HAW HAMBURG

BACHELORTHESIS
Finn Lanz

Suchverfahren zur Positionserfassung eines Permanentmagneten mittels Sensor-Array

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

Finn Lanz

Suchverfahren zur Positionserfassung eines Permanentmagneten mittels Sensor-Array

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. Jürgen Vollmer

Eingereicht am: 15. September 2021

Finn Lanz

Thema der Arbeit

Suchverfahren zur Positionserfassung eines Permanentmagneten mittels Sensor-Array

Stichworte

Magnetfeld, magnetischer Dipol, Monte Carlo, Gradientenabstieg, inverses Problem, TMR-Effekt, TMR-Sensor, Sensor-Array, Algorithmusentwurf

Kurzzusammenfassung

Ziel der Bachelorthesis ist die Positionsbestimmung eines Permanentmagneten über einem Array aus magnetischen Sensoren. Dafür steht aus Vorarbeiten eine entsprechende Hardware aus TMR-Sensoren bereit, welche das Magnetfeld zweidimensional in der x-y-Ebene erfassen können und welche zudem einen Bereich mit linearer Feldabhängigkeit besitzen. Angestrebt wird die Minimierung der Differenz zwischen den aktuell gemessenen Werten (Vektorfeld) des Arrays und einem Zielmodell zur Bestimmung der Position.

Finn Lanz

Title of Thesis

Search method for detecting the position of a permanent magnet by means of sensor array

Keywords

Magnetic fields, magnetic dipole, Monte Carlo, Gradient descent, inverse problem, TMR-effect, TMR sensor sensor array, algorithm design

Abstract

The aim of this bachelor thesis is the position determination of a permanent magnet above an array of magnetic sensors. For this is a hardware made of TMR sensors available, which can detect the magnetic field two-dimensionally in the x-y-plane and also has a range of linear field dependence. The aim is to minimize the difference between the currently measured values (vector field) of the array and a target model to determine the position.

Vorwort

Zuallererst möchte ich mich bei meinem betreuenden Erstprüfer Herrn Prof. Dr-Ing. Karl-Ragmar Riemschneider dafür bedanken, dass er mir die Möglichkeit gegeben hat, meine Bachelorarbeit über diese Thematik schreiben zu können. Vielen Dank auch an Herrn Prof. Dr. Jürgen Vollmer für die Übernahme der Position des Zweitprüfers.

Besonderer Dank gilt Herrn M.Sc. Thorben Schüthe für seine Hilfe während der gesamten Bearbeitungszeit und für die zahlreichen Anregungen, die er mir dargeboten hat. Ich weiß seine investierte Zeit sehr zu schätzen.

Für das Korrekturlesen möchte ich auch Herrn Dipl.-Ing. Günter Müller meinen herzlichen Dank aussprechen.

Danke auch an meine Lerngruppe, die mich das ganze Studium über begleitet hat und ohne welche ich die Studienzeit bei weitem nicht so gut überstanden hätte, wie ich es getan habe.

Mein größter Dank gilt meiner Familie. Vielen Dank an meine Mutter und meiner Schwester für das wiederholte Korrekturlesen und dafür, dass ihr mir immer eine emotionale Stütze seid. Dafür, dass er mich auf diesen Weg gewiesen hat, möchte ich meinem Vater danken. Ich wünschte, du hättest diesen Tag miterleben können.

Also, noch einmal: **Vielen Dank!**

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Bereits seit geraumer Zeit werden magnetische Sensoren in der heutigen Technik genutzt, um berührungslose Messungen von Magnetfeldern zu ermöglichen. So werden diese beispielsweise in der Automobilindustrie eingesetzt, um Drehzahlen oder Winkel zu erfassen, was sich bei sicherheitsrelevanten Systemen, wie zum Beispiel dem Antiblockiersystem (ABS), als vorteilhaft herausstellt. Die HAW Hamburg arbeitet seit einigen Jahren mit an dem Forschungsprojekt **ISAR** (kurz für: Integrated Sensor-Arrays), welches das Ziel hat, auf dem Tunnel-Magneto-Widerstandseffekt (TMR-Effekt) beruhende Sensor-Arrays und die dazugehörige Signalverarbeitung zu entwerfen. Durch Verwendung immer präziserer Sensoren und durch die Weiterentwicklung bereits bestehender Systeme, wird eine immer genauere Erfassung von Magnetfeldern möglich. Dadurch eröffnen sich neue Möglichkeiten der Nutzung einer solchen Messeinrichtung, wie zum Beispiel das Lokalisieren eines Gebermagneten, welcher über dem Sensor-Array positioniert ist, was genau den Kernpunkt dieser Bachelorthesis darstellt.

1.1 Stand der Technik

Sensoren spielen eine tragende Rolle, wenn elektronische oder mechanische Systeme überwacht werden sollen. Sie werden zahlreich eingesetzt, um den Zustand von beispielsweise sicherheitsrelevanten Systemen zu überprüfen. Der Einsatz von magnetoresistiven Sensoren gewinnt immer mehr an Bedeutung, da sich wichtige Kenngrößen berührungslos erfassen lassen. Bisher beruhten viele magnetische Messeinrichtungen auf dem bereits länger bekannten Hall-Effekt oder auf dem anisotropen magnetoresistiven Effekt (AMR). Durch die Nutzung solcher Systeme ist es möglich, Magnetfelder aufzuspüren und sogar Winkel oder Drehzahlen zu messen. Durch die Verbesserung der genutzten Technik und dem Ausnutzen des TMR-Effektes, um welchen es sich in dieser Arbeit handelt, sind immer genauere und weniger störanfällige Messungen möglich, was neue Perspektiven für die Nutzung von Sensor-Arrays eröffnet.

1.2 Ziel dieser Arbeit

Das Ziel dieser Arbeit besteht darin, eine bereits bestehende Softwarestruktur plattformübergreifend in C++ zu implementieren. Dabei sollen die Messdaten eines Sensor-Arrays grafisch anschaulich dargestellt werden und mit Hilfe der aufgenommenen Messdaten und einem entworfenen Suchalgorithmus die Position eines Gebermagneten über dem Array bestimmt werden. Dem Anwender der Applikation sollen verschiedene Bedienmöglichkeiten zur Verfügung gestellt werden, um mit der grafischen Benutzeroberfläche zu interagieren und somit zum Beispiel die Art der Darstellung auswählen zu können oder Messdaten zu speichern. Diese Anwendung soll anschließend auf ihre Genauigkeit und Reliabilität überprüft werden, wofür im Anschluss verschiedene Messungen vorgenommen werden, bei denen im Mittelpunkt stehen soll, wie stark die Abweichungen zwischen tatsächlicher und berechneter Position am Ende sind. Damit lässt sich im Anschluss feststellen, wo sich die Grenzen der entwickelten Software befinden und in welchem Bereich über dem Array die Position präzise und zuverlässig berechnet werden kann. Schlussendlich ist das Gesamtziel der Bachelorthesis eine lauffähige plattformübergreifende Anwendung zu entwerfen, mit welcher sich der Ort eines Permanentmagneten über dem Sensor-Array ermitteln lässt. Um einen Überblick zu geben, welche Aufgaben es dabei zu erfüllen gilt, sind diese in Abbildung 1.1 dargestellt.

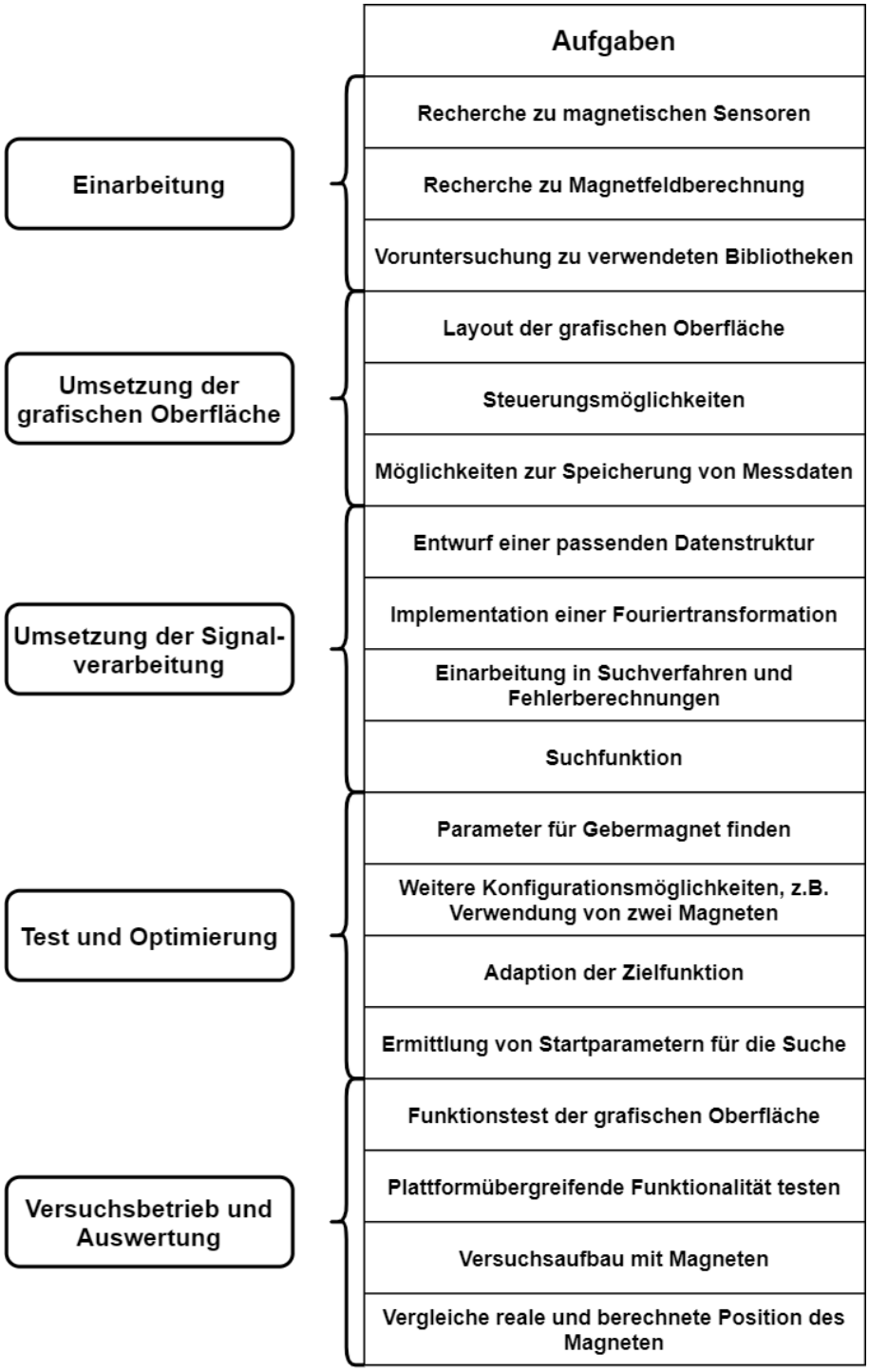


Abbildung 1.1: Übersicht über die zu erfüllenden Aufgaben.

2 Grundlagen

2.1 Magnetismus

Das Thema Magnetismus spielt in dieser Arbeit eine tragende Rolle, da über das Magnetfeld eines Permanentmagneten die Ausgangswerte des Sensor-Arrays berechnet und später mit Hilfe dieser Rückschlüsse auf dessen Lage und Beschaffenheit gezogen werden sollen. Deshalb ist es von Wichtigkeit, die grundlegenden und notwendigen physikalischen beziehungsweise mathematischen Grundlagen nachvollziehen zu können.

2.1.1 Der magnetische Dipol

Im Bereich des Magnetismus stellt der Dipol das einfachste Element dar, welches nur aus einem Nord- und einem Südpol besteht. Die Feldlinien eines Magnetfeldes verlaufen dabei innerhalb eines Dipols vom Süd zum Nordpol und außerhalb vom Nord- zum Südpol. Der Feldverlauf ist in Abbildung 2.1 dargestellt. Im späteren Verlauf wird der über dem Array positionierte Magnet als Dipol angenommen, um die Berechnungen seines Magnetfeldes zu vereinfachen.

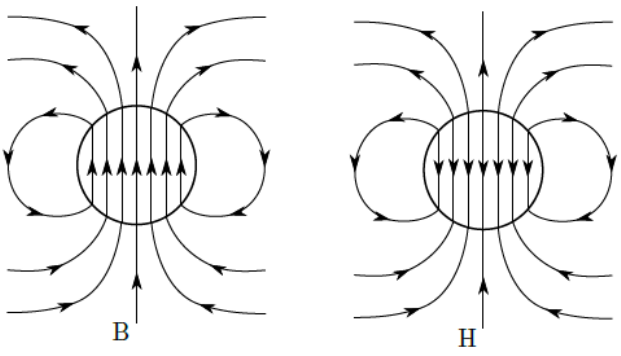


Abbildung 2.1: Feldverlauf eines einfachen magnetischen Dipols [19].

Mit Hilfe der Dipolgleichung lassen sich die magnetische Flussdichte $\underline{B}(\underline{r})$ (2.1) und die magnetische Feldstärke $\underline{H}(\underline{r})$ (2.2) an einem Punkt $\underline{r}$ berechnen.

$$\underline{B}(\underline{r}) = \frac{\mu_0}{4\pi} \cdot \frac{3\hat{r}(\underline{m}^T\hat{r}) - \underline{m}}{|\underline{r}|^3} \quad in \quad T \tag{2.1}$$

$$\underline{H}(\underline{r}) = \frac{1}{4\pi} \cdot \frac{3\hat{r}(\underline{m}^T\hat{r}) - \underline{m}}{|\underline{r}|^3} \quad in \quad A/m \tag{2.2}$$

Mit:

$\underline{r}$ = Abstand zum Nullpunkt in $[m]$
μ_0 = Vakuum Permeabilität in $[H/m]$ oder $[Vs/m^2]$
$\underline{m}$ = Dipolmoment in $[A\ m^2]$

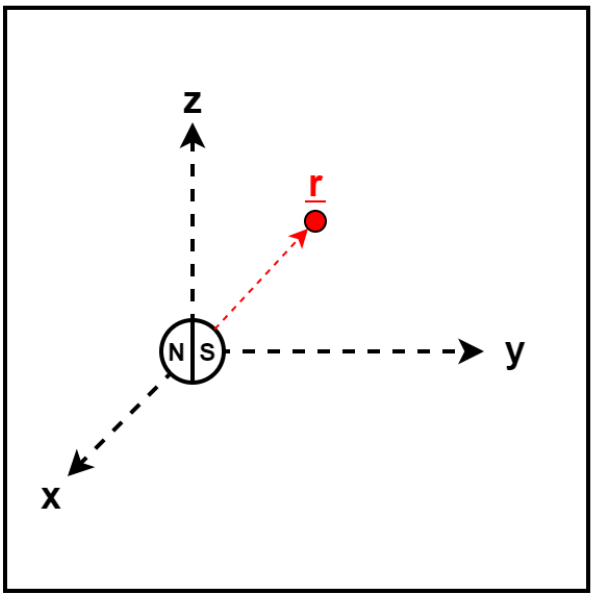


Abbildung 2.2: Gesuchte magnetische Flussdichte am Punkt $\underline{r}$, bei einem im Koordinatenursprung sitzendem Dipol mit einem Dipolmoment $\underline{m}$.

Wird ein Magnet in die Nähe des Sensor-Arrays gebracht, lässt sich mit Hilfe der Gleichung (2.2) die magnetische Flussdichte an den Punkten, an welchen sich die Sensoren befinden, ermitteln (vgl. Abbildung 2.2). Durch eine Abnahme der magnetischen Feldstärke

in der Größenordnung r^3 lassen sich genutzte Permanentmagneten als Dipol nachbilden, sobald sie sich in ausreichender Entfernung zum Array befinden. Jedoch sinkt auch die Verwertbarkeit gemessener Ergebnisse von den TMR-Sensoren mit zunehmender Distanz, da die Sensoren nur bis zu einem gewissen Abstand relativ frei von Messungenauigkeiten die magnetische Flussdichte ermitteln können. Somit muss eine optimale Balance zwischen Entfernung des Magneten und praktischer Verwertbarkeit der Messungen gefunden werden (vgl. Abbildung 2.3).

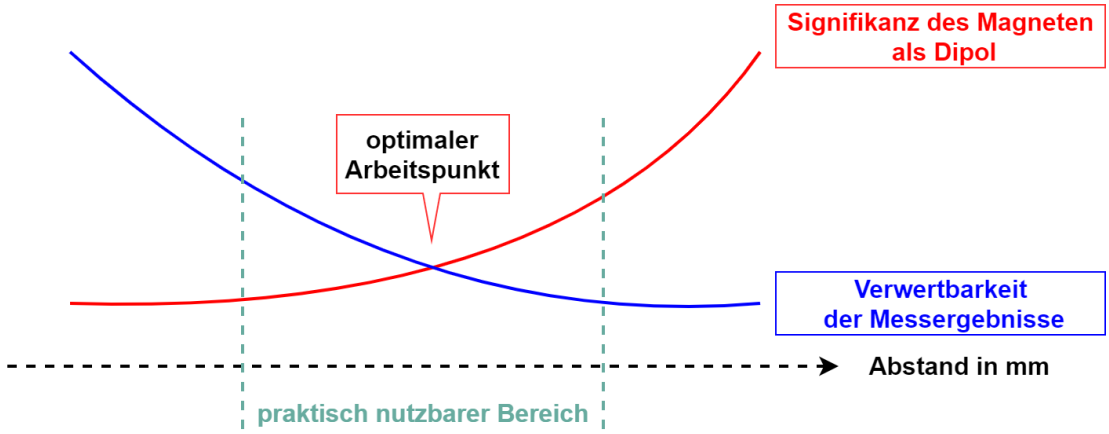


Abbildung 2.3: Darstellung der Beziehung zwischen der praktischen Verwertbarkeit der gemessenen Werte eines Sensors und der Betrachtung eines Gebermagneten als Dipol.

2.1.2 Superposition von Dipolen

Da es wegen des Abstandes nicht möglich sein wird, den Feldverlauf des Gebermagneten über die Betrachtung als Dipol ohne Fehler zu ermitteln, bietet es sich an, statt nur einem Dipol gleich mehrere in die Berechnung miteinzubeziehen. Eine Möglichkeit, den verwendeten Permanentmagneten präziser nachzumodellieren, wäre das Superponieren mehrerer Dipole. Hierbei wird die Form des verwendeten Magneten durch eine gezielte Positionierung von Dipolen nachgeahmt und die resultierenden Feldlinien können über das Superpositionsprinzip aufsummiert werden. Dadurch lässt sich eine gute Näherung des Feldverlaufes von unterschiedlichen Magnetformen erreichen (vgl. Abbildung 2.4).

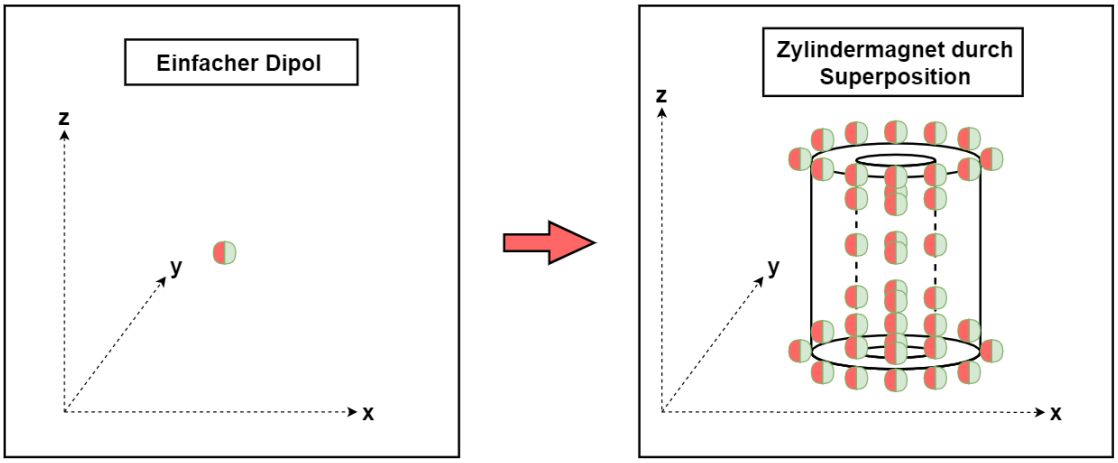


Abbildung 2.4: Modellierung eines Hohlzylindermagneten mittels superponieren von mehreren Dipolen [19, modifiziert].

2.2 Magnetische Winkelsensoren

Die Grundlage des Arrays, mit dem hier gearbeitet wird, sind die sich auf diesem befindenden magnetischen Winkelsensoren, welche die Messungen des angelegten Magnetfeldes ermöglichen. Um ein Verständnis für deren Funktionsweise zu erlangen, wird kurz der Tunnel-Magneto-Widerstandseffekt, der Aufbau der Sensoren und der Aufbau des Sensor-Arrays erläutert.

2.2.1 Tunnel-Magneto-Widerstandseffekt

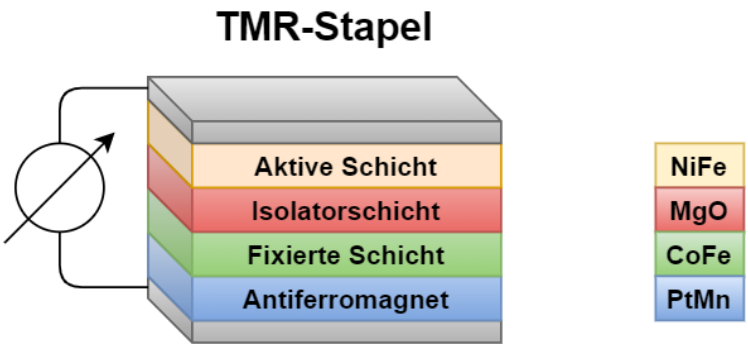


Abbildung 2.5: Schematischer Aufbau eines TMR-Stapels und Anschluss der Betriebsspannung, modifiziert nach [29].

Der Magneto-Widerstands-Effekt besagt „ [...], dass sich der elektrische Widerstand eines stromdurchflossenen Leiters unter dem Einfluss eines Magnetfeldes verändert“ [29]. Dazu zählen insbesondere der anisotrope magnetoresistive Effekt (AMR-Effekt), der Giant- beziehungsweise Riesenmagnetowiderstand (GMR-Effekt) und der magnetische Tunnelwiderstand (TMR-Effekt). Diese lassen sich noch weiter differenzieren, indem unterschieden wird zwischen magnetoresistiven Effekten in magnetischen (AMR) und in Hybrid-Bauteilen (bestehend aus magnetischen und nicht-magnetischen Materialien) (GMR, TMR) [11]. Diese Arbeit fokussiert sich auf den Tunnel-Magneto-Widerstandseffekt, welcher in magnetischen Tunnelkontakten auftritt, was in der einfachsten Form zwei Ferromagneten sind, welche durch einen dünnen Isolator getrennt werden. Diese isolierende Schicht wird auch Tunnelbarriere genannt, denn wenn diese dünn genug gewählt wird (wenige Nanometer), dann ist es Elektronen möglich durch diese hindurch zu wandern. Hierbei handelt es sich um ein rein quantenmechanisches Phänomen, welches auch als „tunneln“ bezeichnet wird [8]. Über das von außen angelegte Magnetfeld kann die Magnetisierung der beiden ferromagnetischen Schichten (in Abbildung 2.5 in gelb und grün dargestellt) gesteuert werden und damit auch die Anzahl der Elektronen, welche durch die Isolatorschicht (rot) tunneln. Sind beide Magnetisierungen gleich ausgerichtet, so ist die Wahrscheinlichkeit höher, dass diese hindurchtunneln, als wenn die beiden Schichten gegensätzlich ausgerichtet sind (vgl. Abbildung 2.6). Somit hängt der Widerstand eines TMR-Stapels, auch Dot genannt, von den Magnetisierungsrichtungen der Ferritschichten ab. Der antiferromagnetische Teil des Sensors (blau) verhindert, dass die Magnetisierung der fixierten Schicht durch das Anlegen eines äußeren Magnetfeldes verändert wird, da es sonst versuchen würde diesem zu folgen. Erreichbare relative Widerstandsänderungen $\frac{\Delta R}{R}$ liegen bei dem Tunnel-Magneto-Widerstandseffekt im Bereich von 30% bis zu 200% [26].

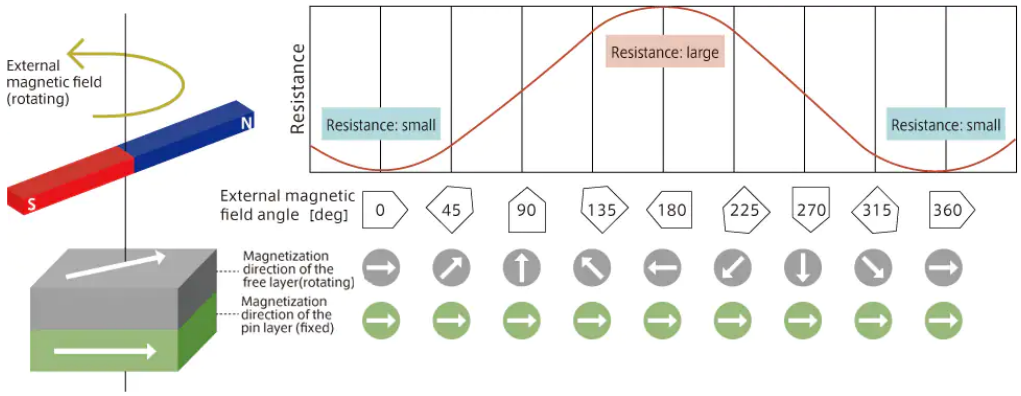


Abbildung 2.6: Widerstand eines TMR-Stapels in Abhängigkeit des Winkels der Magnetisierung zwischen aktiver und fixierter Schicht (vgl. Abbildung 2.5) [29].

2.2.2 TMR-Sensor

Theoretisch reicht ein einziger magnetoresistiver Widerstand aus, um Aussagen über ein Magnetfeld, welches es zu untersuchen gilt, zu treffen. Dieser ist jedoch nicht gegen äußere Störeinflüsse, wie zum Beispiel Temperaturschwankungen, geschützt, sodass ein einzelner TMR-Stapel dabei in ähnlichen Maßstäben seinen Widerstand verändern würde, wie bei dem Einfluss eines von außen angelegten Magnetfeldes. Um dem vorzubeugen und um eine Differenzfeldmessung zu ermöglichen, werden MR-Dots in einer Wheatstonebrücke angeordnet, welche in Abbildung 2.7 zu erkennen ist [11]. Ein an den Sensor angelegtes Magnetfeld hat eine Widerstandsänderung der MR-Widerstände in den beiden Zweigen einer Wheatstonebrücke zur Ursache, wodurch eine Spannungsdifferenz am Mittelabgriff entsteht [11]. Die hier verwendeten TMR-Sensoren des Herstellers TDK, bestehen jeweils aus zwei Vollbrückenschaltungen, welche wiederum aus vier der bereits angesprochenen TMR-Dots bestehen.

Die Verwendung dieser Doppelbrücken wird üblicherweise eingesetzt, wenn der Sensor als Gradiometer [1], mit differentiellem Sensorverhalten, oder als Winkelsensor dienen soll [26]. In beiden Fällen wird ein Magnet benötigt, auf welchen sich die Messungen beziehen.

[1] Gradiometrie bezeichnet die Messung von einer Komponente eines Gradientenfeldes [2]

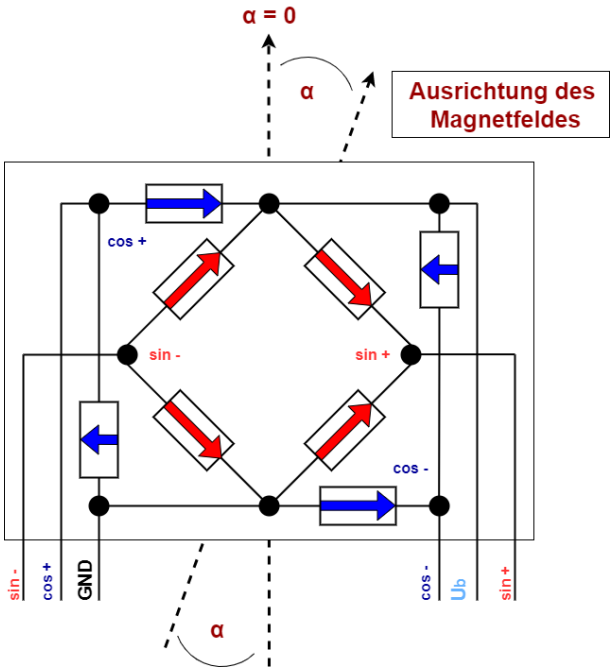


Abbildung 2.7: Aufbau eines TMR-Sensors mit zwei Vollbrücken [20, modifiziert].

Ausführung von TMR-Sensoren

Heutzutage ist es sogar möglich, bei dem Aufbau eines TMR-Dots eine wirbelförmige Magnetisierung zu nutzen. Weist die Bauform eines solchen Stapels eine kreisrunde Form auf, so richten sich auch die magnetischen Domänen vorzugsweise kreisförmig aus [26]. Diese Variante weist besondere Eigenschaften auf und wird auch als TMR-Vortex-Technologie bezeichnet [36, 21, 30]. Beispielsweise besitzen Sensoren, welche mit Vortex-Technologie ausgestattet sind, einen ausgeprägteren linearen Bereich, als die in linearer Ausführung (vgl. Abbildung 2.8).

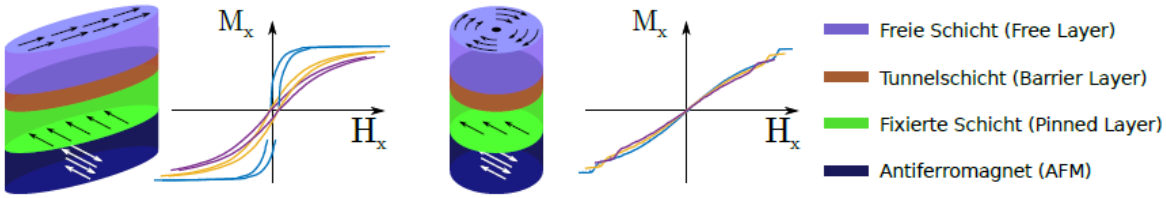


Abbildung 2.8: TMR-Sensoren links in linearer Ausführung, rechts die wirbelförmige Vortex-Variante [27].

2.2.3 Aufbau des Tunnel-Magneto-Widerstand-Arrays

Grundlage dieser Arbeit bildet das magnetoresistive Sensor-Array, welches in der Bachelorthesis von Thorbjoern Mehm ausgearbeitet wurde (Abbildung 2.9) [16]. Das Array besteht aus 8x8 TMR-Sensoren, die kompakt in Matrixstruktur auf der Platine angebracht wurden, um auch schwache Magnetfelder möglichst gut erfassen zu können. Jeder der sich auf dem Array befindlichen Sensoren besitzt zwei voneinander unabhängige Vollbrücken (Sinus- und Cosinus-Messbrücke) und gibt dementsprechend vier Signale aus: + **Sinus, - Sinus,** + **Cosinus, - Cosinus**. Damit kann jeder Sensor des Arrays als ein Pixel des später dargestellten Magnetfeldbildes verstanden werden, welches durch die beiden Brücken zwei differenzielle Spannungssignale liefert. Eines davon repräsentiert hierbei die H_x- und das andere die H_y-Komponente des Magnetfeldes [26].

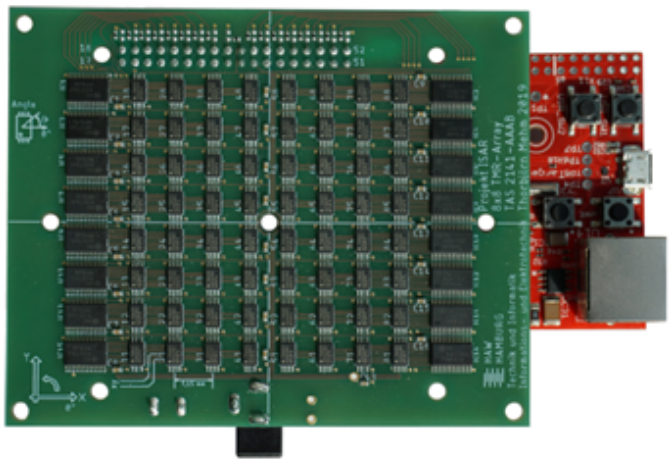


Abbildung 2.9: Magnetoresistives 8x8 Sensor-Array auf Grundlage [16].

Das heißt, der Mikrocontroller (EK-TM4C1294XL Connected LaunchPad von Texas Instruments [1]) erfasst insgesamt 256 Signale, welche dieser über die Differenz zwischen den beiden Brückensignalen berechnet. Das Auslesen erfolgt über ein Zeitmultiplexverfahren, welches anschaulich in Abbildung 2.10 dargestellt ist. Dabei wird zwischen Ein- und Ausgang eine elektrisch leitende Verbindung über die analogen Multiplexer hergestellt, wodurch die Spannung im Multiplexverfahren geschaltet werden kann.

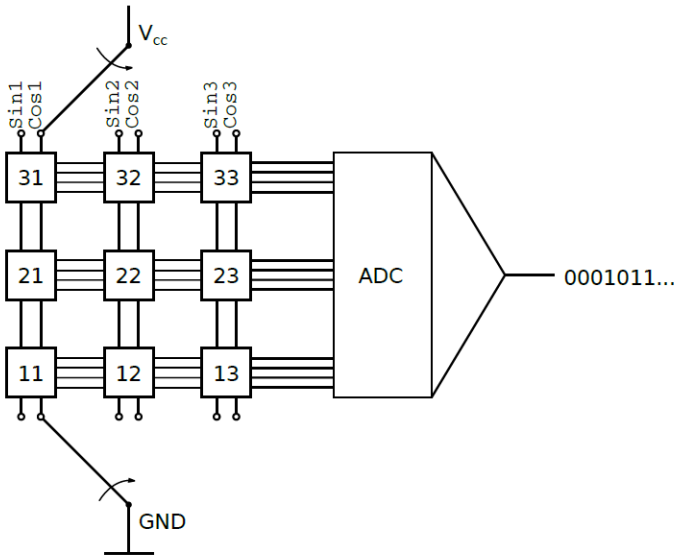
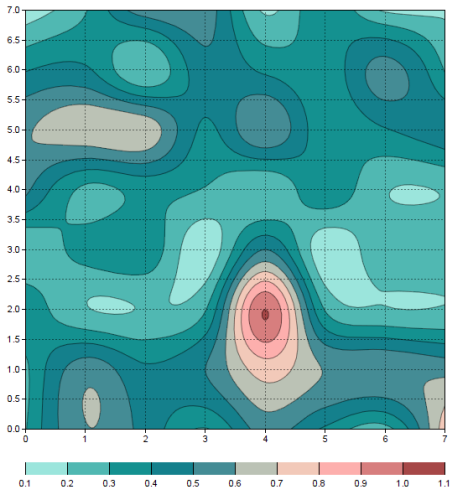


Abbildung 2.10: Zeitmultiplex der Versorgungsspannungen an einem 3x3-Beispiel. Im Bild sind die Cosinus-Brücken der ersten Spalte aktiv [16].

Es werden die Sensorausgänge zeilenweise parallelgeschaltet und jede Spalte mit einer eigenen Versorgungsleitung ausgestattet. Um nicht genutzte Versorgungspins hochohmig zu schalten, sind die Zuleitungen als Tri-State-Schaltung ausgeführt. Somit kann das Array spaltenweise ausgelesen werden, da sich die zeilenweise parallelgeschalteten Sensoren nicht mehr beeinflussen können. Nach dem Auslesen der ersten Spalte, wird diese hochohmig geschaltet und die danebenliegende dafür aktiviert. Dies wird für jede Spalte wiederholt, bis alle 256 Signale erfasst wurden. Durch die Anordnung der Bauteile auf der Platine, kann diese in eine linke und eine rechte Hälfte aufgeteilt werden. Jedem der 16 Multiplexer ist ein ADC-Kanal des Mikrocontrollers zugeteilt und um alle 256 Signale erfassen zu können, sind 16 Zyklen notwendig, welche über zwei Zählschleifen mit je acht Durchläufen realisiert worden sind. Dabei ist eine der Schleifen für das Auslesen der Sinus- und die andere für das der Cosinus-Werte zuständig, welche im Anschluss in einem

globalen Array gespeichert werden. Mit Hilfe der gesicherten Werte können die Differenzsignale berechnet und wiederum in eigenen Arrays gespeichert werden. Abbildung 2.11 zeigt beispielhaft, wie das untersuchte Magnetfeld in zwei Arrays, für die gemessenen Kosinus- und Sinusströme aufgeteilt und abgespeichert wird.



$$cos(i,j) = \begin{pmatrix} -5 & -1 & -8 & -9 & -7 & -4 & 0 & -4 \\ -5 & -6 & -1 & -3 & -9 & -10 & -3 & -4 \\ -1 & -6 & -8 & -7 & -4 & -6 & -2 & 1 \\ -3 & -8 & -11 & -5 & -6 & -5 & 0 & 1 \\ 1 & -14 & -7 & -7 & -2 & 1 & -8 & -4 \\ -12 & -5 & -4 & -6 & 0 & 1 & -7 & -5 \\ -4 & -10 & -5 & -3 & -4 & -4 & -4 & -7 \\ -2 & -7 & -7 & -4 & -3 & -3 & -5 & -2 \end{pmatrix}$$

$$sin(i,j) = \begin{pmatrix} 6 & 5 & 4 & 4 & 5 & -2 & -1 & 3 \\ 3 & 11 & 6 & 2 & 3 & 1 & 2 & -5 \\ 2 & -1 & 6 & 2 & 4 & 2 & -1 & -4 \\ 3 & -1 & 1 & 0 & 3 & 3 & -4 & -4 \\ 2 & 1 & 0 & 4 & -1 & 3 & 3 & 3 \\ -2 & 1 & 0 & 1 & 2 & 6 & 3 & 5 \\ -4 & -2 & -3 & -1 & 3 & 4 & 7 & 6 \\ -3 & 1 & 1 & 5 & 8 & 4 & 4 & 6 \end{pmatrix}$$

Abbildung 2.11: Speicherung der gemessenen differenziellen Ströme als Arrays in Matrixschreibweise.

