{2.11}$$

Dadurch vereinfacht sich auch ω_n, welches bezogen auf die erste Harmonische als ω_{nh} deklariert wird.

$$\begin{aligned} \omega_{nh} &= 2\pi \frac{N}{S \cdot N} \\ \rightarrow \omega_{nh} &= \frac{2\pi}{S} \end{aligned} \tag{2.12}$$

Nun lässt sich die Fourier-Transformation der Harmonischen mit dem Index *h* aufstellen. Man erhält:

$$X[h] = \sum_{k=0}^{N-1} x[k](cos(\omega_{nh} \cdot k \cdot h) - jsin(\omega_{nh} \cdot k \cdot h)) \tag{2.13}$$

Diese ist allerdings nur für diejenigen Harmonischen gültig, die das Abtasttheorem nach Nyquist-Shannon erfüllen. Das ist wie folgt definiert (siehe [9, S.216]) :

$$f_s \, 2 \cdot f_{max} \tag{2.14}$$

Auf diesen Fall angewandt bedeutet das:

$$\begin{aligned} & f_s \, 2 \cdot h \cdot f_{h1} \\ \rightarrow h < & \frac{f_s}{2 \cdot f_{h1}} \end{aligned} \tag{2.15}$$

Durch die Methode der diskreten Fourier-Transformation erhält man die Beträge der Harmonischen. Der Klirrfaktor lässt sich gleichwertig zur Formel 2.2 durch die Beträge der Amplituden der Harmonischen ausdrücken.

$$k_{\%} = 100\% \cdot \sqrt{\frac{\sum_{h=2}^{\infty} |U_h|^2}{\sum_{h=1}^{\infty} |U_h|^2}} \tag{2.16}$$

In der Vorarbeit von [6] wurde experimentell festgestellt, dass es ausreichend genau ist, die ersten fünf Harmonischen zu bestimmen. Folglich:

$$k_{\%} = 100\% \cdot \sqrt{\frac{|U_2|^2 + |U_3|^2 + |U_4|^2 + |U_5|^2}{|U_1|^2 + |U_2|^2 + |U_3|^2 + |U_4|^2 + |U_5|^2}} \tag{2.17}$$

2.1.3. Harmonic Distortion Infinite Algorithmus - HDI

Harmonic-Distortion-Infinite-Verfahren (HDI) ist ein Verfahren, das von Koch in seiner Diplomarbeit [8] entwickelt wurde. Es stellt eine Alternative zu dem HD5 Verfahren dar. Es schätzt den Klirrfaktor *k* über den Ansatz der Signalleistung. Dieser Name soll andeuten, dass im Gegensatz zum HD5 in der Berechnung des Klirrfaktors alle Harmonischen eingeschlossen sind, da sie alle zur Gesamtsignalleistung beitragen.

Es gilt für die Signalleistung:

$$P = \frac{\bar{U}^2}{R} \tag{2.18}$$

Also kann die Gleichung 2.2 auch mittels der Signalleistung ausgedrückt werden:

$$k_{\%} = 100\% \sqrt{\frac{P_{ges} - P_1}{P_{ges}}} = 100\% \sqrt{\frac{P_{ob}}{P_{ges}}} \tag{2.19}$$

P_{ges} ist die Gesamtwechselleistung des Signals, P_1 bezeichnet die Signalleistung der 1. Harmonischen und P_{ob} ist die Leistung der Oberwellen. Durch diesen Ansatz ist es nicht nötig, die einzelnen Anteile der Harmonischen zu berechnen. Lediglich die Gesamtleistung, sowie die Leistung der 1. Harmonischen werden zur Berechnung gebraucht. Dadurch lässt sich der Rechenaufwand verringern. Allerdings geht in die Gesamtleistung auch die Rauschleistung ein. Deswegen, und weil die HD5-Methode nur die Anteile der ersten fünf Harmonischen in die Klirrfaktorberechnung einbezieht, ist der berechnete Klirrfaktor der HDI-Methode zumindest in der Theorie größer bei Anwendung auf das gleiche Signal.

Die Gesamtwechselleistung P_{ges} wird durch folgende Formel beschrieben:

$$P_{ges} = \frac{1}{T} \cdot \int_0^T s_{\sim}(t)^2 \cdot dt \tag{2.20}$$

Da es sich bei dem Signal ($s(t)$) um ein diskret Signal handelt, kann das Integral durch eine Summe ersetzt werden.

$$P_{ges} = \frac{T_a}{T} \cdot \sum_{n=0}^{\frac{T-T_a}{T_a}} s_{\sim}(n \cdot T_a)^2 \tag{2.21}$$

Mit $N = \frac{T}{T_a}$ lässt sich die Formel weiter vereinfachen.

$$P_{ges} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} s_{\sim}(n \cdot T_a)^2 \tag{2.22}$$

Das Wechselsignal $s_{\sim}$ erhält man durch Abzug des Gleichanteils s_{gl} von den einzelnen Sample-Werten.

$$s_{\sim}(n \cdot T_a) = s(n \cdot T_a) - s_{gl} \tag{2.23}$$

Mit $T_a = 1$ und 2.23 ergibt sich aus der Gleichung 2.22:

$$P_{ges} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} (s(n) - s_{gl})^2 \tag{2.24}$$

Nach Auflösen der Binomischen Formel erhält man:

$$P_{ges} = \frac{1}{N} \cdot \left[\sum_{n=0}^{N-1} s^2(n) - 2 \sum_{n=0}^{N-1} s(n) \cdot s_{gl} + \sum_{n=0}^{N-1} s_{gl}^2 \right] \tag{2.25}$$

Da s_{gl} eine Konstante ist gilt:

$$\sum_{n=0}^{N-1} s_{gl}^2 = N \cdot s_{gl}^2 \tag{2.26}$$

Der Gleichanteil ist das arithmetische Mittel der Abtastwerte $s(n)$.

$$s_{gl} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} s(n) \tag{2.27}$$

Dadurch vereinfacht sich die Gleichung 2.25 zu:

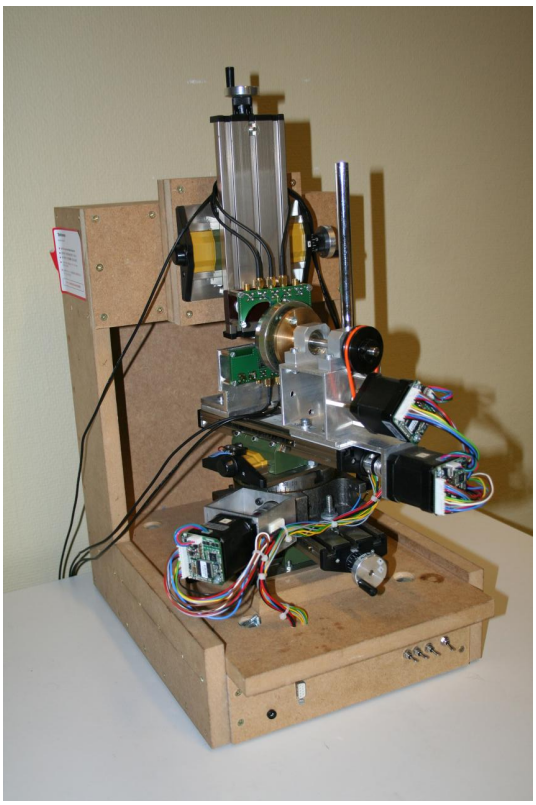
$$P_{ges} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} s^2(n) - 2 \cdot s_{gl}^2 + s_{gl}^2 = \frac{1}{N} \cdot \sum_{n=0}^{N-1} s^2(n) - s_{gl}^2 \tag{2.28}$$

Zur letzendlichen Berechnung des Klirrfaktor ist es nun nur noch nötig mit Hilfe der DFT, wie in Abschnitt 2.1.2 beschrieben, den Anteil der ersten Harmonischen zu berechnen.

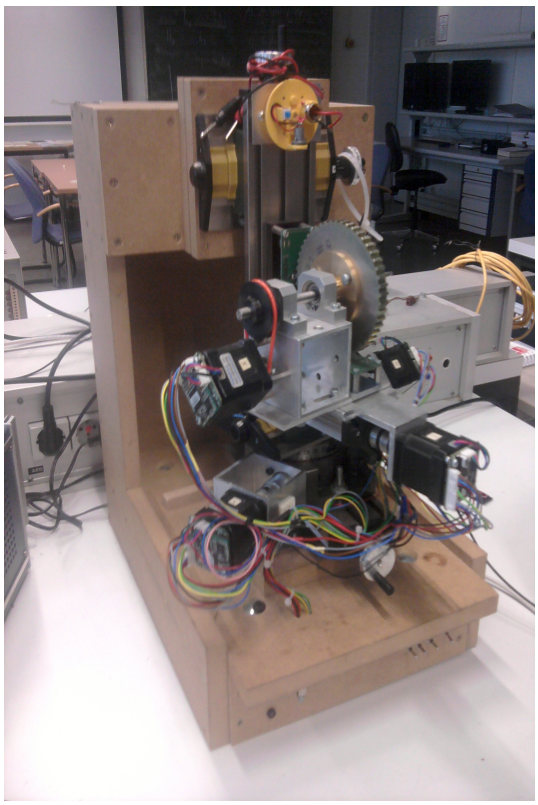
2.2. Radmessplatz

2.2.1. Radmessplatz 2

In Vorarbeiten wurde ein Messplatz entworfen, um die Sensorsignale bei unterschiedlicher Positionierung des Encorderrades vor dem Sensor zu untersuchen. Der Messplatz besitzt einen Drehtisch, darüber ist ein in X-Y-Richtung verstellbarer Tisch angebracht, auf dem sich das Encoderrad befindet. Vier Schrittmotoren ermöglichen den Antrieb des Encoderrads, sowie das Drehen des Drehtisches und das Verstellen der beiden Linearachsen. Diese Motoren lassen sich von einem PC aus steuern. Somit ist es möglich, mittels Matlab automatisierte Messungen vorzunehmen. Zunächst wurde der Messplatz nur für ein aktives Encoderrad konzipiert. Um auch mit einem passiven Encoderrad Messen zu können, musste der Messplatz etwas modifiziert werden. Um das Encoderrad anbringen zu können, wurde ein passender Flansch hergestellt. Ausserdem war es nötig, die Antriebseinheit um 90° zu versetzen. Das ist damit zu Begründen, dass die Magnetisierung am aktiven Encoderrad nicht äquivalent zu den Zähnen des passiven Encoderrads auf dem Umfang des Rades erfolgt, sondern ringförmig an der Außenseite in Richtung der Welle augerichtet ist. Also muss der ABS-Sensor vor einem passiven Encoderrad axial ausgerichtet sein, wohingegen er vor einem aktiven Encoderrad radial ausgerichtet zu sein hat. Siehe Abbildung 2.2.



(a) RMP 2 mit aktivem Encoderrad



(b) RMP 2 mit passivem Encoderrad

Abbildung 2.2.: Radmessplatz 2 (RMP2)

2.2.2. Radmessplatz 3

Schoermer hat im Rahmen seiner Diplomarbeit [12] einen Radmessplatz konstruiert, der präzise Messungen zulässt. Der ABS-Sensor kann mittels dreier Linearachsen auf $10\mu m$ genau im Raum vor dem Encoderrad positioniert werden. Außerdem ist eine Verkippung um diese Linearachsen möglich, wobei eine Genauigkeit von 0,1° erreicht wird. Dieser Radmessplatz erlaubt die Vermessung von original Encoderrädern der Autohersteller. Mittels zweier modular austauschbarer Antriebsmotoren, ist ein Drehzahlbereich von „$< 1min^{-1}$ bis $4000min^{-1}$“ möglich. Dabei ist ein Schrittmotor für den unteren Drehzahlbereich und ein Brushless-DC-Motor für den oberen Drehzahlbereich vorgesehen. Die meisten in dieser Arbeit behandelten Messungen erfolgten an diesem Radmessplatz.

Schoermer hat in seiner Arbeit auch ein Messkoordinatensystem eingeführt, mit dessen Hilfe die Positionierung des Sensors vor dem Encodderrad angegeben werden kann. Auf dieses Koordinatensystem beziehen sich auch die Positionsangabe, die in dieser Arbeit gemacht werden. Herr Schoermer macht bei seiner Definition des Koordinatensystems eine Unter-

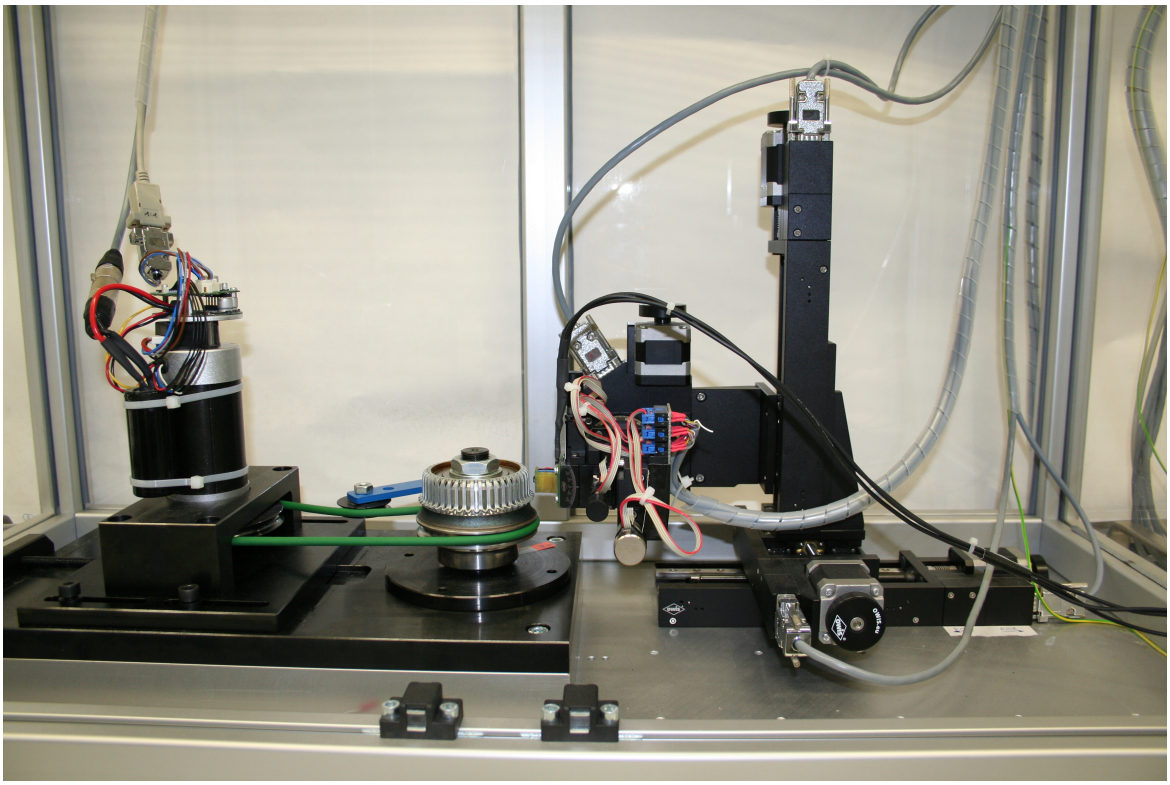


Abbildung 2.3.: RMP3 mit Golf IV Encoderrad

scheidung zwischen aktiven und passiven bzw. radialen und axialen Encoderrädern. Dargestellt ist dieses Mess-Koordinatensystem in Abbildung 2.4

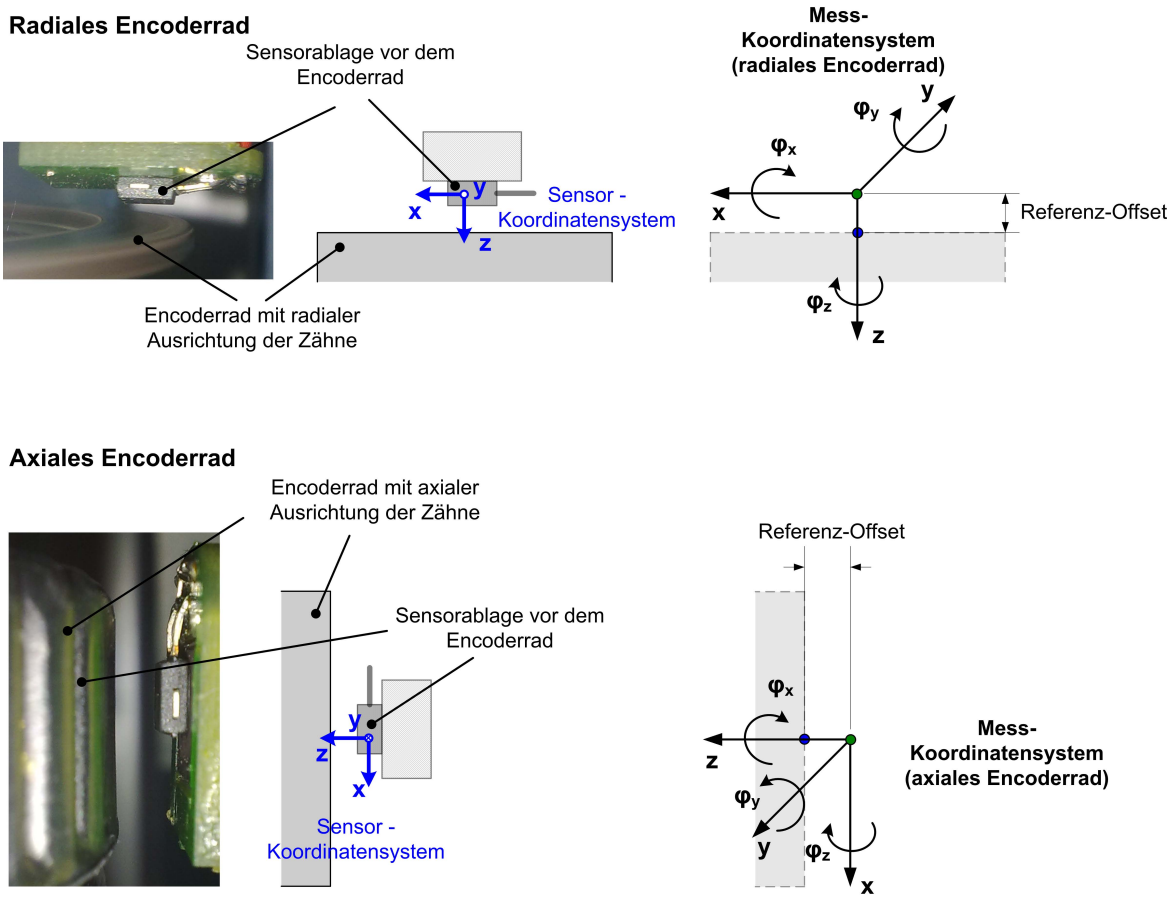


Abbildung 2.4.: Sensor- und Mess-koordinatensystem, entnommen aus [12]

2.3. Problem der Frequenzverdopplung

Durch den nichtlinearen Verlauf der Sensorkennlinie, kann es zu einem seltenen Sonderfall kommen, und zwar zu Encoderabbildungsfehlern. In diesem Fall ist das Sensorsignal durch eine ungünstige Ausrichtung des Sensors vor dem Encoderrad derart verzerrt, sodass die Inkrementierung des Encoderrads zwei weitere Nulldurchgänge verursacht, da sich zwischen den sinusförmigen Halbwellen zwei weitere sinusförmige Halbwellen ausbilden.

Damit ist das Order Tracking der Experimentalplattform, welches in Abschnitt 2.1.1 beschrieben ist, gestört. Jeder positive Nulldurchgang wird als Ende der letzten Encoderinkrementie-

rung und Anfang der folgenden Encoderinkrementierung interpretiert. Das abgetastete und zusammengesetzte Sensorsignal entspricht in diesem Fall folglich nichtmehr dem Signal einer Encoderinkrementierung. Die Abtastung wird die Abtastwerte teils aus der ersten Hälfte, teils aus der zweiten Hälfte einer Encoderinkrementierung aufnehmen, da sich der zusätzliche positive Nulldurchgang ungefähr bei der halben Drehwinkeldifferenz einer Encoderinkrementierung befindet. Die Anwendung von HD5 (Abschnitt 2.1.2) und HDI (Abschnitt 2.1.3) liefert damit kein brauchbares Ergebnis und werden den Fall der Frequenzverdopplung nicht zuverlässig als einen schlechten Zustand identifizieren können.

Dies bestätigen auch Messungen von Jegenhorst, der diesen Fall in seiner Diplomarbeit [6] nachgewiesen hat. Die Abbildung 2.5 stellt rechts eine Messung des Klirrfaktors über die Distanz bei gleichbleibender Raddrehzahl dar. Der Sensor ist dabei in eine Position verkippt, die bei sehr naher Positionierung vor dem Encoderrad zu einer verdoppelten Frequenz des Sensorsignals führt. Der rote Graph stellt den Klirrfaktor dar, der mit Hilfe von Matlab errechnet wurde. Grundlage dieser Berechnung war das mit einem Oszilloskop aufgenommene Sensorsignal. Außerdem wurde als Grundfrequenz für die Klirrfaktorberechnung die Zahnfrequenz angenommen. Anhand des Graphen lässt sich feststellen, dass der Klirrfaktor unmittelbar vor dem Encoderrad sehr hoch ist. Im Gegensatz dazu ist der auf der Experimentalplattform ermittelte Klirrfaktor (blauer Graph) in diesem Bereich äußerst schwankend und wesentlich kleiner. Zum vergleich, ist auf der linken Seite dieser Abbildung eine Messung zu sehen, bei frontaler Ausrichtung des Sensors vor dem Encoderrad, indem im Nahbereich der durch die Experimentalplattform ermittelte Klirrfaktor dem durch Matlab ermittelten Klirrfaktor im wesentlichen folgt.

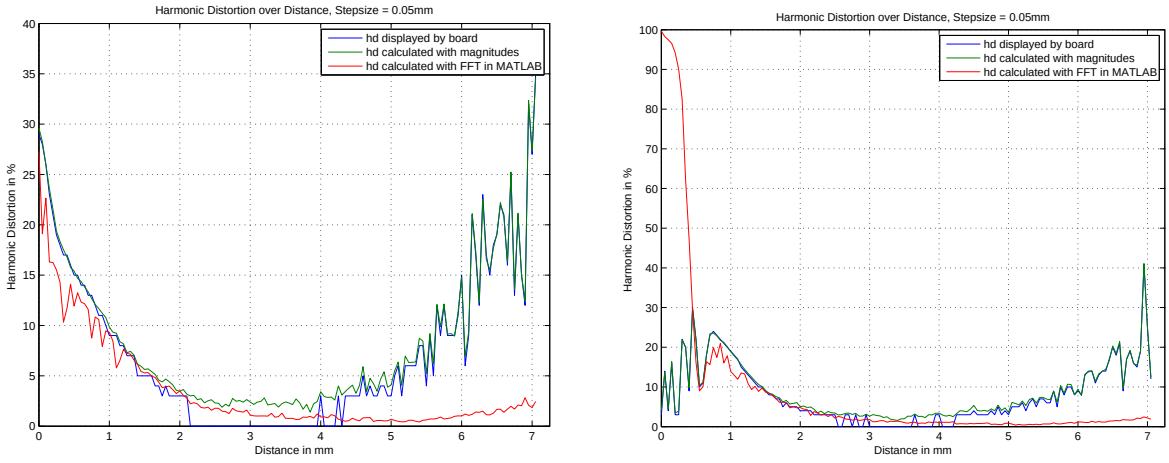


Abbildung 2.5.: Links: Klirrfaktor (HD) aufgetragen über die Distanz bei einer Verkippung von 0°
Rechts: Klirrfaktor (HD) aufgetragen über die Distanz bei einer Verkippung von -15°, entnommen aus [6]

In Abbildung 2.6 sind beispielhaft die Skizzen von Sensorsignalen zu sehen. Sie stellen schematisch das Verhalten des Sensorsignals bei einer bestimmten Konfiguration des Sensors vor dem Encoderrad über zwei Inkrementierungen dar, wobei der Abstand des Sensors vor dem Encoderrad gleich bleibt, die Verkippung allerdings bei jedem Graph variiert. Bei geringer Verkippung besitzt das Sensorsignal annähernd die Form eines Sinus ohne Oberwellen, periodisch mit der Encoderinkrementierung (blauer Graph). Wird verkippt, bilden sich durch Oberwellen zwei zusätzliche Wendepunkte zwischen den Scheitelpunkten und dem Wendepunkt an der Stelle des fallenden Nulldurchgangs aus (grüner Graph). Wird weiter verkippt bilden sich zwei zusätzliches Halbwellen aus. Der fallende Nulldurchgang des oberwellenfreien Signals (blauer Graph) wird somit zu einer steigenden Flanke (pinker und roter Graph).

Somit gilt Gleichung 3.2. Da aus der Dauer zwischen zwei aufeinanderfolgenden, aufsteigen-

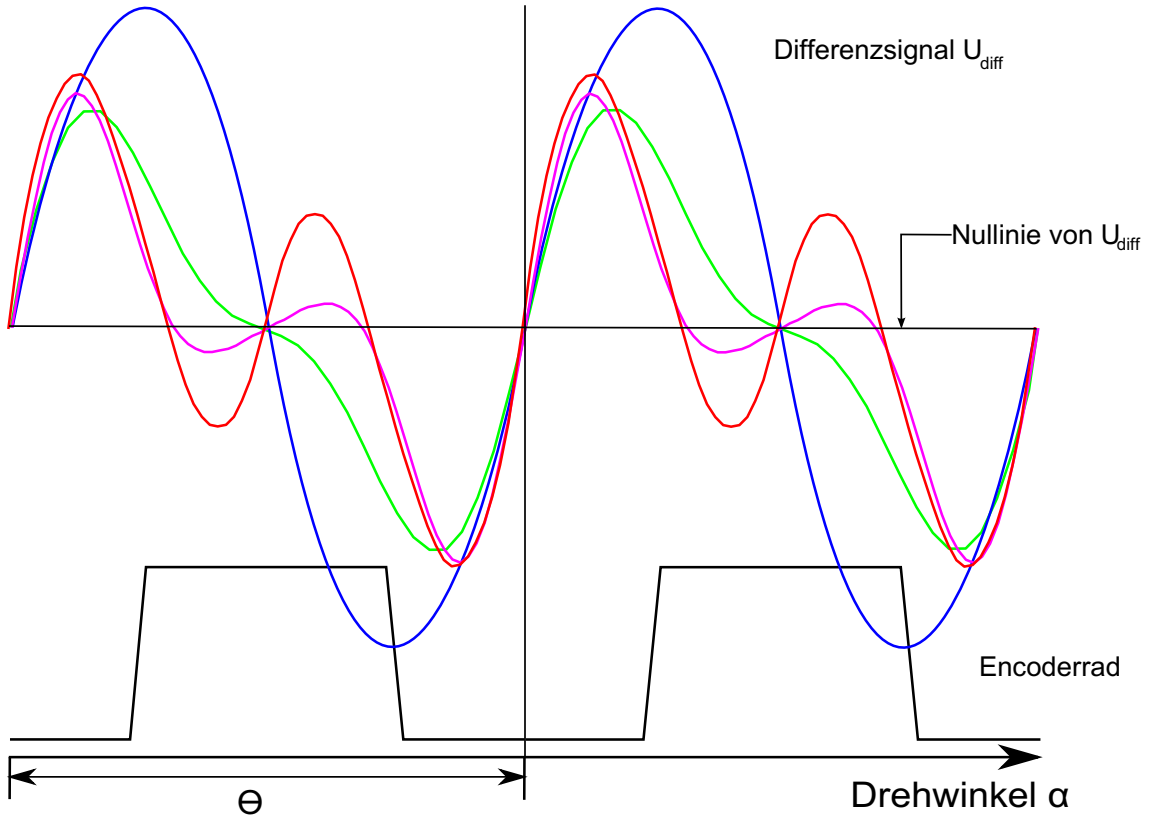


Abbildung 2.6.: Ausbildung weiterer Nulldurchgänge zwischen den Halbwellen

den Nulldurchgängen und der Anzahl der Encoderradzähne die Drehzahl des Encoderrades abgeleitet wird, ist das Eintreten dieses Sensorzustands unbedingt zu vermeiden. Ein Algorithmus, der das elektronische Eingreifen in den Bremsvorgang eines Automobils bestimmt, würde in diesem Fall zur Berechnung die doppelte tatsächliche Raddrehzahl nutzen. Dies

hätte auf den Bremsvorgang Folgen, die nicht absehbar sind. Deswegen ist es Gegenstand dieser Arbeit Möglichkeiten aufzuzeigen, die diesen Zustand identifizieren können.

2.4. Radunrundlauf und Fertigungsfehler

In der Praxis kommt es innerhalb einer Radumdrehung zu Schwankungen in den Signalamplituden, ohne dass der Sensor seine Position vor dem Encoderrad verändert. Das ist damit zu begründen, dass der Abstand zwischen Sensorkopf und Encoderradzahn über eine Umdrehung des Encoderrads differieren kann. Dies bedeutet, dass die Bedingung, formuliert in Gleichung 2.5, nicht erfüllt ist, denn die Signalform und die Signalamplitude ist damit nicht bei jeder Encoderinkrementierung gleich. Dies führt zwangsläufig bei der Berechnung der Harmonischen zu Fehlern. Mögliche Ursachen hierfür ist zum einen, dass das Encoderrad einen Unrundlauf besitzt, zum anderen kann die Ausprägung der einzelnen Zähne aufgrund von Fertigungsungenauigkeiten unterschiedlich sein. Diese Effekte verstärken sich noch, umso weiter sich der Sensor vom Encoderrad entfernt. Um das Sensorsignal abtasten zu können muss es verstärkt werden, um den Eingangsspannungsbereich des Analog-Digital-Umsetzers auszunutzen und ausreichend quantisieren zu können. Eine weitere Problematik ist diejenige, dass es innerhalb einer Radumdrehung teilweise zu einer Frequenzverdopplung kommen kann, teilweise aber auch nicht. Da das Signal wie in Abschnitt 2.1.1 beschrieben sequentiell abgetastet wird, enthält das auf dem Controller zusammengesetzte und analysierte Signal sowohl Abtastwerte aus Bereichen des Sensorsignals, an denen diese Frequenzverdopplung herrscht und als auch aus Bereichen wo dies nicht der Fall ist. Welche Folgen dies für die Analyse des Sensorsignals hat wird in dieser Arbeit dargestellt und mögliche Lösungen werden vorgestellt.

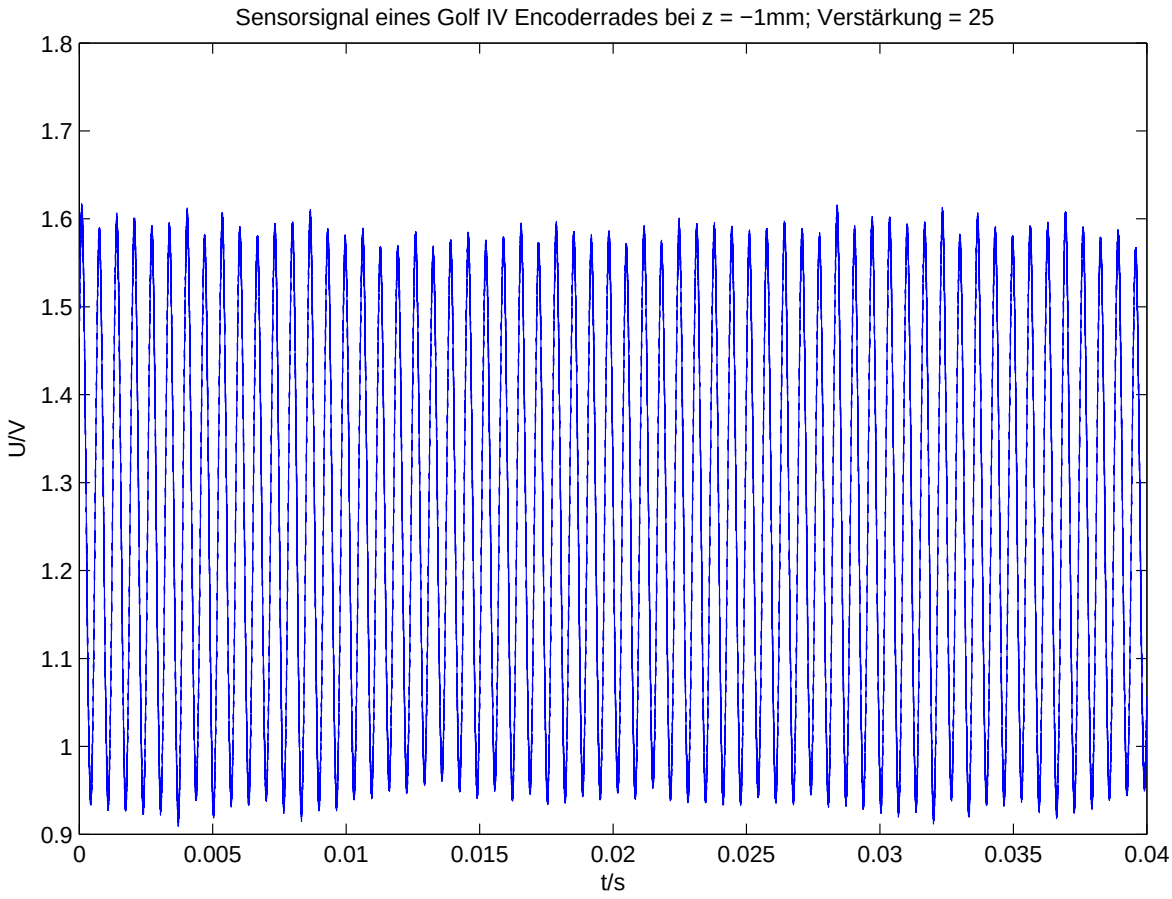


Abbildung 2.7.: Sensorsignal bei geringer Verstärkung, mit leichter Amplitudenschwankung

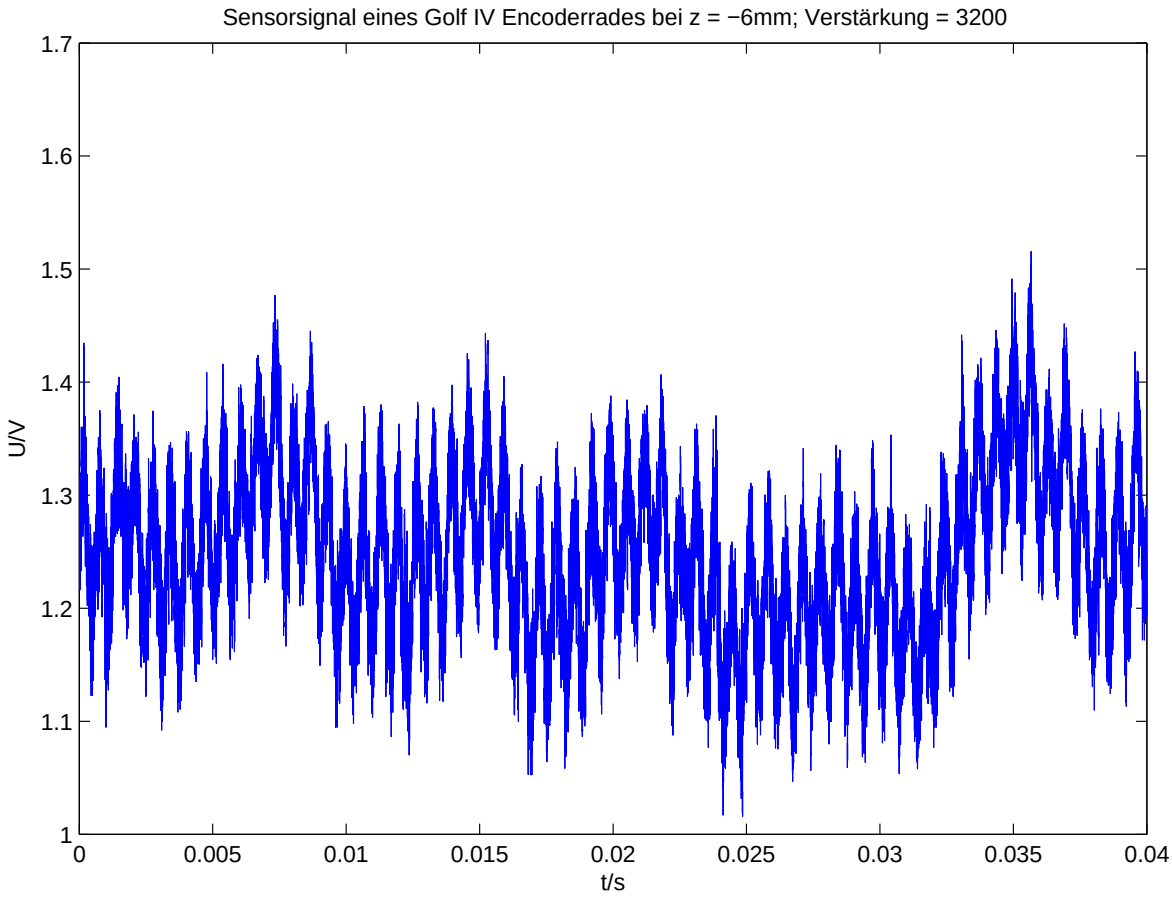


Abbildung 2.8.: Sensorsignal bei großer Verstärkung, mit starker Amplitudenschwankung

3. Lösungskonzept

3.1. Begriffsklärung

In diesem Kapitel werden zunächst einige Bezeichnungen für bestimmte Frequenzen, die in dieser Arbeit eine Rolle spielen festgelegt und erläutert in welchen Beziehungen sie zueinander stehen. Es ist zu beachten, dass sich die Raddrehzahl über die Dauer eines Abtastintervalls der sequentiellen Abtastung (siehe 2.1.1) ständig ändern kann. Darum handelt es sich bei allen unten besprochenen Frequenzen um gemittelte Werte, die über eben diese Dauer des Abtastintervalls gebildet werden.

- f_{GW} bezeichnet die Zahnfrequenz. Sie ist diejenige Frequenz, mit der sich die einzelnen Zähne des Encoderrades am Sensor vorbeibewegen. Diese hängt von der Drehzahl und der Anzahl der Zähne des Encoderrades ab.
- f_D bezeichnet die Detektionsfrequenz. Das ist diejenige Frequenz, die die Software der Experiementalplattform als Zahnfrequenz ermittelt. Auf der Experimentalplattform ist dafür eine Nulldurchgangserkennung mittels eines Smartcomparators realisiert worden.
- f_{HB} bezeichnet die Frequenz der 1. Harmonischen Schwingung, dessen Anteile die Software der Experiementalplattform berechnet.

Der Zusammenhang zwischen Detektionsfrequenz f_D und Zahnfrequenz f_{GW} unterscheiden sich, je nach Art des Sensorsignals. Es werden hierfür zwei Zustände differenziert.

- Der *Normalfall* bezeichnet den Sensorzustand, indem die Zahnfrequenz f_{GW} gleich der Detektionsfrequenz f_D ist. Es gilt also:

$$f_{GW} = f_D \tag{3.1}$$

- Die *Frequenzverdopplung* bezeichnet den Zustand, indem die detektierte Frequenz f_D doppelt so groß ist wie die Zahnfrequenz f_{GW}. Es folgt:

$$f_{GW} = \frac{1}{2} \cdot f_D \tag{3.2}$$

3.2. Anpassung des Softwaredesigns der Experimentalplattform

3.2.1. Abtastung

Wie in Abschnitt 2.1.1 beschrieben wird in der Lösung von [6] aus 64 Perioden *T* sukzessive jeweils ein Abtastwert genommen. Diese Abtastwerte werden zusammengefügt als wären sie die einer einzelnen Periode des Sensorsignals. Tritt jedoch die *Frequenzverdopplung* auf, so entspricht die detektierte Frequenz des Signals f_D nicht der Zahnfrequenz f_{GW} sondern ist doppelt so groß, da die Signalverzerrung dazu führt, dass zwei weitere Nulldurchgänge entstehen. Dadurch wird nur eine halbe Inkrementierung des Encoderrads abgetastet (Siehe 2.3). In diesem Fall ist die Frequenz der harmonischen Basis f_{HB} nicht gleich der Zahnfrequenz f_{GW} sondern der doppelt so großen Detektionsfrequenz f_D.

Um in beiden Fällen eine ganze Encoderinkrementierung aufzunehmen, bzw. durch sequentielle Abtastung zusammenzusetzen, ist es nötig, grundsätzlich zwei Detektionsperioden $T_D = \frac{1}{f_D}$ abzutasten. Das heißt, dass die 64 Abtastwerte zusammengesetzt anstatt zweier Halbwellen, vier Halbwellen ergeben. Die Abbildung 3.1 stellt schematisch dieses Verfahren anhand eines Sensorsignals bei *Frequenzverdopplung* dar.

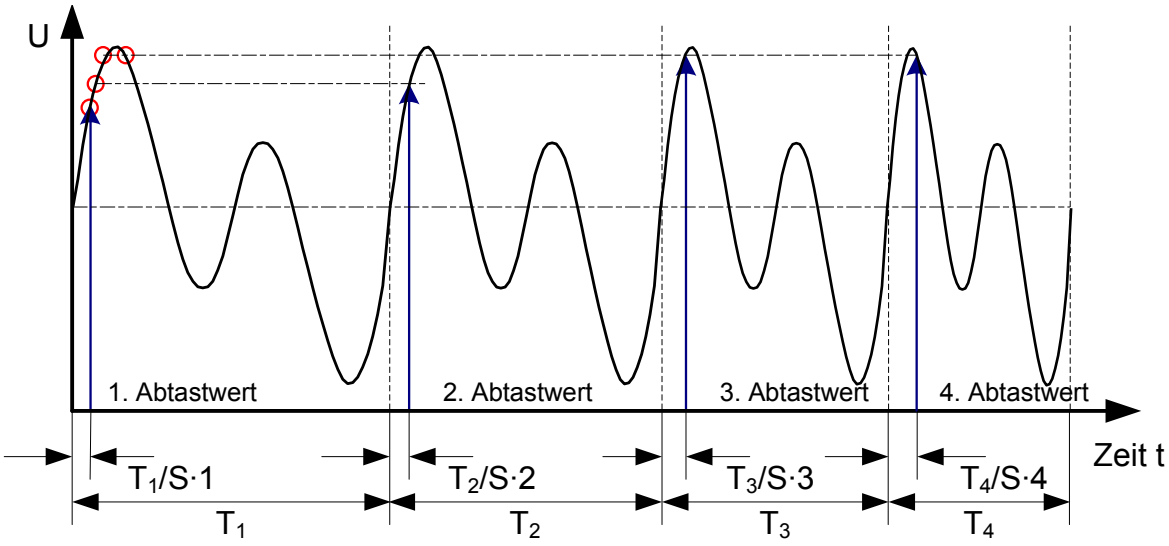


Abbildung 3.1.: Sequentielle Abtastung eines nicht periodischen Signals, bei Frequenzverdopplung

Durch diese Veränderung, verdoppelt sich die Anzahl der Encoderradinkrementationen, die für ein Abtastintervall nötig sind.

