HAW HAMBURG

BACHELORTHESIS
Moritz Herbstmeier

Magnetisches Sensor-Array zur Winkelerfassung an einer Welle – FEM-Simulation, Dimensionierung und Signalverarbeitung –

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Moritz Herbstmeier

Magnetisches Sensor-Array zur Winkelerfassung an einer Welle – FEM-Simulation, Dimensionierung und Signalverarbeitung –

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Lutz Leutelt

Eingereicht am: 26. Oktober 2021

Moritz Herbstmeier

Thema der Arbeit

Magnetisches Sensor-Array zur Winkelerfassung an einer Welle – FEM-Simulation, Dimensionierung und Signalverarbeitung –

Stichworte

Sensorik, Sensor-Array, Magnetfeld, TMR-Effekt, Finite-Elemente-Methode

Kurzzusammenfassung

In dieser Arbeit wird untersucht, wie die Winkelposition einer Welle mit einem Array magnetischer Sensoren ermittelt werden kann. Der System-Aufbau und die Methodik werden anhand von Tests erarbeitet. Es werden Magnetfeld-Simulationen durchgeführt. Die daraus resultierenden Daten werden für die Erstellung eines Signalverarbeitungs-Algorithmus verwendet, der aus Simulationsergebnissen einen Rotationswinkel berechnet.

Moritz Herbstmeier

Title of Thesis

Detection of shaft rotation angle using a magnetic sensor array – FEM simulation, scaling and signal processing –

Keywords

Sensors, Sensor array, Magnetic Field, TMR effect, Finite Element Method

Abstract

This thesis examines how the angular position of a shaft can be determined using an array of magnetic sensors. The system layout and the methodology are developed through testing. Magnetic field simulations are performed. The resulting data is used to create a signal processing algorithm that calculates an angle of rotation from simulation results.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

In Deutschland sollen laut Bundesregierung bis zum Jahr 2030 sieben bis zehn Millionen Kraftfahrzeuge mit Elektroantrieb zugelassen sein [1]. Dies sei ein wichtiger Schritt, um CO2-Emissionen zu verringern.

Für den Betrieb dieser Motoren ist es wichtig, den Rotationswinkel der Welle, die vom Motor angetrieben wird, stets zu wissen. Die Drehstrommaschinen die in den Antrieben zum Einsatz kommen können teilweise durch intelligente Regelung basierend auf dem Winkel effizienter Betrieben werden. Zusätzlich wird durch korrekte Detektion der Rotorposition vermieden, dass das System bei Überlastung beschädigt wird, wodurch die Sicherheit verbessert wird [2, Kapitel 7.3+11.2.1][17]. In dieser Arbeit soll untersucht werden, wie dies optimalerweise mit einem magnetischem Sensor-Array bewerkstelligt wird.

1.1 Ziel dieser Arbeit

Das Anwenden eines magnetischen Sensor-Arrays zur Winkelerfassung ist ein neuartiges Konzept, das viele Entwicklungsschritte umfasst. Ziel dieser Arbeit ist es, auf Basis von Simulationen zu untersuchen, wie ein System für dieses Konzept aufgebaut werden kann und wie anhand von Signalverarbeitung der Rotationswinkel aus Messdaten gewonnen werden kann. Im Folgenden sind die Entwicklungsschritte aufgelistet.

1. Theoretisches Design des Sensor-Arrays
2. Planung und Simulation für Winkelerfassung
3. Signalverarbeitung anhand von Simulationsdaten
4. Hardware-Aufbau des Sensor-Arrays
5. Aufbau von Welle und Array für Winkelmessung

6. Signalverarbeitung mit Messdaten

Es werden hier die Punkte 2. und 3. behandelt.

2 Grundlagen

Im Folgenden wird erläutert, wie die Messung der Winkelposition bewerkstelligt werden kann, wie dies anhand des Sensor-Arrays erledigt wird und wie Magnetfelder mit der Finite-Elemente-Methode numerisch berechnet werden.

2.1 Methoden der Winkelerfassung

Um den Rotationswinkel zu erfassen, gibt es verschiedene Methoden. Es wird hierbei zwischen der absoluten und der inkrementalen Messung unterschieden. Die absolute Erfassung kann zu jedem Zeitpunkt den aktuellen Winkel bestimmen, während für die inkrementale Messung eine Drehung der Welle notwendig ist [10, S.35].

Außerdem spielt die Art des Sensors eine große Rolle. Es werden typischerweise entweder induktive, magnetische oder optische Sensoren verwendet. Vorteile magnetischer Sensoren sind unter anderem, dass mechanische Toleranzen geringen Einfluss auf die Messungen haben und die Möglichkeit, sehr kompakte Bauformen einzusetzen [17, Kapitel 12.2.1].

Es wird hier eine absolute Messung anhand von magnetischen TMR-Sensoren (Tunnel Magnetoresistance) umgesetzt. Bei der TMR-Technologie liegen drei Schichten übereinander. Die zwei äußeren Schichten sind leitend. Eine davon ist in eine feste Richtung magnetisiert, während die Magnetisierungsrichtung der gegenüberliegenden Schicht variabel ist und vom Magnetfeld abhängt, das gemessen werden soll. Durch die Barriereschicht in der Mitte fließt ein Tunnelstrom, der größer wird, je mehr die Magnetisierungsrichtungen der leitenden Schichten übereinstimmen. Der Widerstand zwischen den leitenden Schichten ist abhängig von diesem Strom und gibt somit Auskunft über das externe Magnetfeld [16][18].

2.2 Mechanische Konzeption

Typischerweise werden Sensoren der Magneto Resistance Technik für diese Anwendung am Wellenende angebracht [17, Kapitel 12.2.1.3]. In dieser Arbeit wird das Sensor-Array stattdessen an der langen Seite der Welle eingesetzt. Dadurch können Anwendungen, bei denen die Enden der Welle blockiert sind, realisiert werden. Aufgrund der kleinen Bauform der Sensoren kann die Größe des Arrays für diese Arbeit frei gewählt werden. Die Information des aktuellen Rotationswinkels kann entweder durch Entfernen oder durch Auftragen von Metall an der Welle übermittelt werden. Dies geschieht, indem im Laufe einer Umdrehung diese Modifikationen an der Welle ihre Position so verändern, dass eine Aussage über den Winkel gemacht werden kann. Ein Auftragen von Metall kann anhand von Schweißen oder 3D-Druck geschehen. Das Entfernen von Metall kann durch Fräsen bewerkstelligt werden. Es ist zu erwarten, dass Letzteres weniger Aufwand in der Produktion in Anspruch nimmt. CNC-Fräsen kommen in der Anfertigung von Metallteilen häufig zum Einsatz und können die Modifikationen an der Welle effizient umsetzen.

2.3 Finite-Elemente-Methode (FEM)

Die Berechnung von Magnetfeldern kann auf verschiedene Weisen bewerkstelligt werden. Bei analytischen Methoden wird durch Integration der Maxwellschen Gleichungen die Lösung erlangt. Dies kann auch numerisch berechnet werden, wobei die Resultate stets Näherungen sind.

Die Finite-Elemente-Methode ist ein numerisches Berechnungsverfahren zur Analyse von Strukturen. Der Raum und die darin enthaltenen Objekte werden dabei diskretisiert, indem sie in eine endliche Anzahl an Elementen, wie zum Beispiel Dreiecken, unterteilt werden. Diese neu generierte Struktur ist das Mesh (dt. Gitternetz). Mit jedem der Elemente ergibt sich eine Zeile in einem Gleichungssystem, das numerisch gelöst wird. Im Fall von Magnetfeldern werden pro Element die magnetische Feldstärke und die magnetische Flussdichte berechnet. Ein feines Mesh mit vielen kleinen Elementen ist dementsprechend aufwändiger zu kalkulieren als ein Gröberes, liefert aber genauere Ergebnisse [19] [7, Anhang A].

3 Zweidimensionale Simulation

Es ist nicht offensichtlich, wie die Fräsungen an der Welle auszulegen sind, um klare und verlässliche Ergebnisse zu erzielen. Deswegen werden mehrere Optionen untersucht. Da die dreidimensionale Simulation viel Zeit in Anspruch nimmt (Tabelle 3.1), werden die Konstruktionsarten vorerst als 2D-Modelle simuliert. Dabei wird die Welle von der Seite betrachtet (Abb. 3.1). Anhand der Resultate wird die vielversprechendste Methode gewählt, bei der eine 3D-Simulation voraussichtlich aussagekräftige Ergebnisse liefert.

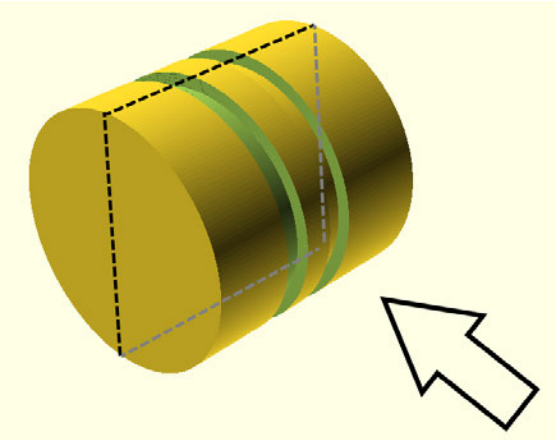


Abbildung 3.1: 3D-zu-2D Wandlung dargestellt. Die Welle wird in Pfeilrichtung betrachtet.

Simulationsart	Zeit pro Durchlauf	Zeit pro 15 Durchläufe
2D	10 s	2 min
3D	4 min	1 h

Tabelle 3.1: Vergleich des Zeitaufwands der Simulationsarten. Je mehr Schritte gemacht werden, desto besser lässt sich über den Einfluss der Fräsungen auf das Magnetfeld im Laufe einer Umdrehung urteilen, jedoch sind 15 Winkelpositionen für diese Tests ausreichend. Die Angaben sind ungefähre Werte, denn es soll nur der Unterschied deutlich gemacht werden.

Die Merkmale

- Anzahl der Fräsungen
- Verlauf der Fräsungen
- Breite der Fräsungen
- Tiefe der Fräsungen
- Form der Fräsungen (V-Form oder Rechteck)

werden betrachtet.

3.1 Kriterien für verlässliche Ergebnisse

Die wichtigen Kriterien für die Auswahl einer Konstruktionsart werden wie folgt gewählt:

1. Insensitivität gegenüber Ungenauigkeiten bei der Positionierung des Sensors
2. Möglichst effiziente und deutliche Informationsübermittlung anhand der Fräsungen

Ersteres lässt sich vorerst schwer überprüfen. Ein aufschlussreicher Test ist erst mit funktionaler Signalverarbeitung möglich. Es ist jedoch klar, dass die Nutzung absoluter Positionen nicht zu verlässlichen Resultaten führt. Stattdessen sollte eine Auswertung anhand von Abständen bzw. Frequenzen möglich sein.

Die Deutlichkeit der Informationsübermittlung wird anhand der Frage

Ist in den Simulationsergebnissen an jedem Punkt einer Rotation eindeutig erkennbar, was der entsprechende Winkel ist?

beurteilt. Der genaue Winkel lässt sich nur abschätzen, jedoch können die Eindeutigkeit und die Effizienz der Verläufe klar bewertet werden.

3.2 Verwendete Software

Die 2D-Simulation wird anhand von mehreren Matlab-Skripten, die die Software FEMM [9] verwenden, vorgenommen (Anhang A). Die Skripte wurden von T. Schüthe übernommen und angepasst [14]. Die Modelle werden textbasiert in Matlab erstellt.

FEMM wendet die Finite-Elemente-Methode an (siehe Kapitel 2.3), um verschiedene Probleme lösen zu können. Das gegebene Modell wird dabei in Dreiecke unterteilt, anhand derer die Berechnungen durchgeführt werden.

3.3 Test von Breite, Tiefe und Form einer Fräsung

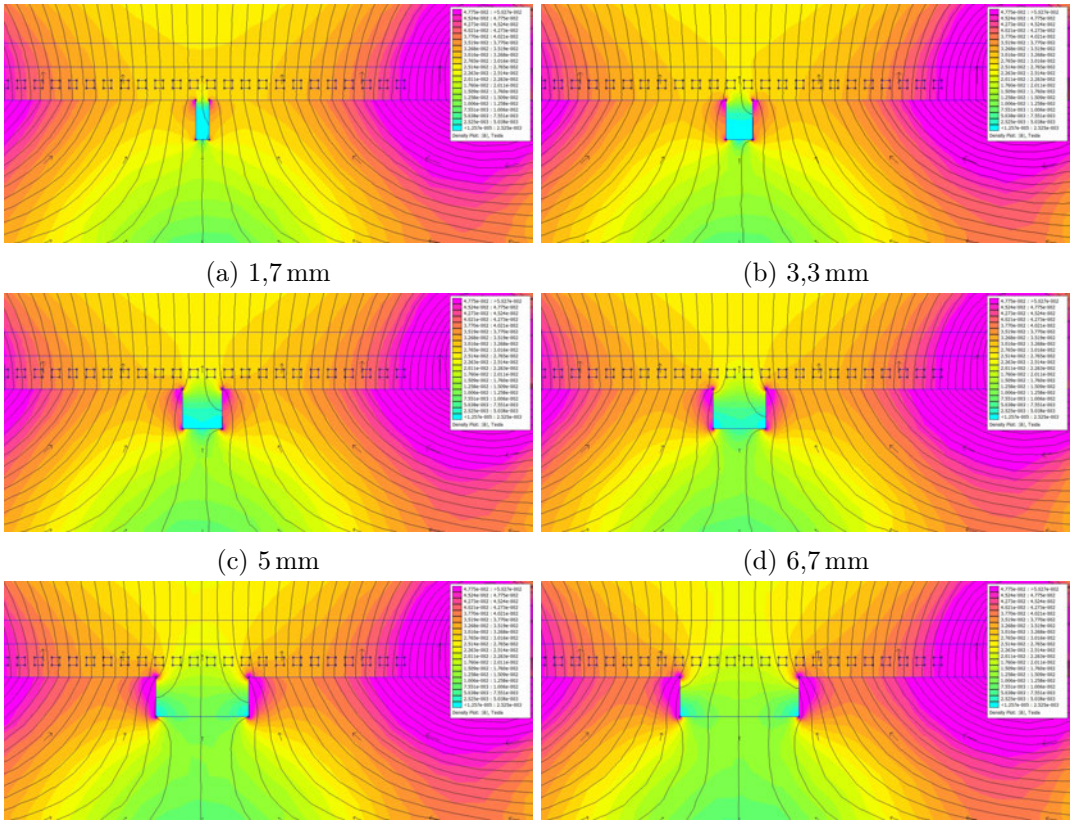
Es werden die Breite, Tiefe und Form einer Fräsung getrennt voneinander untersucht, um jeweils die optimale Einstellung zu ermitteln. Für die Tests wird ein Modell mit einer Fräsung verwendet, an dem der jeweils zu testende Parameter stufenweise verändert wird.

3.3.1 Breite

Aus Abbildung 3.2 wird deutlich, dass bei zu geringen Breiten auf 2 mm Höhe, der Höhe des Sensor-Arrays, kaum ein Unterschied zwischen einer Stelle mit Fräsung und einer Stelle ohne Fräsung erkennbar ist. Breitere Fräsungen sind im Magnetfeld klar zu sehen. Daher kann angenommen werden, dass eine Fräsung besser erkannt werden kann, je breiter sie ist.

3.3.2 Tiefe

Auch bei der Tiefe ist erkennbar, dass ein größerer Einschnitt deutlicher im Magnetfeld erkennbar ist (Abb. 3.3). Dieses Verhalten ist bei Tiefen bis ca. 7 mm zu sehen. Weiteres Vertiefen führt zu keinem signifikanten Unterschied. Aus den Ergebnissen lässt sich erschließen, dass eine Tiefe von midestens 3 mm gewählt werden sollte, um zu garantieren, dass deutliche Messungen möglich sind.



(a) 1,7 mm (b) 3,3 mm

(c) 5 mm (d) 6,7 mm

(e) 11,7 mm (f) 15 mm

Abbildung 3.2: Vergleich verschiedener Fräsungs-Breiten

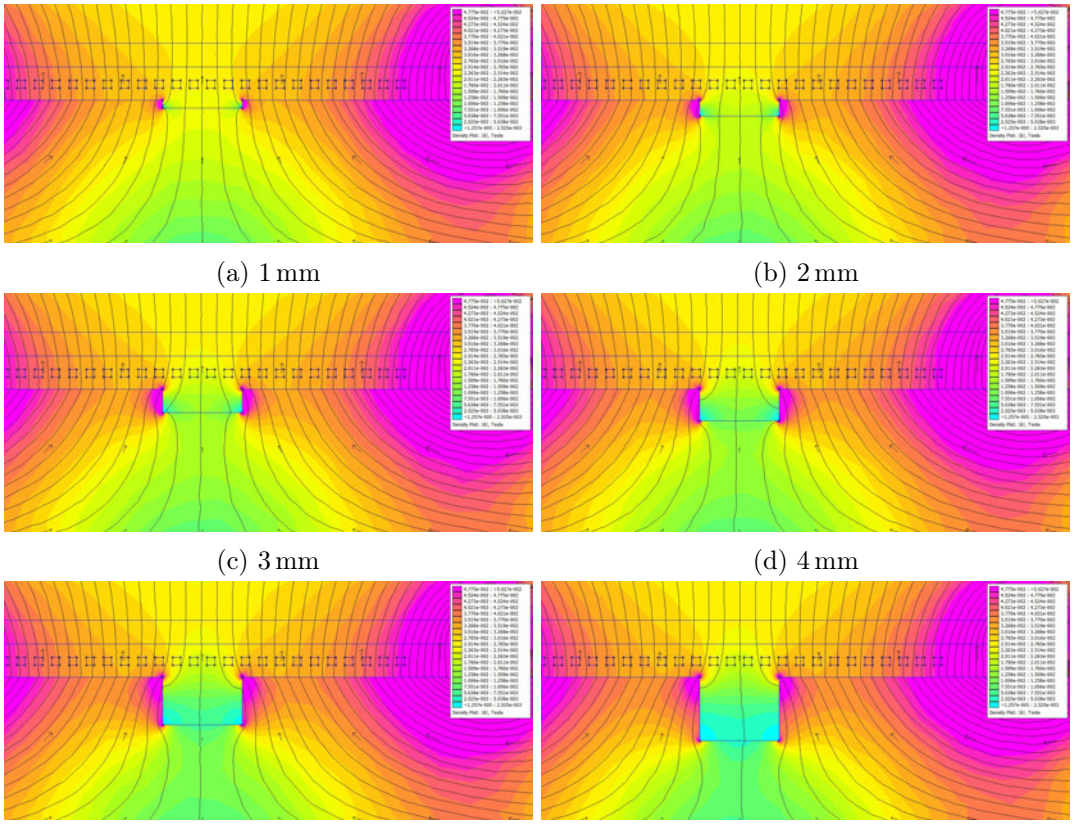
3.3.3 Form

Es werden Fräsungen mit Rechteck-Form und V-Form untersucht.

In Abbildung 3.4 ist zu sehen, dass die Einflüsse der zwei Fräsungen auf das Magnetfeld nahezu identisch sind.

3.4 Test von Anzahl und Verlauf der Fräsungen

Um die ideale Anzahl von Fräsungen und deren Verlauf zu ermitteln, werden im Folgenden Vermutungen in Bezug darauf aufgestellt und jeweils abgeschätzt, ob sich der Rotationswinkel daraus algorithmisch detektieren lässt.



(a) 1 mm (b) 2 mm

(c) 3 mm (d) 4 mm

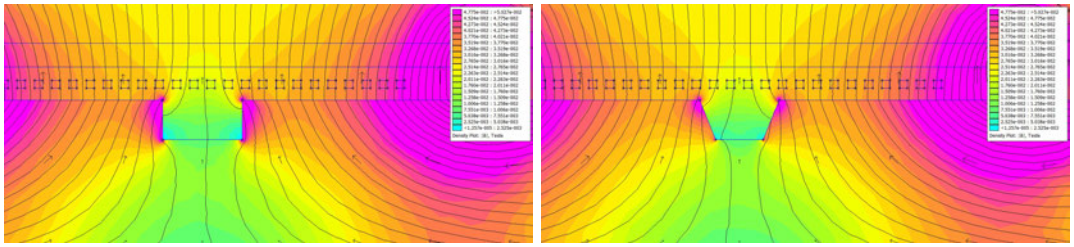
(e) 6 mm (f) 8 mm

Abbildung 3.3: Vergleich von Tiefen einer Fräsung

3.4.1 Rampenförmiger Verlauf als Informationsträger

Eine anschauliche Möglichkeit, den Rotationswinkel auf der Welle darzustellen, ist die Verwendung einer rampenförmigen Fräsung. Das bedeutet, dass die Position der Fräsung sich im Laufe einer Umdrehung linear verändert.

In Abbildung 3.5 ist der Verlauf in Rampenform mit einer Fräsung zu sehen. Die Darstellung zeigt die Oberfläche der Welle im relevanten Bereich über eine gesamte Rotation. Die roten Linien entsprechen der Größe des Sensor-Arrays. Die Fräsung befindet sich innerhalb des mittleren Bereichs, um immer unterhalb des Arrays zu verlaufen. Diese Variante ist eindeutig und effizient, jedoch anfällig für Ungenauigkeiten, da die absolute Position verwendet werden muss.



(a) Rechteck-Form (b) V-Form

Abbildung 3.4: Vergleich von Formen einer Fräsung

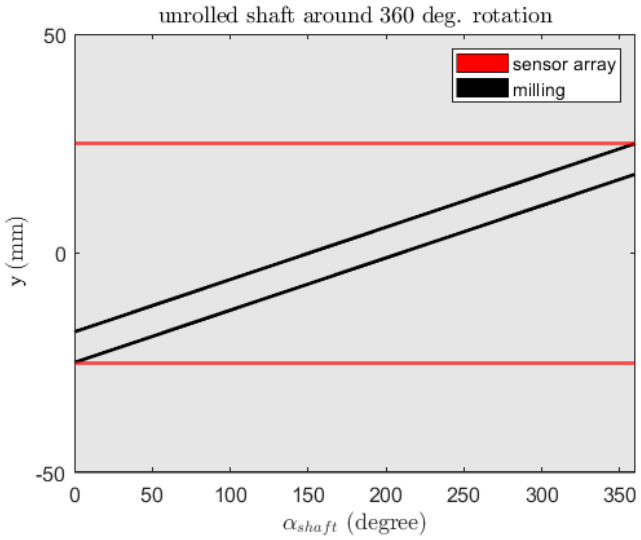


Abbildung 3.5: Rampenförmiger Verlauf mit einer Fräsung

Um auch bei ungenauer Positionierung des Sensors akkurate Ergebnisse zu erhalten, wird eine zweite Fräsung hinzugefügt. Die Position dieser Fräsung verändert sich nicht, sodass sie als Referenz dient (Abb. 3.6). Dadurch ist zu erwarten, dass anstelle der absoluten Position das Zusammenspiel der zwei Fräsungen betrachtet werden kann, zum Beispiel durch eine Fouriertransformation. Der Produktionsaufwand ist ungefähr verdoppelt, aber die Resultate sind in der Theorie eindeutig und verlässlich.

Es stellt sich die Frage, ob die Ergebnisse durch Hinzufügen einer dritten Fräsung weiter verbessert werden können. Um dies zu überprüfen, wird ein Layout entworfen, bei dem die dritte Fräsung auch als Referenz dient, sodass die Rampen-Fräsung zwischen zwei konstanten Einkerbungen verläuft (Abb. 3.7).