2.3 Methoden der Signalverarbeitung

Es ist von Vorteil die im Nachhinein gemessenen Sensorausgangswerte bearbeiten zu können, vor allem um diese von unerwünschtem Rauschen zu befreien. Zum Erreichen dieses Ziels, werden die in diesem Abschnitt folgenden Signalverarbeitungsmethoden genutzt. Andernfalls könnten sich diese ungewollten Messungenauigkeiten negativ auf das gewünschte Ergebnis auswirken.

2.3.1 Die diskrete zweidimensionale Fouriertransformation

Die „Diskrete Fouriertransformation“ (DFT) stellt eine reduzierte Form der „diskreten zeitinvarianten Fouriertransformation“ (DTFT) dar und ermöglicht die Beschreibung des Frequenzgangs von Signalen, welche nicht begrenzter Dauer sind [34]. Durch die Anwendung der DFT können Frequenzmethoden, wie die digitale Filterung, numerisch auf die

Messdaten angewandt werden [6], was sich als notwendig herausstellt, wenn die gemessenen Sensorausgangswerten von unerwünschtem Rauschen getrennt werden sollen. Die diskrete Fouriertransformation wird üblicherweise dazu genutzt, um aus dem Zeit- in den Frequenzbereich zu gelangen. Dafür müssen sich die Messwerte auf den gleichen Bezugspunkt beziehen, jedoch zu unterschiedlichen Zeiten. In dem Fall des Sensor-Arrays werden jedoch 8x8 Messdaten zum selben Zeit-, aber an unterschiedlichen Bezugspunkten aufgenommen, welche als Matrix mit Koordinaten als Inhalt verstanden werden kann. Die Koordinaten stellen hierbei Daten des Ortes und nicht der Zeit dar, weshalb man hier von einer Transformation aus dem Orts- in den Bildbereich spricht, statt aus dem Zeit- in den Frequenzbereich [33].

Definition 1D-DFT

Unter der eindimensionalen DFT versteht man die spaltenweise Transformation des Frequenz- beziehungsweise des Bildbereiches einer Matrix und stellt oft die Vorstufe zur 2D-DFT dar. Die Transformation lässt sich einfach mit Hilfe der Twiddlefaktormatrix berechnen. Dafür müssen die zu transformierende und die Twiddlefaktormatrix quadratisch sowie identisch sein.

Über folgende Gleichung lässt sich ein Vektor mit N Werten im Frequenz- beziehungsweise im Bildbereich abbilden:

$$X_{1D}[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}k \cdot n} \tag{2.3}$$

Mit:

x = Eingangsvektor
X_{1D} = Ausgangsvektor 1D-DFT
n = Laufindex für die Werte des Eingangsvektors
k = Laufindex für die Frequenz

Und durch Anwendung der inversen diskreten Fouriertransformation wieder im Zeit- oder Ortsbereich:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] \cdot e^{+j\frac{2\pi}{N}k \cdot n} \tag{2.4}$$

Das Zusammenfassen von $W_N = e^{-j\frac{2\pi}{N}}$ wird auch als Dreh- oder Twiddlefaktor bezeichnet und vereinfacht die Gleichungen 2.3 und 2.4 zu:

$$X_{1D}[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{k \cdot n} \tag{2.5}$$

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] \cdot W_N^{-k \cdot n} \tag{2.6}$$

Um die Matrix W bestehend aus den Drehfaktoren zu berechnen, kann Gleichung 2.7 genutzt werden:

$$W = \frac{1}{N} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}k \cdot n} \tag{2.7}$$

Somit ergibt sich folgendes Gleichungssystem um die 1D-DFT für eine Spalte von Messwerten des Sensor-Arrays zu berechnen:

$$\begin{pmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \\ X[4] \\ X[5] \\ X[6] \\ X[7] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_N^1 & W_N^2 & W_N^3 & W_N^4 & W_N^5 & W_N^6 & W_N^7 \\ 1 & W_N^2 & W_N^4 & W_N^6 & W_N^8 & W_N^{10} & W_N^{12} & W_N^{14} \\ 1 & W_N^3 & W_N^6 & W_N^9 & W_N^{12} & W_N^{15} & W_N^{18} & W_N^{21} \\ 1 & W_N^4 & W_N^8 & W_N^{12} & W_N^{16} & W_N^{20} & W_N^{24} & W_N^{28} \\ 1 & W_N^5 & W_N^{10} & W_N^{15} & W_N^{20} & W_N^{25} & W_N^{30} & W_N^{35} \\ 1 & W_N^6 & W_N^{12} & W_N^{18} & W_N^{24} & W_N^{30} & W_N^{36} & W_N^{42} \\ 1 & W_N^7 & W_N^{14} & W_N^{21} & W_N^{28} & W_N^{35} & W_N^{42} & W_N^{49} \end{pmatrix} \cdot \begin{pmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ x[6] \\ x[7] \end{pmatrix}$$

Welches auch wie in Gleichung 2.8 in Matrixschreibweise dargestellt werden kann:

$$X_{1D} = W \cdot x \tag{2.8}$$

Definition 2D-DFT

Die eindimensionale diskrete Fouriertransformation ist die Vorstufe für die zweidimensionale Transformation, welche die Betrachtung des Bild- statt des Ortsbereiches ermöglicht. Es werden andere Indizes verwendet, da keine Abhängigkeit der Zeit vorliegt [14].

$$X_{2D}[u,v] = \frac{1}{N}\sum_{n=0}^{N-1} X_{1D}[k] \cdot e^{-j\frac{2\pi}{N}k\cdot n} \tag{2.9}$$

$$= \frac{1}{KN}\sum_{k=0}^{K-1}\left(\sum_{n=0}^{N-1} x[k,n]\cdot e^{-\frac{j2\pi kn}{N}}\right)\cdot e^{-\frac{j2\pi kn}{K}} \tag{2.10}$$

Mit:

x = Eingangsmatrix
X_{2D} = Ausgangsmatrix 2D-DFT

Auch Gleichung 2.9 kann wieder in Matrixschreibweise betrachtet werden:

$$X_{2D} = W \cdot x \cdot W \tag{2.11}$$

Die zweidimensionale diskrete Fouriertransformation kann somit als zwei aufeinanderfolgende Matrixmultiplikationen verstanden werden. Zuerst wird die Twiddlefaktormatrix mit den Eingangswerten und anschließend das resultierende Produkt erneut mit der Twiddlefaktormatrix multipliziert, wie auch in Gleichung 2.11 erkennbar ist. Da bei dem zweiten Schritt die Drehfaktormatrix, rechts statt links von der zu multiplizierenden Matrix steht, müssten Spalten und Zeilen vertauscht werden, da die Eingangswertematrix spaltenweise durchlaufen werden muss, um zum Ergebnis der 2D-DFT zu gelangen. Durch das Transponieren von X_{1D}, der Drehfaktormatrix und der Ausgangsmatrix sowie Anwendung des Kommutativgesetzes, lässt sich eine ansonsten nötige Fallunterscheidung vermeiden. Da die Twiddlefaktormatrix identisch mit ihrer Transponierten ist, erübrigt sich hier ein Vertauschen der Indizes. In Gleichung 2.12 und Abbildung 2.12 wird das Vorgehen noch einmal deutlich:

$$\begin{aligned} X_{2D} &= W \cdot x \cdot W \\ &= ((x \ \cdot W)^T \cdot W)^T \\ &= (X_{1D}^T \cdot W)^T \end{aligned} \tag{2.12}$$

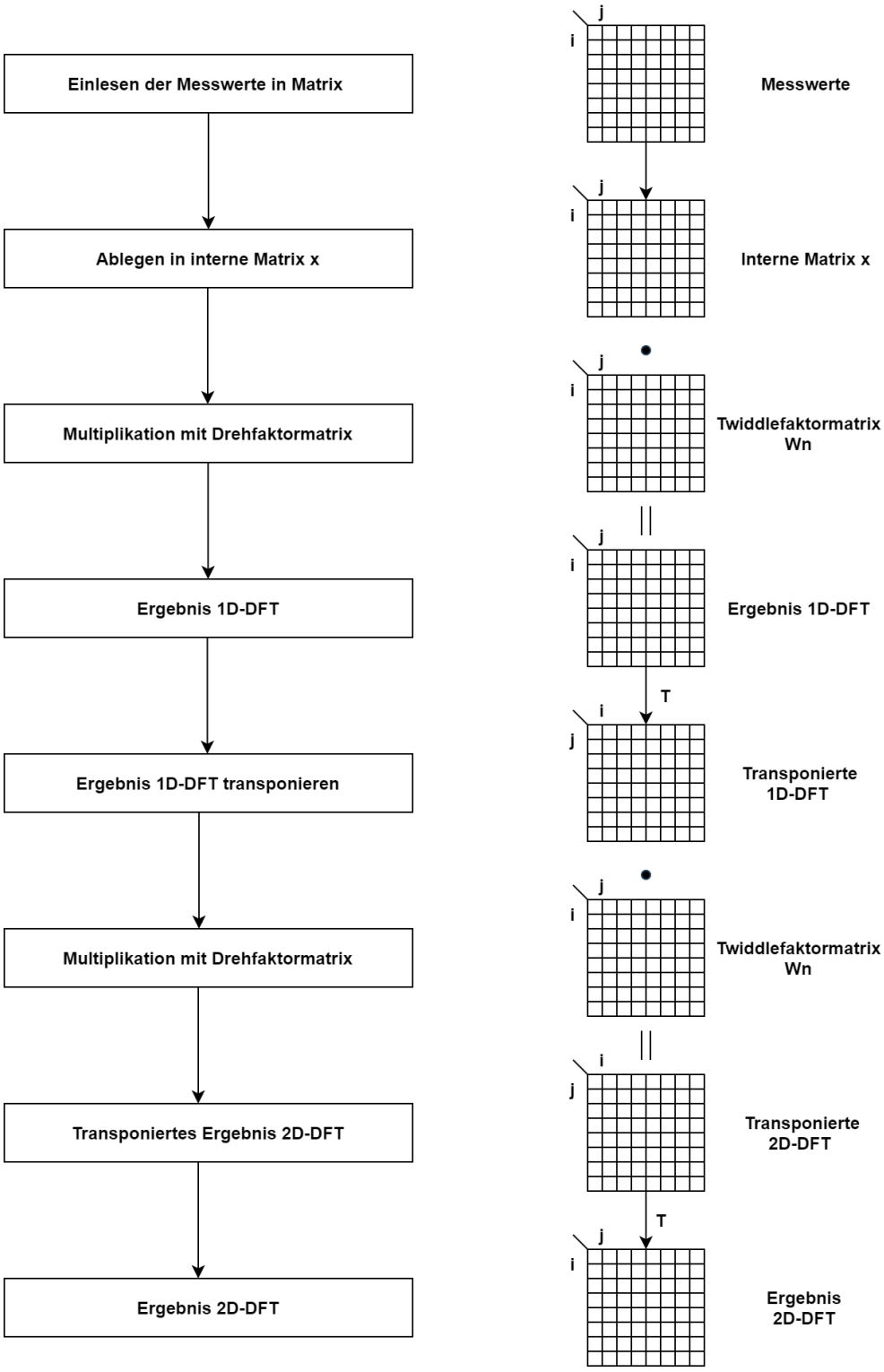


Abbildung 2.12: Ablaufdiagramm für die Berechnung der 2D-DFT [14, modifiziert].

2.3.2 Interpolation

Durch Interpolation lassen sich theoretische Messwerte zwischen zwei oder mehreren Sensoren des Arrays berechnen, wodurch sich eine höhere Auflösung des dargestellten Magnetfeldbildes erreichen lässt. Die prinzipielle Funktionsweise lässt sich mit Hilfe von Abbildung 2.13 einfach nachvollziehen.

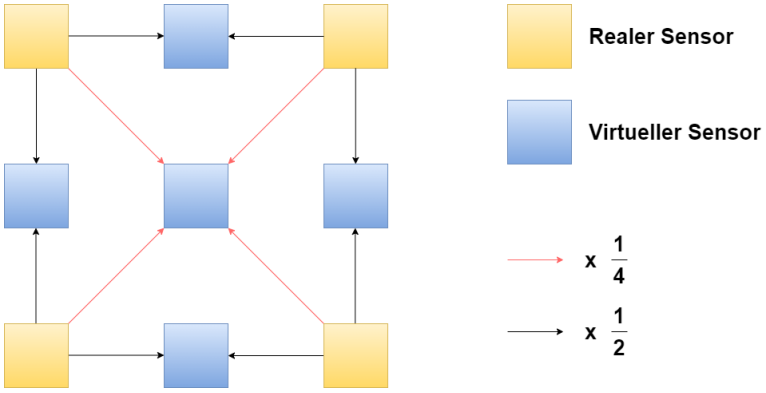


Abbildung 2.13: Funktionsprinzip Interpolation von Simon Rindelaub [25].

In dieser Arbeit soll durch Interpolation aus dem realen 8x8 ein theoretisches 15x15 Sensor-Array gebildet werden, welches nochmal durch Abbildung 2.14 verdeutlicht werden soll.

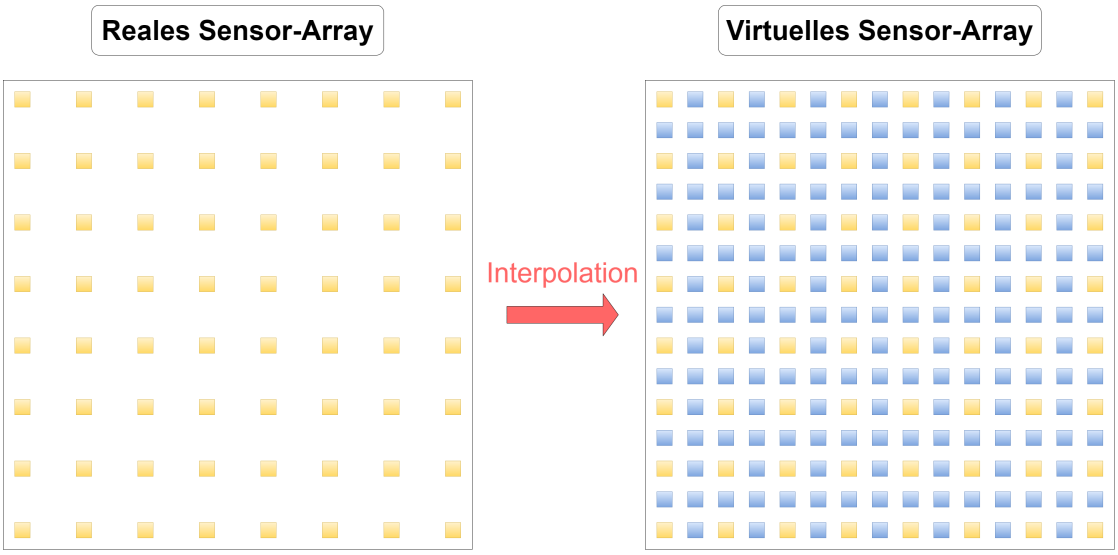


Abbildung 2.14: Verbesserte Auflösung des Sensor-Arrays durch Interpolation der Messwerte.

2.3.3 Vorverarbeitung von Rohdaten

Bevor die gemessenen Daten des Sensor-Arrays weiterverarbeitet werden, sollten Messfehler, welche während des Messvorganges auftreten können, entfernt werden. Diese werden beispielsweise durch das Magnetfeld der Erde, Rauschen, von sich in der Nähe befindender Elektronik oder atomsphärischem Rauschen, Restmagnetisierung der Sensoren, usw. verursacht. Durch Sicherstellung, dass möglichst wenig Restmagnetisierung in der freien ferromagnetischen Schicht der Sensoren vorhanden ist, aber auch durch Verfahren wie **„Offsetkompensation“** oder **„digitale Filterung im Orts- und Frequenzbereich“** können diese Messungenauigkeiten weitgehend korrigiert werden. Ohne diese Vorverarbeitungsschritte könnten sich die Messfehler negativ auf den Suchalgorithmus und somit verfälschend auf die errechneten Positionsdaten auswirken [18].

Offsetkompensation

Offsetkompensation wird dort eingesetzt, wo sich durch äußerliche Einflüsse Messungenauigkeiten eingeschlichen haben. In dem Fall des Sensor-Arrays können durch Störfelder, wie dem Erdmagnetfeld oder künstlichen Elektromagnetfeldern sowie durch Restmagnetisierung der freien Schicht eines TMR-Dots, die Messergebnisse negativ beeinflusst werden, da die TMR-Sensoren auch auf diese reagieren. Diese besitzen dadurch eine Vormagnetisierung, welche zu Ungenauigkeiten bei den eigentlichen Messungen führen. Man spricht hier auch von einem Offset. Um diesen Messungenauigkeiten vorzubeugen, wird vor den Messungen der Ist-Zustand (vgl. Abbildung 4.15 (links)) eines jeden Sensors auf dem Array wiederholt aufgenommen und aufsummiert. Anschließend wird durch die Anzahl der Summanden geteilt, um den Mittelwert für jeden Pixel zu bilden. Bei darauffolgenden Messungen kann den Werten des Arrays der entsprechende Offset abgezogen werden, um Störeinflüsse zu minimieren (vgl. Abbildung 4.15 (rechts)).

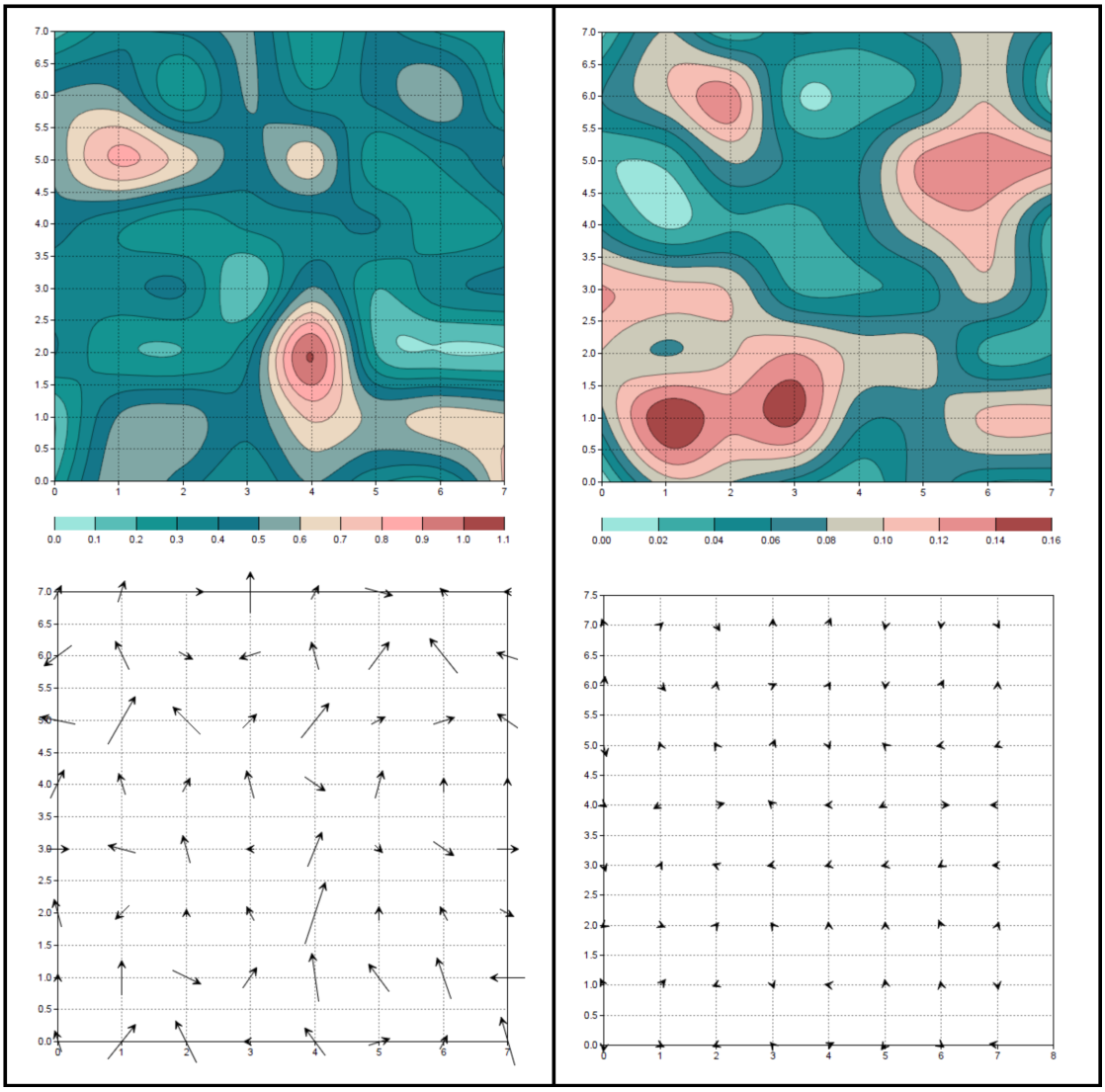


Abbildung 2.15: Vergleich der gemessenen Magnetfeldstärken vor (links) und nach der Offsetkompensation (rechts).

Digitale Filter im Ortsfrequenzraum mittels Filtermasken

Um unerwünschte Messungenauigkeiten aus den Rohdaten zu entfernen, werden einfache lineare digitale Filter eingesetzt. Sie sind deshalb linear, „weil sie die [...] Werte innerhalb der Filterregion in linearer Form, d. h. durch eine gewichtete Summation verknüpfen“ [7]. Als ein solcher linearer Filter kann eine Filtermatrix eingesetzt werden, welche allein aus ihren Filterkoeffizienten besteht. Ein 8x8 Glättungsfilter, welcher auch im Programmcode implementiert ist, sieht dabei wie folgt aus:

$$H(i,j) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.13}$$

Damit dieser Filter sinnvoll eingesetzt werden kann, ist es nötig die Ergebnisse der 2D-DFT im Ursprung zu zentrieren. Denn die verwertbaren Messdaten liegen im niederfrequenten Bereich, welcher sich im ursprünglichen Ergebnis der DFT in den Eckpunkten befindet [7]. Eine Darstellung beispielhafter Rohdaten, deren diskrete Fouriertransformation und anschließende Verschiebung im Frequenzbereich in den Nullpunkt ist in Abbildung 2.16 zu sehen.

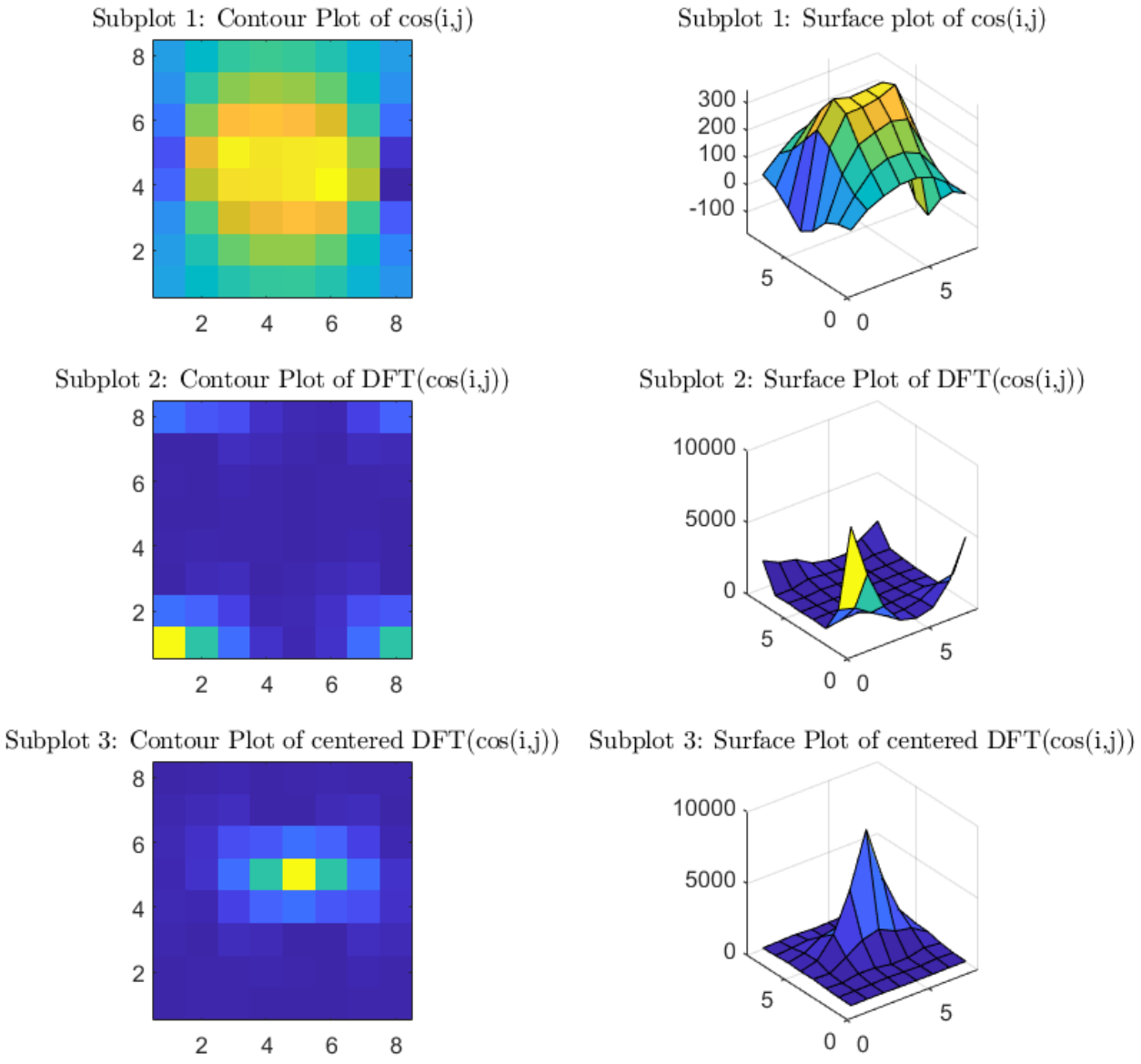


Abbildung 2.16: Darstellung der Messdaten (oben), anschließender DFT vor (mittig) und nach (links) der Zentrierung in den Ursprung.

Durch eine elementweise Multiplikation der Matrix, welche die Messdaten beinhaltet, und der Matrix, welche die Glättungsfilterkoeffizienten umfasst (siehe Gleichung 2.13),

werden die hochfrequenten Anteile und somit das unerwünschte Rauschen herausgefiltert. Das Vorgehen lässt sich noch einmal anschaulich in Abbildung 2.17 nachvollziehen:

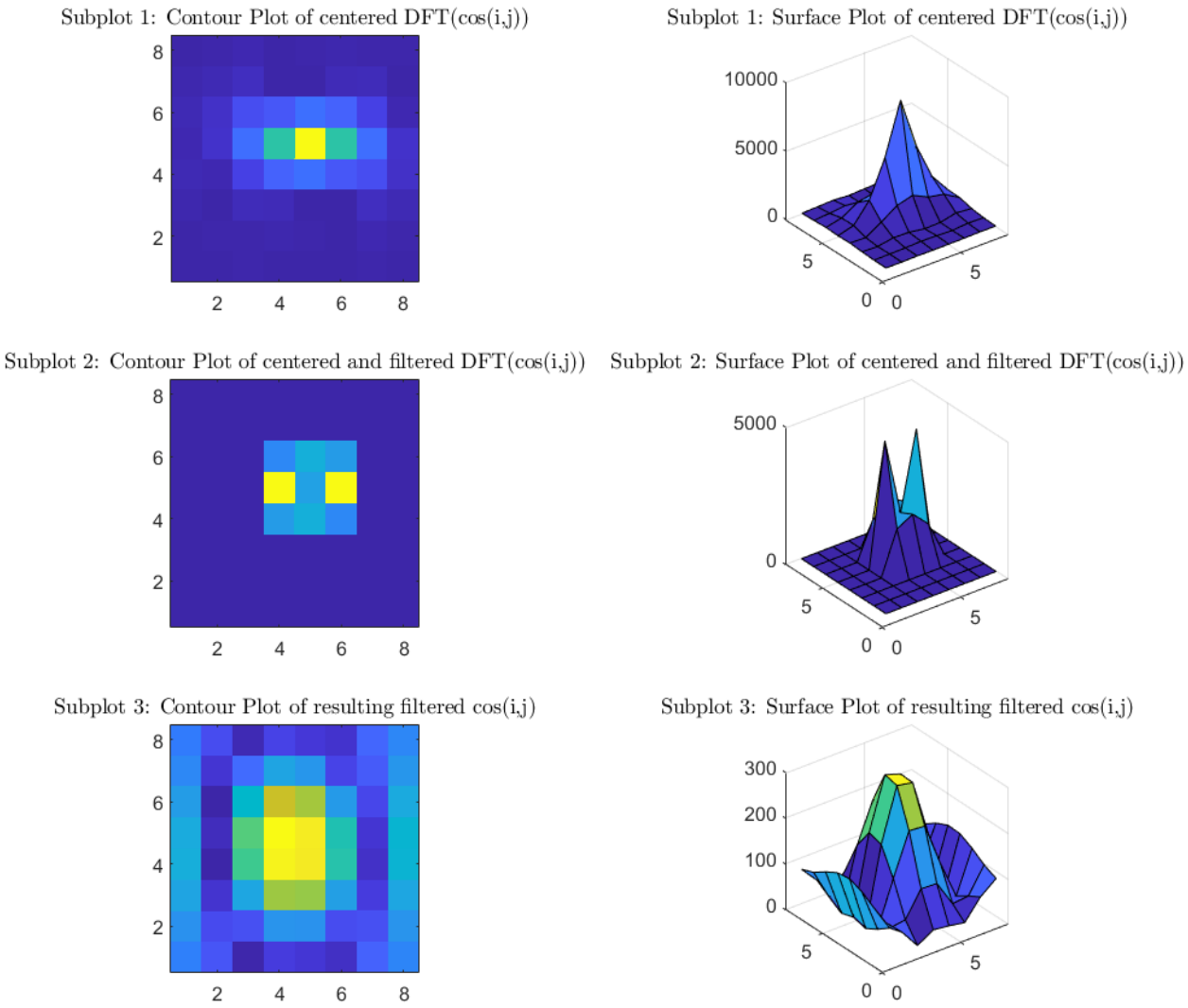


Abbildung 2.17: Darstellung der Messdaten (oben) nach Anwendung des Filters aus Gleichung 2.13 (mittig) und anschließender Rücktransformation (2D-IDFT) (unten).

Vergleicht man Abbildung 2.16 mit Abbildung 2.17 sind die Verbesserungen der gemessenen Sensorausgangsdaten deutlich sichtbar. In Abbildung 2.16 (oben), in welchem die Rohdaten wiedergegeben werden, ist nur ein in der Mitte konzentriertes Feld zu erkennen. In Abbildung 2.17 (unten) hingegen, in welchem die Daten nach Anwendung des Filters präsentiert werden, sind schon weitaus detailliertere Verläufe auszumachen.

2.3.4 Rotationsmatrizen

Der theoretische Hintergrund für Rotationsmatrizen ist hier erläutert, da diese später im entworfenen Algorithmus genutzt werden, um ein hypothetisches Sensor-Array über einem Dipol zu rotieren. Das ist notwendig, damit für jede Position und Lage des Arrays die magnetische Flussdichte ermittelt werden kann. Durch einen Vergleich der realen Sensorausgangswerte mit denen, die im Algorithmus ermitteln wurden, lassen sich anschließend Aussagen über den tatsächlichen Ort des verwendeten Permanentmagneten treffen. Rotationsmatrizen, auch Drehmatrizen genannt, sind reelle, orthogonale Matrizen, welche eine Determinante von +1 besitzen [3]. Wird ein Vektor mit einer solchen Matrix multipliziert, kann dies als eine Drehung des Vektors um seinen Ursprung mit dem Winkel α verstanden werden. In dieser Arbeit werden Rotationen von Vektoren im Raum $\mathbb{R}^3$ vorgenommen, welche sich mit Hilfe von folgenden Matrizen realisieren lassen:

Drehung um die x-Achse

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{2.14}$$

Drehung um die y-Achse

$$R_y(\alpha) = \begin{pmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ sin(\alpha) & 0 & cos(\alpha) \end{pmatrix} \tag{2.15}$$

Drehung um die z-Achse

$$R_y(\alpha) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.16}$$

Ein Beispiel für die Rotation eines Vektors im $\mathbb{R}^3$ ist der Abbildung 2.18 zu entnehmen.

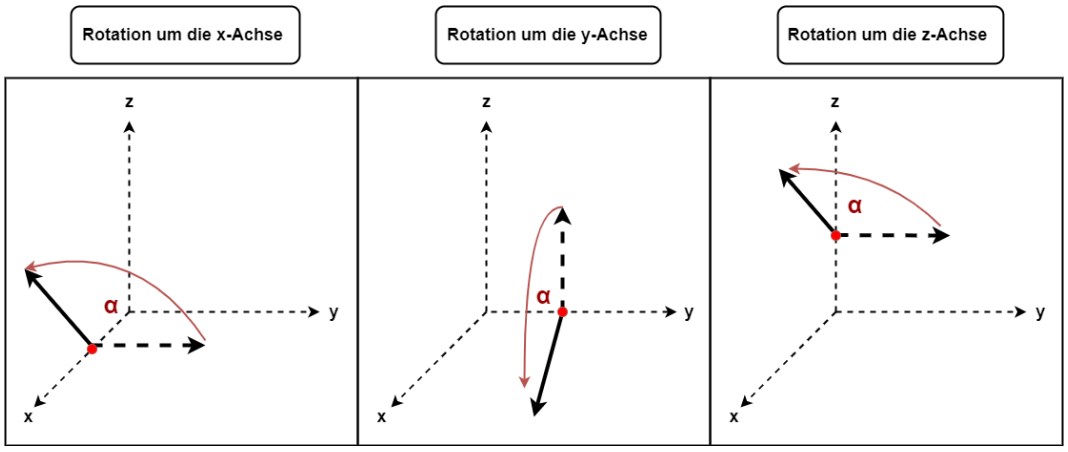


Abbildung 2.18: Einfaches Beispiel für die Rotation eines Vektors im $\mathbb{R}^3$.

3 Algorithmusentwicklung

3.1 Inverses Problem

Mit Hilfe von physikalischen Theorien ist es möglich, Vorhersagen über den Ausgang von Experimenten zu treffen, vorausgesetzt, es liegt eine komplette Beschreibung des physikalischen Systems vor, welches es zu untersuchen gilt. Dieses Vorgehen wird auch **Vorwärtsproblem** oder **Vorwärtsmodellierung** genannt und beschreibt den direkten Zusammenhang von Ursache und Wirkung. Wird umgekehrt versucht, über die Wirkung Informationen über deren Ursache zu erhalten, spricht man von einem **inversen Problem** (vgl. Abbildung 3.1). Ein wichtiger Aspekt bei der Lösungsfindung eines inversen Problems ist der Informationsstand, der zu den Kenngrößen des physikalischen Systems vorliegt, welche sich im Allgemeinen auch als Wahrscheinlichkeitsdichten ausdrücken lassen [32]. Dazu gehören die beobachtbaren Parameter, die a priori Informationen über die Modellparameter (siehe Abschnitt 3.1.4) und Wissen über deren physikalischen Zusammenhänge. Die Messungen, welche notwendig für die Informationsgewinnung sind, können an sich keine Wahrscheinlichkeit für ein auftretendes Ereignis beschreiben, sondern dafür, wie wahrscheinlich es ist, dass zwei Ereignisse zusammenhängen. Beispielsweise lässt sich aufgrund endlicher Genauigkeiten der TMR-Sensoren, die magnetische Flussdichte an einem Punkt $\underline{r}$ nicht exakt bestimmen, stattdessen lässt sich eine Wahrscheinlichkeitsdichte für diese vorschlagen. Diese zeigt auf, wie wahrscheinlich das aktuell angelegte Magnetfeld mit dem gemessenen Wert am Punkt $\underline{r}$ übereinstimmt. Das Vorwärtsproblem besitzt nur eine, das inverse Problem kann jedoch viele verschiedene Lösungen besitzen. Im Fall des Sensor-Arrays oder eines MR-Sensors würde das in die Nähe bringen eines Magneten zum Sensor zu messbaren Sensorausgangswerten führen. Mit Kenntnissen über die Beschaffenheit des Magneten, seiner Position über dem Array und über Eigenschaften des verwendeten MR-Sensors würden sich die ungefähren Messwerte im Vorfeld bestimmen lassen. Dafür wird beispielsweise in diesem Fall mit Hilfe der Dipolgleichung (siehe Gleichung 2.2 Abschnitt 2.1.1) die Magnetfeldstärke an einem

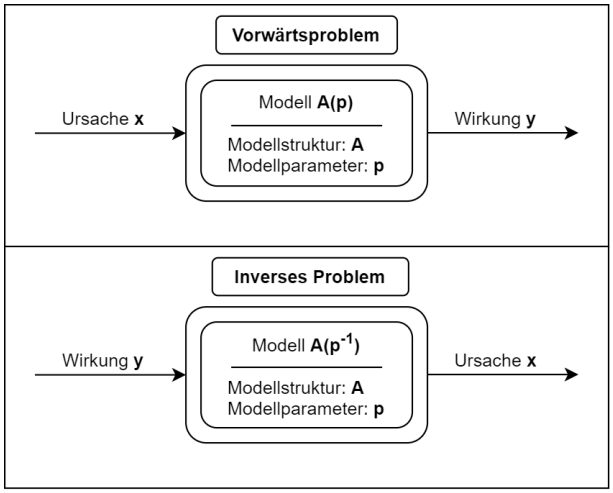


Abbildung 3.1: Schematische Darstellung der Ursache-Wirkungs-Beziehung [9, modifiziert]; hier $\mathbf{x}$: Lage Sensor-Array, $\mathbf{y}$: differenzieller Strom, $\mathbf{A}(\mathbf{p})$: Beziehungsbeschreibung zwischen $\mathbf{x}$ und $\mathbf{y} \rightarrow$ Magnetisierung des Magneten.