3.2.2. Harmonic Distortion 5 - HD5

Aufgrund der veränderten Abtastung muss die HD5-Methode zur Klirrfaktorbestimmung angepasst werden. Im Normalfall werden durch die modifizierte Abtastung zwei Encoderinkrementierungen aufgenommen. Die Abtastfolge bildet folglich zwei Signalperioden ab. Wird also eine DFT auf diese Abtastfolge angewendet, so dominiert die zweite Spektrallinie. Damit gilt:

$$f_{HB} = \frac{1}{2} f_D = \frac{1}{2} f_{GW} \tag{3.3}$$

Da die Frequenz der doppelten harmonischen Basis f_{HB} der Zahnfrequenz f_{GW} entspricht, können keinerlei Frequenzanteile an der Stelle von f_{HB} und deren ungeraden Vielfachen vorhanden sein (Siehe Abbildung 3.2). Folglich muss der veränderte HD5-Algorithmus dahingehend angepasst werden, dass der Grundschwingungsanteil nichtmehr der harmonischen Basis f_{HB} entspricht, sondern $f_D = 2 \cdot f_{HB}$. Außerdem müssen statt fünf, zehn Harmonische errechnet werden um den gleichen Klirrfaktor zu erhalten, wie die HD5-Methode bei der ursprünglichen Art der Abtastung. Also muss die Formel 2.17 folgendermaßen verändert werden:

$$k_{\%} = 100\% \cdot \sqrt{\frac{|U_4|^2 + |U_6|^2 + |U_8|^2 + |U_{10}|^2}{|U_2|^2 + |U_4|^2 + |U_6|^2 + |U_8|^2 + |U_{10}|^2}} \tag{3.4}$$

Der Zusammenhang von harmonischen Basis f_{HB} und Zahnfrequenz f_{GW} verändert sich allerdings, sobald die Signalverzerrung zur Frequenzverdopplung führt. Nun wird nur eine Encoderinkrementierung aufgenommen. Dadurch ist die Zahnfrequenz f_{GW} gleich der harmonischen Basis f_{HB}. Es gilt also:

$$f_{HB} = f_{GW} = \frac{1}{2} f_D \tag{3.5}$$

Es wird im Fall der Frequenzverdopplung somit nicht der selbe Klirrfaktor berechnet, wie im Normalfall, da der Algorithmus grundsätzlich davon ausgeht, dass die Grundschwinung an der Stelle von $f_D = 2 \cdot f_{HB}$ liegt.

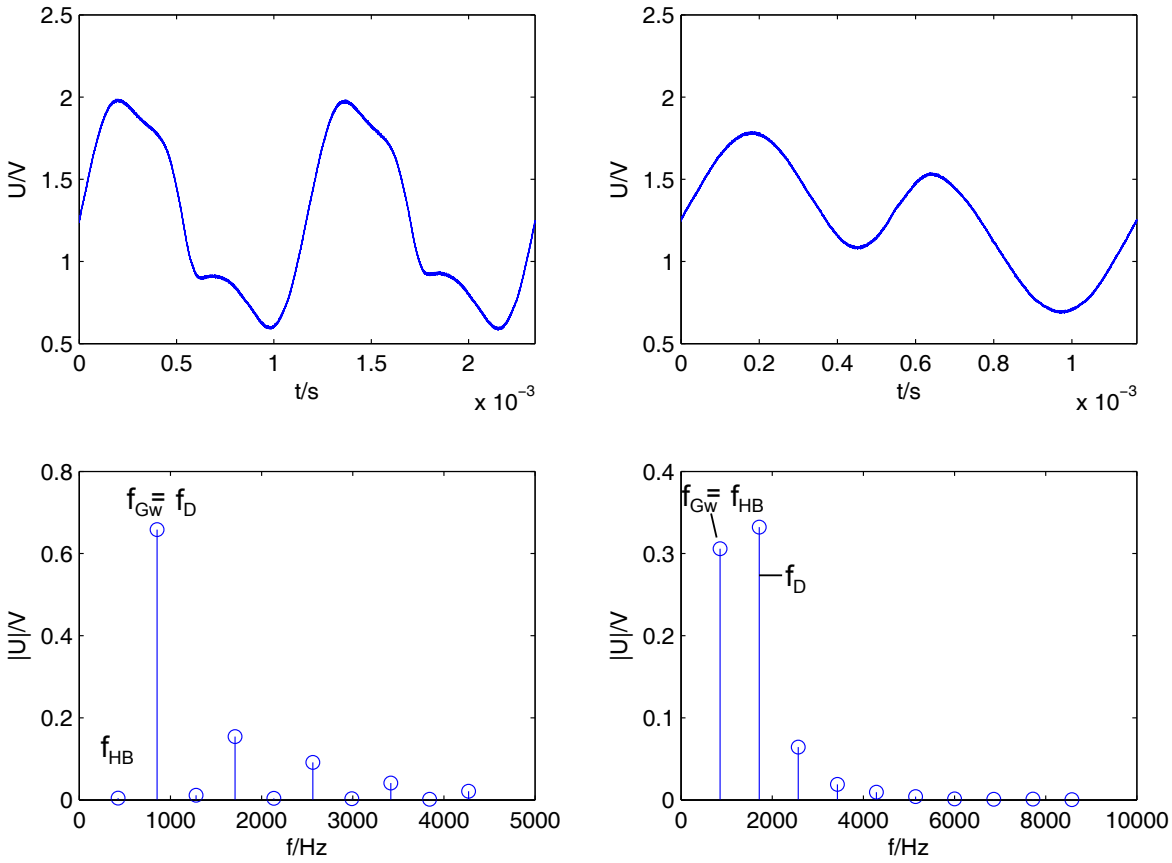


Abbildung 3.2.: Links: Stark verzerrtes Sensorsignal im ohne Frequenzverdopplung,
Rechts: Sensorsignal bei Frequenzverdopplung,
Oben: Sensorsignal, Unten: Harmonischen Spektrum

3.2.3. Harmonic Distortion Infinite - HDI

Auch die HDI-Methode muss an die veränderte Abtastung angepasst werden. Die Implementierung unterscheidet sich damit von der in Abschnitt 2.1.3 und [8] beschriebenen. Der Abschnitt 3.2.2 erläutert, dass die Grundschwingung des Sensorsignals bei dieser Abtastung auf der zweiten Frequenzlinie liegt. Das bedeutet, dass nicht wie zuvor die Leistung von f_{HB} berechnet werden muss. Stattdessen muss die Leistung von f_D berechnet und von der Gesamtleistung abgezogen werden, um die Leistung der Oberwellen der Grundschwingung zu berechnen. Damit verändert sich Gleichung 2.19:

$$k_{\%} = 100\% \sqrt{\frac{P_{ges} - P_2}{P_{ges}}} = 100\% \sqrt{\frac{P_{ob}}{P_{ges}}} \tag{3.6}$$

Es muss also statt der Leistung der ersten Harmonischen P_1 die Leistung der zweiten Harmonischen P_2 des Abgetasteten Signals berechnet werden.

Vergleicht man die Formeln 3.4 und 3.6 der Klirrfaktorberechung für HD5 und HDI und betrachtet das Harmonischen-Spektrum des Sensorsignals bei Frequenzverdopplung in Abbildung 3.2, so wird ersichtlich, dass die HDI-Methode in diesem Fall einen deutlich höheren Klirrfaktor berechnen wird als die HD5-Methode. Der stark ausgeprägte Anteil der Harmonischen Basis am Signal fließt, im Gegensatz zum HD5-Verfahren, in die dem HDI-Verfahren zugrunde liegenden Gesamtwechselleistung ein und erhöht damit das Ergebnis der Formel 3.6. Folglich wird das HDI-Verfahren die Frequenzverdopplung im Gegensatz zum HD5-Verfahren als schlechten Zustand erkennen, aber auch stärker Rauschen oder Niederfrequente-Effekte wie Radunrundlauf mit einbeziehen. Aus diesem Grund ist erwartbar, dass in der Praxis auch ohne Frequenzverdopplung der Klirrfaktor, ermittelt mit der HDI-Methode höher ausfällt im Vergleich zu HD5-Methode. Das hängt damit zusammen, das in der Praxis das Sensorbrückensignal immer irgendwelche Anteile besitzen werden, deren Frequenz nicht auf einem Vielfachen der Zahnfrequenz liegen.

3.2.4. Odd Harmonic Distortion - OHD

In der Abbildung 3.2 und in Abschnitt 3.2.2 angedeutet, haben die Abtastfolgen von *Normalfall* und *Frequenzverdopplung* spektral unterschiedliche Charakteristiken. So sind im Normalfall grundsätzlich keine Anteile an der Stelle von f_{HB} und deren ungeraden Vielfachen vorhanden. Darum bietet es sich an einen Indikator zu definieren, an dem ablesbar ist welcher Fall vorliegt. Sinnvollerweise sollte die Berechnung dieses Indikators auf den vorgestellten Methoden HD5 und HDI aufsetzten.

Es bietet es sich an, zur Bildung des Indikators die auf der Experimentalplattform berechneteten Harmonischen in ein Verhältnis zueinander zu setzen, ähnlich des des Klirrfaktors. Da wie oben beschreiben im *Normalfall* keine Anteile an der Stelle der Harmonischen Basis und deren ungeraden Vielfachen vorhanden sind, bei einer *Frequenzverdopplung* allerdings schon, scheint es sinnvoll, das Verhältnis aus der Summe der Beträge der ungeraden Harmonischen und den der geraden Harmonischen zu bilden. Dieser Indikator wir deswegen als *Odd-Even-Harmonic-Distortion* benannt, abgekürzt als OEHD. Mathematisch ausgedrückt lautet der OEHD:

$$OEHD_{\%} = 100\% \cdot \sqrt{\frac{\sum_{k=0}^{\infty} U_{2k+1}^2}{\sum_{i=1}^{\infty} U_{2i}^2}} \tag{3.7}$$

Dies ist aber nur implementierbar mit der Harmonischen Analyse der modifizierten HD5-Methode, da sie die Anteile der ersten zehn Harmonischen berechnet. Für den OEHD müssten also diese Harmonischen nur in das oben beschriebene Verhältnis gesetzt werden. Der HDI, berechnet wie in Abschnitt 2.1.3 beschrieben die Leistung des Signals und nur den Anteil einer Harmonischen und ist damit nicht in der Lage den OEHD zu berechnen. Um also den Indikator auch mit der HDI-Methode berechnen zu können und das Ergebnis mit dem der HD5-Methode vergleichen zu können, ist es nötig ihn auf eine andere Weise zu definieren. Um die Stärke des geringeren Rechenaufwands des HDI im Vergleich zum HD5-Algorithmus möglichst beizubehalten, sollte dieser Indikator im Nenner die Gesamtleistung bzw. die Summierung aller berechneten Harmonischen enthalten und im Zähler möglichst wenige einzelne harmonische Anteile aufsummieren. Diese Vorgaben legen folgende mathematische Formulierung nahe.

$$OHD_{\%} = 100\% \cdot \sqrt{\frac{U_1^2 + U_3^2}{\sum_{i=1}^{10} U_i^2}} \approx 100\% \cdot \frac{\sqrt{U_1^2 + U_3^2}}{P_{ges}} \tag{3.8}$$

Für diese Berechnung muss die HDI-Methode zusätzlich zwei weitere Harmonische berechnen. Der erweiterte HD5-Algorithmus berechnet ohnehin die ersten zehn Harmonischen. Es ist also lediglich nötig ihn um die Berechnung des OHD zu erweitern.

Allerdings wird die Vereinfachung aus Formel 3.8, die die Summe der zehn Harmonischen mit der Gesamtwechselleistung gleich setzt, dazu führen, dass der OHD, ermittelt durch das HDI-Verfahren, geringer ausfällt als der OHD ermittlet mit der HD5-Methode. Die Gesamtwechselleistung wird in der Regel größer sein, als die Summer der zehn Harmonischen, da sie Rauschleistung sowie Anteile unterhalb und oberhalb der zehn Harmonischen einbezieht.

3.2.5. Discard Sample Counter - DSC

Wie in Abschnitt 2.4 beschrieben, kann es dazu kommen, dass sich die Signalform aufgrund von Radunrundlauf und Deformation von Encoderradzähnen während einer Radumdrehung stark verändert. Dies führt zu einem Sensorzustand, indem es innerhalb einer Radumdrehung nur teilweise zur Frequenzverdopplung kommt. Ein solches Signal ist in Abbildung 3.3 zu sehen. Dieser Zustand ergibt sich bei einer ganz bestimmten Sensorpositionierung. Diese liegt in dem Bereich, indem die Verzerrung des Sensorsignals gerade beginnt zusätzliche Nulldurchgänge zu verursachen. Dieser Zustand ist schwer zu detektieren, da nicht genau vorausgesagt werden kann, aus welchem Bereichen der Encoderradumdrehung Abtastwerte entnommen werden.

Ein Indikator für diesen Zustand ergibt sich daraus, dass wie in Abschnitt 2.1.1 beschrieben, Abtastwerte verworfen werden, sobald die Signalperiode aus der ein Abtastwert entnommen wird, zeitlich signifikant von der vorherigen abweicht. Wird während eines Abtastintervalls mitgezählt wie häufig ein Abtastwert verworfen wird, so lässt sich bei einer hohen Anzahl darauf schließen, dass sich der Sensor in der oben beschriebenen Position befindet.

Im Normalfall, unter Verwendung eines idealen Encoderrads, kann es allenfalls bei einer relativ geringen Geschwindigkeit zu so starken Abweichungen der Dauer von aufeinanderfolgenden Perioden kommen, dass Abtastwerte verworfen werden müssen. Siehe hierfür [6].

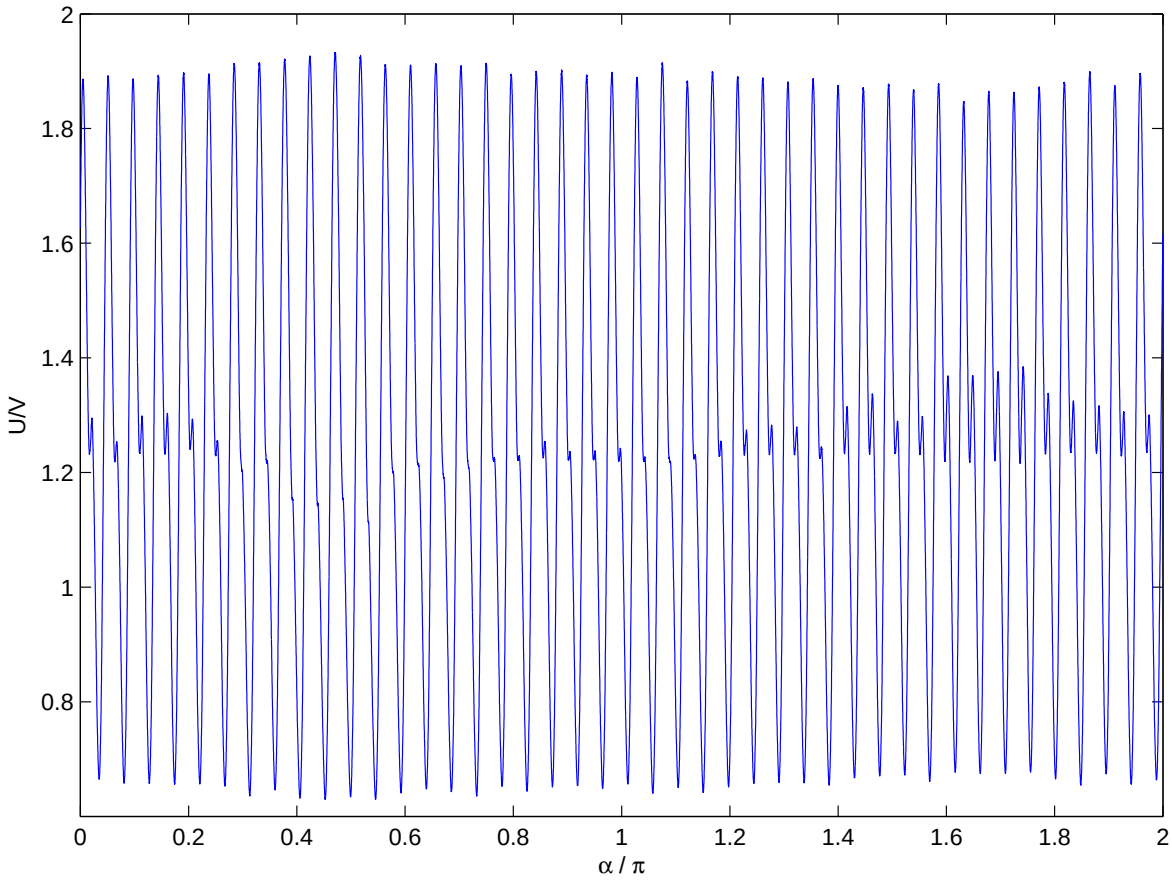


Abbildung 3.3.: Sensorsignal im Grenzbereich zwischen Frequenzverdopplung und Normalfall

4. Realisierung

4.1. Anpassung der Abtastung

In der ursprünglichen Implementierung von [6] wurde ein Zustandsautomat mit sechs Zuständen für die Signalabtastung realisiert. Dieser ist in der Interrupt Service Routine (ISR) *isr_comp_a.c* des Comparators implementiert. Die Zustände werden mit jedem erkannten steigenden Nulldurchgang des Sensorsignals durchlaufen. Dies ist durch eine Switch-Case anweisung realisiert worden.

- *S1_Prepare_Measure_Periode*: In diesem Zustand wird der Taktvorteiler für den *Timer B* so bestimmt, dass der Quantisierungsfehler bei der Messung von zwei Sensorperioden möglichst gering ist, und ohne dass es zu einem Zählerüberlauf kommt. Dafür wird im Übergeordneten Zustandsautomat, mittels des *Timer A*, laufend eine grobe Messung der Periodendauer des Signals vorgenommen.
- *S2_Measure_Periode_Start*: Hier wird der *Timer B* gestartet.
- *S3_Measure_Periode_End*: *Timer B* wird angehalten und ausgelesen. Außerdem wird ein Interruptanforderrung durch den *Port1* ausgelöst, in dessen ISR der nächste Abtastzeitpunkt berechnet wird. Kommt es zu einem Überlauf des *Timer B*, so wird der Automat zurückgesetzt.
- *S4_Take_Sample_Start*: Der Analog-Digital-Umsetzer (*ADC*) wird aktiviert und der *Timer B* wird wieder gestartet.
- *S5_Take_Sample_End*: Der *Timer B* wird gestoppt, erst jetzt wird der Abtastwert in der ISR des ADC verarbeitet. Dies ist nötig um überprüfen zu können, in wie weit sich die Periodendauer, die Grundlage für die Berechnung des Abtastwertes war, und die Dauer der Periode aus der der Abtastwert entnommen wurde, unterscheiden. Bei einer Abweichung von mehr als 3,125% wird der aktuelle Messwert verworfen. Im folgenden Durchlauf des Automatens wird nocheinmal versucht einen gültigen Abtastwert aufzunehmen.
- *S6_Process_Sample*: In diesem Zustand wird überprüft, ob die Verarbeitung des Abtastwertes abgeschlossen ist. Ist dies der Fall, so ist der Folgezustand S1, andernfalls wird im Zustand S6 verweilt.

Diese Zustandsautomat ist nicht ausreichend, sobald die Verzerrung derart stark wird, dass es zu zusätzlichen Nulldurchgängen kommt. Eine Encoderinkrementierung würde nun nicht-mehr zwei, sondern vier Nulldurchgänge erzeugen. Somit wird nur eine halbe Signalperiode abgetastet, bestehend aus zwei Halbwellen mit unterschiedlichen Amplituden. Da der Zustandsautomat im Zustand S6 eine unbestimmte Anzahl an Signalperioden verbleibt, abhängig davon wie viele Signalperioden die Sampleverarbeitung dauert, ist es von der Zahnfrequenz abhängig wieviel Nulldurchgänge zwischen den Perioden liegen aus denen ein Abtastwert entnommen wird (Siehe Abbildung 4.3). Würde folglich der Zustandsautomat im Zustand S6 eine gerade Anzahl an steigenden Nulldurchgängen verweilen, so würde abwechselnd aus den Halbwellen mit unterschiedlichen Amplituden ein Abtastwert entnommen. Dadurch würde also nicht nur keine ganze Encoderinkrementierung aufgenommen werden, sondern die Abtastwerte wären zusätzlich nicht in der richtigen Reihenfolge. Dieser Fall ist in Abbildung 4.1 rechts dargestellt. Links in Abbildung 4.1 befindet sich das dazugehörige abgetastete Signal.

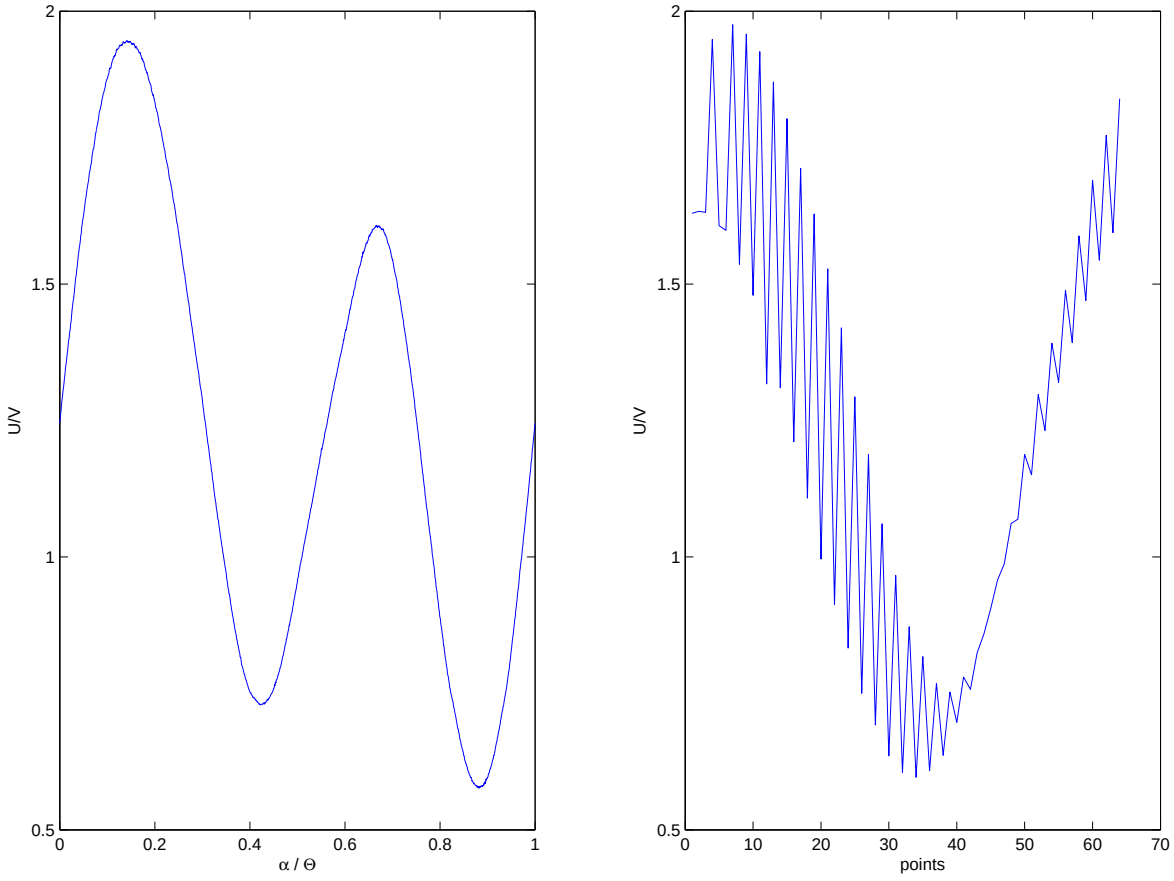


Abbildung 4.1.: Links: Sensorsignal einer Encoderinkrementierung bei Frequenzverdopplung, Rechts: Ergebnis der ursprünglich implementierten Abtastung

Wie in Abschnitt 3.2.1 beschrieben, bietet sich als Lösung an, zwei Signalperioden abzutasten. Im Falle der *Frequenzverdopplung* wird folglich eine Encoderinkrementierung abgetastet, im *Normalfall* sind es hingegen zwei. Um dies zu erreichen muss die ursprüngliche Implementation angepasst werden, indem zwei zusätzliche Zustände eingefügt wurden.

- *S2a_Measure_Periode_Middle* dient dazu, eine steigende Flanke später den *Timer B* zu stoppen. Das beeinflusst die Berechnung des folgenden Abtastzeitpunktes in Zustand *S3_Measure_Periode_End* dahingehend, dass die Schrittweite doppelt so groß wird. Bei gleicher Anzahl an Abtastpunkten S sind diese Abtastzeitpunkte auf den doppelten Signalbereich im Vergleich zur Ursprungsimplementation ausgelegt.
- *S4a_Take_Sample_Middle* wird benötigt, damit auch der Bereich, aus dem ein Abtastwert entnommen werden kann, zu verdoppeln.

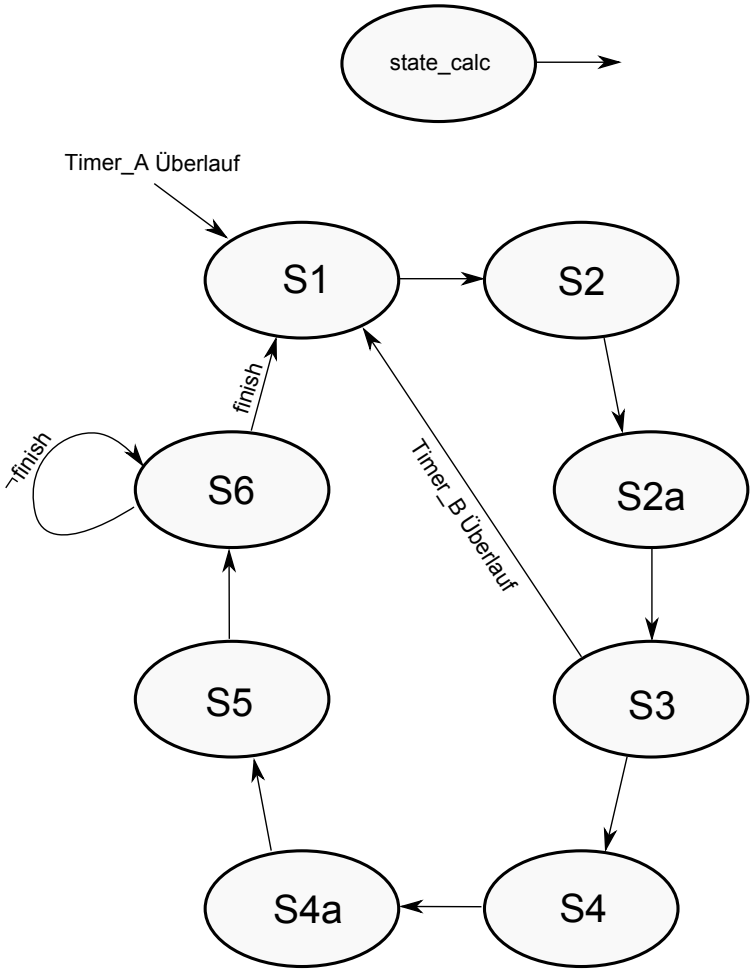


Abbildung 4.2.: Zustandsdiagramm des Automaten *state_calc*, der Zustandswechsel erfolgt bei jeder steigenden Nulldurchgang des Sensorsignals

Um sicherzustellen, dass zwei die Abtastwerte in eine sinnvolle Reihenfolge gebracht werden, ist es schlicht nötig, dass der Zustandsautomat eine ungerade Anzahl an steigenden Flanken im Zustand S6 verweilt (Siehe Abbildung 4.3). Das wird dadurch realisiert, dass die Variable *even_finish* jedes Mal inkrementiert wird, wenn S6 erreicht wird, ohne dass die Sampleverarbeitung abgeschlossen ist. Ist die Sampleverarbeitung schließlich abgeschlossen, aber der Zählerwert ist gerade, so bleibt der Zustandsautomat ein weiteres Mal im Zustand S6. Erst danach wird dieser Zähler auf 1 zurückgesetzt und in den ersten Zustand gewechselt. Im folgenden der Auschnitt aus der ISR des Comparators, der die Case-Anweisung des Zustands S6 betrifft. Dabei ist *finish* das angesprochene Flag, das gesetzt wird, sobald die Sampleverarbeitung abgeschlossen ist.

```
case S6_Process_Sample:
            // Warten bis Verarbeitung des Samples fertig.
            // Und ungerade Anzahl an Messwerten
            // Verarbeitung in der ISR Routine des ADC12
    if(g_calc.finish == TRUE && (g_calc.even_finish & 0x1)) {
        g_calc.finish = FALSE;
        g_calc.even_finish = 1;
        state_calc = S1_Prepare_Measure_Periode;
    }
    else
        //Häufigkeit des Verweilens in State S6 mitzählen
        g_calc.even_finish++;
    break;
```

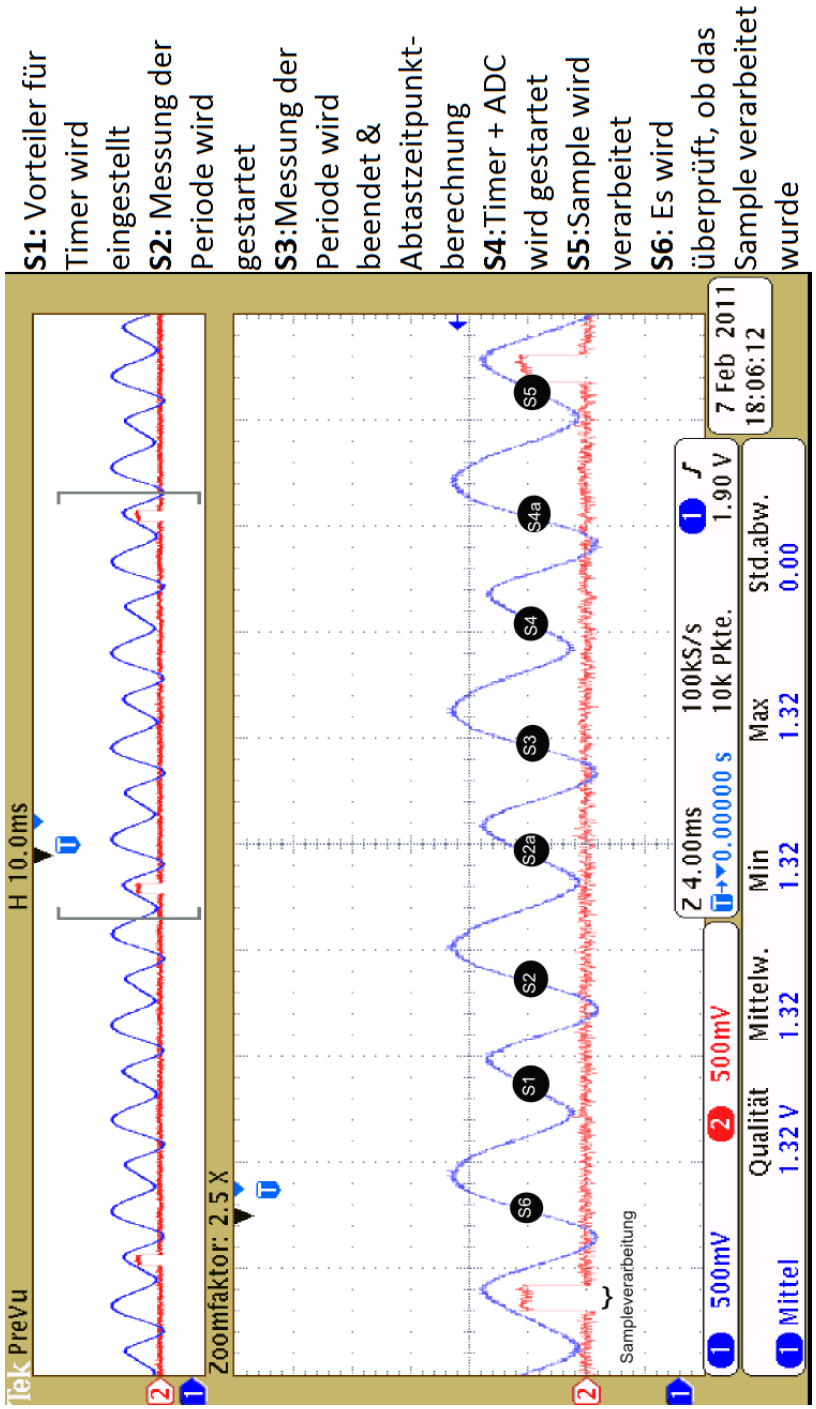


Abbildung 4.3.: Sensorzustände im Signalverlauf

4.2. Anpassung des HD5

Der in der Arbeit [6] implementierte HD5 Algorithmus muss lediglich an die veränderte Abtastung angepasst werden. Die Berechnung der zehn Harmonischen, die benötigt werden (siehe Abschnitt 3.2.2) erfolgt schlicht, indem die Konstante ḦARMONICSëntsprechend eine zehn zugewiesen wird. Zur Berechnung des Klirrfaktors müssen die errechneten Harmonischen entsprechend Gleichung 3.4 in ein Verhältnis gesetzt werden. Nähere Erläuterungen zur Implementierung des HD5-Verfahrens sind in [6] nachlesbar.

4.3. Implementation des HDI

Die von Lennard Koch entwickelte HDI-Methode zur Berechnung des Klirrfaktors wurde in einer Beispielimplementierung auf einem Entwicklungsboard mit einem MSP430 „MSP430-169STK“ realisiert [8]. Diese Implementierung war angepasst an die Abtastung in der ursprünglichen Form wie sie in der Arbeit von Jegenhorst [6] implementiert wurde. Aufgrund dessen und zur Berechnung des OHD wurden diese Implementierung etwas verändert und erweitert. Außerdem wurden einige Optimierungen im Programmablauf vorgenommen.

4.3.1. clac_HD_noise_sampling

Die erste Veränderung betrifft die Methode *calc_HD_noise_sampling*, welche nach jeder Aufnahme eines Samplewertes in der ISR des ADC aufgerufen wird. In dieser Funktion wird der ADC Wert, sowie dessen Quadrat in den Variablen *x_gl* bzw. *l_ges* über ein Abtastintervall aufsummiert. Dieses dient zur späteren Berechnung der Gesamtwechselleistung nach Formel 2.28 und unterscheidet sich nicht von der Ursprungsimplementation [8]. Des Weiteren werden in dieser Funktion Real- und Imaginärteile der ersten drei Harmonischen jeweils aufsummiert. Dies unterscheidet sich von der Ursprungsimplementation, in der nur die Real- und Imaginärteile der ersten Harmonischen aufsummiert wurden. Für Real- bzw. Imaginärteil gilt:

$$\Re[h] = \sum_{k=0}^{N-1} s(k) \cdot cos(\omega \cdot h \cdot k) \tag{4.1}$$

bzw.

$$\Im[h] = \sum_{k=0}^{N-1} s(k) \cdot sin(\omega \cdot h \cdot k) \tag{4.2}$$

Für die Lösung der Sinus- und Kosinusfunktion wurden Werte in eine Look-Up-Table (LUT) abgelegt, die schon in der Implementierung des HD5 von Jegenhorst [6] verwendet wurde

und 32 Werte umfasst. Da der Kosinus zum Sinus nur um 90° bzw. $\frac{\pi}{2}$ verschoben ist, wird für beide Funktionen nur eine LUT benötigt. Diese erhält allerdings bei der Lösung des Kosinus einen Index dem immer *IDX_PI_H* hinzuaddiert wird. Da die 32 hinterlegten Werte im Gradmaß 180° bzw. π im Bogenmaß abdecken, besitzt *IDX_PI_H* entsprechend für die Verschiebung von 90° bzw. $\frac{pi}{2}$ den Wert 16. Um 360° der beiden trigonometrischen Funktionen abzudecken reichen diese 32 Werte, da die Ergebnisse für die verbleibenden 180° sich aufgrund der Symmetrie dieser Funktionen nur durch das Vorzeichen von den restlichen unterscheiden. Beim Zugriff auf den LUT werden dann nur die fünf LSB („Least significant bit“) betrachtet mittels einer And-Maskierung, was einer Modulo-32-Operation entspricht. Dies stellt eine geringfügige Optimierung des Programmablaufs gegenüber der Koch-Imlementierung dar. In dieser wurde der Modulo-Operator verwendet, dessen Ausführungen für den Microcontroller wesentlich aufwendiger ist als eine einfache Bit-Maskierung. Allerdings ist es möglich, dass der Compiler diese Optimierung automatisch vornimmt.

Die Kreisfrequenz ω und der Faktor k werden in der Variable *lut_pos* zusammengefasst, diese wird am Ende der Funktion inkrementiert und nach der Aufnahme des letzten Abtastwerts eines Abtastintervalls auf 0 zurückgesetzt. Dies ist möglich, da die LUT an die 64 Samplewerte angepasst ist und diese durch die drehwinkelsynchrone Abtastung immer zwei bzw. im Ausnahmefall eine Signalperiode umfasst (Siehe Abschnitt 2.1.1 sowie 3.2.1). Um also den Real- und Imaginärteil der ersten drei Harmonischen zu berechnen wird eine Schleife dreimal durchlaufen. Dabei wird der Index der Harmonischen h berücksichtigt, indem am Anfang der Schleife die Variable *lut_pos* mit dem Schleifenindex *i+1* multipliziert wird und das Ergebnis in der Variblen *lut_pos_harmonic* gespeichert wird. Da der LUT nur eine Halbwelle der Sinusfunktion abbildet, muss berücksichtigt werden, dass die Sinusfunktion zwischen 0° und 180° positiv und zwischen 180° und 360° negativ ist. Entsprechend ist der um 90° verschobene Kosinus zwischen 90° und 270° negativ und im restlichen Bereich positiv. Wie die Tabelle 4.1 zeigt, spiegelt das sechste Bit der Variablen *lut_pos_harmonic* wider, in welchen Bereichen des Abtastintervalls das Ergebnis der Sinusfunktion im positiven bzw. negativen Bereich liegt. Dabei entspricht eine „0“ dem positiven Bereich und eine „1“ dem negativen Bereich. Da eine Periode der ersten Harmonischen genau in die 64 Abtastwerte passt, sind die ersten 32 aufeinanderfolgenden Werte positiv und die restlichen negativ. Für die dritte Harmonische wechseln sich diese Bereiche sechs Mal einander ab, da drei Perioden der dritten Harmonischen in die 64 Abtastwerte passen. Damit ebenfalls für die Entscheidung in welchem Bereich die Kosinusfunktion einer bestimmten Harmonischen liegt nur das sechste Bit von *lut_pos_harmonic* ausgewertet werden muss, ist es lediglich nötig zuvor *IDX_PI_H* zu addieren, um die Phasenverschiebung auszugleichen. Quelltext der Funktion *clac_HD_noise_sampling*:

```
1 void calc_HD_noise_sampling(void){
    // **** Neue Implementierung
3   // Berechnung von Gleichanteil und 1-3Harmonischer während der
        Datenaufnahme
```

```
    // Gleichanteil;
    // max. Wortbreite überprüfen
    // w_ADC * ld(64) = 12 + 6 = 18 Bit also 32 Bit Register ausreichend
    THD_calc.x_gl += (int32_t) g_adc.u_diff_b;  //Summe aller Samples
    // Leistung Gesamtsignal mit Gleichanteil
    // max Wortbreite = 2 * w_ADC + ld(64) = 24 + 6 = 30 => 32 Bit
       ausreichend
    THD_calc.x0_quad = (int32_t) g_adc.u_diff_b * g_adc.u_diff_b;
    THD_calc.l_ges += THD_calc.x0_quad;
    //1.-3. Harmonische, Real und Imaginärteil
    // max.Wortbreite überprüfen
    // w_ADC + w_LUT +ld(64) = 12 + 10 + 6 = 28 Bit => 32 Bit Register
       ist ausreichend
    for(THD_calc.i = 0; THD_calc.i < 3; THD_calc.i++){
        THD_calc.lut_pos_harmonic = THD_calc.lut_pos * (THD_calc.i+1);
        // Realteil errechnen
        THD_calc.prod = (int32_t) g_adc.u_diff_b * table_sin[(THD_calc.
           lut_pos_harmonic  + IDX_PI_H) & 0x1F];
        // wenn zwischen 0 und pi/2 oder zwischen 3pi/2 und pi
        if ((THD_calc.lut_pos_harmonic + IDX_PI_H) & 0x20){
            // Cosinus ist positiv
            THD_calc.real[THD_calc.i] += THD_calc.prod;
        }
        else
            // Cosinus ist negativ
            THD_calc.real[THD_calc.i] -= THD_calc.prod;

        // Imaginärteil errechnen
        THD_calc.prod = (int32_t) g_adc.u_diff_b * table_sin[THD_calc.
           lut_pos_harmonic  & 0x1F];
        //zwischen 0 und pi oder pi und 2*pi
        if (THD_calc.lut_pos_harmonic & 0x20)
            // Sinus ist Positiv
            THD_calc.imag[THD_calc.i] += THD_calc.prod;
        else
            //Sinus ist Negativ
            THD_calc.imag[THD_calc.i] -= THD_calc.prod;
        }
    THD_calc.lut_pos++;
}
```

	1	1	h 2	2	3	3
lut_pos	sin	cos	sin	cos	sin	cos
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	1
7	0	0	0	0	0	1
8	0	0	0	1	0	1
9	0	0	0	1	0	1
10	0	0	0	1	0	1
11	0	0	0	1	1	1
12	0	0	0	1	1	1
13	0	0	0	1	1	1
14	0	0	0	1	1	1
15	0	0	0	1	1	1
16	0	1	1	1	1	0
17	0	1	1	1	1	0
18	0	1	1	1	1	0
19	0	1	1	1	1	0
20	0	1	1	1	1	0
21	0	1	1	1	1	0
22	0	1	1	1	0	0
23	0	1	1	1	0	0
24	0	1	1	0	0	0
25	0	1	1	0	0	0
26	0	1	1	0	0	0
27	0	1	1	0	0	1
28	0	1	1	0	0	1
29	0	1	1	0	0	1
30	0	1	1	0	0	1
31	0	1	1	0	0	1
32	1	1	0	0	1	1
33	1	1	0	0	1	1
34	1	1	0	0	1	1
35	1	1	0	0	1	1
36	1	1	0	0	1	1
37	1	1	0	0	1	1
38	1	1	0	0	1	0
39	1	1	0	0	1	0
40	1	1	0	1	1	0
41	1	1	0	1	1	0
42	1	1	0	1	1	0
43	1	1	0	1	0	0
44	1	1	0	1	0	0
45	1	1	0	1	0	0
46	1	1	0	1	0	0
47	1	1	0	1	0	0
48	1	0	1	1	0	1
49	1	0	1	1	0	1
50	1	0	1	1	0	1
51	1	0	1	1	0	1
52	1	0	1	1	0	1
53	1	0	1	1	0	1
54	1	0	1	1	1	1
55	1	0	1	1	1	1
56	1	0	1	0	1	1
57	1	0	1	0	1	1
58	1	0	1	0	1	1
59	1	0	1	0	1	0
60	1	0	1	0	1	0
61	1	0	1	0	1	0
62	1	0	1	0	1	0
63	1	0	1	0	1	0

Tabelle 4.1.: Darstellung des sechsten Bit des Ausdrucks „lut_pos_harmonic“(sin) bzw. „lut_pos_harmonic + IDX_PI_H“(cos)

4.3.2. calc_HD_noise_after

Nach Vollendung eines Abtastintervalls wird die Methode *calc_HD_noise_after* aufgerufen. Sie berechnet zunächst die Gesamtwechselleistung P_{ges} nach Gleichung 2.28. Des Weiteren werden aus den aufsummierten Real- und Imaginärteilen der Harmonischen deren Betragsquadrate berechnet, welche proportional zu deren Leistungen ist. Durch Bit-Schieben wird erreicht, dass die Gesamtwechselleistung und die Betragsquadrate der Harmonischen den gleichen Skalierungsfaktor besitzen, sodass er sich bei der Klirrfaktorberechnung (Siehe Formel 2.19) herauskürzen. Nähere Erläuterungen zu den Schiebefaktoren findet man in der Arbeit von Koch [8]. Für das Betragsquadrat gilt:

$$|X|^2 = \Re^2 + \Im^2 \tag{4.3}$$

Um die Beträge der ersten drei Harmonischen zu bilden wurde ein weiteres Mal eine For-Schleife verwendet, in der jeweils der Real- und Imaginärteil der Harmonischen quadriert und danach summiert wird. Die Ergebnisse werden entsprechend das Arrays *I* gespeichert. Danach kann die Leistung der Oberwellen aus der Gesamtwechselleistung (*I_ges*) abzüglich der Leistung der zweiten Harmonischen (*I[1]*) berechnet werden um schließlich diese durch die Gesamtwechselleistung zu teilen. Das Ergebnis wird der Variablen *factor* übergeben. Genauso wird die Leistung der dritten Harmonischen (*I[2]*) zur Leistung der ersten hinzuaddiert (*I[1]*) und durch die Gesamtwechselleistung geteilt, woraufhin es der Variable *prod* übergeben wird. Zur Berechnung des Klirrfaktors nach Gleichung 3.6 und zur Berechnung des OHD fehlt nun noch jeweils eine Radizierung, dies geschieht mittels einer LUT in der Funktion *decode_LUT*. Quelltext der Funktion *calc_HD_noise_after*:

```
1  void calc_HD_noise_after(void){
2
3      THD_calc.x_gl >>= 3;     // 12 +6 Bit - 15 Bit
4      THD_calc.I_ges >>= SHIFT_N; //
5
6      // Gleichanteil Quadrieren und skallieren
7      // Schiebefaktor = w_Pges / 2 - w_xgl = 15 - 18 = -3
8
9      THD_calc.x0 = (int16_t) THD_calc.x_gl;
10     THD_calc.x0_quad = (int32_t) THD_calc.x0 * THD_calc.x0;
11
12     THD_calc.x0_quad >>= SHIFT_N;            // 2 * (SHIFT_N - 3)
13
14     THD_calc.I_ges -= THD_calc.x0_quad;
15
16     //Leistung der 1. und 2. Harmonischen  aus DFT Ergebnis errechnen
17     for(THD_calc.i = 0; THD_calc.i < 3; THD_calc.i++){
18         // Realteil auf untere 16 Bit schieben
```

```
19          THD_calc.real[THD_calc.i] = THD_calc.real[THD_calc.i] >>
                SHIFT_QUAD;
            THD_calc.x0 = (int16_t) THD_calc.real[THD_calc.i];
21          // Realteil quadrieren
            THD_calc.real_quad = (int32_t)THD_calc.x0 * THD_calc.x0;
23          // Imaginärteil auf untere 16 Bit schieben
            THD_calc.imag[THD_calc.i] = THD_calc.imag[THD_calc.i] >>
                SHIFT_QUAD;
25          THD_calc.x0 = (int16_t) THD_calc.imag[THD_calc.i];
            // Imaginärteil quadrieren
27          THD_calc.imag_quad = (int32_t)THD_calc.x0 * THD_calc.x0;
            THD_calc.l[THD_calc.i] = THD_calc.real_quad + THD_calc.imag_quad;
29          // entspricht  shift_quad
            THD_calc.l[THD_calc.i] = THD_calc.l[THD_calc.i] >> (SHIFT_N +1);
31      }
        //Leistung aller Oberschwingungen errechnen
33      THD_calc.P_ob = THD_calc.l_ges - THD_calc.l[1];

35      // Für Klirrfaktor Berechnung
        if (THD_calc.P_ob > 0)
37          THD_calc.factor = (((uint64_t)THD_calc.P_ob) << 14) / THD_calc.
                l_ges;
        else
39          THD_calc.factor = 0;

41      THD_calc.prod = (((uint64_t)THD_calc.l[0] + THD_calc.l[2]) << 14) /
            THD_calc.l_ges;

43      // Zahnfrequenz für die Ausgabe berechnen, da zwei Signalperioden
            gemessen werden
        // wird g_time.tt_avg halbiert
45      g_time.frequenz = (uint16_t)(100000000000 /((uint64_t)(g_time.tt_avg
            >> 1) * TIMER_B_PER_F));
    }
```

Divisionsergebnis											Bereich	Intervall
0	0	0	0	0	0	0	0	x	y	z	1	xyz
0	0	0	0	0	0	0	1	x	y	z	2	xyz
0	0	0	0	0	0	1	x	y	z	-	3	xyz
0	0	0	0	0	1	x	y	z	-	-	4	xyz
0	0	0	0	1	x	y	z	-	-	-	5	xyz
0	0	0	1	x	y	z	-	-	-	-	6	xyz
0	0	1	x	y	z	-	-	-	-	-	7	xyz
0	1	x	y	z	-	-	-	-	-	-	8	xyz
1	x	y	z	-	-	-	-	-	-	-	9	xyz

Tabelle 4.2.: Umcodierung des Divisionsergebnis

4.3.3. decode_LUT

Für die Lösung der Quadratwurzel hat Herr Koch [8] die Kennline dieser in neun Bereiche eingeteilt, die wiederum in acht Intervalle eingeteilt wurde. Das Ergebnis der Divisionen, die zur Berechnung von Klirrfaktor und OHD nötig war, wird in dieser Funktion umcodiert. Dabei repräsentiert die Stelle der höchstwertigsten 1 den Bereich, der Wert der drei darauffolgenden Bits repräsentiert das Intervall innerhalb dieser Kennline. Dadurch sind die Bereiche logarithmisch eingeteilt, die Intervalle hingegen linear. Zur Erläuterung siehe Tabelle 4.2.