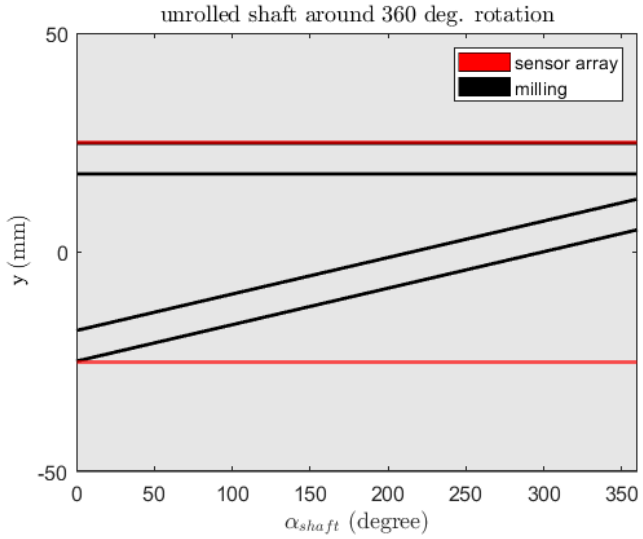


Abbildung 3.6: Verlauf in Rampenform mit Referenzfräsung

Es ist zu erwarten, dass auch diese Methode gute Resultate liefern kann, jedoch gibt es keinen klar erkennbaren Vorteil gegenüber der Variation mit zwei Fräsungen. Außerdem bleibt durch die dritte Fräsung weniger Platz für die Rampe, wodurch die Genauigkeit der Ergebnisse verringert werden könnte.

Generell ist unklar, wie verlässlich Konstruktionen mit Rampen als Informationsträger am Sprungpunkt das korrekte Ergebnis liefern. Der Sprungpunkt ist die Stelle, an der die Position der Rampe zwischen den beiden Extremwerten springt und ist in den Abbildungen der Verläufe bei 0° zu sehen.

3.4.2 Winkeldetektion durch Veränderung der Tiefe

Eine Detektion des Rotationswinkels anhand einer Veränderung der Tiefe ist effizient, da mehr Optionen der Informationsübermittlung ausgenutzt werden. Um zu überprüfen, ob anhand dieser Methode verlässliche Ergebnisse erzielt werden können, wird eine Konstruktion mit einer konstanten Fräsung, deren Tiefe linear mit dem Winkel ansteigt, verwendet. Es wird ein Plot erstellt, der die magnetische Feldstärke über der Fräsung darstellt.

In Abbildung 3.8 wird deutlich, dass eine Ermittlung des Rotationswinkels nur im Bereich von 0 mm bis ungefähr 2 mm möglich ist. Dort ist der Verlauf der magnetischen

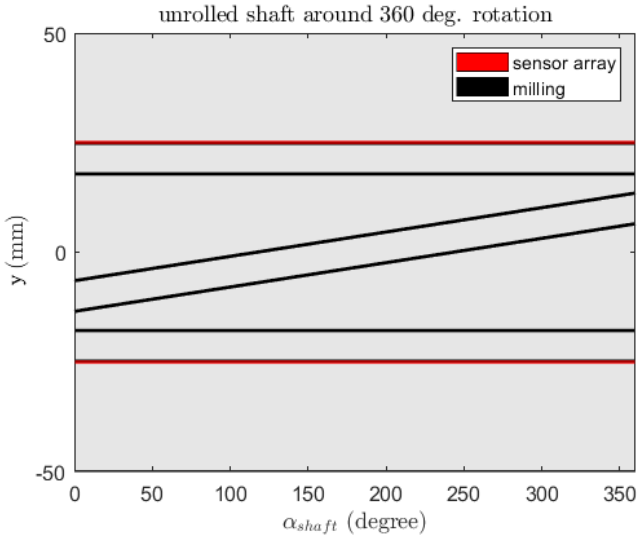


Abbildung 3.7: Verlauf mit Rampe und zwei Referenzfräsungen

Feldstärke nahezu linear. Durch die vorherigen Tests wurde festgestellt, dass eine Tiefe von mindestens 3 mm ideal ist, um die Fräsung deutlich im Magnetfeld erkennen zu können. Es ist dementsprechend fraglich, ob eine Veränderung der Tiefe eingesetzt werden sollte.

3.4.3 Sinusförmige Fräsungen

Im Vergleich zu rampenförmigen Verläufen haben sinusförmige Fräsungen die umgekehrten Vor- bzw. Nachteile. Die Sinusform durchquert den Nullpunkt mehrmals. Dadurch lässt sich ein Sprungpunkt vermeiden. Es bedeutet aber auch, dass jede Position doppelt vorkommt. Durch Verwendung weiterer Fräsungen kann die Eindeutigkeit jeder Position erzielt werden. So wird zum Beispiel in der Konstruktion in Abbildung 3.9 eine Fräsung in der Form einer Kosinuskurve hinzugefügt.

Es ist zu sehen, dass aufgrund der Sinusform auch die Abstände zwischen den zwei Fräsungen doppelt vorkommen. Die Ermittlung der Phase könnte stattdessen zu einer genauen Aussage über den Rotationswinkel führen.

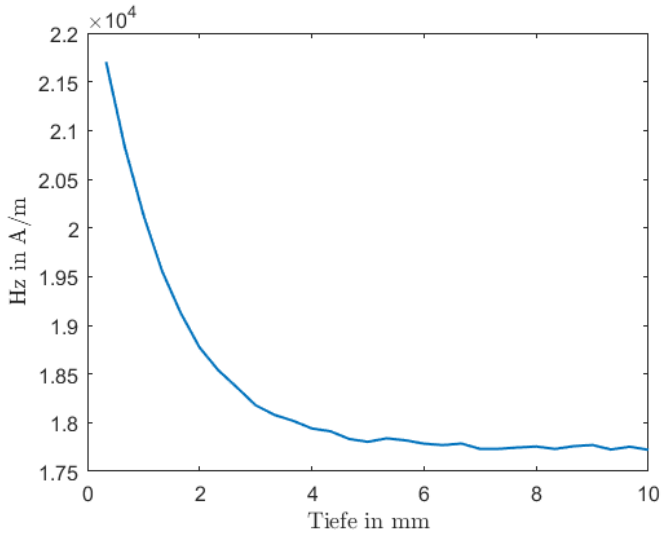


Abbildung 3.8: Plot der Feldstärke über Fräsung mit variabler Tiefe

3.5 Zusammenfassung

Es war zu ermitteln, wie Fräsungen auf einer Welle einzusetzen sind, um klare und verlässliche Ergebnisse zu erhalten. Dafür wurden mehrere Merkmale untersucht.

3.5.1 Breite, Tiefe und Form der Fräsungen

Die Erkenntnisse dieser Tests sind in Tabelle 3.2 zusammengefasst. Sie werden im Folgenden erläutert.

Die Breite der Fräsungen wird optimalerweise so groß gewählt, dass der Verlauf weiterhin ohne Überschneidungen möglich ist. Dann ist die Ausprägung im Magnetfeld am deutlichsten, wodurch möglichst viele Sensoren des Arrays diese erkennen und somit eine genaue Aussage über die Position gemacht werden kann. Es wurde deutlich, dass eine Mindestbreite von 5 mm sinnvoll ist.

Bei der Tiefe der Fräsungen gilt es, die Balance zwischen möglichst deutlichen Ausprägungen im Magnetfeld und einer vielseitig einsetzbaren, effizienten Bauform zu finden. Letzteres wird durch geringe Tiefen erzielt, da so auch bei dünnen Wellen diese Methode der Winkelerfassung verwendet werden kann und die Produktion dabei schneller und mit weniger Verschleiß verläuft. Für die klare Erkennung ist dagegen eine tiefere Fräsung

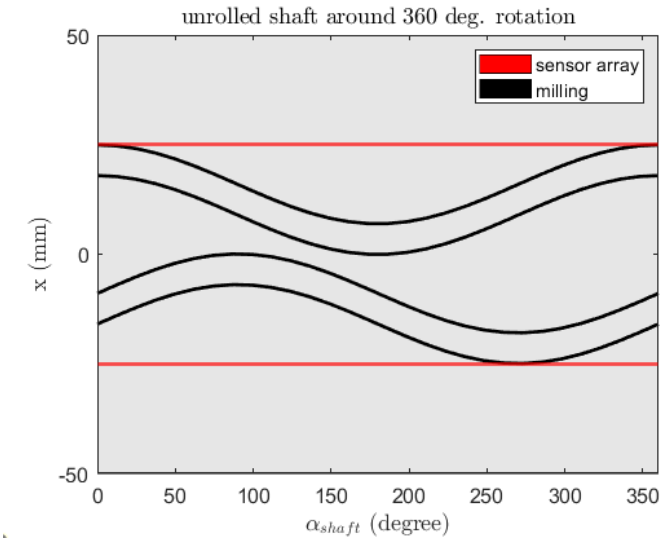


Abbildung 3.9: Konstruktion mit sinusförmigen Fräsungen

von Vorteil. Es hat sich gezeigt, dass der Bereich von Tiefen zwischen 3 mm und 7 mm effizient gute Ergebnisse liefert.

Es ist kein signifikanter Unterschied zwischen der Funktionalität einer Rechteck-Form und der einer V-Form erkennbar. Dementsprechend ist es im Rahmen dieser Arbeit nicht relevant, welche Form verwendet wird. Bei einer Umsetzung in Hardware kann jedoch die V-Form in Betracht gezogen werden, da weniger Material von der Welle entfernt werden muss und somit Produktionszeit und Verschleiß verringert werden können.

Merkmal	Auslegung
Breite	Mindestens 5 mm
Tiefe	3 mm - 7 mm
Form	V-Form oder Rechteckform

Tabelle 3.2: Die Testerkenntnisse zusammengefasst

3.5.2 Anzahl und Verlauf der Fräsungen

Um die Information des Rotationswinkels zu übermitteln, wurden mehrere Möglichkeiten in Betracht gezogen.

Eine Veränderung der Tiefe wird aufgrund des resultierenden nichtlinearen Verlaufs ausgeschlossen.

Die Nutzung einer rampenförmigen Fräsung hat den Vorteil, dass zu jedem Punkt eindeutig erkennbar ist, um welchen Winkel es sich handelt, da die Fräsung jede Position nur einmal durchläuft. Es gibt jedoch einen Sprungpunkt, an dem eine Detektion des Winkels vorraussichtlich erschwert ist. Sinusförmige Fräsungen haben keinen solchen Sprungpunkt, befinden sich aber mehrmals in einer Rotation an der gleichen Position.

Bei Variationen mit einer Rampe wird erwartet, dass eine Frequenzanalyse anhand einer Fouriertransformation zur Ermittlung des Rotationswinkels verwendet werden kann. Für sinusförmige Fräsungen kann durch das Zusammenspiel einer Sinus- und einer Kosinuskurve die Phase erkannt werden. Die Nutzung von Fräsungen mit variablen Tiefen ist nur bei sehr flachen Einschnitten möglich. Dabei besteht das Risiko, keine deutlichen Daten zu erhalten.

Für die weitere Untersuchung anhand der 3D-Simulation und darauf basierenden Signalverarbeitung wird die Variante mit einer Rampe und einer Konstanten (Abb. 3.6) gewählt. Die Verwendung einer dritten Fräsung limitiert den verfügbaren Platz und bietet keinen erkennbaren Vorteil.

4 Dreidimensionale Simulation

Entsprechend der 2D-Tests werden CAD-Modelle mit vielversprechenden Fräsungsverläufen erstellt. An diesen Modellen wird eine dreidimensionale Simulation durchgeführt. Ziel ist es, genaue und möglichst realitätsnahe Daten zu erhalten.

4.1 Verwendete Software

Die Arbeitsschritte der Simulation werden gemäß Tabelle 4.1 von verschiedenen Programmen erledigt.

Aufgabe	Programm
Modell-Erstellung	OpenSCAD
Modell-Export	FreeCAD
Mesh-Generierung	Gmsh
Berechnung	GetDP

Tabelle 4.1: Unterteilung der Simulationsschritte

4.1.1 OpenSCAD

OpenSCAD ist eine CAD-Software, in der sich dreidimensionale Festkörper erstellen lassen. Die Objekte werden in einem Skript beschrieben, welches von OpenSCAD in das 3D-Modell umgesetzt wird. Es ist dadurch möglich, das Modell mit Parametern zu definieren und somit schnell genaue Änderungen durchzuführen. Außerdem gibt der programmierungsbasierte Ansatz große Freiheiten. Das zu simulierende Modell wird mit OpenSCAD erstellt [6].

4.1.2 FreeCad

FreeCad ist eine 3D-CAD-Software mit einer großen Spanne an Anwendungsmöglichkeiten. Für diese Arbeit wird FreeCad aufgrund der Kompatibilät mit vielen Dateitypen verwendet. Für die Simulation extern modellierter Objekte in Gmsh werden diese als .step-Dateien benötigt. Die Datei des in OpenSCAD erstellten Modells wird in FreeCad geöffnet und die einzelnen Objekte werden jeweils als .step-Datei exportiert [11].

4.1.3 Gmsh

Für Simulationen mit der Finite-Elemente-Methode wird das gegebene Modell in viele kleine Formen unterteilt, die zusammen das Mesh ergeben. Die Berechnung der magnetischen Größen erfolgt für jede dieser Formen (siehe Kapitel 2.3). Gmsh verwendet mehrere Algorithmen, um ein Mesh zu erzeugen, das möglichst wenig Arbeitsspeicher in Anspruch nimmt [4].

4.1.4 GetDP

Die Simulation geschieht durch numerische Berechnung eines Gleichungssystems, das auf dem Mesh basiert. Diese Berechnung wird von einem Solver durchgeführt. GetDP ist ein solcher Solver. [3].

4.2 Modell

Das 3D-Modell wird entsprechend der zweidimensionalen Vorlage in OpenSCAD umgesetzt (Abb. 4.1).

Zusätzlich zur Welle werden der Dauermagnet und das Sensor-Array im Modell erstellt (Abb. 4.2). Der Magnet wird für die Simulation als Quelle des Magnetfeldes benötigt, das Array jedoch dient nur als visuelle Hilfestellung um festzulegen, wo und wie fein das Mesh auszulegen ist. Außerdem wird nur mit dem oberen Drittel der Welle gearbeitet, da die signifikanten Anteile der Magnetfeld-Reflexion von diesem Teil abhängen. Es müssen dadurch weniger Berechnungen durchgeführt werden, was die Simulation beschleunigt.

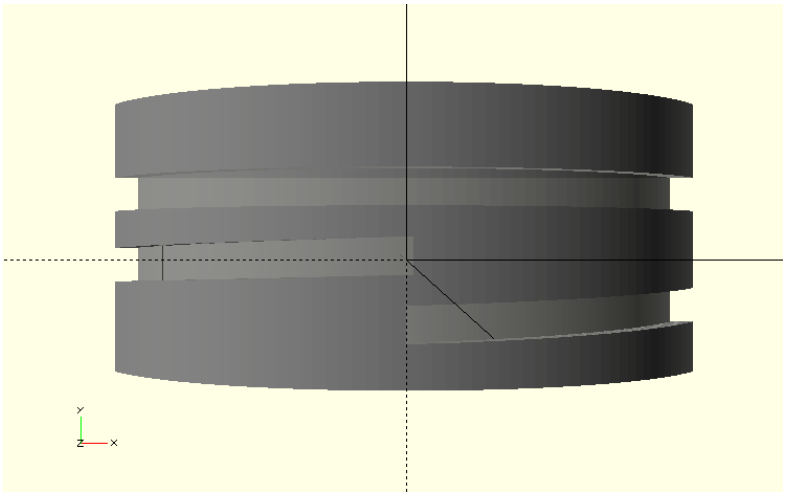


Abbildung 4.1: Das 3D-Modell der Welle mit Fräsungen von oben gezeigt

Der OpenSCAD-Code basiert auf persönlicher Kommunikation mit T. Schüthe und ist im Anhang zu sehen (Anhang B.1) [15]. Der Anhang zur Arbeit befindet sich auf CD und kann beim Erstgutachter eingesehen werden.

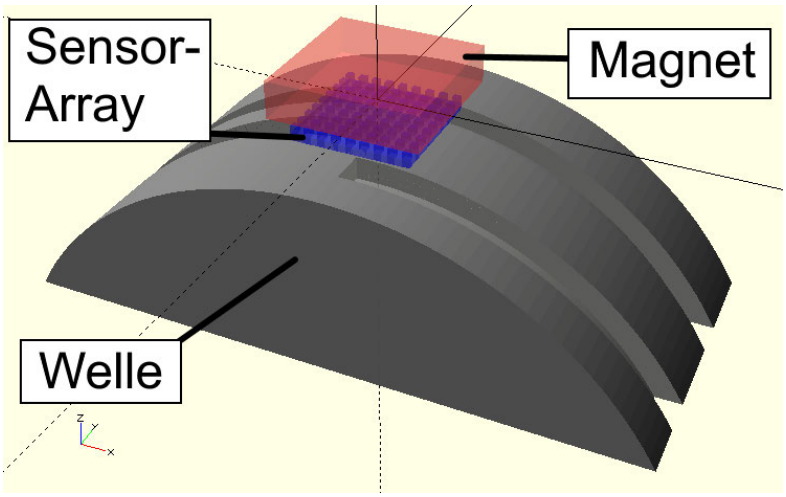


Abbildung 4.2: Das 3D-Modell aus der Vogelperspektive dargestellt

4.3 Vorgehen in Gmsh

Die Simulation wird in mehreren Dateien konfiguriert (Anhang C). Die Dateistruktur ist in Abbildung 4.3 dargestellt und kurz erläutert. Die Dateien wurden von T. Schüthe übernommen und angepasst [13].

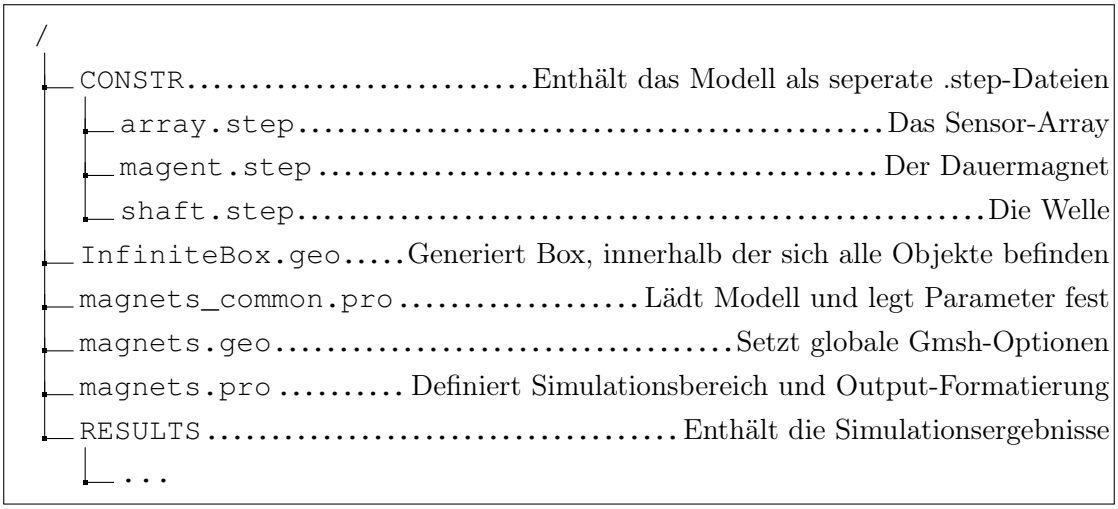


Abbildung 4.3: Dateistruktur des Gmsh-Projekts

4.3.1 Mesh-Auslegung

Um aussagekräftige Ergebnisse zu erhalten, sollte das Mesh möglichst fein eingestellt sein. Jedoch vergrößert dies die Simulationszeit stark. Das Mesh wird deshalb um das Sensor-Array herum räumlich limitiert. Es fallen unwichtige Elemente weg, wodurch das verbleibende Mesh feiner gestaltet werden kann. Es wird eine Auflösung von 67x67 innerhalb des Simulationsbereichs gewählt. Dadurch wird ein guter Kompromiss aus Qualität und Effizienz erzielt. Eine Methode anhand derer die Qualität der Ergebnisse verbessert werden könnte ist es, das Mesh gezielt an relevanten Stellen, in diesem Fall um das Sensor-Array, zu verfeinern. Diese Methode aufgrund der hohen Komplexität konnte nicht umgesetzt werden.

4.3.2 Simulationsparameter

Im Gmsh-UI ist rechts das Modell zu sehen und links können Parameter verändert werden (Abb. 4.4). Pro Objekt lassen sich der Magnetisierungswinkel, die Remanenzflussdichte Br und die relative magnetische Permeabilität μr anpassen. Für den Magneten wird $Br = 0,4\,T$ und $\mu r = 167,532$ festgelegt. Das Sensor-Array wird als neutral angenommen und hat somit eine Permeabilität von $\mu r = 1$, während für die Welle $\mu r = 1000$ gilt, was ein grober Wert für Eisen ist.

Wichtig ist es, den Magnetisierungswinkel des Dauermagneten so auszulegen, dass die Feldlinien in entweder x- oder y-Richtung verlaufen. Dies beruht darauf, dass die TMR-Sensoren aufgrund von der Komplexität, dies in der Produktion umzusetzen, Feldlinien in z-Richtung nicht detektieren können.

4.3.3 Simulation

Nach Start der Simulation werden alle Berechnungen durchgeführt. Die Resultate werden visuell angezeigt und in Textform automatisch in dem dafür vorgesehenen Ordner gespeichert.

Die Ergebnisse sind in z-Richtung in mehreren Höhenschichten unterteilt. Die unterste Schicht liegt in der Welle, während die oberste Schicht über dem Sensor-Array liegt. Diese Aufteilung erlaubt es, zu untersuchen, wo das Magnetfeld am besten verwendet werden kann. Idealerweise ist dies auf Höhe des Arrays. Für die weitere Arbeit wird mit den besten Ergebnissen gearbeitet und abschließend reflektiert, ob dieser Aspekt optimiert werden kann.

Die Simulation wird pro Rotationswinkel durchgeführt. Für die dreidimensionale Simulation wird die Rotation in acht Schritte je 45° unterteilt.

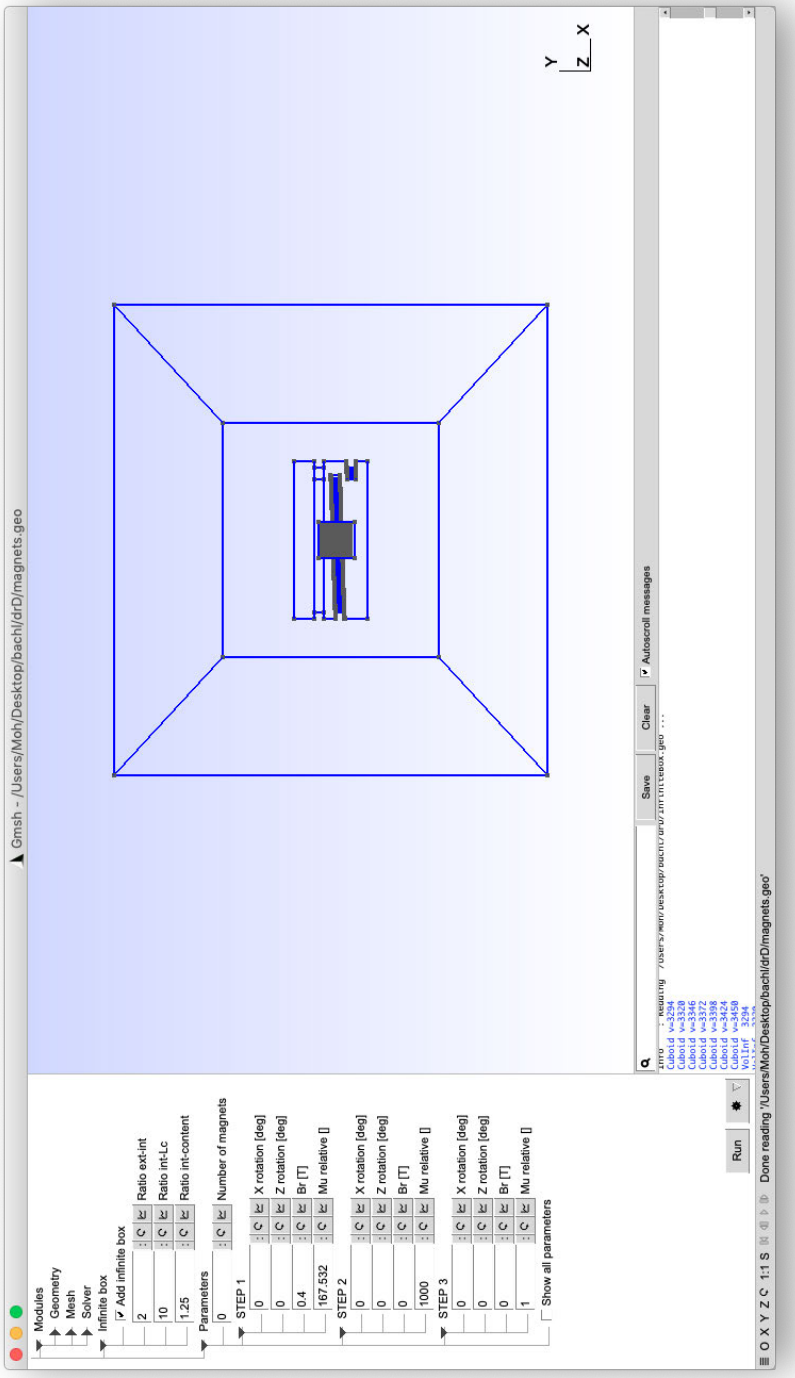


Abbildung 4.4: Screenshot des Gmsh-Fensters

5 Signalverarbeitung

Aus den Daten der dreidimensionalen Simulation soll nun anhand von Signalverarbeitung der Rotationswinkel ermittelt werden. Hierfür wird Matlab verwendet [8].