Punkt des Arrays, an welchem sich ein Sensor befindet, in x- und y-Richtung berechnet. Über die gewonnene Erkenntnis der Magnetfeldstärke, der Magnetisierungsrichtung und der angelegten Betriebsspannung könnte der Widerstand des MR-Sensors und somit die sich theoretisch ergebenden Ausgangswerte von diesem ermittelt werden. Versucht man nun umgekehrt über die Ausgangswerte des Sensors Aussagen über die Beschaffenheit und Position des Gebermagneten zu treffen, ergibt sich eine Vielzahl an Lösungen für Position und Eigenschaften des eingesetzten Magneten, für die das gemessene Magnetfeld übereinstimmen würden. Es ist also unabdingbar, so viel relevante Informationen über das zu simulierende Modell zusammenzutragen wie möglich, um eine genaue Einsicht über die zu untersuchende Ursache zu erhalten. Genauso wichtig ist es, die Ungenauigkeiten, beziehungsweise Schwächen des angefertigten Modells, zu erkennen, um zu verstehen, welche Auswirkungen diese auf das Ergebnis haben können. Bei inversen Problemen gibt es wie bereits angesprochen keine eindeutige Lösung, sondern eine Vielzahl von möglichen Ausgängen, welche alle mit denselben vorher zusammengetragenen Daten erarbeitet wurden und am Ende im Kollektiv einen Überblick über den Ursprung der gemessenen Ausgangswerte geben.

Laut Albert Tarantola [32] lässt sich die Studie eines physikalischen Systems in drei Schritte unterteilen:

1. **Parametrierung des Systems**: Beschreibung des Modells durch eine minimale Anzahl von Parametern, mit welchen das System vollständig repräsentiert werden kann
2. **Vorwärtsmodellierung**: Ermitteln von physikalischen Gesetzen, über welche sich Aussagen über beobachtbare Ausgangswerte von Messungen treffen lassen
3. **Rückwärtsmodellierung**: Die beobachteten Ausgangswerte der Experimente werden herangezogen, um die tatsächlichen Werte der Modellparameter abzuleiten

3.1.1 Modell- und Datenraum

Modellraum

Der erste Schritt ist, wie bereits zuvor erläutert, die Parametrierung des zu beschreibenden Systems und beinhaltet das Aufstellen von geeigneten messbaren Kenngrößen, welche Einfluss auf das Ergebnis des physikalischen Systems haben. Je nach untersuchtem Problem, kann die Anzahl von diesen entweder unendlich sein oder einen diskreten Wert annehmen. Und diese Parameter können entweder kontinuierliche oder auch wieder diskrete Werte besitzen. Zum Beispiel kann die Masse der Sonne eine Zahl zwischen Null und Unendlich annehmen [32], die Geschwindigkeit eines Autos hingegen kann nur durch eine diskrete Zahl beschrieben werden. Die gewählten Modellparameter werden auch durch den Buchstaben **m** gekennzeichnet und somit ergeben sich in dieser Arbeit folgende Parameter, um das hier angefertigte Modell zu beschreiben:

m1: Verschiebung in x-Richtung

m2: Verschiebung in y-Richtung

m3: Verschiebung in z-Richtung

m4: Rotation um die x-Achse

m5: Rotation um die y-Achse

m6: Rotation um die z-Achse

(m7: Magnetisierungsstärke)

Dabei gilt die Magnetisierungsstärke nur als Modellparameter, wenn diese mit in den Lernalgorithmus aufgenommen wird. Andernfalls kann diese vorher auch als diskreter Wert festgelegt werden.

Mit Hilfe dieser Parameter lassen sich eine Vielzahl von unterschiedlichen Modellen aufstellen, welche alle zu anderen Ergebnissen in Bezug auf die theoretisch berechneten Sensorausgangswerte führen würden. Der homogenen Wahrscheinlichkeitsdichte des Modellraums, welche über theoretische Vorkenntnisse aufgestellt werden kann, wird das Symbol $\mu_M(\mathbf{m})$ zugeordnet. Und da die a priori Informationen des Modellraums, welche über $\rho_M(\mathbf{m})$ gekennzeichnet werden, in dieser Arbeit unabhängig von Beobachtungen sind, gilt:

$$\rho_M(\mathbf{m}) = \mu_M(\mathbf{m}) \tag{3.1}$$

Datenraum

„Zur Lösung inverser Probleme ist es zweckmäßig Vorkenntnisse der gesuchten Größen zu berücksichtigen, um [...] sinnvolle Ergebnisse zu erhalten“ [9].

Um Informationen über den Einfluss der Modellraumparameter zu gewinnen, ist es nötig, Messungen durchzuführen und die Ausgangswerte bei Anpassung einer der Kenngrößen festzuhalten.

Durch die Visualisierung der Magnetfeldstärke mit Hilfe der implementierten Software lässt sich relativ einfach beobachten, welchen Einfluss eine Veränderung eines dieser Parameter (m1, m2, .., m7) auf die Sensorausgangswerte besitzt. Die Sensorausgangswerte bestehen zum einen aus dem differentiellen Kosinus- und zum anderen aus dem differentiellen Sinussignal eines jeden Sensors und bilden die Grundlage des Datenraumes für dieses inverse Problem. Gekennzeichnet werden Größen vom Datenraum auch durch den Buchstaben **d**:

d1: Differentielles Sinusausgangssignal

d2: Differentielles Kosinusausgangssignal

Die über theoretische Grundlagen gewonnene Wahrscheinlichkeitsdichte des Datenraumes wird mit dem Symbol $\mu_D(\mathbf{d})$ versehen. Sie beschreibt hier das Wissen zu dem Verhalten der Sensoren und beinhaltet zum Beispiel die Berechnung der Ausgangswerte, wenn

ein Magnetfeld bestimmter Größe angelegt wird. Die Wahrscheinlichkeitsdichte, welche durch Messungen ermittelt wurde, wird über $\rho_D(\mathbf{d})$ gekennzeichnet. Diese beschreibt die tatsächlich gemessenen Sensorausgangswerte, wenn ein Magnetfeld bestimmter Größe in die Nähe eines Sensors gebracht wird.

3.1.2 Vorwärtsproblem

Über Experimente lassen sich physikalische Theorien aufstellen, über welche sich wiederum der Ausgang von Experimenten voraussagen lässt [32]. Durch das Vergleichen der echten mit den im Vorfeld händisch ermittelten Ausgangswerten, lässt sich die vorher aufgestellte Theorie verbessern beziehungsweise zuvor erst einmal beweisen. Wenn in der physikalischen Theorie die Parameter inkludiert werden, welche das System, das es zu untersuchen gilt, beschreiben, dann handelt der Kern der inversen Problemtheorie von den quantitativen Regelungen, die genutzt werden, um die geschätzten mit den beobachteten Werten zu vergleichen [32]. Allerdings können die vorhergesagten Ausgangswerte aus zwei einfachen Gründen nicht den echten gleichen: **Messungenauigkeiten** und **Ungenauigkeiten in der Modellierung**. Somit stellt es sich als vorteilhaft heraus, jeden Zustand von Informationen durch eine Wahrscheinlichkeitsdichte zu repräsentieren. Könnten Ungenauigkeiten vollständig ausgeschlossen werden, würde sich ein Zusammenhang bei dem Vorwärtsproblem von

$$\mathbf{m} \mapsto \mathbf{d} = \mathbf{g}(\mathbf{m}) \tag{3.2}$$

ergeben. Für ein aufgestelltes Modell $\mathbf{m}$ könnten also die fehlerfreien Datenwerte $\mathbf{d}$ abgeschätzt werden, welche zu diesem korrespondieren würden. Dabei wird $\mathbf{g}(\cdot)$ auch als Vorwärtsoperator bezeichnet und stellt das mathematische Modell für das physikalische System dar. Können Ungenauigkeiten nicht ausgeschlossen werden, lässt sich nur ein Bereich angeben, in welchem sich die Lösung befindet [32] (vgl. Abbildung 3.2):

$$\theta(\mathbf{d}, \mathbf{m}) \tag{3.3}$$

In dem Fall des Sensor-Arrays entspräche das mathematische Modell zum Beispiel der Dipolgleichung 2.2 im Grundlagenteil. Die gemeinsame Wahrscheinlichkeitsdichte, welche die Korrelationen beschreibt, die der physikalischen Theorie entsprechen mitsamt ihrer innewohnenden Unsicherheiten, wird mit $\Theta(\mathbf{d}, \mathbf{m})$ notiert und ist wie folgt definiert:

$$\Theta(\mathbf{d}, \mathbf{m}) = \theta(\mathbf{d}|\mathbf{m})\mu_M(\mathbf{m}) \tag{3.4}$$

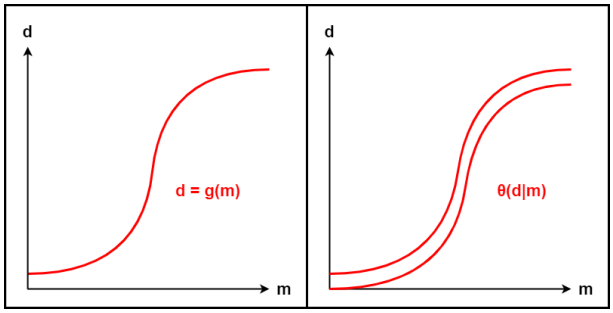


Abbildung 3.2: Vorwärtsmodellierung ohne (links) und mit (rechts) Ungenauigkeiten im Modell.

3.1.3 Homogene Wahrscheinlichkeitsverteilung

Angenommen ein Parameterraum $\mathbf{X}$ beinhaltet eine Vorstellung von Volumen und es ist möglich, unabhängig von der Wahrscheinlichkeit, welche über $\mathbf{X}$ definiert wurde, jedem Bereich A, der sich in $\mathbf{X}$ befindet, ein Volumen $V(A)$

$$dV(\mathbf{x}) = v(\mathbf{x})d\mathbf{x} \tag{3.5}$$

zuzuordnen. Dann kann das Volumen einer Region über

$$V(A) = \int_A d\mathbf{x}v(\mathbf{x}) \tag{3.6}$$

gekennzeichnet werden, wobei $v(\mathbf{x})$ als Volumendichte bezeichnet wird [32]. Ist das vollständige Volumen des Parameterraums $\mathbf{X}$ ein endlicher Wert, dann ist die Wahrscheinlichkeitsdichte

$$\mu(\mathbf{x}) = \frac{v(\mathbf{x})}{V} \tag{3.7}$$

normiert und assoziiert mit jedem Bereich A, der sich in $\mathbf{X}$ befindet eine Wahrscheinlichkeit, welche proportional zum Volumen $V(A)$ ist [32].

$$M(A) = \int_A d\mathbf{x}\mu(\mathbf{x}) \tag{3.8}$$

Die Wahrscheinlichkeit M und die zugehörige Wahrscheinlichkeitsdichte $\mu(\mathbf{x})$ werden als **homogen** bezeichnet.

3.1.4 A priori Informationen

Als **A Priori Informationen** werden Informationen bezeichnet, die unabhängig von aufgestellten Messungen gewonnen werden [32]. Sie werden durch

$$\rho_M(\mathbf{m}) \quad wobei \quad \rho_M(\mathbf{m}) = \mu_M(\mathbf{m}) \tag{3.9}$$

gekennzeichnet. Sind beispielsweise Kenntnisse über die Wahrscheinlichkeitsdichten von H_x und H_y eines Magneten über dem Sensor-Array bekannt, könnten Modelle mit Positions- und Lagedaten (m1, m2, ..., m7) angefertigt werden, die den Werten der magnetischen Flussdichte entsprechen. Eine Kombination der Wahrscheinlichkeitsdichten des Modell- sowie des Datenraumes wird symbolisiert über:

$$\rho(\mathbf{d}, \mathbf{m}) = \rho_D(\mathbf{d})\rho_M(\mathbf{m}) \tag{3.10}$$

3.1.5 Definieren einer Lösung für inverse Probleme

Es ist bei inversen Problem von Interesse, Informationen vom Daten- in den Modellraum zu übersetzen [32]. Durch Kombinieren beider Wahrscheinlichkeitsdichten (vgl. Abbildung 3.3) aus Gleichungen 3.10 $\rho(\mathbf{d}, \mathbf{m})$, welche die a priori Informationen und beobachteten Kenntnisse repräsentiert, und 3.4 $\Theta(\mathbf{d}, \mathbf{m})$, welche Informationen beinhaltet, die zum Beispiel aus physikalischen Gesetzgebungen erlangt wurden, erhält man die **a posteriori state Informationen**, welche über Gleichung 3.11 definiert ist [32].

$$\sigma(\mathbf{d}, \mathbf{m}) = k \cdot \frac{\rho(\mathbf{d}, \mathbf{m})\Theta(\mathbf{d}, \mathbf{m})}{\mu(\mathbf{d}, \mathbf{m})} \tag{3.11}$$

Mit :

$k =$ Normierungskonstante

Die a posteriori Informationen des Modellraums sind über die Wahrscheinlichkeitsdichte in Gleichung 3.12 und die des Datenraumes in Gleichung 3.13 gegeben.

$$\sigma_M(\mathbf{m}) = \int_D d\mathbf{d}\, \sigma(\mathbf{d}, \mathbf{m}) \tag{3.12}$$

$$\sigma_D(\mathbf{d}) = \int_M d\mathbf{m}\, \sigma(\mathbf{d}, \mathbf{m}) \tag{3.13}$$

Über $\sigma_M(\mathbf{m})$ (vgl. Gleichung 3.12) lassen sich einige Kenngrößen für die Modellparameter ableiten: Mittelwerte, Mediane, Werte für die maximale Wahrscheinlichkeit, etc. [32]. Es lässt sich aber vor allem die Wahrscheinlichkeit eines Modells **m** berechnen, welches (gewisse) Charakteristika des zu simulierenden Systems erfüllt, indem dort über die Wahrscheinlichkeitsdichte des Modellraums integriert wird, wo die gestellten Anforderungen an das Modell erfüllt werden. Somit lassen sich dadurch verschiedene Proben (Modelle) aufstellen (m1, m2,..), welche das Verständnis für die aufgestellten Modellparameter vertiefen.

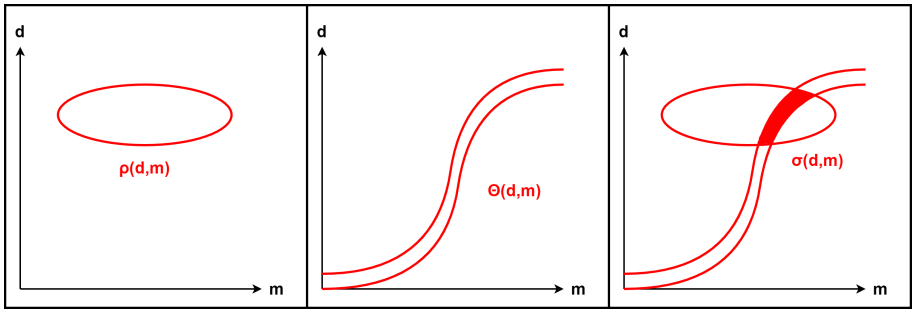


Abbildung 3.3: Kombination der Wahrscheinlichkeitsdichten (rechts) für die observierten Daten (links) und den Daten die aus physikalischen Gesetzen gewonnen werden (mitte).

3.1.6 Lösung für inverse Probleme nutzen

Wie im vorherigen Abschnitt bereits erläutert wurde, handelt es sich bei der Lösung inverser Probleme um die Wahrscheinlichkeitsverteilung über den Modellraum $\sigma_M(\mathbf{m})$. Allgemeiner lässt sich aber die Frage stellen, wie wahrscheinlich es ist, dass ein bestimmtes Modells **m** zu einer gegebenen Region **A** des Modellraums gehört [32]. Um solche

Wahrscheinlichkeiten zu berechnen, lassen sich Monte-Carlo-Methoden nutzen, welche eine hohe Anzahl an möglichen Lösungen berechnen.

Zufällige Untersuchung des Modellraums

Bei einer niedrigen Anzahl von Modellparametern (kleiner 10) lässt sich ein Gitter über dem Modellraum definieren und $\sigma_M(\mathbf{m})$ kann für jede Stelle von diesem berechnet werden [32]. Die Informationen, die dadurch gewonnen werden, können direkt eingesetzt werden, um Schlüsse über die Modellparameter zu ziehen. Andernfalls bietet es sich an, den Modellraum mit Hilfe von Monte-Carlo-Methoden zu untersuchen.

3.2 Monte-Carlo-Methoden

Bei Monte-Carlo-Methoden handelt es sich um Algorithmen, welche Zufallszahlen benutzen, um zu einem gesuchten Ergebnis zu gelangen [17]. Bei einer Simulation wird eine reale Situation durch ein Modell, beispielsweise auf einem Computer, nachgeahmt. Dafür wird sich der optimalen Lösung durch experimentelles Ausprobieren einer Vielzahl von Möglichkeiten angenähert. Obwohl die Monte-Carlo-Simulation schon eine ältere Technik ist, findet sie erst seit einigen Jahren betriebswirtschaftliche Praxisanwendungen, da sich die Simulation besser mit leistungsfähigeren Computern umsetzen lässt [31].

In „**Inverse Problem Theory and Methods for Model Parameter Estimation**“ [32] wird folgendes Beispiel für eine Monte-Carlo-Methode beschrieben, um die Zahl π annähernd zu berechnen:

Auf einem Boden, welcher aus Streifen besteht, die alle die Breite w besitzen, werden Nadeln mit der Breite $\frac{w}{2}$ geworfen. Die Wahrscheinlichkeit, dass eine Nadel eine Rille des Bodens trifft, beläuft sich auf $\frac{1}{\pi}$. Nach 50 Beobachtungen, welche jeweils 100 Würfe beinhaltet haben, wurde ein Wert für π von 3.1596 ± 0.0524 bestimmt.

Dieses Vorgehen wird heutzutage durch Pseudozufallsgeneratoren ersetzt und stellt sich als vorteilhaft heraus, wenn Integrale in mehrdimensionalen Räumen abgeschätzt werden sollen. Es stellt sich als unmöglich heraus, eine Wahrscheinlichkeitsdichte in zu großen, mehrdimensionalen Räumen darzustellen. Der Grund dafür ist, dass mit steigender Anzahl an Dimensionen auch der Leerraum zunimmt. Die Wahrscheinlichkeit, Flächen mit

Signifikanz zu treffen, sinkt immer stärker gegen Null, je mehr Dimensionen der Modellraum also umfasst. Ein anschauliches Beispiel ist der Abbildung 3.4 zu entnehmen.

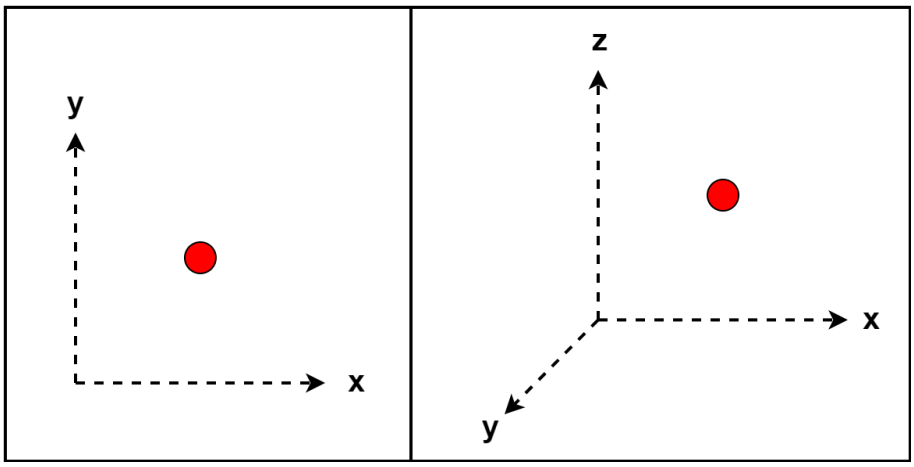


Abbildung 3.4: Mit zunehmender Anzahl von Dimensionen, steigt auch der Leerraum und erschwert somit das Treffen von Flächen mit Signifikanz.

Jedoch lassen sich Wahrscheinlichkeiten in mehrdimensionalen Räumen über Monte-Carlo-Methoden stichprobenartig erforschen, wie in Abbildung 3.5 zu erkennen ist. Dafür müssen laut Albert Tarantola [32] allerdings zwei Probleme bewältigt werden:

1) Lokalisieren von Regionen mit signifikanten Wahrscheinlichkeiten

2) Die ganze Region dicht genug erproben

Dabei stellt das Lokalisieren von Regionen mit signifikanten Wahrscheinlichkeiten das größere Problem dar, denn wie bereits erklärt, steigt mit der Anzahl der Dimension die Schwierigkeit dieser Herausforderung. Eine Möglichkeit diese Hürde zu überwinden, ist das Erproben der Regionen mit Hilfe von Zufallsbewegungen, wie zum Beispiel über den Metropolis-Algorithmus.

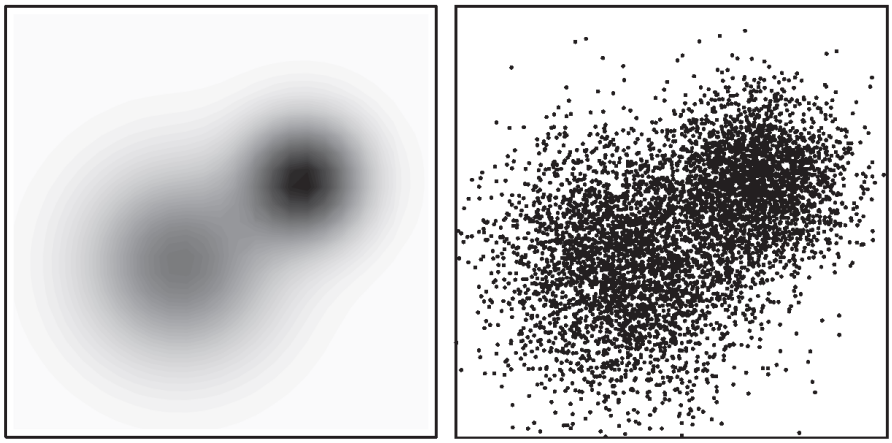


Abbildung 3.5: Stichprobenartige Erforschung einer Wahrscheinlichkeitsdichte [32].

3.2.1 Metropolis-Algorithmus

Bei dem Metropolis-Algorithmus handelt es sich um einen gedächtnislosen Monte-Carlo-Zufallsalgorithmus, was bedeutet, dass jeder Schritt nur von dem vorherigen abhängt. Mit ihm ist ein zufälliges Erproben einer Wahrscheinlichkeitsdichte möglich. Allerdings werden nicht alle Schritte, die der Algorithmus vorschlägt, auch umgesetzt. In „**Inverse Problem Theory and Methods for Model Parameter Estimation**“ [32] wird folgendes Beispiel genannt: Angenommen es sind zwei Wahrscheinlichkeitsdichten $g(\mathbf{x})$ und $f(\mathbf{x})$ und deren homogenes Limit $\mu(\mathbf{x})$ gegeben. Es ist ein Algorithmus vorhanden, mit welchem sich Proben von $f(\mathbf{x})$ aufstellen lassen. Wie muss der Algorithmus angepasst werden, um auch Erprobungen von der Konjunktion der beiden Wahrscheinlichkeitsdichten

$$h(\mathbf{x}) = k\frac{f(\mathbf{x})g(\mathbf{x})}{\mu(\mathbf{x})} \tag{3.14}$$

zu erhalten? Das genutzte Kriterium hängt nicht von den Werten von $g(\mathbf{x})$ ab, sondern von denen der assoziierten Likelihood-Funktion (vgl. Gleichung 3.16) und ist wie folgt definiert:

$$\gamma(\mathbf{x}) = \frac{g(\mathbf{x})}{\mu(\mathbf{x})} \tag{3.15}$$

Theoretisch definieren zufällige Regelungen eine Zufallsbewegung, welche die Wahrscheinlichkeitsdichte $f(\mathbf{x})$ erprobt. Zu einem bestimmten Zeitpunkt des Algorithmus wird dieser sich am Schritt $\mathbf{x_i}$ befinden und soll aufgrund der getroffenen Regelungen zum Schritt $\mathbf{x_j}$

übergehen. Diese Transition findet jedoch nur statt, wenn der Übergang den folgenden Regeln entspricht:

- Wenn $\gamma(\mathbf{x_j}) \geq \gamma(\mathbf{x_i})$ wird der Übergang durchgeführt
- Wenn $\gamma(\mathbf{x_j}) < \gamma(\mathbf{x_i})$ wird zufällig entschieden ob zu $\mathbf{x_j}$ übergegangen wird oder bei $\mathbf{x_i}$ bleibt. Die Wahrscheinlichkeit für den Übergang beträgt hier: $P_{i \to j} = \frac{\gamma(\mathbf{x_j})}{\gamma(\mathbf{x_i})}$

Somit lässt sich über diese Zufallsbewegung die Konjunktion $h(\mathbf{x})$ erproben.

3.2.2 Lösung inverser Probleme mittels Monte-Carlo

Aus Abschnitt 3.1.4 (**A Priori Informationen**) ist bekannt, dass die beiden Inputs eines inversen Problems die Wahrscheinlichkeitsdichten $\rho_M(\mathbf{m})$, welche die a priori Informationen zu den Modellparametern repräsentiert, und $\rho_D(\mathbf{d})$, welche Informationen zu den Datenparameter, die beispielsweise über Messungen erlangt wurden, sind. Die Lösung eines inverses Problems ist hierbei $\sigma_M(\mathbf{m})$, welche definiert ist als eine Multiplikation von einer Normierungskonstante k, der normierten Dichte $\rho_M(\mathbf{m})$ und der sogenannten „Likelihood"-Funktion $L(\mathbf{m})$ [32]:

$$\sigma_M(\mathbf{m}) = k\rho_M(\mathbf{m})L(\mathbf{m}) \tag{3.16}$$

$L(\mathbf{m})$ repräsentiert dabei, wie gut ein Modell zu den gewonnen Daten passt.

3.2.3 Erproben der a priori Wahrscheinlichkeitsdichte

Die a priori Wahrscheinlichkeitsdichte $\rho_M(\mathbf{m})$ ist im Gegensatz zur a posteriori Wahrscheinlichkeitsdichte $\sigma_M(\mathbf{m})$ verhältnismäßig einfach, weil sich das Erproben dort mit Hilfe von einfachen Methoden umsetzen lässt. Da diese Thematik nicht im Vordergrund dieser Arbeit steht, wird diese hier nicht weiter ausgeführt. In „**Inverse Problem Theory and Methods for Model Parameter Estimation**" [32] sind mögliche Methoden zur Erprobung erläutert und zusätzliche Beispiele nachzulesen.

3.2.4 Erproben der a posteriori Wahrscheinlichkeitsdichte

Das Erproben der a posteriori Wahrscheinlichkeitsdichte lässt sich mit Hilfe der Gleichung 3.16 realisieren. Vorausgesetzt, es stehen beliebig viele Samples der a priori Wahrscheinlichkeitsdichte $\rho_M(\mathbf{m})$ zur Verfügung, wird zu einem Zeitpunkt des Algorithmus der Übergang von Punkt $\mathbf{m_i}$ zu $\mathbf{m_j}$ bevorstehen. Ob dieser stattfindet, hängt dabei nach Albert Tarantola [32] von folgenden Regeln ab:

- Ist $L(\mathbf{m_j}) \geq L(\mathbf{m_i})$, wird der Schritt durchgeführt
- Ist $L(\mathbf{m_j}) < L(\mathbf{m_i})$, wird per Zufall entschieden, ob der Übergang getätigt wird, wobei die Wahrscheinlichkeit dafür $P_{i \to j} = \frac{L(\mathbf{m_j})}{L(\mathbf{m_i})}$ beträgt

Dabei stellen die gesamten Zufallsbewegungen des Metropolis-Algorithmus $\sigma_M(\mathbf{m})$ dar.

3.2.5 Entwurf der Zufallsbewegung

Um eine aussagekräftige Wahrscheinlichkeitsdichte $\sigma_M(\mathbf{m})$ zu erhalten, ist es von Wichtigkeit, voneinander unabhängige Stichproben zu erhalten. Dies wäre der Fall, ständen voneinander unabhängige Samples in $\rho_M(\mathbf{m})$ zu Verfügung. Diese Voraussetzung ist jedoch nur für Probleme mit wenigen Dimensionen gegeben, da durch den zunehmenden Leerraum von höherdimensionalen Problemen das Erproben von $\rho_M(\mathbf{m})$ nur über kleine Schritte erfolgen kann [32]. Somit können die produzierten a priori Erprobungen nicht voneinander unabhängig sein und somit die resultierenden a posteriori Stichproben ebenfalls nicht. Um dieses Problem zu lösen, wird nachdem ein Sample aufgenommen wurde, eine ausreichende Anzahl von Schritten abgewartet, bevor das nächste genutzt wird. Dadurch „vergisst“ der Algorithmus das erste Sample. Wie lange gewartet werden soll, kann dabei von $L(\mathbf{m})$ abhängig gemacht werden, indem erst das nächste Sample aufgenommen wird, wenn diese Funktion ausreichend klein ist. Wann der Algorithmus gestoppt wird, birgt wiederum zwei weitere Probleme. Das erste ist die Frage, wie viele Stichproben nötig sind, um ein gegebenes Maximum der Wahrscheinlichkeitsdichte $\sigma_M(\mathbf{m})$ ausreichend zu untersuchen. Das weitaus schwerere Problem ist die Möglichkeit, signifikante Stellen zu verpassen, wie zum Beispiel ein isoliertes Maximum.

3.3 Optimierungsverfahren

3.3.1 Gradientenabstieg

Das Gradientenabstiegsverfahren wird genutzt, um die Fehlerfunktion in einem Lernalgorithmus zu minimieren. Dafür werden die Parameter des Algorithmus in jedem Iterationsschritt minimal angepasst. Eins der bekanntesten Beispiele für das Gradientenabstiegverfahren ist das Hinabsteigen eines Berges. Angestrebt wird dabei das Erreichen des niedrigsten Tales des Berges und somit das globale Minimum. Um dieses zu erreichen, bewegt man sich in jedem Durchlauf in die Richtung des steilsten Abstiegs mit einer Schrittweite, die abhängig vom gewählten Algorithmus ist. Ist kein weiterer Abstieg mehr möglich, ist ein Minimum der Funktion erreicht worden. Abbildung 3.6 zeigt das prinzipielle Verfahren des Gradientenabstiegs. Es ist oftmals nicht möglich, das globale Minimum einer Funktion mit Hilfe des Gradientenabstiegsverfahrens zu bestimmen, deshalb wird das Verfahren so oft wiederholt, bis ein zufriedenstellendes Resultat erzielt wurde.

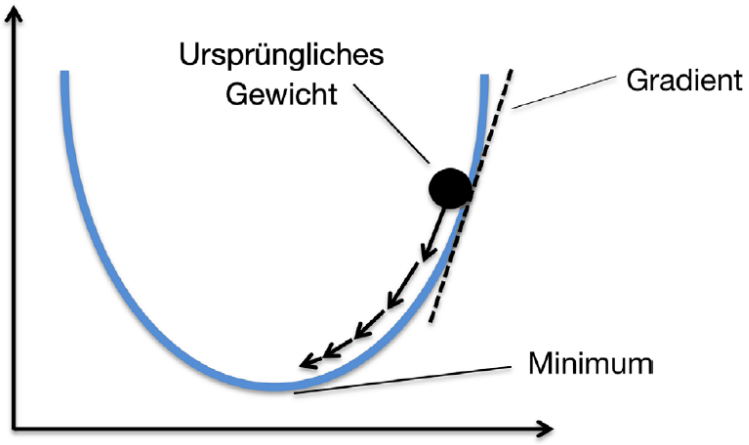


Abbildung 3.6: Prinzip des Gradientenverfahrens [22].

Um den Fehler zwischen errechneten und realen Sensorausgangswerten zu minimieren, wird auch hier das Gradientenverfahren genutzt, um die Parameter des Modells zu optimieren und letztlich dadurch die Position des zu suchenden Magneten zu ermitteln. Dafür werden die Anfangswerte des Lernalgorithmus, wie Startposition und Magnetfeldstärke des Dipols, nicht zufällig generiert, sondern vorgegeben. Es wird in diesem Fall angenommen, dass sich im ersten Suchdurchlauf der Dipol genau in der Mitte des waagerechten

Arrays in vier Zentimeter Höhe befindet. Anschließend werden Position und Winkel des Arrays sowie die Stärke der Magnetisierung in kleinen Schritten mit zufälligen Werten zwischen [-0,5; 0,5] angepasst.

Als Schwellenwertelement dient hier die Abweichung zwischen gemessenen und errechneten Werten, welcher anfänglich als ausreichend hoch angenommen wird. „Wenn es gelingt, aus der Fehlerfunktion die Richtung abzuleiten, in denen die Gewichte [...] geändert werden müssen, um den Fehler zu verringern, verfügen wir über eine Möglichkeit, die Parameter [...] zu trainieren. Wir bewegen uns einfach ein kleines Stück in diese Richtungen, bestimmen erneut die Richtungen der notwendigen Änderungen, bewegen uns wieder ein kleines Stück usf.“ [13].

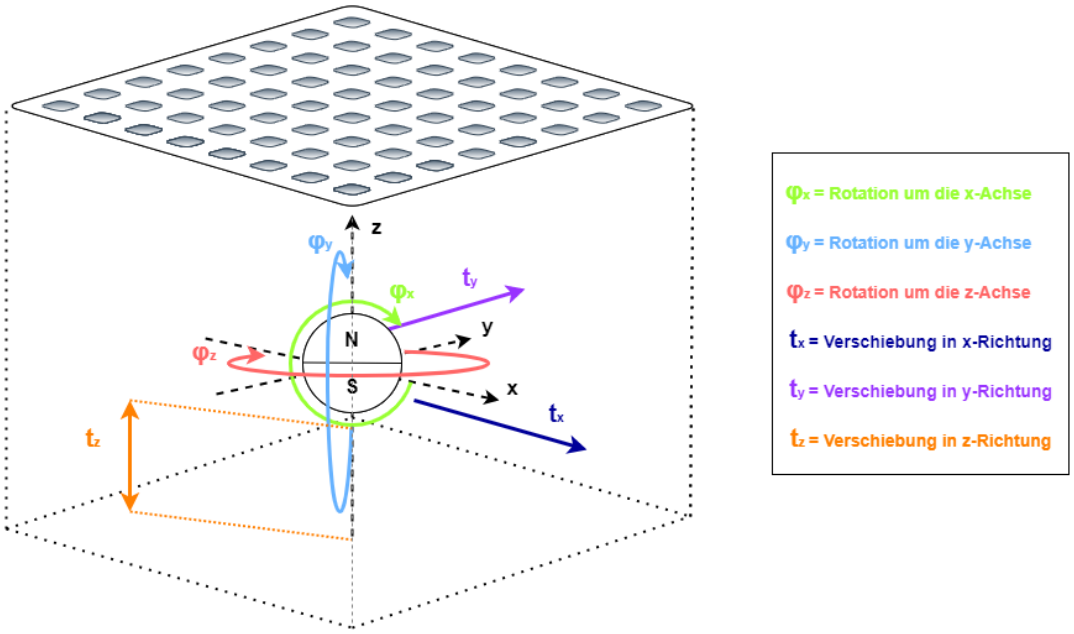


Abbildung 3.7: Darstellung der sechs Parameter, welche Einfluss auf die Messwerte des Sensor-Arrays haben.

Dabei wird, wie bereits im Abschnitt des inversen Problems erwähnt, ein hypothetisches Array mit TMR-Sensoren über einem hypothetischen Dipol modelliert und über die Dipolgleichung (Gleichung 2.2) die sich theoretisch ergebenden Sensorausgangswerte errechnet.