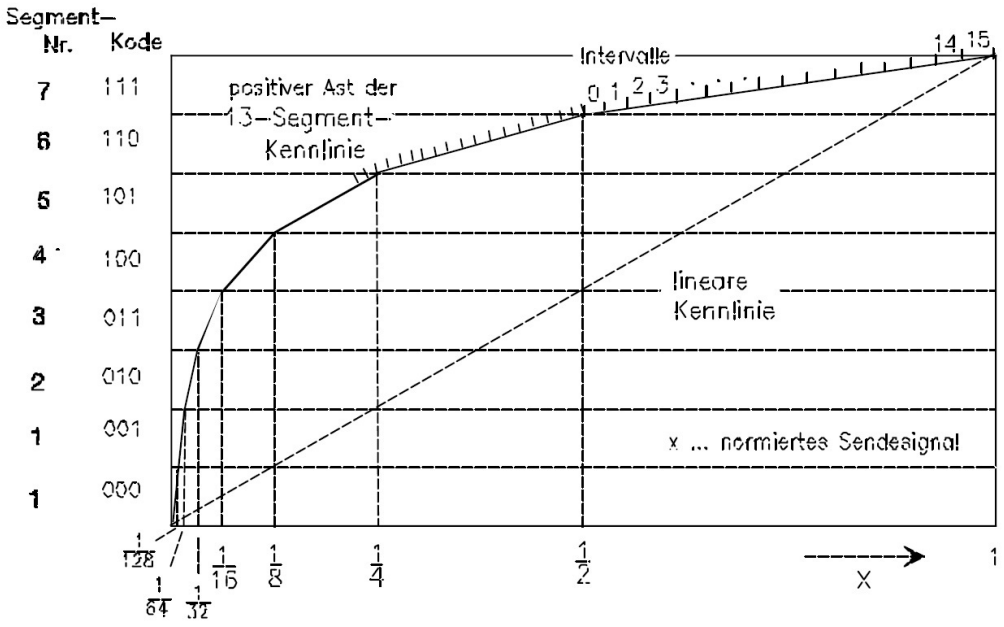


Abbildung 4.4.: Beispiel einer Einteilung der Wurzelkennline, entnommen aus [8]

Abbildung 4.4 zeigt beispielhaft eine solche Einteilung mit sieben Bereichen (hier Segmente genannt) und sechzehn Intervallen. Die LUT ist so aufgebaut, dass die Reihen den Bereichen und die Spalten den Intervallen entsprechen.

Diese Reihe und Spalte, die benötigt werden um die Lösung der Quadratwurzel aus dem LUT zu entnehmen, wird in der Funktion *decode_LUT* aus dem Divisionsergebnis ermittelt. Der Lösungsvorschlag für dieses Problem von Koch [8] sah vor, dass das Divisionsergebnis innerhalb einer Schleife mit einer Maske verglichen wird. Diese Maske besitzt eingangs eine 1 an der höchst möglichen Stelle, durch eine Und-Verknüpfung lässt sich feststellen, ob das Divisionsergebnis ebenfalls an dieser Stelle eine 1 besitzt. ist das nicht der Fall, so wird die 1 in der Maske um eine Stelle nach rechts verschoben. Die Schleife wird solange wiederholt, bis die höchstwertige 1 detektiert wurde. Das Problem dieser Lösung ist, dass unter Umständen viele Schleifendurchläufe benötigt werden, bis das Ergebnis feststeht. Dabei gilt, je kleiner das Divisionsergebnis, also je geringer der Klirrfaktor, umso mehr Schleifendurchläufe werden benötigt.

Aufgrund dessen wurde eine optimierte Lösung implementiert. Die Idee ist hierbei, dass die Divisionsergebnisse in Bereiche eingeteilt werden in denen nach der höchstwertigen 1 gesucht wird. Diese Bereiche werden mit jedem Schleifendurchlauf verkleinert, bis er nur noch aus einer Stelle besteht, an der sich die gesuchte 1 befindet.

Dafür wird eine Variable *mask* benötigt, die zunächst den Wert 128 zugewiesen bekommt. Binär ausgedrückt bedeutet dies, dass sich eine 1 an der achten Stelle befindet. Die Variable *digit* zeigt dies an, indem ihr eine 8 zugewiesen wird. Als nächstes erhält die Variable *digit_interval* den Wert 4. Es gibt innerhalb der Schleife drei Möglichkeiten.

- Das Divisionsergebnis *factor* ist kleiner als der Wert von *mask*. Das bedeutet, dass die höchstwertige 1 von *factor* auf der rechten Seite von der 1 in *mask* liegt. *mask* wird daraufhin um *digit_interval* nach rechts verschoben. *digit* wird der Wert von *digit_interval* abgezogen, um wieder die Stelle der 1 in *mask* richtig anzuzeigen.
- Das Divisionsergebnis *factor* hat seine höchstwertigste 1 an der gleichen Stelle wie die Variable *mask*. Dies wird erkannt indem überprüft wird, ob *factor* kleiner ist als *mask* um eine Stelle nach links verschoben. Ist der Wert in *factor* größer ist als der Wert in *mask*, aber kleiner ist als *mask* um eine Stelle nach links verschoben, kann sich die höchstwertige 1 in beiden Variablen nur an der gleichen Stelle befinden. Die 1 wurde also gefunden, die Reihe (Variable *row*) der LUT entspricht der Variable *digit* abzüglich 3 (Siehe hierfür Tabelle 4.2). Der Wert der drei darauffolgenden Bits, die die Spalte (Variable *col*) der LUT repräsentieren wird durch Und-Verknüpfung von *factor* mit dem der Bitinvertierung von *mask* sowie dem Schieben um den zuvor ermittelten Wert *row* -1 ermittelt. Eine Ausnahme bildet der Fall, in dem für *row* ein Wert von 0 ermittelt wurde. Dann entpricht die *col* dem Wert von *factor*. In jedem Fall sind nun

Reihe und Spalte der LUT gefunden und weitere Schleifendurchläufe werden durch ein *break* verhindert.

- Das Divisionsergebnis *factor* ist größer als der Wert von *mask*. Analog zu dem oben beschreibenen Fall, liegt nun die gesuchte 1 in *factor* auf der linken Seite der 1 in *mask*. *mask* wird also noch links verschoben um *digit_interval* und *digit* wird durch Abzug von *digit_interval* auf den neuen Stand gebracht.

Am Ende eines jeden Schleifendurchlaufs wird *digit_interval* um ein Bit nach rechts verschoben. Damit wird mit jedem Schleifendurchlauf das Intevall kleiner in dem sich die gesuchte 1 befinden kann. Dieses Verfahren stellt sicher, dass die gesuchte 1 in *mask* innerhalb von maximal vier Schleifendurchläufen gefunden wird. Ist allerdings die ermittelte Bereich größer als die Reihen der LUT, so bedeutet dies, dass das Ergebnis der Radizierung größer ist, als der maximale Wert, der in der LUT vorhanden ist. In diesem Fall wird als Ergebniss der Maximalwert der LUT entnommen. Der Quelltext:

```
void decode_LUT(void){
    int col;
    int row;
    int i;
    int digit;
    int digit_interval;
    uint32_t mask;

    for( i = 0; i<2;i++){

        mask= 0x80;
        //Zeigt Stelle an der sich die 1 in mask befindet
        digit = 8;
        //Bestimmt, um wieviel Bits die 1 in mask verschoben wird
        digit_interval = 4;
        if(THD_calc.factor > 3){
            do{
                //Ist Factor kleiner als Mask?
                if(THD_calc.factor < mask){
                    // Die 1 in mask wird nach rechts geschoben
                    // da sich die höchstwertige  "1" im rechten
                        Abschnitt befindet
                    mask >>= digit_interval;
                    //digit wird entsprechend angepasst
                    digit -= digit_interval;
                }
                //Ist die höchstwertige "1" an der gleichen Stelle wie in
                    mask?
```

```
            else if(THD_calc.factor < (mask << 1))
28              {
                    //Reihe im LUT entspricht der Stelle der
                       höchstwertigen "1" - 3
30                  row = digit - 3;
                    if(row)
32                      //Spalte entspricht den 3 Bits, die der
                           höchstwertigen "1" folgen
                        col = (THD_calc.factor & ~mask) >> (row-1);
34                  else
                        //Ausnahme stellt Reihe "0" dar
36                      col = THD_calc.factor;
                    break;
38              }
                else{
40                  //factor ist größer als mask, deswegen muss links von
                        der "1" in mask
                    //nach der "1" in factor gesucht werden, dazu wird
                       die "1" in mask
42                  //nach links verschoben
                    mask <<= digit_interval;
44                  //digit wird entsprechen angepasst
                    digit += digit_interval;
46              }
                //Suchintervall wird mit jedem Schleifendurchlauf kleiner
48              digit_interval >>= 1;
            }while(1);
50          //Falls das Ergebnis der Radizierung größer ist, als die LUT
               darstellen kann
            //Wird der maximale Wert aus der LUT entnommen
52          if (row > MAX_ZEILE || col > MAX_SPALTE){
                row = MAX_ZEILE;
54              col = MAX_SPALTE;
            }
56          // Ergebnis der Radizierung wird aus der LUT entnommen
            if (i == 0)
58              g_calc.hd =  sqrt_LUT[row][col];
            else
60              g_calc.ohd = sqrt_LUT[row][col];
        }
62      else{
            if (i == 0)
64              g_calc.hd = 0;
            else
```

```
66                      g_calc.ohd = 0;
            }
68          THD_calc.factor = THD_calc.prod;
        }
70  }
```

4.4. DSC

Beim DSC handelt es sich um einen Zähler (*discard_counter*), der beim Verwerfen eines Abtastwertes hochgezählt wird und nach Abschluss der Verarbeitung eines Abtastintervalls zurückgesetzt wird.

5. Funktionsnachweise

5.1. Abtastung

Für den Funktionsnachweis der Abtastung wurde ein sinusförmiges Signal bei unterschiedlichen Frequenzen mit Hilfe des *Tektronix AFG3022B* Funktionsgenerators eingespeist. Abbildung 5.1 zeigt beispielhaft einen Sinus mit eine Frequenz von 100Hz. Die 64 Abtastwerte umfassen zwei Perioden des eingespeisten Sinussignals. Im Anhang A.1.1 befinden sich Abbildungen die die Abtastfolge des Sinussignals bei den Frequenzen 10, 250, 750, 1500 und 2500 Hz zeigen.

Des Weiteren werden Sensorsignale mit Frequenzverdopplung bei unterschiedlichen Frequenzen eingespeist um zu verifizieren, dass der in Abschnitt 4.1 erläuterte und in Bild 4.1 dargestellte Effekt, bei dem die Abtastwerte in einer falschen Reihenfolge genommen werden, nicht auftritt. Auch hier ist beispielhaft die Abtastfolge des Signals bei 100Hz abgebildet (Abbildung 5.2). Da dieser Effekt abhängig von der Dauer der Sampleverarbeitung und der Frequenz des Signals ist, befinden sich dazu im Angang ebenfalls die Abtastfolgen bei den Frequenzen 10, 250, 750, 1500 und 2500 Hz A.1.1.

Die unterschiedlichen Phasen der Abtastungfolgen bei verschiedenen Frequenzen rührt laut Herrn Jegenhorst daher, dass die Nulldurchgangserkennung konstant ca. $50\mu s$ benötigt, näheres dazu in [6] nachlesbar. Die Phasenverschiebung hat aber keinerlei auswirkungen auf das Ergebnis der Verfahren HD5 und HDI.

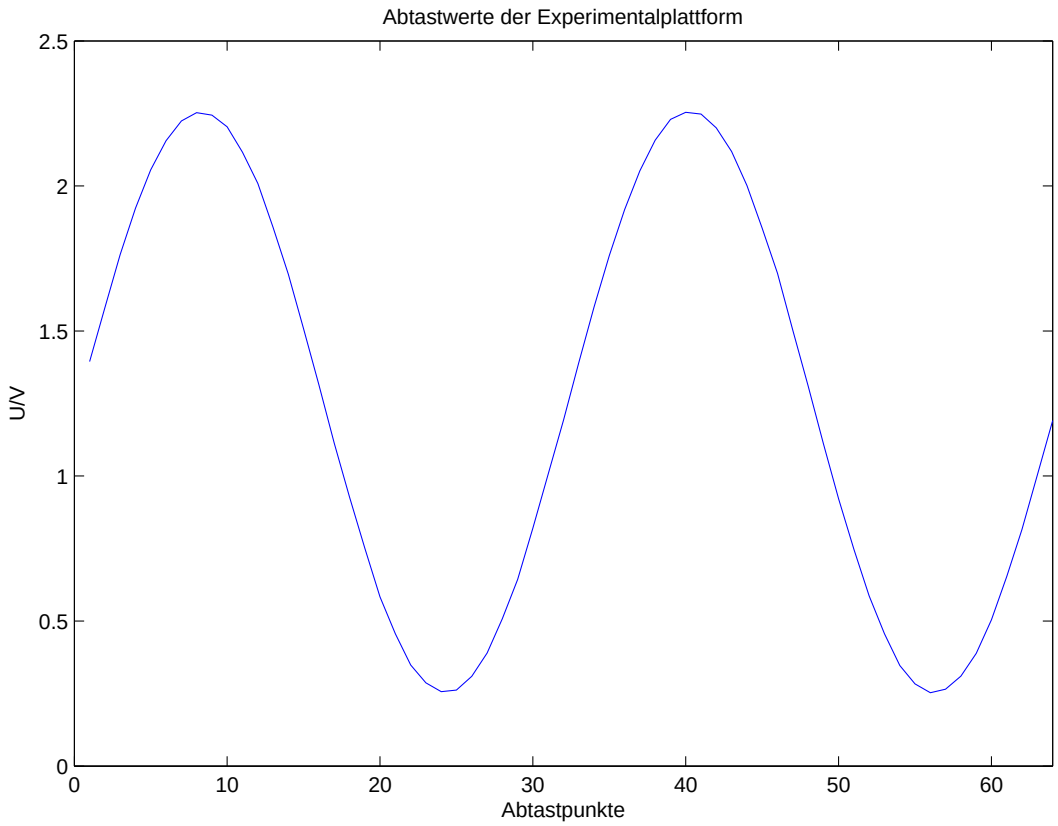


Abbildung 5.1.: Abgetastetes Sinus-Signal bei 100Hz

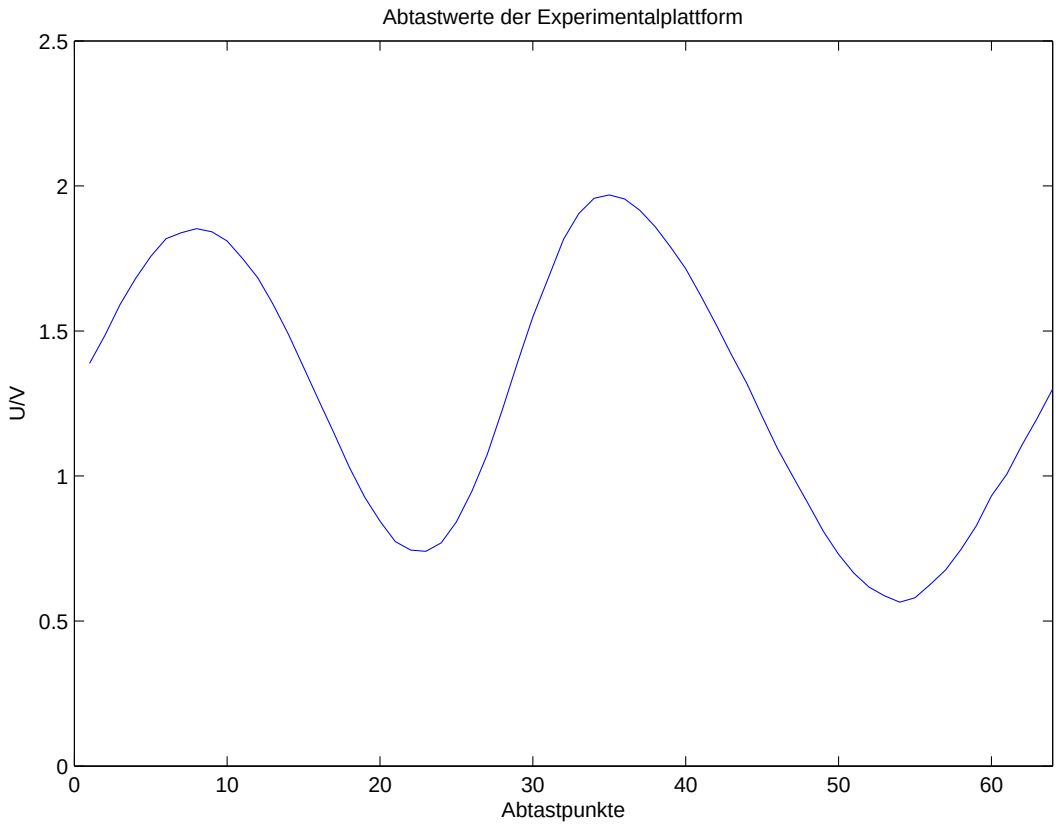


Abbildung 5.2.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 100Hz

Methode	Einzelsampleverarbeitung	Schlussberechnung
HD5	$1160\mu s$	$3967\mu s$
HDI	$351\mu s$	$2700\mu s$

Tabelle 5.1.: Laufzeitmessungen

5.2. Sensordiagnosedunktionen

5.2.1. Laufzeiten von HD5 und HDI

Ein Vorteil der HDI-Methode stellt die Schnelligkeit des Algorithmus dar. Wie in Abschnitt 4.3 besprochen, werden nur die ersten drei Harmonischen sowie die Gesamtwechselleistung des Signals berechnet. Im Gegensatz dazu berechnet die HD5-Methode insgesamt zehn Harmonische, das stellt einen höheren Rechenaufwand dar, der sich auf die Laufzeit der Berechnungen auswirkt. Messungen haben ergeben, dass die HDI-Methode für die Verarbeitung jedes einzelnen Abtastwertes lediglich 30% der Zeit, die die HD5-Methode benötigt. Für die letztendliche Berechnung von Klirrfaktor und OHD sind es 68% (Siehe Tabelle5.1). Die Oszilloskopbilder der Messung befinden sich im Anhang A.1.2.

Signalform	Klirrfaktor in %			
	HD5	HDI	Theorie	
			HD5	HDI
Sinus	0	0	0	0
Dreieck	11...12	11...13	11,72	12,03
Rechteck	36	35	36,23	43,52
Arb1	32	32	32,46	32,48
Arb2	6	35	94.96	94.97

Tabelle 5.2.: Gegenüberstellung der Ergebnisse der Klirrfaktor- und OHD-Berechnung, siehe für Signalform Arb1 Abbildung 5.3 und für Arb2 Abbildung 5.4

5.2.2. Klirrfaktor

Um zu überprüfen, ob die implementierten Methoden zur Klirrfaktorberechung korrekte Ergebnisse liefern, wurden anstatt des Sensorsignals einige markante Signale mit Hilfe des Funktionsgenerators in die Experimentalplattform eingespeist.

In der Tabelle 5.2 sind die Ergebnisse der beiden Methoden zur Klirrfaktorberechnung dem theoretischen Klirrfaktor gegenübergestellt. Dass es für den HD5 und den HDI unterschiedliche theoretische Ergebnisse gibt, ist damit zu begründen, dass die HD5-Methode nur fünf Harmonische der Zahnfrequenz, der HDI dagegen mittels der Gesamtwechselleistung sämtliche Harmonischen mit einbezieht. Außerdem ist zu beachten, dass die HDI-Methode maximal einen Klirrfaktor von 35 berechnet, auch bei Signalen, die eigentlich einen größeren Klirrfaktor besitzen. Der Grund dafür ist, das ddie LUT keine größeren Werte beinhaltet.

Auffällig ist der Unterschied zwischen HD5 und HDI-Methode bei Arbiträr Signal 2. Dieses Signal stellt das Sensorsignal über eine Encoderradinkrementierung dar, bei Auftreten der Frequenzverdopplung. Die Gründe für diese unterschiedlichen Ergebnisse wurden in Abschnitt 3.2.3 besprochen.

Um eine Frequenzabhängkeit der Ergebnisse der Klirrfaktorberechnung zu überprüfen wurde ein Dreiecksignal eingespeist, dessen Frequenz in 10Hz-Schritten von 10Hz bis 2,5KHz erhöht wurde. Diese Messreihe wurde sowohl für die HDI als auch die HD5-Methode durchgeführt und ist in Abbildung 5.5 abgebildet. Es ist zu erkennen, dass der Klirrfaktor, ermittelt durch die HD5-Methode, frequenzunabhängig zwischen 11 und 12% schwankt. Die HDI-Methode berechnet als Klirrfaktor einen Wert zwischen 11 und 13%, schwankt aber weitaus weniger als es der Klirrfaktor ermittelt durch das HD5-Verfahren tut und verweilt meist bei einem Wert von 12%.

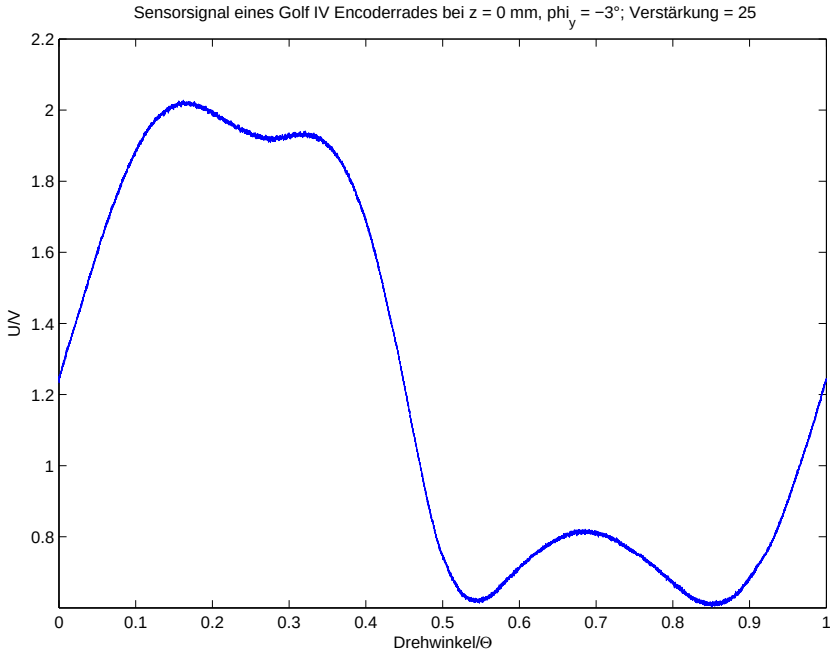


Abbildung 5.3.: Arb1: Periode eines Sensorsignals mit einem Klirrfaktor von 32,5%, reproduziert mit Hilfe eines Funktionsgenerator s

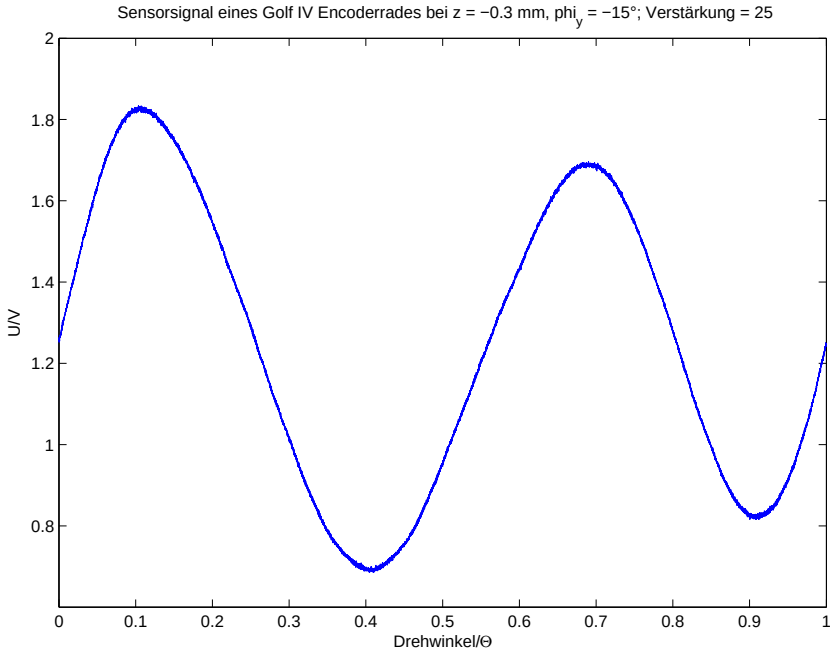


Abbildung 5.4.: Arb2: Periode eines Sensorsignals mit einem Klirrfaktor von 95%, reproduziert mit Hilfe eines Funktionsgenerators

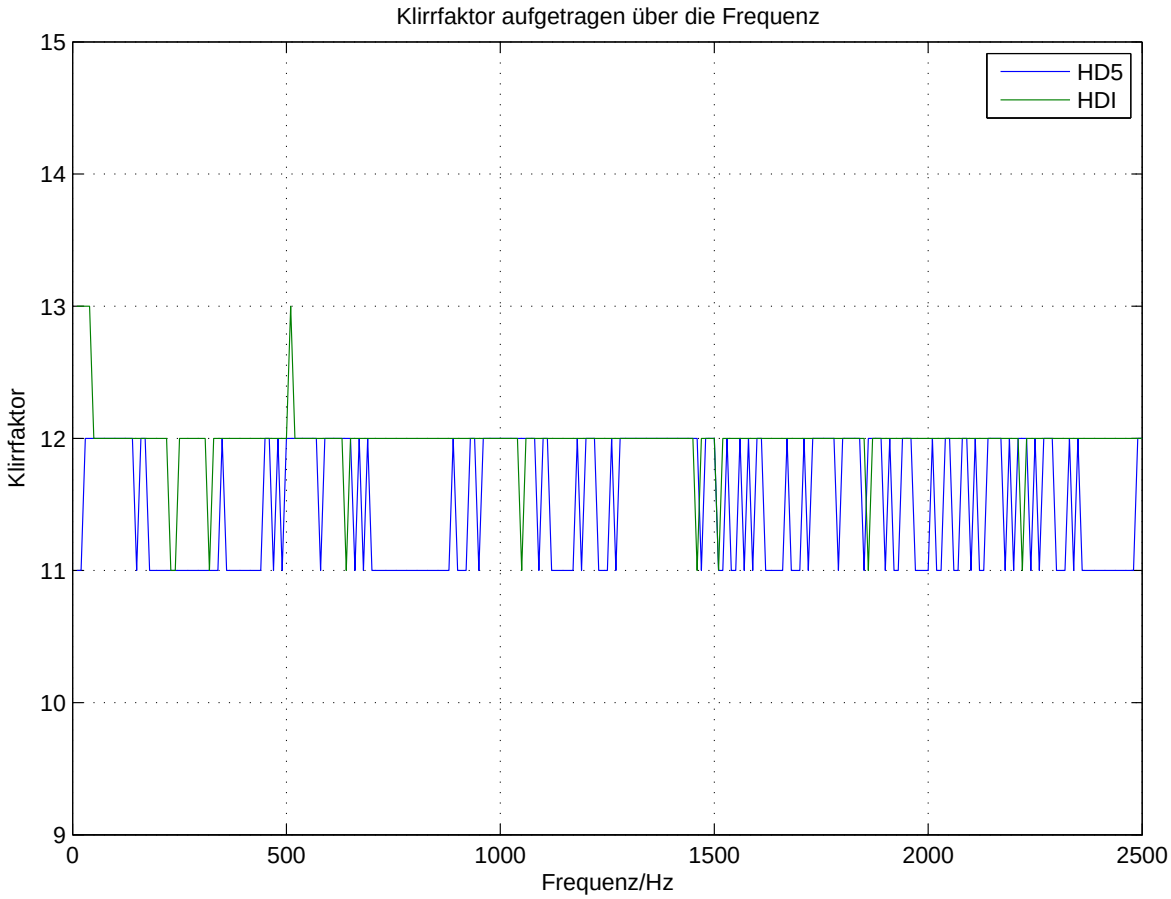


Abbildung 5.5.: Klirrfaktor eines dreieckförmigen Signals ermittelt mit HD5 und HDI

5.3. Frequenzverdopplungserkennung - OHD

Um die Frequenzverdopplungserkennung auf ihre Funktion zu testen, wurden Messungen mit dem RMP3 mit original Encoderrädern aus der Automobilindustrie vorgenommen [5]. Durch Tests ist bekannt, dass die Frequenzverdopplung bei passiven Encoderrädern, bei einer Verkippung der ϕ_y-Achse des Sensors bei einer nahen Positionierung vor dem Encoderrad in z-Richtung auftritt. Zur Verdeutlichung dieser Positionierung des Sensors ist diese in Abbildung 5.6 skizziert. Für die im Folgenden beispielhaft dargestellten Messungen wurde ein Golf IV - Encoderrad bei einer Zahnfrequenz von 1512 Hz verwendet. Die Messergebnisse werden in einer besonderen Darstellungsweise visualisiert (Siehe Abbildung 5.7). Jedes Feld stellt einen Messpunkt dar. Dabei ist die Entfernung auf einer Linearachse vor dem Sensor auf der Ordinate aufgetragen, der Winkel der Verkippung auf einer Verkippungsachse ist aufgetragen auf der Abszisse. An der Spitze der Grafik befindet sich ein gedachtes Encoderrad. In die Felder ist ein Messwert, in diesem Fall der OHD, farblich hineincodiert. Die weiß angekreuzten Bereiche stellen die Messpunkte dar, an denen die Experimentalplattform eine erhöhte Abweichung der Zahnfrequenz von mehr als 6% des Medians der Zahnfrequenz aller Messpunkte der Messreihe gemessen hat. Die Felder die weiß dargestellt sind, sind Messpositionen, die aufgrund der Geometrie von Encoderrad und AMR-Sensor nicht anfahrbar sind, da es sonst zur Kollision kommen würde.

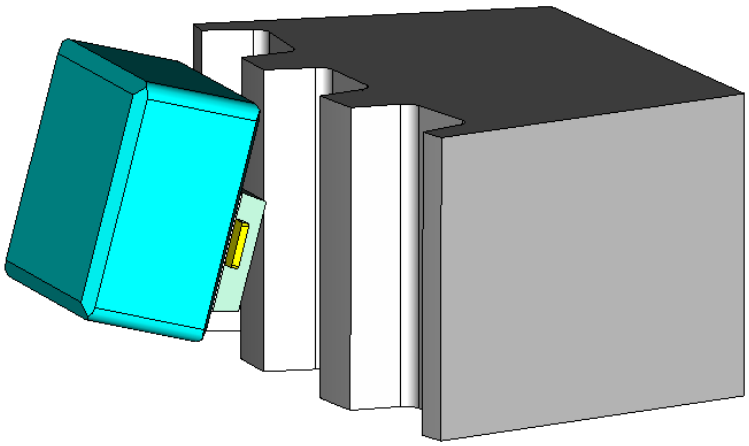


Abbildung 5.6.: Sensor über die ϕ_y-Achse verkippt

5.3.1. OHD ermittelt durch das HD5-Verfahren

Die Abbildung 5.7 ist zu erkennen, dass die Messpunkte an denen eine erhöhte Zahnfrequenz ermittelt wurde sich klar abheben von den übrigen. Eine Ausnahme bildet ein Messpunkt der hier hellblau eingefärbt ist bei einem Abstand von 0,1mm vor dem Encoderrad, sowie ein in dunklerem Blau eingefärbter Messpunkt bei 0,4mm Abstand. Dies sind Bereiche in denen es zu dem in Abschnitt 2.4 beschriebenen Problem kommt, dass das Sensorsignal zum Teil während einer Encoderradumdrehung zusätzliche Nulldurchgänge besitzt, in anderen aber nicht. Diese Bereiche lassen sich offensichtlich mit dieser Methode nicht zuverlässig bestimmen.

Die gleiche Messreihe ist nochmal in Abbildung 5.8 dargestellt, allerdings über einer größere Distanz. Ab circa 3mm Distanz beginnt sich der OHD aufgrund von in Abschnitt 2.4 beschriebenen Effekten langsam zu erhöhen. Bei einer Entfernung des Sensors von 4,5 bis 5mm ist der OHD so stark erhöht, dass dieser ungefähr ein ähnliches Level erreicht wie an den Messpunkten im Nahbereich, an denen eine erhöhte Zahnfrequenz gemessen wurde.

5.3.2. OHD ermittelt durch das HDI-Verfahren

Für diesen Funktionsnachweis wurde der OHD anhand der von der Experimentalplattform aufgenommenen Abtastwerte in Matlab berechnet. Dazu wurden die gleichen Rechenschritte ausgeführt, wie auch auf der Experimentalplattform, allerdings wurde auf die Radizierung mittels LUT verzichtet. Der Quelltext dieser Matlab-Funktion befindet sich im Anhang A.2.1. Der Grund für dieses Vorgehen ist, dass eine Messreihe mit dem RMP3 bei sovielen Messpunkten über 30 Stunden dauern kann. Bei den Messreihen wurde bis jetzt immer das HD5-Verfahren genutzt. Da die Verfahren sich nur durch die unterschiedliche Auswertung der Abtastwerte unterscheiden, ist der Aufwand die gleichen Messreihen mehrfach abzufahren nicht gerechtfertigt gewesen.

Bei Anwendung des HDI-Verfahren, erhält man ein ähnliches Ergebnis für den OHD wie durch das HD5-Verfahren. Allerdings fällt bei Vergleich von Abbildung 5.9 und 5.7 auf, dass die Ergebnisse, ermittelt durch die HDI-Methode etwas geringer ausfallen. Dies entspricht der in Abschnitt 3.2.4 formulierten Erwartung. Dennoch ist der Bereich der Messpunkte mit erhöhter ermittelter Zahnfrequenz klar von den dem restlichen Bereich unterscheidbar. Ausgenommen sind die schon unter Abschnitt 5.3.1 besprochenen Ausnahmen. Bei größerer Entfernung fällt der OHD ermittelt durch die HDI-Methode ebenfalls geringer aus wie der Vergleich von Abbildung 5.10 und 5.8 zeigt.