5.1 Aufbau der Daten

Die Ergebnisse liegen in Textdateien vor. Wie in Kapitel 4 besprochen, gibt es mehrere Schichten, die jeweils in einer Datei gespeichert sind. Die Daten beinhalten drei Matrizen für die Dimensionen X, Y und Z sowie drei Matrizen für das Magnetfeld, wie es in den verschiedenen Richtungen jeweils gemessen wird. Diese Matrizen werden als U, V und W gekennzeichnet. Jede Matrix ist abhängig von der gewählten Auflösung und entspricht hier dem Format 67x67.

5.2 Vorgehensweise

Über eine Rotation ändert sich die Position der rampenförmigen Fräsung. Es wird ein Modell erstellt, das diese Veränderung darstellt [16]. Grundlage der Signalverarbeitung ist es, dass die Positionsänderung durch Anwendung der Fouriertransformation gemessen wird. Aus persönlicher Kommunikation mit K.-R. Riemschneider hat sich ergeben, dass dafür der Schwerpunkt der Spektrallinien berechnet werden kann [12]. Ziel ist es, über eine Rotation einen linearen Verlauf des Schwerpunkts zu erhalten. Dadurch kann ein Winkel ermittelt werden, indem überprüft wird, wo sich der jeweilige Schwerpunkt auf dem linearen Verlauf befindet. Um dies zu erreichen, werden die Daten so bearbeitet, dass relevante Teile hervorgehoben und andere ignoriert werden.

5.3 Ablauf

Die Modell-Erstellung geschieht anhand von acht Simulationsergebnissen, die eine Rotation in 45°-Schritten durchlaufen. Auf Basis dieses Modells werden andere Winkel berechnet. Die folgenden Teile der Signalverarbeitung befinden sich in einer Schleife, die pro 45°-Schritt einmal durchlaufen wird. In Abbildung 5.7 sind die einzelnen Bearbeitungsschritte pro Winkelposition über eine Rotation dargestellt. Zum Test mit anderen Rotationswinkeln werden dieselben Berechnungen durchgeführt und anhand des Modells der Winkel ermittelt.

5.3.1 Import der Rohdaten

Listing 5.1 zeigt den Teil des Matlab-Skripts, der die Daten importiert. Die Dateipfade der Simulationsergebnisse sind in der Variable „str“ gespeichert. Per *load*()-Befehl werden die Daten in eine Variable geladen. Diese wird daraufhin auf sechs Matrizen, wie in Kapitel 5.1 beschrieben, aufgeteilt.

```
f = load(str(i));
X = reshape(f(:,1),67,67);
Y = reshape(f(:,2),67,67);
Z = reshape(f(:,3),67,67);
U = reshape(f(:,4),67,67);
V = reshape(f(:,5),67,67);
W = reshape(f(:,6),67,67);
```

Listing 5.1: Auszug aus processing.m
Import der Daten

Die Rohdaten sind in Abbildung 5.1 dargestellt. Es ist deutlich zu sehen, wie der unbearbeitete Teil der Welle in Blau von Fräsungen durchzogen ist. Die genauen Werte variieren jedoch stark.

5.3.2 Kontrastverstärkung

Um den Unterschied zwischen den Fräsungen und der unbehandelten Welle hervorzuheben, wird der Kontrast verstärkt [5, Kap. 7.2.3]. Dazu werden der kleinste Wert, der größte Wert und der daraus resultierende Wertebereich ermittelt. Alle Werte die über

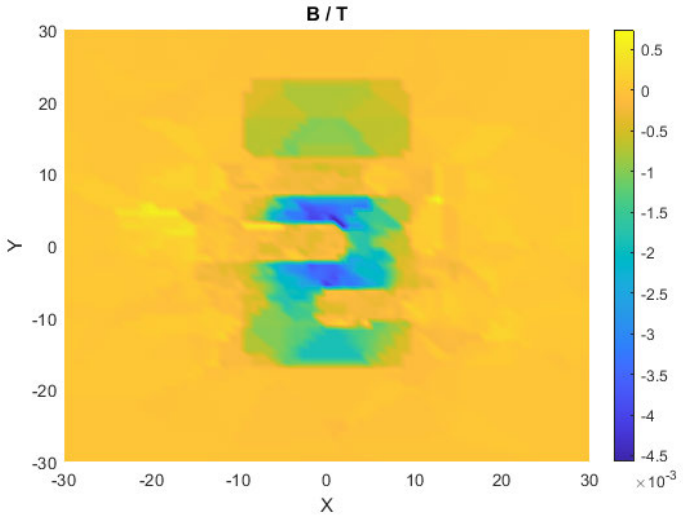


Abbildung 5.1: Unbearbeitete Daten

bzw. unter einem bestimmten Schwellwert liegen, werden dann mit dem entsprechenden Extremwert überschrieben (Listing 5.2). Die genauen Schwellwerte wurden experimentell ermittelt.

```
% increase contrast
minval = min(U,[],'all');
maxval = max(U,[],'all');
valrange = abs(minval-maxval);
u(U>(minval+0.55*valrange)) = maxval;
u(U<(minval+0.47*valrange)) = minval;
```

Listing 5.2: Auszug aus processing.m
Kontrastverstärkung

Durch den verstärkten Kontrast sind die Variationen innerhalb eines Blockes nun einheitlich. Es entstehen jedoch aggressive Übergänge zwischen den großen und kleinen Werten. Dies kommt vor allem an den äußeren Teilen der Welle vor (Abb. 5.2).

5.3.3 Anwendung einer Fensterfunktion

Die ungewollten Artefakte, die durch die Kontrastverstärkung entstehen, liegen meist an den äußeren Enden der Welle. Da die benötigte Information, die Lage der Fräsungen,

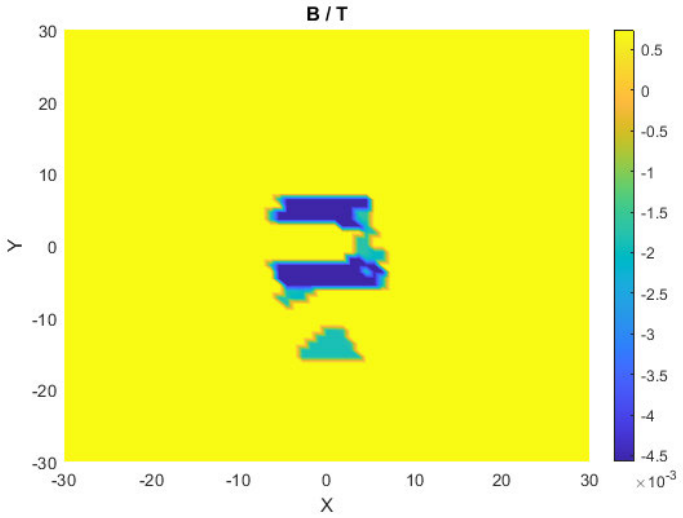


Abbildung 5.2: Kontrastverstärkte Daten

großteils in der Mitte gelegen ist, können die Daten dementsprechend gefiltert werden. Es wird eine Fensterfunktion angewendet, die entlang der y-Achse äußere Werte gegen Null laufen lässt [5, Kap. 7.3.5]. Vor der Multiplikation mit der Fensterfunktion werden alle Werte um Null zentriert (Listing 5.3). Ziel der Zentrierung ist es, konstant bipolare Maxima und Minima zu erzeugen. Die äußeren Werte, die durch die Fensterfunktion gegen Null laufen, haben so möglichst wenig Einfluss auf die weiteren Operationen.

```
% center around zero
u = u + (valrange/2 - maxval);

% window
w = flattopwin(length(U));
u = u.*w';
```

Listing 5.3: Auszug aus processing.m
Zentrierung und Fensterfunktion

Das Fenster ist in Abbildung 5.3 deutlich erkennbar. Die Art des Fensters wurde experimentell ausgewählt. Das verwendete Flat-Top-Fenster verhält sich aggressiver als zum Beispiel ein Hamming-Fenster. Es hat sich herausgestellt, dass diese Eigenschaft für die vorliegende Konfiguration vorteilhaft ist, da ungewollte Werte stärker unterdrückt werden.

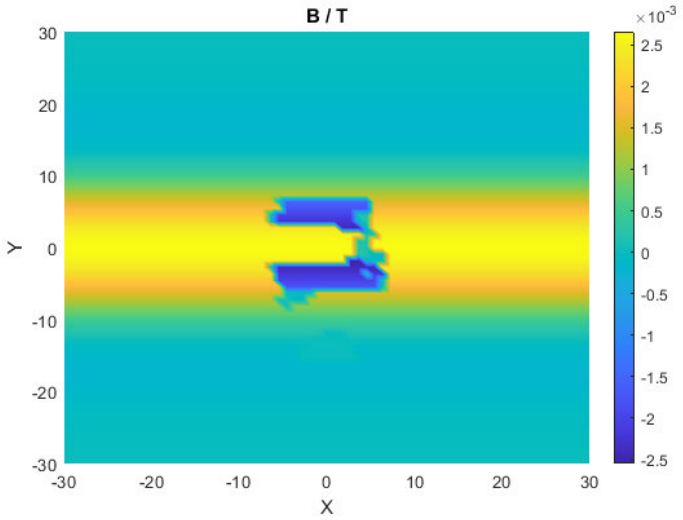


Abbildung 5.3: Daten mit Fensterfunktion multipliziert

5.3.4 Berechnung einer repräsentativen Linie

In x-Richtung sollen auch äußere Werte unterdrückt werden. Es kann hierfür auch eine Fensterfunktion angewendet werden. Ziel ist es aber, eine eindimensionale Fouriertransformation durchzuführen. Deshalb wird stattdessen die Linie in y-Richtung bei $x = 0$ für die weiteren Berechnungen verwendet. Wird jedoch nur diese eine Linie berücksichtigt, ist es nicht möglich, zwischen Winkeln zu unterscheiden, die nah beieinander liegen. Um dies zu ermöglichen, wird der Durchschnitt der neun zentralen Linien berechnet (Listing 5.4). So haben Änderungen am Rand des berücksichtigten Bereichs Einfluss auf das Resultat. Je näher die Änderung an der Mitte liegt, desto mehr Linien haben den neuen Wert, wodurch sich der Einfluss verstärkt. Zusätzlich dazu wird die Funktion *smoothdata*() mit der „movmedian"-Methode angewendet, um die Linie zu glätten und Ausreißer zu korrigieren. Kleine Schwingungen, die nicht bei jedem Rotationswinkel auftreten, führen sonst oft zu inkorrekten Ergebnissen.

```
% center line average
uc = mean(u(30:38,:));

% smooth data
uc = smoothdata(uc,'movmedian');
```

Listing 5.4: Auszug aus processing.m
Berechnung des Linien-Durchschnitts und Glättung

In Abbildung 5.4 ist eine resultierende Linie zu sehen.

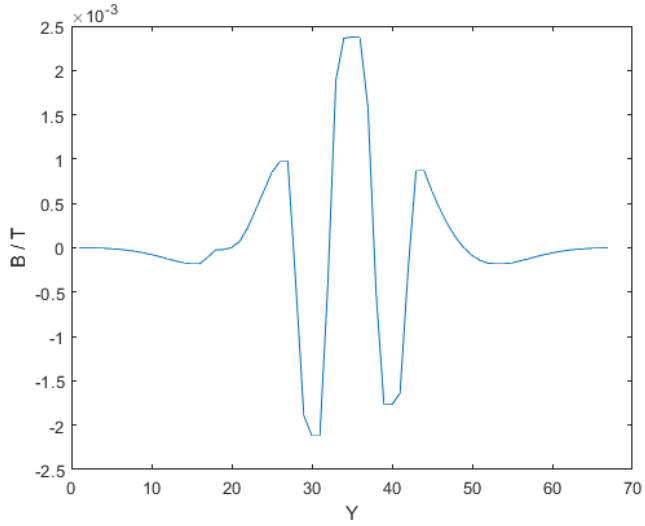


Abbildung 5.4: Verlauf der zentralen Linien im Durchschnitt

5.3.5 FFT und Schwerpunktberechnung

Es wird nun die FFT der repräsentativen Linie berechnet und der Schwerpunkt der Spektrallinien gemäß der Formel

$$Schwerpunkt = \frac{\sum(x \cdot y)}{\sum y} \tag{5.1}$$

ermittelt. Die ersten zehn Spektrallinien werden dafür verwendet. Je mehr der Linien berücksichtigt werden, desto mehr haben kleine Schwingungen Einfluss auf das Ergebnis. Werden zu wenige Linien verwendet, fehlen die nötigen Details und die Grundschwingung kommt stärker zum Vorschein. Es handelt sich hierbei um den horizontalen Schwerpunkt. Dieser gibt an, wo auf der x-Achse die Spektrallinien im Durchschnitt am stärksten ausgeprägt sind. Der Schwerpunkt wird jeweils gespeichert (Listing 5.5).

```
% get fft
fU = abs(fft(uc));

% center of gravity
spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));
```

```
% save center of gravity to results
95 cog(i) = spx;
```

Listing 5.5: Auszug aus processing.m
Berechnung der FFT und des Schwerpunkts der Spektrallinien

Abbildung 5.5 zeigt die berechneten Spektrallinien und deren Schwerpunkt für einen Rotationswinkel. In Abbildung 5.6 sind alle Schwerpunkte über eine Rotation zu sehen. Der nahezu lineare Verlauf ist wichtig, um den Rotationswinkel bei Simulationsergebnissen, die nicht zur Erstellung des Modells verwendet wurden, ermitteln zu können.

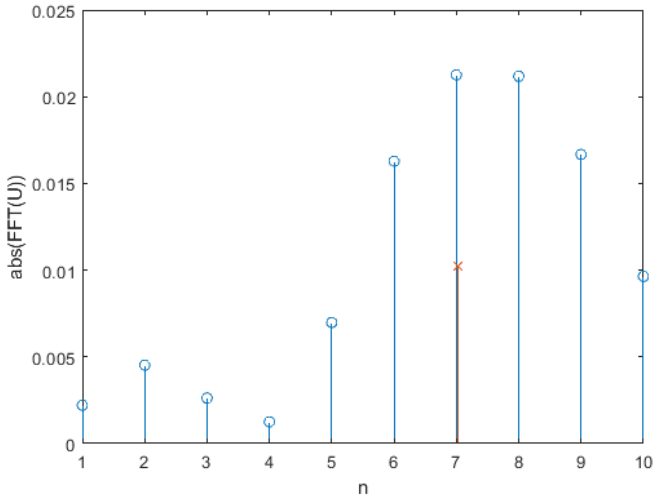


Abbildung 5.5: Fouriertransformation der zentralen Linien als Stem-Plot dargestellt. Spektrallinien-Schwerpunkt als rotes Kreuz markiert

5.3.6 Test des Modells

Um das Modell zu testen, wird der Verlauf der Schwerpunkte der Spektrallinien als linear angenommen. Es wird der Wertebereich der Schwerpunkte berechnet (Listing 5.6). Das Modell geht eine Rotation von 45° bis 360° durch. Der Bereich von 0° bis 45° fehlt dementsprechend. Deshalb wird der Wertebereich erweitert (Z.115). Der Faktor $\frac{1}{7}$ ergibt sich dadurch, dass

$$360° - 45° = 315°$$

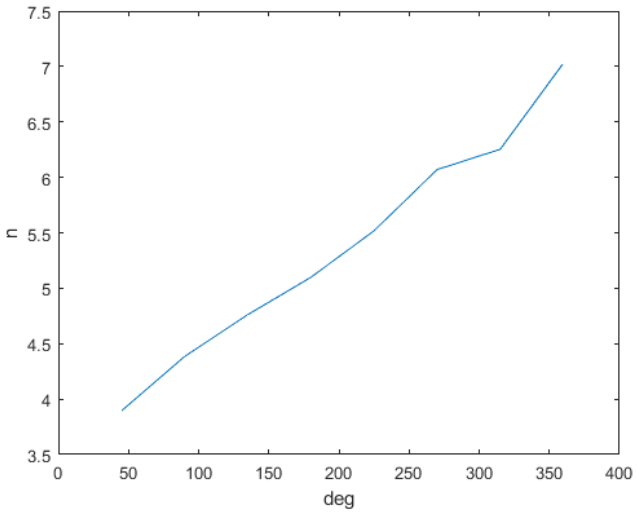


Abbildung 5.6: Verlauf der Spektrallinien-Schwerpunkte über eine Rotation

der bisherige Bereich ist und ein 45° Schritt fehlt. Dieser entspricht also gemäß

$$\frac{315°}{45°} = 7$$

einem Siebtel des bisherigen Bereichs.

```
% get range
cogmin = min(cog);
cogmax = max(cog);
frange = abs(cogmin - cogmax);
cogrange = frange + frange/7;
realmin = cogmin - abs(cograngе - frange);
```

Listing 5.6: Auszug aus processing.m
Berechnung des Wertebereichs der Schwerpunkte

Für die Berechnung des Winkels wird zum Schwerpunkt des Testergebnisses der Minimalwert des Modells addiert. Dadurch kann eine simple Prozentrechnung durchgeführt werden (Listing 5.7).

```
%calculate angle
test_angle = ((spx+(-1*realmin))/cograngе)*360;
```

Listing 5.7: Auszug aus processing.m
Berechnung des Winkels

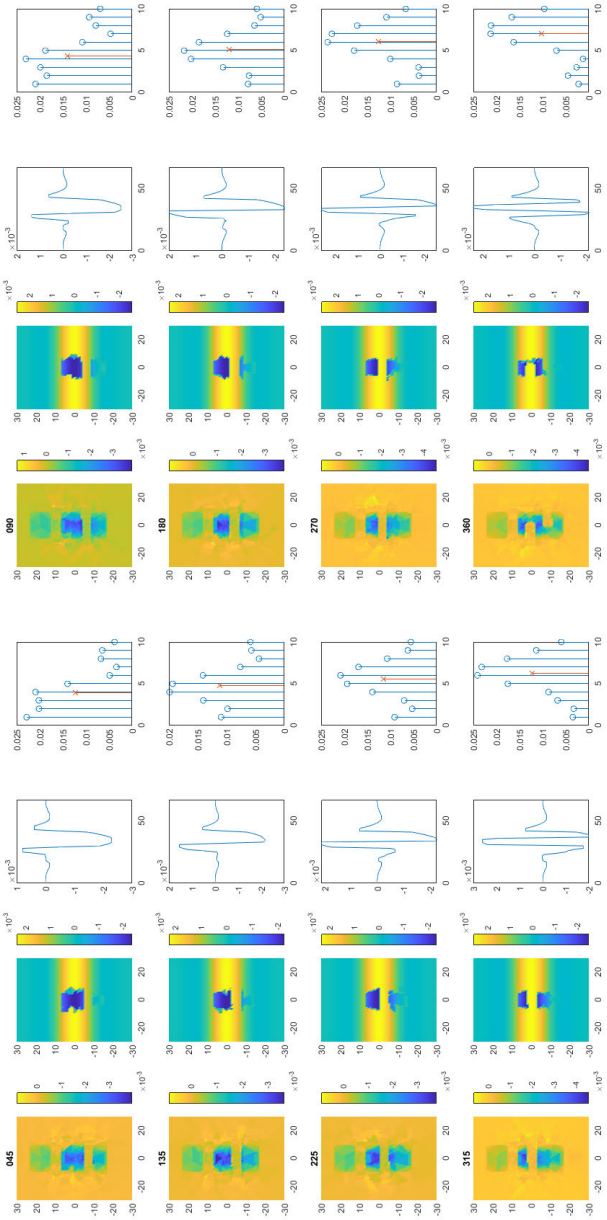


Abbildung 5.7: Signalverarbeitungs-Prozess über acht Winkelpositionen dargestellt. Je Position vier Bilder von links nach rechts: Unbearbeitet, mit Kontrastverstärkung und mit Fensterfunktion, Verlauf der zentralen Linien, FFT und Spektrallinien-Schwerpunkt

6 Test, Auswertung und Anpassung

Die Signalverarbeitung wird nun getestet. Dafür wird die Welle in Rotationswinkeln simuliert, die nicht bei der Modellerstellung verwendet werden. Wie in Kapitel 5.3.6 beschrieben wird daraus ein Winkel berechnet. Die Ergebnisse werden dokumentiert und Abweichungen werden untersucht. Es werden nach Bedarf Änderungen erarbeitet und umgesetzt.

6.1 Erster Testdurchlauf

Für den ersten Testdurchlauf werden alle Einstellungen von der Erstellung der Signalverarbeitung beibehalten.

6.1.1 Durchführung

Die getesteten Winkelpositionen werden willkürlich gewählt. Bis auf den 5°-Test, der aufgrund der Nähe zum Sprungpunkt der Fräsung gewählt wurde, liegen sie stets grob zwischen den acht Werten des Modells. Die Ergebnisse sind in Tabelle 6.1 zusammen mit der absoluten Abweichung dargestellt. Die relative Abweichung bewertet gleichwertige absolute Abweichungen stärker je kleiner der erwartete Wert ist. Dies ist für die Auswertung von Rotationswinkeln nicht hilfreich, da gleichwertige Abweichungen überall den gleichen Effekt haben.

Es ist zu sehen, dass das Modell zu vielen kleinen Abweichungen führt, jedoch auch mehrere starke Fehler zulässt. Diese liegen bei 5°, 68° und 341°.