Um das Array und seine Messwerte zu modellieren (vgl. Listing 3.1), werden die Koordinaten der jeweiligen Sensoren xh, yh und zh [1] abhängig von der vorgegebenen Position, welche mit dem Vektor posvec übergeben wird, mit Hilfe von zwei verschachtelten for-Schleifen ermittelt. Anschließend kann über die implementierte Funktion für die Dipolgleichung (vgl. Gleichung 2.2) Dipol() die Magnetfeldstärke in x-, y- und z-Richtung an den Stellen der Sensoren berechnet werden, welche anschließend im entsprechenden Array gespeichert werden.

[1] In dieser Arbeit werden aus Gründen der Übersicht Programmcodeausschnitte, Variablen, o.ä. im Text grau hinterlegt

Listing 3.1: Berechnung der Ausgangswerte des hypothetischen Sensor-Arrays.

```
genhyp(double W_real[8][8], double W_imag[8][8],
        double posvec[], double anglevec[], double m)
{

double xh, yh, zh, Hx, Hy, Hz;

for (int j = 0; j < 8;j++)
{
      for (int i = 0; i < 8;i++)
      {
          // Abstand in mm
            xh = posvec[0] - ((j + 1) - 5 + 0.5) * 7.15;
            yh = posvec[1] - ((i + 1) - 5 + 0.5) * 7.15;
            zh = posvec[2];

            Dipol(Hx, Hy, Hz, xh, yh, zh,
                anglevec[0] * m,
                anglevec[1] * m,
                anglevec[2] * m);

            W_real[i][j] = Hx;
            W_imag[i][j] = Hy;
          }
    }
}
```

Diese Werte werden mit den real gemessenen verglichen und es wird die Summe der Quadrate der Differenzen über jedem Sensor gebildet, aus welcher anschließend die Wurzel gezogen wird. Diese Art der Fehlerfunktion wird auch „Root-mean-square error“ (RMSE) genannt. Im Listing 3.2 ist die Implementation der Fehlerberechnung zu erkennen. Hierbei wird der Fehler für die realen und imaginären Sensorausgangswerte getrennt in zwei ineinander geschachtelten for-Schleifen berechnet. Es werden die im Suchalgorithmus ermittelten Werte cos_8x8 und sin_8x8 von den echt gemessenen Werten Wh_real

und Wh_imag subtrahiert und quadriert. Anschließend werden die Wurzeln der Summen gezogen und die Beträge der Ergebnisse addiert.

Listing 3.2: Berechnung der Fehlerfunktion im Suchalgorithmus.

```
1  double aux_real = 0, aux_imag = 0;
2
3  for (int i = 0; i < ROWS8; ++i) {
4      for (int j = 0; j < COLUMNS8; ++j) {
5
6          aux_real += (Wh_real[i][j]-cos_8x8[i][j])*
7                  (Wh_real[i][j]-cos_8x8[i][j]);
8
9          aux_imag += (Wh_imag[i][j]-sin_8x8[i][j])*
10                 (Wh_imag[i][j]-sin_8x8[i][j]);
11     }
12 }
13
14 aux = abs(sqrt(aux_real)) + abs(sqrt(aux_imag));
```

Nun werden die Position und Lage des Magneten (anpassbare Parameter in Abbildung 3.7 dargestellt) mit zufälligen kleinen Zahlen verändert und erneut die Abweichung ermittelt.

Listing 3.3: Anpassung der Parameter im Suchalgorithmus.

```
1  // Learnfactor
2  fak = 2.0;
3
4  // Random angle
5  ax = fak * (2.0 / 180.0) * (((double)rand() / RAND_MAX) - 0.5);
6  bx = fak * (2.0 / 180.0) * (((double)rand() / RAND_MAX) - 0.5);
7  cx = fak * (2.0 / 180.0) * (((double)rand() / RAND_MAX) - 0.5);
8
9  // Rotation around the x, y and z-axis
10 rotationmatrix_x(mch, ax);
11 rotationmatrix_y(mch, bx);
12 rotationmatrix_z(mch, cx);
13
14 // Positioning on the x, y and z-axis
15 tch[0] = tc[0] + fak * (((double)rand() / RAND_MAX) - 0.5);
16 tch[1] = tc[1] + fak * (((double)rand() / RAND_MAX) - 0.5);
17 tch[2] = tc[2] + fak * (((double)rand() / RAND_MAX) - 0.5);
```

Im Codebeispiel 3.3 ist die Anpassung der Position und des Winkels von dem sich über dem Dipol befindenden Sensor-Arrays zu sehen. Das Array tch beinhalted hier die Lage des Arrays und die Variablen ax, bx und cx stellen die Winkel, mit denen das Array um die x-, y- oder z-Achse rotiert wird, dar. Bei einem Lernfaktor von fak=2.0 ergibt sich für eine Verschiebung in x-, y- oder z-Richtung eine maximale Schrittweite von $-0.011 \leq x \leq 0.011$, welche abhängig ist von der sich ergebenden zufälligen Zahl ((double)rand() / RAND_MAX) - 0.5), mit welcher multipliziert wird. Verschoben wird das hypothetische Array mit einer maximalen Schrittweite von $-1 \leq x \leq 1$, wieder abhängig von der zufälligen Zahl des Terms. Ist der Fehler kleiner geworden, werden die neuen Werte als Basis für die weiteren Berechnungen genommen, andernfalls bleiben die vorherigen Werte als Basis der Suche bestehen. Im nächsten Suchdurchlauf werden die gegebenenfalls neuen oder andernfalls die alten Werte wieder mit neuen, anderen zufälligen Zahlen verändert [13]. Im Codebeispiel 3.4 ist die Übernahme der neuen Parameter, wenn der alte Fehlerwert mit diesen unterschritten wurde, zu sehen. Dabei stellen mch das Array für die

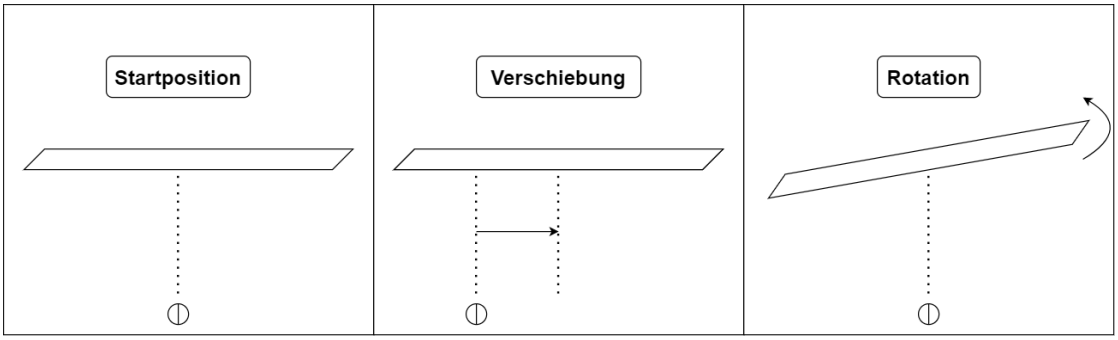


Abbildung 3.8: Skizzierung der Verschiebung und Rotation des simulierten Sensor-Arrays.

Magnetisierung und tch die Verschiebung des Sensor-Arrays in x-, y-, und z-Richtung dar.

Listing 3.4: Übernahme der neuen Parameter im Suchalgorithmus.

```
1 if (aux < auxmin)
2 {
3     for (int i = 0;i < 3;i++)
4     {
5         mc[i] = mch[i];
6         tc[i] = tch[i];
7     }
8 }
```

Das Verfahren des Lernalgorithmus wird solange wiederholt, wie der Fehler zwischen den gemessenen und errechneten Werten kleiner wird und bis schlussendlich keine Verbesserung mit der implementierten Schrittweite mehr erreicht werden kann [15]. Nach dem Erreichen der maximalen Anzahl von Suchdurchläufen stellen die zum Schluss ermittelten besten Werte der Suche das resultierende Schätzwertemodell dar. Der Ablauf lässt sich in Abbildung 3.9 noch einmal anschaulich nachvollziehen.

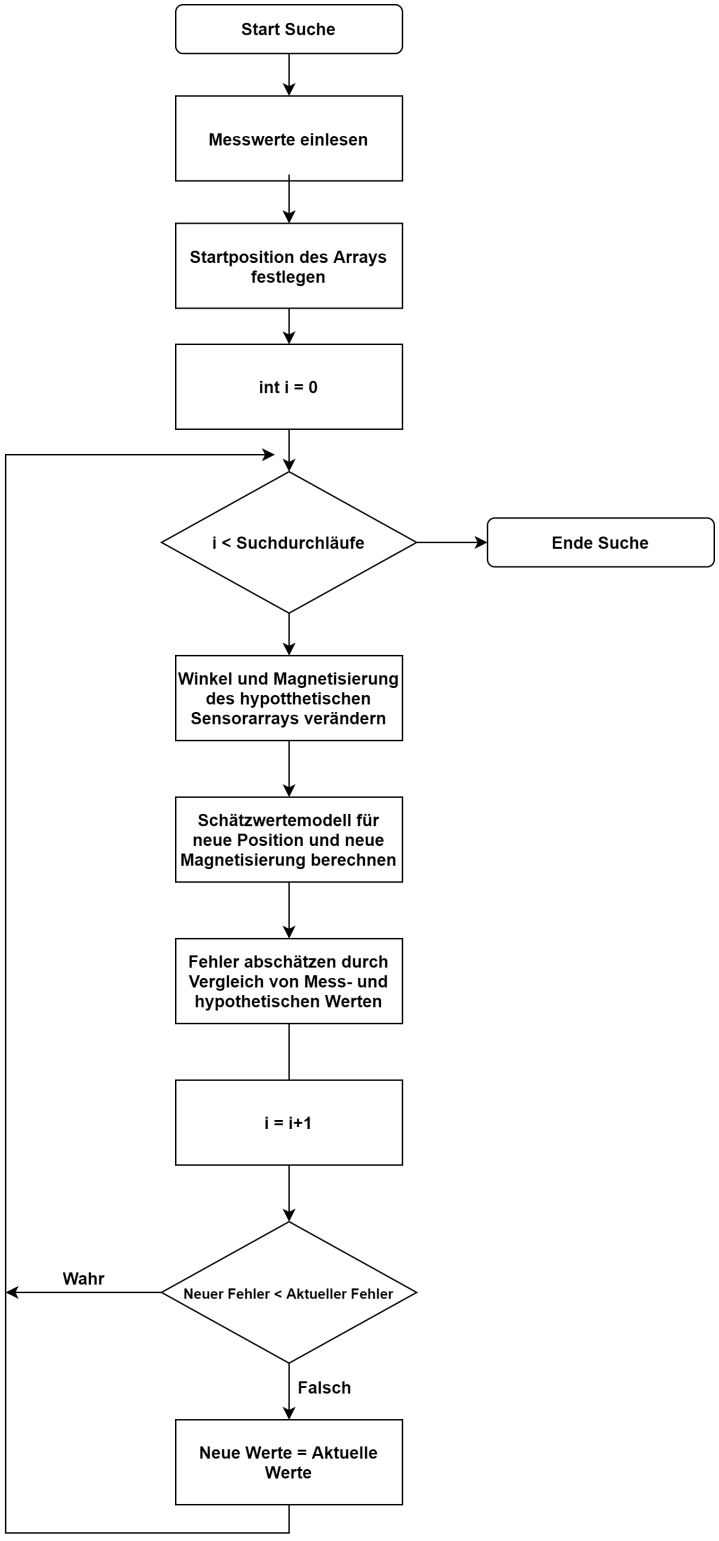


Abbildung 3.9: Ablaufdiagramm für die Suche der Magnetposition.

4 Entwicklung einer Crossplattform GUI in C++

4.1 Vorteile einer Neuimplementation in C++

Weshalb es sinnvoll ist, eine Software plattformübergreifend zu gestalten, ist vermutlich selbsterklärend. Warum eine Neuimplementation des Suchalgorithmus unter C++ einige Vorteile birgt, ist möglicherweise nicht so offensichtlich. Dabei kann eine Neuimplementation in dieser Programmiersprache eine markante Zunahme der Prozessgeschwindigkeit bedeuten. Das liegt mitunter daran, dass der Nutzer vollständige Kontrolle über die Speicherverwaltung des zu erstellenden Programms besitzt. Bei anderen Programmiersprachen wird zum Beispiel der Speicher je nach Bedarf dynamisch alloziert, was unter Umständen gar nicht vom Anwender gewünscht ist und das Programm bei langen Laufzeiten überlädt und schlussendlich verlangsamt. Dem lässt sich in C++ entgegenwirken, indem der benötigte Speicher passend für die Anwendung im Vorfeld freigegeben wird. Des Weiteren besitzt es von Anfang an eine gewisse Plattformunabhängigkeit, wodurch es relativ einfach ist, die entworfene Applikation auf verschiedenen Betriebssystemen lauffähig zu gestalten, was genau eins der Ziele dieser Arbeit darstellt. Durch das objektorientierte Programmieren in C++ können mehrere Objekte derselben Klasse erzeugt werden, wodurch sich das Erweitern des Programmcodes als verhältnismäßig einfach einstufen lässt. Auch die Nutzung von mehreren Threads, um verschiedene Prozesse der zu entwickelnden Anwendung zu parallelisieren, stellt sich als vorteilhaft heraus und ist in dieser Programmiersprache gut umsetzbar.

4.2 Notwendige Bibliotheken

4.2.1 WxWidgets

Abbildung 4.1: WxWidgets Logo [37].

Bei „WxWidgets“ handelt es sich um ein Crossplattform Toolkit, um grafische Benutzeroberflächen zu designen. Gestartet wurde das Projekt 1992 von Julian Smart, welcher zu der Zeit an der Universität von Edinburgh gearbeitet hat. Zuerst war das Ziel, Anwendungen portabel zwischen UNIX und Windows gestalten zu können. Doch mittlerweile sind noch wesentlich mehr Betriebssysteme zum Repertoire hinzugefügt worden, was vermutlich auch an der Anzahl der Teilnehmer des Projekts liegt, welche mittlerweile bis in den Hunderterbereich vorgedrungen ist. „WxWidgets“ stellt eine einfache Anwendungsprogrammierschnittstelle dar, um GUIs in der bevorzugten Programmiersprache zu implementieren. Zur Verfügung stehen hierbei unter anderem C++ und Python. Es nutzt die betriebssystemeigenen Designs und Einstellungen für das Aussehen des Interfaces, damit sich die entworfene Anwendung auch nach der genutzten Plattform anfühlt [37]. Die in dieser Arbeit hauptsächlich genutzten Features und Klassen werden in den folgenden Abschnitten kurz vorgestellt.

Events

Events werden genutzt, um implementierte Bedienelemente mit Funktionen zu verknüpfen. Sie sind die Verbindung zwischen GUI und dem implementierten Programmcode. Wird beispielsweise der „Play-Button“ geklickt, startet ein neuer Thread, welcher die Messwerte vom Sensor-Array ausliest. Wird hingegen der „Stop-Button“ ausgewählt, so wird der Thread wieder beendet. Die Verknüpfung davon findet in einer sogenannten Event Table statt, wo das Bedienelement über seine spezifische ID mit der gewünschten Funktion gekoppelt wird (vgl. Listing 4.1). Weitere Informationen zu Events sind unter folgendem Link nachzulesen: https://docs.wxwidgets.org/trunk/overview_events.html

Listing 4.1: Verknüpfung von „Start-“ und „Stop-Button“ mit der zugehörigen Funktion.

```
1 wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
2 EVT_BUTTON(ID_BTN_START, MyFrame::OnStart)
3 EVT_BUTTON(ID_BTN_STOP, MyFrame::OnStop)
4 wxEND_EVENT_TABLE()
```

Multithreading

Durch Multithreading lassen sich Prozessabläufe der Anwendung parallelisieren und es stellt einen Kernbaustein dieser GUI dar. Wird die Messung über „Start“ begonnen, werden kontinuierlich neue Werte vom Sensor-Array ausgelesen. Dies geschieht in einer Endlosschleife, welche die Bedienung des Interfaces blockieren würde, da die Anwendung fortwährend mit dem Auslesen beschäftigt wäre. Durch Multithreading lassen sich verschiedene Prozesse simultan im Hintergrund bearbeiten und es ist möglich, die Werte auszulesen und gleichzeitig darzustellen. Um dieses Feature unkompliziert zu integrieren, bietet „WxWidgets“ unter anderem die Klasse „wxThreadHelper“ an. Vererbt man zum Beispiel der Klasse des Hauptfensters die Methode „wxThread::ExitCode Entry()“ vom „wxThreadHelper“, erhält diese einen Thread, welcher im Hintergrund abläuft. Die Methode „wxThread::ExitCode Entry()“ läuft bis die in ihr aufgerufene Funktion abgeschlossen oder vom Anwender angefragt wurde den Thread zu beenden. Das Vorgehen ist in Listing 4.2 vereinfacht dargestellt. Mehr zu Multithreading in wxWidgets ist hier zu finden: https://docs.wxwidgets.org/trunk/overview_thread.html

Listing 4.2: Vererbung der Thread-Funktionalität (Zeile 1-6) und Thread-Aufruf (Zeile 6-10).

```
1  class MyFrame : public wxMDIParentFrame, public wxThreadHelper
2  {
3  private:
4  virtual wxThread::ExitCode Entry();
5  }
6  wxThread::ExitCode MyFrame::Entry() {
7   while (!GetThread()->TestDestroy())
8       {
9       ... // function()
10      }
```

Sizer

Sizer sind der Grundbaustein zum Organisieren von Anwendungselementen auf der grafischen Benutzeroberfläche. Es stehen verschiedene Typen von Sizern zur Verfügung, darunter Boxsizer und Gridsizer. Es lassen sich unterschiedliche Einstellungen vornehmen, damit Elemente, die sich auf der Oberfläche befinden, zum Beispiel links- oder rechtsbündig sowie horizontal oder vertikal angeordnet werden können. Der Boxsizer besitzt eine quadratische Form und je nach Einstellung werden die ihm zugefügten Bedienstrukturen an die jeweiligen Stellen deponiert. Der Gridsizer strukturiert seine Elemente in der Form einer zweidimensionalen Tabelle mit NxM Feldern. Im nachfolgenden Listing 4.3 lässt sich das Vorgehen anhand eines Beispiels nachvollziehen:

Listing 4.3: Erstellen von drei Boxsizern (Zeile 1-3) und exemplarisches Hinzufügen von weiteren Sizern (Zeile 5 u. 7) bzw. dem Bild-Panel (Zeile 6).

```
1 wxBoxSizer* sizer_1 = new wxBoxSizer(wxHORIZONTAL);
2 wxBoxSizer* sizer_2 = new wxBoxSizer(wxHORIZONTAL);
3 wxBoxSizer* sizer_3 = new wxBoxSizer(wxVERTICAL);
4
5 sizer_1->Add(sizer_2, 1, wxEXPAND, 0);
6 sizer_2->Add(panel_image, 1, wxEXPAND, 0);
7 sizer_1->Add(sizer_3, 1, wxEXPAND, 0);
```

Um mehr über Sizer nachzulesen, kann folgende Seite genutzt werden: https://docs.wxwidgets.org/3.0/overview_sizer.html

Panel

Ein Sizer benötigt noch ein Panel, damit eine Oberfläche zur Verfügung steht, auf welcher die ihm zugefügten Elemente angeordnet werden können. In dieser Arbeit werden für die Oberfläche im Hauptfenster zwei Panels zur Organisation genutzt: Bild- und Control-Panel. Auf dem Bild-Panel wird eine Bitmap-Datei abgelegt, auf welcher später die Grafiken für die Messungen gezeichnet werden. Die Bedienelemente der GUI befinden sich auf dem Control-Panel. Im Listing 4.4 wird gezeigt, wie die beiden Panels erzeugt werden und in Listing 4.5 lässt sich bei dem Erzeugen eines Buttons erkennen, dass als ihr Bezugspunkt panel_control angegeben wurde. Für weitere Informationen zu Panels

lässt sich die Dokumenation hier nachlesen: https://docs.wxwidgets.org/3.0/overview_sizer.html

Listing 4.4: Erstellen von zwei Panels.

```
wxPanel* panel_controls;
wxPanel* panel_image;

panel_controls = new wxPanel(this, wxID_ANY);
panel_image = new wxPanel(panel_controls, wxID_ANY);
```

Buttons

Buttons sind eine weitere Möglichkeit für den Benutzer, um mit der grafischen Benutzeroberfläche zu interagieren. Es stehen zwei Arten zur Auswahl, welche die mit Text gekennzeichnet werden oder Bitmap-Buttons, die durch Bilder auf ihre Funktion aufmerksam machen. Es können spezifische IDs an die Buttons vergeben werden, um diese später über die Event-Tabelle mit den zugehörigen Funktionen zu verknüpfen, wie es bereits in Abschnitt 4.2 Events angesprochen wurde. Alles zur Dokumentation von Buttons in wxWidgets und deren Funktionsweise ist hier nachzulesen: https://docs.wxwidgets.org/trunk/classwx_button.html

Listing 4.5: Erstellen von zwei Bitmap-Buttons.

```
wxBitmapButton* play_button = nullptr;
wxBitmapButton* stop_button = nullptr;

play_button = new wxBitmapButton(panel_controls, ID_BTN_START,
            wxBitmap(wxT("C:/.../icons8-play-64.png"),
            wxBITMAP_TYPE_ANY));
stop_button = new wxBitmapButton(panel_controls, ID_BTN_STOP,
          wxBitmap(wxT("C:/.../icons8-stop-64.png"),
          wxBITMAP_TYPE_ANY));
```

Checkbox

Durch Hinzufügen von Checkboxen erhält der Benutzer eine einfache Bedienmöglichkeit, um beispielsweise Einstellungen vornehmen zu können. In dieser Arbeit wird diese Art der Bedienung genutzt, um zwischen unterschiedlichen Plotmöglichkeiten wählen zu können. Durch eine einfache Abfrage, ob das entsprechende Auswahlkästchen markiert ist, lässt sich in dem Thread des Hauptfensters die zugehörige Darstellungsform auswählen (4.6 Zeile 13-14). Für weitere Informationen zu Checkboxen, steht anschließender Link zur Verfügung: https://docs.wxwidgets.org/trunk/classwx_check_box.html

Listing 4.6: Erstellen einer Checkbox und beispielhafte Abfrage in ThreadHelper Entry().

```
1  wxCheckBox* m_CheckBox_Contour = nullptr;
2  m_CheckBox_Contour = new wxCheckBox(panel_controls,
3                ID_CBOX_CONTOUR, wxT("Contour Plot"),
4                wxDefaultPosition, wxDefaultSize,
5                0);
6  sizer_3->Add(m_CheckBox_Contour, 0, 0, 0);
7
8  ...
9
10 wxThread::ExitCode MyFrame::Entry() {
11     while (!GetThread()->TestDestroy())
12     {
13         if (m_CheckBox_Contour->IsChecked()){
14               contour_plot();
15          }
16     }
17 }
```

Choicebox

Für Auswahllisten mit Drop-Down-Menü steht in WxWidgets die Klasse ChoiceBox zur Verfügung. Mit ihr lassen sich mehrere Bedienelemente in einer Liste vereinen, welche mit unterschiedlichen Funktionen gekoppelt sein können. In dieser GUI befindet sich im Hauptfenster eine Auswahlliste, um den gewünschten Filtertyp einzustellen. Über ein

wxString-Array lassen sich Namen für die Einträge zu der ChoiceBox hinzufügen und über SetSelection() wird eingestellt, welcher Eintrag standardmäßig eingestellt sein soll (vgl. Listing 4.7).

Listing 4.7: Erstellen der Filter-Auswahlliste.

```
1  wxString m_choice1Choices[] = { wxT("No Filter"),
2                                  wxT("Filter 1"),
3                                  wxT("Filter 2"),
4                                  wxT("Filter 3")};
5  int m_choice1NChoices = sizeof(m_choice1Choices) /
6                          sizeof(wxString);
7  m_choice_filter = new wxChoice(panel_controls, wxID_ANY,
8                          wxDefaultPosition, wxDefaultSize,
9                          m_choice1NChoices, m_choice1Choices, 0);
10 m_choice_filter->SetSelection(0);
11 sizer_3->Add(m_choice_filter, 0, 0, 0);
```

4.2.2 Chartdirector

Mit Hilfe von ChartDirector werden die Grafiken, welche für die Darstellung der realen und berechneten Werte genutzt werden sollen, erstellt. Dafür steht eine Vielzahl von verschiedenen Diagrammarten zur Verfügung, unter anderem auch Vektor-, Kontur-, Kurven- und Balkendiagramme, welche sich wiederum weiter in unterschiedliche Ausführungsmöglichkeiten aufteilen. Es werden laut Quelle [4] eine Vielzahl von Programmiersprachen (C++, Python, Java) und Betriebssystemen (Windows, MacOS, Linux) unterstützt. Mit ChartDirector erstellte Grafiken können zudem nicht nur auf dem Bildschirm angezeigt werden, sondern auch als Datei lokal auf dem Computer gespeichert werden. Beispiele für die verwendeten Grafiken folgen später im Abschnitt 4.4 **Implementation der grafischen Benutzeroberfläche**. Um ein Diagramm zu implementieren, muss im Programmcode zuerst ein Objekt vom Typen Chart, wie zum Beispiel XYChart (vgl. Listing 4.8 Zeile 1), erstellt werden, welchem anschließend die gewünschte Darstellungsform über einen sogenannten Layer angefügt wird.

Abbildung 4.2: Chartdirector Logo [4].

Dieser Layer enthält die Information der gewählten Diagrammart und der zu zeichnenden Datenpunkte (vgl. Listing 4.8 Zeile 2-5).

Listing 4.8: Erstellen eines Kontur-Diagramms mit Chartdirector.

```
1 XYChart* chart_mainframe = nullptr;
2 ContourLayer* layer = chart_mainframe->addContourLayer(
3                       DoubleArray(dataX, 8),
4                       DoubleArray(dataY, 8),
5                       DoubleArray(datR, 64));
```

4.2.3 Serial.h

Die Serial-Library von „Wjwwood“ wird laut Quelle [35] genutzt, um RS-232-Geräte über serielle Schnittstellen mit Hilfe von C++ anbinden zu können. Da es auf mehreren Betriebssystemen unterstützt wird, eignet sich diese Bibliothek für die Umsetzung von plattformübergreifenden Projekten.

Um eine Verbindung zu einem angeschlossenen Gerät unter Windows herzustellen, muss vor allem spezifiziert werden, an welchem COM-Port das Gerät angeschlossen wurde und mit welcher Baudrate die Symbole übertragen werden sollen. Dafür wird ein Objekt der Klasse Serial mit den gewünschten Parametern erstellt, wie es in Listing 4.9 exemplarisch gezeigt wird.

Listing 4.9: Anbindung des Arrays über COM-Port 8 mit einer Baudrate von 625000 Bd.

```
1 Serial* my_serial;
2 my_serial = new Serial("COM8",
3               625000,
4               serial::Timeout::simpleTimeout(1000));
```

4.2.4 CMake

CMake ist der Kernbaustein zur Umsetzung der plattformübergreifenden Funktionalität. Es handelt sich hierbei um eine Open-Source-Software, welche eingesetzt wird, um den Prozess des Kompilierens zu steuern, indem Konfigurationsdateien erstellt werden (vgl. Listing 4.10), die je nach Entwicklungsumgebung und Compiler das passende Makefile erzeugen. Diese Files werden „CMakeLists.txt“ genannt und werden in jedem Quellverzeichnis der Anwendung platziert. Durch diese kann es die benötigten Dateien, Bibliotheken und ausführbare Programme ausfindig machen und speichert diese Informationen in einem Cache ab.

Abbildung 4.3: CMake Logo [4].

Listing 4.10: Beispielhaftes „CMakeLists.txt“-File für das Einbinden der „Serial.h“-Bibliothek von Wjwwood.

```
cmake_minimum_required (VERSION 3.8)

if(WIN32)
    add_executable (main WIN32
                    "serial.cc"
                    "win.cc"  "serial/impl/win.h"
                    "list_ports_win.cc")
endif()
if(APPLE)
    add_executable (main MACOSX_BUNDLE "main.cpp" "CmakeWidget.h"
                    "serial.cc"
                    "unix.cc" "serial/impl/unix.h"
                    "list_ports_linux.cc")
endif()
if(UNIX)
    add_executable (main "main.cpp" "CmakeWidget.h"
                    "serial.cc"
                    "unix.cc" "serial/impl/unix.h"
                    "list_ports_linux.cc")
endif()
```

4.3 Softwarestruktur

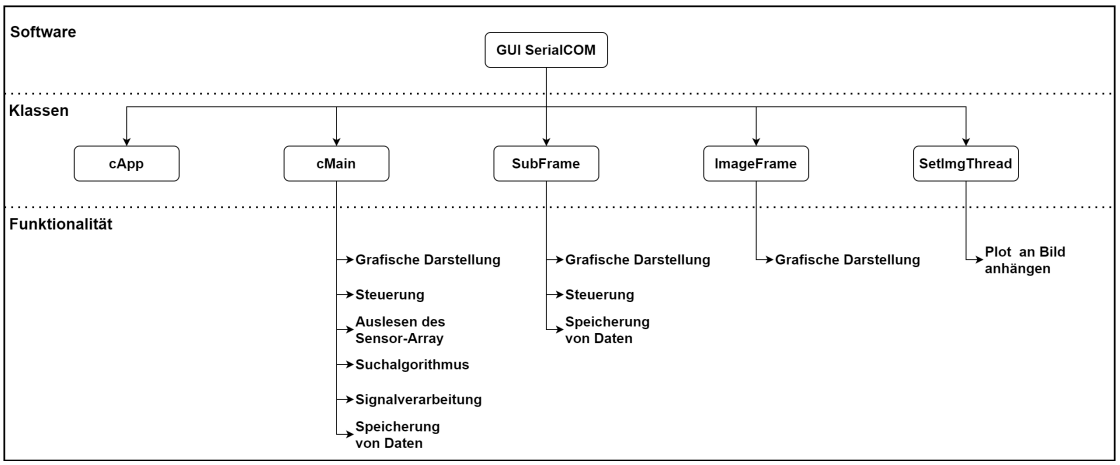


Abbildung 4.4: Grobe Strukturierung der implementierten Software.

Um die hier entwickelte Software ausreichend zu präsentieren, wird in diesem Abschnitt auch der prinzipielle Aufbau näher erläutert. Es wird zuerst näher auf die verwendeten Klassen und deren Funktionalität eingegangen und im Anschluss die wichtigsten Funktionen von diesen näher beschrieben. Es soll ein Eindruck entstehen, wie die Software funktioniert und die einzelnen Komponenten von dieser zusammenspielen. Die sieben hauptsächlichen Aufgaben, von welcher jede erstellte Klasse mindestens eine übernimmt, sind dabei wie folgt gegliedert:

1) Grafische Darstellung Die Aufgabe der grafischen Darstellung beinhaltet alles, was nötig ist, um dem Anwender eine ansprechende Darstellung der Messergebnisse zu präsentieren. Dazu zählen vor allem das Erstellen und Plotten der ausgewählten Grafiken, aber auch das Erstellen der zugehörigen Bitmap-Dateien und das Löschen von nicht mehr gebrauchten Daten, um den Prozessspeicher nicht zu überladen.

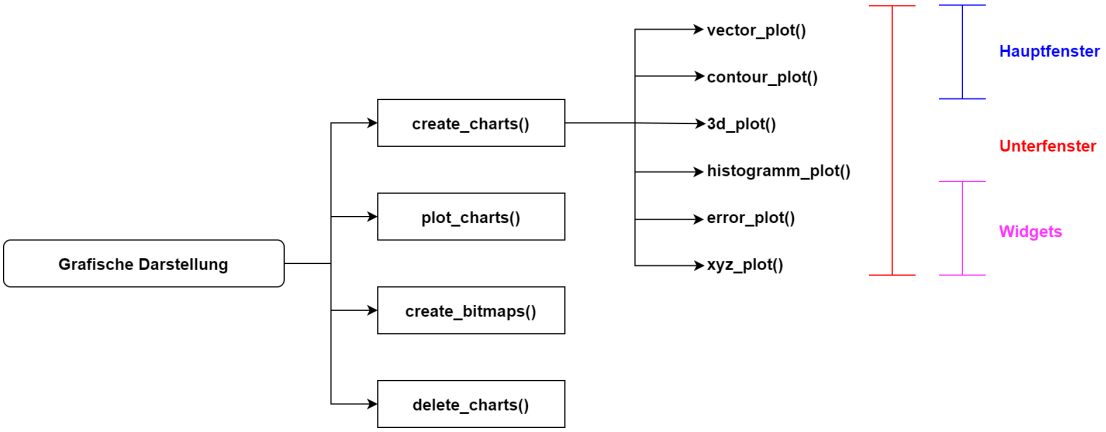


Abbildung 4.5: Grobe Übersicht der hauptsächlich genutzten Funktionen für die grafische Darstellung sowie deren Beziehungen zueinander.

2) Steuerung Unter den Begriff der Steuerung fallen in dieser Arbeit alle möglichen Einstellungen, welche der Benutzter vornehmen kann, um mit der implementierten Software zu interagieren. Gemeint sind damit alle auf der grafischen Benutzeroberfläche untergebrachten Bedienelemente, wie zum Beispiel Buttons und Kontrollkästchen. Es ist anzumerken, dass in Abbildung 4.7 Funktionen aufgelistet sind, welche über die **Event Table** mit den entsprechenden Buttons verknüpft worden sind. Die Kontrollkästchen interagieren über if-Abfragen mit den zugehörigen Funktionen, indem beispielsweise in einer while-Schleife abgefragt wird, ob dieses ausgewählt wurde. Ein Beispiel dazu folgt im späteren Textverlauf (vgl. Abschnitt 4.4.2 Listing 4.11).

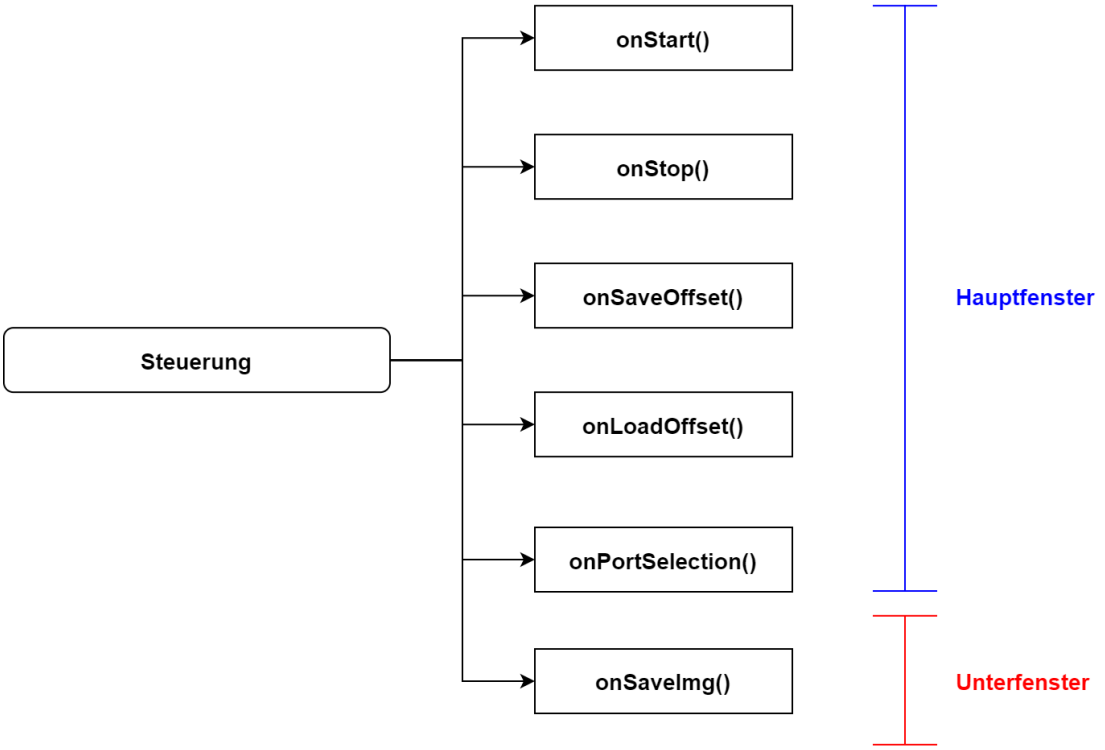


Abbildung 4.6: Grobe Übersicht der hauptsächlich genutzten Funktionen für die Interaktion zwischen Bediener und entwickelter Software sowie deren Beziehungen zueinander.

3) Auslesen des Sensor-Arrays Das Auslesen des Sensor-Arrays beinhaltet im Grunde nur eine Funktion, welche die Aufgabe übernimmt die Messergebnisse in entsprechenden Arrays des Programms zu speichern, damit diese für den Rest der Software zugänglich gemacht werden.

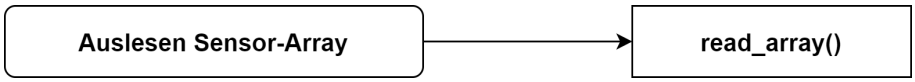


Abbildung 4.7: Übersicht der genutzten Funktionen für das Auslesen der Ausgangswerte des Sensor-Arrays.