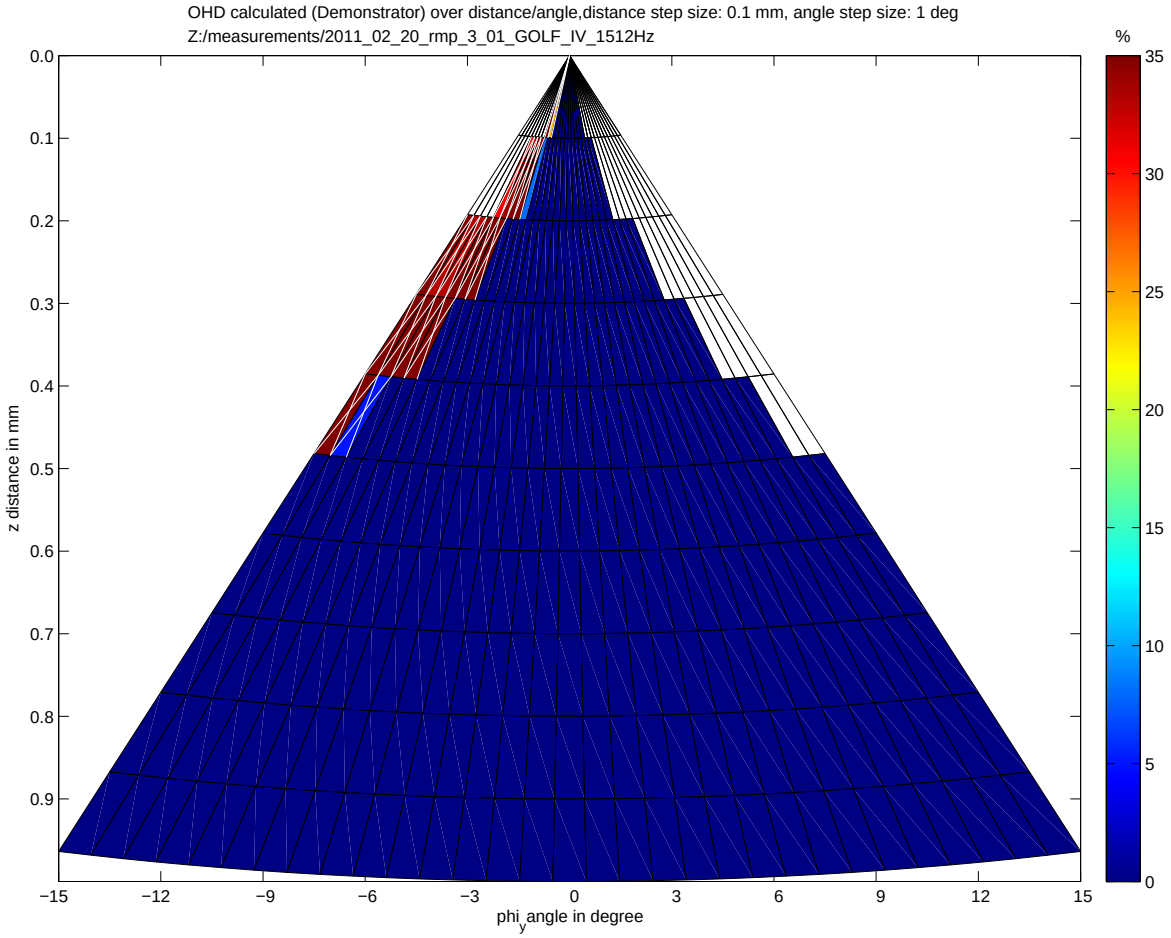


Abbildung 5.7.: OHD ermittelt mit dem HD5-Verfahren, bei Verschiebung auf der z-Achse und Verkippung über die der ϕ_y-Achse des Sensors

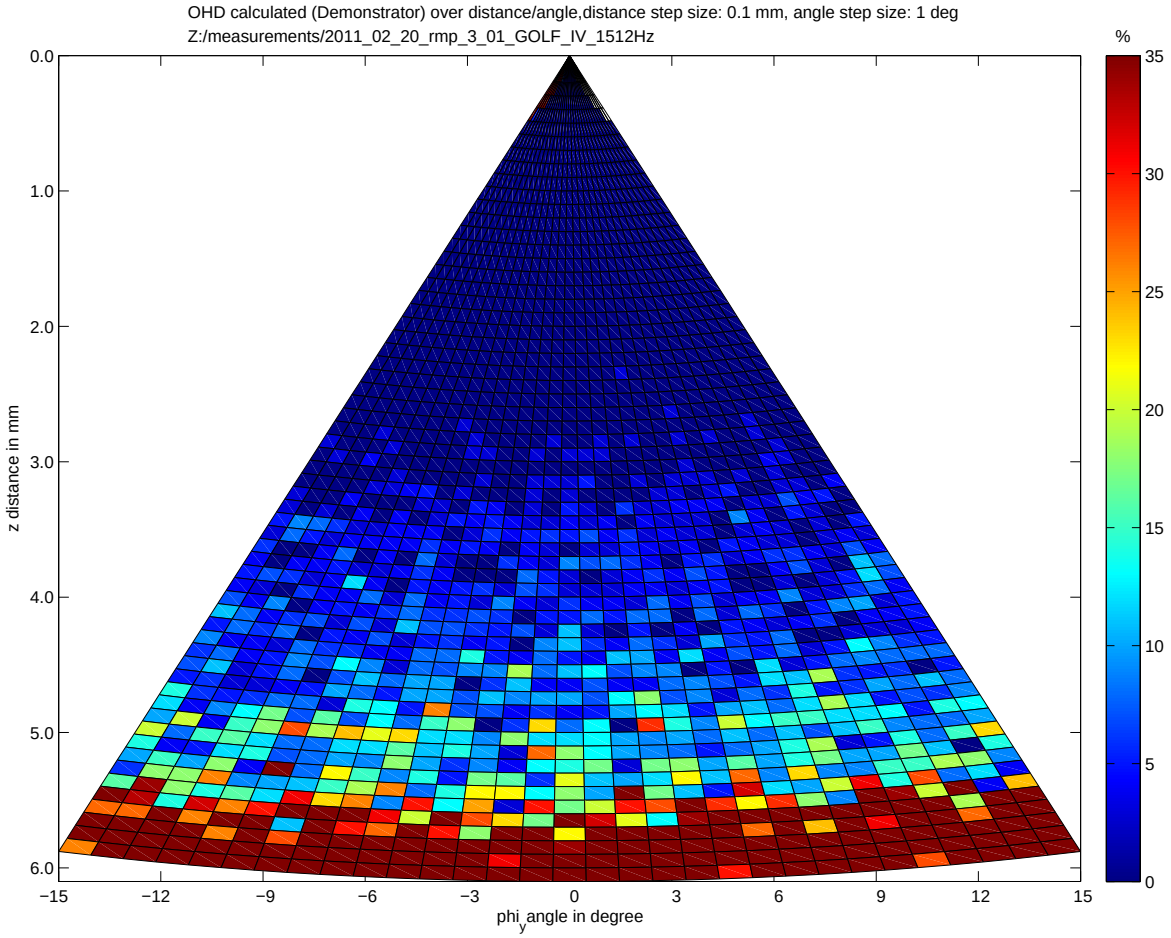


Abbildung 5.8.: OHD ermittelt mit dem HD5-Verfahren, bei Verschiebung auf der z-Achse und Verkippung der ϕ_y-Achse des Sensors

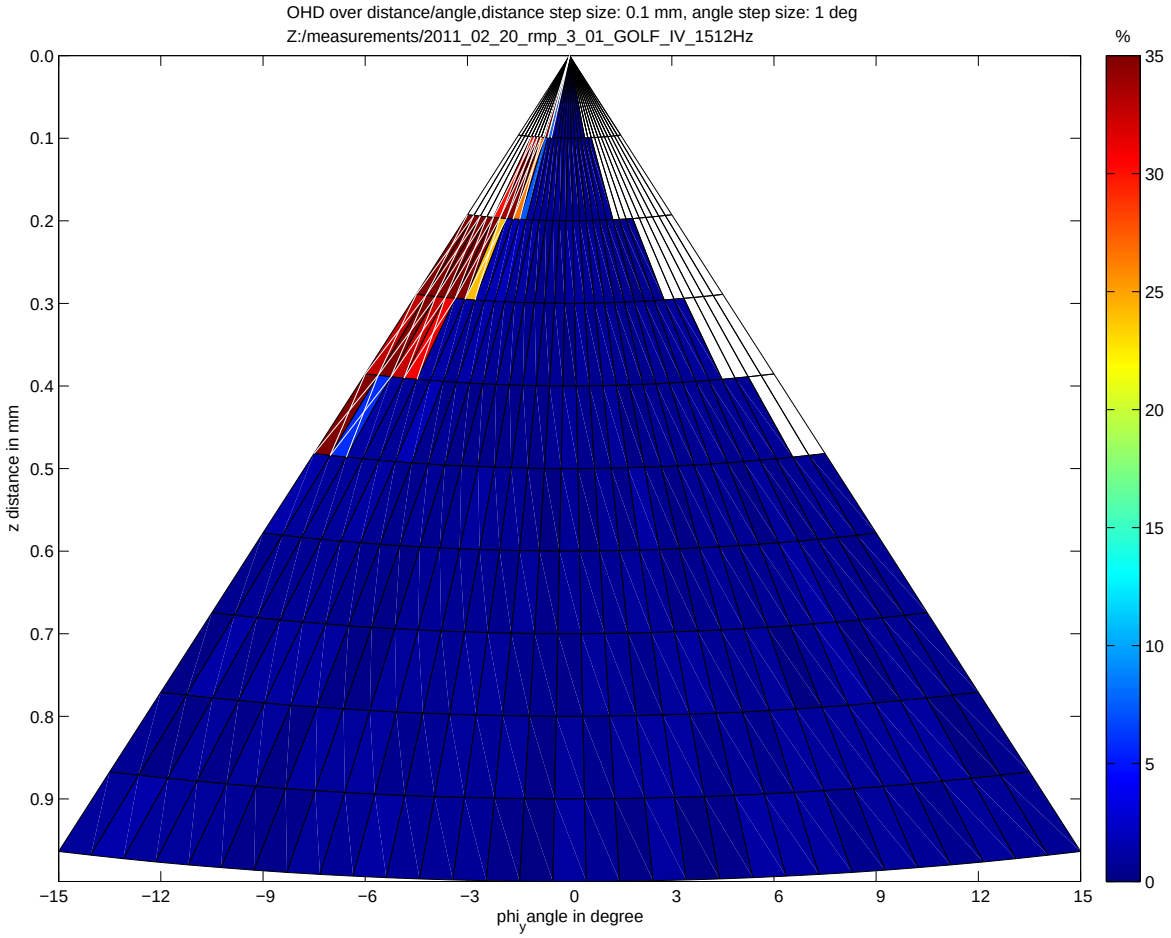


Abbildung 5.9.: OHD ermittelt mit dem HDI-Verfahren, bei Verschiebung auf der z-Achse und Verkippung der ϕ_y-Achse des Sensors

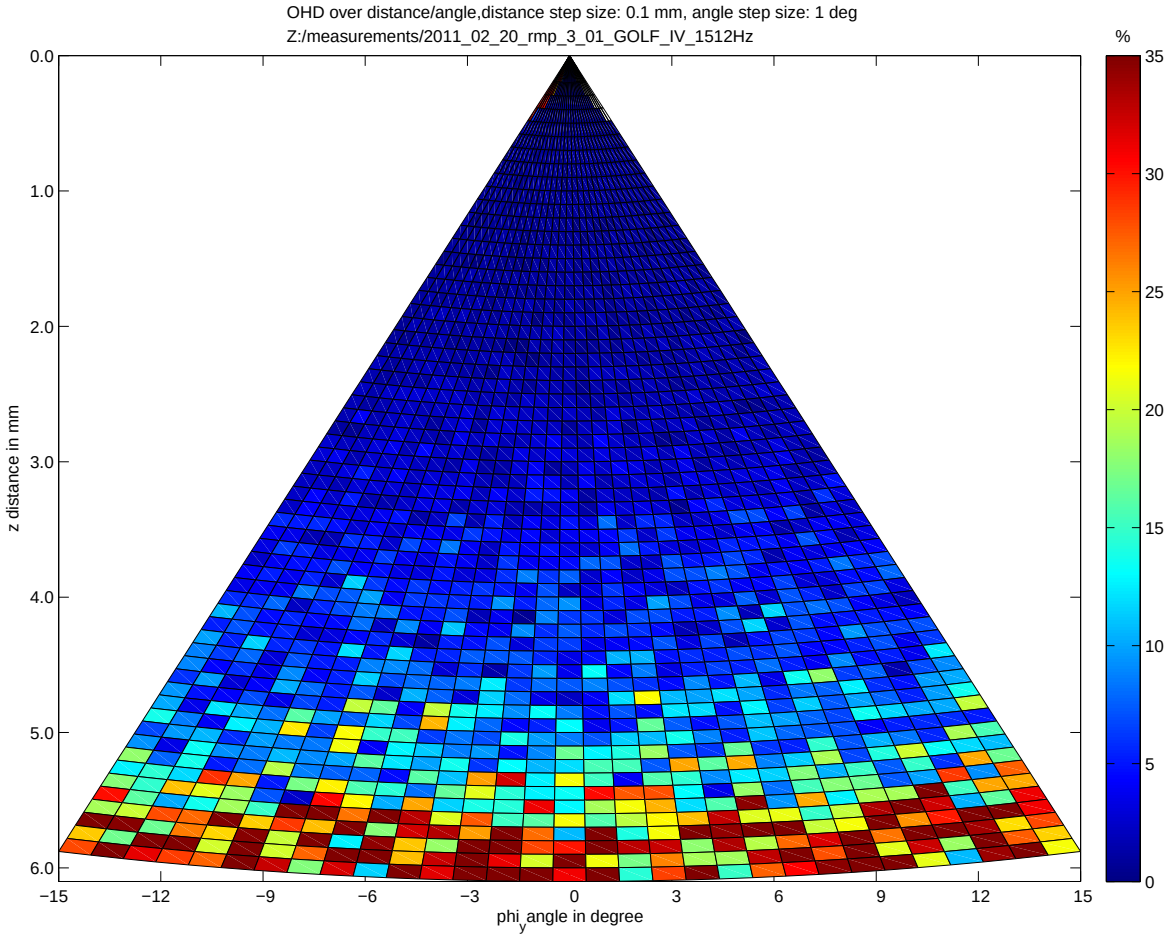


Abbildung 5.10.: OHD ermittelt mit dem HDI-Verfahren, bei Verschiebung auf der z-Achse und Verkippung der ϕ_y-Achse des Sensors

5.4. DSC

Die Folgende Messung ist durchgeführt worden, indem Sensorhalbbrückensignale, die bei einer Messreihe mit dem digitalen Speicheroszilloskop *Tektronix MSO 3034* aufgenommen wurden, mit dem Funktionsgenerator reproduziert wurden und auf die Experimentalplattform gegeben wurde. Das Messskript befindet sich im Anhang A.2.1.

In Abbildung 5.11 ist zunächst der OHD farblich codiert zu sehen. Auch in dieser Messreihe gibt es wieder im Grenzbereich zwischen Frequenzverdopplung und Normalfall einen Messpunkt, an dem eine erhöhte Frequenz gemessen wurde ohne dass der OHD erhöht ist.

Abbildung 5.12 zeigt, dass der DSC diesen Bereich sicher identifizieren kann.

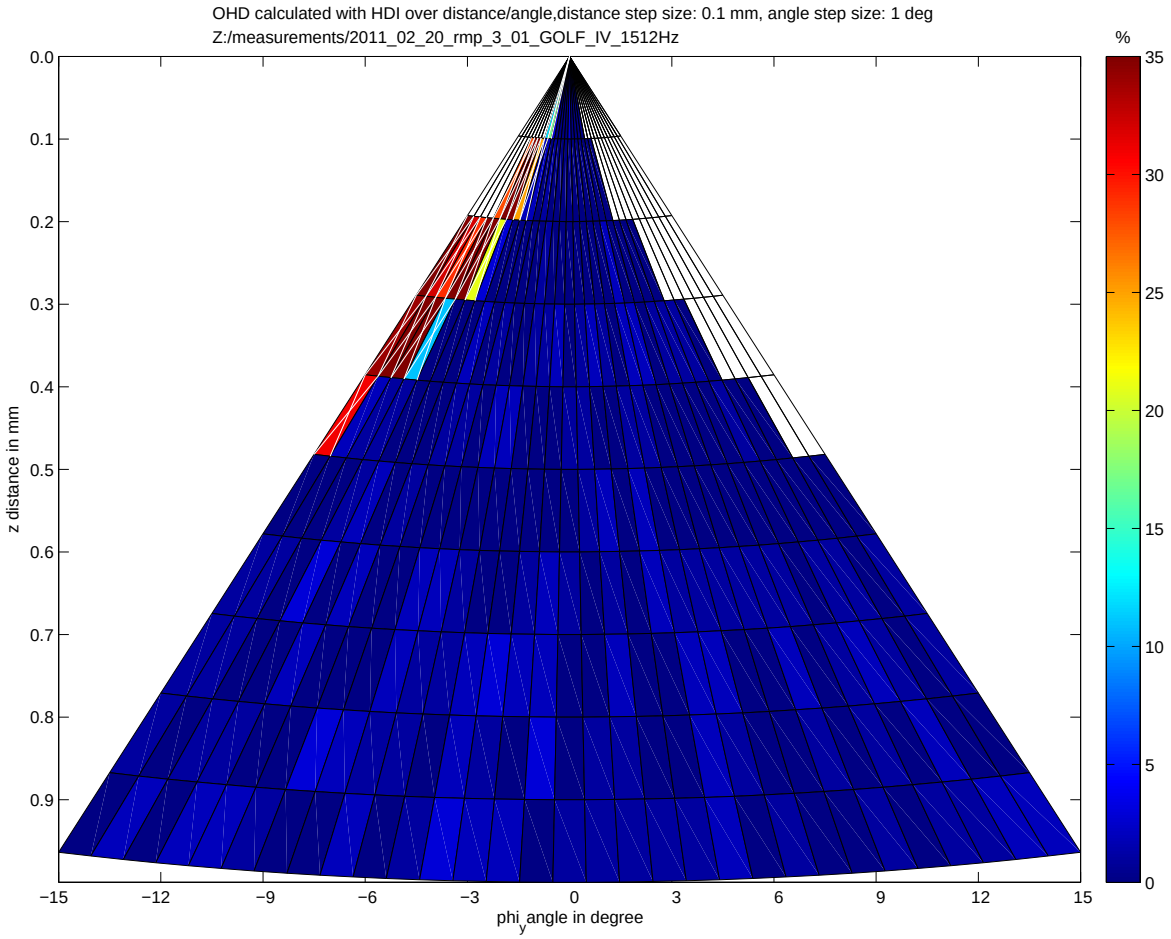


Abbildung 5.11.: OHD ermittelt mit dem HDI-Verfahren, bei Verschiebung auf der z-Achse und Verkippung der ϕ_y-Achse des Sensors

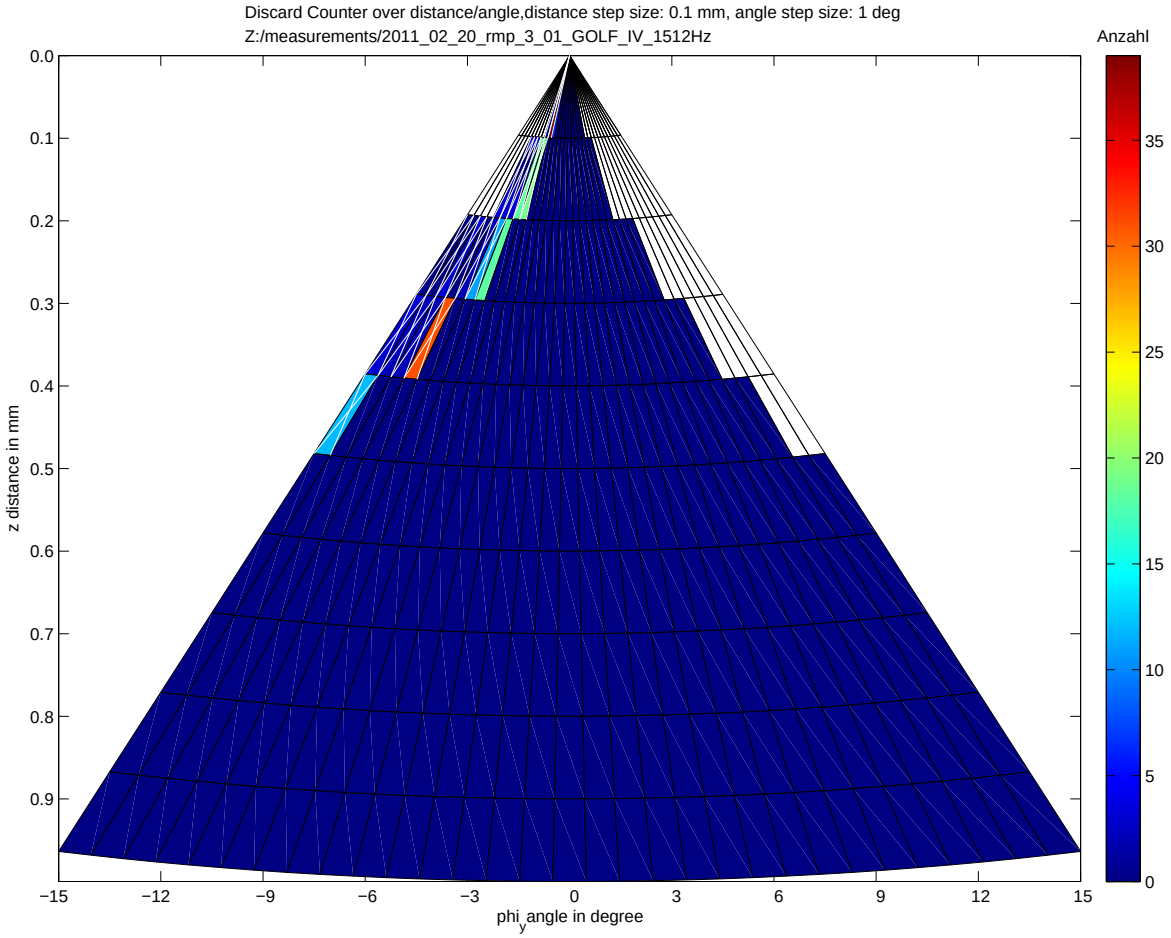


Abbildung 5.12.: DSC, bei Verschiebung auf der z-Achse und Verkippung der ϕ_y-Achse des Sensors

6. Messergebnisse

Die hier dargestellten Messergebnisse beschränken sich ausschließlich auf Messreihen, in denen eine Frequenzverdopplung vorkommt. Im Rahmen des Forschungsprojektes ESZ-ABS wurde dies in zwei Fällen beobachtet. Die folgenden Positionierungsangaben wurden in Abschnitt 2.2.2 erläutert.

- Verkippung des Sensors in negativer Richtung von ϕ_y um mehr als 8° bei einer Positionierung unter 0.5mm in z-Richtung vor dem Encoderrad, bei der Vermessung des passiven Encoderrad des Golf IV. Die Positionierung ist in Abbildung 5.6 skizziert.
- Verfahren der x-Achse in Negativrichtung bei unmittelbarer Positionerung in z-Richtung vor dem Encoderrad. Dies gilt für das passive Encoderrad des Golf IV als auch für das aktive Encoderrad des Golf V. Beispielhaft ist diese Positionierung in Abbildung 6.1 vor einem passiven Encoderrad skizziert.

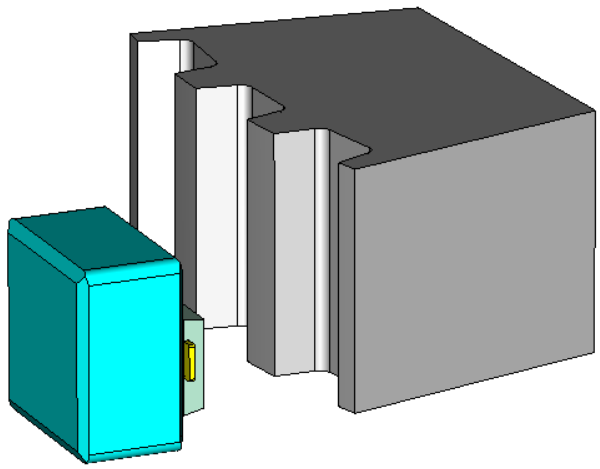


Abbildung 6.1.: Sensor verschoben auf der x-Achse

6.1. Passives Encoderrad

6.1.1. Golf IV - Encoderrad - Verfahren der z-Achse und Verkippung von ϕ_y

Im Folgenden sind Messergebnisse gelistet, die bei einer Messung an einem Golf IV Enderrad bei einer konstanten Zahnfrequenz von 1512Hz durchgeführt wurden. Dabei entfernte sich der Sensor sukzessive von 0 bis 6mm in z-Richtung und wurde von -15 bis 15° um die y-Achse verkippt. Die Darstellungsweise wurde in Abschnitt 5.3 erläutert.

Klirrfaktor

In diesem Abschnitt werden die Ergebnisse der unterschiedlichen Methoden zu Ermittlung des Klirrfaktor gegenübergestellt. In Abbildung 6.2 ist zunächst der Klirrfaktor dargestellt, der aus Sensorsignalen gewonnen wurde, die mit einem Speicheroszilloskop aufgenommen wurden über mindestens eine Encoderradumdrehung pro Messpunkt. Die dafür verwendeten Matlab-Skripte sind im Anhang A.2.1 und A.2.1 einsehbar.

Die Klirrfaktoren, ermittelt aus den mit einem Speicheroszilloskop aufgenommenen Sensorbrückensignalen, dargestellt in Abbildung 6.2, zeigen deutlich, dass die harmonischen Verzerrung nur bei sehr naher Positionierung des Sensors vor dem Encoderrad auftreten. In Abbildung 6.12 ist zu erkennen, dass das der Bereich ist, in dem die Spitze-Spitze-Spannung des Sensorbrückensignals am größten ist.

Auffällig ist, dass bei großer Entfernung die durch die HD5- und HDI-Methode ermittelten Klirrfaktoren bei großer Entfernung stark erhöht sind (Siehe Abbildungen 6.4 und 6.6). Dies steht im Gegensatz zu dem was auf Grundlage der Oszilloskopdaten berechnet wurde. Eine Erklärung für diesen Effekt wurde 2.4 gegeben. Dieses Verhalten ist aber durchaus gewollt, da eine solche Entfernung des ABS-Sensors vom Encoderrad ein Störfall darstellt, der detektiert werden sollte.

Des Weiteren fallen die durch die HDI-Methode ermittelten Klirrfaktoren, wie schon in der theoretischen Betrachtung in Abschnitt 3.2.2 erwartet und in Abschnitt 3.2.3 nochmals näher erläutert, grundsätzliche höher aus im Vergleich zu denen der HD5-Methode (siehe Abbildungen 6.4 und 6.6. Es konnte auch bestätigt werden, dass der Klirrfaktor, ermittelt durch die HDI-Methode, groß ist im Bereichen der Frequenzverdopplung (siehe Abbildung 6.7).

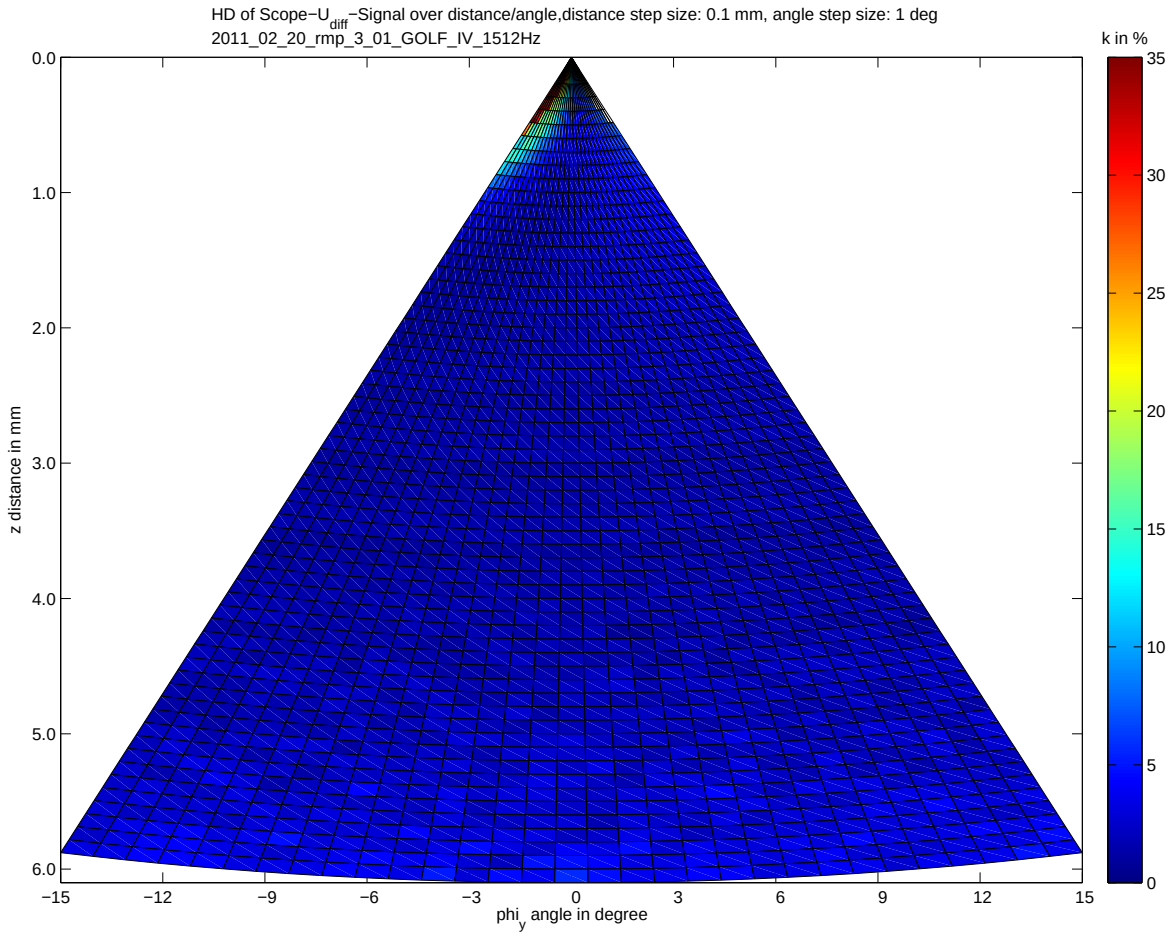


Abbildung 6.2.: Klirrfaktor ermittelt aus dem Sensordifferenzsignal, aufgenommen mit einem Speicheroszilloskop, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

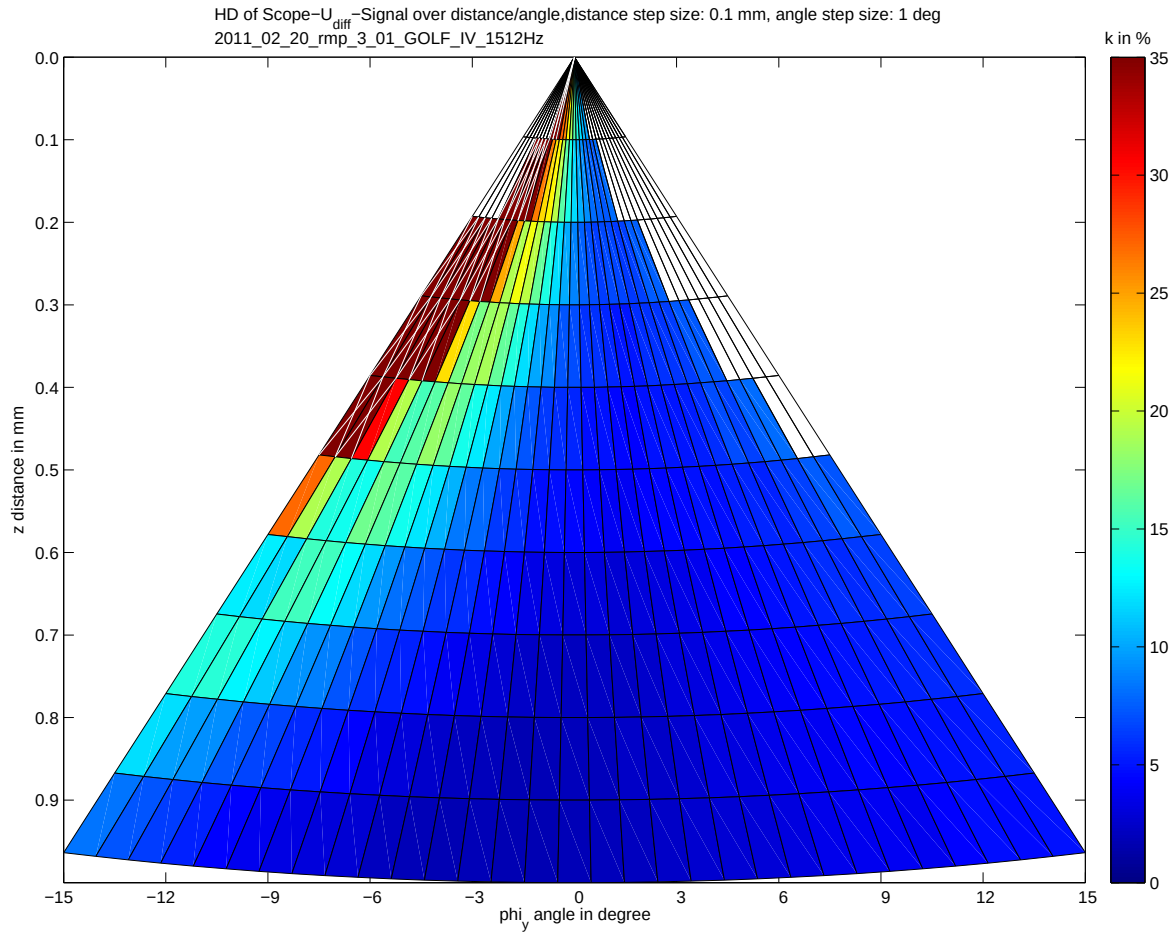


Abbildung 6.3.: Klirrfaktor ermittelt aus dem Sensordifferenzsignal, aufgenommen mit einem Speicheroszilloskop, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

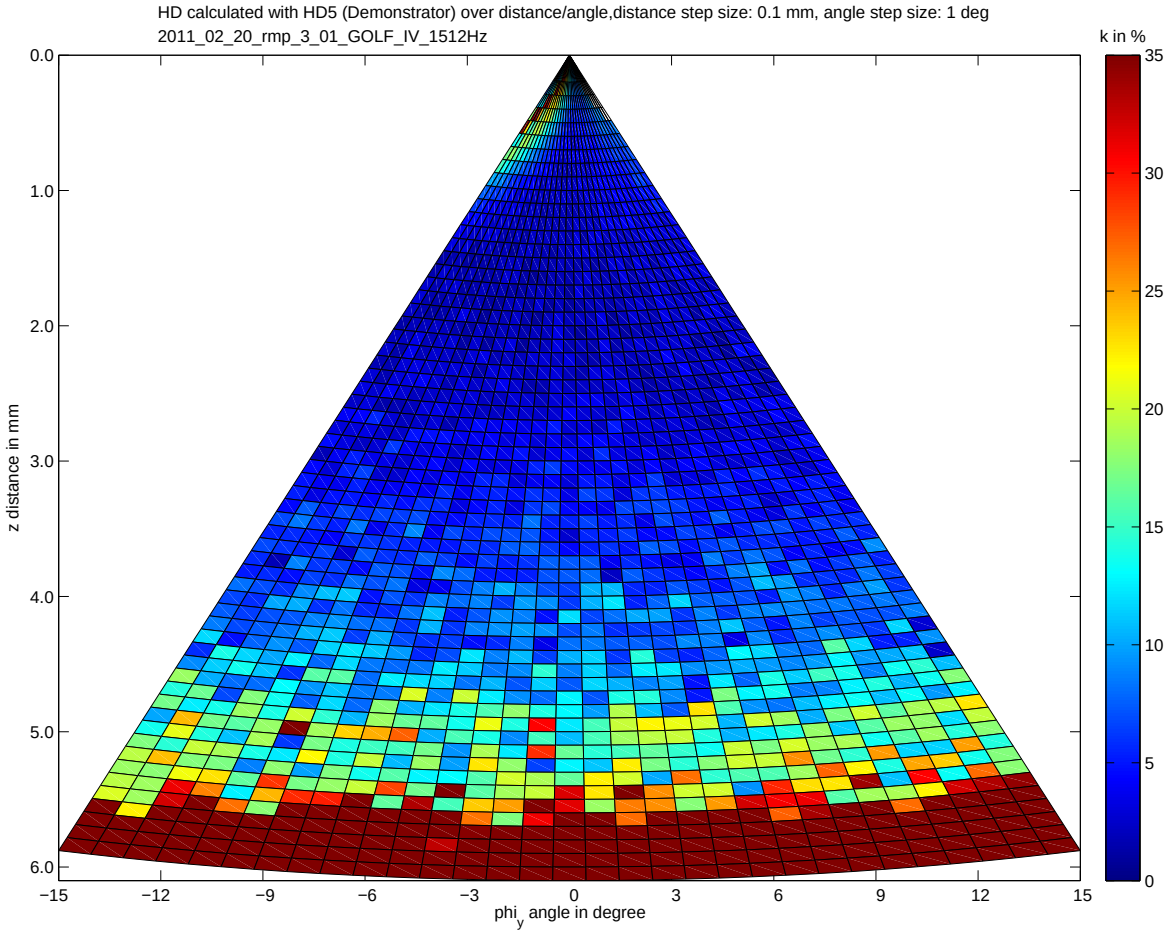


Abbildung 6.4.: Klirrfaktor ermittelt mit dem HD5-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

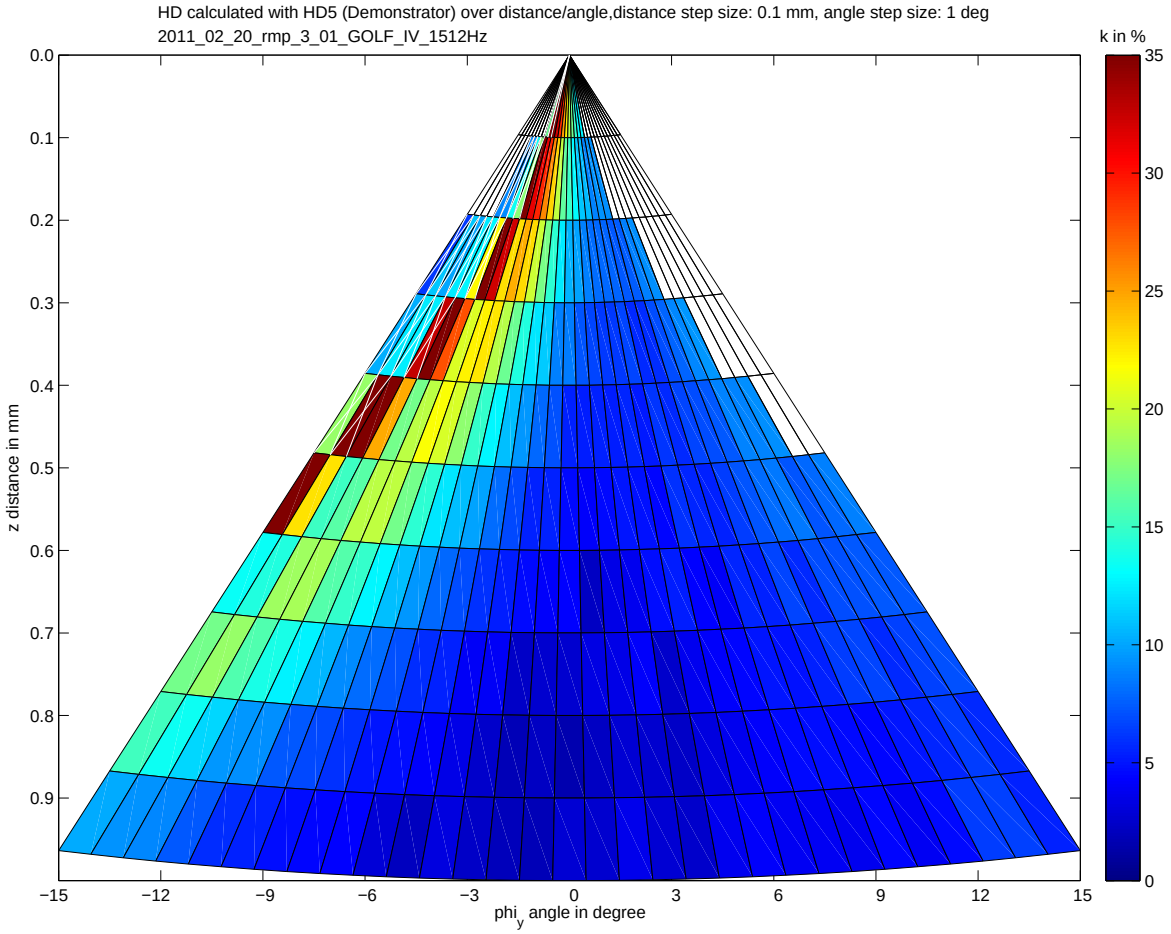


Abbildung 6.5.: Klirrfaktor ermittelt mit dem HD5-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

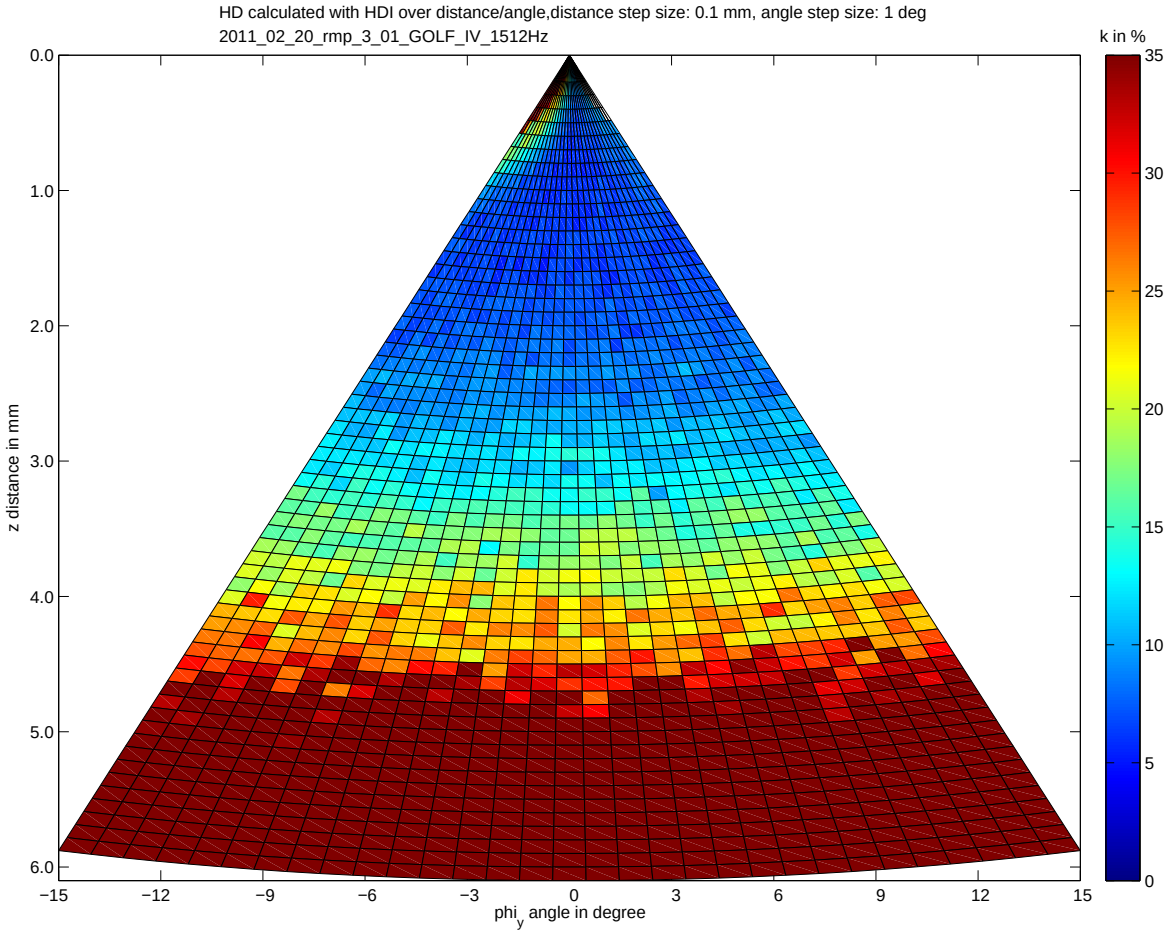


Abbildung 6.6.: Klirrfaktor ermittelt mit dem HDI-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

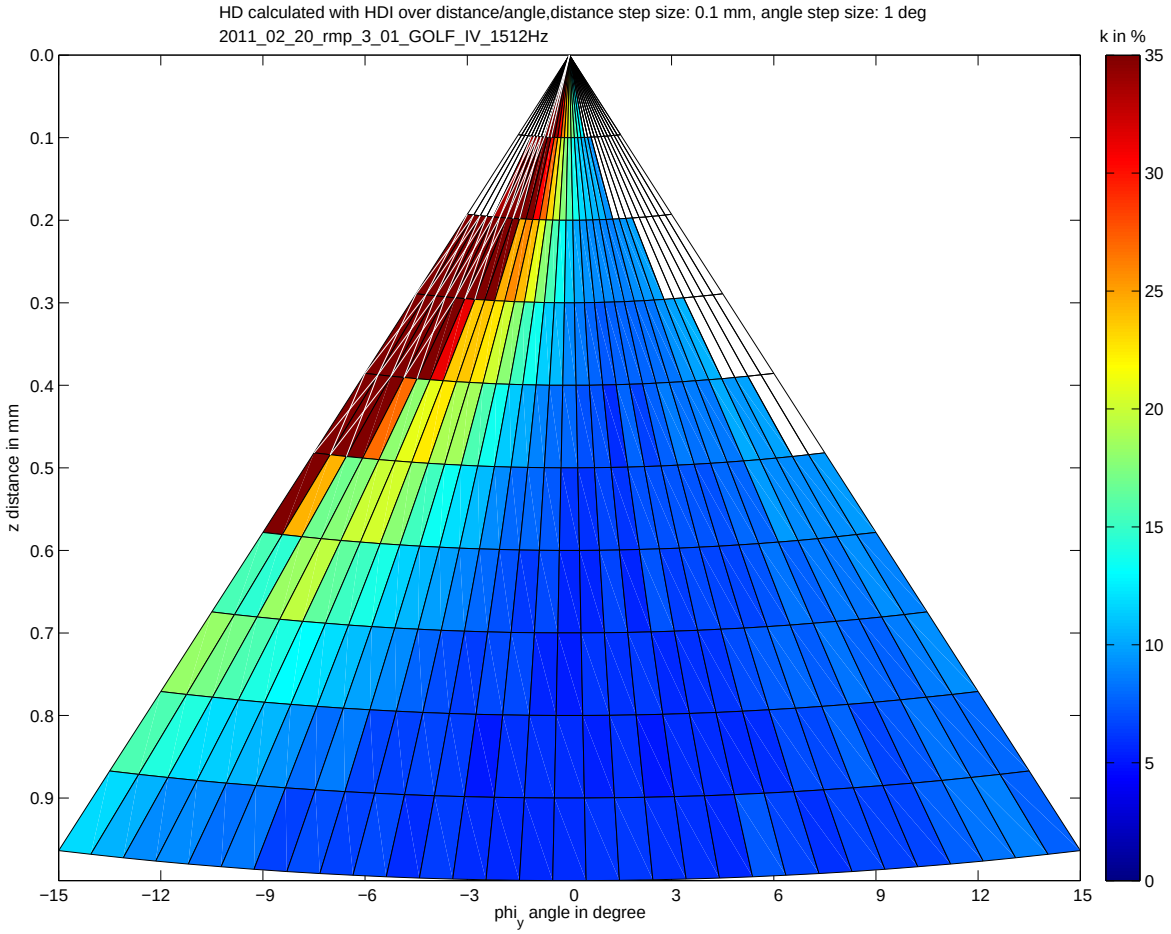


Abbildung 6.7.: Klirrfaktor ermittelt mit dem HDI-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