Winkel / °	Ermittelter Winkel / °	Absolute Abweichung / °
5	90,1418	85,1418
20	19,8722	0,1278
68	88,8727	20,8727
115	121,1457	6,1457
150	147,4536	2,5464
205	199,8474	5,1523
246	239,3478	6,6522
295	297,1151	2,1151
341	290,9779	50,0221

Tabelle 6.1: Ergebnisse des ersten Testdurchlaufs

6.1.2 Auswertung

Nach näherer Betrachtung können die starken Abweichung in zwei Kategorien eingeteilt werden. Zum einen gibt es Fehler, die um den Sprungpunkt der Fräsung bei 0° auftreten. Grund hierfür ist, dass durch die Mittelung vieler Linien bei der Signalverarbeitung (Kapitel 5.3.4) der Anfang und das Ende der Fräsung gleichzeitig Einfluss auf das Ergebnis haben. Zum anderen kommt es zu Fehlern, wenn die rampenförmige Fräsung als komplett horizontal gemessen wird. Dies geschieht aufgrund von zu geringer Auflösung. Die kleinen Ungenauigkeiten können darauf zurückgeführt werden, dass entweder der Verlauf der Schwerpunkte nicht perfekt linear ist oder die Berechnung anhand des minimalen und maximalen Schwerpunkts nicht ausreicht. Die Gerade des Verlaufs wird deshalb für die weiteren Tests anhand von linearer Regression ermittelt. Die Testergebnisse werden dementsprechend mit den daraus resultierenden Koeffizienten berechnet.

Um die falschen Werte zu beheben, die aufgrund von unzureichender Auflösung entstehen, wird der Simulationsbereich verkleinert. Dadurch muss die Auflösung selbst nicht erhöht werden, was die benötigte Zeit pro Simulation stark erhöhen würde. Die relevanten Informationen für die Ermittlung des Rotationswinkels befinden sich zentral in den Bildern der Simulation. Ein kleinerer Bereich sollte dementsprechend keinen Einfluss auf die Resultate haben. Es muss jedoch überprüft werden, ob die bisherige Flat-Top-Fensterfunktion weiterhin geeignet ist. Für die Korrektur der Fehler um den Sprungpunkt, wird die Fräsung so verändert, dass es zwischen Anfang und Ende keine Überlappung mehr gibt. Dadurch wird ein steilerer Übergang der Schwerpunkte erwartet. Außerdem kann die Anzahl der zentralen Linien, deren Durchschnitt berechnet wird, reduziert werden. Dies

kann jedoch zu Fehlern der anderen Art führen, da kleine Veränderungen des Winkels nicht erkannt werden.

6.2 Zweiter Testdurchlauf

Für diese Tests wird der Simulationsbereich auf ein Viertel der bisherigen Größe reduziert und zwischen Anfang und Ende der rampenförmigen Fräsung wird ein kleiner Abstand gelassen, sodass es auch im Magnetfeld nicht zu Überlagerungen kommt.

6.2.1 Anpassung der Signalverarbeitung

Es wurde deutlich, dass in der Signalverarbeitung Änderungen vorgenommen werden müssen, um den linearen Verlauf der Schwerpunkte beizubehalten. Dieser wird anhand linearer Regression berechnet. Für die Ermittlung eines Winkels aus Simulationsdaten wird die Gerade um den jeweils zu testenden Schwerpunkt nach unten verschoben, sodass die Nullstelle der Funktion mit dem Wert des Winkels übereinstimmt. Anstelle der Flat-Top-Fensterfunktion wird ein Blackman-Harris-Fenster verwendet, weil Letzteres breiter verläuft und somit weniger Werte des kleineren Simulationsbereichs ausblendet. Die Kontrastverstärkung wird so angepasst, dass die Daten vor der Multiplikation mit dem Fenster binäre Werte haben. Die unbehandelten Werte bei der sanfteren Variante liegen nicht konstant an der gleichen Position, wodurch es zu Ungenauigkeiten bei den Spektrallinien-Schwerpunkten kommt. Die Anzahl der Spektrallinien, die zur Berechnung des Schwerpunkts verwendet werden, wird von zehn zu fünf verändert. Die höhere Anzahl führt nicht unbedingt zu Ungenauigkeiten, jedoch ist der Verlauf eher leicht S-förmig als gerade. Eine weitere Änderung ist die Erhöhung der Anzahl an zentralen Linien deren Durchschnitt für die FFT verwendet wird von 9 auf 19. Durch die Verkleinerung des Simulationsbereichs deckt jeder Schritt entlang einer Achse in den Simulationsergebnissen im Vergleich zum ersten Testdurchlauf die Hälfte des Weges ab. Die Änderung entspricht grob einer Verdoppelung, wodurch ungefähr derselbe Bereich Einfluss auf das Ergebnis hat. Außerdem wird zusätzlich zur „movmedian“-Methode auch mit der „gaussian“-Methode geglättet. Diese Änderung hat keinen sichtbaren Einfluss auf den Verlauf der Schwerpunkte, verhindert aber, dass Ergebnisse durch eckige Verläufe der zentralen Linien verfälscht werden.

In Abbildung 6.1 sind die Einflüsse der Veränderungen auf den Verlauf der Spektrallinien-Schwerpunkte dargestellt. Abbildung 6.1a zeigt den Verlauf bei dem alle beschriebenen Änderungen aktiv sind. In den Abbildungen 6.1b - 6.1e unterscheidet sich jeweils nur das angegebene Merkmal.

6.2.2 Durchführung

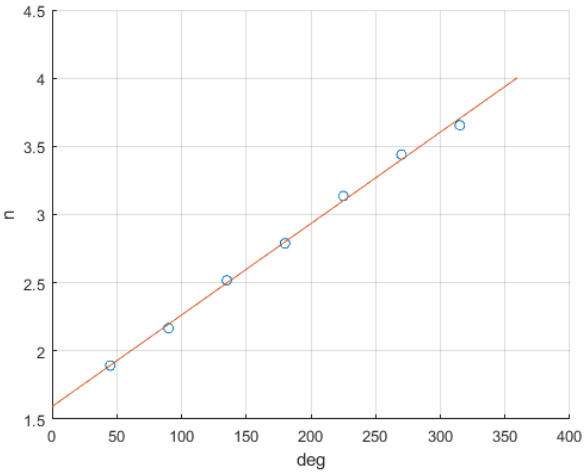
Es werden Winkelpositionen in 20°-Schritten getestet. Die jeweils berechneten Schwerpunkte der Spektrallinien sind in Abbildung 6.2 zusammen mit dem linearen Verlauf des Modells dargestellt. In Tabelle 6.2 sind die daraus ermittelten Winkel zu sehen.

Winkel / °	Ermittelter Winkel / °	Absolute Abweichung / °
20	26,5761	6,5761
40	40,0763	0,0763
60	49,4195	10,5805
80	77,4338	2,5662
100	96,8579	3,1421
120	124,3924	4,3924
140	138,4353	1,5647
160	161,4115	1,4115
180	178,4236	1,5764
200	199,3092	0,6908
220	221,9468	1,9468
240	242,0234	2,0234
260	265,8766	5,8766
280	287,5680	7,5680
300	303,2281	3,2281
320	308,6962	11,3038
340	307,7196	32,2804
360	264,3046	95,6954

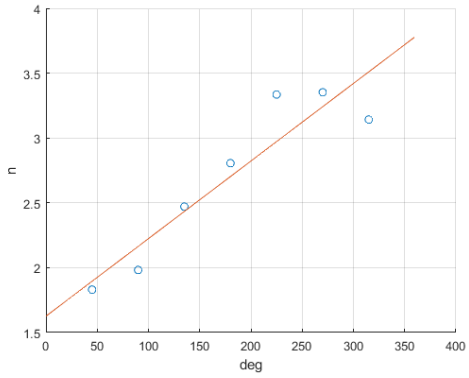
Tabelle 6.2: Ergebnisse des zweiten Testdurchlaufs

6.2.3 Auswertung

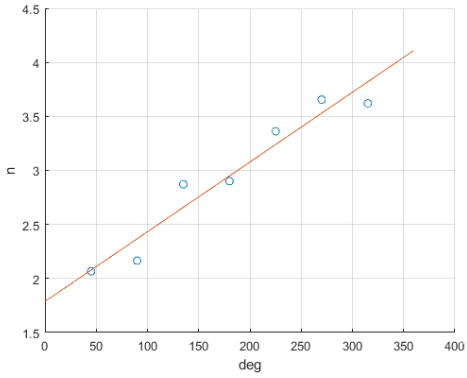
In den Testergebnissen wird deutlich, dass ein lineares Modell die berechneten Schwerpunkte um den Sprungpunkt nicht korrekt einordnen kann. Dort bilden die Testpunkte eine Kurve, die in Richtung des entgegengesetzten Extrempunkts verläuft. Die restlichen Werte stimmen meist grob mit den Erwartungen überein. Die Abweichungen könnten



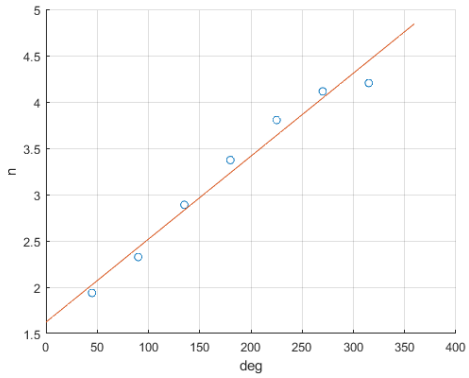
(a) Signalverarbeitung des zweiten Durchlaufs mit allen Veränderungen



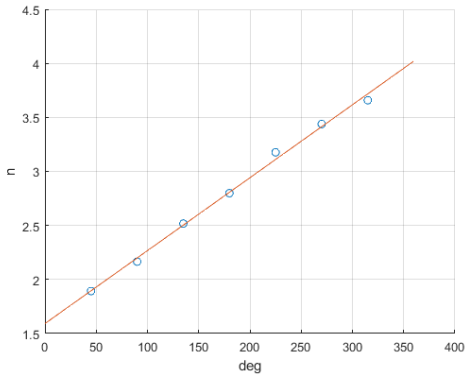
(b) Flat-Top-Fenster



(c) Sanftere Kontrastverstärkung



(d) 10 Spektrallinien



(e) 9 zentrale Linien im Durchschnitt

Abbildung 6.1: Vergleich von Signalverarbeitung des ersten und zweiten Durchlaufs

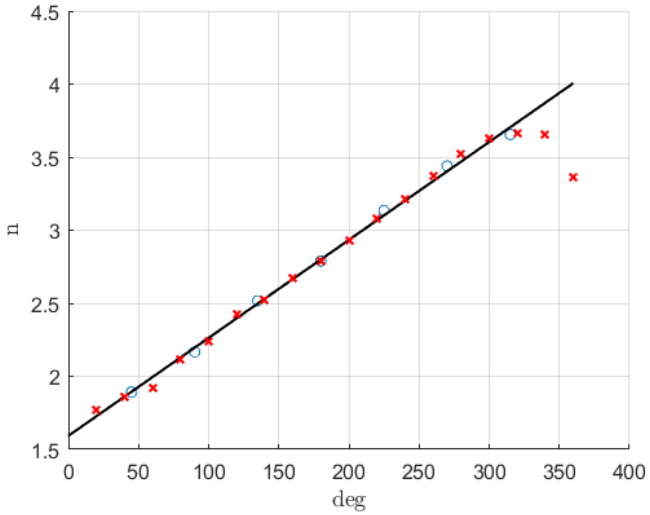


Abbildung 6.2: Testergebnisse (Kreuze) im Vergleich zu linearem Verlauf

weiterhin aufgrund von unzureichender Auflösung entstehen. Jedoch kann auch die Verwendung eines anderen Modells, das die Kurven miteinbezieht, zu besseren Ergebnissen führen.

6.3 Dritter Testdurchlauf

In dieser Iteration wird untersucht, ob durch Verwendung eines kurvenförmigen Modells genauere Ergebnisse erzielt werden können.

6.3.1 Anpassung der Signalverarbeitung

Anstelle der linearen Regression wird nun eine polynomiale Regression verwendet. Dies hat den Nachteil, dass es mehrere Lösungen für den Winkel zu einem Schwerpunkt-Wert gibt. Komplexe Lösungen und Werte außerhalb des 360°-Bereichs werden ignoriert, sodass nur relevante Werte vorkommen können. Trotzdem bleiben stets zwei Möglichkeiten: Eine Lösung bei steigendem und eine bei fallendem Schwerpunkt. Ist die Drehrichtung der Welle bekannt, kann je nach Veränderung des Schwerpunkts die richtige Lösung für den Winkel gewählt werden. Der optimale Grad des Polynoms wird experimentell ermittelt. Für die Berechnung der Koeffizienten werden dabei zusätzlich zu den Schwerpunkten,

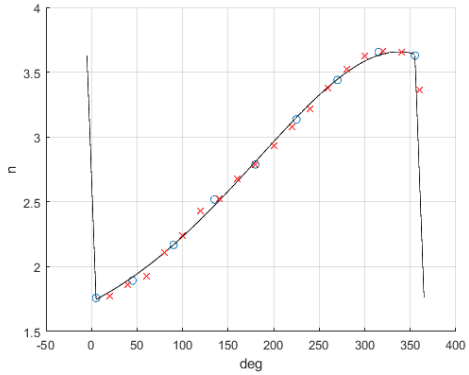
Winkel / °	Grad 4	Grad 5	Grad 6	Grad 7	Grad 8	Grad 9
20	11,2023	14,3737	16,4231	10,0130	5,6861	13,9202
40	32,5913	39,3187	40,1474	38,5577	13,5627	40,1918
60	45,8725	51,0216	51,3293	51,5654	56,2168	49,2455
80	80,5584	80,4317	79,9639	81,2351	83,6412	78,7668
100	101,5458	99,1278	98,5862	99,1522	98,4878	102,5589
120	128,6591	125,2022	124,9807	124,2918	121,4565	125,2676
140	141,6849	138,5521	138,5852	137,5516	135,5165	135,3492
160	162,2687	160,5451	160,9363	160,1336	161,6773	154,3490
180	177,1710	176,8844	177,3867	177,2636	180,0000	180,0000
200	195,3779	196,8794	197,2875	198,0100	199,3814	206,0800
220	215,4210	218,3926	218,4799	219,4962	218,2717	220,7134
240	233,9744	237,4510	237,1894	237,7795	235,0930	230,8905
260	258,0972	260,6746	260,1083	259,5860	258,3367	243,1095
280	284,6329	283,9268	283,4314	281,8266	287,2514	311,8577
300	313,3803	305,2785	305,5535	304,5692	309,4325	314,4014
320	0,0000	316,3323	317,8105	322,6202	316,4566	315,1521
340	341,6872	348,6078	347,7086	340,1543	353,3403	354,9104
360	356,4078	356,4078	356,4078	356,4078	356,4078	278,9862

Tabelle 6.3: Ermittelter Winkel in ° für verschiedene Polynomgrade

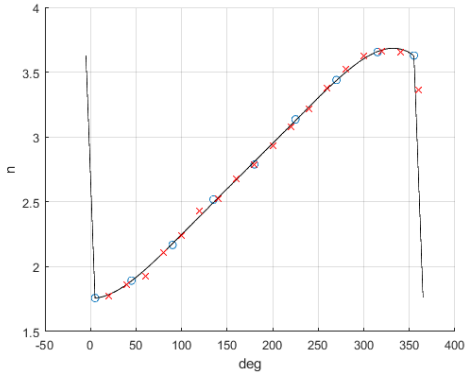
die einen linearen Verlauf bilden, die Werte bei 5° und 355° verwendet. Bei niedrigen Polynomgraden sind die Kurven meist zu flach und verfälschen mittig gelegene Werte. Wird der Grad zu hoch gewählt entstehen ungewollte Kurven. Die Grade 4, 5, 6, 7, 8 und 9 werden im Folgenden verglichen. Zwischen 5° und 355° kann das Polynom aufgrund des steilen Verlaufs nicht angewendet werden. Stattdessen wird für diesen Bereich ein linearer Verlauf angenommen.

6.3.2 Durchführung

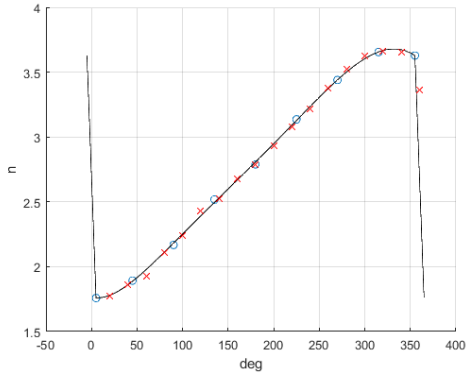
In Abbildung 6.3 sind die Modelle mit den Testergebnissen, die als Kreuze markiert sind, zu sehen. Die Ergebnisse der polynomialen Regression mit den Graden 4, 5, 6, 7, 8 und 9 werden verglichen (Tabelle 6.3 + 6.4). Es werden dabei dieselben Simulationsdaten aus Kapitel 6.2.2 verwendet.



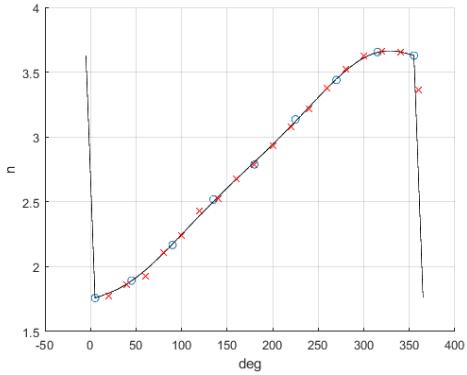
(a) Schwerpunkt-Verlauf Polynom 4. Grades



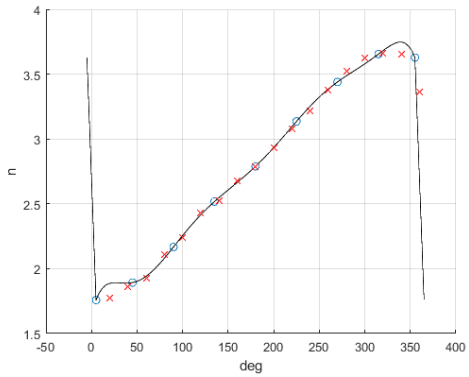
(b) Schwerpunkt-Verlauf Polynom 5. Grades



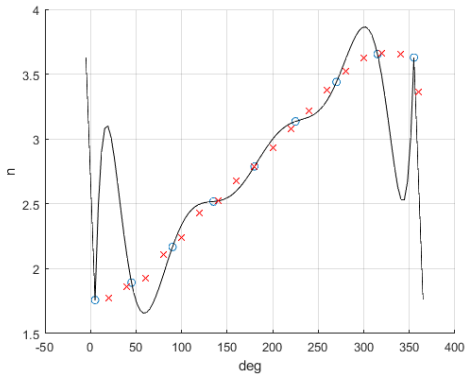
(c) Schwerpunkt-Verlauf Polynom 6. Grades



(d) Schwerpunkt-Verlauf Polynom 7. Grades



(e) Schwerpunkt-Verlauf Polynom 8. Grades



(f) Schwerpunkt-Verlauf Polynom 9. Grades

Abbildung 6.3: Verläufe von Polynomen verschiedener Grade mit Testergebnissen

Winkel / °	Grad 4	Grad 5	Grad 6	Grad 7	Grad 8	Grad 9
20	8,7977	5,6263	3,5769	9,987	14,3139	6,0798
40	7,4087	0,6813	0,1474	1,4423	26,4373	0,1918
60	14,1275	8,9784	8,6707	8,4346	3,7832	10,7545
80	0,5584	0,4318	0,0361	1,2351	3,6412	11,2332
100	1,5458	0,8722	1,4138	0,8478	1,5122	2,5589
120	8,6591	5,2022	4,9807	4,2918	1,4565	5,2676
140	1,6849	1,4479	1,4148	2,4484	4,4835	4,6508
160	2,2687	0,5451	0,9363	0,1336	1,6773	5,6510
180	2,8290	3,1156	2,6133	2,7364	0,0000	0,0000
200	4,6221	3,1206	2,7125	1,9900	0,6186	6,0800
220	4,5790	1,6074	1,5201	0,5038	1,7283	0,7135
240	6,0256	2,5490	2,8106	2,2205	4,9070	9,1095
260	1,9028	0,6746	0,1083	0,4140	1,6633	16,8905
280	4,6329	3,9268	3,4314	1,8266	7,2514	31,8577
300	13,3803	5,2785	5,5535	4,5692	9,4325	14,4014
320	320,0000	3,6677	2,1895	2,6202	3,5434	4,8479
340	1,6872	8,6078	7,7086	0,1543	13,3403	14,9104
360	3,5922	3,5922	3,5922	3,5922	3,5922	81,0138

Tabelle 6.4: Absolute Abweichung in ° für verschiedene Polynomgrade

6.3.3 Auswertung

Aus den berechneten Winkeln wird für die Modelle die durchschnittliche Abweichung ermittelt (Tabelle 6.5). Dies zeigt, dass, aus den sechs getesteten Optionen, ein Polynom siebten Grades den Verlauf der Schwerpunkte am genauesten beschreibt. Da ein Polynom achten Grades keine vergleichbar genauen Ergebnisse liefert, ist anzunehmen, dass für diese Methodik ein Polynom siebten Grades ideal ist.

Grad des Polynoms	Durchschnittliche Abweichung / °
4	22,6834
5	3,3292
6	2,9676
7	2,7471
8	5,7435
9	12,0118

Tabelle 6.5: Durchschnittliche Abweichung der Modelle

Eine durchschnittliche Abweichung von knapp unter 3° ist nicht ideal, da dementsprechend teilweise deutliche Fehler auftreten können. Diese entstehen vermutlich weiterhin

durch zu geringe Auflösung und könnten somit durch eine Verkleinerung des Simulationsbereichs oder eine Verfeinerung des Gitternetzes verbessert werden.

Trotzdem wird durch diese Tests bewiesen, dass anhand eines magnetischen Sensor-Arrays und einem Signalverarbeitungs-Algorithmus der Rotationswinkel einer Welle ermittelt werden kann.

6.4 Schwächen der beschriebenen Methodik

In diesem Abschnitt werden die Schwächen der Methodik, die in dieser Arbeit erstellt und beschrieben wurde, untersucht.

6.4.1 Erstellung des Modells

Für die Erstellung der linearen und polynomialen Modelle wurden meist willkürlich gewählte Winkelpositionen in gleichmäßigen Abständen verwendet. Aufgrund der Limitation durch Simulationszeiten wurde die Optimierung dieses Aspekts nicht näher betrachtet. Es ist demnach möglich, dass durch eine Verschiebung der Winkel bessere Ergebnisse erzielt werden können.

6.4.2 Übergangsbereich im Schwerpunkt-Verlauf

Der Übergangsbereich im Schwerpunkt-Verlauf führt dazu, dass ein Schwerpunkt zu zwei verschiedenen Winkeln führen kann. Der Übergang wird durch den Sprungpunkt der rampenförmigen Fräsung verursacht und wird durch die Berechnung des zentralen Durchschnitts (Kapitel 5.3.4) verstärkt. Der Übergangsbereich ist maximal 10° breit. Dieser Wert ist für den Zweck dieser Arbeit zufriedenstellend, jedoch noch verbesserungsfähig.

6.4.3 Abhängigkeit von der Fensterfunktion

In Abbildung 6.1b ist deutlich erkennbar, dass der Verlauf der Schwerpunkte stark von der verwendeten Fensterfunktion abhängt. Um dies noch eindeutiger darzustellen, ist in Abbildung 6.4 der Verlauf zu sehen, bei dem keine Fensterfunktion angewendet wurde.

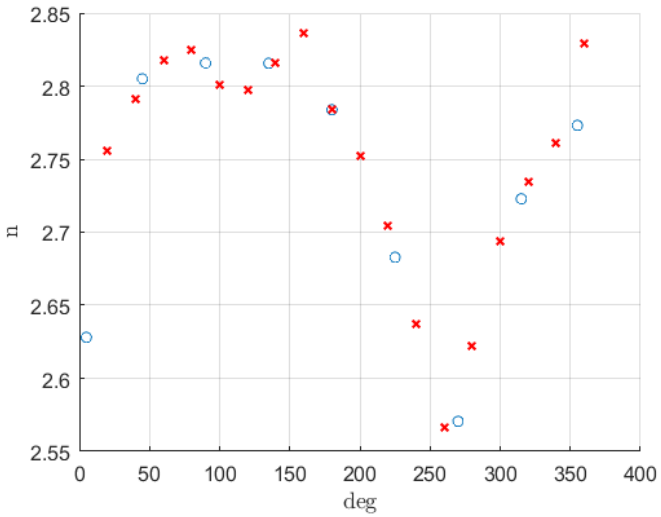


Abbildung 6.4: Verlauf der Schwerpunkte ohne Fensterfunktion

Darin kann keine klare Struktur erkannt werden. Verbindet man die einzelnen Punkte, ergibt sich eine Kurve die grob einem quadrierten Sinus ähnelt, jedoch sonst keine Struktur aufweist. Dieses Verhalten ist ein klarer Nachteil dieser Methodik, da die Fensterfunktion stark vom Messbereich und somit vom Abstand des Sensors abhängt, wie durch Verkleinerung des Simulationsbereichs deutlich wurde. Demnach werden die verschiedenen Anwendungsmöglichkeiten begrenzt, bzw. es muss für jede Anwendung ein anderes Fenster verwendet werden.