4) Suchalgorithmus Das Suchen der Magnetposition ist eine der wichtigsten Aufgaben der implementierten Software und wie der zugehörige Algorithmus funktioniert, wurde in Kapitel 3 **Algorithmusentwicklung** ausführlich behandelt. Dort befindet sich

auch ein Ablaufdiagramm, nach welchem Schema die Suche arbeitet. Eine zugehörige Funktionsübersicht ist in Abbildung 4.8 wiederzufinden.

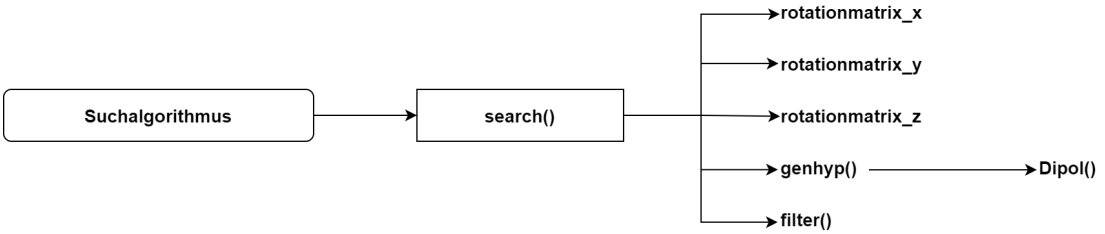


Abbildung 4.8: Grobe Übersicht der hauptsächlich genutzten Funktionen für den verwendeten Suchalgorithmus sowie deren Beziehungen zueinander.

5) Signalverarbeitung Warum die Aufgabe der Signalverarbeitung wichtig ist, wurde im Grundlagenteil **Vorverarbeitung von Rohdaten** näher erläutert. Zusammenfassend lässt sich sagen, dass diese Funktion ein wichtiger Bestandteil ist, um die Genauigkeit und Reliabilität des Suchalgorithmus zu steigern.

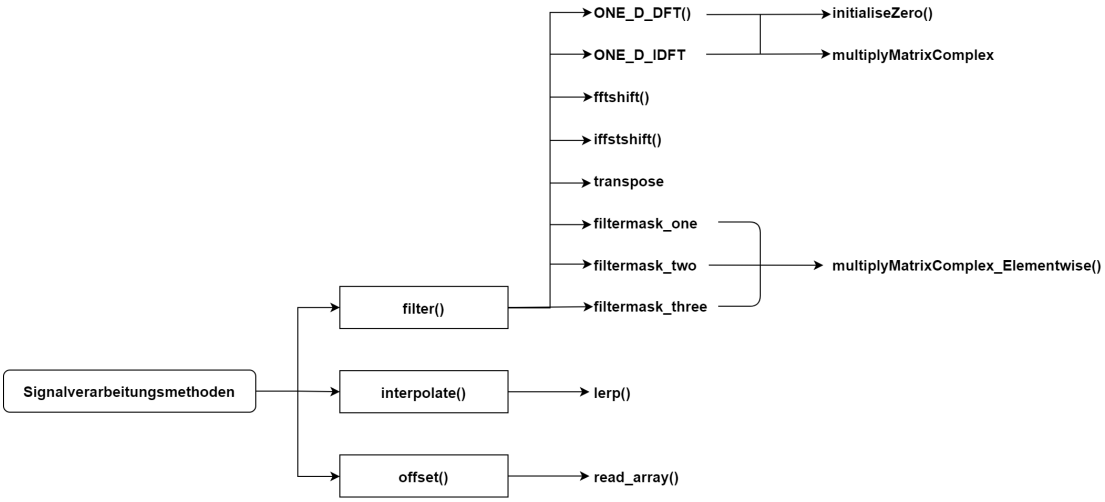


Abbildung 4.9: Grobe Übersicht der hauptsächlich genutzten Funktionen für die Methoden der Signalverarbeitung sowie deren Beziehungen zueinander.

6) Speicherungen von Daten Um die durch die Software gewonnen Messergebnisse festzuhalten, stehen verschiedene Möglichkeiten zur Verfügung, diese auf dem verwendeten Computer zu speichern. Einerseits kann ein ermittelter Offset als eine dat-Datei gespeichert und geladen werden, andererseits können erstellte Plots als PNG-Datei im

gewünschten Dateipfad abgelegt werden. Auch eine Ausgabe als Textdatei von wichtigen Kenngrößen der Suche, wie zum Beispiel Position, Abweichungen, Mittelwerte, etc. ist in der Software vorgesehen worden.

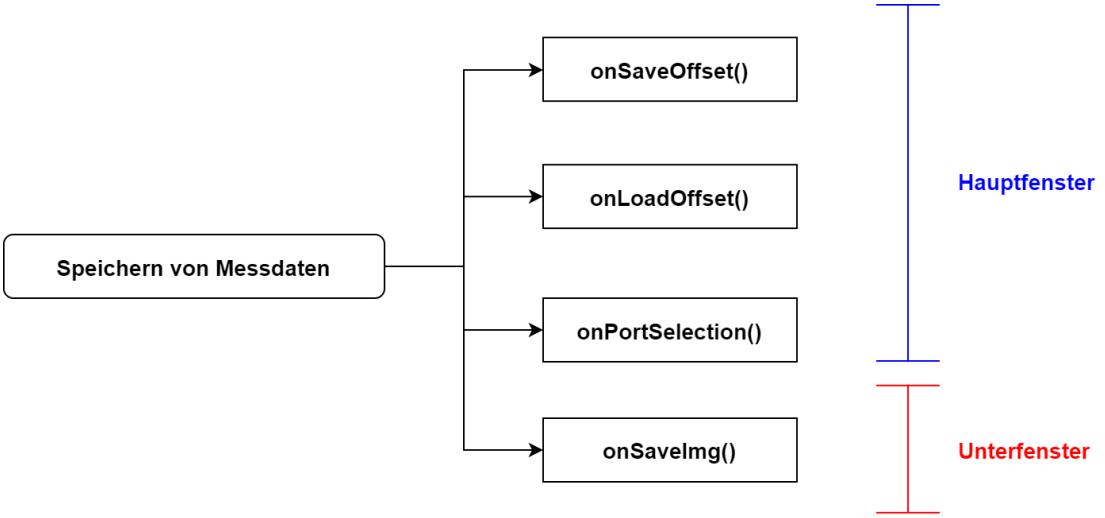


Abbildung 4.10: Grobe Übersicht der hauptsächlich genutzten Funktionen für die Methoden der Signalverarbeitung sowie deren Beziehungen zueinander.

7) Plot an Bild-Datei anhängen Um die erstellten Grafiken auszugeben, nutzt **Chartdirector** eine der grafischen Benutzeroberfläche bereits angehängte Bild-Datei. Es wird der für diese Datei geschaffene Speicher genutzt, um den Plot an diese Stelle zu schreiben und somit auf der Oberfläche auszugeben. Wird dieser Prozess während der Ausführung unterbrochen, führt das zu einem Fehler, welches das Programm einfrieren lässt, weshalb der Hilfsthread **SetImgThread** implementiert worden ist. Dieser nutzt nur eine bereits vorhandene Funktion von ChartDirector, aus welchem Grund hier keine Funktionsübersicht nötig ist. Ein Codebeispiel dazu ist in Abschnitt 4.4.5 **Hilfsthread - SetImgThread** im Listing 4.23 wiederzufinden.

4.4 Implementation der grafischen Benutzeroberfläche

In diesem Abschnitt sollen die genannten Aufgaben aus dem vorherigen Unterkapitel aufgegriffen und für jede erstellte Klasse der Applikation anhand einiger Beispiele erläutert werden. Dabei wird versucht, den Fokus auf die wichtigsten Aspekte des Programms zu legen, um den Umfang dieser Arbeit nicht zu überschreiten.

4.4.1 Applikation - cApp

Bei der Klasse **cApp** handelt es sich um die eigentliche Applikation, welche aufgerufen wird, sobald die Software gestartet wird. In dieser wird ein Objekt der Klasse des Hauptfensters **cMain** erstellt.

4.4.2 Hauptfenster - cMain

Diese Klasse erzeugt das Hauptfenster der hier entworfenen Applikation, welches sich öffnet, sobald die Anwendung gestartet wird (vgl. Abbildung 4.11). Sie ist das Kernstück der Software und in ihr befindet sich der Großteil der implementierten Funktionen, welche für die erforderlichen Aufgaben nötig sind. Über ein erzeugtes Objekt dieser Klasse kann das Sensor-Array ausgelesen, der Suchalgorithmus gestartet, Messergebnisse angezeigt und die Software gesteuert werden. Es ist anzumerken, dass die hier erstellte Klasse von einer anderen Klasse **wxThreadHelper** erbt, um einen Hintergrundthread zu erhalten. Dadurch wird die Funktion `Entry()` vererbt, in welcher der Großteil der Aufgaben bewältigt wird. In dieser neuen Funktion befindet sich eine Endlosschleife, in welcher abgefragt wird, welche Funktionen ausgeführt werden sollen. Dies wiederum hängt davon ab, welche Bedienelemente markiert oder angewählt wurden. Das Hauptfenster teilt sich in zwei Unterbereiche auf: Bild- (Abbildung 4.11 1) und Control-Panel (Abbildung 4.11 2-7).

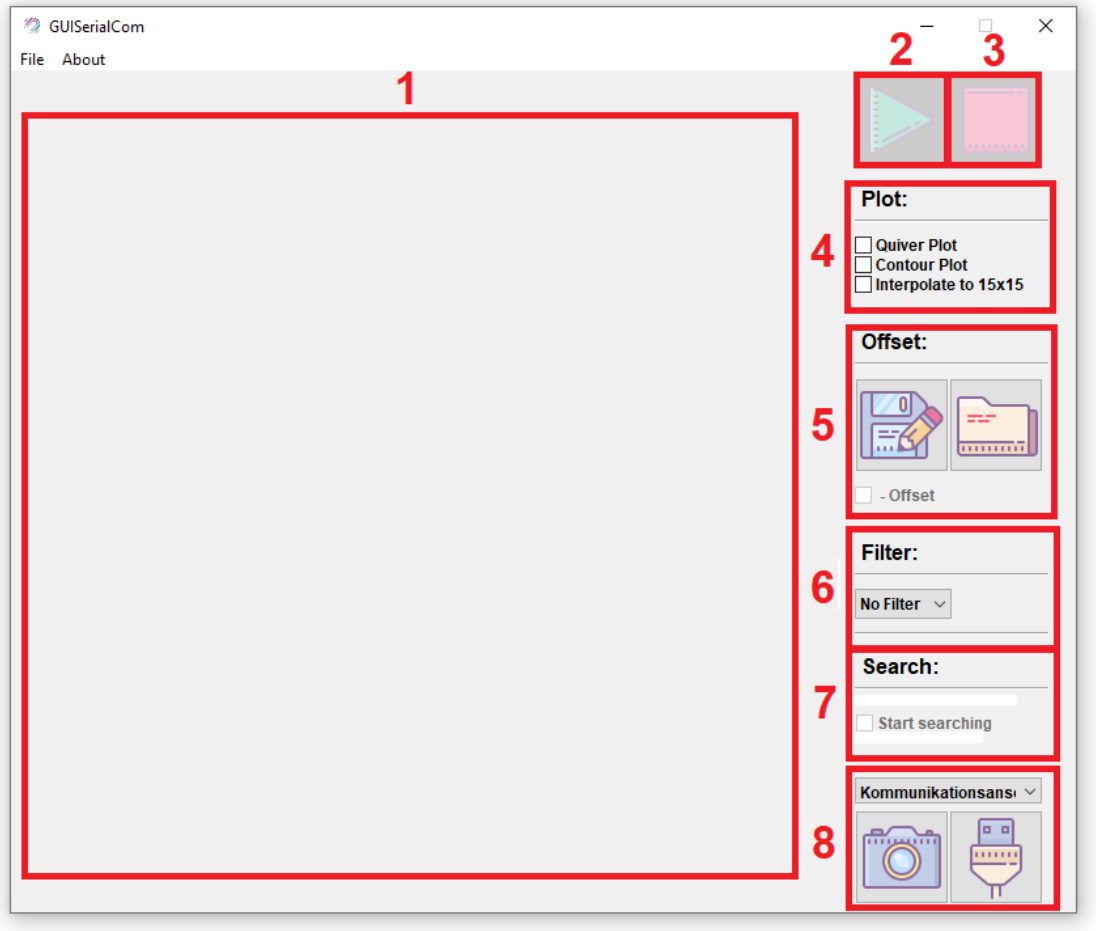


Abbildung 4.11: Hauptfenster der Anwendung.

Grafische Darstellung

Im Bild-Panel werden die aktuell eingelesenen Sensorausgangswerte je nach gewählter Darstellungsform (Abbildung 4.11 (4)) angezeigt. Der Anwender hat dabei die Möglichkeit, sich die Messdaten entweder in einem Kontur- oder einem Vektordiagramm darstellen zu lassen. Durch zusätzliches Auswählen von Interpolation, kann die Auflösung der Diagramme von 8x8 noch auf 15x15 erhöht werden. Auch eine Mehrfachauswahl ist hier möglich, um die Diagramme miteinander zu kombinieren (Abbildung 4.12).

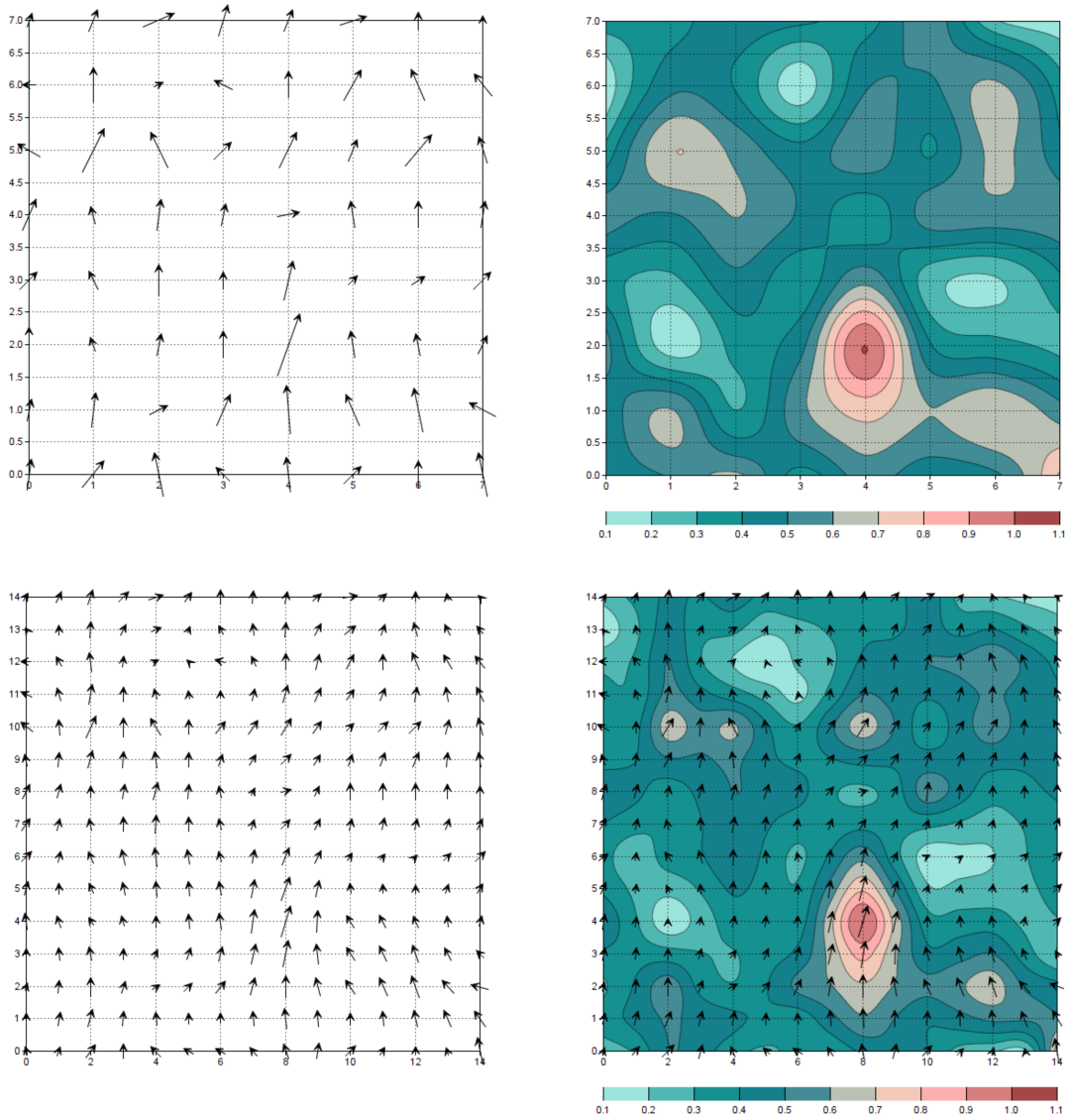


Abbildung 4.12: Verschiedene Möglichkeiten um die Messwerte darzustellen. Oben links: Vektorplot, oben rechts: Kontur-Plot, unten links: Interpolierter Vektorplot, unten-rechts: Interpolierter Kontur- und Vektorplot.

Steuerung

Im Control-Panel befinden sich die Bedienelemente über welche der Nutzer mit der Anwendung interagieren kann, wobei die wichtigsten davon bereits in Abschnitt 4.2.1 **wxWidgets** näher erläutert wurden. Über die Buttons Play (vgl. Abbildung 4.11 (2)) lässt

sich die Messung des Sensor-Arrays starten und über Stopp (3) wieder anhalten. Eine Steuerung über implementierte Buttons kann erreicht werden, indem ein erstellter Button der Klasse **wxButtons** beziehungsweise **wxBitmapButton** mit der gewünschten Funktion über die **Event Table** verknüpft wird. Wie das funktioniert, wurde wiederum in Abschnitt 4.2.1 an einem Beispiel aufgezeigt. Diese sind jedoch so lange deaktiviert, bis eine Darstellungsform in (4) ausgewählt wurde. Dafür wird in dem geerbten Thread, welcher zum Hauptfenster gehört, über die Funktion plot_charts() abgefragt, ob das zugehörige Kontrollkästchen ausgewählt wurde und wenn ja, wird die entsprechende Funktion des jeweiligen Plots ausgeführt (vgl. Listing 4.11).

Listing 4.11: Einfache If-Abfrage, um den Vektor-Plot zu realisieren.

```
1 //--> VECTOR PLOT: Showing vector of each sensor
2     if (m_CheckBox_Quiver->IsChecked()) vector_plot();
```

Die in Abschnitt 2.3.3 erklärte Offsetkompensation lässt sich im Bereich (5) über ein Kontrollkästchen einstellen. Über den linken Button wird der gemessene Offset als ein dat-File gespeichert oder es lässt sich über den rechten Button eine Datei mit Offset in das Programm laden. Durch das Markieren des Auswahlkastens - Offset wird der Offset von den aktuellen Messwerten abgezogen, wie in Abbildung 4.15 (2.3.3) erkennbar ist. Um die Ausgangswerte zu filtern, kann in Abbildung 4.11 (6) durch eine Auswahlliste einer von drei möglichen Filtern ausgewählt werden. Das Auswahlkästchen Start searching startet den Suchalgorithmus und öffnet ein neues Unterfenster. Unter (8) sind weitere Bedienmöglichkeiten zu finden, wie die Auswahl des COM-Ports, an welchem das Sensor Array angeschlossen wird (Auswahlliste und rechter Button zum Bestätigen) und die Möglichkeit, das Diagramm als Bild im PNG-Format zu speichern (linker Button).

Auslesen des Sensor-Arrays

Das Messen der Ausgangswerte des Sensor-Arrays ist eine der Hauptaufgaben der hier entwickelten Software. Wird der Thread des Hauptfensters über den Play-Button gestartet, werden die gemessenen Werte des Arrays in einer Endlosschleife ausgelesen und zunächst in einem Buffer gespeichert. Anschließend wird ermittelt, wie viele Werte in diesem abgelegt wurden. In Listing 4.12) ist das einmalige Auslesen und speichern der Sensorausgansgwerte in einen Buffer zu sehen.

Listing 4.12: Funktion zum Auslesen des Sensor-Arrays.

```
1     transmit_dat = 48;
2     my_serial->write(&transmit_dat, 1);
3     while (tr_size < 256)
4     {
5         tr_size = my_serial->available();
6     }
7     my_serial->read(buff, tr_size);
```

Im Anschluss werden die gemessenen Werte, welche sich im Buffer befinden, den zugehörigen Variablen dbl_sin beziehungsweise dbl_cos zugewiesen und in dem entsprechenden Array diff_sin beziehungsweise diff_cos gespeichert.

Listing 4.13: Funktion zum Auslesen des Sensor-Arrays.

```
int i_mval = 90;
for (i_run = 0; i_run < i_mval; i_run++)
    {
    cidx = 0;
    maxVal = 0;
    for (idx = 0; idx <= 63; idx++)
        {
            dbl_sin = (double)(buff[cidx] |
            (buff[cidx + 1] << 8));
            dbl_cos = (double)(buff[cidx + 128] |
            (buff[cidx + 129] << 8));
            if (dbl_sin > 4095) dbl_sin -= 65536;
            if (dbl_cos > 4095) dbl_cos -= 65536;

            if (i_run == 0)
            {
                diff_sin[idx] = dbl_sin;
                diff_cos[idx] = dbl_cos;
            }
            else
            {
                diff_sin[idx] = (diff_sin[idx] + dbl_sin) / 2;
                diff_cos[idx] = (diff_cos[idx] + dbl_cos) / 2;
        }
    }
}
```

Für eine Mittelwertbildung der einzelnen Sensorausgangswerte wird das Vorgehen 90 mal wiederholt und jeder neue gemessene Wert eines Sensors auf den alten aufaddiert und die Summe anschließend halbiert (vgl. Listing 4.13)). Diese Art der Mittelwertbildung wird auch als gleitender Durchschnitt bezeichnet. Schlussendlich lassen sich mit den ermittelten Werten die Länge datR und Ausrichtung datA über jedem Sensor ermitteln. Zusätzlich werden die Werte normiert, damit die Länge der Vektorpfeile im Quiver-Plot übersichtlich bleibt (vgl. Listing 4.14)).

Listing 4.14: Funktion zum Auslesen des Sensor-Arrays.

```
datR[idx] = sqrt(diff_sin[idx] * diff_sin[idx] +
      diff_cos[idx] * diff_cos[idx]);
if (maxVal < datR[idx])  maxVal = datR[idx];
datA[idx] = (atan2(diff_cos[idx], diff_sin[idx])) * (180 / PI);
```

Suchalgorithmus

Der genutzte Suchalgorithmus wurde bereits im dritten Kapitel behandelt und anhand von Beispielen erläutert.

Implementation der Signalverarbeitungsmethoden

Die im Grundlagenteil beschriebenen Signalverarbeitungsmethoden sind auch in der Klasse des Hauptfensters implementiert worden, wobei vor allem die Offsetkompensation und die Filterung der Sensorausgangswerte erwähnenswert sind.

Offsetkompensation Den Offset des Sensor-Arrays zu entfernen, gestaltet sich als relativ einfach, da dafür einfach die Ausgangswerte des Sensors zu Anfang jeder Messung einmalig in einem anderen Array gespeichert und in nachfolgenden Messungen abgezogen werden. Es bietet sich an, das Vorgehen mehrfach zu wiederholen und den Mittelwert der Messungen zu bilden, um starken Ausreißern in nur einer anfänglichen Messung entgegenzuwirken. Also werden in dieser Arbeit zu Beginn 50 mal die Werte des Arrays ausgelesen, der Mittelwert gebildet und anschließend im Array für den Offset datR_-Offset gespeichert. Der implementierte Code des erklärten Vorgehens ist im Listing 4.15 dargestellt.

Listing 4.15: Funktion zur Bestimmung des Offsets.

```
void MyFrame::Offset()
{    int idx = 0;
     for (int i = 0;i < 50; i++) {
          read_array();
          for (idx = 0; idx <= 63; idx++)
          // offset of cosine is offset array index 0 to 63
          {
               datR_Offset[idx] += diff_cos[idx];
          }
          for (idx = 64; idx <= 127; idx++)
          // offset of sine is offset array index 64 to 127
          {
               datR_Offset[idx] += diff_sin[idx - 64];
          }
     }
     // mean value for each entry
     for (idx = 0; idx <= 127; idx++)
     {
          datR_Offset[idx] = datR_Offset[idx] / 50;
     }
}
```

Wird nun das Kontrollhäkchen zur Kompensation gesetzt, wird in den nachfolgenden Messungen der entsprechende Offsetwert subtrahiert (vgl. Listing 4.16).

Listing 4.16: If-Abfrage, ob der Offset von den Messdaten subtrahiert werden soll.

```
if (m_CheckBox_Offset->IsChecked()) substract_Offset();
```

Digitale Filterung Um unerwünschtes Rauschen aus den Messdaten zu filtern, müssen diese zunächst im zweidimensionalen Fourierraum betrachtet werden. Das Vorgehen, um dies umzusetzen, wurde im Grundlagenteil beschrieben und ist softwareseitig mit entsprechenden Funktionen umgesetzt worden (vgl. Abbildung 3.9 in Abschnitt 3.3.1). Die zugehörige Umsetzung in C++ ist in Listing 4.17 abgebildet.

Listing 4.17: Implementation der Filter-Funktion.

```
1   // 1D-DFT X*
2   ONE_D_DFT(A_real, A_imag, X_real, X_imag, 8);
3   Transpose(X_real, ROWS8, COLUMNS8);
4   Transpose(X_imag, ROWS8, COLUMNS8);
5   // 2D-DFT X*
6   ONE_D_DFT(X_real, X_imag, F_real, F_imag, 8);
7   Transpose(F_real, ROWS8, COLUMNS8);
8   Transpose(F_imag, ROWS8, COLUMNS8);
9   // shift zero-frequency component to the center of the array
10  fftshift(F_real, F_imag, ROWS8, COLUMNS8);
11  // shift zero-frequency component back to original position
12  ifftshift(F_real, F_imag, ROWS8, COLUMNS8);
13  Transpose(F_real, ROWS8, COLUMNS8);
14  Transpose(F_imag, ROWS8, COLUMNS8);
15  // 1D-IDFT
16  ONE_D_IDFT(F_real, F_imag, X_real, X_imag, 8);
17  Transpose(X_real, ROWS8, COLUMNS8);
18  Transpose(X_imag, ROWS8, COLUMNS8);
19  // 2D-IDFT
20  ONE_D_IDFT(X_real, X_imag, F_real, F_imag, 8);
```

In den Funktionen ONE_D_DFT beziehungsweise ONE_D_IDFT werden die im Array gespeicherten Werte mit der Twiddlefaktor-Matrix multipliziert, welche sich über zwei verschachtelte for-Schleifen für eine Matrixgröße **N** unkompliziert berechnen lässt (vgl. Listing 4.18).

Listing 4.18: Berechnung der Twiddlefaktor-Matrix.

```
for (int k = 0; k < N; k++)
    {
        for (int n = 0; n < N; n++)
        {
            W_real[k][n] += cos((2 * PI * k * n) / N);
            W_imag[k][n] += -sin((2 * PI * k * n) / N);
        }
    }
```

Auch das Transponieren ist einfach umsetzbar, indem die Indizes der Arrays über zwei weitere for-Schleifen vertauscht werden. Die Verschiebung der niederfrequenten Anteile in das Zentrum des Arrays (hier: fftshift()) gestaltet sich als etwas schwieriger. Zunächst muss ermittelt werden, wo genau sich die Nullfrequenz in der Matrix befindet und die Indizes gespeichert werden. Dies geschieht über das Nutzen des in Listing 4.19 dargestellten Programmcode.

Listing 4.19: Ausschnitt aus der fftshift()-Funktion.

```
    int index_horz = 0, index_vert = 0;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (F_imag[i][j] == 0) {
                index_vert = i;
                index_horz = j;
            }
        }
    }
```

Durch die ermittelten Indizes können die entsprechenden Zeilen und Spalten vertauscht werden, um die Nullfrequenz im Zentrum der Matrix zu platzieren. Durch eine elementweise Multiplikation der Matrix mit den transformierten und verschobenen Sensorausgangswerten mit einer einfachen Filtermaske, lässt sich unerwünschtes hochfrequentes

Rauschen entfernen. Da der niederfrequente Anteil sich nun im Mittelpunkt befindet, kann die Filtermaske überall auf Null gesetzt werden, bis auf das Zentrum der Filtermatrix, wo die relevanten Messdaten sitzen (vgl. Listing 4.20).

Listing 4.20: Filterkoeffizienten-Matrix.

```
1  double filter_real[8][8] =
2      {      //      0      1      2      3      4      5      6      7
3          /* 0 */ {  0,     0,     0,     0,     0,     0,     0,     0},
4          /* 1 */ {  0,     0,     0,     0,     0,     0,     0,     0},
5          /* 2 */ {  0,     0,     0,     0,     0,     0,     0,     0},
6          /* 3 */ {  0,     0,     0,     1,     1,     1,     0,     0},
7          /* 4 */ {  0,     0,     0,     1,     0.2,   1,     0,     0},
8          /* 5 */ {  0,     0,     0,     1,     1,     1,     0,     0},
9          /* 6 */ {  0,     0,     0,     0,     0,     0,     0,     0},
10         /* 7 */ {  0,     0,     0,     0,     0,     0,     0,     0},
11     };
```

Im Anschluss werden die so erhaltenen Werte über iffstshift() wieder an ihre ursprüngliche Stelle verschoben und danach in den Zeitbereich zurücktransformiert.

Speicherung von Daten

Im Hauptfenster ist es möglich, einen aufgenommenen Offset als dat-Datei auf dem Computer zu speichern, damit dieser für weitere Messungen genutzt werden kann, ohne dass nach jedem erneuten Starten der Applikation dieser wieder ermittelt werden muss. Wie der Offset gemessen und intern gespeichert wird, wurde bereits zuvor erläutert. Wird die Funktion onSaveOffset() (vgl. Listing 4.21) über den zugehörigen Speicherbutton gestartet, wird der Offset zunächst bestimmt und es öffnet sich anschließend ein Dialogfenster, welches danach fragt, wo die Datei gespeichert werden soll. Dafür werden die im Offset-Array gespeicherten Daten zum Typ vector umgewandelt und anschließend über write_vector_to_file() in einer dat-Datei gespeichert.

Listing 4.21: Funktion zum Speichern der Offset-Daten.

```
1  wxFileDialog saveFileDialog(this, _("Save Offset"),
2                 "", "", "Offset files (*.dat)|*.dat",
3                 wxFD_SAVE | wxFD_OVERWRITE_PROMPT);
4      Offset();
5      if (saveFileDialog.ShowModal() == wxID_OK) {
6
7          std::vector<double> myVector(127);
8
9          for (int idx = 0; idx < 127; idx++)
10         {
11             myVector.at(idx) = datR_Offset[idx];
12         }
13         write_vector_to_file(myVector,
14                     (std::string)saveFileDialog.GetPath());
15     }
```

Des Weiteren können die erstellten Grafiken des Hauptfensters als PNG-Datei gespeichert werden. Um dies umzusetzen, stellt Chartdirector eine bereits fertige Funktion SaveFile() (vgl. Listing 4.22 Zeile 6) zur Verfügung. Wird auf den entsprechenden Button zur Speicherung des Bildes geklickt, öffnet sich wieder ein Dialogfenster, um den Speicherort auszuwählen.

Listing 4.22: Funktion zum Speichern der erstellten Grafiken.

```
1  wxFileDialog saveFileDialog(this, _("Save Image as PNG"),
2                 "", "", "PNG files (*.PNG)|*.PNG",
3                 wxFD_SAVE | wxFD_OVERWRITE_PROMPT);
4
5      if (saveFileDialog.ShowModal() == wxID_OK) {
6          temp_bmp.SaveFile(saveFileDialog.GetPath(),
7          wxBITMAP_TYPE_PNG);
8      }
```

4.4.3 Sub-Fenster - SubFrame

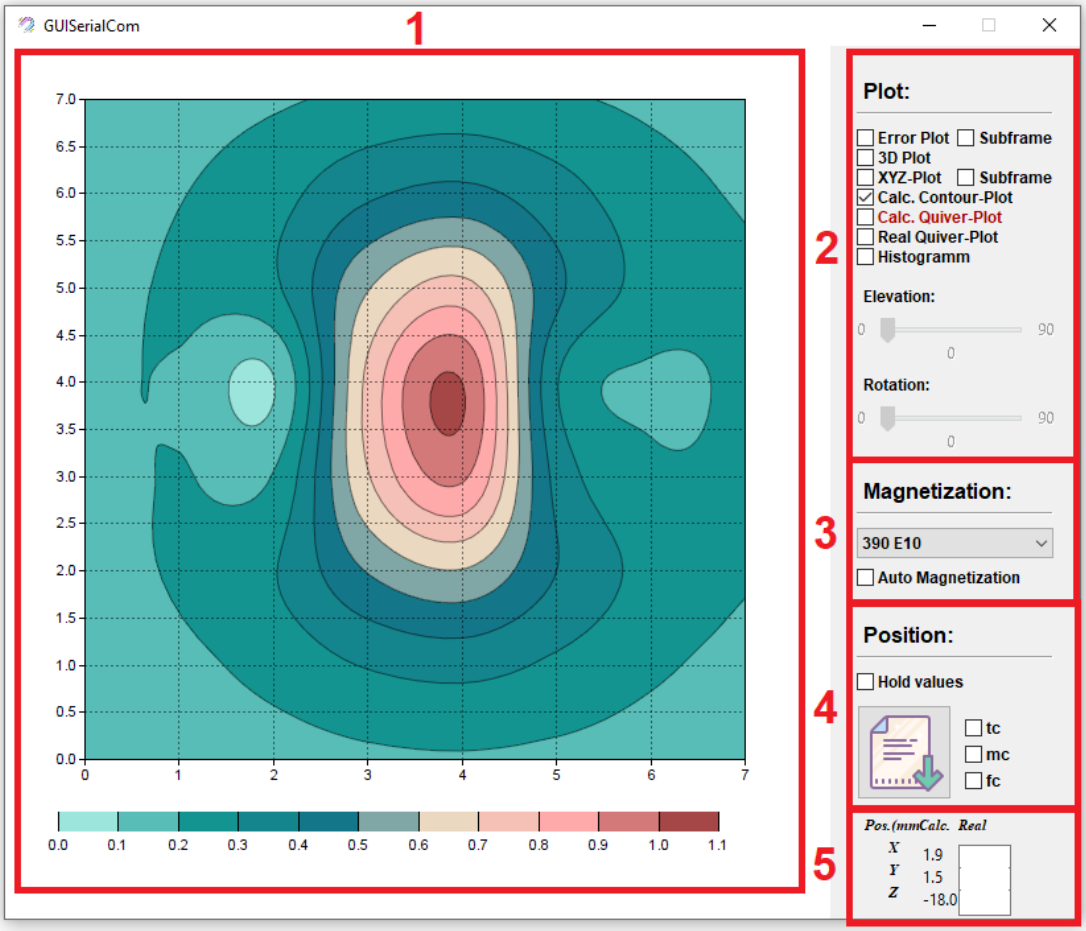


Abbildung 4.13: Unterfenster der Anwendung.

Das Unterfenster, welches sich öffnet, wenn die Suche gestartet wird, ist ähnlich strukturiert wie das Hauptfenster. Es unterteilt sich wieder in Bild (vgl. Abbildung 4.13 (1)) - und Control-Panel (2-5), jedoch unterscheidet es sich in den Möglichkeiten der Bedienung. Vor allem stehen mehr Arten der Darstellung zur Verfügung, um die ermittelte Position des Magneten so genau wie möglich untersuchen zu können.

Grafische Darstellung

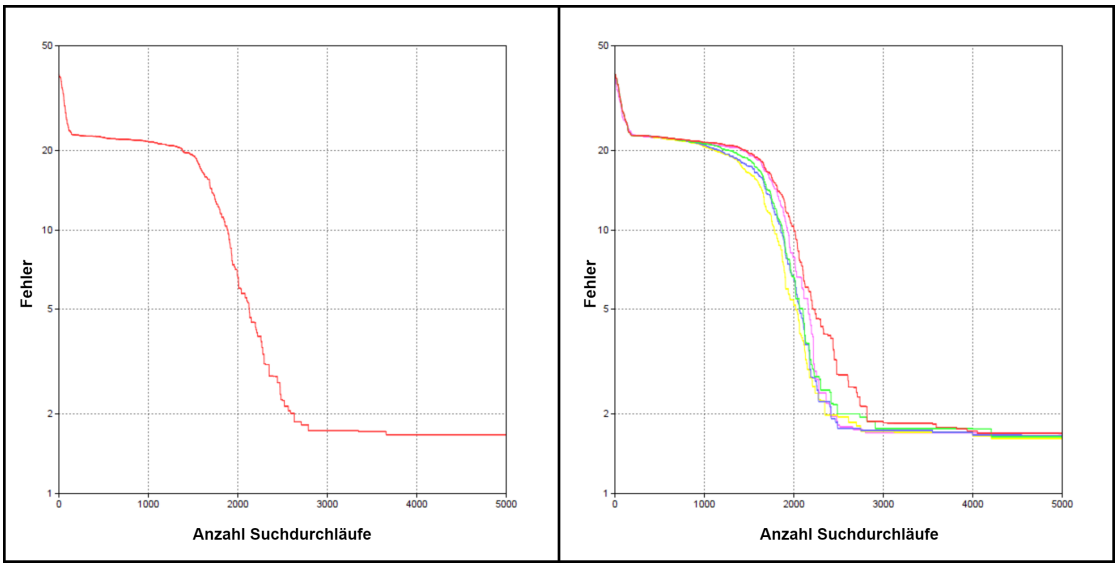


Abbildung 4.14: Lernkurve des Suchalgorithmus.