OHD

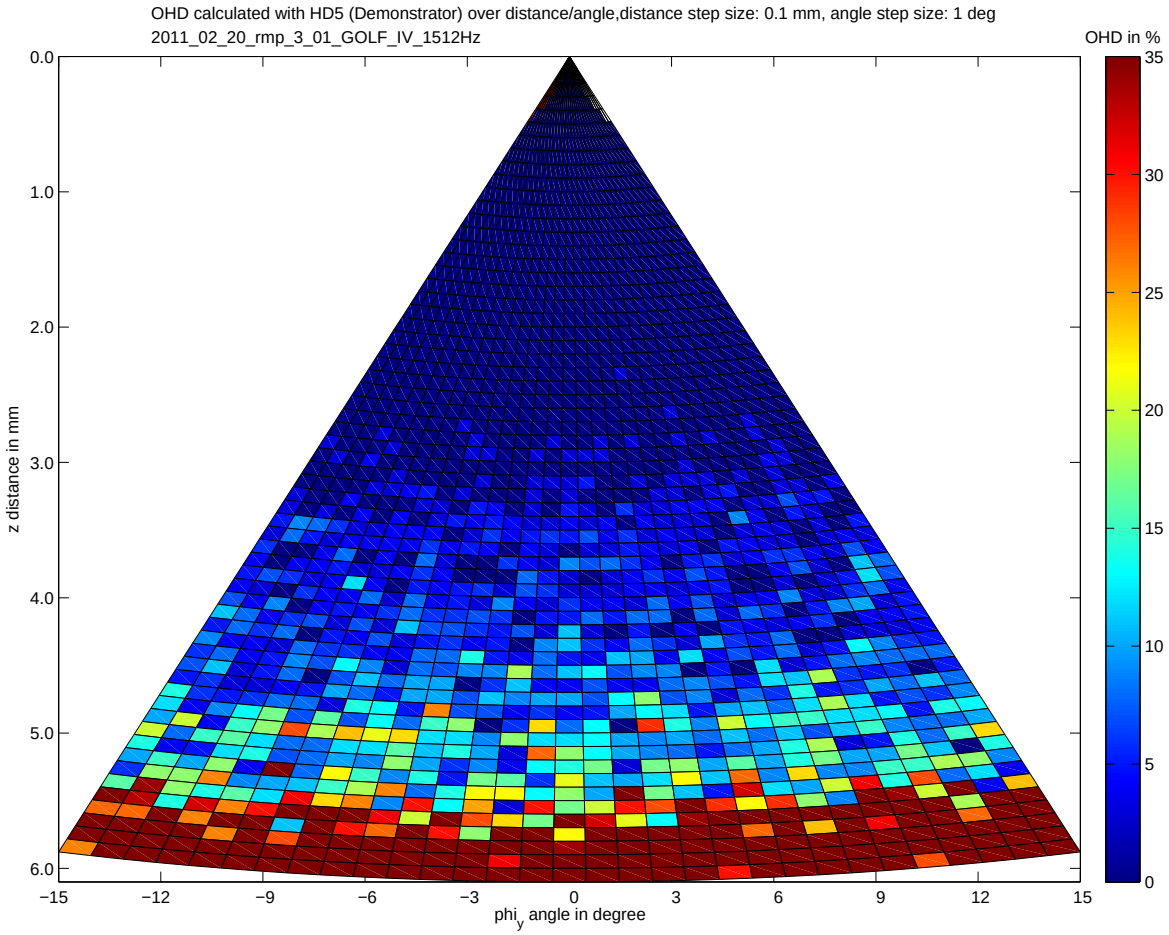


Abbildung 6.8.: OHD ermittelt mit dem HD5-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

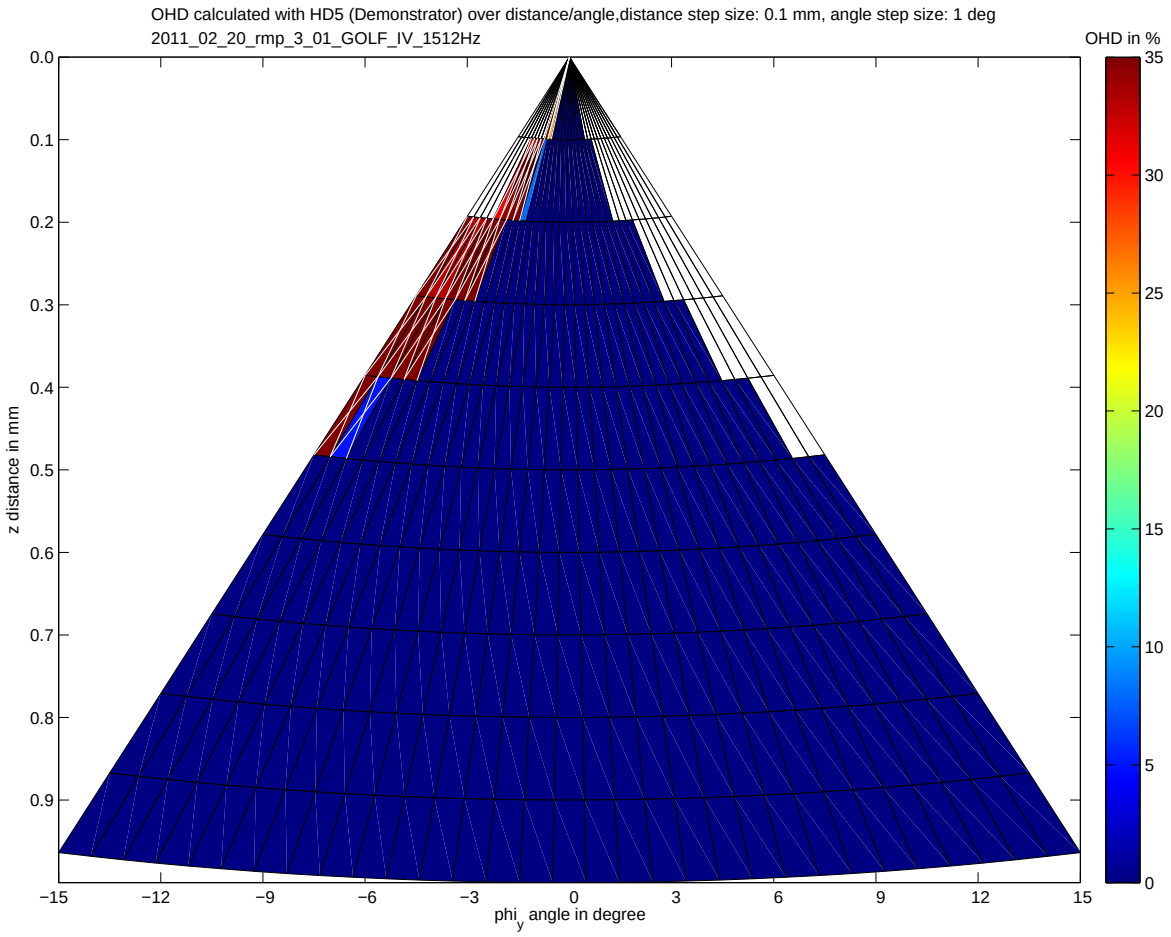


Abbildung 6.9.: OHD ermittelt mit dem HD5-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

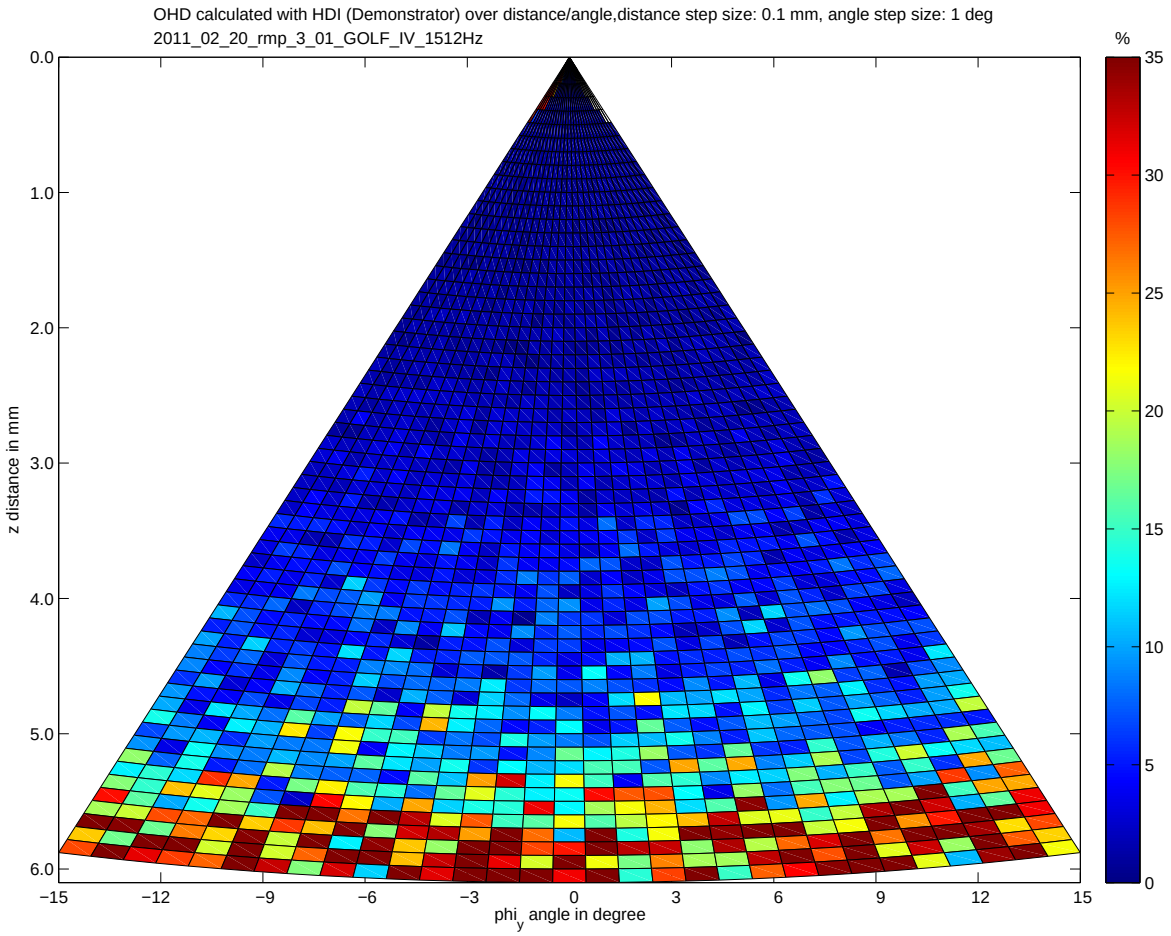


Abbildung 6.10.: OHD ermittelt mit dem HDI-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

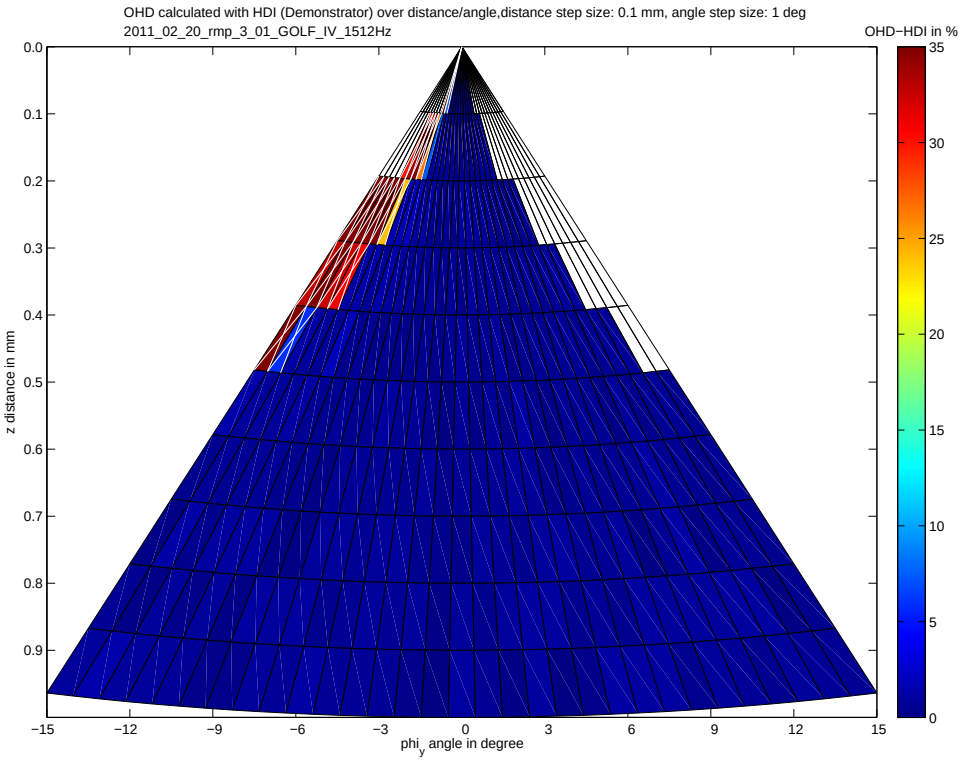


Abbildung 6.11.: OHD ermittelt mit dem HDI-Verfahren, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

Sensorbrückenspannung

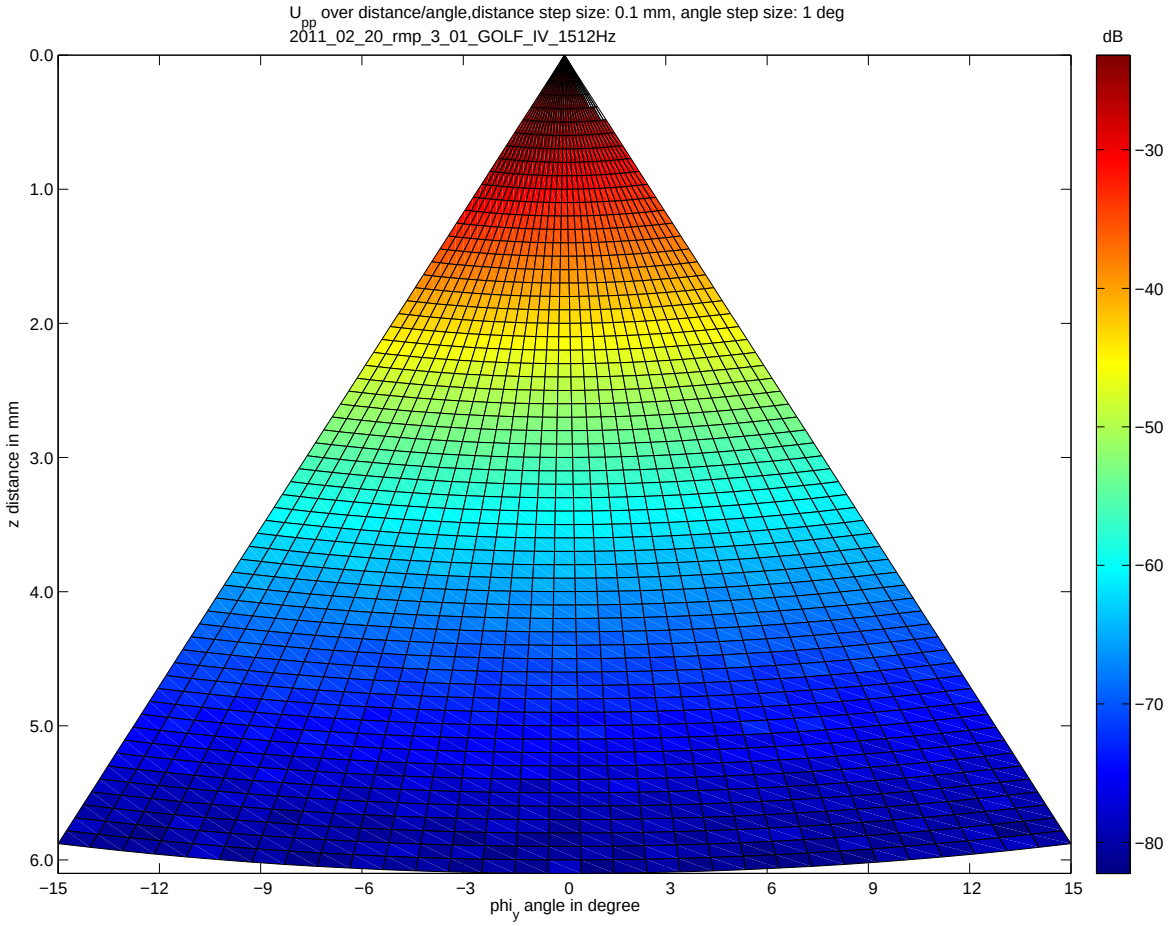


Abbildung 6.12.: Sensorbrückenspannung unverstärkt, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

Offset

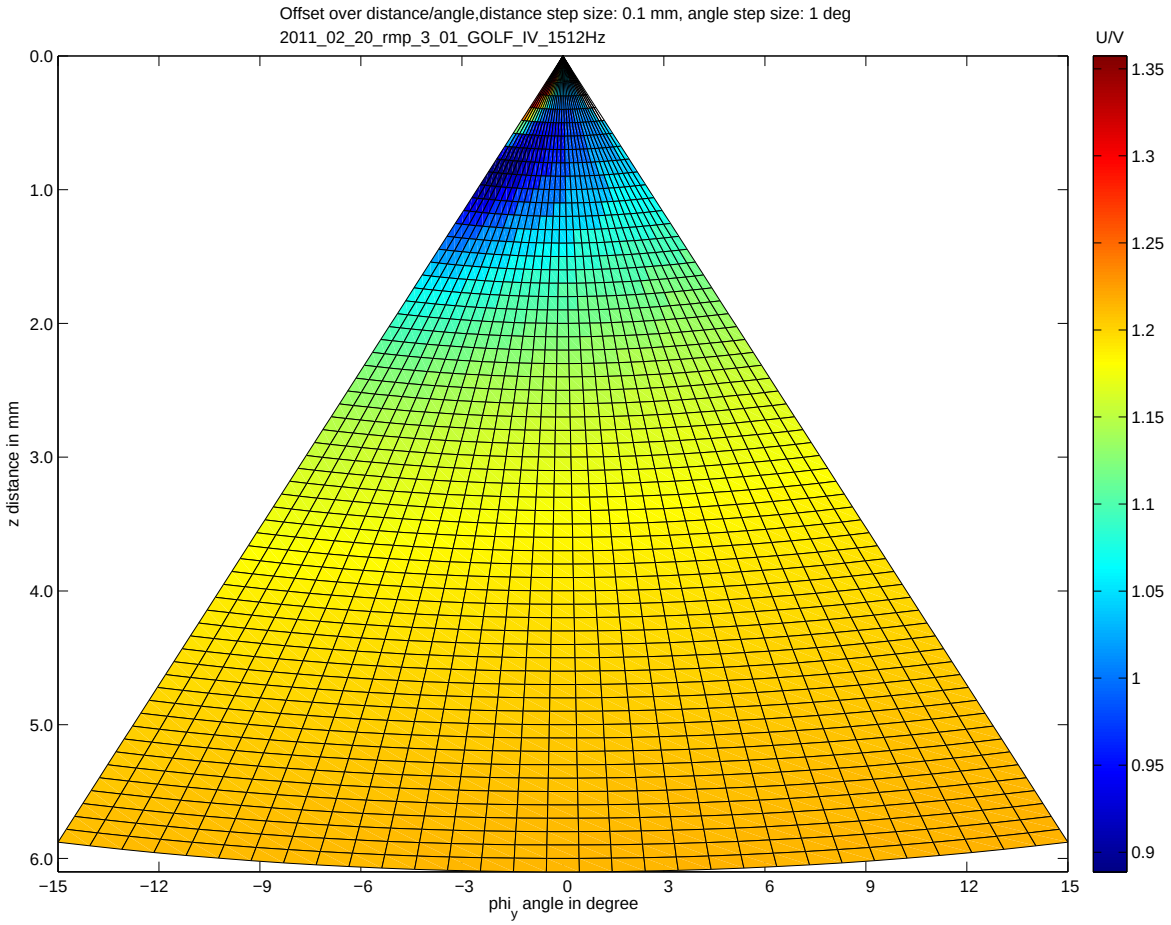


Abbildung 6.13.: Offset, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

Verstärkung

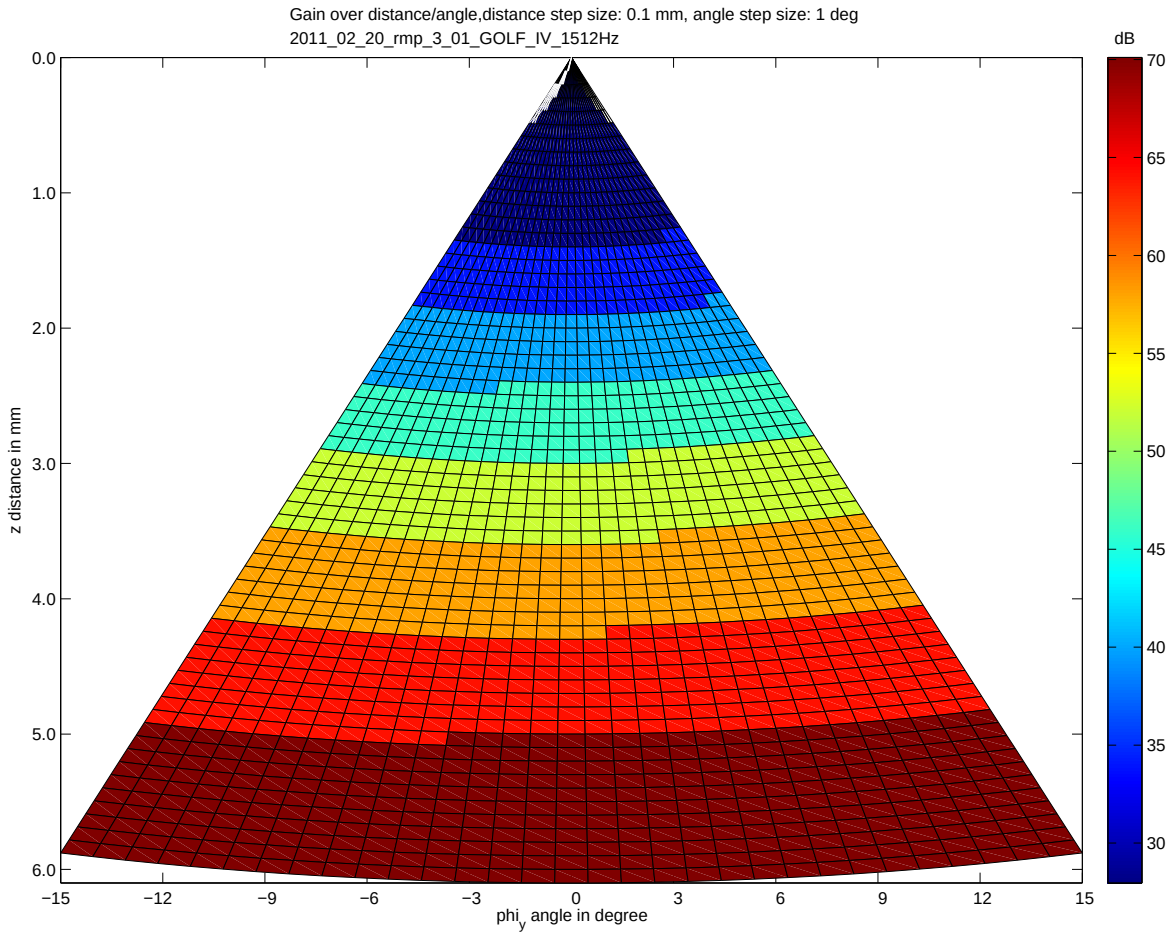


Abbildung 6.14.: Verstärkung, bei Verfahren der z-Achse und Verkippung über die y-Achse des Sensors

6.1.2. Golf IV - Encoderrad - Verfahren der z- und x-Achse

Für diese Messreihe wurden die Linearachsen z und x verfahren. Wie in Abschnitt 6 beschrieben gibt es innerhalb dieser Messreihe Messpunkte, an denen sich die Frequenz verdoppelt. Es bestätigen sich die Bemerkungen die schon im Abschnitt 6.1.1 gemacht wurden.

Klirrfaktor

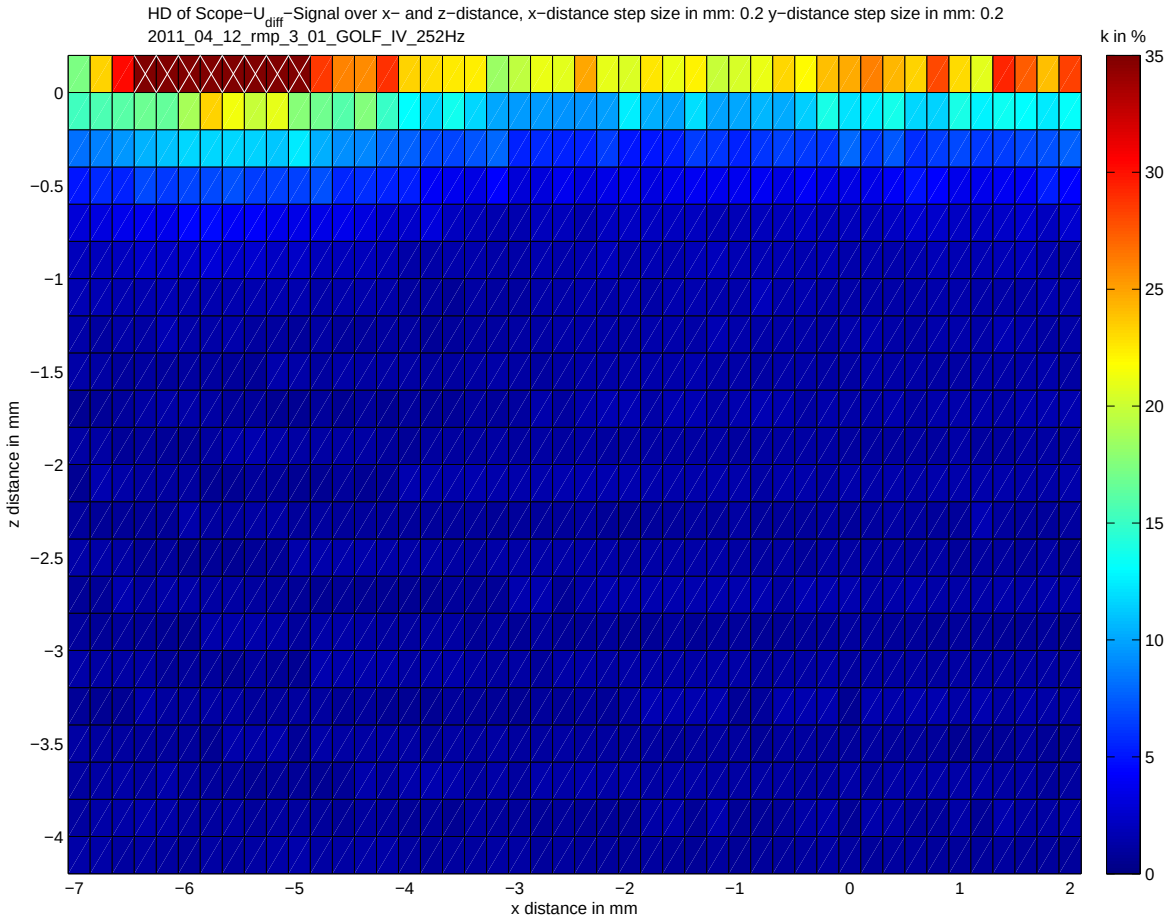


Abbildung 6.15.: Klirrfaktor ermittelt aus dem Sensordifferenzsignal, aufgenommen mit einem Speicheroszilloskop, bei Verfahren der x-Achse und z-Achse des Sensors

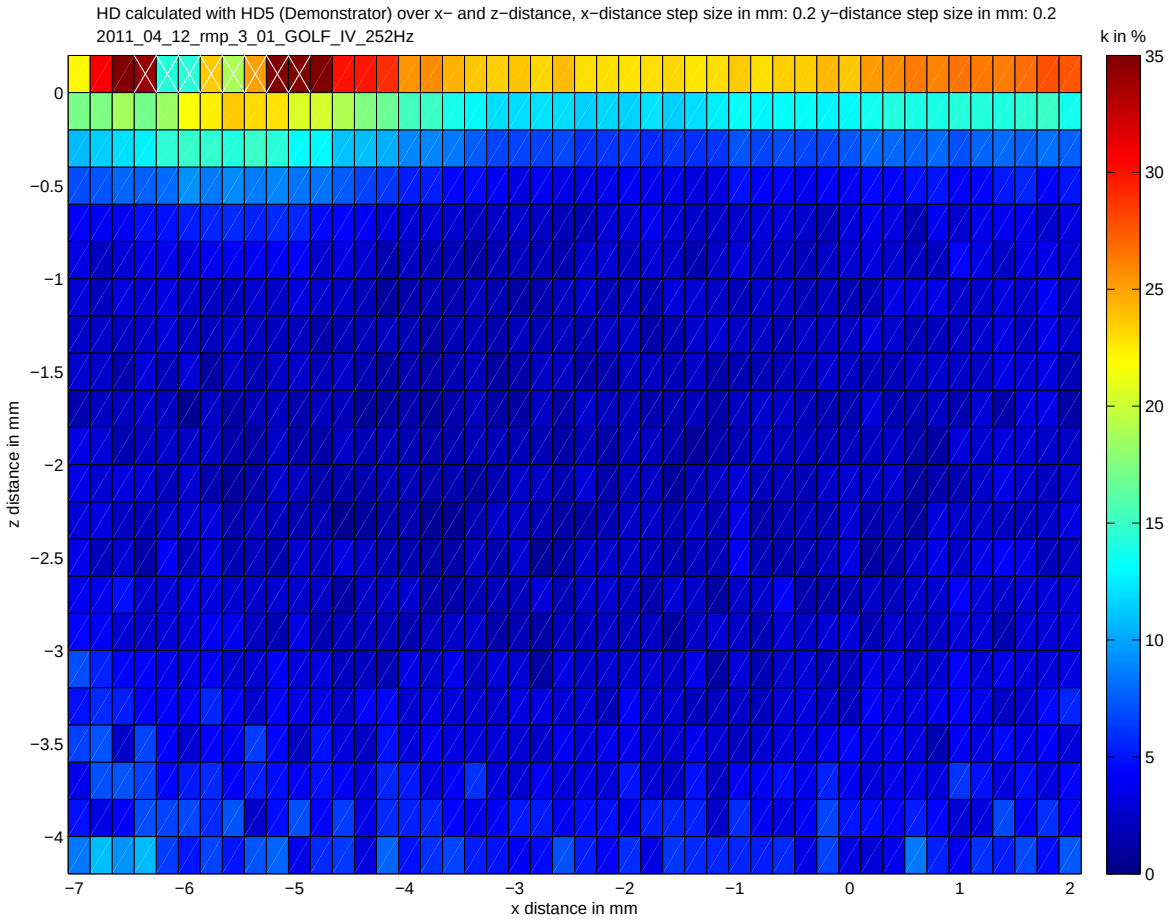


Abbildung 6.16.: Klirrfaktor ermittelt mit dem HD5-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

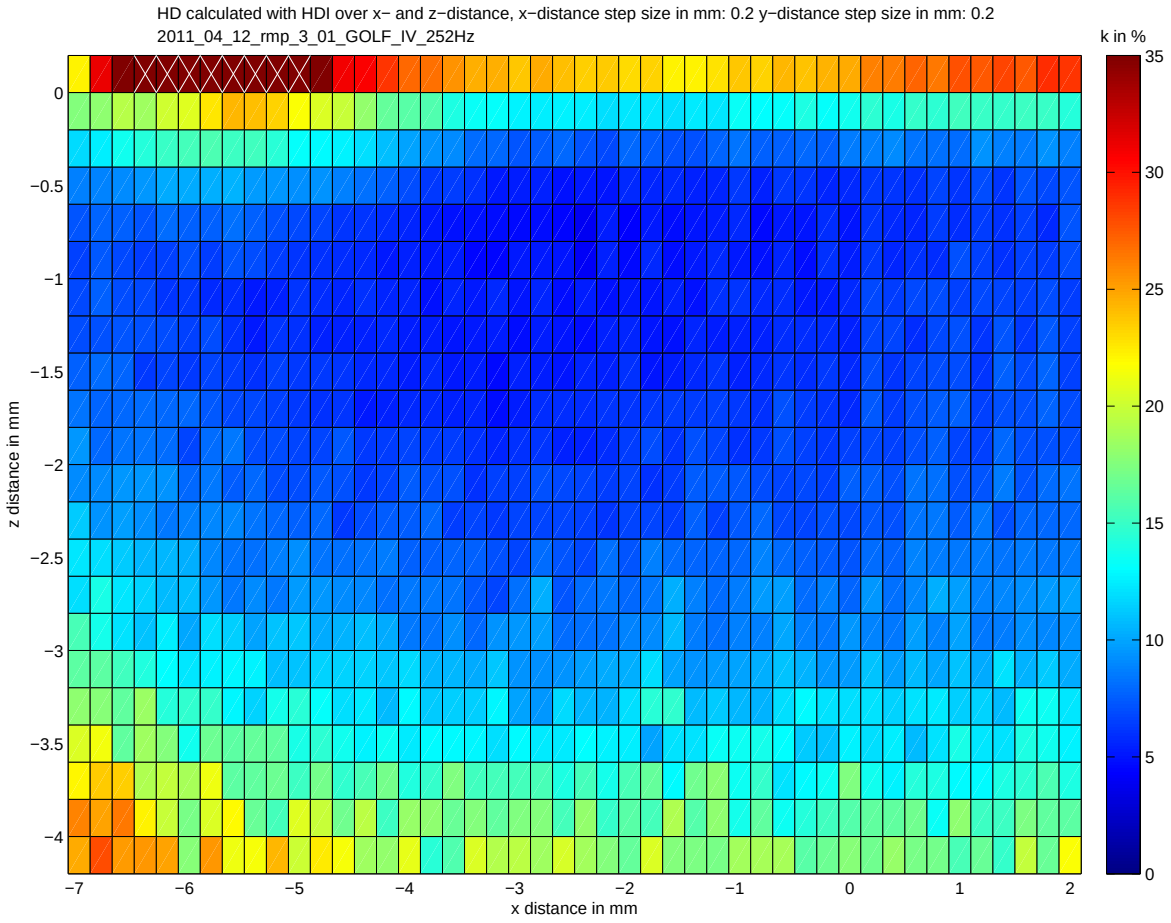


Abbildung 6.17.: Klirrfaktor ermittelt mit dem HDI-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

OHD

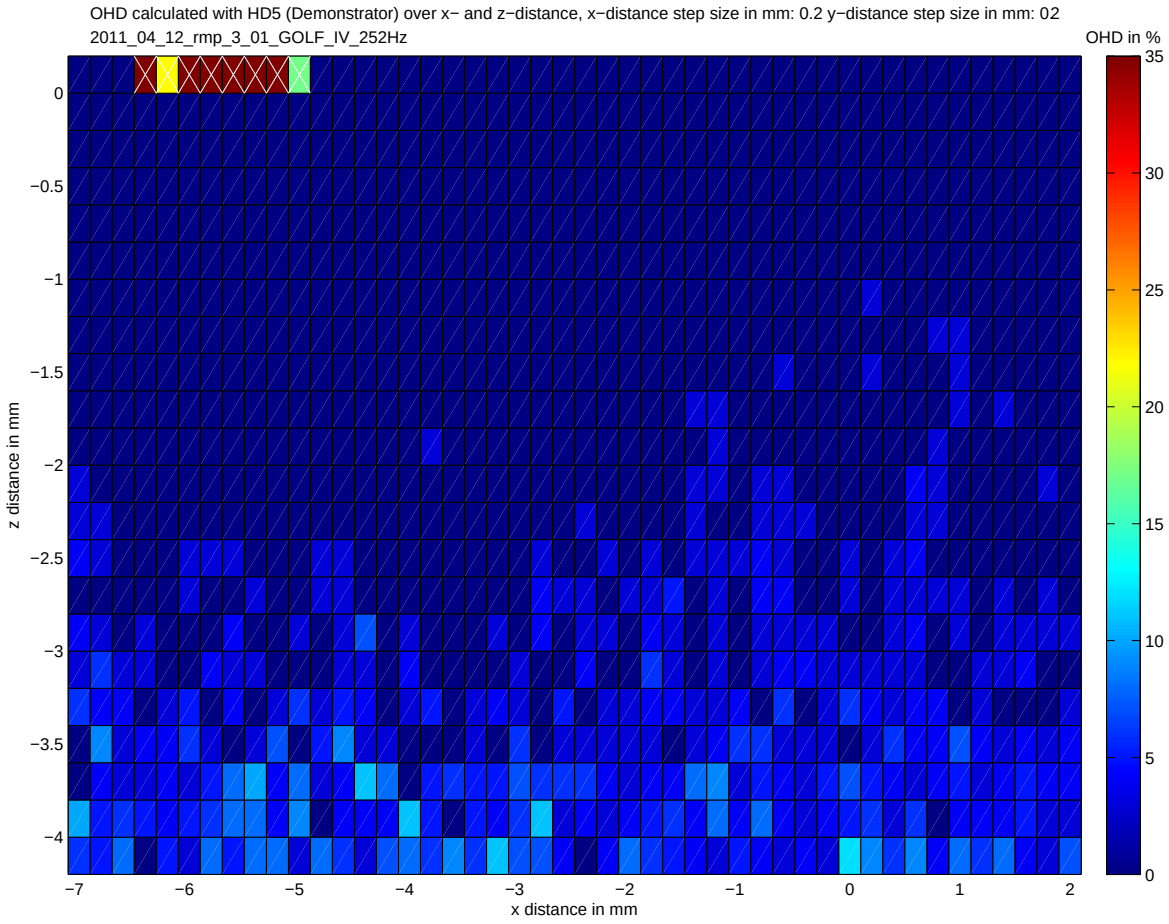


Abbildung 6.18.: OHD ermittelt mit dem HD5-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

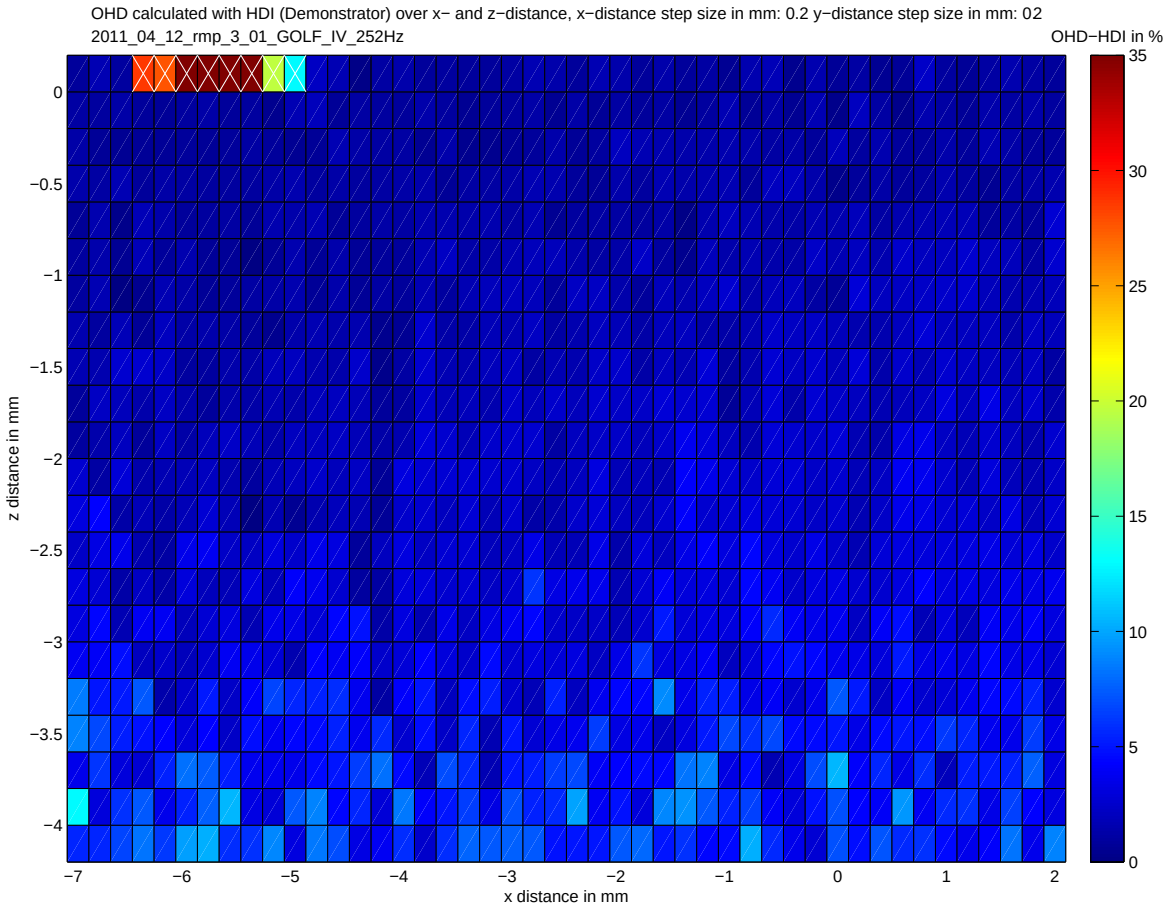


Abbildung 6.19.: OHD ermittelt mit dem HDI-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