Die Benutzung einer Fensterfunktion und die Art des Fensters haben großen Einfluss auf die Endergebnisse. Es kann sein, dass nicht der Abstand zwischen den zwei Fräsungen zu einer Verschiebung des Schwerpunkts der Spektrallinine führt, sondern wie viel der rampenförmigen Fräsung innerhalb der Fensterfunktion liegt. Dies erklärt auch das Plateau des Schwerpunkts um 300°, da ab dort die Fräsung voll sichtbar ist und sich nur noch innerhalb des Fensters verschiebt und warum dieses Plateau zwischen 0° und 50° nicht genauso stark ausgeprägt ist. Dort ist der Anteil der Fräsung innerhalb der Fensterfunktion nicht konstant.

Diese Vermutung gibt einen klaren Weg vor, um noch genauere und eindeutigere Ergebnisse zu erhalten. Um sie zu bestätigen, müssen weitere Tests durchgeführt werden. Zum Beispiel kann überprüft werden, ob sich das gleiche Verhalten ohne die Verwendung einer Fensterfunktion, eventuell durch eine Fräsung, deren Breite sich stetig ändert, erreichen

lässt (Abb. 6.5). Dadurch würde sich auch der Anteil der Fräsung im messbaren Bereich ändern, was den Test des vermuteten Verhaltens ohne eine Fensterfunktion ermöglicht.

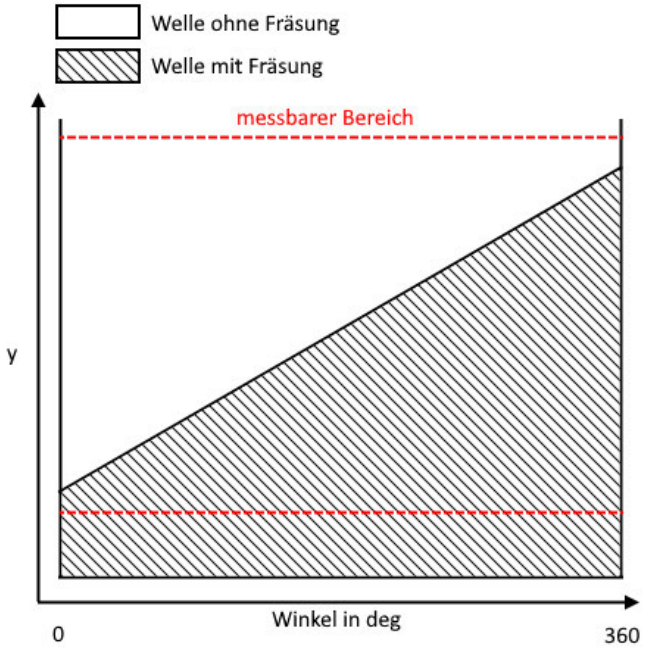


Abbildung 6.5: Mögliche Auslegung der Welle zum Test der beschriebenen Vermutung

6.5 Zusammenfassung

Es wurde die grundlegende Signalverarbeitung, die in Kapitel 5 beschrieben ist, anhand mehrerer Testdurchläufe so angepasst und verändert, dass den Testdaten bestmögliche Winkel zugeordnet werden können. Die Art des Modells wurde stark verändert. In der ersten Iteration wurde die Gerade des Verlaufs der Spektrallinien-Schwerpunkte anhand des kleinsten und größten Punkts errechnet. Die Berechnung mit linearer Regression hat die Verlässlichkeit des Modells erhöht. Durch polynomiale Regression wurden Kurven in das Modell miteinbezogen. Ein Polynom siebten Grades hat mit einer durchschnittlichen Abweichung von $2,7471°$ sich für diese Methodik als optimal erwiesen. Es wurde deutlich, dass aufgrund von unzureichender Simulationsauflösung Ungenauigkeiten entstehen. Diese wurden durch Verkleinerung des Bereichs der Simulation verbessert. Es wurden mögliche Defizite der beschriebenen Methodik betrachtet und, wie durch Betrachtung dieser Schwachstellen, bessere Ergebnisse erzielt werden können.

7 Schlussfolgerungen

Die Ergebnisse und Feststellungen, die sich im Laufe dieser Arbeit ergeben haben, werden in diesem Kapitel zusammengefasst. Es wird betrachtet, ob die Ziele erreicht wurden und wie auf den Resultaten aufgebaut werden kann. Weitere Ideen für die Weiterführung und Optimierung werden beschrieben.

7.1 Zusammenfassung

Ziel dieser Arbeit war es im Wesentlichen, zu untersuchen, wie mit einem magnetischen Sensor-Array der Rotationswinkel einer Welle erfasst werden kann. Dafür wurde der mechanische Aufbau des Systems ausgelegt und anhand von zweidimensionalen Simulationen getestet. Darauf basiert wurde ein CAD-Modell erstellt. Es wurde ein Signalverarbeitungs-Algorithmus erstellt, der aus Daten von 3D-Simulationen den jeweiligen Winkel berechnet. Dieser Algorithmus wurde mehrmals getestet und verbessert.

Im Grundlagenteil wurde betrachtet, wie bei der Nutzung des Sensor-Arrays an der langen Seite der Welle die Information der Winkelposition übermittelt werden kann. Aufgrund der Einfachheit der Umsetzung wurde gefolgert, dass dies gut durch Fräsungen erledigt werden kann. Die 2D-Simulation wurde verwendet, um einzelne Merkmale einer Fräsung zu untersuchen und so optimale Parameterwerte zu finden. Außerdem wurden verschiedene Möglichkeiten, die Informationen auf der Welle anzubringen, verglichen und bezüglich ihrer Insensitivität gegenüber ungenauer Positionierung des Sensors, ihrer Effizienz und ihrer Eindeutigkeit beurteilt. Für die dreidimensionalen Simulationen und die Erstellung der Signalverarbeitung wurde ein CAD-Modell erstellt, das eine Referenzfräsung und eine rampenförmige Fräsung hat. Um die Positionsänderung der Welle zu erkennen und weiterzuverarbeiten, wird die Fouriertransformation verwendet und der Schwerpunkt der Spektrallinien berechnet. Die Signalverarbeitung basiert darauf, anhand einer begrenzten Anzahl an Simulationen, bei denen der Rotationswinkel der Welle

gleichmäßig ansteigt, ein Modell zu erstellen, das den Verlauf dieser Schwerpunkte über eine Umdrehung genau darstellt. Um dies zu erreichen, wird der Kontrast in den Bildern der Daten verstärkt, es wird eine Fensterfunktion angewendet und es wird aus mehreren zentralen Linien der Durchschnitt berechnet. Es wurden mehrere Testdurchläufe durchgeführt, die jeweils ausgewertet wurden und so zu Verbesserungen der Signalverarbeitung führten. Es wurde deutlich, dass ein Polynom siebten Grades den Verlauf der Schwerpunkte am genauesten darstellt. Schwächen der entwickelten Methodik wurden betrachtet, wobei vor allem die starke Abhängigkeit von der Fensterfunktion hervorsticht. Es wurde dabei die Vermutung aufgestellt, dass der resultierende Winkel nicht von der Position der Fräsung abhängt, sondern davon wie groß der Anteil der Fräsung ist, der innerhalb der Fensterfunktion liegt.

7.2 Fazit

Die erarbeiteten Ergebnisse zeigen, dass die Winkelerfassung mit einem magnetischen Sensor-Array anhand der verwendeten Methoden möglich ist. Das Kernziel der Arbeit wurde somit erreicht. Die Genauigkeit des Modells, das mit der Signalverarbeitung erstellt wird, ist mit einer durchschnittlichen Abweichung von $2,7471°$ durchaus zufriedenstellend, lässt sich jedoch weiter optimieren. Vor allem anhand der Feststellungen bezüglich der Abhängigkeit von der Fensterfunktion kann die Methodik weiter verbessert werden. Diese werden im Laufe der Auswertung als Schwäche der Vorgehensweise bezeichnet, jedoch wird daraus direkt deutlich, wie noch robustere und genauere Ergebnisse erzielt werden können, was dem Ziel der Arbeit entspricht.

7.3 Ausblick

Ein offensichtlicher Weg, die Methodik, die in dieser Arbeit erstellt wurde, weiter zu testen, ist es, die Winkelerfassung mit dem Sensor-Array in Hardware aufzubauen und so weitere Tests durchzuführen. Die Anzahl der getesteten Winkelpositionen in dieser Arbeit ist durch die Dauer eines Simulationsdurchlaufs limitiert. Somit kann die Genauigkeit einer Methodik in Hardware anhand einer größeren Anzahl an Messpunkten bewertet werden. Dabei sollte überprüft werden, welchen Einfluss Produktionstoleranzen und andere Ungenauigkeiten auf die Ergebnisse haben.

Außerdem kann untersucht werden, ob andere Ansätze zur Auslegung der Fräsungen bessere Ergebnisse erzielen. Vor allem ein Ansatz, bei dem der Anteil einer Fräsung im messbaren Bereich verändert wird, wie in Kapitel 6.4.3 beschrieben, wirkt vielversprechend.

Literaturverzeichnis

[1] Bundesministerium für Umwelt, Naturschutz und nukleare Sicherheit (BMU): *Klimaschutz in Zahlen*. Ausgabe 2020. – URL https://www.bmu.de/publikation/klimaschutz-in-zahlen-2020/. – Zugriffsdatum: Oktober 2021

[2] Doppelbauer, M.: *Grundlagen der Elektromobilität*. Springer Fachmedien Wiesbaden, 2020. – URL https://doi.org/10.1007/978-3-658-29730-5

[3] Dular, P. ; Geuzaine, C. ; Henrotte, F. ; Legros, W.: A general environment for the treatment of discrete problems and its application to the finite element method. In: *IEEE Transactions on Magnetics* 34 (1998), Nr. 5, S. 3395–3398. – URL https://doi.org/10.1109/20.717799

[4] Geuzaine, C. ; Remacle, J.-F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. In: *Int J Numer Meth Eng* 79 (2009), Nr. 11, S. 1309–1331. – URL https://doi.org/10.1002/nme.2579

[5] Jähne, B.: *Digitale Bildverarbeitung*. Springer Berlin Heidelberg, 2012. – URL https://doi.org/10.1007/978-3-642-04952-1

[6] Kintel, M.: *OpenSCAD*. Version 2021.01 - Besucht September 2021. – URL https://openscad.org/. – Zugriffsdatum: September 2021

[7] Marinescu, M.: *Elektrische und magnetische Felder*. Springer Berlin Heidelberg, 2012. – URL https://doi.org/10.1007/978-3-642-25794-0

[8] MATLAB: *Version 9.10.0.1669831 (R2021a Update 2)*. Natick, Massachusetts : The MathWorks Inc., 2021

[9] Meeker, D.C.: *Finite Element Method Magnetics*. Version 4.2 (21Apr2019 Build). – URL https://www.femm.info/. – Zugriffsdatum: September 2021

[10] Reif, K. (Hrsg.): *Sensoren im Kraftfahrzeug*. Vieweg+Teubner, 2010. – URL https://doi.org/10.1007/978-3-8348-9718-3

[11] Riegel, J. ; Mayer, W. ; Van Havre, Y. (2001-2017): *FreeCAD*. Version 0.19-24291. – URL https://www.freecadweb.org/. – Zugriffsdatum: September 2021

[12] Riemschneider, K.-R.: *(personal communication, Juni 15, 2021)*

[13] Schüthe, T.: *(personal communication, April 1, 2021)*

[14] Schüthe, T.: *(personal communication, Juni 29, 2021)*

[15] Schüthe, T.: *(personal communication, Mai 28, 2021)*

[16] Schüthe, T. ; Petrak, O. ; Jünemann, K. ; Riemschneider, K.-R.: *Positionserfassung mittels Sensor-Array aus Tunnel-Magnetoresistiven Vortex-Dots und lernender Signalverarbeitung*. S. 373–396. In: Tille, T. (Hrsg.): *Automobil-Sensorik 3: Prinzipien, Technologien und Anwendungen*, Springer Berlin Heidelberg, 2020. – URL https://doi.org/10.1007/978-3-662-61260-6_14

[17] Slama, P. ; Aichriedler, L.: *Hoch performante Rotorlage-Sensorik für bürstenlose E-Maschinen in Hybridantrieben*. S. 233–250. In: Tille, T. (Hrsg.): *Automobil-Sensorik: Ausgewählte Sensorprinzipien und deren automobile Anwendung*, Springer Berlin Heidelberg, 2016. – URL https://doi.org/10.1007/978-3-662-48944-4_12

[18] TDK: *Product Datasheet TAD4140 360° angle sensor (Document revision 3.11)*. – URL https://product.tdk.com/en/system/files?file=dam/doc/product/sensor/angle/tmr-angle/data_sheet/ds_sensor_tmr-angle_tad4140_en.pdf. – Zugriffsdatum: Oktober 2021

[19] Villwock, J. ; Hanau, A.: Finite Berechnungsverfahren. In: *Dubbel*. Springer Berlin Heidelberg, 2018, S. 133–138. – URL https://doi.org/10.1007/978-3-662-54805-9_18

A Matlab-Code für zweidimensionale Simulation

```
function writefile(PTS,SEG,BLOCK_POS,fname)
%WRITEFILE(PTS,SEG,BLOCK_POS,fname)
%   .....
%=========================================================================
% Author: Thorben Schuethe
% Date  : June 2021
% Mail  : thorben.schuethe@haw-hamburg.de, thorben@t-schuethe-de
%=========================================================================
fileID = fopen([fname,'.fem'],'W','n','UTF-8');
fprintf(fileID,'[Format]      =  4.0\n');
 fprintf(fileID,'[Frequency]   =  0\n');
 fprintf(fileID,'[Precision]   =  1e-008\n');
 fprintf(fileID,'[MinAngle]    =  30\n');
 fprintf(fileID,'[DoSmartMesh] =  1\n');
 fprintf(fileID,'[Depth]       =  1\n');
 fprintf(fileID,'[LengthUnits] =  millimeters\n');
 fprintf(fileID,'[ProblemType] =  planar\n');
 fprintf(fileID,'[Coordinates] =  cartesian\n');
 fprintf(fileID,'[ACSolver]    =  0\n');
 fprintf(fileID,'[PrevType]    =  0\n');
 fprintf(fileID,'[PrevSoln]    =  ""\n');
 fprintf(fileID,'[Comment]     =  "Add comments here."\n');
 fprintf(fileID,'[PointProps]   = 0\n');
 fprintf(fileID,'[BdryProps]   = 1\n');
 fprintf(fileID,'  <BeginBdry>\n');
 fprintf(fileID,'    <BdryName> = "BOUNDARY"\n');
 fprintf(fileID,'    <BdryType> = 4\n');
 fprintf(fileID,'    <A_0> = 0\n');
 fprintf(fileID,'    <A_1> = 0\n');
 fprintf(fileID,'    <A_2> = 0\n');
 fprintf(fileID,'    <Phi> = 0\n');
 fprintf(fileID,'    <c0> = 0\n');
 fprintf(fileID,'    <c0i> = 0\n');
 fprintf(fileID,'    <c1> = 0\n');
 fprintf(fileID,'    <c1i> = 0\n');
 fprintf(fileID,'    <Mu_ssd> = 0\n');
 fprintf(fileID,'    <Sigma_ssd> = 0\n');
 fprintf(fileID,'    <innerangle> = 0\n');
 fprintf(fileID,'    <outerangle> = 0\n');
 fprintf(fileID,'  <EndBdry>\n');
 fprintf(fileID,'[BlockProps]  = 3\n');
 fprintf(fileID,'  <BeginBlock>\n');
 fprintf(fileID,'    <BlockName> = "455 Stainless Steel"\n');
 fprintf(fileID,'    <Mu_x> = 470\n');
 fprintf(fileID,'    <Mu_y> = 470\n');
 fprintf(fileID,'    <H_c> = 0\n');
 fprintf(fileID,'    <H_cAngle> = 0\n');
 fprintf(fileID,'    <J_re> = 0\n');
 fprintf(fileID,'    <J_im> = 0\n');
 fprintf(fileID,'    <Sigma> = 1.6699999999999999\n');
 fprintf(fileID,'    <d_lam> = 0\n');
```

```
fprintf(fileID,'    <Phi_h> = 20\n');
fprintf(fileID,'    <Phi_hx> = 0\n');
fprintf(fileID,'    <Phi_hy> = 0\n');
fprintf(fileID,'    <LamType> = 0\n');
fprintf(fileID,'    <LamFill> = 1\n');
fprintf(fileID,'    <NStrands> = 0\n');
fprintf(fileID,'    <WireD> = 0\n');
fprintf(fileID,'    <BHPoints> = 26\n');
fprintf(fileID,'      0 0\n');
fprintf(fileID,'      0.067337999999999995  342.70800000000003\n');
fprintf(fileID,'      0.10261000000000001   514.03099999999995\n');
fprintf(fileID,'      0.18909000000000001   713.09400000000005\n');
fprintf(fileID,'      0.29154000000000002   854.58199999999999\n');
fprintf(fileID,'      0.40999999999999998   995.75300000000004\n');
fprintf(fileID,'      0.55725000000000002   1107.72\n');
fprintf(fileID,'      0.73009999999999997   1247.9300000000001\n');
fprintf(fileID,'      0.86777000000000004   1417.4300000000001\n');
fprintf(fileID,'      0.97345000000000004   1616.1400000000001\n');
fprintf(fileID,'      1.0632999999999999    2273.6900000000001\n');
fprintf(fileID,'      1.1339999999999999    2931.6300000000001\n');
fprintf(fileID,'      1.224  4076.3600000000001\n');
fprintf(fileID,'      1.2724 4906.5900000000001\n');
fprintf(fileID,'      1.3500000000000001    6968.6800000000003\n');
fprintf(fileID,'      1.3761000000000001    8085.8699999999999\n');
fprintf(fileID,'      1.3991 9661.5\n');
fprintf(fileID,'      1.419  11438.5\n');
fprintf(fileID,'      1.4359999999999999    13874.299999999999\n');
fprintf(fileID,'      1.4462999999999999    15736.4\n');
fprintf(fileID,'      1.4533 17169.599999999999\n');
fprintf(fileID,'      1.464  19777.400000000001\n');
fprintf(fileID,'      1.478  22986.700000000001\n');
fprintf(fileID,'      1.4887999999999999    25795\n');
fprintf(fileID,'      1.4959 27715.200000000001\n');
fprintf(fileID,'      1.4995000000000001    28604.099999999999\n');
fprintf(fileID,'  <EndBlock>\n');
fprintf(fileID,'  <BeginBlock>\n');
fprintf(fileID,'    <BlockName> = "Air"\n');
fprintf(fileID,'    <Mu_x> = 1\n');
fprintf(fileID,'    <Mu_y> = 1\n');
fprintf(fileID,'    <H_c> = 0\n');
fprintf(fileID,'    <H_cAngle> = 0\n');
fprintf(fileID,'    <J_re> = 0\n');
fprintf(fileID,'    <J_im> = 0\n');
fprintf(fileID,'    <Sigma> = 0\n');
fprintf(fileID,'    <d_lam> = 0\n');
fprintf(fileID,'    <Phi_h> = 0\n');
fprintf(fileID,'    <Phi_hx> = 0\n');
fprintf(fileID,'    <Phi_hy> = 0\n');
fprintf(fileID,'    <LamType> = 0\n');
fprintf(fileID,'    <LamFill> = 1\n');
fprintf(fileID,'    <NStrands> = 0\n');
fprintf(fileID,'    <WireD> = 0\n');
fprintf(fileID,'    <BHPoints> = 0\n');
fprintf(fileID,'  <EndBlock>\n');
fprintf(fileID,'  <BeginBlock>\n');
fprintf(fileID,'    <BlockName> = "NdFeB 10 MGOe (Bonded)"\n');
fprintf(fileID,'    <Mu_x> = 1.2230000000000001\n');
fprintf(fileID,'    <Mu_y> = 1.2230000000000001\n');
fprintf(fileID,'    <H_c> = 445634\n');
fprintf(fileID,'    <H_cAngle> = 0\n');
fprintf(fileID,'    <J_re> = 0\n');
fprintf(fileID,'    <J_im> = 0\n');
fprintf(fileID,'    <Sigma> = 0\n');
fprintf(fileID,'    <d_lam> = 0\n');
fprintf(fileID,'    <Phi_h> = 0\n');
fprintf(fileID,'    <Phi_hx> = 0\n');
fprintf(fileID,'    <Phi_hy> = 0\n');
fprintf(fileID,'    <LamType> = 0\n');
fprintf(fileID,'    <LamFill> = 1\n');
```

```
 fprintf(fileID,'    <NStrands> = 0\n');
 fprintf(fileID,'    <WireD> = 0\n');
 fprintf(fileID,'    <BHPoints> = 0\n');
 fprintf(fileID,'  <EndBlock>\n');
 fprintf(fileID,'[CircuitProps]  = 0\n');
 fprintf(fileID,'[NumPoints] = %.0f\n',length(PTS));
 for n = 1 : length(PTS)
    fprintf(fileID,'%.4f %.4f %.4f %.4f\n',PTS(n,1),PTS(n,2),PTS(n,3),PTS(n,4));
 end
 fprintf(fileID,'[NumSegments] = %.0f\n',length(SEG));
 for n = 1 : length(SEG)
    fprintf(fileID,'%.0f %.0f %.0f %.0f %.0f\n',SEG(n,1),SEG(n,2),SEG(n,3),SEG(n,4),SEG(n,5));
 end
fprintf(fileID,'[NumArcSegments] = 2\n');
%%
fprintf(fileID,'%f ',[length(PTS)-2 length(PTS)-1 180  1    1    0    0    1]);
fprintf(fileID,'\n');
fprintf(fileID,'%f ',[length(PTS)-1 length(PTS)-2 180  1    1    0    0    1]);
fprintf(fileID,'\n');
%%
fprintf(fileID,'[NumHoles] = 0\n');
fprintf(fileID,'[NumBlockLabels] = %.0f\n',length(BLOCK_POS));
[ny,nx] = size(BLOCK_POS);
 for n = 1 : ny
    fprintf(fileID,'%f ',BLOCK_POS(n,:));
    fprintf(fileID,'\n');
 end

fclose(fileID);
end
```