Error-Plot Über die Auswahl vom „Error-Plot“ lässt sich die Abweichung des hypothetischen vom realen Sensor-Array für jeden Suchschritt eines Durchlaufes beobachten (vgl. Abbildung 4.14). Sie stellt die Lernkurve des Algorithmus für die Bestimmung der Position des Magneten dar. Da der Fehler vor Beginn der Suche als ausreichend hoch angenommen wird, fällt dieser erst abrupt bevor sich ein eher gleichmäßiger Verlauf einstellt. Aus Darstellungsgründen ist die Y-Achse logarithmisch und auf ihr lässt sich die Abweichung in jedem Suchdurchlauf, welche auf der linearen X-Achse abzulesen ist, bestimmen. Durch zusätzliches Anwählen von Subframe rechts neben Error-Plot, kann das Diagramm in einem extra Fenster (vgl. Abschnitt 4.4.4 **Widgets**) geöffnet werden. Wird das Kontrollkästchen Hold Values ausgewählt (vgl. Abbildung 4.13 (4)), werden die fünf Lernkurven der letzten Suchdurchläufe innerhalb eines Plots dargestellt (vgl. Abbildung 4.14 rechts).

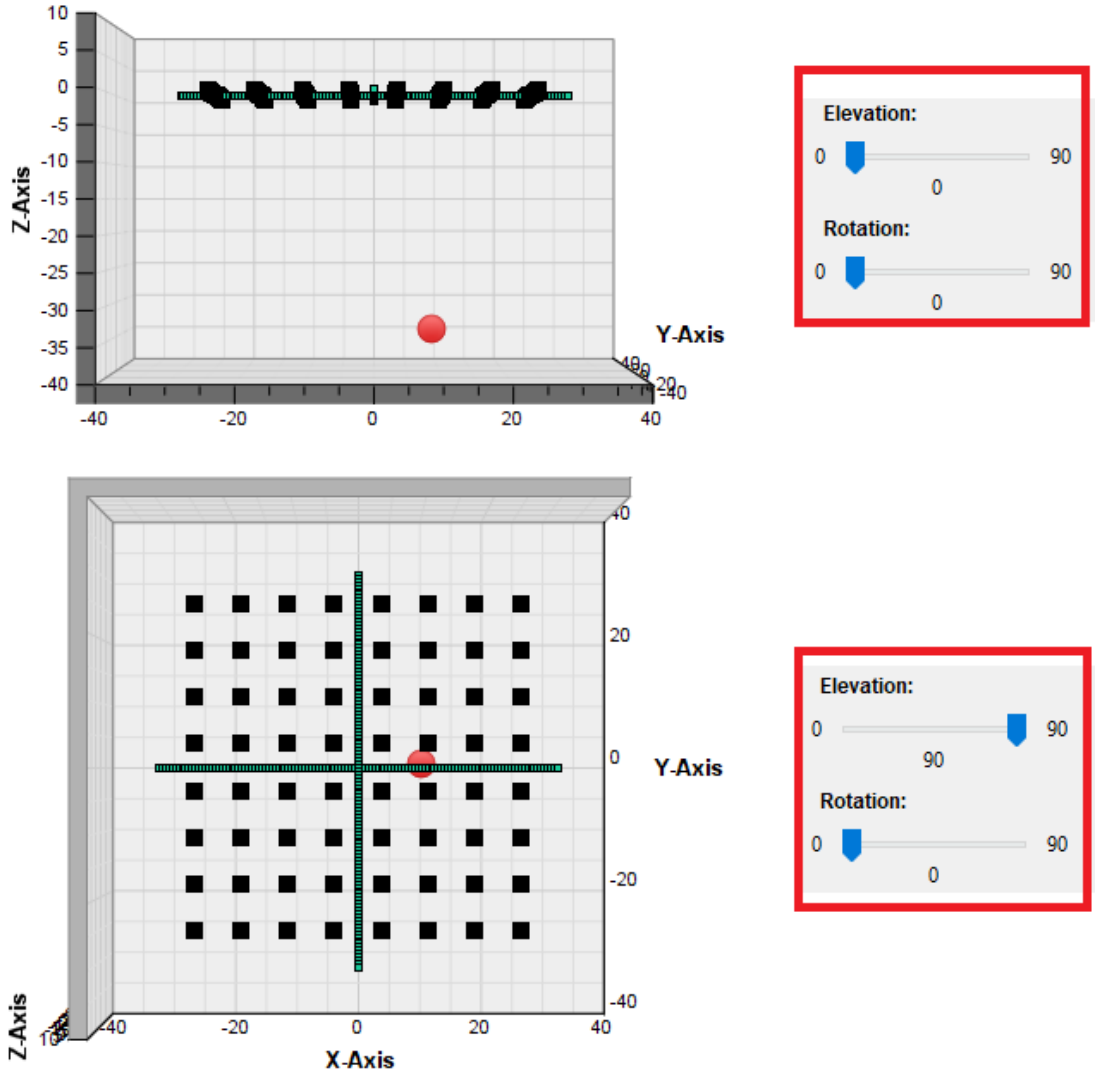


Abbildung 4.15: 3D-Darstellung der errechneten Magnetposition.

3D-Plot Um sich ein genaueres Bild von der Lage des Magneten machen zu können, bietet die Option „3D-Plot“ (vgl. Abbildung 4.15) die Möglichkeit, dessen Position räumlich darzustellen. Mit Hilfe der Schieberegler „Elevation“ und „Rotation“ können Höhe beziehungsweise Drehung der Ansicht nach Wunsch angepasst werden. Die 8x8 schwarzen Quadrate sollen hierbei das Sensor-Array nachmodellieren und das Kreuz im Ursprung soll bei der Orientierung helfen. Der rote Punkt stellt die aktuell berechnete Position des Permanentmagneten dar.

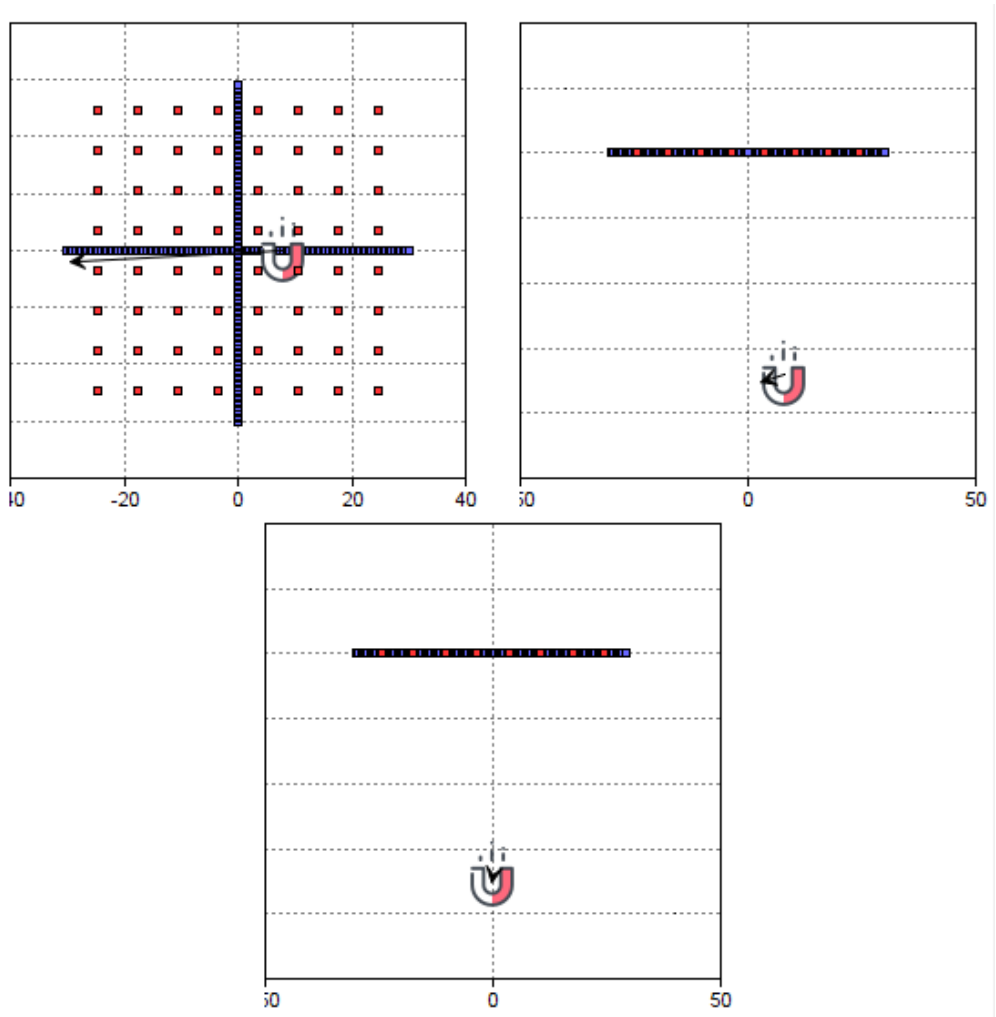


Abbildung 4.16: XY-, XZ- und YZ-Darstellung der errechneten Magnetposition.

XYZ-Plot Durch Markieren des „XYZ-Plots“ (vgl. Abbildung 4.16) lässt sich eine etwas aussagekräftigere Methode der Darstellung auswählen. Durch diese kann der Magnet gleichzeitig in jeder der drei Ebenen (XY, XZ, YZ) angezeigt werden und zusätzlich ist die berechnete Magnetisierungsrichtung durch einen an den Magneten gezeichneten Vektorpfeil erkennbar. Wie auch bereits im 3D-Diagramm sind Sensor-Array und Koordinatenkreuz eingezeichnet, um eine bessere Orientierung für den Nutzer zu schaffen. Durch Selektieren von Subframe neben dem eigentlichen Auswahlkästchen, kann auch der XYZ-Plot in einem extra Fenster (vgl. Abschnitt 4.4.4 **Widgets**) geöffnet werden.

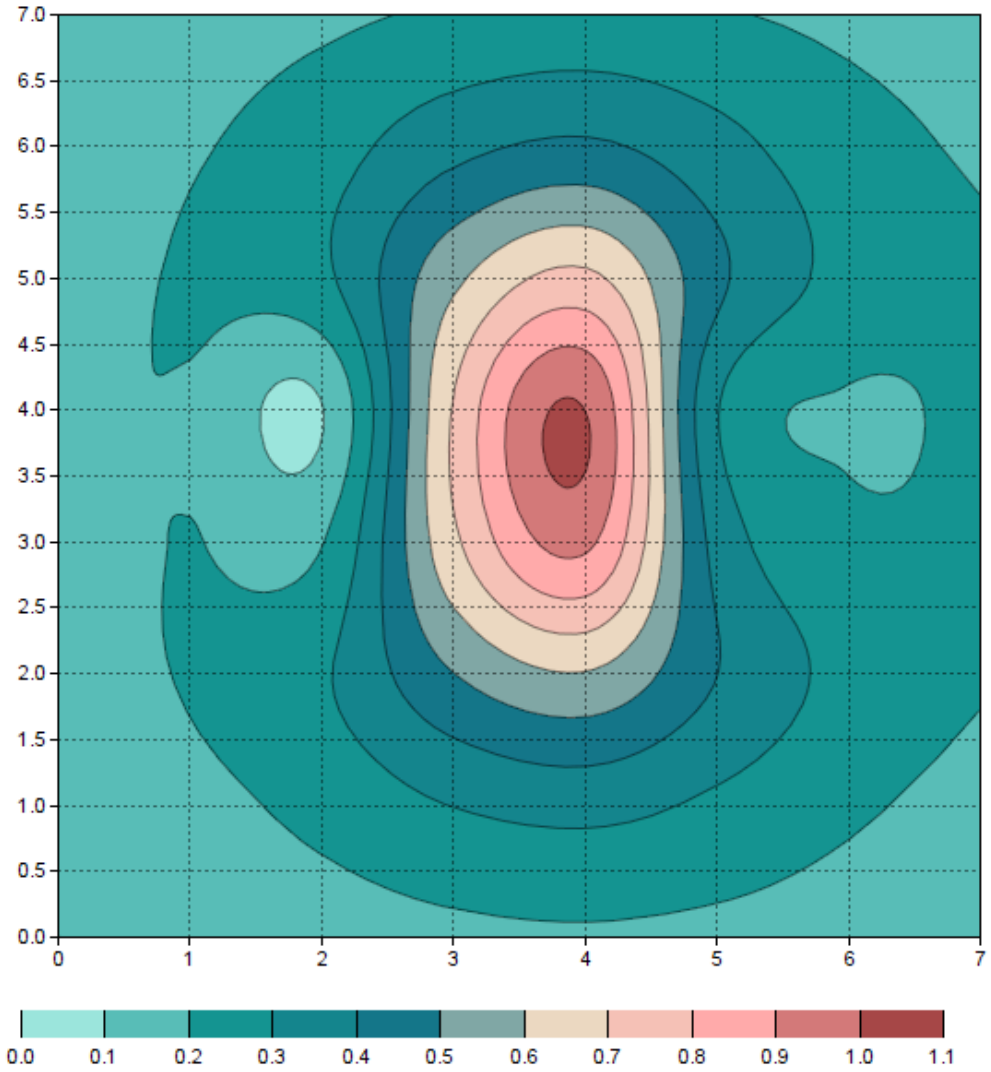


Abbildung 4.17: Berechnete Kontur-Darstellung des hypothetischen Sensor-Arrays.

Calculated Contour-Plot Für die Betrachtung des Schätzwertemodells in einem Kontur-Plot kann die Option „Calc. Contour-Plot“ genutzt werden (vgl. Abbildung 4.17). Es stellt die berechneten Werte des hypothetischen Arrays in derselben Form dar, wie die real gemessenen im Hauptfenster (vgl. Abschnitt 4.3.1) und ermöglicht einen besseren Vergleich zwischen tatsächlichen und berechneten Werten für den Betrachter.

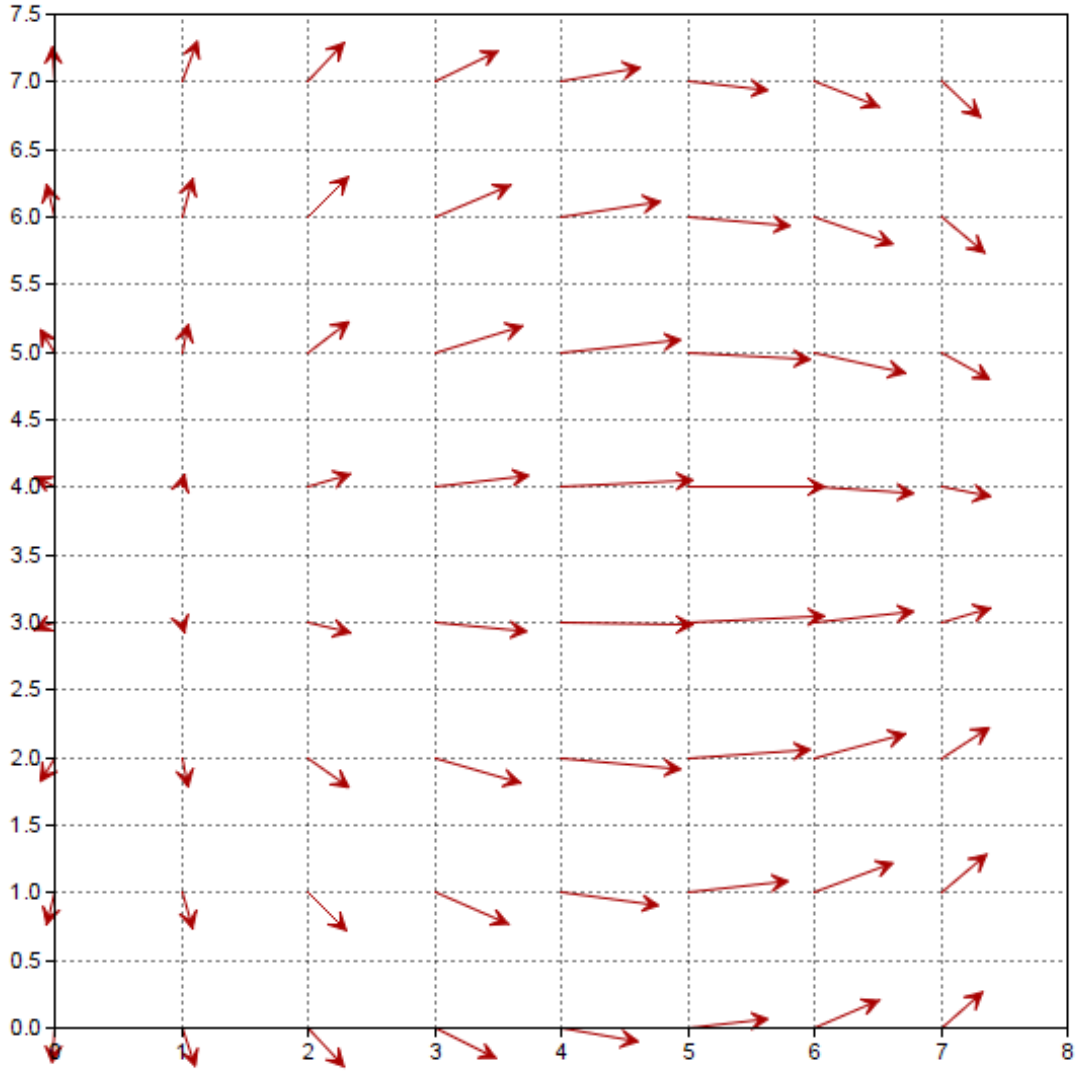


Abbildung 4.18: Vektordiagramm-Darstellung des hypothetischen Sensor-Arrays.

Calculated Quiver-Plot Das Selektieren von „Calculated Quiver Plot“ (vgl. Abbildung 4.18) stellt die Ergebnisse des Suchalgorithmus in einem Vektorplot dar, welcher bereits in Abschnitt 4.3.1 vorgestellt wurde.

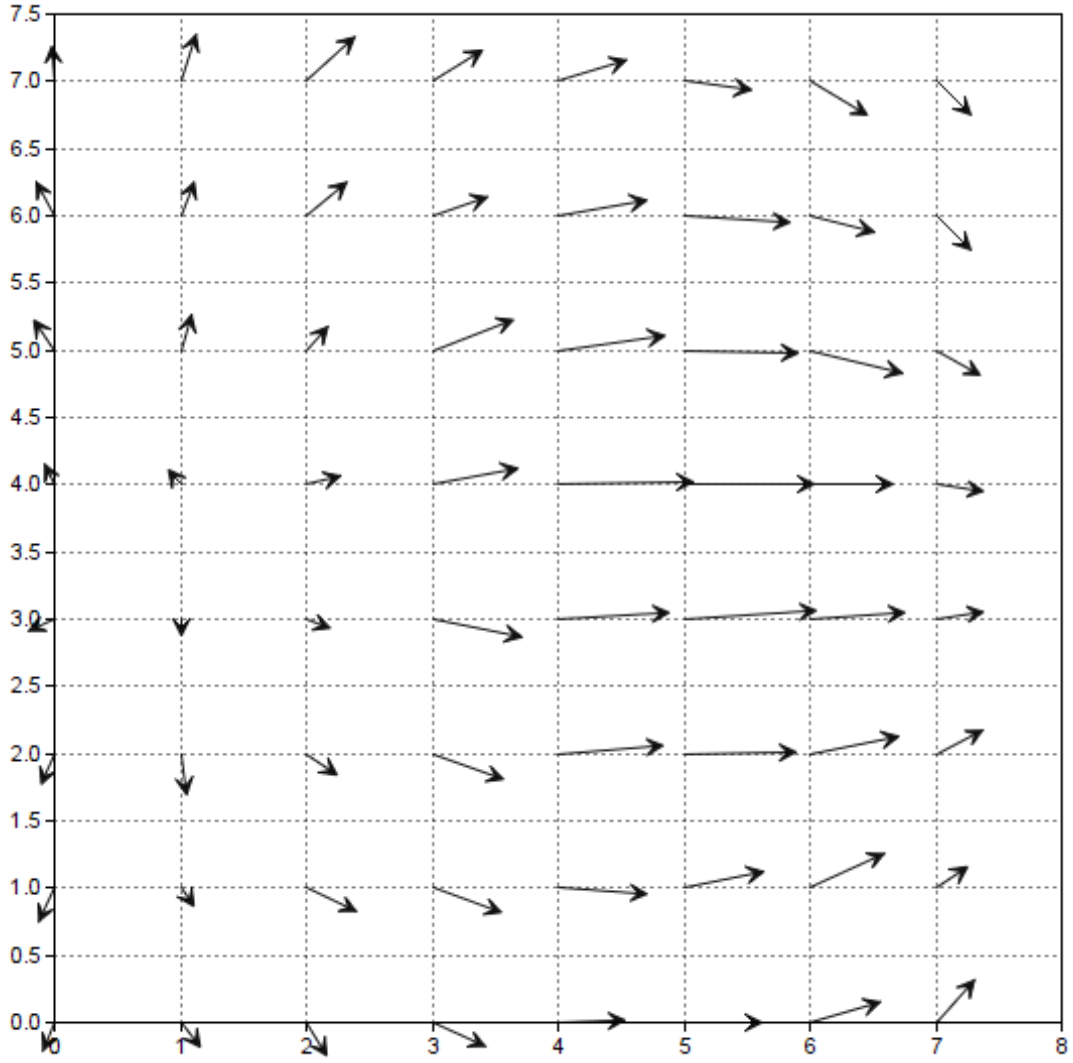


Abbildung 4.19: Vektordiagramm der Sensorausgangswerte des Sensor-Arrays.

Real Quiver-Plot Auch beim Anwählen des „Real Quiver-Plot“ wird wieder ein Vektordiagramm, jedoch mit den real gemessenen Werten, im Bild-Panel des Unterfensters angezeigt (vgl. Abbildung 4.19). Durch gleichzeitiges Auswählen von berechneten und realen Sensorausgangswerten als Vektordiagramm, ist ein einfacher Vergleich zwischen Schätzwertemodell und Sensor-Array möglich.

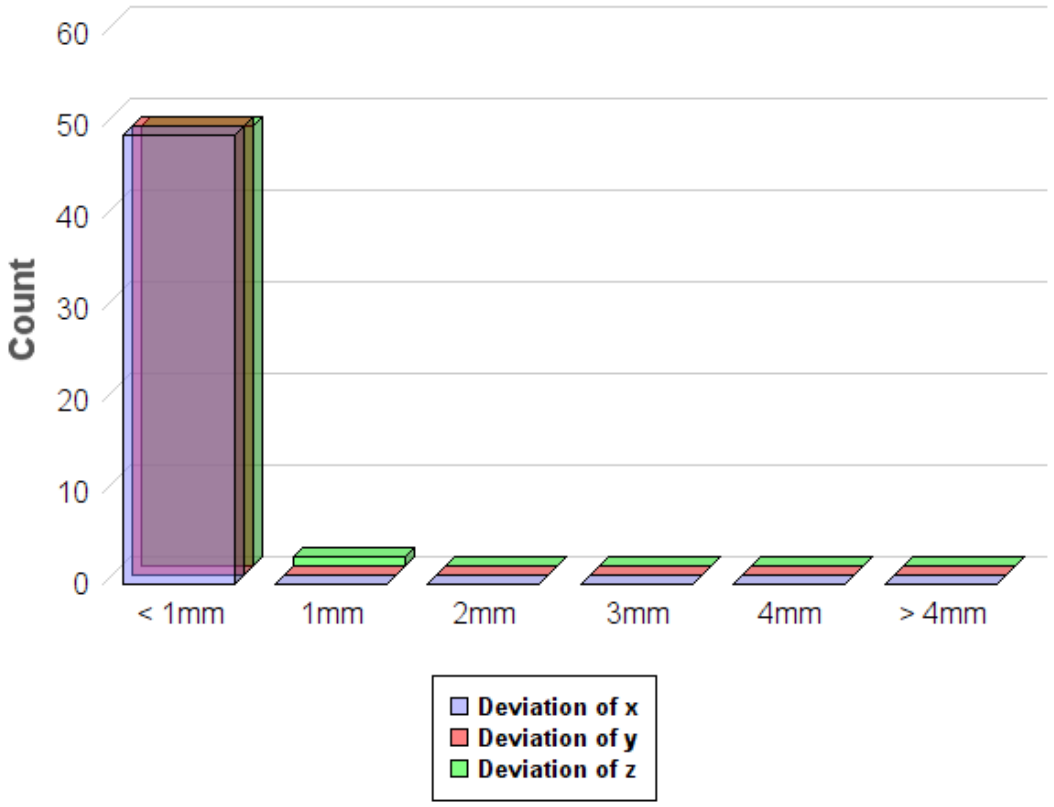


Abbildung 4.20: Histogramm-Darstellung der Abweichung zwischen realer und berechneter Magnetposition.

Histogramm Mit Hilfe der Histogramm-Darstellung (vgl. Abbildung 4.20) lässt sich einfach nachvollziehen, wie groß die Abweichung zwischen berechneter und realer Position des Bezugsmagneten ist. Dafür müssen unten rechts im Control-Panel (vgl. 4.13 (5)) die echten Koordinaten händisch eingetragen werden. Anschließend wird die Abweichung von der Anwendung ermittelt und in Form eines Balkendiagramms im Bild-Panel dargestellt.

Steuerung

Die Steuerung des Sub-Fensters funktioniert genauso wie die des Hauptfensters. Sie unterscheidet sich jedoch bei der Anzahl der möglichen Einstellungen, die vorgenommen werden können. Hier stehen mehr Plotmöglichkeiten zur Verfügung, um sich die errechnete Magnetposition darstellen zu lassen und einen räumlichen Überblick der Lage zu verschaffen. Dabei wird wieder über die Kontrollkästchen die gewünschte Darstellungsart ausgewählt, wobei die mit **Subframe** betitelten Checkboxen neue Fenster öffnen, welche zu der Klasse **ImageFrame** gehören. Es ist möglich, über den Button (vgl. Abbildung 4.13 (4)) die berechneten Daten des Algorithmus zu speichern, wobei die Textfelder unten links (5) die Möglichkeit bieten, die reale Position des Magneten einzutragen, damit beispielsweise die Abweichungen zwischen tatsächlicher und vom Algorithmus ermittelter Position bestimmt werden können.

Speicherung von Daten

Auch im Unterfenster der Anwendung ist es möglich, Messergebnisse auf dem Computer zu speichern. Es lassen sich Suchdurchläufe sowie deren wichtigsten Kenngrößen als Textdatei anschaulich in einer Tabelle ausgeben. Dafür stehen rechts neben dem Speicher-Button drei Auswahlkästchen zur Verfügung, mit welchen ausgewählt werden kann, welche Daten gespeichert werden sollen (vgl. Abbildung 4.13 (4)). Ein Beispiel für eine solche Tabelle ist in Abbildung 4.21 zu erkennen.

```
------------------------------
Date and Time: Tue Aug 17 14:24:17 2021
+--------------+--------------+--------------+-------------------+-------------------+-------------------+------------+
|Meanvalue pos x|Meanvalue pos y|Meanvalue pos z|Meanvalue mag x     |Meanvalue mag y     |Meanvalue mag z    |Meanvalue fc|
+--------------+--------------+--------------+-------------------+-------------------+-------------------+------------+
|13.724460      |-3.381114     |     -92.914673|2999743364037.529297|1108981952569.446777|860437852365.662964|0.960227    |
+--------------+--------------+--------------+-------------------+-------------------+-------------------+------------+
```

Abbildung 4.21: Ausgabe des Mittelwertes der ermittelten Position bei 100 Suchdurchläufen als Textdatei.

4.4.4 Widgets - ImageFrame

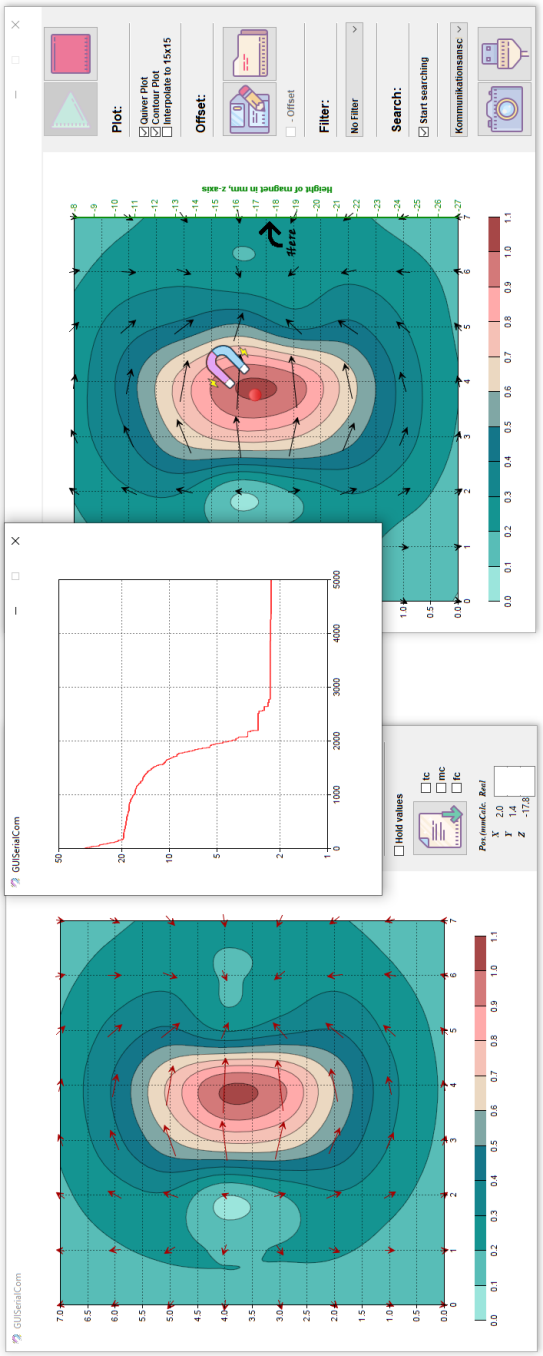


Abbildung 4.22: Haupt- und Unterfenster sowie Widget „Error-Plot“ in gleichzeitiger Benutzung.

Grafische Darstellung

Wie bereits unter „Error-Plot“ und „XYZ-Plot“ erwähnt wurde, lassen sich diese beiden Plot-Darstellungen in einem Extra-Fenster öffnen, was auch als Widget bezeichnet wird. Diese Optionen stehen zur Verfügung, um die Lernkurve vom Algorithmus sowie die aktuelle Lage und Magnetisierungsrichtung des Permanentmagneten besser verfolgen zu können. So könnten beispielsweise ein Kontur-Plot mit den realen Werten im Hauptfenster, mit den berechneten Werten im Unterfenster und Lernkurve in einem extra Fenster gleichzeitig auf dem Bildschirm angezeigt werden (vgl. Abbildung 4.22). Damit lassen sich zum Beispiel der Unterschied zwischen Schätzwertmodell und Sensorausgangswerten und der bleibende Fehler simultan untersuchen.

4.4.5 Hilfsthread - SetImgThread

Während der Implementation der Software ist wiederholt der Fehler aufgetreten, dass wenn der Thread über den Stopp-Button angehalten wurde, die gesamte Applikation eingefroren ist. Die Ursache konnte darauf zurückgeführt werden, dass ein Stoppen des Threads, während die erstellte Grafik der PNG-Datei angehängt wurde, einen Fehler verursachte und das Programm zum Abstürzen brachte. Durch das Einführen eines neuen Threads, dessen Aufgabe nur das Anhängen an die Bild-Datei beinhaltet, kann der Fehler im Vorfeld verhindert werden. Das liegt daran, dass der Hilfsthread noch weiter im Hintergrund läuft, selbst wenn der Thread des Hauptfensters beendet wurde. Dafür werden dem Thread als Parameter die erstellte Grafik und die Bild-Datei, an welche diese angehängt werden soll, übergeben. Somit wird für jeden Durchlauf der Endlosschleife in `Entry()` und somit für jede neu erzeugte Grafik ein neuer Thread erzeugt, welcher die Aufgabe übernimmt, diese der PNG-Datei des entsprechendes Fensters anzuhängen. Der dazu implementierte Code ist im Listing 4.23 abgebildet.

Listing 4.23: Entry()-Funktion des **SetImgThreads** für das Anhängen einer Grafik an die Bild-Datei.

```
1 void* SetImgThread::Entry()
2 {
3     if (image != nullptr && !(TestDestroy())) {
4         image->SetBitmap(*bitmap);
5     }
6     delete(bitmap);
7     return nullptr;
8 }
```

5 Evaluation

5.1 Exemplarischer Test der GUI

Um die Funktionalität des Algorithmus zu Testen, wird ein Magnet an unterschiedlichen Orten über dem Array positioniert und mit Hilfe der entwickelten Software beziehungsweise mit dem implementierten Suchalgorithmus sollen die Koordinaten von diesem ermittelt werden. Aus Gründen der Symmetrie ist es ausreichend, wenn nur ein Quadrant des Arrays getestet wird, da die Ergebnisse bei den anderen redundant wären. Des Weiteren soll überprüft werden, wie sich Fehllagen des Magneten, zum Beispiel eine Positionierung außerhalb der Reichweite des Arrays (vgl. Abbildung 5.1 e) und f)) oder Schräglagen (vgl. Abbildung 5.23), auf das Ergebnis auswirken. Dafür wird vor jedem Test die Position des Permanentmagneten händisch ermittelt und im Anschluss mit dem errechneten Ergebnis verglichen. Es ist anzumerken, dass keine exakte Positionierung des Magneten über dem Array möglich sein wird, da der Messaufbau nicht ideal für solche Zwecke ist. Bei diesem handelt es sich um eine einfache Vorrichtung, mit Hilfe welcher sich verschiedene Position einstellen lassen, jedoch nur mit bestimmter Genauigkeit. Aufgrund dessen wird das Ergebnis von Beginn an Ungenauigkeiten aufweisen, welche durch das Ermitteln und Einstellen der Position per Hand auftreten. Für die Messungen werden fünf Neodym-Ringmagnete genutzt, welche einen Außendurchmesser von 12 mm und einen Innendurchmesser von 9 mm besitzen. Jeder von diesen weist eine Güte von N45 und eine Haftkraft von 1.5 kg auf. Zur Auswertung der Messergebnisse stehen verschiedene Plotmöglichkeiten und eine Ausgabe der notwendigen Daten als Textdatei zur Verfügung.

5.1.1 Genauigkeit des Suchalgorithmus bei unterschiedlicher Positionierung des Magneten

Als erstes soll untersucht werden, wie das Ergebnis des Algorithmus beeinflusst wird, wenn sich der Permanentmagnet an verschiedenen Punkten über dem Array befindet (vgl. Abbildung 5.1). Als Startwert der Suche sind die Koordinaten (0, 0, -40) vorgegeben und es ist zu erwarten, dass je weiter der Gebermagnet von diesem Wert entfernt wird, desto ungenauer wird das am Ende der Suche ermittelte Ergebnis. In diesem Abschnitt befindet sich der Magnet immer in einer Höhe von -45 mm, welche dem nicht idealen Messaufbau zu verschulden sind, da sich -40 mm nicht genau einstellen lassen. Da die Höhe allerdings nicht weit von der anfänglich eingestellten entfernt ist, sollte dies das Ergebnis nicht allzu sehr beeinflussen.

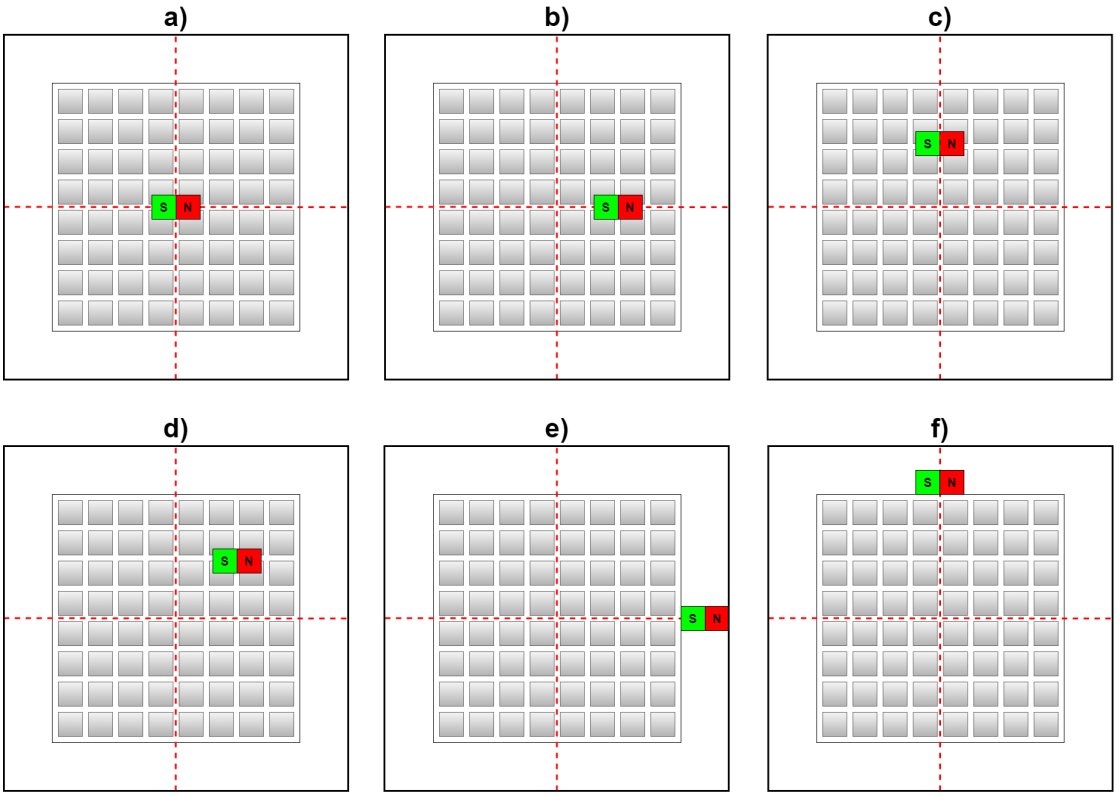


Abbildung 5.1: Verschiedene Magnetpositionen zum Testen des Algorithmus.