Sensorbrückenspannung

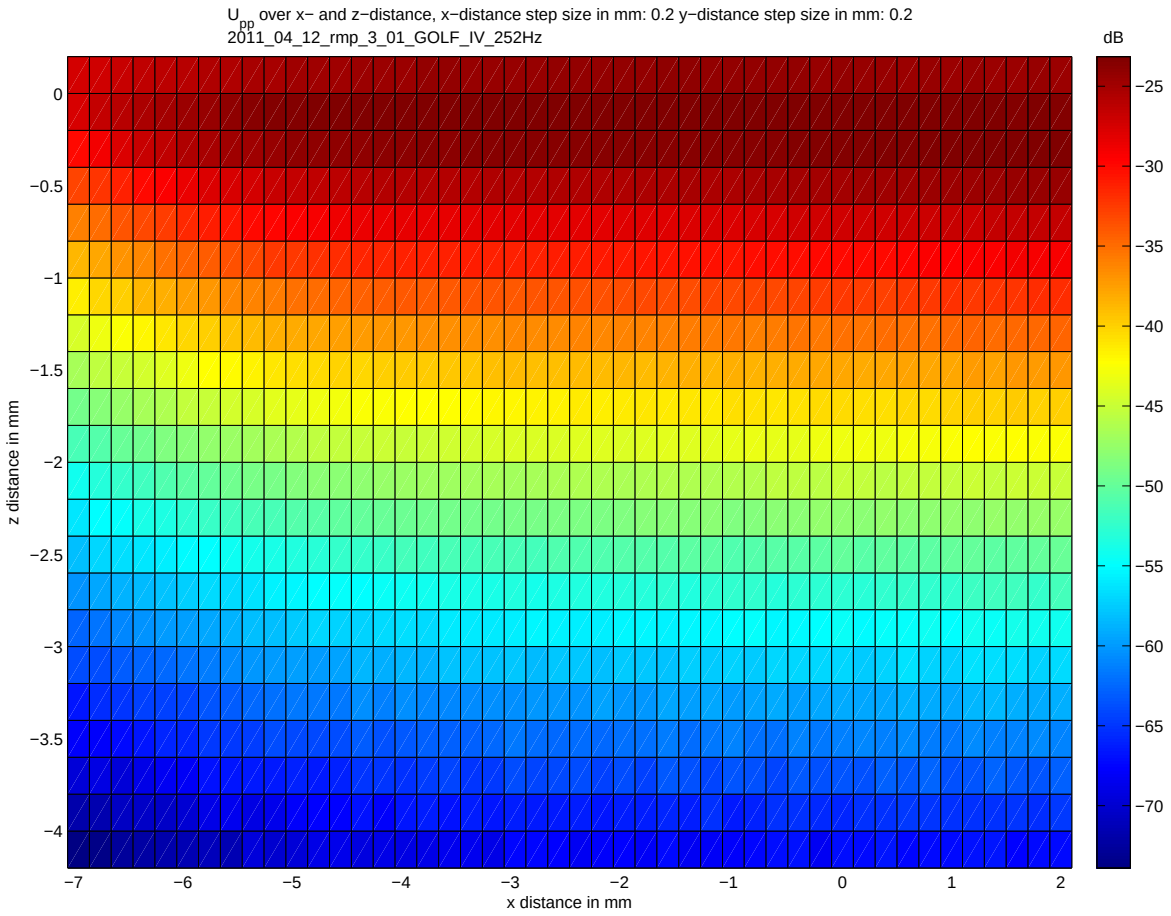


Abbildung 6.20.: Sensorbrückenspannung unverstärkt, bei Verfahren der x-Achse und z-Achse des Sensors

Offset

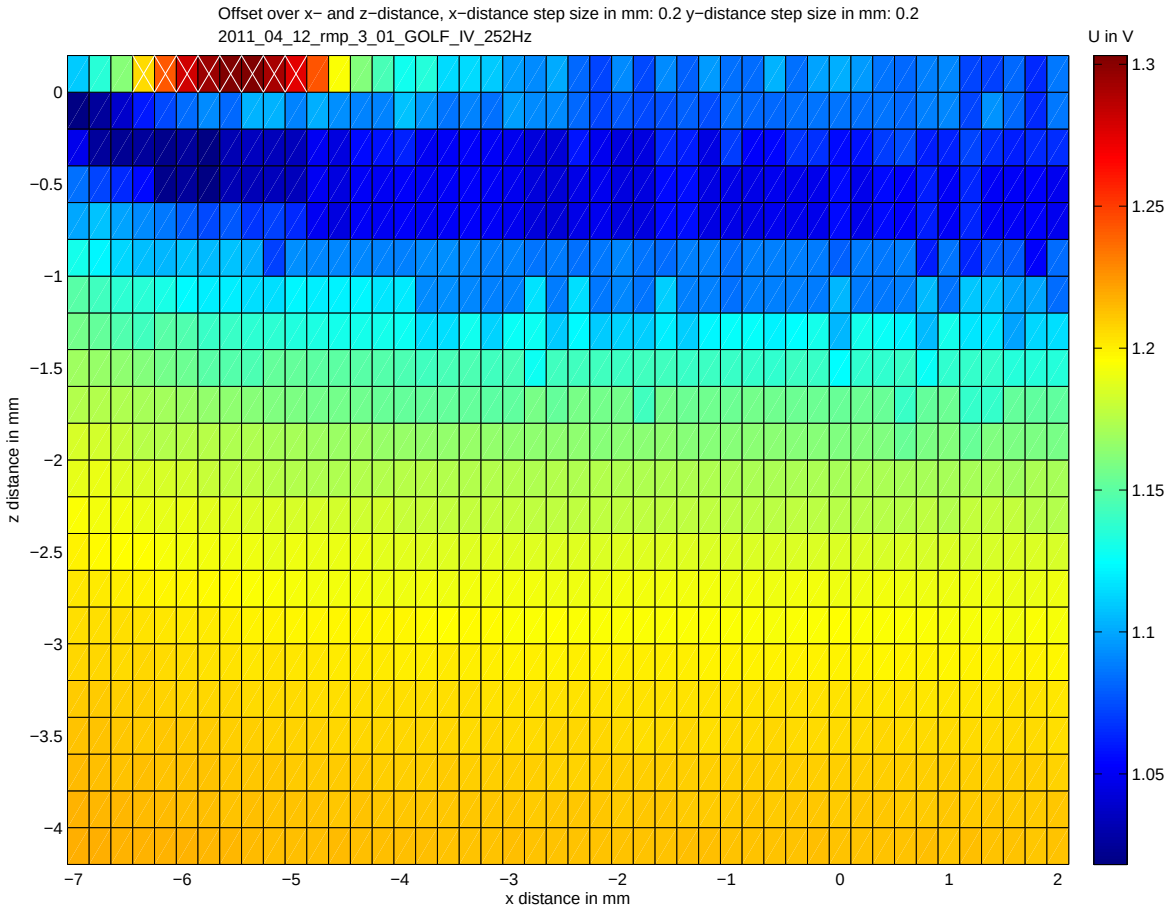


Abbildung 6.21.: Offset, bei Verfahren der x-Achse und z-Achse des Sensors

Verstärkung

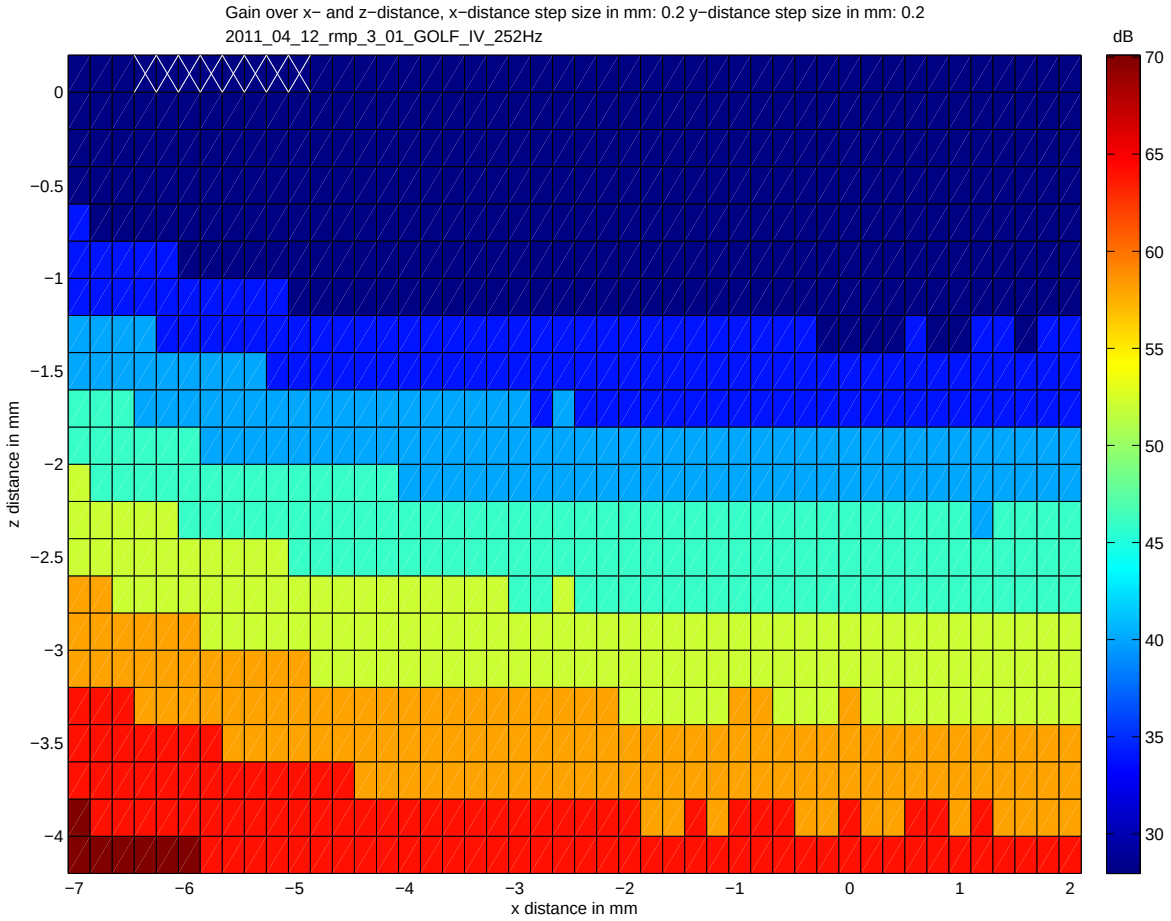


Abbildung 6.22.: Verstärkung, bei Verfahren der x-Achse und z-Achse des Sensors

6.2. Aktives Encodderrad

6.2.1. Golf V - Encoderrad

Die meisten Aussagen die zuvor gemacht worden sind, bestätigen sich auch durch die Messreihe, die am aktiven Encoderrad durchgeführt wurden. Allerdings kommt es zu einem Effekt, der in den Messreihen am passiven Encoderrad nicht beobachtet werden konnte. Die Signalverzerrung, die zusätzliche Nulldurchgänge verursacht und damit die Frequenzverdopplung verursacht, ist an zwei Messpunkten besonders stark. Die zusätzlichen Halbwellen, die üblicherweise bei Frequenzverdopplung auftreten (siehe Abbildung 2.6) sind an diesen Punkten so ausgeprägt, dass die Amplituden genauso groß sind wie die der anderen beiden Halbwellen. Dadurch ist der Frequenzanteil an der Stelle der Harmonischen Basis f_{HB} so gering, wie es ansonsten nur im Normalbetrieb des Sensors ist. Aufgrund dieser Besonderheit, soll an dieser Stelle das Sensorbrückensignal von einem dieser Messpunkte sowie dessen Amplitudenspektrum dargestellt werden (siehe Abbildungen 6.23 und 6.24). Das Amplitudenspektrum zeigt im Bereich von 252Hz keine ausgeprägte Frequenzlinie, obwohl dies der Zahnfrequenz entspricht. Aus diesem Grund bleibt dieser Messpunkt (weißer Bereich) in Abbildung 6.25 ohne Wert, da das Matlab-Skript (Siehe Anhang A.2.1), welches die Harmonischen Anteile aus dem Spektrum heraussucht für die Klirrfaktorberechnung, vergeblich an dieser Stelle sucht.

Dessen ungeachtet ist der Klirrfaktor, ermittelt durch die HDI-Methode, an diesen Messpositionen erhöht und lässt sich damit von den bläulich eingefärbten Bereichen, in denen die Signalqualität am besten ist, unterscheiden (siehe Abbildung 6.27).

Klirrfaktor

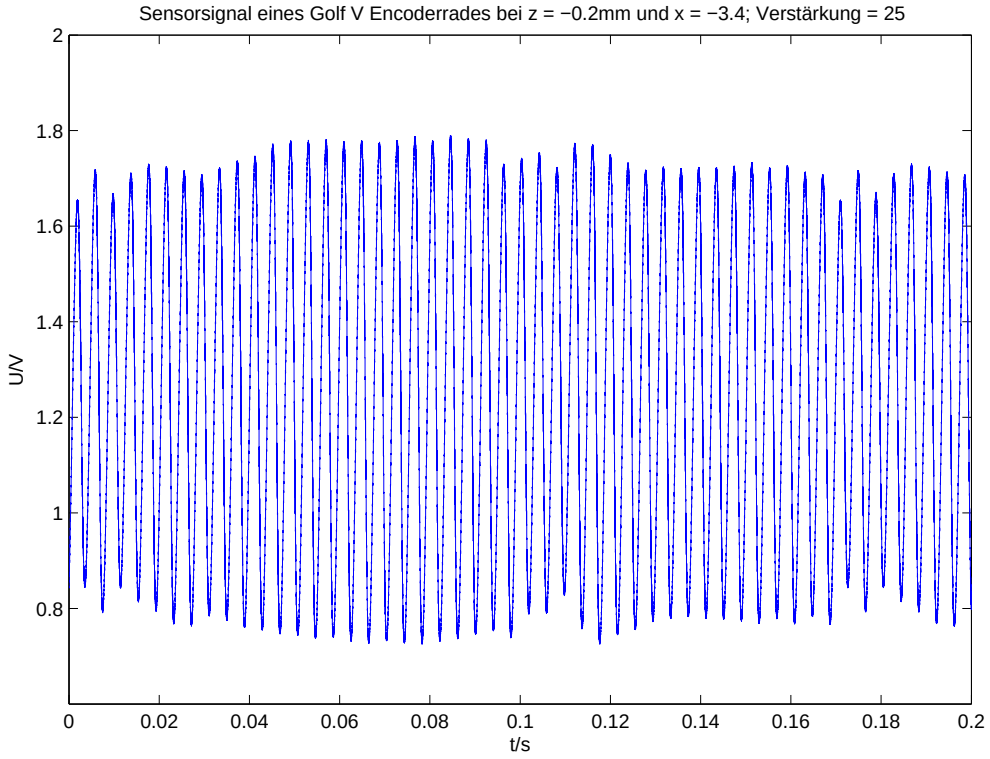


Abbildung 6.23.: Zeitsignal der Sensorbrückenspannung an Position z = -0.2 und x = -3.4

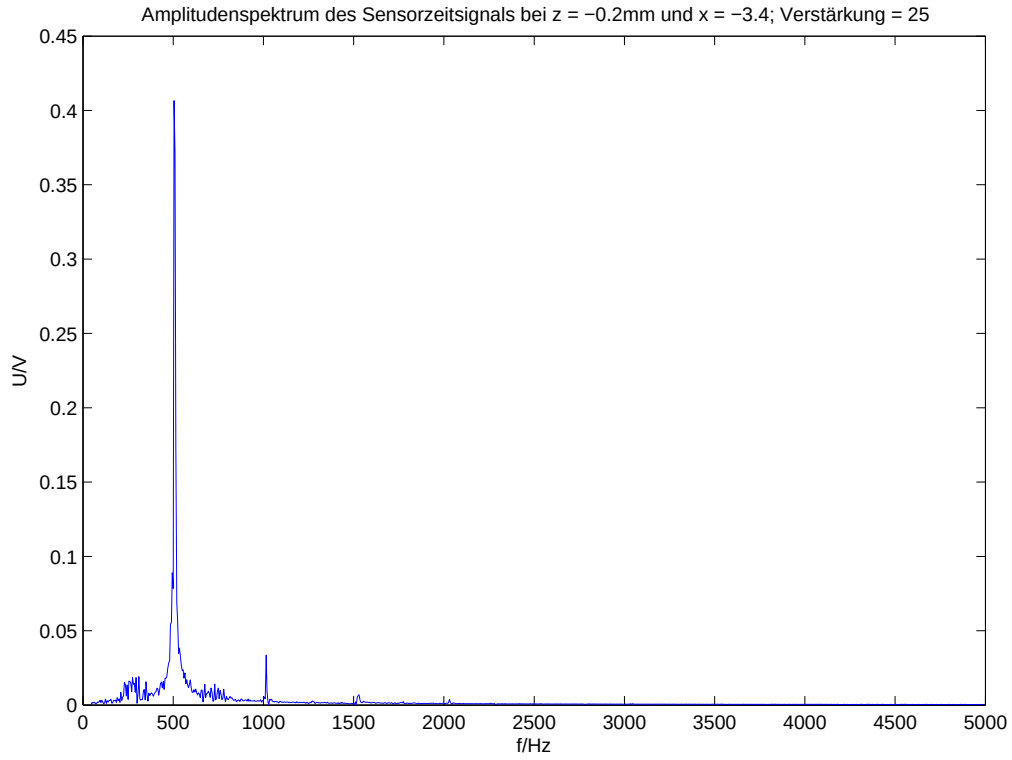


Abbildung 6.24.: Amplitudenspektrum der Sensorbrückenspannung an Position z = -0.2 und x = -3.4

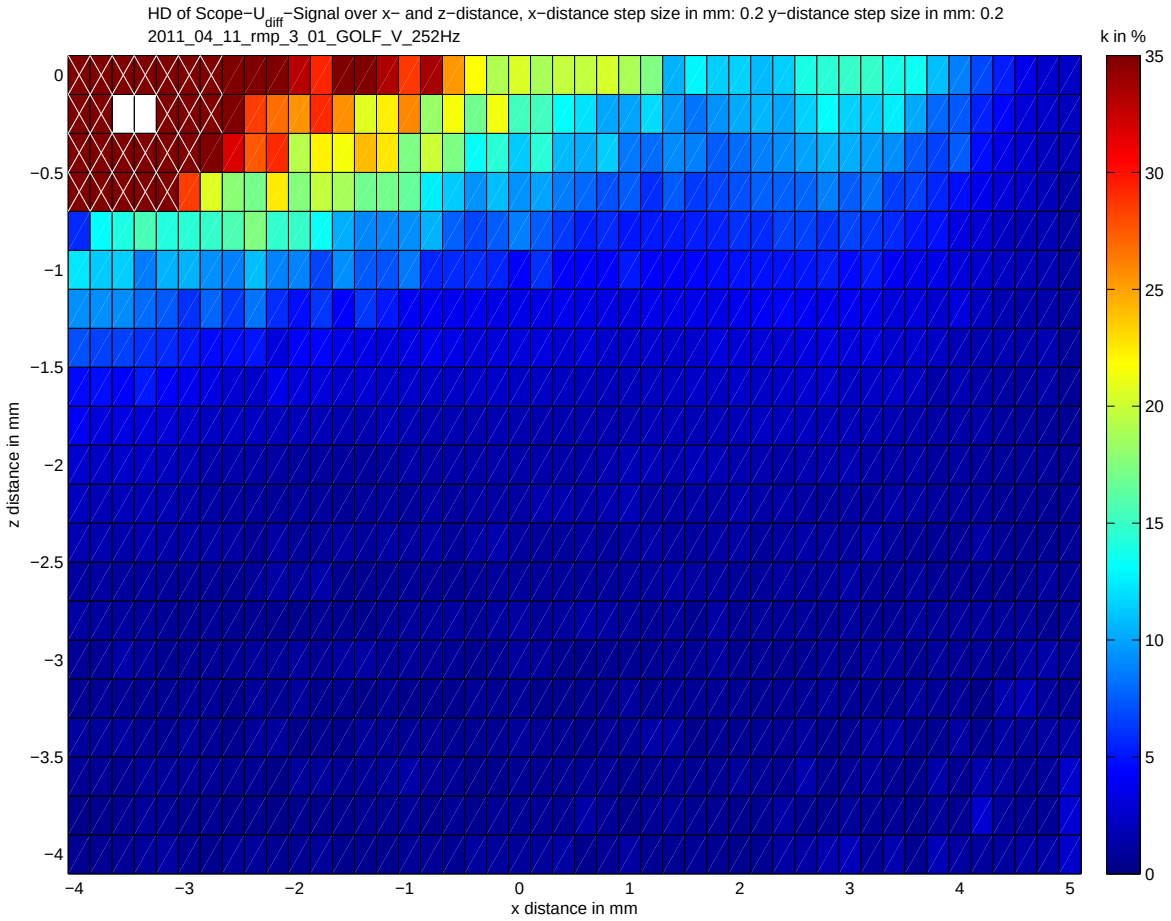


Abbildung 6.25.: Klirrfaktor ermittelt aus dem Sensordifferenzsignal, aufgenommen mit einem Speicheroszilloskop, bei Verfahren der x-Achse und z-Achse des Sensors

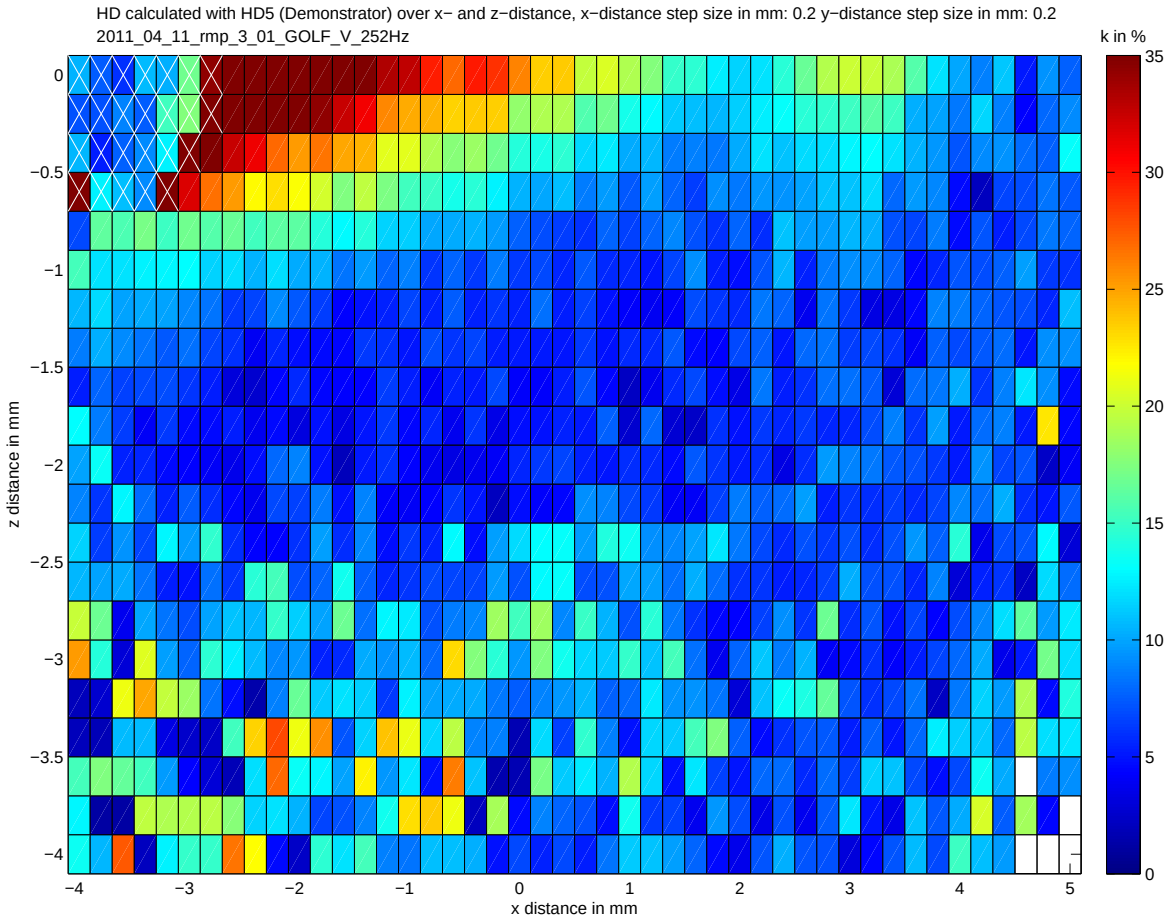


Abbildung 6.26.: Klirrfaktor ermittelt mit dem HD5-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

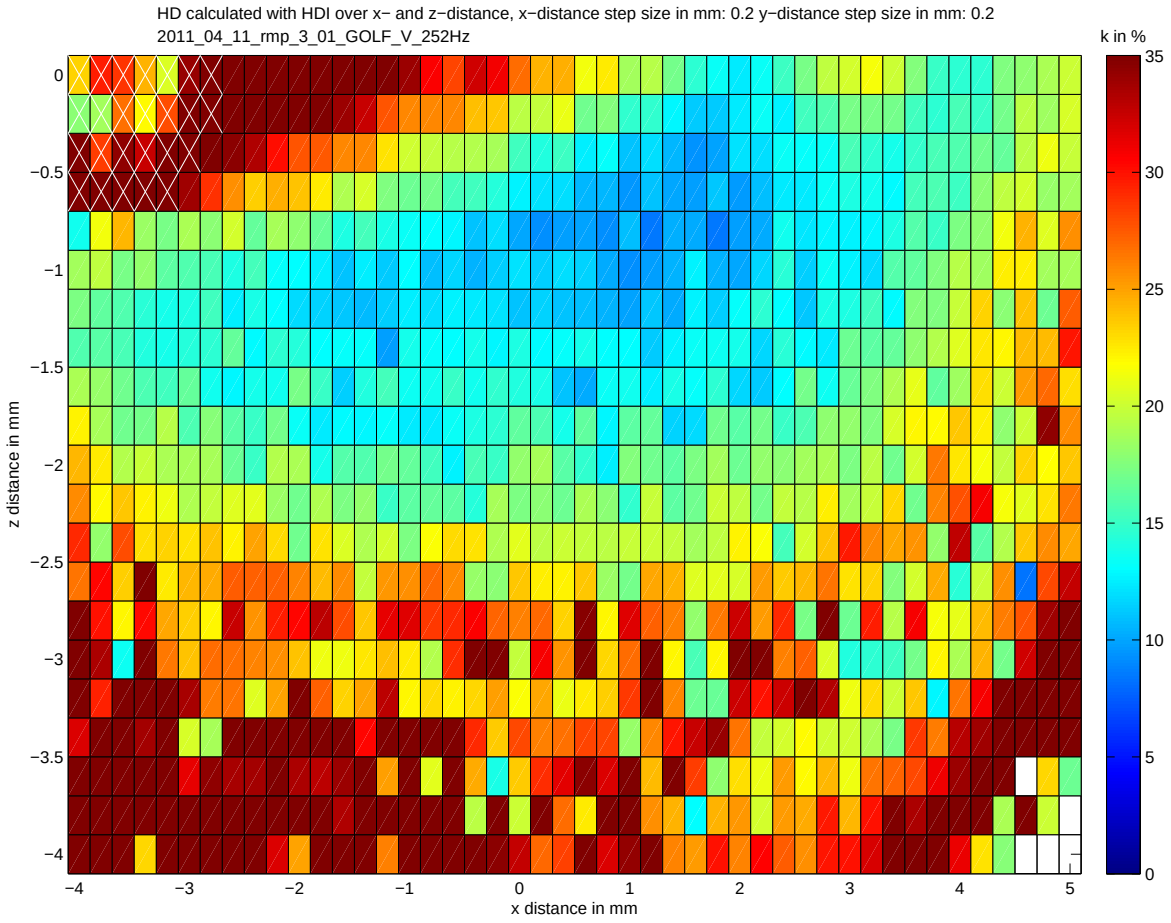


Abbildung 6.27.: Klirrfaktor ermittelt mit dem HDI-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

OHD

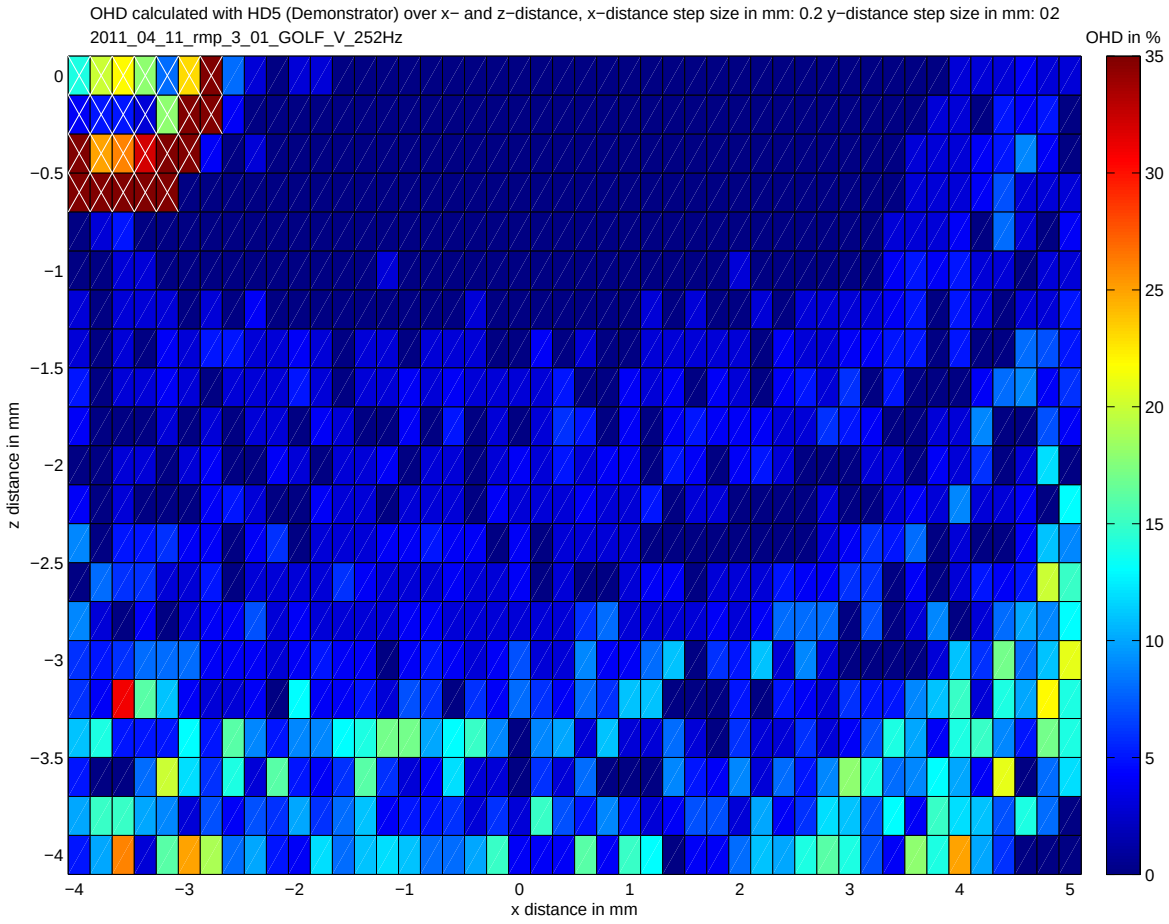


Abbildung 6.28.: OHD ermittelt mit dem HD5-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

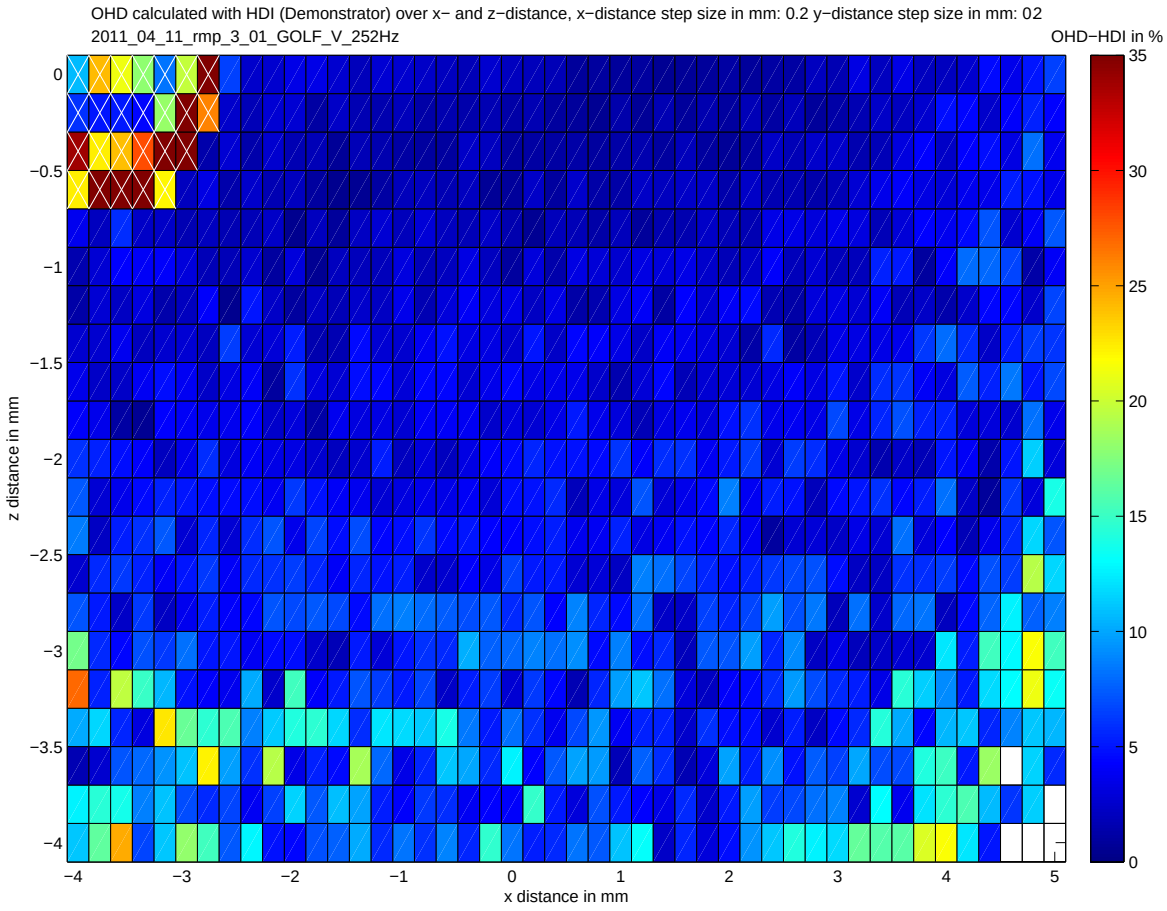


Abbildung 6.29.: OHD ermittelt mit dem HDI-Verfahren, bei Verfahren der x-Achse und z-Achse des Sensors

Sensorbrückenspannung

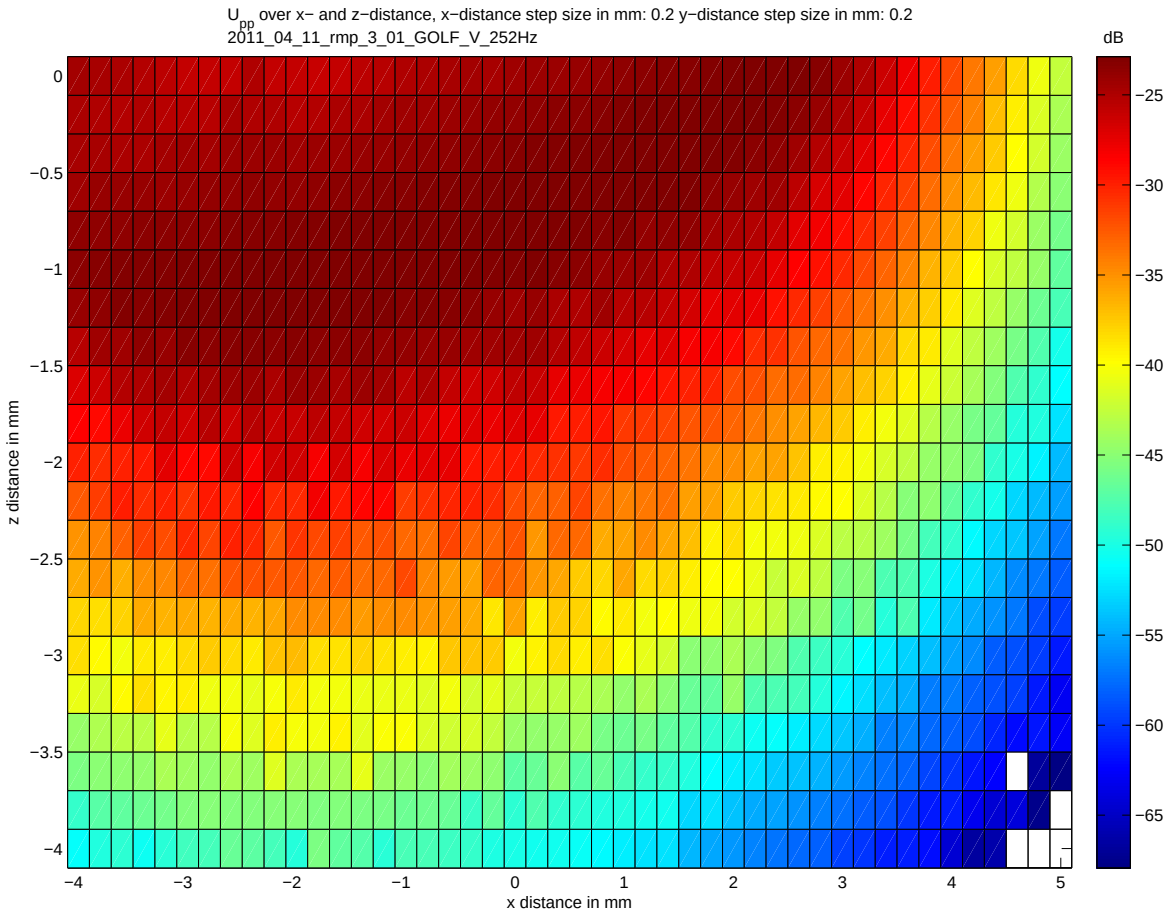


Abbildung 6.30.: Sensorbrückenspannung unverstärkt, bei Verfahren der x-Achse und z-Achse des Sensors

Offset

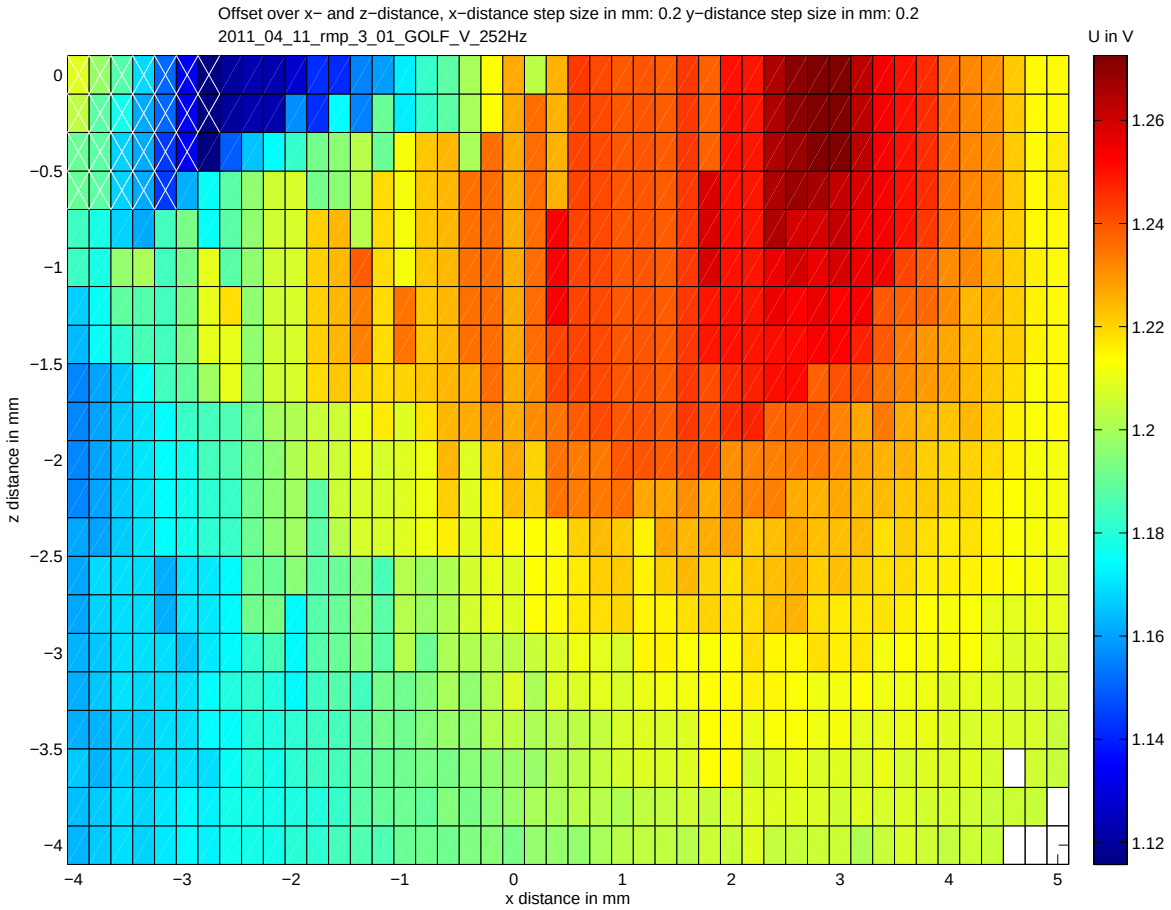


Abbildung 6.31.: Offset, bei Verfahren der x-Achse und z-Achse des Sensors

Verstärkung

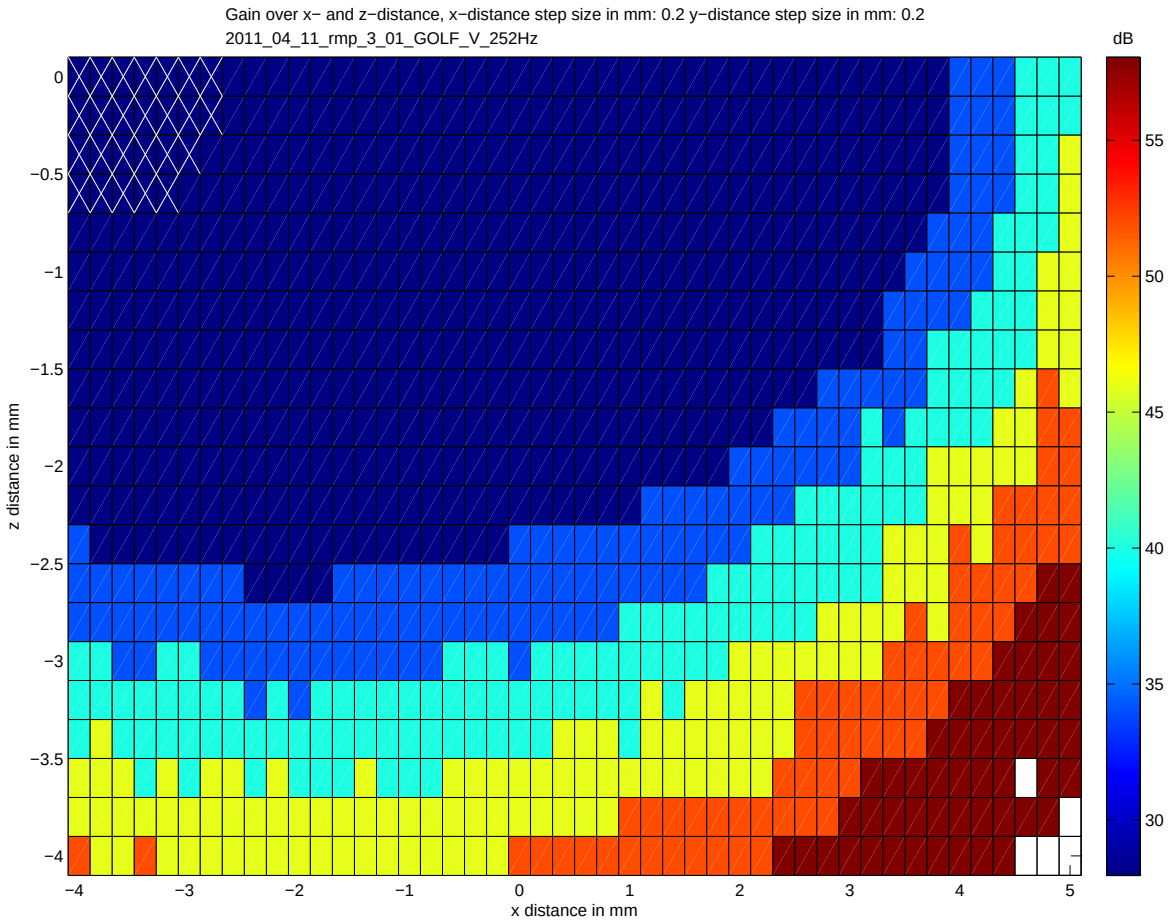


Abbildung 6.32.: Verstärkung, bei Verfahren der x-Achse und z-Achse des Sensors

7. Fazit und Ausblick

7.1. Fazit

Ein Schwerpunkt dieser Arbeit lag auf der Erkennung eines Sensorzustands, der durch Signalverzerrung zu zusätzlichen Nulldurchgängen führt, die von der Controllereinheit der Experimentalplattform als Verdopplung der Raddrehzahl interpretiert wird. In dieser Arbeit konnte mit wenigen Einschränkungen durch Einführung des neuen Indikators OHD diese Signalverzerrung erkannt werden. Eine Einschränkung betrifft den Fall, bei dem es innerhalb einer Radumdrehung nicht durchgängig zu zusätzlichen Nulldurchgängen führt. Dieser Fall konnte durch den anderen neu eingeführten Indikator DSC detektiert werden.