Listing A.1: writefile.m

```
%=========================================================================
% Author: Thorben Schuethe
% Date  : June 2021
% Mail  : thorben.schuethe@haw-hamburg.de, thorben@t-schuethe-de
%=========================================================================
set(0, 'DefaultLineLineWidth', 1.5,'DefaultAxesFontSize',12,'defaultTextInterpreter','latex');
clc ;clear all; close all;

%% create save folder

newFname = strrep(date,'-','_');
i = 1;
dir = '';

if not(isfolder(strcat('results/',newFname)))
    mkdir('results',newFname)
    mkdir(strcat('results/',newFname),'FEMFILES')
    dir = strcat('results/',newFname);
else
    while 1
        if not(isfolder(strcat('results/',newFname,'_',string(i))))
            mkdir('results',strcat(newFname,'_',string(i)))
            mkdir(strcat('results/',newFname,'_',string(i)),'FEMFILES')
            dir = strcat('results/',newFname,'_',string(i));
            break;
        end
        i = i + 1;
    end
end

%%
% Define Parameters
AIR     = [2 -1 0 0 0 1 0];
NDFEB   = [3 -1 0 90 0 1 0];
STEEL   = [1 -1 0 0 0 1 0];
```

```
% height of magnet (mm)
mag_h = 3;
% width of magnet +-50 if set to 100 (mm)
mag_w = 70;
% distance magnet and sensor (mm)
dist_mag_s = 1.65;
% distance shaft and sensor (mm)
dist_shaft_s = 2;
% height of shaft (mm)
%%
shaft_h     = 10;
% width of shaft +-50 if set to 100 (mm)
shaft_w     = 100;
% depth of shaft (mm)
shaft_d     = shaft_h-5;
% width of milling (mm)
shaft_wm     = 7;
% dist of milling 0 position (mm)
shaft_dm     = 0.5*7.15;
% first watch the unrolled result
offset = 9;
% number of files / number of steps per Rotation
Npoint = 30;
rot = linspace(0,360,Npoint);
% untere fraesung
y0  = offset.*-sind(rot*0)-shaft_dm/2-shaft_wm;
y1  = offset.*-sind(rot*0)-shaft_dm/2;
% obere fraesung
y2  = offset.*-sind(rot*0)+shaft_dm/2;
y3  = offset.*-sind(rot*0)+shaft_dm/2+shaft_wm;

% tiledlayout(1,2)
% nexttile
    area(rot,ones(Npoint,1).*shaft_w/2,'FaceColor',[1,1,1].*0.9),hold on
    area(rot,-ones(Npoint,1).*shaft_w/2,'FaceColor',[1,1,1].*0.9)
    plot(rot,y0,...
         rot,y1,...
         rot,y2,...
         rot,y3,'LineWidth',2,'color','k'),hold off
    xlabel('$\alpha_{shaft}$ (degree)')
    ylabel('x (mm)')
    xlim([0 360])
    yline([-25],'r','linewidth',2);
    yline([25],'r','linewidth',2);
%     yline([-mag_w/2],'b','linewidth',2);
%     yline([mag_w/2],'b','linewidth',2);
    [~,h_legend] = legend('sensor array','milling');
    PatchInLegend = findobj(h_legend, 'type', 'patch');
    set(PatchInLegend(1), 'FaceColor', 'r');
    set(PatchInLegend(2), 'FaceColor', 'k');
title({'unrolled shaft around 360 deg. rotation'})
%% define array points center
pxl_size = 1; % mm
arr_ctr_x = linspace(-25,25,24);
arr_ctr_y = zeros(1,length(arr_ctr_x));
arr_phy_p = [arr_ctr_x',arr_ctr_y',repmat(AIR,length(arr_ctr_x),1)];
nn = 1;
pmx = [-1 -1 1 1].*pxl_size/2;
pmy = [-1 1 1 -1].*pxl_size/2;
for n = 1 : length(arr_ctr_x)
    for m = 1 : 4
        arr_edge_x(nn) = arr_ctr_x(n)+pmx(m);
        arr_edge_y(nn) = arr_ctr_y(n)+pmy(m);
        SEG_ARR(nn,1) = nn-1;
        SEG_ARR(nn,2) = nn;
        SEG_ARR(nn,3) = -1;
        SEG_ARR(nn,4:5) = 0;
        if m == 4
            SEG_ARR(nn,2) = nn-4;
```

```
        end
        nn = nn+1;
    end
end
clear pmx pmy n nn m;
NPTS = length(arr_edge_x);
PTS_ARR = [arr_edge_x',arr_edge_y',zeros(NPTS,2)];
%% DEFINE MAGNET
mag_x = [-1 -1 1 1].*mag_w/2;
mag_y = [0 1 1 0].*mag_h+pxl_size/2+dist_mag_s;
% physical parameter of magnet
mag_ctrx  = mean(mag_x);
mag_ctry  = mean(mag_y);
mag_phy_p = [mag_ctrx,mag_ctry,NDFEB];
% points of magnet
PTS_MAG = [mag_x',mag_y',zeros(4,2)];
for m = 1 : 4
    SEG_MAG(m,:) = [NPTS-1+m,NPTS+m,-1,0,0];
end
SEG_MAG(end,2) = SEG_MAG(1,1);
NPTS = NPTS+4;
%% DEFINE SHAFT
% In this case a rotation around 360 degree is performed with two sine
% depth of mag
% folder: FEMFILES
for num = 1:length(y0)
    y = [y0(num),y1(num),y2(num),y3(num)];
    y = sort(y);
    y = repmat(y,2,1);
    y = y(:)';
    %y = y+[0,+2,-2,0,0,+2,-2,0];

fname = sprintf('%s/FEMFILES/%04.0f_deg_rot',dir,rot(num));
%==========================================================================
%
%==========================================================================
shaft_phy_p = [0,(-shaft_h-dist_shaft_s)+shaft_d/3,STEEL];
shaft_x0 = [-1 -1 zeros(1,length(y)) 1 1].*shaft_w/2;
shaft_x1 = [ 0  0       y            0 0];
shaft_x  = shaft_x0+shaft_x1;
shaft_y0 = ([-1  0 repmat([0 -1 -1  0],1,length(y)/4) 0 -1].*shaft_h-dist_shaft_s);
shaft_y1 = ([ 0  0 repmat([0  1  1  0],1,length(y)/4) 0  0].*shaft_d);
shaft_y = shaft_y0+shaft_y1;
PTS_SHAFT = [shaft_x',shaft_y',zeros(length(shaft_x),2)];
for m = 1 : length(shaft_x)
    SEG_SHAFT(m,:) = [NPTS-1+m,NPTS+m,-1,0,0];
end
SEG_SHAFT(end,2) = SEG_SHAFT(1,1);
%==========================================================================
%
%==========================================================================
% Point for air condition
SEG_AIR = [0,-dist_shaft_s/2,AIR];
% add points for boundary
PT_BOUND = [-shaft_w,0,0,0;
             shaft_w,0,0,0];
%
ALL_PTS = [PTS_ARR;PTS_MAG;PTS_SHAFT;PT_BOUND];
ALL_SEG = [SEG_ARR;SEG_MAG;SEG_SHAFT];
ALL_PHY = [arr_phy_p;mag_phy_p;shaft_phy_p;SEG_AIR];
writefile(ALL_PTS,ALL_SEG,ALL_PHY,fname);
%==========================================================================
%
%==========================================================================
% set(gca,'DataAspectRatio',[1 1 1])
%     drawnow
%     hold on
end
%%
```

```
nexttile
scatter(ALL_PTS(:,1),ALL_PTS(:,2))%,hold on
    hold on
for n = 1 : length(ALL_PTS)-2
    px = ALL_SEG(n,1)+1;
    py = ALL_SEG(n,2)+1;
    line([ALL_PTS(px,1) ALL_PTS(py,1)],[ALL_PTS(px,2) ALL_PTS(py,2)])
end
xlabel('x (mm)')
ylabel('y (mm)')
grid on
box on
hold off
title({'slice through the setup','-- magnet, sensors, shaft --'})
% scatter(arr_edge_x,arr_edge_y),hold off

%%
set(gcf,'Color','white')
set(gcf, 'Position',  [100, 100, 1800, 900])
%% save figure and properties

saveas(gcf,fullfile(dir,'layout.png'))
T = table(mag_h,mag_w,dist_mag_s,dist_shaft_s,shaft_h,shaft_w,shaft_d,shaft_wm,shaft_dm,offset);
writetable(T,fullfile(dir,'properties.txt'));

FileNameAndLocation=[mfilename('fullpath')];
newbackup=fullfile(dir,strcat(mfilename,'.m'));
currentfile=strcat(FileNameAndLocation, '.m');
copyfile(currentfile,newbackup);
```

Listing A.2: a_gen_coordinates_2cut.m

```
%=========================================================================
% Author: Thorben Schuethe
% Date  : June 2021
% Mail  : thorben.schuethe@haw-hamburg.de, thorben@t-schuethe-de
%=========================================================================
set(0, 'DefaultLineLineWidth', 1.5,'DefaultAxesFontSize',12,'defaultTextInterpreter','latex');
clc ;clear all; close all;

fdir = uigetdir('results');
fdir = extractAfter(fdir,'results\');
folder = fullfile('results',fdir,'FEMFILES');
% folder = eraseBetween(folder,1,'results');
% addpath('C:\femm42\mfiles');
%% get data from folder and enter folder path
DAT = dir([folder,'\*.fem']);
NUM_FILES = length(DAT);
cd(folder);
%% run simulation procedure
parfor ndat = 1 : NUM_FILES
    % open file
    openfemm(1);
    opendocument(DAT(ndat).name);
    % create mesh
    mi_createmesh;
    mi_analyse;
    mi_loadsolution;
%     x = linspace(-25,25,150);
    %% Get magnetic components and other values
    %1  A     Potential A or flux?
    %2  B1    Bx if planar,Br if axisymmetric
    %3  B2    By if planar,Bz if axisymmetric
    %4  Sig   conductivity?
    %5  E     stored energy density
    %6  H1    Hx if planar, Hr if axisymmetric <=====
    %7  H2    Hy if planar, Hz if axisymmetric <=====
    %8  Je    eddy current density
```

```
    %9  Js    source current density
    %10 Mu1   ?x if planar,?r if axisymmetric
    %11 Mu2   ?y if planar,?z if axisymmetric
    %12 Pe    Power density dissipated through ohmic losses
    %13 Ph    Power density dissipated by hysteresis
    %14 ff    Winding fill factor
%     for n= 1 : length(x)
%         V = mo_getpointvalues(x(n),0);
%         Hx(ndat,n) = V(6);
%         Hy(ndat,n) = V(7);
%     end
%     closefemm;
end
%% turn back into top folder
cd('..');
% remove femm path from search directory
%rmpath('C:\femm42\mfiles');
%%

% save('DAT1','x','Hx','Hy')
%%
clear all
%%
% tiledlayout(2,1)
% nexttile
%     plot(x,V(:,6))
%     xlabel('x (mm)')
%     ylabel('$H_y$ (A/m)')
% nexttile
%     plot(x,V(:,7))
%     xlabel('x (mm)')
%     ylabel('$H_y$ (A/m)')
```

Listing A.3: b_femm_parfor.m

```
%========================================================================
% Author: Thorben Schuethe
% Date  : June 2021
% Mail  : thorben.schuethe@haw-hamburg.de, thorben@t-schuethe-de
%========================================================================
set(0, 'DefaultLineLineWidth', 1.5,'DefaultAxesFontSize',12,'defaultTextInterpreter','latex');
clc ;clear all; close all;

fdir = uigetdir('results');
fdir = extractAfter(fdir,'results\');
folder = fullfile('results',fdir,'FEMFILES');
%addpath('C:\femm42\mfiles');
%% get data from folder and enter folder path
DAT = dir([folder,'\*.fem']);
NUM_FILES = length(DAT);
cd(folder);
%% generate bitmap with contour plot
parfor ndat = 1 : NUM_FILES
    openfemm;
    opendocument(DAT(ndat).name);
    mi_loadsolution;
    am_to_t = 1 / (10^7/(4*pi));
    % limits for plot in A/m
    h_lim_lo = 10;
    h_lim_hi = 40e3;
    mo_zoom(-25,-20,25,10);
    % ======> mo_showdensityplot(legend,gscale,upper_B,lower_B,type)
    mo_showvectorplot(5,1)
    mo_showcontourplot(2000,-h_lim_hi*am_to_t,h_lim_hi*am_to_t,'imag')
    mo_showdensityplot(1,0,h_lim_lo*am_to_t,h_lim_hi*am_to_t,'mag');
    mo_savebitmap([DAT(ndat).name(1:end-4),'.png'])
    closefemm
end
```

```
%%
%% run simulation procedure
for ndat = 1 : NUM_FILES
    % open file
    openfemm;
    opendocument(DAT(ndat).name);
    % create mesh
    mi_loadsolution;
    x = linspace(-25,25,150);
    %% Get magnetic components and other values
    %1  A     Potential A or flux?
    %2  B1    Bx if planar,Br if axisymmetric
    %3  B2    By if planar,Bz if axisymmetric
    %4  Sig   conductivity?
    %5  E     stored energy density
    %6  H1    Hx if planar, Hr if axisymmetric <=====
    %7  H2    Hy if planar, Hz if axisymmetric <=====
    %8  Je    eddy current density
    %9  Js    source current density
    %10 Mu1   ?x if planar,?r if axisymmetric
    %11 Mu2   ?y if planar,?z if axisymmetric
    %12 Pe    Power density dissipated through ohmic losses
    %13 Ph    Power density dissipated by hysteresis
    %14 ff    Winding fill factor
    for n= 1 : length(x)
        V = mo_getpointvalues(x(n),0);
        Hx(ndat,n) = V(6);
        Hy(ndat,n) = V(7);
    end
end
    closefemm;
%% turn back into top folder
cd('..');
% remove femm path from search directory
rmpath('C:\femm42\mfiles');
save('DAT1','x','Hx','Hy')
```

Listing A.4: c_load_solution_matfile_png.m

B OpenSCAD-Code

```
// Construction for shaft simulation using gmsh and getdp
// sensor array lies in center between magnet and shaft
//-------------------------------------------------------
// Author: Thorben Schuethe
// E-Mail: thorben.schuethe@haw-hamburg.de
//         thorben@t-schuethe.de
// Date  : 21. Mai 2021
//-------------------------------------------------------

// Display whole part?
// If true  => no cut off the shaft
// If falls => construction of upper third of shaft
DP_all   = false;
DP_mag   = true;
DP_arr   = true;
DP_shaft = true;

// for fine mesh generation set step width
$fn = 180;
// rotation of shaft (degree)
shaft_rot = 160;
// length of shaft (mm)
shaft_l   = 40;
// radius of shaft (mm)
shaft_r   = 50;
// Definition of magnet dimension
mag_x = 20; // (mm)
mag_y = 20; // (mm)
mag_z = 5;  // (mm)
// distance between magnet and shaft
dist_ms = 5;
// define depth of helix and reference
hr_depth = 3;
// width of helix and reference
hr_wth   = 5;
// distance between ref and helix
hr_dist = 4;
/************************************
 define arrayPoints
************************************/
// height of ground plane of array
arr_h = 2;
// Array size in x and y direction
arr_szxy = 15;
// number of array points
n_pt = 8;
/************************************
 define helix on shaft
************************************/
// radius of helix
hx_r    = shaft_r-hr_depth;
// width of helix
hx_w    = hr_wth;
// depth of helix
hx_d    = hr_depth+1;
```

```
// distance between start and end
hx_tz   = hr_dist;
// rotation of helix 360    => start at 0, end at 360
//                  360+15 => start at 0, end at 15 degree
hx_rot  = 356;

///************************************
//    Start construction of the shaft
//************************************/
if(DP_shaft==true)
{
    translate([0,3,-shaft_r-dist_ms])
    rotate([90,0,0])
    color([.5,.5,.5,0.9])
    difference()
    {
        // shaft as a simple cylinder
        cylinder(shaft_l,shaft_r,shaft_r,center=true);
        /************************************
            Start construction of helix and reference
        ************************************/
        //translate([0,0,-hr_wth/2-hr_dist])
        translate([0,0,0])
        rotate([0,0,90+shaft_rot])
        union()
        {
            // The helix
            for ( z = [0:hx_rot])
            {
                rotate(z)
                translate([hx_r,0,((hx_rot/360)*z/(hx_rot))*(hx_tz+hx_w)])
                rotate([90,0,0])
                cube(size = [hx_d,hx_w,1], center = false);
            }
            // The reference cut
            translate([0,0,-hr_wth/2-hr_dist])
            difference(){
                cylinder(hr_wth,hx_r+hx_d,hx_r+hx_d,center=true);
                cylinder(hr_wth+1,hx_r,hx_r,center=true);
            }
        }
        /*
        For Simulation create only the upper third
        of the shaft
        */
        if(DP_all==false)
        {
            rotate([0,0,90])
            translate([-shaft_r*1.5,0,0])
            cube([shaft_r*4,shaft_r*4,shaft_l*2],center=true);
        }
    }
}
/************************************
 Construction of Array
************************************/
if(DP_arr==true)
{
    arr_shift = arr_szxy/n_pt;
    color([0,0,1,0.3])
    translate([0,0,-dist_ms/2-arr_h/2])
    union()
        {
        translate([0,0,0.5])
        cube([arr_szxy+arr_shift/2,
              arr_szxy+arr_shift/2,1], center = true);
        for (sy = [-arr_szxy/2:arr_szxy/n_pt:arr_szxy/2])
        {
        for (sx = [-arr_szxy/2:arr_szxy/n_pt:arr_szxy/2])
```

```
        {
            translate([sx,sy,+arr_h/2])
            cube([arr_shift/2,arr_shift/2,arr_h], center = true);
        }
        }
    }
}

/************************************
 Construction of magnet
************************************/
if(DP_mag==true)
{
    color([1,.2,.2,0.3])
    translate([0,0,+mag_z/2])
    cube([mag_x,mag_y,mag_z],center=true);
}
```

Listing B.1: Construction.scad

C Gmsh-Konfigurations-Dateien

```
SetFactory("OpenCASCADE");

Function Cuboid
l12 = newl; Line(l12) = {pnt[0], pnt[1]};
l23 = newl; Line(l23) = {pnt[1], pnt[2]};
l34 = newl; Line(l34) = {pnt[2], pnt[3]};
l41 = newl; Line(l41) = {pnt[3], pnt[0]};
l56 = newl; Line(l56) = {pnt[4], pnt[5]};
l67 = newl; Line(l67) = {pnt[5], pnt[6]};
l78 = newl; Line(l78) = {pnt[6], pnt[7]};
l85 = newl; Line(l85) = {pnt[7], pnt[4]};
l15 = newl; Line(l15) = {pnt[0], pnt[4]};
l26 = newl; Line(l26) = {pnt[1], pnt[5]};
l37 = newl; Line(l37) = {pnt[2], pnt[6]};
l48 = newl; Line(l48) = {pnt[3], pnt[7]};

ll1 = newll; Line Loop(ll1) = { l12, l23, l34, l41 };
ll2 = newll; Line Loop(ll2) = { l56, l67, l78, l85 };
ll3 = newll; Line Loop(ll3) = { l12, l26, -l56, -l15 };
ll4 = newll; Line Loop(ll4) = { l23, l37, -l67, -l26 };
ll5 = newll; Line Loop(ll5) = { l34, l48, -l78, -l37 };
ll6 = newll; Line Loop(ll6) = { l41, l15, -l85, -l48 };

s1 = news; Plane Surface(s1) = { ll1 } ;
s2 = news; Plane Surface(s2) = { ll2 } ;
s3 = news; Plane Surface(s3) = { ll3 } ;
s4 = news; Plane Surface(s4) = { ll4 } ;
s5 = news; Plane Surface(s5) = { ll5 } ;
s6 = news; Plane Surface(s6) = { ll6 } ;

sl = newsl; Surface Loop(sl) = { s1, s2, s3, s4, s5, s6 };
v = newv; Volume(v) = { sl };
Printf("Cuboid v=%g", v);

If( Flag_TransfInf )
  Mesh.Algorithm3D = 6;
  // (4=Frontal, 5=Frontal Delaunay, 6=Frontal Hex, 7=MMG3D, 9=R-tree)
  For num In { l12:l85 }
    Transfinite Line{ num } = 10;
  EndFor
  For num In { l15:l48 }
    Transfinite Line{ num } = 5;
  EndFor
  For num In { s1:s6 }
    Transfinite Surface{ num } ;
  EndFor
  Transfinite Volume{ v } ;
EndIf
Return


DefineConstant[
  Flag_InfiniteBox = {1, Choices{0,1}, Name "Infinite box/Add infinite box"}
  Flag_TransfInf = {0, Choices{0,1}, Name "Infinite box/Transfinite mesh", Visible 0}
  ratioInf = {2, Name "Infinite box/Ratio ext-int", Visible Flag_InfiniteBox}
```

```
  ratioBox = {1.25, Name "Infinite box/Ratio int-content", Visible Flag_InfiniteBox}
  ratioLc = {15, Name "Infinite box/Ratio int-Lc", Visible Flag_InfiniteBox}

  xInt = {1, Name "Infinite box/xInt", Visible 0}
  yInt = {1, Name "Infinite box/yInt", Visible 0}
  zInt = {1, Name "Infinite box/zInt", Visible 0}
  xExt = {xInt*ratioInf, Name "Infinite box/xExt", Visible 0}
  yExt = {yInt*ratioInf, Name "Infinite box/yExt", Visible 0}
  zExt = {zInt*ratioInf, Name "Infinite box/zExt", Visible 0}
  xCnt = {0, Name "Infinite box/xCenter", Visible 0}
  yCnt = {0, Name "Infinite box/yCenter", Visible 0}
  zCnt = {0, Name "Infinite box/zCenter", Visible 0}
];

// Compute parameters related to the Infinite box
BoundingBox;
xCnt = (General.MinX + General.MaxX) / 2.;
yCnt = (General.MinY + General.MaxY) / 2.;
zCnt = (General.MinZ + General.MaxZ) / 2.;

xInt = (General.MaxX - General.MinX)/2.;
yInt = (General.MaxY - General.MinZ)/2.;
zInt = (General.MaxZ - General.MinZ)/2.;
// If BoundingBox is empty, replace it with a unit cube.
// This makes this file callable in isolation, e.g., for testing
If (xInt == 0 )
  xInt=1;
EndIf
If (yInt == 0 )
  yInt=1;
EndIf
If (zInt == 0 )
  zInt=1;
EndIf

// Compute aspect ratio of the Infinite Box
diagInt = Sqrt[ xInt^2 + yInt^2 + zInt^2 ];
pp = 1.2; // pp=1 square InfBox, pp>3 Box and content have the same aspect ratio
xInt *= ratioBox*(diagInt/xInt)^(1/pp);
yInt *= ratioBox*(diagInt/yInt)^(1/pp);
zInt *= ratioBox*(diagInt/zInt)^(1/pp);


// FIXME is there another (better) way to convey the values calculated here to getDP
SetNumber("Infinite box/xInt",xInt);
SetNumber("Infinite box/yInt",yInt);
SetNumber("Infinite box/zInt",zInt);
SetNumber("Infinite box/xExt",xExt);
SetNumber("Infinite box/yExt",yExt);
SetNumber("Infinite box/zExt",zExt);
SetNumber("Infinite box/xCenter",xCnt);
SetNumber("Infinite box/yCenter",yCnt);
SetNumber("Infinite box/zCenter",zCnt);


// Change the step size of the voxels to the ...
lc1inf = Sqrt[ xInt^2 + yInt^2 + zInt^2] / (ratioLc);
//lc1inf = Sqrt[ xInt^2 + yInt^2 + zInt^2] / 10;
//lc1inf = 1;
p1 = newp; Point (p1) = {xCnt-xInt, yCnt-yInt, zCnt-zInt, lc1inf};
p2 = newp; Point (p2) = {xCnt+xInt, yCnt-yInt, zCnt-zInt, lc1inf};
p3 = newp; Point (p3) = {xCnt+xInt, yCnt+yInt, zCnt-zInt, lc1inf};
p4 = newp; Point (p4) = {xCnt-xInt, yCnt+yInt, zCnt-zInt, lc1inf};
p5 = newp; Point (p5) = {xCnt-xInt, yCnt-yInt, zCnt+zInt, lc1inf};
p6 = newp; Point (p6) = {xCnt+xInt, yCnt-yInt, zCnt+zInt, lc1inf};
p7 = newp; Point (p7) = {xCnt+xInt, yCnt+yInt, zCnt+zInt, lc1inf};
p8 = newp; Point (p8) = {xCnt-xInt, yCnt+yInt, zCnt+zInt, lc1inf};

VolInf[]={}; // Define empty array in case Flag_InfiniteBox is not active
```

```
If(Flag_InfiniteBox)
  xExt = xInt * ratioInf;
  yExt = yInt * ratioInf;
  zExt = zInt * ratioInf;

  lc2inf = lc1inf;
  pp1 = newp; Point (pp1) = {xCnt-xExt, yCnt-yExt, zCnt-zExt, lc2inf};
  pp2 = newp; Point (pp2) = {xCnt+xExt, yCnt-yExt, zCnt-zExt, lc2inf};
  pp3 = newp; Point (pp3) = {xCnt+xExt, yCnt+yExt, zCnt-zExt, lc2inf};
  pp4 = newp; Point (pp4) = {xCnt-xExt, yCnt+yExt, zCnt-zExt, lc2inf};
  pp5 = newp; Point (pp5) = {xCnt-xExt, yCnt-yExt, zCnt+zExt, lc2inf};
  pp6 = newp; Point (pp6) = {xCnt+xExt, yCnt-yExt, zCnt+zExt, lc2inf};
  pp7 = newp; Point (pp7) = {xCnt+xExt, yCnt+yExt, zCnt+zExt, lc2inf};
  pp8 = newp; Point (pp8) = {xCnt-xExt, yCnt+yExt, zCnt+zExt, lc2inf};

  pnt[]={p1,p2,p3,p4,pp1,pp2,pp3,pp4}; Call Cuboid;
  VolInf[] = v;
  // FIXME seems to be forbidden to have the command "VolInf[] = v;"
  // on the same line as "Call Cuboid"
  pnt[]={p5,p6,p7,p8,pp5,pp6,pp7,pp8}; Call Cuboid;
  VolInf[] += v;
  pnt[]={p1,p2,p6,p5,pp1,pp2,pp6,pp5}; Call Cuboid;
  VolInf[] += {v};
  pnt[]={p3,p4,p8,p7,pp3,pp4,pp8,pp7}; Call Cuboid;
  VolInf[] += {v};
  pnt[]={p2,p3,p7,p6,pp2,pp3,pp7,pp6}; Call Cuboid;
  VolInf[] += {v};
  pnt[]={p4,p1,p5,p8,pp4,pp1,pp5,pp8}; Call Cuboid;
  VolInf[] += {v};

  Physical Volume("InfiniteX", 1) = { VolInf[4], VolInf[5] };
  Physical Volume("InfiniteY", 2) = { VolInf[2], VolInf[3] };
  Physical Volume("InfiniteZ", 3) = { VolInf[0], VolInf[1] };
EndIf

pnt[]={p1,p2,p3,p4,p5,p6,p7,p8}; Call Cuboid;
InteriorInfBox = v;

For num In {0:#VolInf()-1}
   Printf("VolInf %5g", VolInf[num]);
EndFor

/*
CTR_BOX = 0;
xpos =  60;
xneg = -60;
ypos =  0.8;
yneg = -0.8;
zpos =  100;
zneg = -30;
sz_mesh = 1e-12;
mp1 = newp; Point (mp1) = {CTR_BOX+xneg, CTR_BOX+yneg, CTR_BOX+zneg, sz_mesh};
mp2 = newp; Point (mp2) = {CTR_BOX+xpos, CTR_BOX+yneg, CTR_BOX+zneg, sz_mesh};
mp3 = newp; Point (mp3) = {CTR_BOX+xpos, CTR_BOX+ypos, CTR_BOX+zneg, sz_mesh};
mp4 = newp; Point (mp4) = {CTR_BOX+xneg, CTR_BOX+ypos, CTR_BOX+zneg, sz_mesh};
mp5 = newp; Point (mp5) = {CTR_BOX+xneg, CTR_BOX+yneg, CTR_BOX+zpos, sz_mesh};
mp6 = newp; Point (mp6) = {CTR_BOX+xpos, CTR_BOX+yneg, CTR_BOX+zpos, sz_mesh};
mp7 = newp; Point (mp7) = {CTR_BOX+xpos, CTR_BOX+ypos, CTR_BOX+zpos, sz_mesh};
mp8 = newp; Point (mp8) = {CTR_BOX+xneg, CTR_BOX+ypos, CTR_BOX+zpos, sz_mesh};

pnt[]={mp1,mp2,mp3,mp4,mp5,mp6,mp7,mp8}; Call Cuboid;
myBox = v;
Physical Volume("myBox",1000) = {myBox};
*/
```