Position (0, 0, -45)

Im ersten Schritt wird überprüft, wie genau das Resultat der Suche ist, wenn sich der Magnet in der Nähe der im Algorithmus voreingestellten Startkoordinaten (0, 0, -40) (vgl. Abbildung 5.1 a)) befindet.

Ein Vergleich zwischen dem realen und dem berechneten Vektorplot (vgl. Abbildung 5.2 (links)) verschafft einen ersten Eindruck, wie gut das ermittelte Schätzwertemodell mit den realen Werten übereinstimmt. Es ist zu erkennen, dass die über den Algorithmus ermittelten Magnetisierungsrichtungen an den Sensoren gut mit den echten übereinstimmen, jedoch die Länge der einzelnen Vektorpfeile Unterschiede aufweisen. Über das mit der Software erstellte Histogramm wird deutlich, wie stark die reale von der berechneten Position des Magneten abweicht. Dafür werden die Koordinaten des Magneten in 100 Suchdurchläufen berechnet und mit den tatsächlichen verglichen. Im XYZ-Plot lässt sich der berechnete Ort vom verwendeten Magneten auch noch einmal in Ebenendarstellung betrachten und verschafft zusätzlich einen besseren räumlichen Überblick (vgl. Abbildung 5.2 (rechts)).

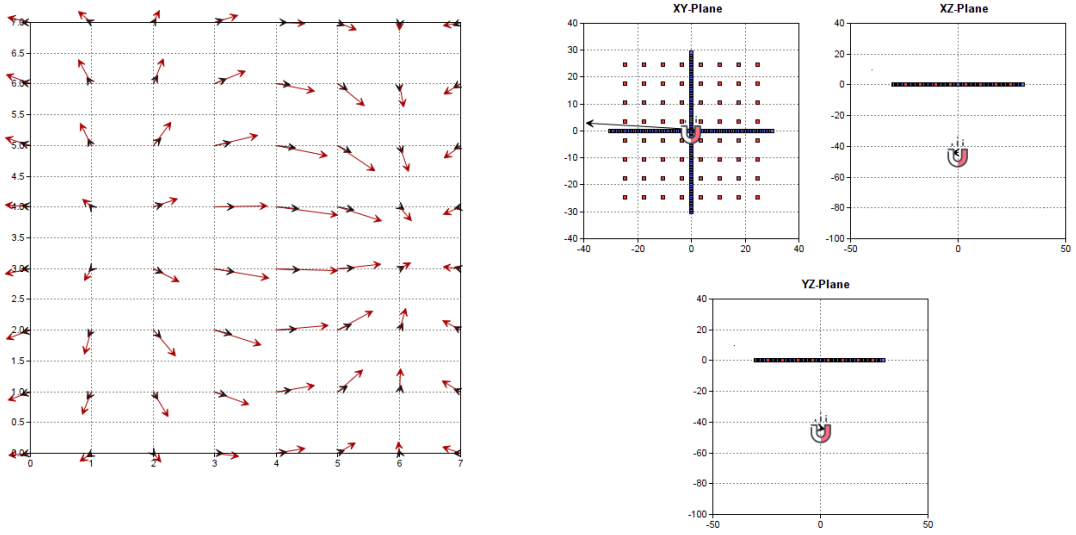


Abbildung 5.2: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position a).

Mit Hilfe der Darstellungsmöglichkeit des Histogrammes lässt sich beurteilen, wie gut reale und errechnete Position übereinstimmen. Die Ergebnisse sind in Tabelle 5.1 fest-

gehalten. Auffällig ist, dass in x-Richtung kaum größere Abweichungen als 1 mm zu verzeichnen sind und in y-Richtung auch noch gute Ergebnisse erzielt wurden, aber ein wenig schlechter als bei x. Dem zu Grunde liegen könnten die bereits angedeuteten, nicht vermeidbaren Ungenauigkeiten wie zum Beispiel der nicht ideale Messaufbau. Trotzdem besitzen nur rund 25% eine größere Abweichung als 1 mm, was insgesamt als gutes Resultat eingestuft werden kann.

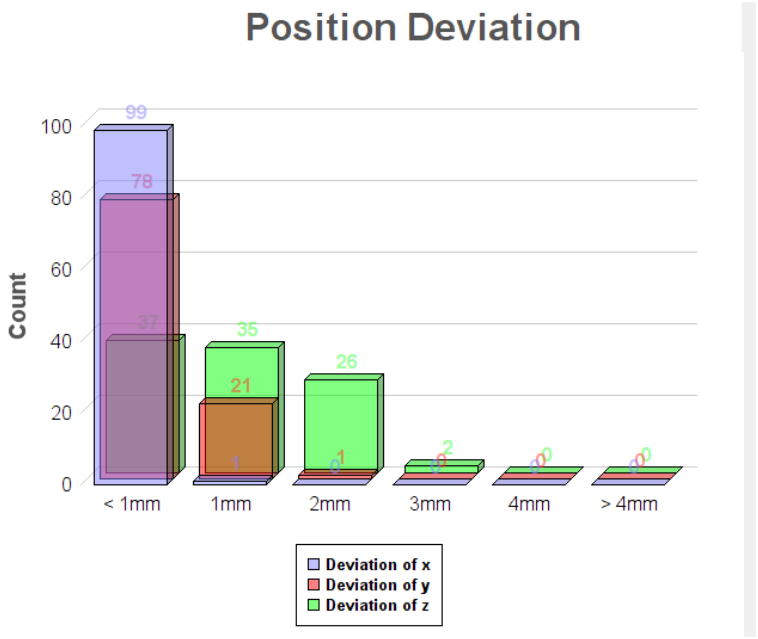


Abbildung 5.3: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position a).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	99	1	0	0	0	0
y	78	21	1	0	0	0
z	37	35	26	2	0	0

Tabelle 5.1: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position a).

In Tabelle 5.2 sind noch einmal relevante Kenngrößen festgehalten, welche sich nach den 100 Suchdurchläufen aufstellen lassen. Es ist zu erkennen, dass im Mittelwert die

tatsächlichen und die vom Algortihmus ermittelten Positionen gut übereinstimmen und die Standardabweichung der Messwerte insgesamt nicht größer als 0.5 mm sind.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.148	0.154	0.392	
y	0.691	0.184	0.429	0.453
z	-43.705	0.429	0.538	

Tabelle 5.2: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position a).

Position (14, 0, -45)

Im nächsten Schritt wird die Magnetposition um 14 mm in x-Richtung verschoben. Es lässt sich die Vermutung aufstellen, dass die Ungenauigkeiten bei der Positionserfassung zunehmen werden, da sich der Gebermagnet weiter weg von der Startposition befindet als beispielsweise noch im Abschnitt zuvor.

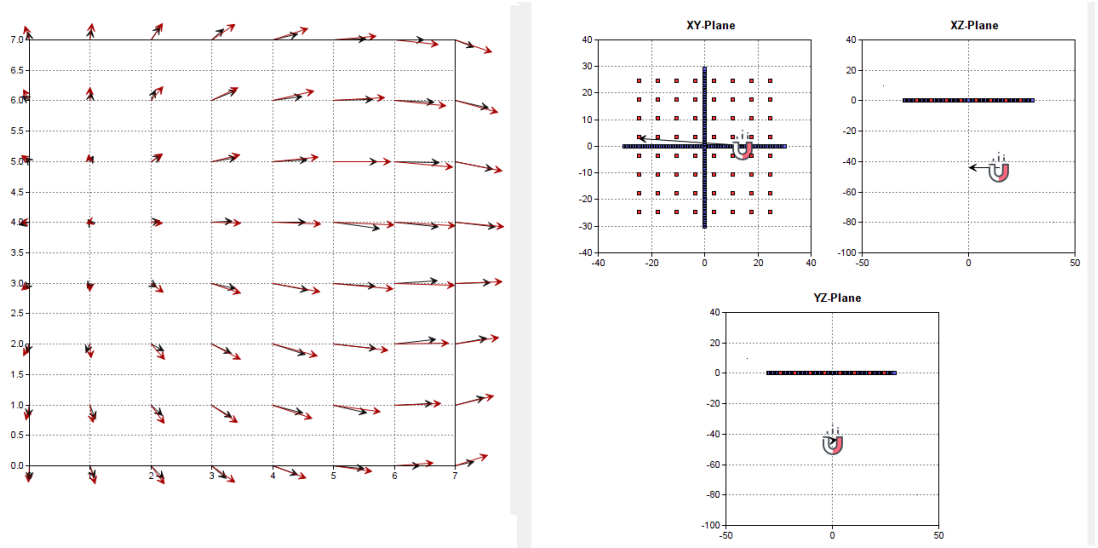


Abbildung 5.4: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position b).

Die Standortveränderung hat auf die Übereinstimmung der Vektorplots in diesem Fall keinen merklichen Einfluss und auch die räumliche Darstellung im XYZ-Plot stellt eine gute Repräsentation der tatsächlichen Position dar.

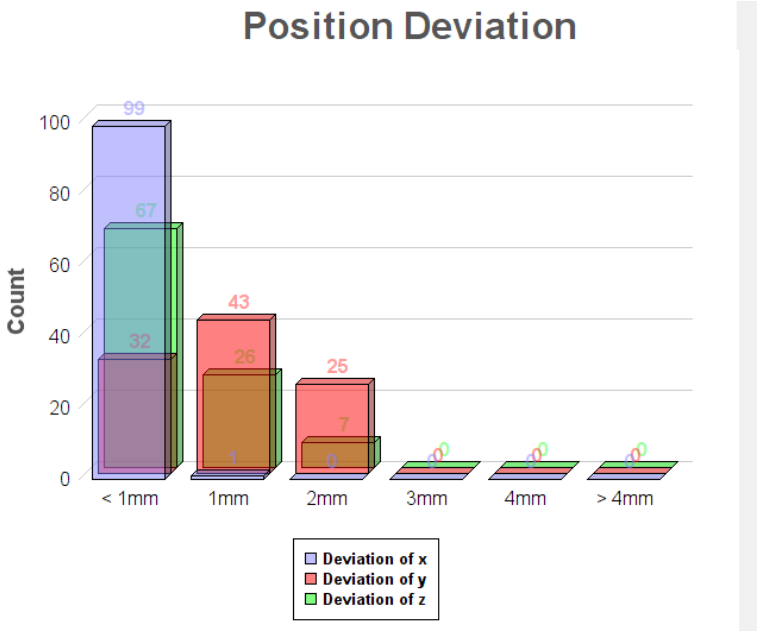


Abbildung 5.5: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position b).

Im Histogramm (vgl. Abbildung 5.5) ist wieder zu erkennen, wie gut die realen Werte den Werten des Suchalgorithmus gleichen. Den Erwartungen entgegen hat sich die Genauigkeit in x-Richtung nicht verändert, nur die in y-Richtung. In z-Richtung wurden ähnliche Ergebnisse erzielt wie im Abschnitt zuvor.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	99	1	0	0	0	0
y	67	25	7	0	0	0
z	32	43	25	0	0	0

Tabelle 5.3: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position b).

Die Mittelwerte der berechneten Koordinaten (vgl. Tabelle 5.4) sind knapp über 1 mm von den realen entfernt und die Gesamtstandardabweichung ist auch nur minimal angestiegen,

im Gegensatz zur vorherigen Position. Somit wurden den Erwartungen entsprechend zwar weniger genaue, aber noch immer gute Resultate erzielt.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	13.98	0.177	0.422	
y	1.184	0.203	0.451	0.511
z	-44.243	0.435	0.659	

Tabelle 5.4: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position b).

Position (0, 14, -45)

Bei Messungen an dieser Position (vgl. Abbildung 5.1 c)) ergeben sich, wie erwartet, ähnliche Messwerte wie auch schon bei der vorherigen Position (vgl. Abbildung 5.1 b)), da der Magnet auch hier nur in eine Richtung verschoben wird. Die Messwerte sind den folgenden Abbildungen und Tabellen zu entnehmen.

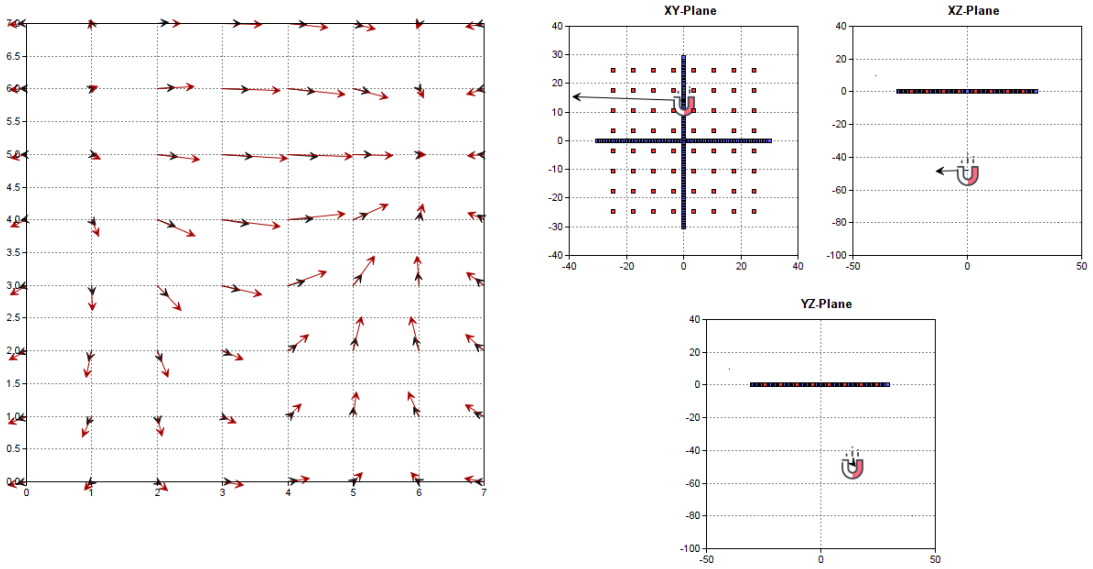


Abbildung 5.6: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position c).

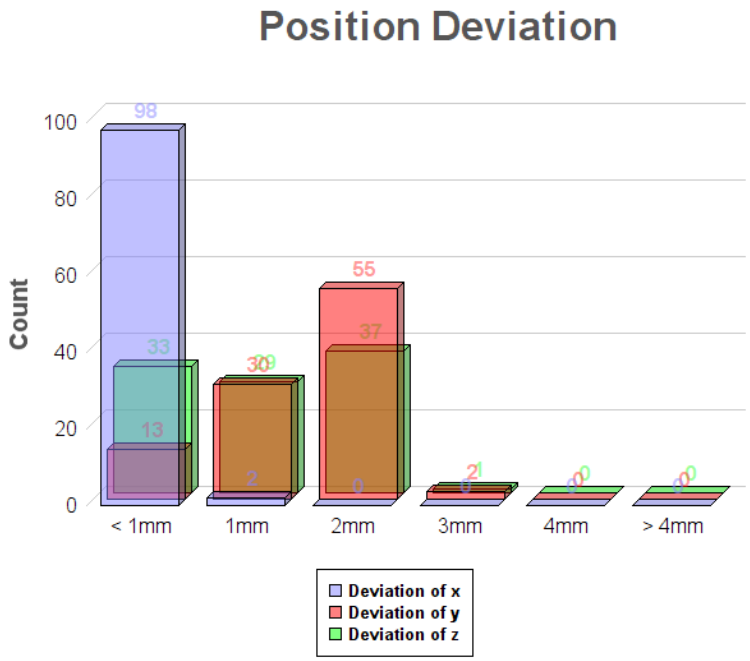


Abbildung 5.7: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position c).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	98	2	0	0	0	0
y	13	30	55	2	0	0
z	33	29	27	1	0	0

Tabelle 5.5: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position c).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.852	0.212	0.461	
y	16.839	0.179	0.423	0.53
z	-46.175	0.498	0.706	

Tabelle 5.6: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten an Position c).

Position (14, 14, -45)

Bei der hier verwendeten Anordnung des Magneten werden wieder etwas schlechtere Ergebnisse erwartet als bei den beiden zuvor. Diesmal ist die Position nicht nur in x-, sondern auch in y-Richtung verschoben.

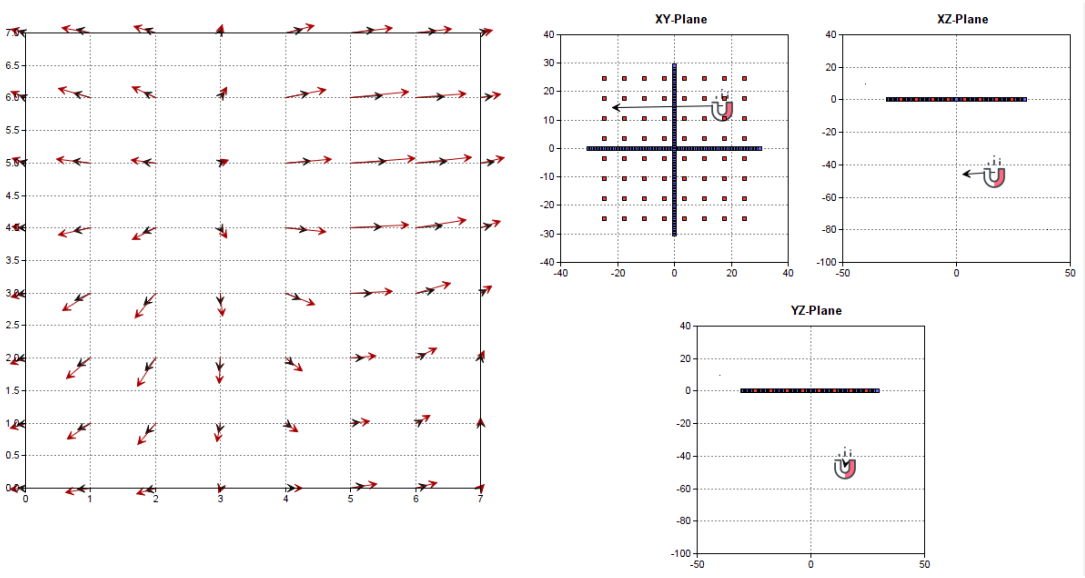


Abbildung 5.8: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position d).

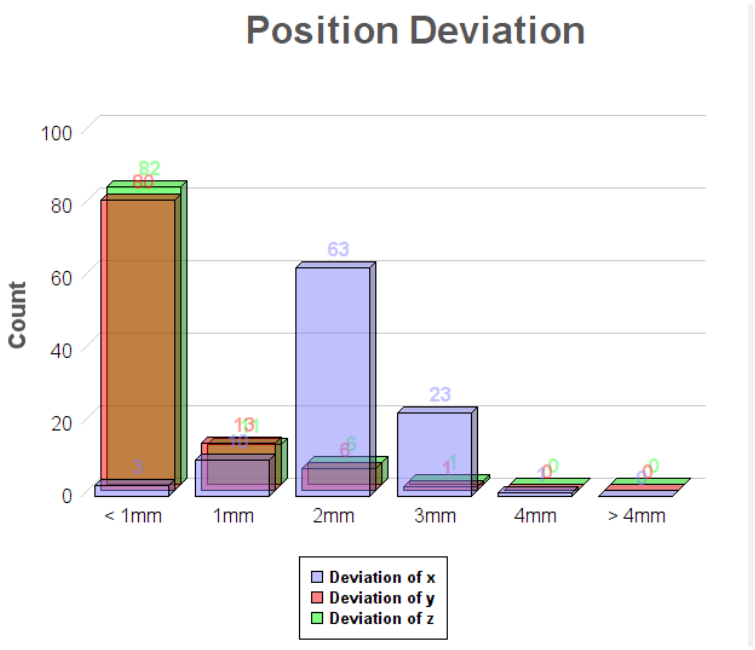


Abbildung 5.9: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position d).

Den Vermutungen entsprechend, hat sich die Genauigkeit der Suche verschlechtert, jedoch auffälligerweise nur in x-Richtung. Es wurden unerwartet gute Genauigkeiten bei den anderen beiden Koordinaten verzeichnet. Insgesamt sind die Abweichungen vom Mittelwert sichtbar angestiegen, allerdings können die erzielten Ergebnisse immer noch als gute Resultate eingestuft werden.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	3	10	63	23	0	0
y	80	13	6	1	0	0
z	82	11	6	1	0	0

Tabelle 5.7: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position d).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	16.093	0.290	0.538	
y	14.151	0.289	0.538	0.604
z	-45.107	0.541	0.735	

Tabelle 5.8: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position d).

Position (30, 0, -45)

Zum Schluss soll untersucht werden, was für eine Auswirkung eine Fehlplatzierung des Magneten über dem Array besitzt. Dafür wird dieser leicht außerhalb des letzten Sensors auf der x-Achse platziert (vgl. Abbildung 5.1 e)). Es ist davon auszugehen, dass der Algorithmus unter diesem Umstand kein außerordentlich aussagekräftiges Ergebnis mehr erbringen kann, da dieser nicht für solche Fälle ausgelegt wurde.

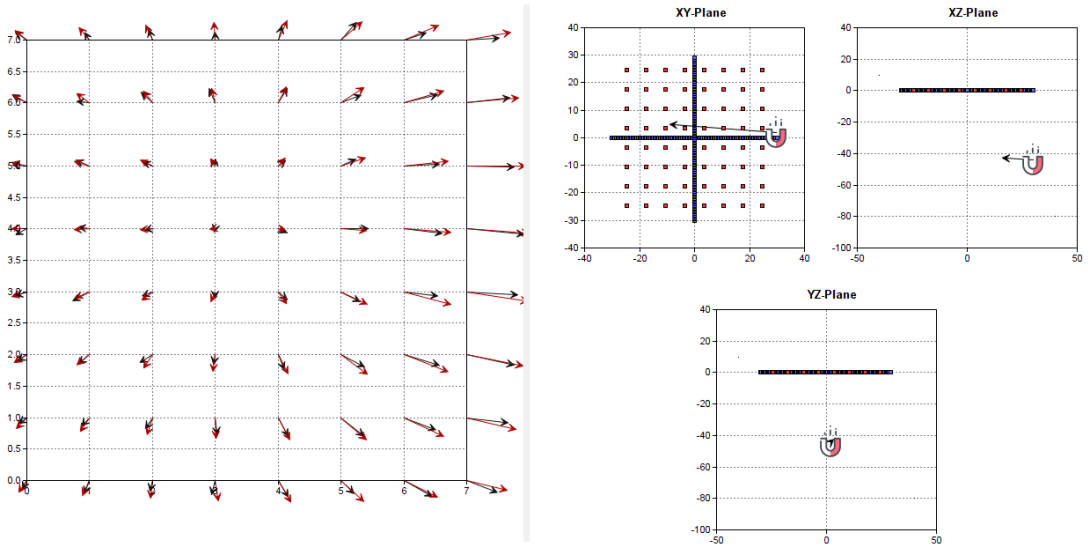


Abbildung 5.10: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position e).

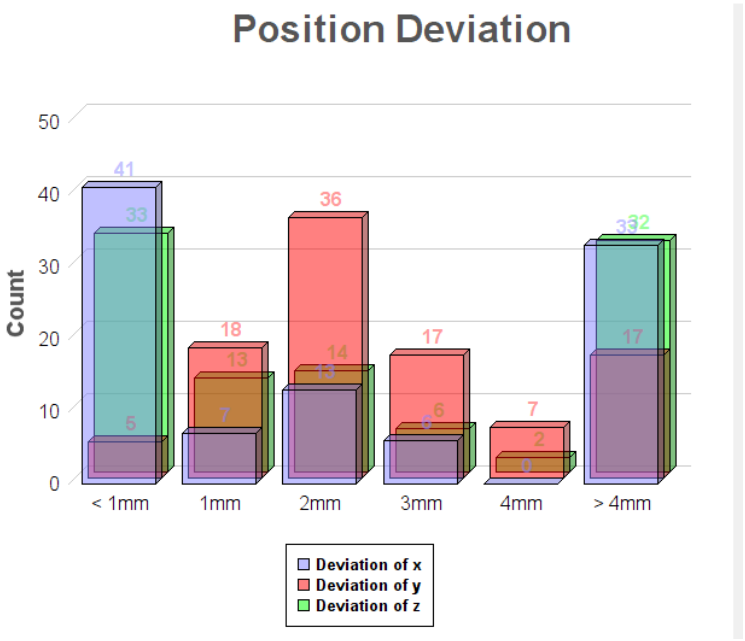


Abbildung 5.11: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position e).

Das Schätzwertmodell (vgl. Abbildung 5.10 (links)) stimmt unerwartet gut mit dem überein, welches das Sensor-Arrays gemessen hat und auch in der Ebenendarstellung (vgl. Abbildung 5.10 (rechts)) wird teilweise die Position richtig dargestellt. Allerdings springt die Position oft von einem Punkt zu einem völlig anderen, da die Suche nicht mehr in jedem Durchlauf zu einem zufriedenstellenden Ergebnis führt (vgl. Abbildung 5.12).

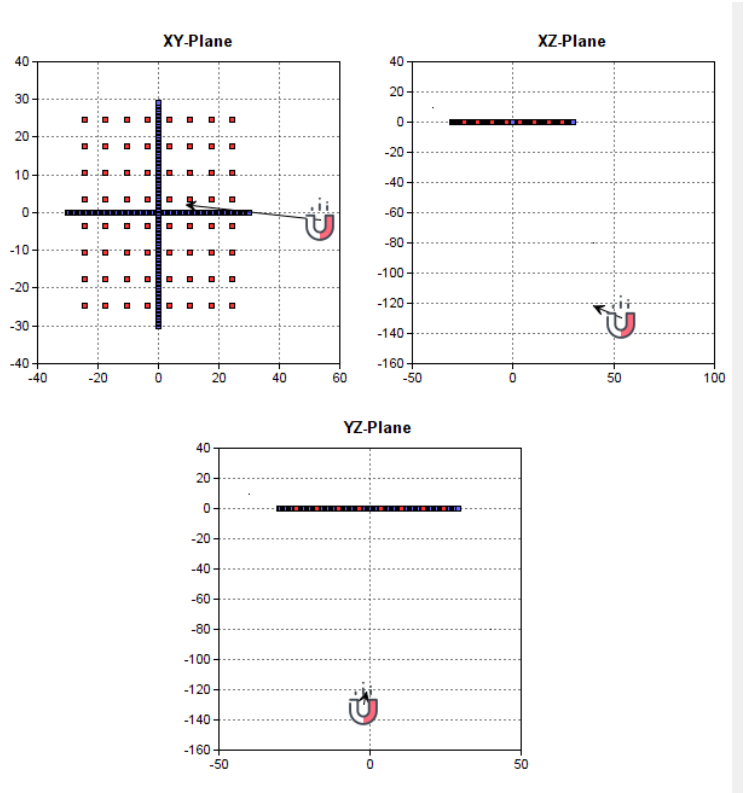


Abbildung 5.12: Berechnete Position des Magneten bei einer Positionierung außerhalb des Arrays mit nicht mehr ausreichendem Ergebnis an Position e).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	41	7	13	6	0	33
y	5	18	36	17	7	17
z	33	13	14	6	2	32

Tabelle 5.9: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position e).

Den beigefügten Tabellen ist zu entnehmen, dass sich die Genauigkeit des Suchalgorithmus drastisch verschlechtert hat und keine aussagekräftigen Schlüsse mehr gezogen werden können. Zum Teil stimmt die Erkennung des Ortes mit der tatsächlichen Position überein, aber nicht mehr ausreichend oft genug, um mit den Resultaten eine eindeutige Positionsbestimmung durchzuführen.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	51.658	1136.681	33.715	
y	2.650	22.539	4.748	61.675
z	-59.320	539.036	23.217	

Tabelle 5.10: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position an Position e).

Position (0, 30, -45)

Es soll noch eine weitere Positionierung eines Permanentmagneten außerhalb des Arrays getestet werden. Es lässt sich die Vermutung anstellen, dass ähnliche Ergebnisse erzielt werden, wie bei der vorherigen Messung.

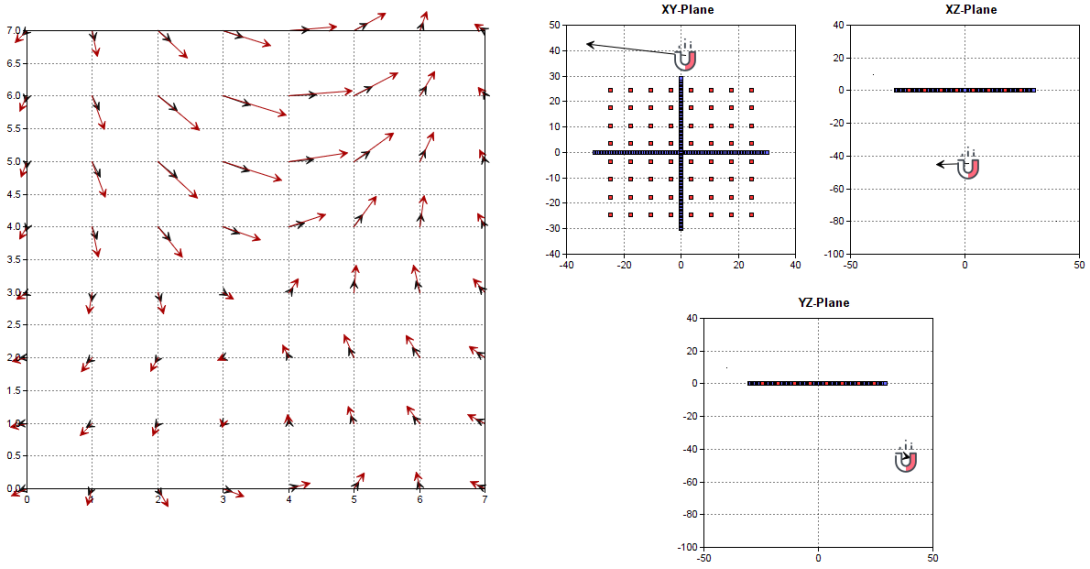


Abbildung 5.13: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) an Position f).

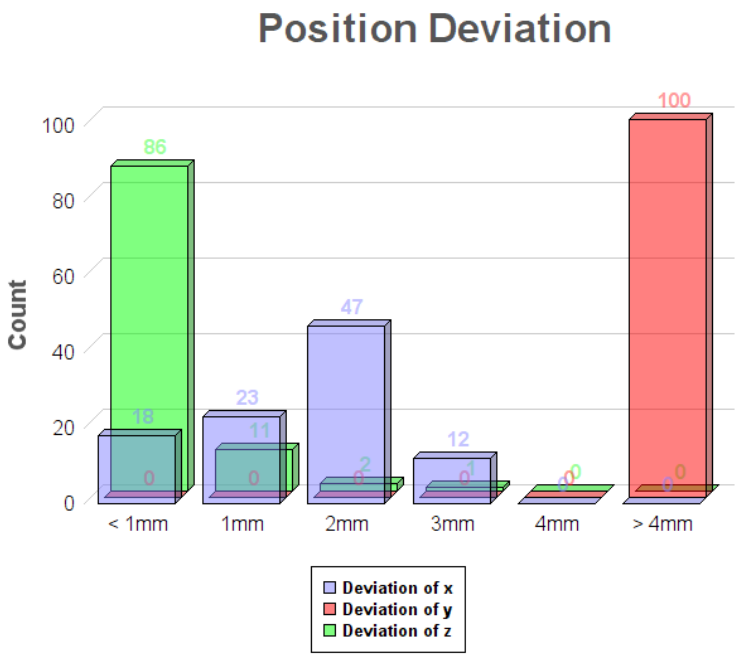


Abbildung 5.14: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen an Position f).

In diesem Fall wurde die Höhenlage überraschenderweise sehr genau ermittelt, bei der Positionsbestimmung in der XY-Ebene sind, wie vermutet wurde, jedoch keine eindeutigen Ergebnisse erreicht worden.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	18	23	47	12	0	0
y	0	0	0	0	0	100
z	86	11	2	1	0	0

Tabelle 5.11: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position f).

Insgesamt ist bei dieser gewählten Anordnung die Genauigkeit wesentlich besser als bei der vorherigen Messung und es würde sich die Magnetposition je nach Anwendungsfall sogar noch zufriedenstellend lokalisieren lassen. Da die Messungen aber nicht immer gleich stabil ablaufen, wenn der Magnet nicht in dem geplanten Bereich positioniert wird, wird davon abgeraten, sich auf Messergebnisse zu stützen, welche so erzielt worden sind.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	1.574	0.342	0.584	
y	37.433	0.997	0.999	0.801
z	-44.825	0.674	0.821	

Tabelle 5.12: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position an Position f).

5.1.2 Genauigkeit des Algorithmus bei verschiedenen Höhenlagen des Magneten

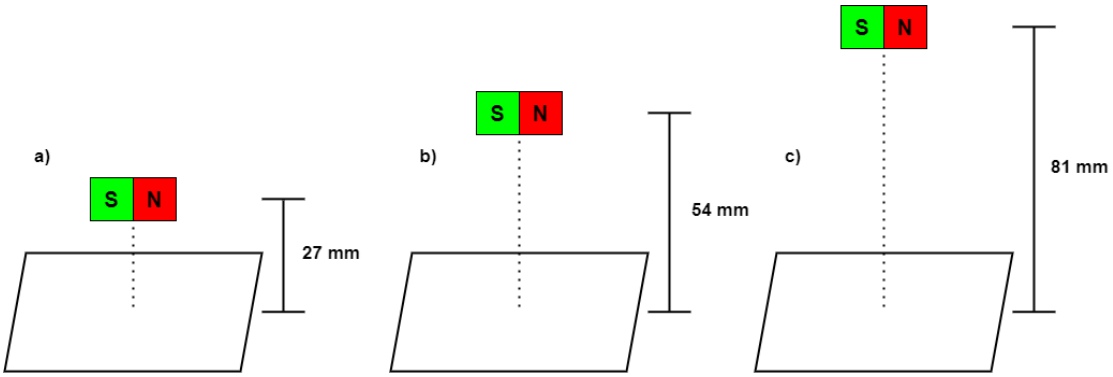


Abbildung 5.15: Verschiedene Höhenlagen des Magneten zum Testen des Algorithmus.

In diesem Abschnitt soll die Reliabilität des Suchalgorithmus bei der Positionierung des Magneten in unterschiedlichen Höhen untersucht werden. Die Vermutung liegt nahe, dass je weiter entfernt der Gebermagnet von dem Sensor-Array angeordnet wird, desto größer die Abweichung zwischen tatsächlicher und von der Software ermittelten Position. Für die Messungen wird der Permanentmagnet in der Mitte des Arrays in drei unterschiedlichen Entfernungen positioniert und anschließend die Messergebnisse festgehalten (vgl. Abbildung 5.15).

Position (0, 0, -27)

Zuerst soll überprüft werden, wie gut die Positionsbestimmung des Algorithmus funktioniert, wenn der Magnet sich relativ nah am Sensor-Array befindet. Da hier die Magnet-

flussdichte noch gut von den TMR-Sensoren gemessen werden kann, ist zu erwarten, dass die Abweichung hier relativ kleine Werte annimmt.

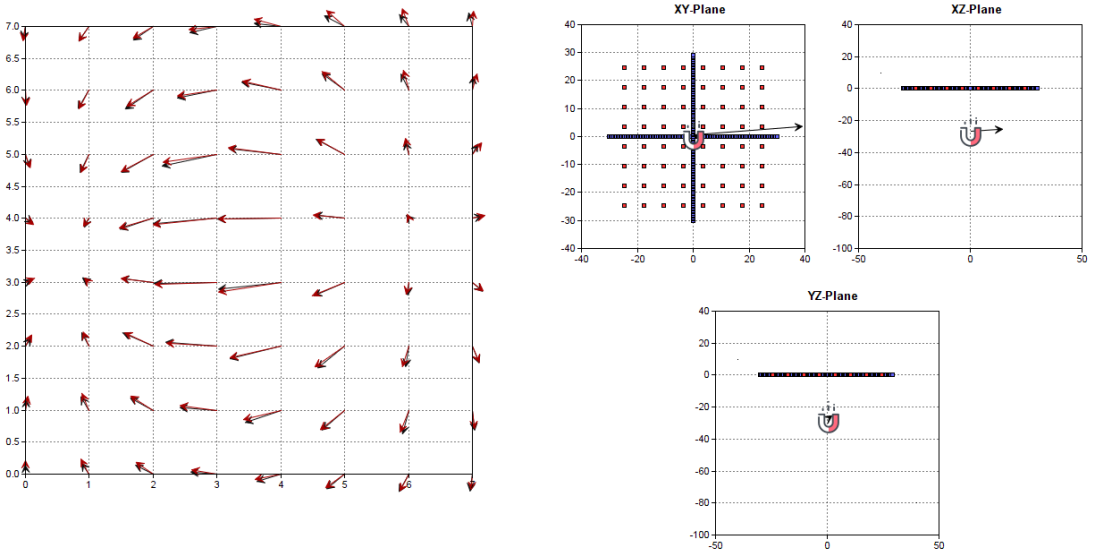


Abbildung 5.16: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Höhe a).