Eine andere Einschränkung betrifft den Fall, indem die durch Signalverzerrung zusätzlich ausgebildeten Halbwellen (siehe Abbildung 2.6) innerhalb einer Encoderinkrementierung, näherungsweise die gleiche Amplitude besitzen wie die anderen beiden Halbwellen. Dieser Fall konnte allerdings nur bei dem aktiven Encoderrad des Golf V beobachtet werden (siehe Abschnitt 6.2.1).

Es zeigte sich, dass der Klirrfaktor, berechnet durch die HDI-Methode ein guter Indikator zur Erkennung von Signalverzerrungen und Frequenzverdopplung ist. Allerdings differenziert er nicht zwischen diesen beiden Fällen.

Ein verzerrtes Sensorsignal bedeutet, dass entweder der ABS-Sensor aus der idealen Einbaulage geraten ist, Beschädigungen am Encoderrad vorliegen oder das Sensorsignal auf eine andere Weise gestört ist. Es ist somit zu empfehlen in einem solchen Fall das ABS-System abzuschalten und eine Warnleuchte im Fahrzeug zu aktivieren um auf das gestörte ABS-System hinzuweisen. Für einen solchen Anwendungsfall reicht es aus, den einen Indikator zu betrachten.

7.2. Ausblick

Der Klirrfaktor, berechnet durch die HDI-Methode, könnte der gesuchte universelle Indikator für die Qualität des Sensorsignals und damit für die Diagnose einer Störung des Sensors

sein. Vermutlich würde die HD5-Methode ein ähnliches Ergebnis liefern, wenn die der Klirrfaktorberechnung zugrunde liegenden Formel 3.4 abgewandelt wird. Dazu müssten nicht nur die geraden berechneten Harmonischen, sondern sämtliche Harmonische mit einbezogen werden. Dies könnte in zukünftigen Arbeiten untersucht werden.

Durch eine Chip-Implementierung könnten die Probleme, die bei einem Radunrundlauf auftreten, beseitigt werden. Ein solcher Chip könnte wesentlich schneller arbeiten und somit eventuell auf eine sequentielle Abtastung verzichten. Da anzunehmen ist, dass dieser Radunrundlauf und auch Fertigungsungenauigkeiten unter Realbedingungen im Fahrzeug den Normalfall darstellen ist es nötig, dass demgegenüber die Sensordiagnose in Maßen unempfindlich ist.

Literaturverzeichnis

[1] Daimler AG. Requirement Specifications for Standardized Interface for Wheel Speed Sensors with Additional Information „AK-Protokoll“, Februar 2008.

[2] R. Boll and K. J. Overshott. Magnetic Sensors. In W. Göpel, J. Hesse, and J. N. Zemel, editors, *Sensors a Comprehensive Survey*, volume 5. VCH, Weinheim, 1989. insb. Kap. 9 von U. Dibbern.

[3] Wechselstromgrößen; Zweileiter-Stromkreise, 1994.

[4] K. R. Fyfe and E. D. S. Munck. Analysis of Computed Order Tracking, 1997.

[5] Kalin Ivanov. Fehlersichere Automatisierung eines Encoder-Messplatzes zur Untersuchung von ABS-Sensoren. laufende diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[6] Niels Jegenhorst. Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, Oktober 2009.

[7] R. Wirth K. Uchtmann. Maschinendiagnose an drehzahlveränderlichen Antrieben mittels Ordnungsanalyse, July 2001.

[8] Lennart Koch. Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, April 2010.

[9] H. Lindner, H. Brauer, and C. Lehmann. *Taschenbuch der Elektrotechnik und Elektronik*. Fachbuchverlag, Leipzig, 8. aufl. edition, 2005.

[10] Abdelkhalek Mahtouf. Messungen und Signalanalyse an einem magnetischen Sensor. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, Dezember 2008.

[11] Christian Schoermer. AMR-Messbrücken fur ABS-Sensoren. Studienarbeit, Hochschule für Angewandte Wissenschaften Hamburg / NXP Semiconductors, März 2008.

[12] Christian Schoermer. Automatisierter Radmessplatz fur ABS-Sensoren mit aktiven und passiven Encodern verschiedener Automobil-Hersteller. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2010.

[13] Martin Stahl. Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion. Bachelorarbeit, Hochschule für Angewandte Wissenschaften Hamburg, Februar 2010.

A. Anhang

A.1. Messungen

A.1.1. Abtastung

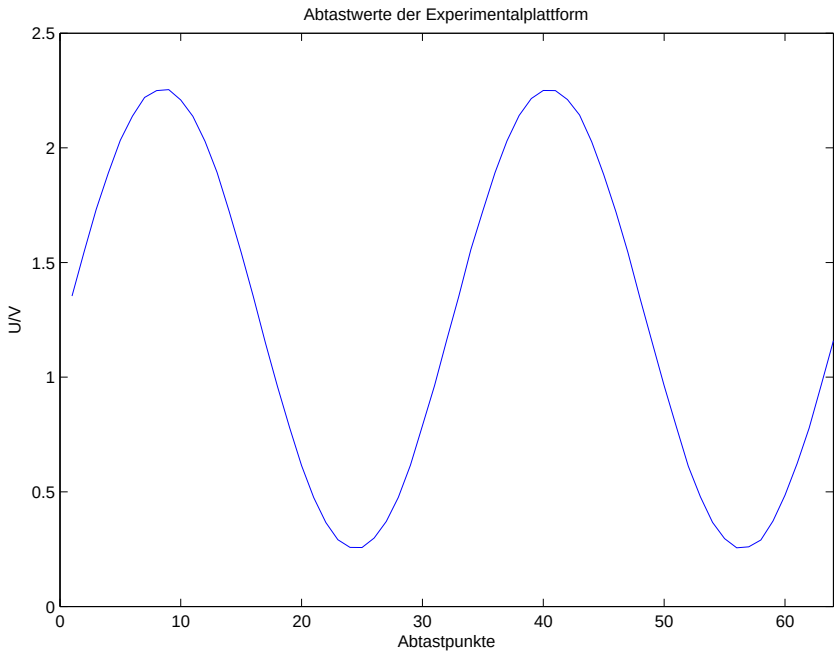


Abbildung A.1.: Abgetastetes Sinus-Signal bei 10Hz

A.1.2. Laufzeitmessung der HD5 und HDI Implementation

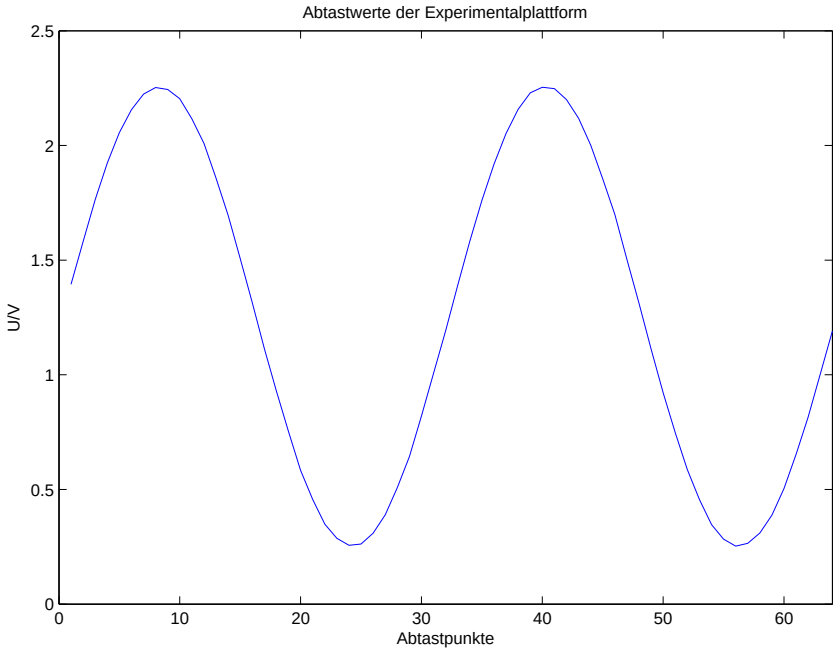


Abbildung A.2.: Abgetastetes Sinus-Signal bei 100Hz

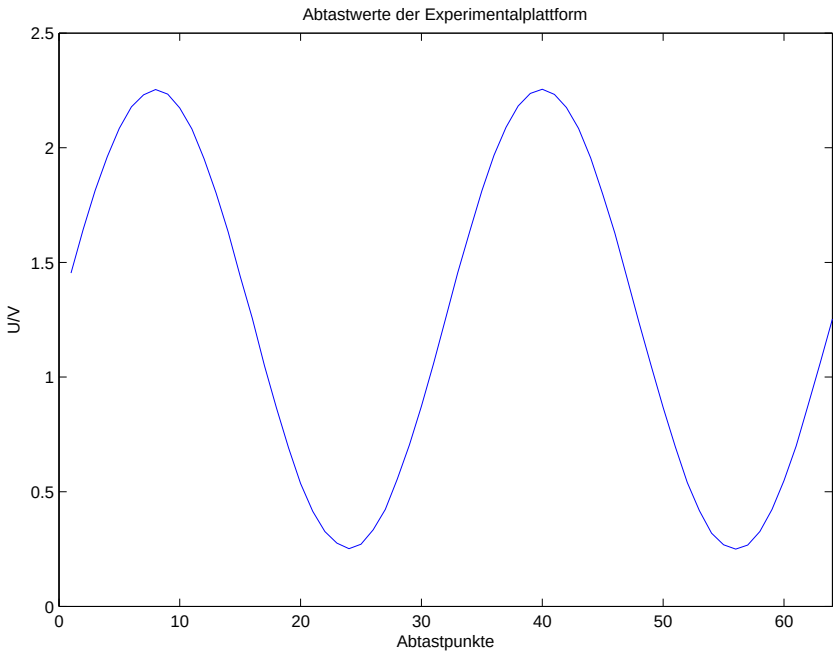


Abbildung A.3.: Abgetastetes Sinus-Signal bei 250Hz

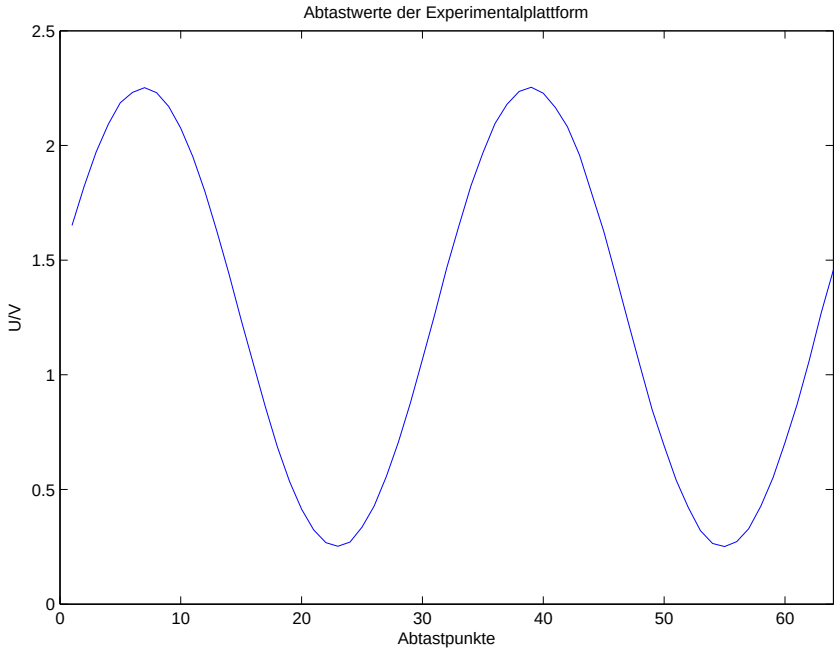


Abbildung A.4.: Abgetastetes Sinus-Signal bei 750Hz

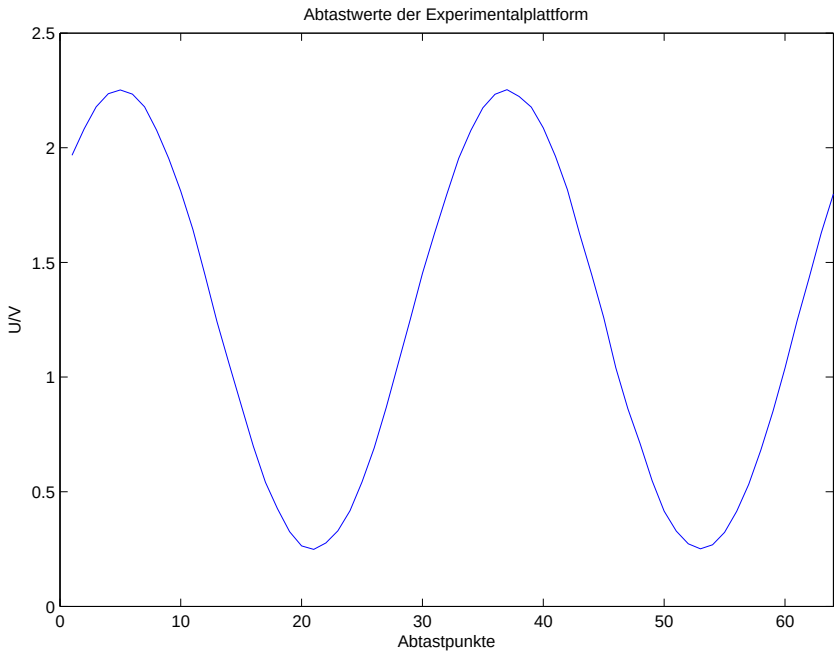


Abbildung A.5.: Abgetastetes Sinus-Signal bei 1500Hz

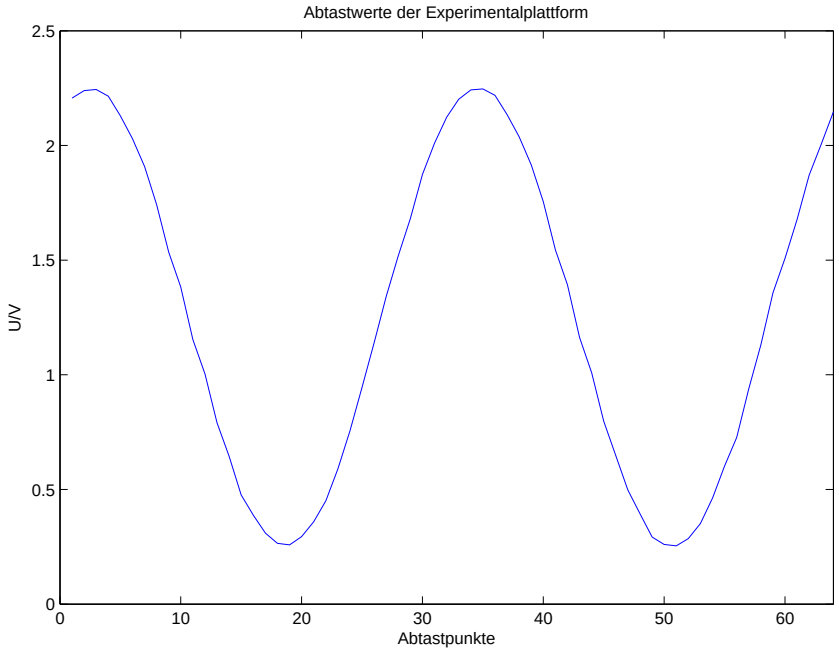


Abbildung A.6.: Abgetastetes Sinus-Signal bei 2500Hz

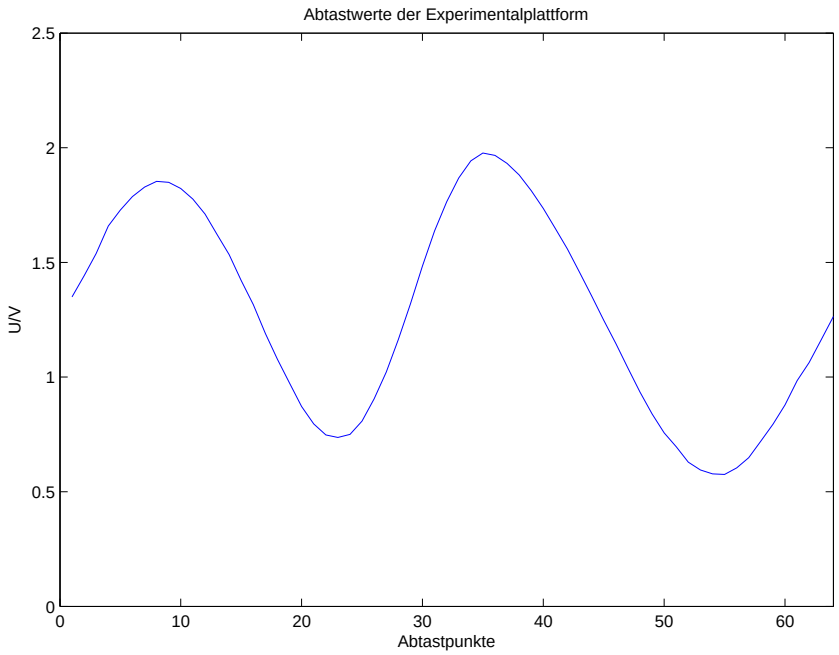


Abbildung A.7.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 10Hz

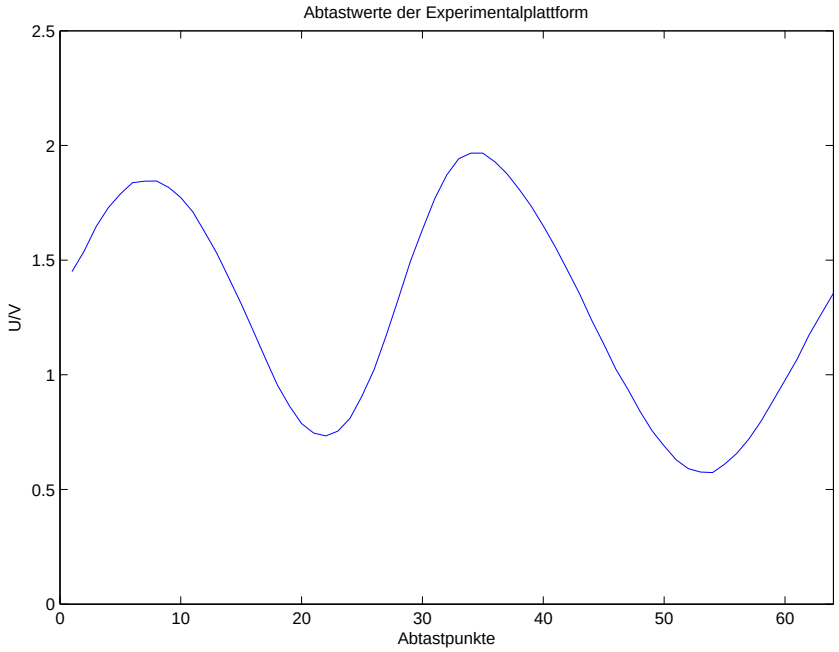


Abbildung A.8.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 250Hz

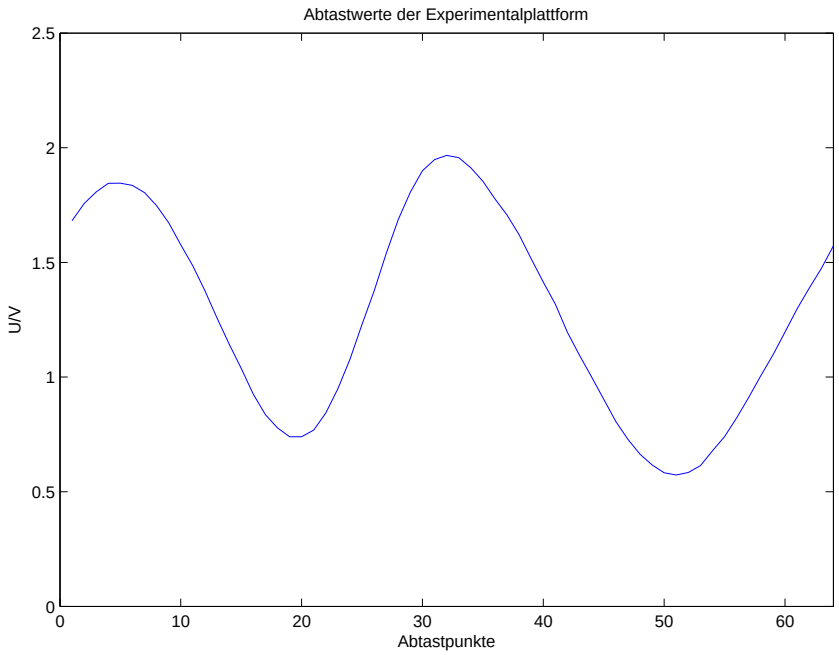


Abbildung A.9.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 750Hz

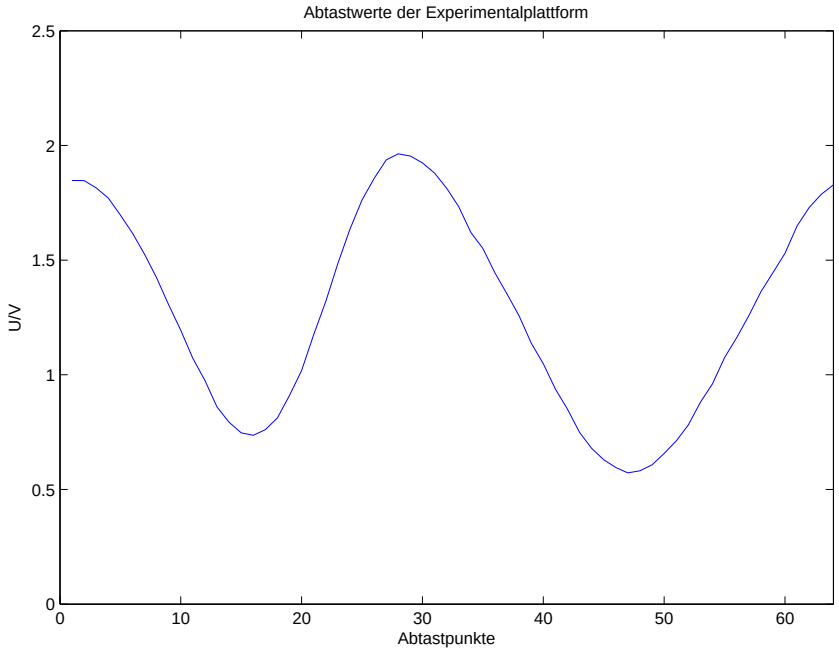


Abbildung A.10.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 1500Hz

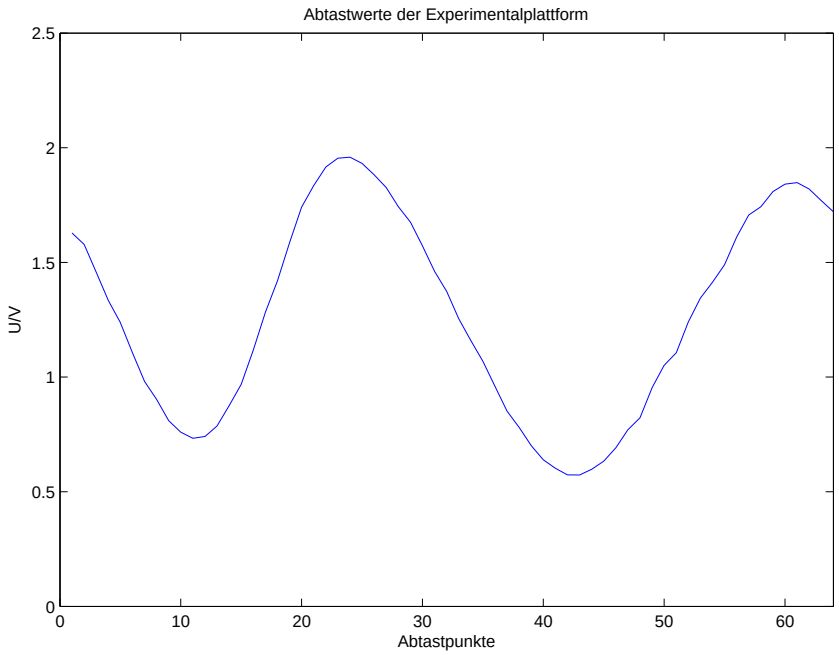


Abbildung A.11.: Abgetastetes Sensorsignal mit Frequenzverdopplung bei 2500Hz

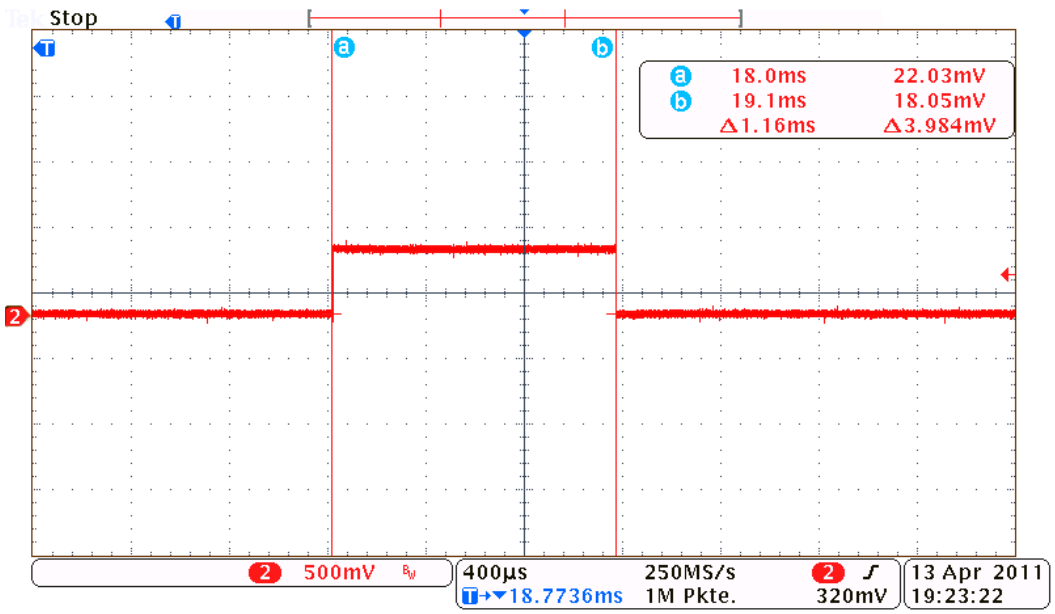


Abbildung A.12.: HD5: Verarbeitungszeit eines einzelnen Abtastwertes

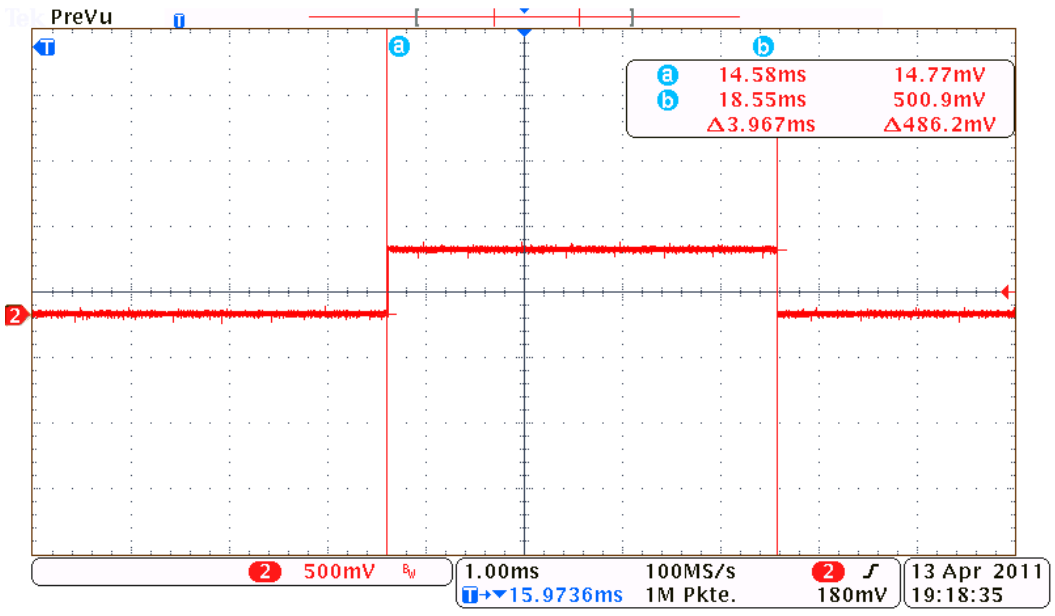


Abbildung A.13.: HD5: Berechnungszeit von Klirrfaktor und OHD aus sämtlichen Abtastwerten

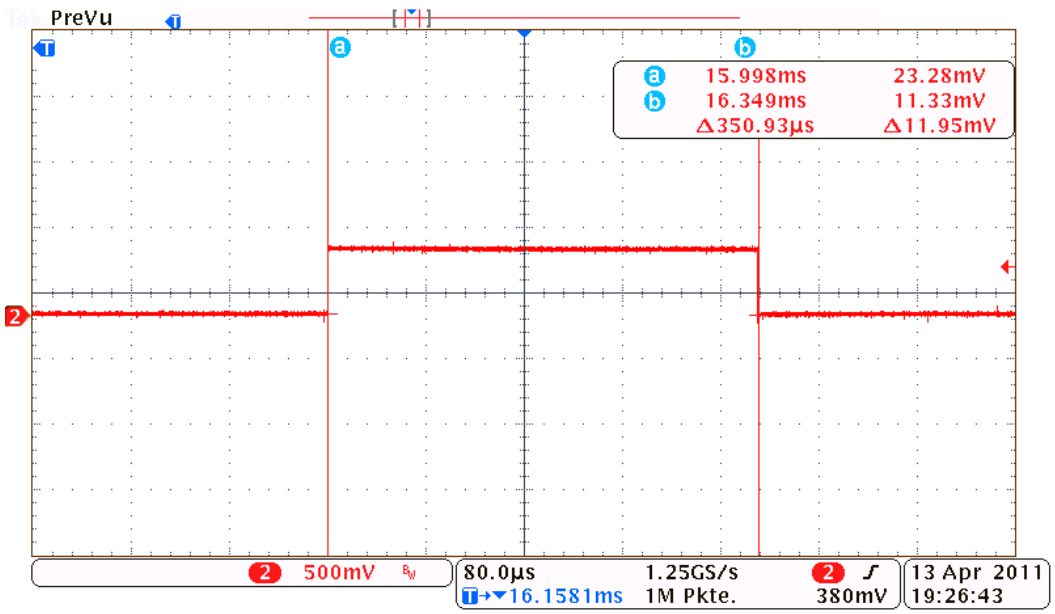


Abbildung A.14.: HDI: Verarbeitungszeit eines einzelnen Abtastwertes

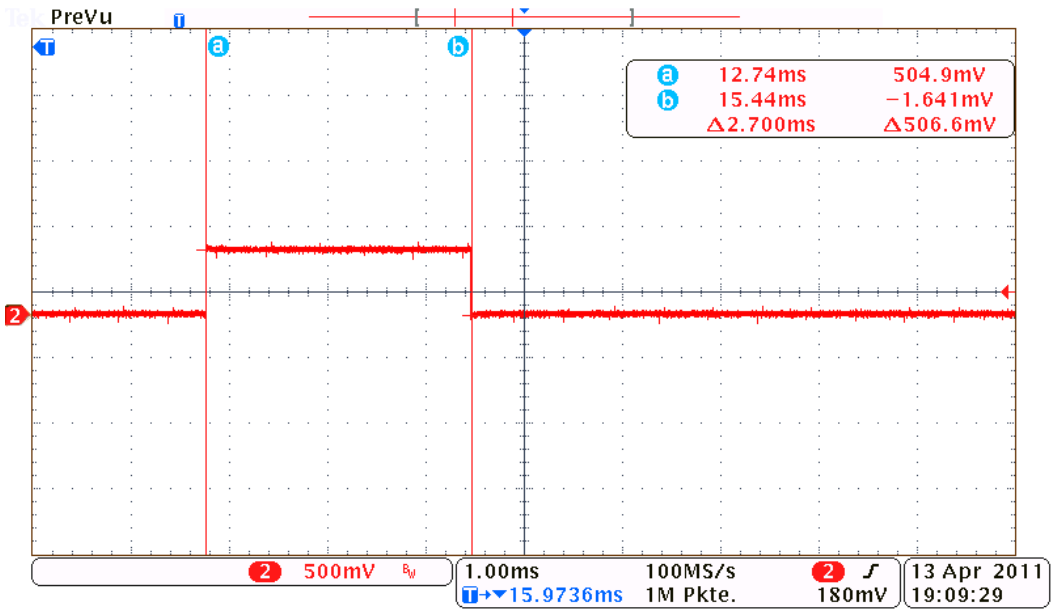


Abbildung A.15.: HDI: Berechnungszeit von Klirrfaktor und OHD aus sämtlichen Abtastwerten