Listing C.1: InfiniteBox.geo

```
mm = 1.e-3;
```

```
deg = Pi/180.;

DefineConstant[
  NumMagnets = {0, Min 1, Max 20, Step 1, Name "Parameters/0Number of magnets"}
  Flag_InfiniteBox = {1, Choices{0,1}, Name "Infinite box/Add infinite box"}
  Flag_FullMenu = {0, Choices{0,1}, Name "Parameters/Show all parameters"}
];
/* source: https://de.wikipedia.org/wiki/Magnetische_Permeabilit%C3%A4t
================================================================================
Medium                              r                                Einteilung
================================================================================
Supraleiter 1. Art        |   0                              |   ideal diamagnetisch
Blei, Zinn                 |   < 1 (ca. 0,999   )        |   diamagnetisch
Kupfer                     |   0,9999936 = 1     6,4   106    |    diamagnetisch
Wasserstoff                |   1     2,061   109               |   diamagnetisch
Wasser                     |   0,999991 = 1     9   106        |   diamagnetisch
Vakuum                     |   1                               |   (neutral)
Polyethylen                |   ~1                              |   (neutral)
Luft                    |   ca. 1 + 4   107                |   paramagnetisch
Aluminium               |   1 + 2,2   105                  |   paramagnetisch
Platin                     |   1+ 2,57   104                   |   paramagnetisch
Kobalt                     |   80  200                         |   ferromagnetisch
Eisen                      |   300  10  .000                   |   ferromagnetisch
Ferrite                    |   4  15  .000                     |   ferromagnetisch
Mumetall (NiFe)            |   50.000  140  .000               |   ferromagnetisch
amorphe Metalle            |   700    500  .000                |   ferromagnetisch
nanokristalline Metalle   |   20.000  150  .000               |   ferromagnetisch
================================================================================
 */
Br_mag = 0.4; // Remanenz in T
Hc_mag = 190; // Koerzitiffeldstaerke in kA/m
MUMAG = Br_mag/Hc_mag/(1.25663706212*10e-6);
FNAME~{1}="CONSTR/magnet.step";
FNAME~{2}="CONSTR/shaft.step";
FNAME~{3}="CONSTR/array.step";
NumFiles = 3;

// Konstanten f r die Erstellung einer Step-Datei
      For i In {1:NumFiles}
  DefineConstant[
     Xr~{i} = {0, Min 0, Max 360, Step 1,// Visible Flag_FullMenu,
                           Name Sprintf("Parameters/STEP %g/2X rotation [deg]",i) },
     Yr~{i} = {0, Min 0, Max 360, Step 1, Visible Flag_FullMenu,
                           Name Sprintf("Parameters/STEP %g/2Y rotation [deg]", i) },
     Zr~{i} = {0, Min 0, Max 360, Step 1,
                           Name Sprintf("Parameters/STEP %g/2Z rotation [deg]", i) },
     MUR~{i}= {(i==3)?1.:(i==1)?MUMAG:1000.,
                              Name Sprintf("Parameters/STEP %g/3Mu relative []", i)},
     BR~{i} = {(i==1)?Br_mag:0.0  ,
                              Name Sprintf("Parameters/STEP %g/3Br [T]", i)}
  ];
EndFor

//The geometrical parameters of the Infinite box.
DefineConstant[
  xInt = {1, Name "Infinite box/xInt", Visible 0}
  yInt = {1, Name "Infinite box/yInt", Visible 0}
  zInt = {1, Name "Infinite box/zInt", Visible 0}
  xExt = {xInt*2, Name "Infinite box/xExt", Visible 0}
  yExt = {yInt*2, Name "Infinite box/yExt", Visible 0}
  zExt = {zInt*2, Name "Infinite box/zExt", Visible 0}
  xCnt = {0, Name "Infinite box/xCenter", Visible 0}
  yCnt = {0, Name "Infinite box/yCenter", Visible 0}
  zCnt = {0, Name "Infinite box/zCenter", Visible 0}
];
```

Listing C.2: magnets_common.pro

```
SetFactory("OpenCASCADE");

Include "magnets_common.pro";

mm = 1.e-3;

//SetFactory("OpenCASCADE");
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Load a STEP file (using `ShapeFromFile' instead of `Merge' allows to directly
// retrieve the tags of the highest dimensional imported entities):
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// set some global Gmsh options

Mesh.Optimize       = 1; // optimize quality of tetrahedra
Mesh.OptimizeNetgen = 1;
Mesh.VolumeEdges    = 0; // Toggle mesh display
Mesh.SurfaceEdges   = 0;
Mesh.AngleSmoothNormals       = 1;

Solver.AutoMesh = 1; // always remesh if necessary (don't reuse mesh on disk)

For i In {1:NumFiles}
     //iLabel = newv;
     iLabel = ShapeFromFile(FNAME~{i});
     Physical Volume(Sprintf("Magnet_%g",i),10*i) = { iLabel };
     skin~{i}[] = CombinedBoundary{ Volume{ iLabel }; };
     Physical Surface(Sprintf("SkinMagnet_%g",i),10*i+1) = -skin~{i}[]; // magnet skin
     VolMagnets[] += { iLabel };
EndFor

Include "InfiniteBox.geo";

// The overall dimensions of the model have been calculated in InfiniteBox.geo
// So we use to characteristic length set for the infinite box for the whole mesh.
Mesh.CharacteristicLengthMin = lc1inf;
Mesh.CharacteristicLengthMax = lc1inf;


AirBox[] = BooleanDifference{
  Volume{ InteriorInfBox }; Delete;
}{
  Volume{ VolMagnets[] };
};

Physical Volume("AirBox",4) = { AirBox[] };




Volumes[] = { VolInf[], VolMagnets[], AirBox[]};

vv[] = BooleanFragments{
  Volume { Volumes[] }; Delete; }{};

If( #vv[] > #Volumes[] )
  Error("Overlapping magnets");
  Abort;
EndIf

Printf("Check whether BooleanFragments has preserved volume numbering:");
For num In {0:#vv()-1}
   Printf("Fragment %5g -> %g", Volumes[num], vv(num));
EndFor
```

```
Outer[] = CombinedBoundary{ Volume{ Volumes[] }; };
Physical Surface("OuterSurface", 5) = { Outer[]  };//+
MeshSize {969, 968, 808} = 0.1;
//+
MeshSize {953, 952, 954, 847, 845} = 0.1;
//+
Field[1] = MathEval;
//+
Field[2] = Restrict;
//+
Field[2].SurfacesList = {};
//+
Field[2].VolumesList = {3};
//+
Field[1].F = "0.05";
//+
Background Field = 1;
//+
Background Field = -1;
```

Listing C.3: magnets.geo

```
/* -------------------------------------------------------------------
   Tutorial 9 : 3D magnetostatic dual formulations and magnetic forces

   Features:
   - 3D Magnetostatics
   - Dual vector and scalar magnetic potentials formulations
   - Boundary condition at infinity with infinite elements
   - Maxwell stress tensor and rigid-body magnetic forces

   To compute the solution in a terminal:
   First generate the (3D) mesh and then run getdp with the chosen resolution
       gmsh magnets.geo -3
       getdp magnets.pro -solve MagSta_a
       OR
       getdp magnets.pro -solve MagSta_phi

   To compute the solution interactively from the Gmsh GUI:
       File > Open > magnets.pro
       Resolution can be chosen from the menu on the left:
       MagSta_a (default) or MagSta_phi
       Run (button at the bottom of the left panel)
   ------------------------------------------------------------------- */

/*
 This tutorial solves the electromagnetic field and the rigid-body forces acting
 on a set of magnetic pieces of either parallelepipedic or cylindrical shape.
 Besides position and dimension, each piece is attributed a (constant) magnetic
 permeability and/or a remanence field.  Hereafter, the pieces are all, simply
 though imprecisely, referred to as "Magnet", irresective of whether they are
 truly permanent magnets or ferromagnetic barrels.

 The tutorial model proposes two dual 3D magnetostatic formulations:

 - the magnetic vector potential formulation with spanning-tree gauging;
 - the scalar magnetic potential formulation.

 As there are no conductors, the later is rather simple. The source field "hs"
 is directly the the known coercive field hc[]:

   h = hs - grad phi   ,  hs = -hc.

 If the "Add infinite box" box is ticked, a transformation to infinity shell is
 used to impose the exact zero-field boundary condition at infinity.  See also
 Tutorial 2: magnetostatic field of an electromagnet. The shell is generated
 automatically by including "InfiniteBox.geo" at the end of the geometrical
```

```
 description of the model. It can be placed rather close of the magnets without
 loss of accuracy.

 The preferred way to compute electromagnetic forces in GetDP is as an explicit
 by-product of the Maxwell stress tensor "TM[{b}]", which is a material
 dependent function of the magnetic induction "b" field.  The magnetic force
 acting on a rigid body in empty space can be evaluated as the flux of the
 Maxwell stress tensor through a surface "S" (surrounding the body).  A special
 auxiliary function "g(S)" linked "S" is defined for each magnet, i.e.
 "g(SkinMagnet~{i}) = un~{i}".  The resultant magnetic force acting on
 "Magnet~{i}" is given by the integral:

 f~{i} = Integral [ TM[{b}] * {-grad un~{i}} ] ;

 This approach is analogous to the computation of heat flux "q(S)" through a
 surface "S" described in "Tutorial 5: thermal problem with contact
 resistances".

 Note that the Maxwell stress tensor is always discontinuous on material
 discontinuities, and that magnetic forces acting on rigid bodies depend only on
 the Maxwell stress tensor in empty space, and on the "b" and "h" field
 distribution, on the external side of "SkinMagnet~{i}" (side of the surface in
 contact with air).

 "{-grad un~{i}}" in the above formula can be regarded as the normal vector to
 "SkinMagnet~{i}" in the one element thick layer "layer~{i}" of finite elements
 around "Magnet~{i}", and "f~{i}", is thus indeed the flux of "TM[]" through the
 surface of "Magnet~{i}".

 The support of "{-grad un~{i}}" is limited to "layer~{i}", which is much
 smaller than "AirBox".  To speed up the computation of forces, a special domain
 "Vol_Force" for force integrations is defined, which contains only the layers
 "layer~{i}" of all magnets.
*/

// cut plane in z-direction with n steps and a start and stop position
  Zstart       = -10;
  Zstop   = 0;
  Zstep   = 1;
  // >> das ist die haelfte der groe e !
  szxy         = 15;
  npoints      = 66;

Include "magnets_common.pro"


mu0 = 4*Pi*1e-7;

//CHG = 1;//
     // hier fuer insgesamt fuenf Bauteile
     //MUR~{2} = CHG;
     //MUR~{3} = CHG;
     //MUR~{4} = CHG;
     //MUR~{5} = CHG;

     DefineConstant[
       // preset all getdp options and make them (in)visible
       R_ = {"MagSta_a", Name "GetDP/1ResolutionChoices", Visible 1,
       Choices {"MagSta_a", "MagSta_phi"}},
       C_ = {"-solve -v 3 -v2 -bin", Name "GetDP/9ComputeCommand", Visible 0}
       P_ = {"", Name "GetDP/2PostOperationChoices", Visible 0}
     ];

     Group{
       // Geometrical regions (give litteral labels to geometrical region numbers)
          domInfX = Region[1];
          domInfY = Region[2];
          domInfZ = Region[3];
          AirBox  = Region[4];
```

```
        //AirBox   = Region[1000];
        Outer   = Region[5];

        For i In {1:NumFiles}
             Magnet~{i} = Region[ {(10*i)}];
             SkinMagnet~{i} = Region[ {(10*i+1)} ];
             Layer~{i} =  Region[AirBox, OnOneSideOf SkinMagnet~{i}] ;
        EndFor

      // Abstract Groups (group geometrical regions into formulation relevant groups)
      Vol_Inf = Region[ {domInfX, domInfY, domInfZ} ];
      Vol_Air = Region[ {AirBox, Vol_Inf} ];

      Vol_Magnet = Region[{}];
      Sur_Magnet = Region[{}];
      Vol_Force  = Region[{}];
      For i In {1:NumFiles}
         Sur_Magnet += Region[SkinMagnet~{i}];
         Vol_Magnet += Region[Magnet~{i}];
         Vol_Layer  += Region[Layer~{i}];
      EndFor

      Vol_mu = Region[ {Vol_Air, Vol_Magnet}];

      Sur_Dirichlet_phi = Region[ Outer ];
      Sur_Dirichlet_a   = Region[ Outer ];

      Dom_Hgrad_phi = Region[ {Vol_Air, Vol_Magnet, Sur_Dirichlet_phi} ];
      Dom_Hcurl_a = Region[ {Vol_Air, Vol_Magnet, Sur_Dirichlet_a} ];
      Vol_Force = Region [ Vol_Layer ];
      //Vol_Force = Region [ Vol_Air ];
    }

    Function{
      mu[ Vol_Air ] = mu0;

         i = 1;
      For i In {1:NumFiles}
         // coercive field of magnets
         DefineConstant[
           HC~{i} = {-BR~{i}/mu0,
               Name Sprintf("Parameters/Magnet %g/0Coercive magnetic field [Am^-1]", i), Visible 0}
         ];
         hc[Magnet~{i}] = Rotate[Vector[0, HC~{i}, 0], Xr~{i}*deg, Yr~{i}*deg, Zr~{i}*deg];
         br[Magnet~{i}] = Rotate[Vector[0, BR~{i}, 0], Xr~{i}*deg, Yr~{i}*deg, Zr~{i}*deg];
         mu[Magnet~{i}] = mu0*MUR~{i};
      EndFor

      nu[] = 1.0/mu[];

      // Maxwell stress tensor (valid for both formulations and linear materials
      TM[] = ( SquDyadicProduct[$1] - SquNorm[$1] * TensorDiag[0.5, 0.5, 0.5] ) / mu[] ;
    }

    Jacobian {
      { Name Vol ;
         Case {
           { Region All ; Jacobian Vol ; }
           {Region domInfX; Jacobian VolRectShell {xInt,xExt,1,xCnt,yCnt,zCnt};}
           {Region domInfY; Jacobian VolRectShell {yInt,yExt,2,xCnt,yCnt,zCnt};}
           {Region domInfZ; Jacobian VolRectShell {zInt,zExt,3,xCnt,yCnt,zCnt};}
         }
      }
    }

    Integration {
      { Name Int ;
         Case {
           { Type Gauss ;
```

```
            Case {
185           { GeoElement Triangle    ; NumberOfPoints 4 ; }
              { GeoElement Quadrangle  ; NumberOfPoints 4 ; }
                  { GeoElement Tetrahedron ; NumberOfPoints 4 ; }
              { GeoElement Hexahedron  ; NumberOfPoints  6 ; }
              { GeoElement Prism       ; NumberOfPoints  6 ; }
190         }
            }
          }
        }
195
        Constraint {
          { Name phi ;
            Case {
              { Region Sur_Dirichlet_phi ; Value 0. ; }
200         }
          }
          { Name a ;
            Case {
              { Region Sur_Dirichlet_a ; Value 0. ; }
205         }
          }
          { Name GaugeCondition_a ; Type Assign ;
            Case {
              { Region Dom_Hcurl_a ; SubRegion Sur_Dirichlet_a ; Value 0. ; }
210         }
          }
          For i In {1:NumFiles}
            { Name Magnet~{i} ;
              Case {
215             { Region SkinMagnet~{i} ; Value 1. ; }
              }
            }
          EndFor
        }
220
        FunctionSpace {
          { Name Hgrad_phi ; Type Form0 ; // magnetic scalar potential
            BasisFunction {
              { Name sn ; NameOfCoef phin ; Function BF_Node ;
225             Support Dom_Hgrad_phi ; Entity NodesOf[ All ] ; }
            }
            Constraint {
              { NameOfCoef phin ; EntityType NodesOf ; NameOfConstraint phi ; }
            }
230       }
          { Name Hcurl_a; Type Form1; // magnetic vector potential
            BasisFunction {
              { Name se;  NameOfCoef ae;  Function BF_Edge;
            Support Dom_Hcurl_a ;Entity EdgesOf[ All ]; }
235         }
            Constraint {
              { NameOfCoef ae;  EntityType EdgesOf ; NameOfConstraint a; }
              { NameOfCoef ae;  EntityType EdgesOfTreeIn ; EntitySubType StartingOn ;
                NameOfConstraint GaugeCondition_a ; }
240         }
          }
          // auxiliary field on layer of elements touching each magnet, for the
          // accurate integration of the Maxwell stress tensor (using the gradient of
          // this field)
245       For i In {1:NumFiles}
            { Name Magnet~{i} ; Type Form0 ;
              BasisFunction {
                { Name sn ; NameOfCoef un ; Function BF_GroupOfNodes ;
                  Support Vol_Air ; Entity GroupsOfNodesOf[ SkinMagnet~{i} ] ; }
250           }
              Constraint {
                { NameOfCoef un ; EntityType GroupsOfNodesOf ; NameOfConstraint Magnet~{i} ; }
```

```
          }
        }
    EndFor
  }

  Formulation {
    { Name MagSta_phi ; Type FemEquation ;
      Quantity {
        { Name phi ; Type Local ; NameOfSpace Hgrad_phi ; }
        For i In {1:NumFiles}
          { Name un~{i} ; Type Local ; NameOfSpace Magnet~{i} ; }
        EndFor
      }
      Equation {
        Galerkin { [ - mu[] * Dof{d phi} , {d phi} ] ;
          In Vol_mu ; Jacobian Vol ; Integration Int ; }
        Galerkin { [ - mu[] * hc[] , {d phi} ] ;
          In Vol_Magnet ; Jacobian Vol ; Integration Int ; }
        For i In {1:NumFiles} // dummy term to define dofs for fully fixed space
        i = 1;
          Galerkin { [ 0 * Dof{un~{i}} , {un~{i}} ] ;
            In Vol_Air ; Jacobian Vol ; Integration Int ; }
        EndFor
      }
    }
    { Name MagSta_a; Type FemEquation ;
      Quantity {
        { Name a  ; Type Local  ; NameOfSpace Hcurl_a ; }
        For i In {1:NumFiles}
          { Name un~{i} ; Type Local ; NameOfSpace Magnet~{i} ; }
        EndFor
      }
      Equation {
        Galerkin { [ nu[] * Dof{d a} , {d a} ] ;
          In Vol_mu ; Jacobian Vol ; Integration Int ; }
        Galerkin { [ nu[] * br[] , {d a} ] ;
          In Vol_Magnet ; Jacobian Vol ; Integration Int ; }
        For i In {1:NumFiles}
        // dummy term to define dofs for fully fixed space
          Galerkin { [ 0 * Dof{un~{i}} , {un~{i}} ] ;
            In Vol_Air ; Jacobian Vol ; Integration Int ; }
        EndFor
      }
    }
  }

  Resolution {
    { Name MagSta_phi ;
      System {
        { Name A ; NameOfFormulation MagSta_phi ; }
      }
      Operation {
        Generate[A] ; Solve[A] ; SaveSolution[A] ;
        PostOperation[MagSta_phi] ;
      }
    }
    { Name MagSta_a ;
      System {
        { Name A ; NameOfFormulation MagSta_a ; }
      }
      Operation {
        Generate[A] ; Solve[A] ; SaveSolution[A] ;
        PostOperation[MagSta_a] ;
      }
    }
  }

  PostProcessing {
    { Name MagSta_phi ; NameOfFormulation MagSta_phi ;
```

```
        Quantity {
          { Name b   ;
        Value { Local { [ - mu[] * {d phi} ] ; In Dom_Hgrad_phi ; Jacobian Vol ; }
                  Local { [ - mu[] * hc[] ]    ; In Vol_Magnet ; Jacobian Vol ; } } }
          { Name h   ;
        Value { Local { [ - {d phi} ]        ; In Dom_Hgrad_phi ; Jacobian Vol ; } } }
          { Name hc  ; Value { Local { [ hc[] ]  ; In Vol_Magnet ; Jacobian Vol ; } } }
          { Name phi ; Value { Local { [ {phi} ] ; In Dom_Hgrad_phi ; Jacobian Vol ; } } }
          For i In {1:NumFiles}
             { Name un~{i} ; Value { Local { [ {un~{i}} ] ; In Vol_Force ; Jacobian Vol ; } } }
             { Name f~{i} ; Value { Integral { [ - TM[-mu[] * {d phi}] * {d un~{i}} ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fx~{i} ; Value { Integral { [ CompX[- TM[-mu[] * {d phi}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fy~{i} ; Value { Integral { [ CompY[- TM[-mu[] * {d phi}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fz~{i} ; Value { Integral { [ CompZ[- TM[-mu[] * {d phi}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
          EndFor
        }
      }
      { Name MagSta_a ; NameOfFormulation MagSta_a ;
        PostQuantity {
          { Name b ; Value { Local { [ {d a} ]; In Dom_Hcurl_a ; Jacobian Vol; } } }
          { Name a ; Value { Local { [ {a} ]; In Dom_Hcurl_a ; Jacobian Vol; } } }
          { Name br ; Value { Local { [ br[] ]; In Vol_Magnet ; Jacobian Vol; } } }
          For i In {1:NumFiles}
             { Name un~{i} ; Value { Local { [ {un~{i}} ] ; In Dom_Hcurl_a ; Jacobian Vol ; } } }
             { Name f~{i} ; Value { Integral { [ - TM[{d a}] * {d un~{i}} ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fx~{i} ; Value { Integral { [ CompX[- TM[{d a}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fy~{i} ; Value { Integral { [ CompY[- TM[{d a}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
             { Name fz~{i} ; Value { Integral { [ CompZ[- TM[{d a}] * {d un~{i}} ] ] ;
                    In Vol_Force ; Jacobian Vol ; Integration Int ; } } }
          EndFor
        }
      }
    }

    PostOperation {
      { Name MagSta_phi ; NameOfPostProcessing MagSta_phi;
        Operation {
          Print[ b, OnElementsOf Vol_mu, File "RESULTS/b.pos" ] ;
          Print[ b, OnElementsOf Vol_mu, Format SimpleTable,  File >> "RESULTS/b.txt" ] ;

          M = 0;
          For i In {Zstart : Zstop : Zstep}
                  Print[ b, OnPlane{
                       {-szxy,-szxy,i} {-szxy,szxy,i} {szxy,-szxy,i}
                  } {npoints, npoints},File "RESULTS/b_cut.pos"];
                  Print[ b, OnPlane{
                       {-szxy,-szxy,i} {-szxy,szxy,i} {szxy,-szxy,i}
                  } {npoints, npoints},Format SimpleTable,File Sprintf("RESULTS/%g_bcut.txt", M)];
                  /*Print[ b, OnPlane{
                       {-szxy,-szxy,zpos} {-szxy,szxy,zpos} {szxy,-szxy,zpos}
                  } {npoints, npoints},Format SimpleTable,File Sprintf("RESULTS/%g_bcut.txt", M)];*/
                  M = M+1;
          EndFor
          Echo[ Str["l=PostProcessing.NbViews-1;",
             "View[l].ArrowSizeMax = 100;",
             "View[l].CenterGlyphs = 1;",
             "View[l].VectorType = 1;" ] ,
             File "RESULTS/tmp.geo", LastTimeStepOnly] ;
        }
      }
      { Name MagSta_a ; NameOfPostProcessing MagSta_a ;
        Operation {
```

```
            /*Print[ b, OnPlane{
                    {-szxy,-szxy,zpos} {-szxy,szxy,zpos} {szxy,-szxy,zpos}
               } {npoints, npoints},File "RESULTS/b_cut.pos"];*/

            M = 0;
            For i In {Zstart : Zstop : Zstep}
                    Print[ b, OnPlane{
                         {-szxy,-szxy,i} {-szxy,szxy,i} {szxy,-szxy,i}
                    } {npoints, npoints},File "RESULTS/b_cut.pos"];
                    Print[ b, OnPlane{
                         {-szxy,-szxy,i} {-szxy,szxy,i} {szxy,-szxy,i}
                    } {npoints, npoints},Format SimpleTable,File Sprintf("RESULTS/%g_mur_%g_bcut.txt",M,MUR~{2})];
                    /*Print[ b, OnPlane{
                         {-szxy,-szxy,zpos} {-szxy,szxy,zpos} {szxy,-szxy,zpos}
                    } {npoints, npoints},Format SimpleTable,File Sprintf("RESULTS/%g_bcut.txt", M)];*/
                    M = M+1;
            EndFor
            /*
               Make a cut through the z direction in in x-plane
               from -y to y   => -40 to 40
            */
            Print[ b, OnPlane{
               {0,-40,-20} {0,-40,5} {0,40,-20}
          } {npoints, npoints},File "RESULTS/b_cut_Z.pos"];
            Print[ b, OnPlane{
               {0,-40,-20} {0,-40,5} {0,40,-20}
          } {npoints, npoints},Format SimpleTable,File Sprintf("RESULTS/z_cut_mur_%g.txt",MUR~{2})];

            Print[ b,  OnElementsOf Vol_mu,  File "RESULTS/b.pos" ];
            Print[ b,  OnElementsOf Vol_mu,  Format Table,  File >> "RESULTS/b.txt" ];

            Echo[ Str["l=PostProcessing.NbViews-1;",
               "View[l].ArrowSizeMax = 100;",
               "View[l].CenterGlyphs = 1;",
               "View[l].VectorType = 1;" ] ,
               File "RESULTS/tmp.geo", LastTimeStepOnly] ;
            Print[ br,  OnElementsOf Vol_Magnet,  File "RESULTS/br.pos" ];
            Print[ br,  OnElementsOf Vol_Magnet,Format SimpleTable,  File "RESULTS/br.txt" ];
            Print[ a,  OnElementsOf Vol_mu,  File "RESULTS/a.pos" ];
            /*For i In {1:NumFiles}
            //Print[ un~{i}, OnElementsOf Domain, File "RESULTS/un.pos"  ];
               Print[ f~{i}[Vol_Air], OnGlobal, Format Table, File > "RESULTS/F.dat"  ];
               Print[ fx~{i}[Vol_Air], OnGlobal, Format Table, File > "RESULTS/Fx.dat"];
               //SendToServer Sprintf("Output/Magnet %g/X force [N]", i), Color "Ivory"  ];
               Print[ fy~{i}[Vol_Air], OnGlobal, Format Table, File > "RESULTS/Fy.dat"];
               //SendToServer Sprintf("Output/Magnet %g/Y force [N]", i), Color "Ivory"  ];
               Print[ fz~{i}[Vol_Air], OnGlobal, Format Table, File > "RESULTS/Fz.dat"];
               //SendToServer Sprintf("Output/Magnet %g/Z force [N]", i), Color "Ivory"  ];
            EndFor*/
          }
        }
      }
```

Listing C.4: magnets.pro

D Matlab-Code für Signalverarbeitung

```
clc ;clear; close all;

%% generating filenames
N_step = 8;
d_step = 360 / N_step;
deg = d_step:d_step:360;
deg_padded = string(zeros(1,length(deg)));
% Z-Axis step in the FEM simulation
level = 4;

for i=1:length(deg)
    deg_padded(i) = sprintf('%03d',deg(i));
end
%% access to simulation results
% ramp
str = 'rotations/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% sine
%str = 'RES_SINE/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% single sim test
%str = 'RESULTS/'+string(level)+'_mur_1000_bcut.txt';

disp('Path of simulation results :')
disp(str)

% number of spectral lines for center of gravity
nl=10;

% center of gravity array
cog = zeros(N_step,0);

figure()
tiledlayout(4,8)

%% sweep of over rotation angles
for i=1:1:N_step
%% load and reshape
disp(['Rotation angle ' num2str(i*d_step,3)])
f = load(str(i));
X = reshape(f(:,1),67,67);
Y = reshape(f(:,2),67,67);
Z = reshape(f(:,3),67,67);
U = reshape(f(:,4),67,67);
V = reshape(f(:,5),67,67);
W = reshape(f(:,6),67,67);

% display raw data
nexttile
pcolor(X,Y,U);
colorbar;
shading interp
title(deg_padded(i))

%% edit data
% substitute variable
u = U;
```

```
% increase contrast
minval = min(U,[],'all');
maxval = max(U,[],'all');
valrange = abs(minval-maxval);
u(U>(minval+0.55*valrange)) = maxval;
u(U<(minval+0.47*valrange)) = minval;
%%
% center around zero
u = u + (valrange/2 - maxval);

% window
w = flattopwin(length(U));
u = u.*w';

% display edited data
nexttile
pcolor(X,Y,u);
colorbar;
shading interp
%%
% center line average
uc = mean(u(30:38,:));

% smooth data
uc = smoothdata(uc,'movmedian');

% display edited center line
nexttile
plot(uc);

%% show fft
% get fft
fU = abs(fft(uc));

% center of gravity
spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

% save center of gravity to results
cog(i) = spx;

%display fft stem
nexttile
stem(1:nl,fU(1:nl));

% display center of gravity
hold on;
stem(spx,mean(fU(1:nl)),'x');

end

%% show center of gravity
figure()
plot(deg,cog);

% get range
cogmin = min(cog);
cogmax = max(cog);
frange = abs(cogmin - cogmax);
cogrange = frange + frange/7;
realmin = cogmin - abs(cogrange - frange);

%% determine angle from test
clc;
tstr = 'RESULTS/'+string(level)+'_mur_1000_bcut.txt';
figure()
tiledlayout('flow')

f = load(tstr);
```

```
X = reshape(f(:,1),67,67);
Y = reshape(f(:,2),67,67);
Z = reshape(f(:,3),67,67);
U = reshape(f(:,4),67,67);
V = reshape(f(:,5),67,67);
W = reshape(f(:,6),67,67);

% display raw data
nexttile
pcolor(X,Y,U);
colorbar;
shading interp

% substitute variable
u = U;

% increase contrast
minval = min(U,[],'all');
maxval = max(U,[],'all');
valrange = abs(minval-maxval);
u(U>(minval+0.55*valrange)) = maxval;
u(U<(minval+0.47*valrange)) = minval;

% center around zero
u = u + (valrange/2 - maxval);

% window
w = flattopwin(length(U));
u = u.*w';

% display edited data
nexttile
pcolor(X,Y,u);
colorbar;
shading interp

% center line average
uc = mean(u(30:38,:));

% smooth data
uc = smoothdata(uc,'movmedian');

% display edited center line
nexttile
plot(uc);

% get fft
fU = abs(fft(uc));

%display fft stem
nexttile
stem(1:nl,fU(1:nl));

% center of gravity
spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

% display center of gravity
hold on;
stem(spx,mean(fU(1:nl)),'x');

%calculate angle
test_angle = ((spx+(-1*realmin))/cograngе)*360;
disp(['Rotation angle result: ' num2str(test_angle)])
```

Listing D.1: processing_1.m

```
clc ;clear; close all;
```

```
%% generating filenames
N_step = 8;
d_step = 360 / N_step;
deg = d_step:d_step:360-d_step;
deg_padded = string(zeros(1,length(deg)));
% Z-Axis step in the FEM simulation
level = 4;

for i=1:length(deg)
    deg_padded(i) = sprintf('%03d',deg(i));
end
%% access to simulation results
% ramp
str = 'testtwo/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% sine
%str = 'RES_SINE/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% single sim test
%str = 'RESULTS/'+string(level)+'_mur_1000_bcut.txt';

disp('Path of simulation results :')
disp(str)

% number of spectral lines for center of gravity
nl = 5;

% number of extra center lines to average over
nc = 18;

% polynomial degree
pd = 1;

% center of gravity array
cog = zeros(length(deg),0);

% figure()
% tiledlayout(4,8)

%% sweep of over rotation angles
for i=1:1:length(deg)
%% load and reshape
disp(['Rotation angle ' num2str(i*d_step,3)])
f = load(str(i));
X = reshape(f(:,1),67,67);
Y = reshape(f(:,2),67,67);
Z = reshape(f(:,3),67,67);
U = reshape(f(:,4),67,67);
V = reshape(f(:,5),67,67);
W = reshape(f(:,6),67,67);

% display raw data
% nexttile
% pcolor(X,Y,U);
% colorbar;
% shading interp
% title(deg_padded(i))

%% edit data
% substitute variable
u = U;

% increase contrast
% minval = min(U,[],'all');
% maxval = max(U,[],'all');
minval = -4e-3;
maxval = 0;
valrange = abs(minval-maxval);
u(U>(minval+0.8*valrange)) = maxval;
u(U<(minval+0.8*valrange)) = minval;
%%
```

```
% center around zero
u = u + (valrange/2 - maxval);

% window
% w = ones(1,length(U));
% w = sin(linspace(0,pi,length(U)));
w = blackmanharris(length(U))';
% w = flattopwin(length(U))';
u = u.*w;

% display edited data
% nexttile
% pcolor(X,Y,u);
% colorbar;
% shading interp
%%
% center line average
if nc == 0
    uc = u(34,:);
else
    uc = mean(u(34-nc/2:34+nc/2,:));
end
uc = smoothdata(uc,'movmedian');
uc = smoothdata(uc,'gaussian');

% minval = min(uc);
% maxval = max(uc);
% valrange = abs(minval-maxval);
% uc = uc * (4e-3/valrange);
% uc = uc + (valrange/2 - maxval);

% display edited center line
% nexttile
% plot(uc);

%% show fft
% get fft
fU = abs(fft(uc));

% center of gravity
spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

% save center of gravity to results
% if i ~= 8
    cog(i) = spx;
% end
%display fft stem
% nexttile
% stem(1:nl,fU(1:nl));

% display center of gravity
% hold on;
% stem(spx,mean(fU(1:nl)),'x');

end

%% show center of gravity
figure()
% plot(deg,cog);

% get range
cogmin = min(cog);
cogmax = max(cog);
frange = abs(cogmin - cogmax);
cogrange = frange + frange/7;
realmin = cogmin - abs(cogrange - frange);

%% polynomial
coeffs = polyfit(deg,cog,pd);
```

```
resx = linspace(0,360);
res = polyval(coeffs,resx);
scatter(deg,cog)
xlabel('deg')
ylabel('n')
grid on
hold on
plot(resx,res,'k')

%% determine angle from test
clc;
% tdeg = [1 5 10 20 30 60 70 80 100 110 120 140 160 240 260 280 300 320 330 340 350 355 359 360];
tdeg = 20:20:360;
% tdeg = [tdeg];
% tdeg=[];
tdeg_padded = string(zeros(1,length(tdeg)));
for i=1:length(tdeg)
    tdeg_padded(i) = sprintf('%03d',tdeg(i));
end

tstr = 'testtwo/'+tdeg_padded+'/'+level+'_mur_1000_bcut.txt';
lastspx = 0;
aa = zeros(1,length(tdeg));
er = zeros(1,length(tdeg));

for i = 1:1:length(tstr)
    f = load(tstr(i));
    X = reshape(f(:,1),67,67);
    Y = reshape(f(:,2),67,67);
    Z = reshape(f(:,3),67,67);
    U = reshape(f(:,4),67,67);
    V = reshape(f(:,5),67,67);
    W = reshape(f(:,6),67,67);

    % substitute variable
    u = U;

    % increase contrast
    minval = -4e-3;
    maxval = 0;
    valrange = abs(minval-maxval);
    u(U>(minval+0.8*valrange)) = maxval;
    u(U<(minval+0.8*valrange)) = minval;

    % center around zero
    u = u + (valrange/2 - maxval);

    % window
    u = u.*w;

%     if tdeg(i) == 350
%         figure();
%         pcolor(X,Y,u);
%         colorbar;
%         shading interp
%     end

    % center line average
    if nc == 0
        uc = u(34,:);
    else
        uc = mean(u(34-nc/2:34+nc/2,:));
    end

    % smooth data
    uc = smoothdata(uc,'movmedian');
    uc = smoothdata(uc,'gaussian');

    % get fft
```

```
    fU = abs(fft(uc));

    % center of gravity
    spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

    %calculate angle
    ws = warning('off','all');  % turn off centering warning
    coeffs = polyfit(deg,cog-spx,pd);
    warning(ws)  % turn warnings back on
    ta = roots(coeffs);
    plot(tdeg(i),spx,'xr');

    aa(i) = ta;
    er(i) = abs(aa(i)-tdeg(i));
    disp(['Angle: ' num2str(tdeg(i)) ' Result : ' num2str(aa(i)) ' Error: ' num2str(er(i))])
end

disp(['Average error: ' num2str(mean(er))])
```

Listing D.2: processing_2.m

```
clc ;clear; close all;

%% generating filenames
N_step = 8;
d_step = 360 / N_step;
deg = d_step:d_step:360-d_step;
deg = [5 deg 355];
deg = sort(deg);
deg_padded = string(zeros(1,length(deg)));
% Z-Axis step in the FEM simulation
level = 4;

for i=1:length(deg)
    deg_padded(i) = sprintf('%03d',deg(i));
end
%% access to simulation results
% ramp
str = 'testtwo/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% sine
%str = 'RES_SINE/'+deg_padded+'/'+level+'_mur_1000_bcut.txt';
% single sim test
%str = 'RESULTS/'+string(level)+'_mur_1000_bcut.txt';

disp('Path of simulation results :')
disp(str)

% number of spectral lines for center of gravity
nl = 5;

% number of extra center lines to average over
nc = 18;

% polynomial degree
pd = 7;

% center of gravity array
cog = zeros(length(deg),0);

% figure()
% tiledlayout(4,8)

%% sweep of over rotation angles
for i=1:1:length(deg)
%% load and reshape
disp(['Rotation angle ' num2str(i*d_step,3)])
f = load(str(i));
X = reshape(f(:,1),67,67);
```

```
Y = reshape(f(:,2),67,67);
Z = reshape(f(:,3),67,67);
U = reshape(f(:,4),67,67);
V = reshape(f(:,5),67,67);
W = reshape(f(:,6),67,67);

% display raw data
% nexttile
% pcolor(X,Y,U);
% colorbar;
% shading interp
% title(deg_padded(i))

%% edit data
% substitute variable
u = U;

% increase contrast
% minval = min(U,[],'all');
% maxval = max(U,[],'all');
minval = -4e-3;
maxval = 0;
valrange = abs(minval-maxval);
u(U>(minval+0.8*valrange)) = maxval;
u(U<(minval+0.8*valrange)) = minval;
%%
% center around zero
u = u + (valrange/2 - maxval);

% window
w = ones(1,length(U));
% w = sin(linspace(0,pi,length(U)));
% w = blackmanharris(length(U))';
% w = flattopwin(length(U))';
u = u.*w;

% display edited data
% nexttile
% pcolor(X,Y,u);
% colorbar;
% shading interp
%%
% center line average
if nc == 0
    uc = u(34,:);
else
    uc = mean(u(34-nc/2:34+nc/2,:));
end
uc = smoothdata(uc,'movmedian');
uc = smoothdata(uc,'gaussian');

% minval = min(uc);
% maxval = max(uc);
% valrange = abs(minval-maxval);
% uc = uc * (4e-3/valrange);
% uc = uc + (valrange/2 - maxval);

% display edited center line
% nexttile
% plot(uc);

%% show fft
% get fft
fU = abs(fft(uc));

% center of gravity
spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

% save center of gravity to results
```

```
% if i ~= 8
    cog(i) = spx;
% end
%display fft stem
% nexttile
% stem(1:nl,fU(1:nl));

% display center of gravity
% hold on;
% stem(spx,mean(fU(1:nl)),'x');

end

%% show center of gravity
figure()
% plot(deg,cog);

% get range
cogmin = min(cog);
cogmax = max(cog);
frange = abs(cogmin - cogmax);
cogrange = frange + frange/7;
realmin = cogmin - abs(cogrange - frange);

%% polynomial
coeffs = polyfit(deg,cog,pd);
resx = linspace(5,355);
res = polyval(coeffs,resx);
scatter(deg,cog)
xlabel('deg')
ylabel('n')
grid on
hold on
plot(resx,res,'k')

lcoeffs = polyfit([deg(end) deg(1)+360],[cog(end) min(cog)],1);
resxl = linspace(355,365);
resl = polyval(lcoeffs,resxl);
plot(resxl,resl,'k')

lcoeffs = polyfit([deg(end)-360 deg(1)],[cog(end) min(cog)],1);
resxl = linspace(-5,5);
resl = polyval(lcoeffs,resxl);
plot(resxl,resl,'k')

%% determine angle from test
clc;
% tdeg = [1 5 10 20 30 60 70 80 100 110 120 140 160 240 260 280 300 320 330 340 350 355 359 360];
tdeg = 20:20:360;
% tdeg = [tdeg];
% tdeg=[];
tdeg_padded = string(zeros(1,length(tdeg)));
for i=1:length(tdeg)
    tdeg_padded(i) = sprintf('%03d',tdeg(i));
end

tstr = 'testtwo/'+tdeg_padded+'/'+level+'_mur_1000_bcut.txt';
lastspx = 0;
aa = zeros(1,length(tdeg));
er = zeros(1,length(tdeg));

for i = 1:1:length(tstr)
    f = load(tstr(i));
    X = reshape(f(:,1),67,67);
    Y = reshape(f(:,2),67,67);
    Z = reshape(f(:,3),67,67);
    U = reshape(f(:,4),67,67);
    V = reshape(f(:,5),67,67);
    W = reshape(f(:,6),67,67);
```

```
    % substitute variable
    u = U;

    % increase contrast
    minval = -4e-3;
    maxval = 0;
    valrange = abs(minval-maxval);
    u(U>(minval+0.8*valrange)) = maxval;
    u(U<(minval+0.8*valrange)) = minval;

    % center around zero
    u = u + (valrange/2 - maxval);

    % window
    u = u.*w;

%     if tdeg(i) == 350
%         figure();
%         pcolor(X,Y,u);
%         colorbar;
%         shading interp
%     end

    % center line average
    if nc == 0
        uc = u(34,:);
    else
        uc = mean(u(34-nc/2:34+nc/2,:));
    end

    % smooth data
    uc = smoothdata(uc,'movmedian');
    uc = smoothdata(uc,'gaussian');

    % get fft
    fU = abs(fft(uc));

    % center of gravity
    spx=sum((1:nl).*fU(1:nl))/sum(fU(1:nl));

    %calculate angle
    ws = warning('off','all');  % turn off centering warning
    coeffs = polyfit(deg,cog-spx,pd);
    warning(ws)  % turn warnings back on
    ta = roots(coeffs);
    ta = ta(imag(ta)==0 & real(ta)>0 & real(ta)<=360);
    ta = flip(ta);
    if length(ta) < 2
        ta(2) = roots(polyfit([deg(end) deg(1)+360],[cog(end) min(cog)]-spx,1));
%         ta(2) = acos(((spx-cog(1))/(cog(end)-cog(1)))*2-1)*(360+deg(1)-deg(end))/pi+355;
        ta(ta>360) = ta(ta>360)-360;
    end

%     disp(['Angle: ' num2str(tdeg(i)) ' Result 1: ' num2str(ta(1)) ' Result 2: ' num2str(ta(2))])

    if spx > lastspx
        aa(i) = ta(1);
    else
        aa(i) = ta(2);
    end
    er(i) = abs(aa(i)-tdeg(i));
    disp(['Angle: ' num2str(tdeg(i)) ' Result : ' num2str(aa(i)) ' Error: ' num2str(er(i))])
    plot(tdeg(i),spx,'xr');
    lastspx = spx;
end

disp(['Average error: ' num2str(mean(er))])
```

Listing D.3: processing_3.m

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original