A.2. Quelltexte

A.2.1. Matlab

Analyseskript

```
%
% Programm zur Auswertung der aufgezeichneten Demoboard-Daten
%
% Version 1.0
% Datei:          rmp_active_enc_stepper_demo_scope_record_analyse.m
%
%
% erstellt von: Heiko Poppinga
% erstellt am:  23.02.2010
%
% geï¿½ndert von:  Heiko Poppinga & Martin Krey
%                 15.04.2011
% ï¿½nderungen:   Einlesen der Daten, Darstellung, Speicherung der Plots,
%               Speicherung der Verlï¿½ufe in .mat File
%
%
% Beschreibung: Aufgrund der Datenmenge werden mehrere mat-files
    eingelesen
%               Dueses Skript eignet sich nur fuer Messreihen, mit
%               Verfahren der x-Achse un Verkippung.
%
% Folgende Plots werden erstellt:   Radarplots : HD, Gain, upeak
%
%
%
%function rmp_active_enc_stepper_demo_scope_record_analyse(DirName)

if ~exist('DirName','var')
```

```
28      disp('-- Start rmp_active_enc_stepper_demo_scope_record_analyse.m --'
            );
        % Workspace loeschen
30      clear all;
        % Verzeichnis zum Einlesen auswaehlen
32      DirName = uigetdir(pwd,'Choose directory with data-files');
        function_call=false;
34  else
        % script is called as funtion
36      function_call=true;
    end
38
    <<<<<<< .mine
40  PLOT_ALL = false;
    =======
42  plot_all = true;
    >>>>>>> .r1244
44  rmp = 2;
    list_of_files    = what(DirName);
46  list_of_files.mat = sort(list_of_files.mat);

48  % nur Messdateien auswaehlen
    files = regexp(list_of_files.mat, '.*_part[0-9]+.rmp_3.mat', 'match');
50  files=[files{:}];
    files = sort(files);
52
    %% -- Einlesen der Messdaten
       ------------------------------------------
54  k = 1;
    load(fullfile(list_of_files.path, files{1}), 'parameters');
56  if isfield(parameters,'stepper')
        x_values          = parameters.stepper.x.values;
58      y_values          = parameters.stepper.y.values;
        phi_values        = parameters.stepper.phi.values;
60  else
        rmp = 3;
62          %x-values
        x_values      = parameters.measurement_parameter.x_min : ...
64          parameters.measurement_parameter.x_step : ...
            parameters.measurement_parameter.x_max;
66      % y-values
        y_values      = parameters.measurement_parameter.y_min : ...
68          parameters.measurement_parameter.y_step : ...
            parameters.measurement_parameter.y_max;
```

```
70      % z-values
        z_values      = parameters.measurement_parameter.z_min : ...
72          parameters.measurement_parameter.z_step : ...
            parameters.measurement_parameter.z_max;
74      % phi_x-values
        phi_x_values = parameters.measurement_parameter.phi_x_min : ...
76          parameters.measurement_parameter.phi_x_step : ...
            parameters.measurement_parameter.phi_x_max;
78      % phi_y-values
        phi_y_values = parameters.measurement_parameter.phi_y_min : ...
80          parameters.measurement_parameter.phi_y_step : ...
            parameters.measurement_parameter.phi_y_max;
82      % phi_z-values
        phi_z_values = parameters.measurement_parameter.phi_z_min : ...
84          parameters.measurement_parameter.phi_z_step : ...
            parameters.measurement_parameter.phi_z_max;
86  end

88  demoAFG_file = regexp(list_of_files.mat, '.*demo_from_AFG.rmp_3.mat', '
        match');
    demoAFG_file=[demoAFG_file{:}];
90  if size(demoAFG_file,1)
        load(fullfile(list_of_files.path, demoAFG_file{1}), 'demoAFG');
92      isDemoAFG = true;
    else
94      isDemoAFG = false;
    end
96  isDemoAFG = false;

98  try
        if exist('scope_analysed','var')
100         load(fullfile(list_of_files.path,files{1}), 'scope_analysed');
            ANALYSED_SCOPE_APPEND= true;
102     else
            ANALYSED_SCOPE_APPEND = false;
104     end
    catch ME
106     ANALYSED_SCOPE_APPEND = false;
    end
108 analyse_file = regexp(list_of_files.mat, '.*scope_analysed.rmp_3.mat', '
        match');
    analyse_file=[analyse_file{:}];
110 if size(analyse_file,1)
        load(fullfile(list_of_files.path, analyse_file{1}), 'scope_analysed');
```

```
112 end
    if exist('scope_analysed','var')
114     ANALYSED_SCOPE = true;
    else
116     ANALYSED_SCOPE = false;
    end
118 h = waitbar(0,'Please wait..., loading files');
    for i=1:length(files);          % Dateien durchlaufen
120     waitbar(i / length(files));
        load(fullfile(list_of_files.path,files{i}), 'demo');            %
            jeweilige Datei laden
122     if ANALYSED_SCOPE_APPEND
            load(fullfile(list_of_files.path,files{i}), 'scope_analysed');
124     end

126                                         % einzelne Strukturen auslesen


128
        for j=1:length(demo)
130         x(k)              = demo(j).x;
            y(k)              = demo(j).y;
132         if rmp == 3
                z(k)          = demo(j).z;
134             phi_x(k)      = demo(j).phi_x;
                phi_y(k)      = demo(j).phi_y;
136             phi_z(k)      = demo(j).phi_z;
                freq(k)           = demo(j).freq;
138         elseif rmp == 2
                phi(k)        = demo(j).phi;
140             freq(k)           = demo(j).frequency;
            end
142         gain(k)           = demo(j).gain;
            offset(k)         = demo(j).offset;
144         harm_dist_lut(k)  = demo(j).hd_lut;
            sub_hd_lut(k)     = demo(j).sub_hd_lut;
146         harm_dist_abs(k)  = demo(j).hd_abs;
            magnitudes(k,:)   = demo(j).mag(:,1);
148         magnitudes_v(k,:)= demo(j).mag_v(:,1);  % in Volt
            complex(k,:)      = demo(j).comp(:,1);
150         phases(k,:)       = demo(j).angle(:,1);
            u_peak(k)         = demo(j).u_pp;       % in Volt
152         u_diff(k,:)       = demo(j).u_diff;     % in Volt
            u_hb1(k,:)        = demo(j).u_hb1;
154         u_hb2(k,:)        = demo(j).u_hb2;
```

```
        if ANALYSED_SCOPE_APPEND
156         scope_mag_u_diff(k,:) = scope_analysed(j).mag_u_diff;
            scope_mag_u_hb1(k,:) = scope_analysed(j).mag_u_hb1;
158         scope_mag_u_hb2(k,:) = scope_analysed(j).mag_u_hb2;
            scope_angle_u_diff(k,:) = scope_analysed(j).angle_u_diff;
160         scope_angle_u_hb1(k,:) = scope_analysed(j).angle_u_hb1;
            scope_angle_u_hb2(k,:) = scope_analysed(j).angle_u_hb2;
162         scope_freq_u_diff(k,:) = scope_analysed(j).freq_u_diff;
            scope_freq_u_hb1(k,:) = scope_analysed(j).freq_u_hb1;
164         scope_freq_u_hb2(k,:) = scope_analysed(j).freq_u_hb2;
        elseif ANALYSED_SCOPE
166         scope_mag_u_diff(k,:) = scope_analysed(k).mag_u_diff;
            scope_mag_u_hb1(k,:) = scope_analysed(k).mag_u_hb1;
168         scope_mag_u_hb2(k,:) = scope_analysed(k).mag_u_hb2;
            scope_angle_u_diff(k,:) = scope_analysed(k).angle_u_diff;
170         scope_angle_u_hb1(k,:) = scope_analysed(k).angle_u_hb1;
            scope_angle_u_hb2(k,:) = scope_analysed(k).angle_u_hb2;
172         scope_freq_u_diff(k,:) = scope_analysed(k).freq_u_diff;
            scope_freq_u_hb1(k,:) = scope_analysed(k).freq_u_hb1;
174         scope_freq_u_hb2(k,:) = scope_analysed(k).freq_u_hb2;
        end
176     if isDemoAFG && length(demoAFG) >= k
            freqAFG(k)              = demoAFG(k).frequency;
178         gainAFG(k)          = demoAFG(k).gain;
            offsetAFG(k)        = demoAFG(k).offset;
180         harm_dist_lutAFG(k) = demoAFG(k).hd_lut;
            sub_hd_lutAFG(k)    = demoAFG(k).sub_hd_lut;
182         harm_dist_absAFG(k) = demoAFG(k).hd_abs;
            u_peakAFG(k)        = demoAFG(k).u_pp;        % in Volt
184         u_diffAFG(k,:)      = demoAFG(k).u_diff;      % in Volt
            u_hb1AFG(k,:)       = demoAFG(k).u_br1;
186         u_hb2AFG(k,:)       = demoAFG(k).u_br2;
            discard_counter(k)   = demoAFG(k).discard_counter;
188     end

190     k=k+1;
    end
192 clear('demo');                         % cleanup workspace
end
194 close(h);

196 %% OHD Werte

198 for k =1 : size(u_diff,1)
```

```
    [hdi_abs(k) o13hdi(k)]= hdi(u_diff(k,1:64) ./ 2.5 .* 2^12);
end

%%
if(max(gain)<8)
    gaindb = 20*(log10(2.^gain*25));
    u_p_real = double(u_peak)./(double(2.^gain).*25); %
        Verstaerkungsfaktoren herrausrechnen
else
    gaindb = 20*(log10(gain*25));
    u_p_real = double(u_peak)./(double(gain).*25); %
        Verstaerkungsfaktoren herrausrechnen
end

u_p_real_dB = 20*(log10(u_p_real));

u_p_real = u_p_real .* 1000;            %auf mV skalieren
%%
medianFreq = median(freq); %Median der Frequenz bilden
%HD berechnen, bei Frequenzverdopplung
%Index der Messerwerte, an den Frequenzverdopplung auftrat
%ermitteln(Frequenz > Median der Frequenz + 6% )
indexOfDoubleFrequency = freq >= medianFreq + medianFreq *0.06;
if isDemoAFG
    medianFreqAFG = median(freqAFG(z>-1));
    indexOfDoubleFrequencyAFG = freqAFG >= medianFreqAFG + medianFreqAFG
        *0.06;
end
MAX_HARM = length(magnitudes(1,:)); %Wieviele Harmonische wurden
    aufgenommen?

%% HDs von Scopedaten
if ANALYSED_SCOPE
    if(scope_freq_u_diff(1,1) < medianFreq/2 + medianFreq/10)
        hd_scope_u_diff = calc_hd(scope_mag_u_diff(:,2:2:end));
        %ohd_scope_u_diff = calc_ohd(scope_mag_u_diff(:,1:end));
    else
        hd_scope_u_diff = calc_hd(scope_mag_u_diff(:,1:end));
    end

    if(scope_freq_u_hb1(1,1) < medianFreq/2 + medianFreq/10)
        hd_scope_u_hb1 = calc_hd(scope_mag_u_hb1(:,2:2:end));
    else
        hd_scope_u_hb1 = calc_hd(scope_mag_u_hb1(:,1:end));
```

```
    end
240 if(scope_freq_u_hb2(1,1) < medianFreq/2 + medianFreq/10)
        hd_scope_u_hb2 = calc_hd(scope_mag_u_hb2(:,2:2:end));
242 else
        hd_scope_u_hb2 = calc_hd(scope_mag_u_hb1(:,1:end));
244 end
end
246 %% RMP 2 oder 3 haben unterschiedliche Linear- und Kippachsendefinitionen
if rmp == 3
248
    active_lin_axes = [length(x_values) length(y_values) length(z_values)
        ] > 1;
250 active_tilt_axes = [length(phi_x_values) length(phi_y_values) length(
        phi_z_values)] > 1;

252 active_lin_axes_name = char(active_lin_axes .* ['x' 'y' 'z']);
    active_lin_axes_name = active_lin_axes_name(active_lin_axes_name>0);
254
    active_tilt_axes_name = char(active_tilt_axes .* ['x' 'y' 'z']);
256 active_tilt_axes_name = active_tilt_axes_name(active_tilt_axes_name
        >0);

258 switch sum(active_lin_axes)*10+sum(active_tilt_axes) % convert to
        decimal
        case 11
260         disp('1 lin 1 tilt');
            plot_handle=@radar_plot;
262         axis1 = eval(active_lin_axes_name(1));
            axis2 = eval(['phi_' active_tilt_axes_name(1)]);
264         axis1_label = [active_lin_axes_name(1) ' distance in mm'];
            axis2_label = ['phi_' active_tilt_axes_name(1) ' angle in
                degree'];
266         activate_zoom = true;
        case 20
268         disp('2 lin');
            plot_handle=@rect_plot;
270         axis1 = eval(active_lin_axes_name(1));
            axis2 = eval(active_lin_axes_name(2));
272         axis1_label = [active_lin_axes_name(1) ' distance in mm'];
            axis2_label = [active_lin_axes_name(2) ' distance in mm'];
274         activate_zoom = false;
        case 02
276         disp('2 tilt');
            plot_handle=@rect_plot;
```

```
278             axis1 = eval(['phi_' active_tilt_axes_name(1)]);
                axis2 = eval(['phi_' active_tilt_axes_name(2)]);
280             axis1_label = ['phi_' active_tilt_axes_name(1) ' angle in
                    degree'];
                axis2_label = ['phi_' active_tilt_axes_name(2) ' angle in
                    degree'];
282             activate_zoom = false;
            otherwise
284             disp('more/less than 2 axes activated');
                return;
286     end

288 else
        axis2 = phi;
290     axis2_label = 'phi angle in degree';
        axis1 = x;
292     axis1_label = 'x distance in mm';
end
294
if activate_zoom == false
296     indexOfDoubleFrequencyWithoutZoom=indexOfDoubleFrequency;
else
298     indexOfDoubleFrequencyWithoutZoom=zeros(size(indexOfDoubleFrequency))
            >0;
end
300
if plot_all || function_call
302
    %% HD-Abs Plot mit Werten der Experimentalplattform
304      plot_handle(axis1,axis2,harm_dist_abs,'mark',
             indexOfDoubleFrequencyWithoutZoom,...
             'figure name','HD-Abs(Demonstrator)-Plot','plot title','HD
                 calculated with HD5 (Demonstrator)',...
306          'colorbar title','k in %','filename','HD_HD5_Abs_Demo_plot','plot
                 format',...
             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
308          axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
             'double_click_plot_sig', u_diff);
310     %% HD-Abs Plot mit Werten der Experimentalplattform - gezoomt
        if activate_zoom
312          plot_handle(axis1(axis1 >-1),axis2(axis1 >-1),harm_dist_abs(axis1
                 >-1),'mark',indexOfDoubleFrequency(axis1 >-1),...
                 'figure name','HD-Abs(Demonstrator)-Plot','plot title','HD
                     calculated with HD5 (Demonstrator)',...
```

```
314             'colorbar title','k in %','filename','
                    HD_HD5_Abs_Demo_zoom_plot','plot format',...
                'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
316             axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
                'double_click_plot_sig', u_diff);
318         end
            %% HD mit HDI berechnet mit Abtastwerten der Experimentalplattform
320         plot_handle(axis1,axis2,hdi_abs,'mark',
                indexOfDoubleFrequencyWithoutZoom,...
                'figure name','HD-Abs-Plot-mit-HDI','plot title','HD calculated
                    with HDI',...
322             'colorbar title','k in %','filename','HD_HDI_Abs_Demo_plot','plot
                    format',...
                'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
324             axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
                'double_click_plot_sig', u_diff);
326         %% HD mit HDI berechnet mit Abtastwerten der Experimentalplattform -
                gezoomt
            if activate_zoom
328             plot_handle(axis1(axis1>-1),axis2(axis1>-1),hdi_abs(axis1>-1),
                    'mark',indexOfDoubleFrequency(axis1>-1),...
                    'figure name','HD-Abs-Plot-mit-HDI','plot title','HD
                        calculated with HDI',...
330                 'colorbar title','k in %','filename','
                        HD_HDI_Abs_Demo_zoom_plot','plot format',...
                    'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
332                 axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
                    'double_click_plot_sig', u_diff);
334         end
            %% HD  mit Scopedaten
336         if ANALYSED_SCOPE
                plot_handle(axis1,axis2,hd_scope_u_diff,'mark',
                    indexOfDoubleFrequencyWithoutZoom,...
338                 'figure name','HD-Scope_U_diff-Plot','plot title','HD of
                        Scope-U_{diff}-Signal',...
                    'colorbar title','k in %','filename','HD_Scope_plot','plot
                        format',...
340                 'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
                    axis2_label,'marker color',[1 1 1],'dirname',DirName);
342         end
            %% HD  mit Scopedaten -gezoomt
344         if ANALYSED_SCOPE && activate_zoom
                plot_handle(axis1(axis1>-1),axis2(axis1>-1),hd_scope_u_diff(
                    axis1>-1),'mark',indexOfDoubleFrequency(axis1>-1),...
```

```
346             'figure name','HD-Scope_U_diff-Plot','plot title','HD of
                    Scope-U_{diff}-Signal',...
                'colorbar title','k in %','filename','HD_Scope_zoom_plot','
                    plot format',...
348             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
                axis2_label,'marker color',[1 1 1],'dirname',DirName);
350     end
        %% OHD-Lut mit HD5 von der Experimentalplattform
352     plot_handle(axis1,axis2,sub_hd_lut,'mark',
            indexOfDoubleFrequencyWithoutZoom,...
             'figure name','OHD-HD5-LUT(Demonstrator)-Plot','plot title','OHD
                calculated with HD5 (Demonstrator)',...
354          'colorbar title','OHD in %','filename','OHD_HD5_Demo_plot','plot
                format',...
             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
356          axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
             'double_click_plot_sig', u_diff);
358     %% OHD-Lut mit HD5 von der Experimentalplattform - gezoomt
        if activate_zoom
360     plot_handle(axis1(axis1 >-1),axis2(axis1 >-1),sub_hd_lut(axis1 >-1),'
            mark',indexOfDoubleFrequency(axis1 >-1),...
             'figure name','OHD-HD5-LUT(Demonstrator)-Plot','plot title','OHD
                calculated with HD5 (Demonstrator)',...
362          'colorbar title','OHD in %','filename','OHD_HD5_Demo_zoom_plot','
                plot format',...
             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
364          axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
             'double_click_plot_sig', u_diff);
366     end
         %% OHD-Lut mit HDI von der mti Abtastwerten der
             Experimentalplattform
368     plot_handle(axis1,axis2,o13hdi,'mark',
            indexOfDoubleFrequencyWithoutZoom,...
             'figure name','OHD-HDI','plot title','OHD calculated with HDI (
                Demonstrator)',...
370          'colorbar title','OHD-HDI in %','filename','OHD_HDI_plot','plot
                format',...
             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
372          axis2_label,'marker color',[1 1 1],'dirname',DirName);
         %% OHD-Lut mit HDI von der mti Abtastwerten der
             Experimentalplattform - gezoomt
374     if activate_zoom
        plot_handle(axis1(axis1 >-1),axis2(axis1 >-1),o13hdi(axis1 >-1),'mark
            ',indexOfDoubleFrequency(axis1 >-1),...
```

```
376         'figure name','OHD-HDI','plot title','OHD calculated with HDI (
                Demonstrator)',...
            'colorbar title','OHD-HDI in %','filename','OHD_HDI_plot_zoom','
                plot format',...
378         'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
            axis2_label,'marker color',[1 1 1],'dirname',DirName);
380     end

382     %% Upeak Plot_Log
        plot_handle(axis1,axis2,u_p_real_dB,...
384         'figure name','U-Peak-dB-Plot','plot title','U_{pp}',...
            'colorbar title','dB','filename','U-Peak_dB_plot','plot format'
                ,...
386         'A4L','caxis',[],'y label',axis1_label,'x label',...
            axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
388         'double_click_plot_sig', u_diff);
        %% Gain Plot
390     plot_handle(axis1,axis2,gaindb,'mark',indexOfDoubleFrequency,...
            'figure name','Gain-Plot','plot title','Gain',...
392         'colorbar title','dB','filename','Gain_plot_db','plot format',...
            'A4L','caxis',[],'y label',axis1_label,'x label',...
394         axis2_label,'marker color',[1 1 1],'dirname',DirName);
        %% Offset Plot
396     plot_handle(axis1,axis2,2.5/2^12*offset,'mark',
            indexOfDoubleFrequencyWithoutZoom,...
            'figure name','Offset-Plot','plot title','Offset',...
398         'colorbar title','U in V','filename','Offset_plot','plot format'
                ,...
            'A4L','caxis',[],'y label',axis1_label,'x label',...
400         axis2_label,'marker color',[1 1 1],'dirname',DirName);

402     %% Offset Plot - gezoomt
        if activate_zoom
404         plot_handle(axis1(axis1 >-1),axis2(axis1 >-1),2.5/2^12*offset(
                axis1 >-1),'mark',indexOfDoubleFrequency(axis1 >-1),...
                'figure name','Offset-zoom-Plot','plot title','Offset-zoom'
                    ,...
406             'colorbar title','U in V','filename','Offset_plot_zoom','plot
                    format',...
                'A4L','caxis',[],'y label',axis1_label,'x label',...
408             axis2_label,'marker color',[1 1 1],'dirname',DirName);
        end
410
        %% Discard Counter Plot, gezoomt
```

```
412     if isDemoAFG && activate_zoom
        plot_handle(axis1(z > -1),axis2(z > -1),discard_counter(z > -1),'mark
            ',indexOfDoubleFrequencyAFG(z > -1),...
414             'figure name','DSC-Plot','plot title','Discard Counter',...
                'colorbar title','Anzahl','filename','DSC_Demo_plot','plot
                    format',...
416             'A4L','caxis',[],'y label',axis1_label,'x label',...
                axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
418             'double_click_plot_sig', u_diffAFG);
        end
420
         %% HD Plot mit HDI und LUT, von Experimentalplattform, gezoomt
422      if isDemoAFG && activate_zoom
             plot_handle(axis1(z > -1),axis2(z > -1),harm_dist_lutAFG(z > -1)
                 ,'mark',indexOfDoubleFrequencyAFG(z > -1),...
424             'figure name','HD-HDI-LUT(Demonstrator)-Plot','plot title','
                    HD calculated with HDI(Demonstrator)',...
                'colorbar title','OHD in %','filename','
                    HD_HDI_LUT_Demo_zoom_plot','plot format',...
426             'A4L','caxis',[0 35],'y label',axis1_label,'x label',...
                axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
428             'double_click_plot_sig', u_diff);
         end
430
        %% OHD Plot mit HDI und LUT, von Experimentalplattform, gezoomt
432     if isDemoAFG && activate_zoom
            plot_handle(axis1(z > -1),axis2(z > -1),sub_hd_lutAFG(z > -1),'
                mark',indexOfDoubleFrequencyAFG(z > -1),...
434             'figure name','OHD-HDI-LUT-Zoom-Plot','plot title','OHD-LUT
                    calculated with HDI(Demonstrator)',...
                'colorbar title','OHD in %','filename','OHD_HDI_LUT_plot_zoom
                    ','plot format',...
436             'A4L','caxis',[0 45],'y label',axis1_label,'x label',...
                axis2_label,'marker color',[1 1 1],'dirname',DirName, ...
438             'double_click_plot_sig', u_diff);
        end
440
    end
442
    if function_call
444     close all;
    end
```

Analyse-Skript für die Oszilloskopdaten

```
function analyse_scope_data(dir_name)

    HARM = 5; %Amount of Harmonics
    RMP = 3; %Measurment belongs to which RMP?
    FUND_TOLERANCE = 25;
    HARM_TOLERANCE = 25;

    if ~exist('dir_name','var')
        dir_name = uigetdir(pwd,'Choose directory with data-files');
    end

    data_dir=textscan(dir_name, '%s', 'delimiter', filesep);
    filename=[data_dir{1}{end} '_scope_analysed.rmp_3.mat' ];

    list_of_files = what(dir_name);
    files = regexp(list_of_files.mat, '.*_part[0-9]+.rmp_3.mat', 'match')
        ;
    files=[files{:}];
    files = sort(files);

    if( exist(fullfile(list_of_files.path, filename), 'file')~=0 )
        disp([filename ' already exists, skipping...']);
        return;
    end

    load( fullfile(list_of_files.path, files{1}), 'parameters');
    tooth_frequency= parameters.tooth_frequency;    %tooth-frequency of
        measurement

    fs = parameters.samplerate; %Samplerate

    %% ---- Einlesen der Messdaten
        ------------------------------------------------

    scope_analysed_buff = [];

    for i=1:length(files);          % Dateien durchlaufen
        scope_analysed = struct(    'x',            {},...
                                    'y',            {},...
                                    'phi',          {},...
                                    'mag_u_diff',   {},...
                                    'mag_u_hb1',    {},...
```

```
                                    'mag_u_hb2',     {},...
                                    'angle_u_diff', {},...
                                    'angle_u_u_hb1',{},...
                                    'angle_u_hb2',   {},...
                                    'freq_u_diff',   {},...
                                    'freq_u_hb1',    {},...
                                    'freq_u_hb2',    {} );

    disp(['loading ' files{i}]);
    load( fullfile(list_of_files.path, files{i}), 'scope'); % jeweilige Datei laden
    for j=1:length(scope)
        res_u_diff = NaN(3,length(0.5*(1:HARM*2)));
        res_u_hb1 = NaN(3,length(0.5*(1:HARM*2)));
        res_u_hb2 = NaN(3,length(0.5*(1:HARM*2)));

        try
        res_u_diff = harmonics(scope(j).u_diff,fs,0.5*(1:HARM*2),'freq',tooth_frequency,...
            'fundamental_tolerance',FUND_TOLERANCE,...
            'tolerance',HARM_TOLERANCE);
        catch e1
            warning(['while executing harmonics on u_diff @ ' num2str(j)]);
        end

        try
        res_u_hb1 = harmonics(scope(j).u_hb1,fs,0.5*(1:HARM*2),'freq',tooth_frequency,...
            'fundamental_tolerance',FUND_TOLERANCE,...
            'tolerance',HARM_TOLERANCE);
        catch e2
            warning(['while executing harmonics on u_hb1 @ ' num2str(j)]);
        end

        try
        res_u_hb2 = harmonics(scope(j).u_hb2,fs,0.5*(1:HARM*2),'freq',tooth_frequency,...
            'fundamental_tolerance',FUND_TOLERANCE,...
            'tolerance',HARM_TOLERANCE);
        catch e3
            warning(['while executing harmonics on u_hb2 @ ' num2str(j)]);
```

```
77              end

79              scope_analysed(j).x = scope(j).x;
                scope_analysed(j).y = scope(j).y;
81              if(RMP == 3)
                    scope_analysed(j).z = scope(j).z;
83                  scope_analysed(j).phi_x = scope(j).phi_x;
                    scope_analysed(j).phi_y = scope(j).phi_y;
85                  scope_analysed(j).phi_z = scope(j).phi_z;
                else
87                  scope_analysed(j).phi = scope(j).phi;
                end
89              scope_analysed(j).mag_u_diff = res_u_diff(1,:);
                scope_analysed(j).mag_u_hb1 = res_u_hb1(1,:);
91              scope_analysed(j).mag_u_hb2 = res_u_hb2(1,:);
                scope_analysed(j).angle_u_diff = res_u_diff(2,:);
93              scope_analysed(j).angle_u_hb1 = res_u_hb1(2,:);
                scope_analysed(j).angle_u_hb2 = res_u_hb2(2,:);
95              scope_analysed(j).freq_u_diff = res_u_diff(3,:);
                scope_analysed(j).freq_u_hb1 = res_u_hb1(3,:);
97              scope_analysed(j).freq_u_hb2 = res_u_hb2(3,:);
            end
99
            if ~isempty(scope_analysed)
101             scope_analysed_buff = [scope_analysed_buff scope_analysed];
            end
103     end

105     scope_analysed = scope_analysed_buff;

107     disp(['saving ' filename]);
        save( fullfile(list_of_files.path, filename), 'scope_analysed');
109
end
```

Skript zur Berechnung von Harmonischen

```
%HARMONICS Harmonics in the signal val
2 %
% val:      signal
4 % fs:       sample frequency in Hz
% orders:   vector which include the orders to be calculated
```

```
%
% optional parameters:
%   freq:       fundamental frequency
%   tolerance:  search tolerance for frequencies in Hz
%   debug:      enable debug output
%
function [res] = harmonics(val, fs, orders, varargin)

debug=false;
fundamental_freq=0;

n_fft=length(val);
freq_res=fs/n_fft;

% tolerance range for searching harmonics
harm_tol=round(1/freq_res);
fund_tol=5;

if mod(size(varargin,2), 2)>0
    error('invalid number of arguments');
end

for i=1:2:size(varargin,2)
    switch lower(varargin{i})
        case 'freq'
            fundamental_freq = varargin{i+1};
        case 'debug'
            debug = (varargin{i+1})>0;
        case 'tolerance'
            % convert frequency to sample value
            harm_tol=round(varargin{i+1}/freq_res);
        case 'fundamental_tolerance'
            % convert frequency to sample value
            fund_tol=round(varargin{i+1});
        otherwise
            error('invalid argument');
    end
end

magnitudes=zeros(1, length(orders));
phases=zeros(1, length(orders));
frequencies=zeros(1, length(orders));

val_spec=fft(val)./n_fft;
```

```
val_mag=abs(val_spec);
val_phase=angle(val_spec);
%val_f=fs/2*linspace(0,1,n_fft/2); % -->

% skip dc
val_f=freq_res*(1:1:(n_fft/2)-1);
val_mag_half=val_mag(2:n_fft/2).*2;
val_phase_half=val_phase(2:n_fft/2);

if fundamental_freq > 0
    % find matching frequency
    [fundamental_center_min, fundamental_center_i]=min(abs(val_f-
        fundamental_freq));
    if fundamental_center_min > fund_tol
       error('specified frequency not found');
    end
    [dummy, fundamental_i]=max(val_mag_half(fundamental_center_i-harm_tol
        :fundamental_center_i+harm_tol));
    fundamental_i=fundamental_i+fundamental_center_i-harm_tol-1;

    if (val_mag_half(fundamental_i)/max(val_mag_half)*100) < 5
        error('your specified frequency seems to be wrong');
    end
else
    [dummy, fundamental_i]=max(val_mag_half);
end

if debug
    display(['val_f@fundamental_i(1): ' num2str(val_f(fundamental_i)) '
        Hz']);
    display(['val_mag_half@fundamental_i(1): ' num2str(val_mag_half(
        fundamental_i)) ' ']);
    display(['val_phase_half@fundamental_i(1): ' num2str(val_phase_half(
        fundamental_i)) ' rad']);
end

for i=1:length(orders)
    % calculate the approx. position
    order_center_i=round(fundamental_i*orders(i));

    % find the max in a spec. range
    [dummy, order_i]=max(val_mag_half(order_center_i-harm_tol:
        order_center_i+harm_tol));
    % calculate real index in the whole vector
```

```
88      order_i=order_i+order_center_i-harm_tol-1;

90      magnitudes(i)=val_mag_half(order_i);
        phases(i)=val_phase_half(order_i);
92      frequencies(i)=val_f(order_i);
    end
94
    if debug
96      plot(val_f(1:3000),val_mag_half(1:3000));
    end
98
    res=[magnitudes; ...
100      phases;...
         frequencies];
102
    end
```

Skript zur Messreihen wiederhlung mittels Funktionsgenerator

```
1   function signal_to_AFG(DirName, z_max)
    tooth_count = 43;   % Anzahl der Zähne am Encoderra
3   MAX_SAMPLE = 2^16-14; %Maximale Anzahl Samples, die der Speicher des
        Funktionsgenerator aufnimmt
    demoAFG=[];
5   fail = false;
    list_of_files    = what(DirName);
7   list_of_files.mat = sort(list_of_files.mat);
    files = regexp(list_of_files.mat, '.*_part[0-9]+.rmp_3.mat', 'match');
9   files=[files{:}];
    files = sort(files);
11  dummy = struct('u_diff', NaN, ...
                   'u_br1', NaN, ...
13                 'u_br2', NaN, ...
                   'frequency', NaN, ...
15                 'mag', NaN, ...
                   'mag_v', NaN, ...
17                 'mag_db', NaN, ...
                   'comp', NaN, ...
19                 'hd_lut', NaN, ...
                   'sub_hd_lut', NaN,...
21                 'hd_abs', NaN, ...
                   'angle', NaN, ...
```

```
23                'gain',    NaN, ...
                  'u_pp',    NaN,...
25                'discard_counter', NaN, ...
                  'offset',NaN ...
27                                  );
if ispc
29      filesep_plus = '\\';
else
31      filesep_plus = filesep;
end
33  data_dir=textscan(DirName, '%s', 'delimiter', filesep_plus);
      filename=[data_dir{1}{end} '_demo_from_AFG.rmp_3.mat' ];
35  %% ---- Einlesen der Messdaten
    ------------------------------------------
    k = 1;
37  load(fullfile(list_of_files.path, files{1}), 'parameters');
    load(fullfile(list_of_files.path, filename),'demoAFG');
39
      tooth_frequency = parameters.tooth_frequency;
41    samplerate = parameters.samplerate;


43
    h = waitbar(0,'Please wait..., loading files');
45  for i=1:length(files);        % Dateien durchlaufen
      waitbar(i / length(files));
47    load(fullfile(list_of_files.path,files{i}), 'demo');            %
        jeweilige Datei laden              % einzelne Strukturen auslesen


49
      for j=1:length(demo)
51      x(k)              = demo(j).x;
        y(k)              = demo(j).y;
53      z(k)          = demo(j).z;
        phi_x(k)      = demo(j).phi_x;
55      phi_y(k)      = demo(j).phi_y;
        phi_z(k)      = demo(j).phi_z;
57      freq(k)       = demo(j).freq;
        gain(k)           = demo(j).gain;
59      offset(k)         = demo(j).offset;
        k=k+1;
61    end
      clear('demo');                          % cleanup workspace
63  end
    close(h);
```

```
%% Connect to Regelplatine
interfaceObj_demo_slave_ctrl = instrfind('Name', 'Serial-COM4');

if isempty(interfaceObj_demo_slave_ctrl)
    interfaceObj_demo_slave_ctrl = serial('COM4');
else
    fclose(interfaceObj_demo_slave_ctrl);
    interfaceObj_demo_slave_ctrl = interfaceObj_demo_slave_ctrl(1);
end

deviceObj_demo_slave_ctrl=icdevice('demo_slave_ctrl.mdd',
   interfaceObj_demo_slave_ctrl);
connect(deviceObj_demo_slave_ctrl);
%%
encoder_period_approx = round(samplerate/(tooth_frequency/tooth_count));
search_range = round(encoder_period_approx / (tooth_count*4));
frequency = tooth_frequency/tooth_count;
signal = get_scope_data('z',z(length(demoAFG)+1),'phi_y',phi_y(length(
   demoAFG)+1)...
    ,'x',x(length(demoAFG)+1),'dirname',DirName,'u_hb1','dummy','u_hb2');
set(deviceObj_demo_slave_ctrl.Control,'offset',2000);
h = waitbar(0,'Processing measurement');

    for i = length(demoAFG)+1 : k - 1
        waitbar(i/k);
        if abs(z(i)) < abs(z_max)  && (length(demoAFG) < i || isnan(
           demoAFG(i).gain))
            %Signale auf eine Radumdrehung begrenzen
            if signal.u_hb1(1) >= signal.u_hb1(encoder_period_approx)
                [useless period_index] = min(abs(signal.u_hb1(1) - signal
                   .u_hb1 ...
                    (encoder_period_approx - search_range:
                       encoder_period_approx - (round(search_range/10))))
                    );
                period_index = encoder_period_approx - search_range +
                   period_index;
            else
                [useless period_index] = min(abs(signal.u_hb1(1) - signal
                   .u_hb1...
                    (encoder_period_approx - search_range :
                       encoder_period_approx )));
                period_index = encoder_period_approx - search_range +
                   period_index;
            end
```

```
99          %geringst mögliche Dezimierung berechnen
            [useless deci_val] = max(sign(MAX_SAMPLE - period_index ./
                [1:50]));
101         %%Dezimieren
            hb1 = decimate(signal.u_hb1(1:period_index),deci_val);
103         hb2 = decimate(signal.u_hb2(1:period_index),deci_val);
            %Signal auf den Funktionsgenerator geben
105         signal_to_function_generator(hb1,hb2,frequency, true);
            %Offset und gain regelung
107         for g = 6:-1: 0
                set(deviceObj_demo_slave_ctrl.Control,'gain',g);
109             pause(3);
                %%
111             while(1)
                    try
113                     var = get_measurement;
                    catch
115                     fail = true;
                        break;
117                 end
                    os = mean(var.u_diff) - 1.25;
119                 if abs(os) < 0.01
                        break;
121                 end
                    os_actual = get(deviceObj_demo_slave_ctrl.Control,'
                        offset');
123                 set(deviceObj_demo_slave_ctrl.Control,'offset',
                        os_actual + os *100);
                end
125             %%
                if fail
127                 break;
                elseif max(var.u_diff) < 2.3 && min(var.u_diff) < 0.3
129                     break;
                end
131         end
            %%
133         if i+1 < k
                signal = get_scope_data('z',z(i+1),'phi_y',phi_y(i+1),'x'
                    ,x(i+1),...
135                 'dirname',DirName,'u_diff','dummy','u_hb1','dummy','
                        u_hb2');
            end
137         if fail
```

```
                var = dummy;
            else
                fail = false;
                try
                    var = get_measurement();
                catch
                    var = dummy;
                end
            end
            var.gain = 2^get(deviceObj_demo_slave_ctrl.Control,'gain');
            var.offset =get(deviceObj_demo_slave_ctrl.Control,'offset');
            if length(demoAFG) < i
                demoAFG = [demoAFG var];
            else
                demoAFG(i) = var;
            end
        else
            if length(demoAFG) < i
                demoAFG = [demoAFG dummy];
            end
            if abs(z(i+1)) < abs(z_max) && i+1 < k
                signal = get_scope_data('z',z(i+1),'phi_y',phi_y(i+1),'x'
                    ,x(i+1),'dirname',...
                    DirName,'u_diff','dummy','u_hb1','dummy','u_hb2');
            end
        end
         save( fullfile(list_of_files.path, filename), 'demoAFG');
    end
 close(h);
end
```

Skript zur Übertragung von Halbbrückensignalen auf den Funktionsgenerator

```
function signal_to_function_generator( hb1, hb2,freq ,full_signal,
    varargin)
if full_signal == false
   periods = varargin{1};
end
%SIGNAL_TO_FUNCTION_GENERATOR Summary of this function goes here
SAMPLERATE = 25e6;
TOOTH_COUNT = 43;   % Anzahl der Zähne am Encoderra
```

```
MAX_SAMPLE = 2^16-14; %Maximale Anzahl Samples, die der Speicher des
    Funktionsgenerator aufnimmt

      hb1 = hb1/25; %Verstärkung entfernen
      hb2 = hb2/25;
      mean1 = (max(hb1) + min(hb1))/2;
      mean2 = (max(hb2) + min(hb2))/2;
      off1 = min(hb1);
      off2 = min(hb2);
      hb1 = hb1-off1;
      hb2 = hb2-off2;
      norm1 = max(hb1);
      norm2 = max(hb2);
      hb1 =(hb1/norm1)*16383;
      hb2 =(hb2/norm2)*16383;

if full_signal == false
    [scr scf]= zero_detection(hb1,'noise_buffer',1000);
    hb1 = hb1(scr(5):scr(6+periods-1)-1);
    hb2 = hb2(scr(5):scr(6+periods-1)-1);
    if length(hb1) > MAX_SAMPLE
        [useless deci_val] = max(sign(MAX_SAMPLE - length(hb1) ./ [1:50])
            );
        hb1 = decimate(hb1,deci_val);
        hb2 = decimate(hb2,deci_val);
    end
    figure('name','U_HB1');
    plot(hb1);
    figure('name','U_HB2');
    plot(hb2)
end

%% ---- Instrument verbinden
    ------------------------------------------
disp('Instrument verbinden...');
% Create a VISA-USB object.
if( ispc() == 1 )
    interfaceObj = instrfind('tek', 'TCPIP::192.168.0.10::INSTR');
else
    interfaceObj = instrfind('ni', 'TCPIP::192.168.0.10::INSTR');
end

% Create the VISA-USB object if it does not exist
% otherwise use the object that was found.
```

```
if isempty(interfaceObj)
    if( ispc() == 1 )
        interfaceObj = visa('tek', 'TCPIP::192.168.0.10::INSTR');
        set(interfaceObj, 'ByteOrder', 'bigEndian');
    else
        % use NI-VISA interface driver and change to littleEndian for
            Linux
        interfaceObj = visa('ni', 'TCPIP::192.168.0.10::INSTR');
        set(interfaceObj, 'ByteOrder', 'littleEndian');
    end
    set(interfaceObj, 'OutputBufferSize', 128*1024);
    set(interfaceObj, 'InputBufferSize', 128*1024);
    set(interfaceObj, 'EOIMode', 'off');
    set(interfaceObj, 'EOSMode', 'write');
else
    fclose(interfaceObj);
    interfaceObj = interfaceObj(1);
end

% Create a device object.
deviceObj = icdevice('tek_afg3000.mdd', interfaceObj);

% Connect device object to hardware.
connect(deviceObj);

%% ---- Instrument einstellen
    -------------------------------------------------
disp('Instrument einstellen...');

invoke(deviceObj.Data, 'datawrite', hb1);
% figure('name','hb1');
% plot(hb1);
invoke(deviceObj.Data, 'datacopy', 'EMEMory', 'USER1');
set(deviceObj.Waveform(1), 'Shape', 'USER1');
set(deviceObj.Frequency(1), 'Frequency', freq);
set(deviceObj.Voltage(1), 'Amplitude', norm1);    % 80mV -> G=25: 2V
set(deviceObj.Voltage(1), 'Unit', 'VPP');          % Voltage Peak-Peak
set(deviceObj.Voltage(1), 'Offset', 1.5);
set(deviceObj.Output(1), 'State', 'on');
%%
if ~isempty(hb2)
%     figure('name','hb2');
%     plot(hb2);
    invoke(deviceObj.Data, 'datawrite', hb2);
```

```
    invoke(deviceObj.Data, 'datacopy', 'EMEMory', 'USER2');
92  set(deviceObj.Waveform(2), 'Shape', 'USER2');
    set(deviceObj.Frequency(2), 'Frequency', freq);
94  set(deviceObj.Voltage(2), 'Amplitude', norm2);    % 80mV -> G=25: 2V
    set(deviceObj.Voltage(2), 'Unit', 'VPP');          % Voltage Peak-Peak
96  set(deviceObj.Voltage(2), 'Offset', 1.5);
    set(deviceObj.Output(2), 'State', 'on');
98  end
    %% ---- Instrument trennen
       ------------------------------------------------
100 disp('Instrument trennen...');
    % Disconnect device object from hardware.
102 disconnect(deviceObj);

104 % Delete objects.
    delete([deviceObj interfaceObj]);
106
    disp('---- End    Arb_Wave_Gen ----');
108
    end
```

Skript zur Berechnung des HDI

```
1   function [k ohd] = hdi(sig)
        xgl = 0;
3       l_ges = 0;
        lut_pos = 0;
5       IDX_PI_H  = 16;
        real = [0 0 0];
7       imag = [0 0 0];
        l = [0 0 0];
9       N=length(sig);

11      table_sin = [0     100     200     297     392 ...
                     483     569     650     724     792 ...
13                   851     903     946     980    1004 ...
                    1019    1024    1019    1004     980 ...
15                   946     903     851     792     724 ...
                     650     569     483     392     297 ...
17                   200     100];
        %% Samples werden verarbeitet
19      for i = 1: N
```

```
        xgl = xgl + sig(i);
        l_ges = l_ges + sig(i)^2;
        for j = 0 : 2
            lut_pos_harmonic = lut_pos*(j+1);
            prod = sig(i) * table_sin(1+bitand((lut_pos_harmonic +
                IDX_PI_H) ,hex2dec('1F')));
            if bitand(lut_pos_harmonic + IDX_PI_H,hex2dec('20'))
                real(j +1) = real(j +1) + prod;
            else
                real(j +1) = real(j +1) -prod;
            end
            prod = sig(i) * table_sin(1+bitand((lut_pos_harmonic) ,
                hex2dec('1F')));
            if bitand(lut_pos_harmonic, hex2dec('20'))
                imag(j +1) = imag(j +1) + prod;
            else
                imag(j +1) = imag(j +1) -prod;
            end
        end
        lut_pos = lut_pos +1;
    end
    %% Berechnung des Klirrfaktors
    xgl = xgl / N;
    l_ges = l_ges / N - xgl^2;

    real = fix(real ./(4096));
    imag = fix(imag ./(4096));
    lquadrat = fix((real.^2 + imag.^2)./(N*2));
    P_ob = l_ges - lquadrat(2);
    k = 100 * sqrt(P_ob / l_ges);
    %[row col i] = decode_lut_hdi(bitshift(fix(P_ob),14) / l_ges)
    ohd = 100 * sqrt ( (lquadrat(1) + lquadrat(3))/l_ges);
end
```

Skript zur Darstellung der Messergebnisse bei Verkippung einer Achse

A.2.2. Script zur Darstellen der Messergebnisse bei Verfahren zweier Linearachsen

```
%RADAR_PLOT radar_plot plots a radar-plot
%
```

```
%   distance_values: array has to include the distance of every measured
    point
%   angle_values: array has to include the angle of every measured point
%   val: array of measured values related to distance_values and
    angle_values
%
%
%   optional parameters
%       mark: array of type logical, marks related points in the plot
%       mark_color: color of the marks
%       figure_name: title in the headbar of the figure window
%       plot_title: title of the plot
%       colorbar_title: think about it
%       save_file_name: filename of the pdf-file (without ending ".pdf")
%       scale_colorbar: values of the 2 fields array lineary map to the
%                                                              colormap
%       y_label: y-axis label
%       x_label: x-axis label
%       dir_name: puts the directory of the measurment into the plot
    title
%
%   return parameters
%       [surface_handle title_handle colorbar_handle]
%
%

function [surface_handle title_handle colorbar_handle] = radar_plot(
    distance_values,angle_values,val,varargin)
    distance_values = abs(distance_values);
    mark = zeros(size(distance_values));
    figure_name = '';
    plot_title = [];
    colorbar_title = '';
    save_file_name = [];
    plot_format = 'A4L';
    mark_color = [0 0 0];
    scale_colorbar = [];
    y_label = 'distance in mm';
    x_label = 'angle in degree';
    dir_name = [];
    double_click_plot_sig=[];
    shift_click_plot_sig=[];
    ctrl_click_plot_sig=[];

```

```
    if mod(size(varargin,2), 2)>0
44      error('invalid number of arguments');
    end
46
    for i = 1 : 2 :size(varargin,2)
48      switch varargin{i}
            case 'mark'
50              mark = varargin{i+1};
            case 'figure name'
52              figure_name = varargin{i+1};
            case 'plot title'
54              plot_title = varargin{i+1};
            case 'colorbar title'
56              colorbar_title = varargin{i+1};
            case 'filename'
58              save_file_name = varargin{i+1};
            case 'marker color'
60              mark_color = varargin{i+1};
            case 'plot format'
62              plot_format = varargin{i+1};
            case 'caxis'
64              scale_colorbar = varargin{i+1};
            case 'y label'
66              y_label = varargin{i+1};
            case 'x label'
68              x_label = varargin{i+1};
            case 'dirname'
70              dir_name = varargin{i+1};
            case 'double_click_plot_sig'
72              double_click_plot_sig = varargin{i+1};
            case 'shift_click_plot_sig'
74              shift_click_plot_sig = varargin{i+1};
            case 'ctrl_click_plot_sig'
76              ctrl_click_plot_sig = varargin{i+1};
            otherwise
78              error(['"' varargin{i} '"' ' is an invalid argument']);
        end
80  end
    %%
82  %pcolor plottet die letzten Reihe und Spalte nicht mit, deswegen wird
    %jeweils eine angehaengt
84  distance = unique(distance_values); %Distanzen einzeln, aufsteigend
        speichern
    angles = unique(angle_values); %Winkel einzeln, aufsteigend speichern
```

```
86      distance_stepsize = abs(distance(2) - distance(1));%Schrittweite
            ermitteln
        distance = [distance distance(end)+ distance_stepsize]; % append one
            value
88      angle_stepsize = abs(angles(2) - angles(1));%Schrittweite ermitteln
        angles = [angles (angles(end)+angle_stepsize)];% append one value
90
        % Name der aktuellen Messung extrahieren
92      if( ispc() == 1 )
            % windows
94          data_dir=textscan(dir_name, '%s', 'delimiter', '\\');
        else
96          % unix
            data_dir=textscan(dir_name, '%s', 'delimiter', filesep);
98      end
        meas_name=data_dir{1}{end};
100
        %Titel des Plots wird zusammengesetzt
102     dir_name_title = strrep(meas_name,'\','/');
        dir_name_title = strrep(dir_name_title,'_','\_');
104     plot_title = [plot_title ' over distance/angle,distance step size: '
            ...
            num2str(distance_stepsize) ' mm, angle step size: '...
106     num2str(angle_stepsize) ' deg\newline' dir_name_title];
        matrix(1:length(distance),1:length(angles)) = NaN; %Matrix
            initialisieren
108     mark_matrix = zeros(length(distance),length(angles));
        %Matritzen mit Messwerten fuellen
110     for i =1 : length(val)
            row = find(distance_values(i) == distance);
112         col = find(angle_values(i) == angles);
            matrix(row,col) = val(i);
114         mark_matrix(row,col) = mark(i);
        end
116 %%
        angles = angles - (angle_stepsize/2);
118     theta = -pi/2+(angles)*pi/180; %angle in radian measure
        X = (distance)'*cos(theta);
120     Y = (distance)'*sin(theta);

122     figure('name',[figure_name 'Radarplot (Messung in X-Richtung
            aufsteigend)']);
        surface_handle = pcolor(X,Y,matrix);                    % radarplot
```

```
124     mark_points(X,Y,mark_matrix,mark_color);                    % mark
            fields

126     set(gca, 'YTickLabel',sprintf('%3.1f|',abs(get(gca,'YTick'))));
        xtick = min(X(end,:)) : abs(min(X(end,:))-max(X(end,:)))/10:max(X(end
            ,:)); %10 marking on X-axis
128     set(gca,'Xtick',xtick);
        xticklabel = unique((acos(xtick./max(distance)))./pi * 180)-90; %
            calculate angle
130     xticklabel = fix(xticklabel);
        set(gca, 'XTickLabel', sprintf('%3.0f|', xticklabel));
132
        ylabel(y_label);
134     xlabel(x_label);

136     title_handle = title(plot_title);
        %set(title_handle,'interpreter','none');
138
        click_plot_sigs=struct( 'double', double_click_plot_sig, ...
140                             'shift', shift_click_plot_sig, ...
                                'ctrl', ctrl_click_plot_sig);
142     set(surface_handle, 'ButtonDownFcn', ...
            {@cb_radar_plot, distance_values, angle_values, click_plot_sigs});
144

146     g = jet(256);
        colormap(g);
148     if(~isempty(scale_colorbar))
            caxis(scale_colorbar);
150     end

152     colorbar_handle = colorbar();
        set(get(colorbar_handle,'Title'),'String',colorbar_title);
154     if(~isempty(save_file_name))
            export_fig(gcf,fullfile(dir_name,[meas_name '_' save_file_name '.
                pdf']), plot_format);
156     end
end
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM 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, 20. April 2011	
Ort, Datum	Unterschrift