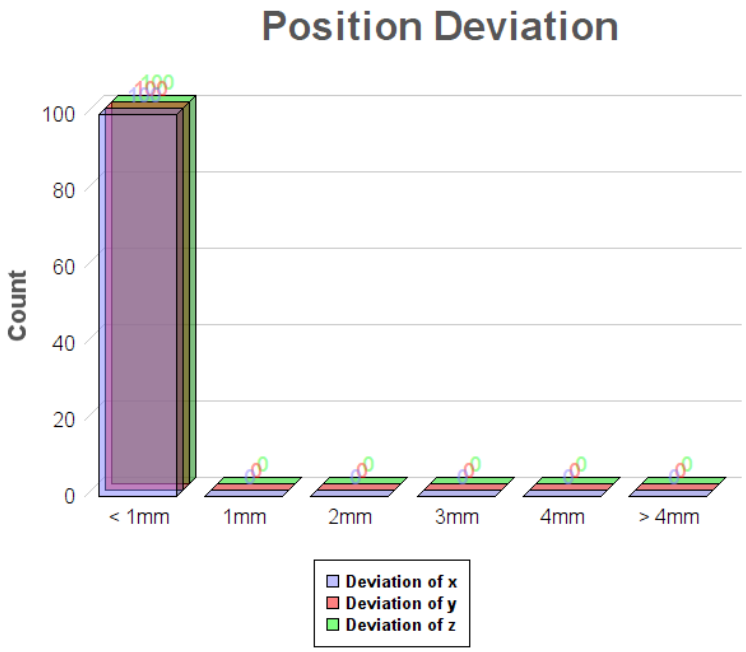


Abbildung 5.17: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Höhe a).

Im Histogramm ist deutlich zu erkennen, dass, wie erwartet, die Position sehr gut bestimmt werden kann. Alle Abweichungen sind kleiner als 1 mm und somit können die Koordinaten des Permanentmagneten außerordentlich präzise bestimmt werden.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	100	0	0	0	0	0
y	100	0	0	0	0	0
z	100	0	0	0	0	0

Tabelle 5.13: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei Höhe a).

Die guten Resultate spiegeln sich auch noch einmal in den Tabellen mit den Messergebnissen wieder. Die Gesamtstandardabweichung beträgt im Mittel gerade einmal ungefähre 0.1 mm.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.170	0.008	0.087	
y	-0.233	0.009	0.095	0.107
z	-27.117	0.019	0.139	

Tabelle 5.14: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei Höhe a).

Position (0, 0, -54)

In nächsten Schritt wird die Höhe des Magneten über dem Sensor-Array verdoppelt und es liegt die Vermutung nahe, dass deshalb auch die Abweichungen zwischen realer und ermittelter Position zunehmen werden.

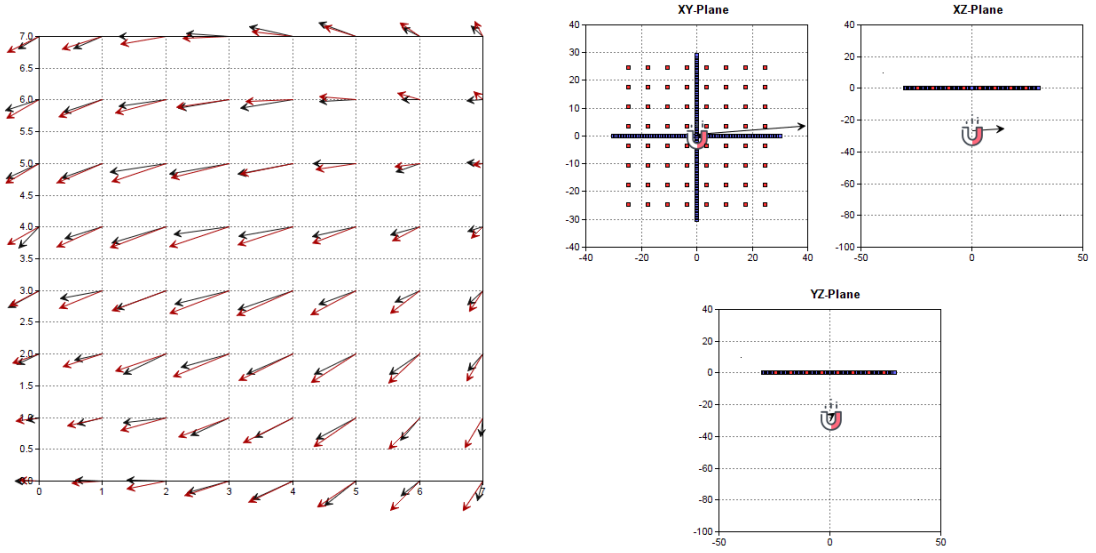


Abbildung 5.18: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Höhe b).

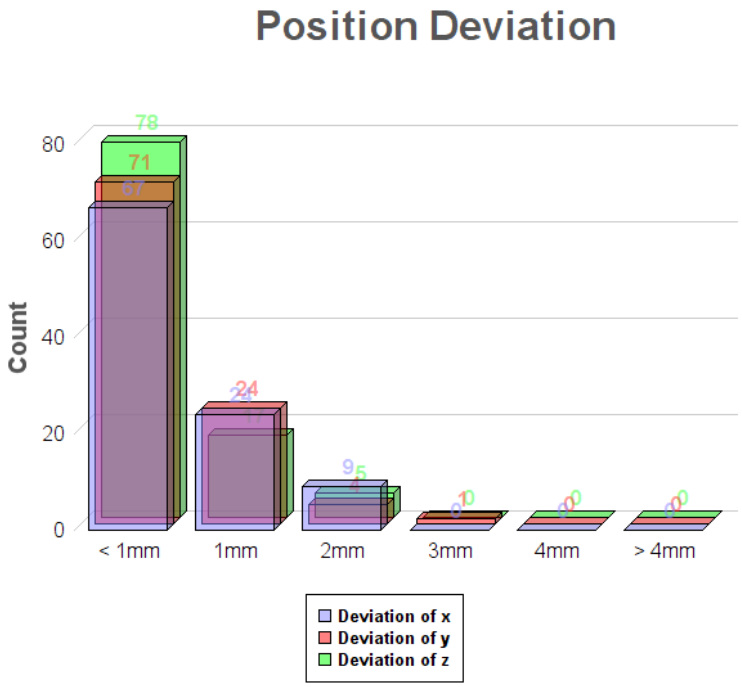


Abbildung 5.19: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Höhe b).

Alle Koordinaten des Gebermagneten werden mit mindestens zwei Drittel Genauigkeit richtig ermittelt und stellen insgesamt noch immer ein gutes Ergebnis dar. Die Abweichungen sind zwar wie angenommen gestiegen, aber noch nicht in einem Maß, welches die Reliabilität des Suchalgorithmus stark beeinträchtigt.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	67	24	9	0	0	0
y	71	24	4	1	0	0
z	78	17	5	0	0	0

Tabelle 5.15: Abweichung zwischen berechneter und tatsächlicher Position des Magneten an Position b).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.833	0.282	0.531	
y	0.493	0.400	0.632	0.661
z	-50.881	0.671	0.819	

Tabelle 5.16: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position b).

Position (0, 0, -81)

Im letzten Untersuchungsschritt wird der Magnet um weitere 27 mm angehoben. Auch hier ist eine weitere Verschlechterung der Messergebnisse zu erwarten. Unter Umständen ist die Magnetisierungsstärke nicht mehr aussreichend, damit die Software den Ort des Magneten noch korrekt ermitteln kann.

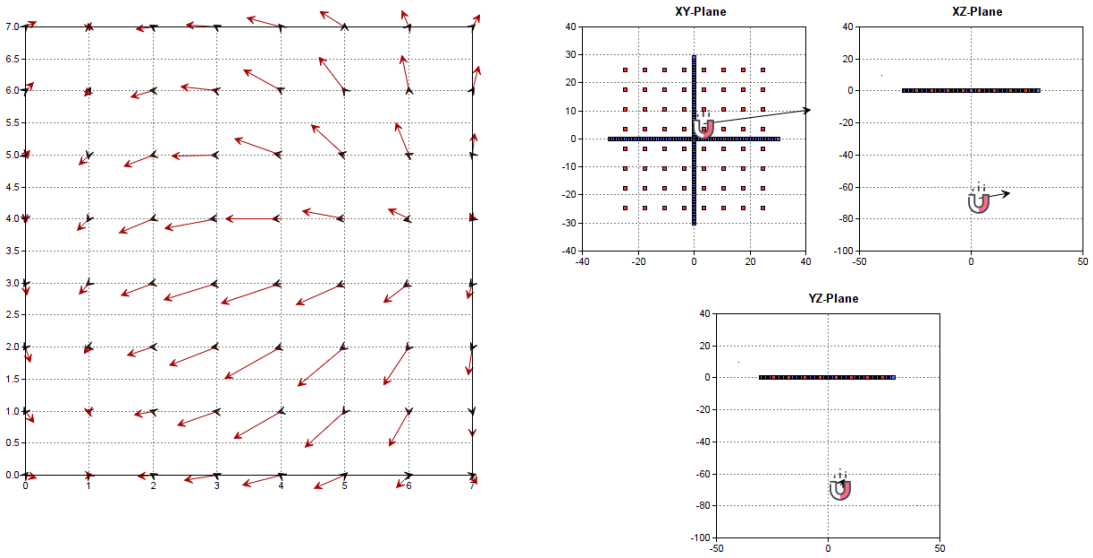


Abbildung 5.20: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Höhe c).

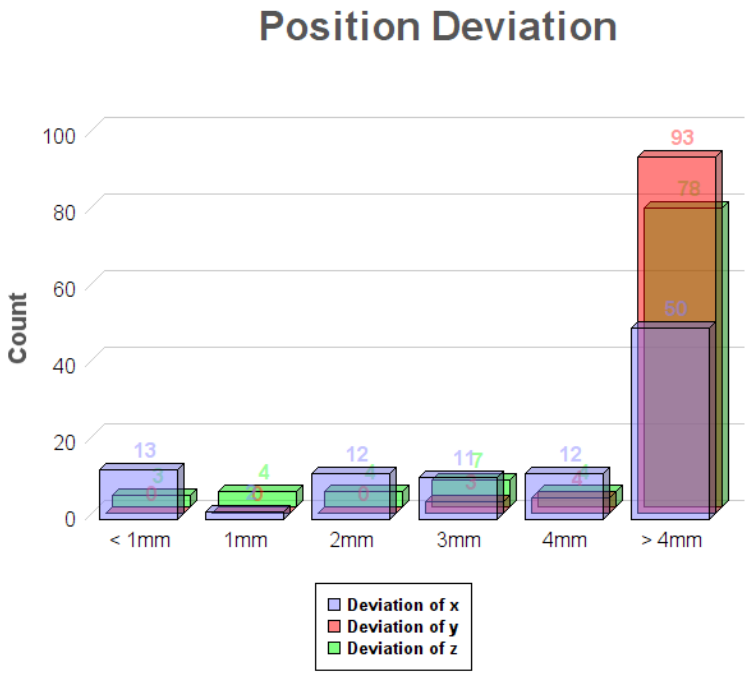


Abbildung 5.21: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Höhe c).

In Abbildung 5.21 ist merklich sichtbar, dass die ermittelte Position nicht mehr ausreichend Aufschluss über den tatsächlichen Ort des Magneten geben kann. Auch in der XYZ-Darstellung wird dieser nicht mehr korrekt wiedergegeben und die Position springt hier von einem Punkt zum nächsten (vgl. Abbildung 5.20 und 5.22).

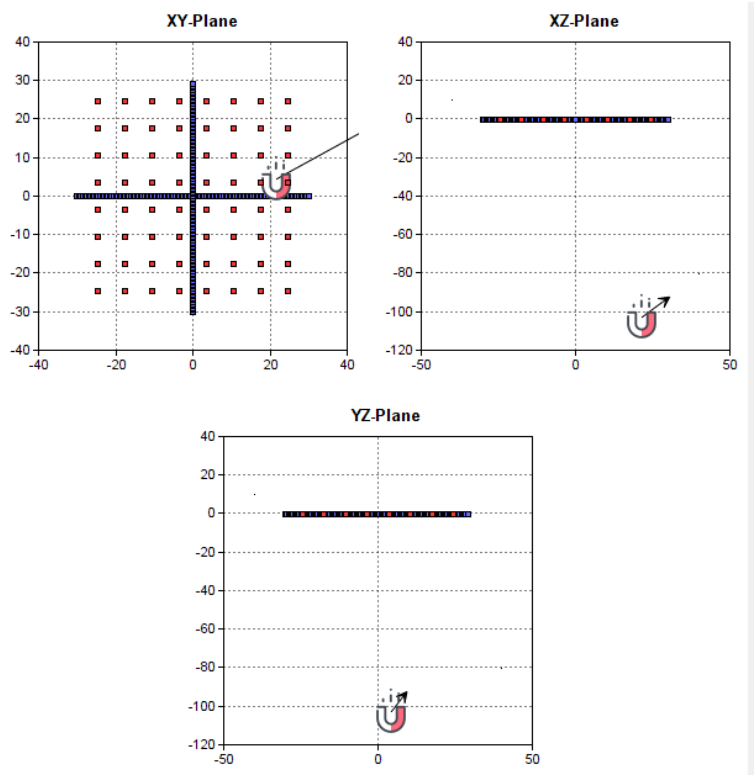


Abbildung 5.22: Berechnete Position des Magneten bei einer Positionierung außerhalb des Arrays mit nicht mehr ausreichendem Ergebnis bei Höhe c).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	13	2	12	11	12	50
y	0	0	0	3	4	93
z	3	4	4	7	4	78

Tabelle 5.17: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei Höhe c).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-3.759	39.179	6.259	
y	-15.158	70.352	8.388	11.963
z	-101.085	451.227	21.243	

Tabelle 5.18: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei Höhe c).

5.1.3 Genauigkeit des Algorithmus bei unterschiedlichen Winkellagen des Magneten

Im letzten Schritt wird der Magnet in verschiedenen Winkellagen in einer Höhe von ungefähr 45 mm über dem Sensor-Array positioniert. Es soll überprüft werden, ob die Lage des Permanentmagneten Auswirkungen auf das Ergebnis des Algorithmus hat und wenn ja, in welchem Ausmaß.

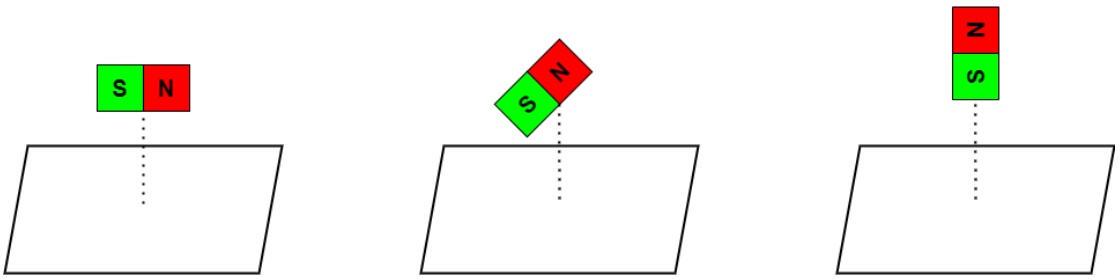


Abbildung 5.23: Verschiedene Magnetlagen zum Testen des Algorithmus.

Position (0, 0, -45), Winkel 30°

Zuallererst wird getestet, wie genau der Algorithmus arbeitet, wenn der Magnet in einer leichten Schräglage (hier 30°) über dem Array angeordnet wird. Die Vermutung liegt nahe, dass sich die Genauigkeit mit steigendem Winkel verschlechtert, da solche Lagen nicht explizit in der Software vorgesehen sind.

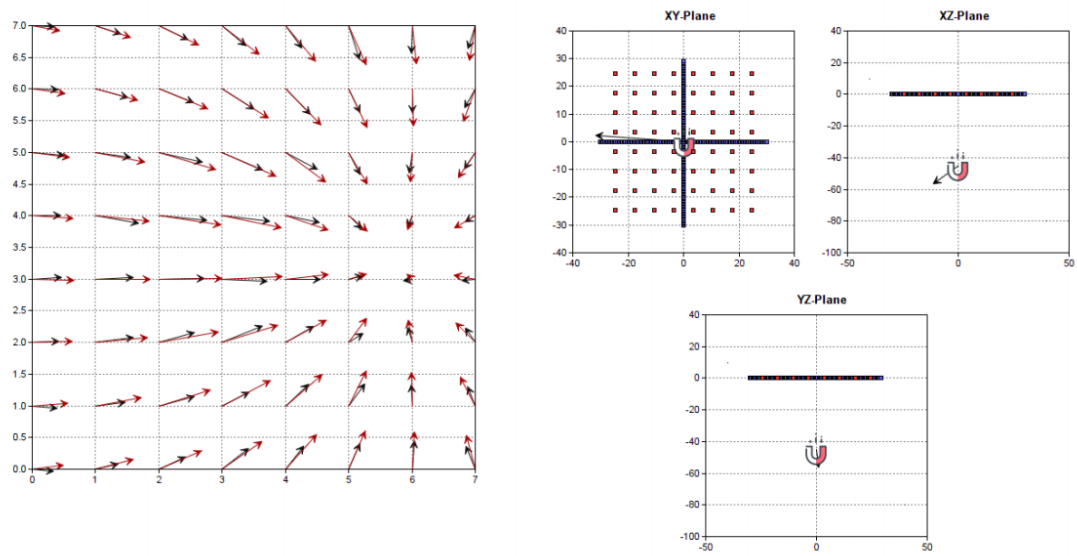


Abbildung 5.24: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Winkel a).

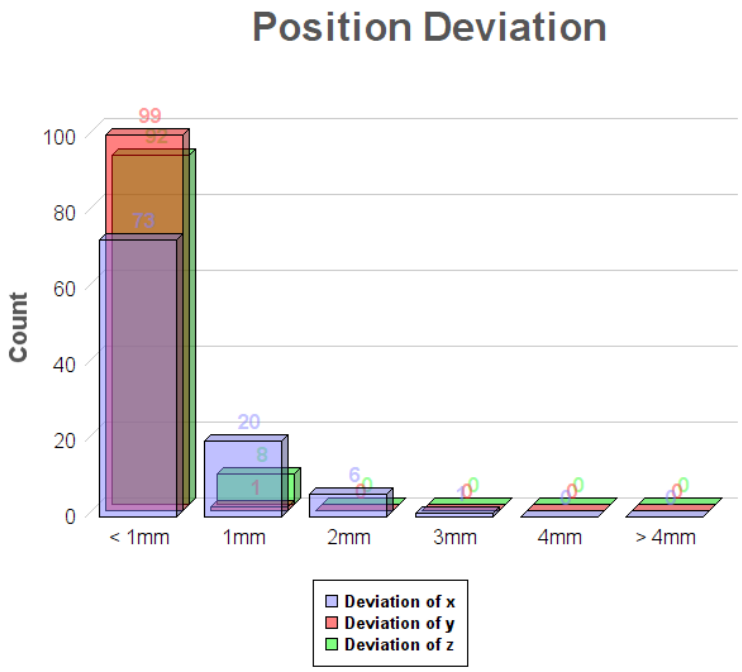


Abbildung 5.25: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Winkel a).

Trotz der leicht angewinkelten Magnetposition sind im Histogramm nur wenige Abweichungen zwischen tatsächlichem und errechnetem Ort zu erkennen (vgl. Abbildung 5.25).

Abweichung in mm	< 1 mm	1 mm	2 mm	3 mm	4 mm	> 4 mm
x	73	20	6	0	0	50
y	99	8	0	0	0	0
z	92	1	0	0	0	0

Tabelle 5.19: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei Höhe a).

In den beiden Tabellen wird deutlich, dass die Präzision der Suche, im Gegensatz zu einer vollständigen waagerechten Lage des Gebermagneten, abgenommen hat, jedoch in keinem Ausmaß, welches das Resultat stark beeinflussen würde. Der Ort des Magneten wird immer noch gut erkannt und es sind keine zu starken Differenzen gemessen worden.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.586	1.114	1.056	
y	-0.627	0.265	0.514	0.702
z	-45.296	0.288	0.536	

Tabelle 5.20: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei Höhe a).

Position (0, 0, -45), Winkel 60°

In diesem Abschnitt wird der Magnet wieder in derselben Höhe und Stelle positioniert, jedoch wird der Winkel verdoppelt auf 60°. Es ist zu erwarten, dass sich die Ergebnisse ein weiteres Mal verschlechtern und die Reliabilität des Algorithmus abnimmt.

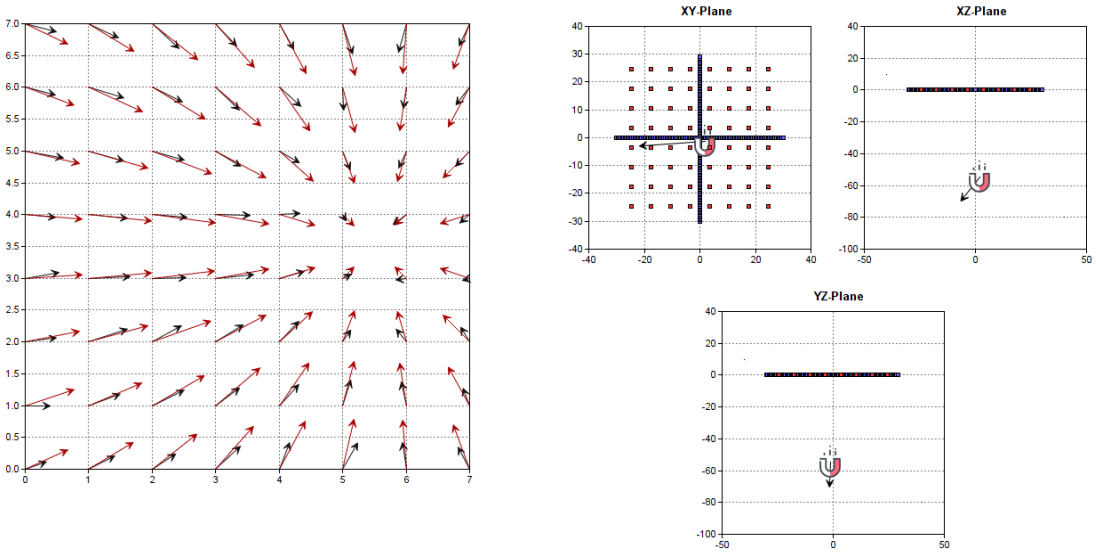


Abbildung 5.26: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Winkel b).

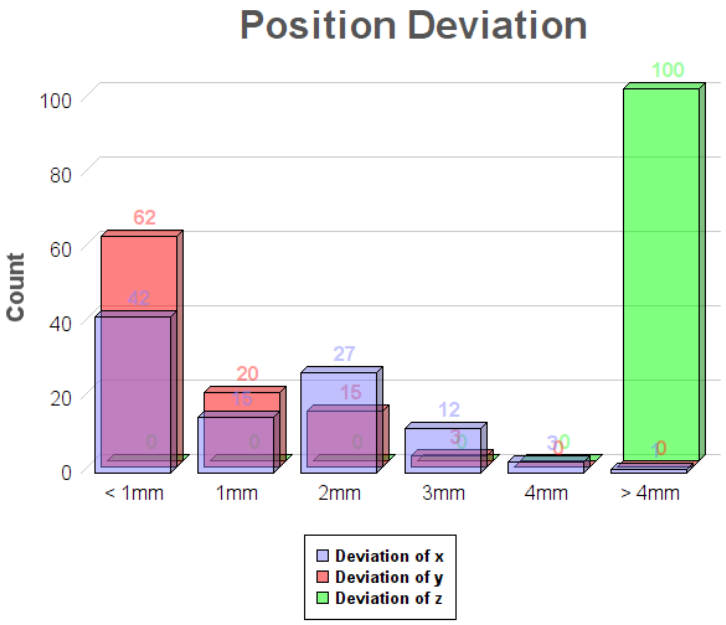


Abbildung 5.27: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Winkel b).

In dem Histogramm zu diesem Versuch (vgl. Abbildung 5.27) ist zu erkennen, dass auch die Abweichungen mit steigendem Winkel größer geworden sind. Die Höhe des Magneten kann nicht mehr so zuverlässig wie zuvor bestimmt werden, auch wenn die Lokalisierung in der XY-Ebene noch besser als zunächst vermutet funktioniert.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	42	15	27	12	3	1
y	62	20	15	3	0	0
z	0	0	0	0	0	100

Tabelle 5.21: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei Winkel b).

Auch in den Tabellen ist nochmal erkenntlich, dass die X- und Y-Koordinaten besser bestimmt werden konnten als die Z-Koordinate und auch die Gesamtstandardabweichung sich fast verdoppelt hat. Für Anwendungen, bei welchen die Höhe keinen signifikanten Parameter darstellt, könnten diese Ergebnisse noch als zufriedenstellende Resultate eingestuft werden.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.781	2.321	1.524	
y	0.566	0.798	0.894	1.243
z	-54.953	1.717	1.310	

Tabelle 5.22: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei Winkel b).

Position (0, 0, -45), Winkel 90°

Zum Schluss wird der Permanentmagnet in einer vertikalen Lage (hier 90°) über dem Sensor-Array an derselben Stelle positioniert. Es wird angenommen, dass bei einer solchen Anordnung wieder präzisere Resultate erzielt werden können.

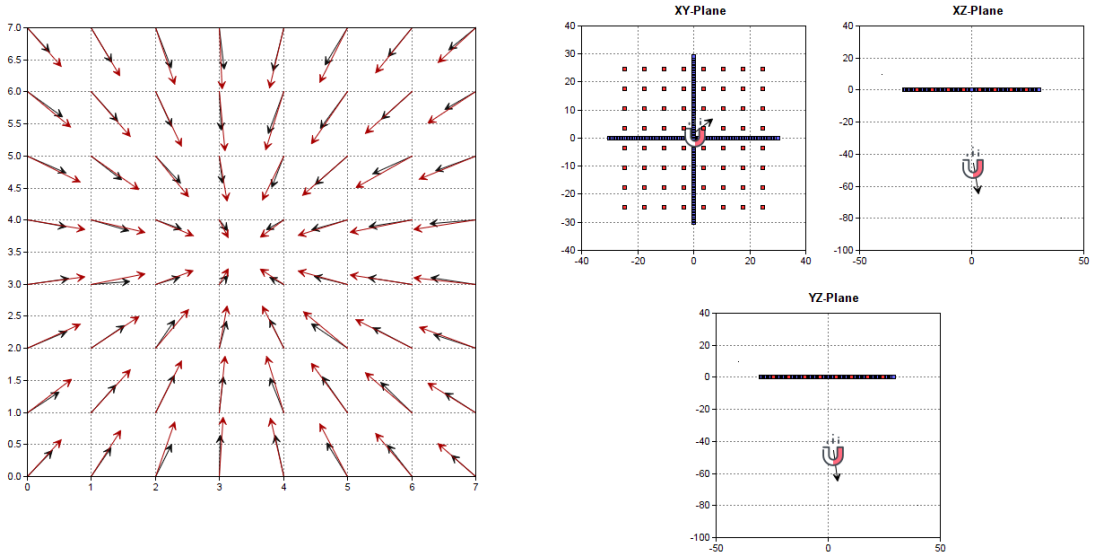


Abbildung 5.28: Vergleich der Vektorplots des realen Sensorarrays (schwarz) und des Schätzwertemodells (rot) bei Winkel c).

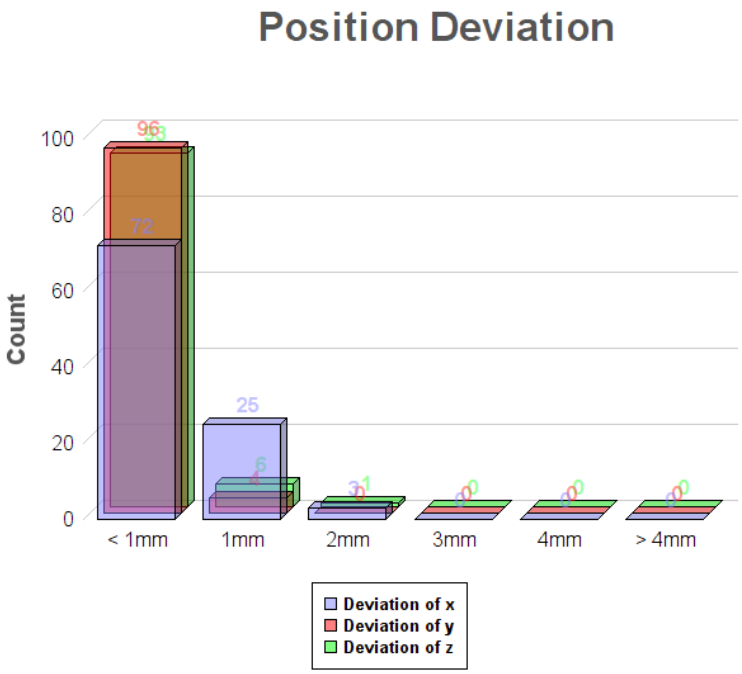


Abbildung 5.29: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei Winkel c).

Es ist eine merkliche Verbesserung zum vorherigen Versuch sichtbar (vgl. Abbildung 5.29), da kaum noch Abweichungen zwischen echter und berechneter Position zu verzeichnen sind.

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	72	25	3	0	0	0
y	96	4	0	0	0	0
z	93	6	1	0	0	0

Tabelle 5.23: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei Winkel c).

Im Mittelwert stimmt die von der Software bestimmte Position des Magneten gut mit der tatsächlichen überein. Auch die Gesamtstandardabweichung ist wieder auf einen kleinen Wert gesunken.

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.780	0.135	0.368	
y	0.513	0.104	0.323	0.406
z	-44.951	0.277	0.526	

Tabelle 5.24: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei Winkel c).

5.1.4 Computerauslastung

Ein weiterer sinnvoller Test ist das Überprüfen der Ressourcennutzung bei langen Laufzeiten der Applikation. Dafür wurde die Anwendung 240 Minuten lang auf einem Computer aktiviert und untersucht, wie stark Prozessor und Prozessspeicher von dieser belastet werden. Aktiv bedeutet in diesem Fall, dass die gemessenen Werte des Sensor-Arrays als Vektor-Plot angezeigt und auch der Suchalgorithmus gestartet wurde. Die gesamte Laufzeit über blieben die Werte stabil und die Speicherauslastung stieg nicht über 52 MB (vgl. Abbildung 5.30). Auch der Prozessor wird kaum beansprucht, was dafür spricht, dass moderne Rechner durch die entworfene Software kaum belastet werden. Somit lässt sich abschließend festhalten, dass die Applikation auch längere Zeit genutzt werden kann, ohne Probleme zu verursachen.

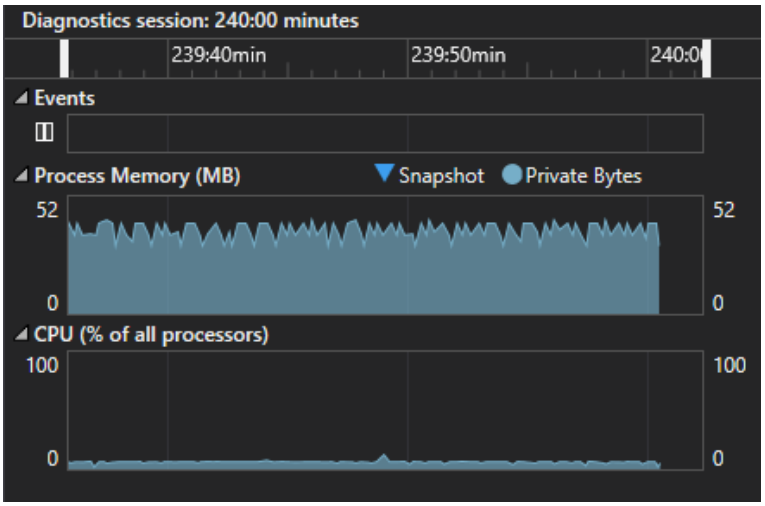


Abbildung 5.30: Langzeitüberwachung des Prozessspeichers der Anwendung bei einer Laufzeit von 240 Minuten.

5.1.5 Funktionstest unter Linux

Um die Platzformunabhängigkeit zu beweisen, wird in diesem Abschnitt die Funktionalität der entwickelten Applikation unter Linux getestet. Dafür wird die Software zuallererst mit Hilfe von CMake unter Linux erstellt und anschließend gestartet. Als erstes baut sich das Hauptfenster der Anwendung auf, welches bis auf das von der Plattform genutzte dunkle Design, fast identisch aussieht wie unter Windows (vgl. Abbildung 5.31).

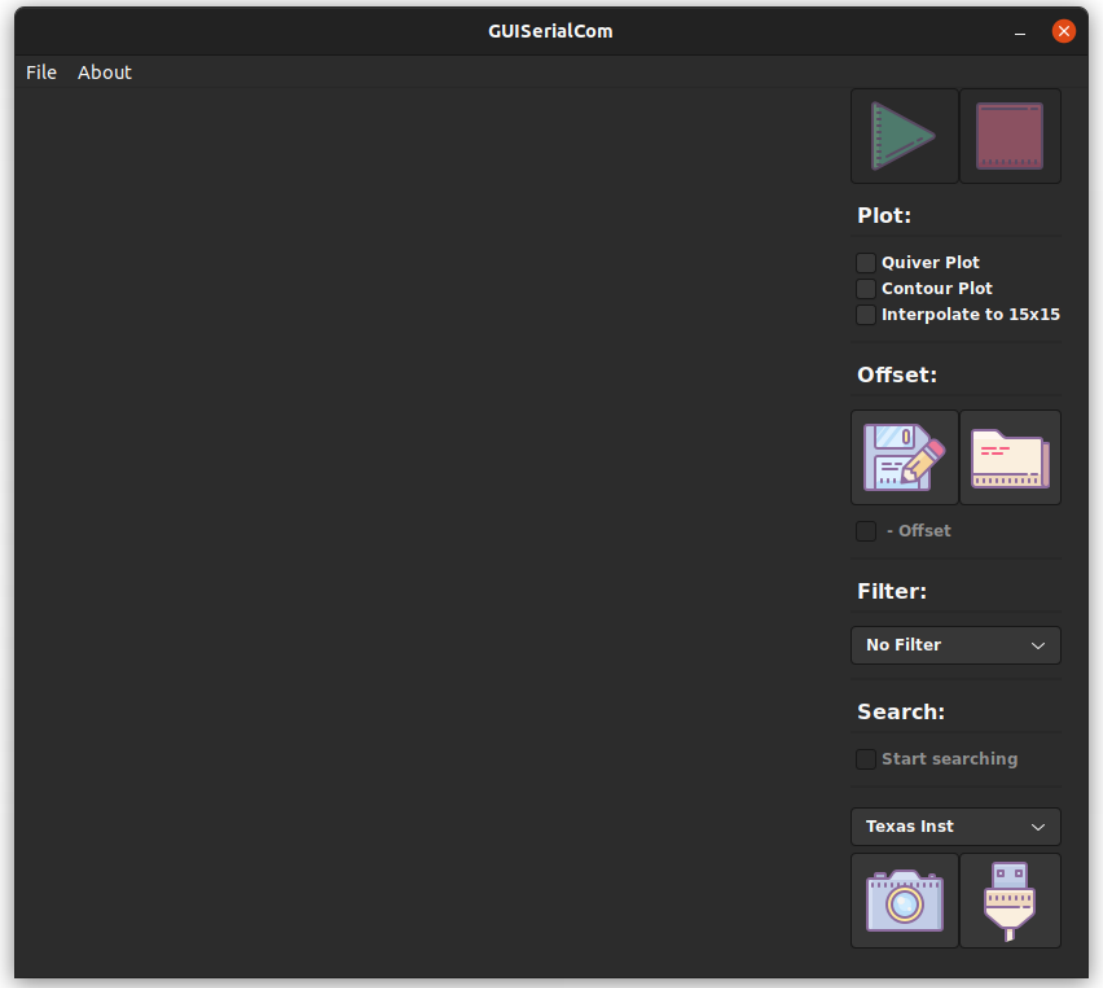


Abbildung 5.31: Hauptfenster der Anwendung unter Linux.

Anschließend wird die Funktionalität der Bedien- und Einstellungsmöglichkeiten getestet. Dafür werden alle Plotmöglichkeiten einmal ausgewählt, der Offset gespeichert und

geladen, die Filter eingestellt, der COM-Port ausgewählt, Bilder gespeichert und anschließend die Suche über das zugehörige Kontrollkästchen gestartet. Alles arbeitet genauso gut, wie zuvor unter Windows. In Abbildung 5.32 wurden diverse Einstellungen vorgenommen und es ist die daraus resultierende Grafik zu erkennen.

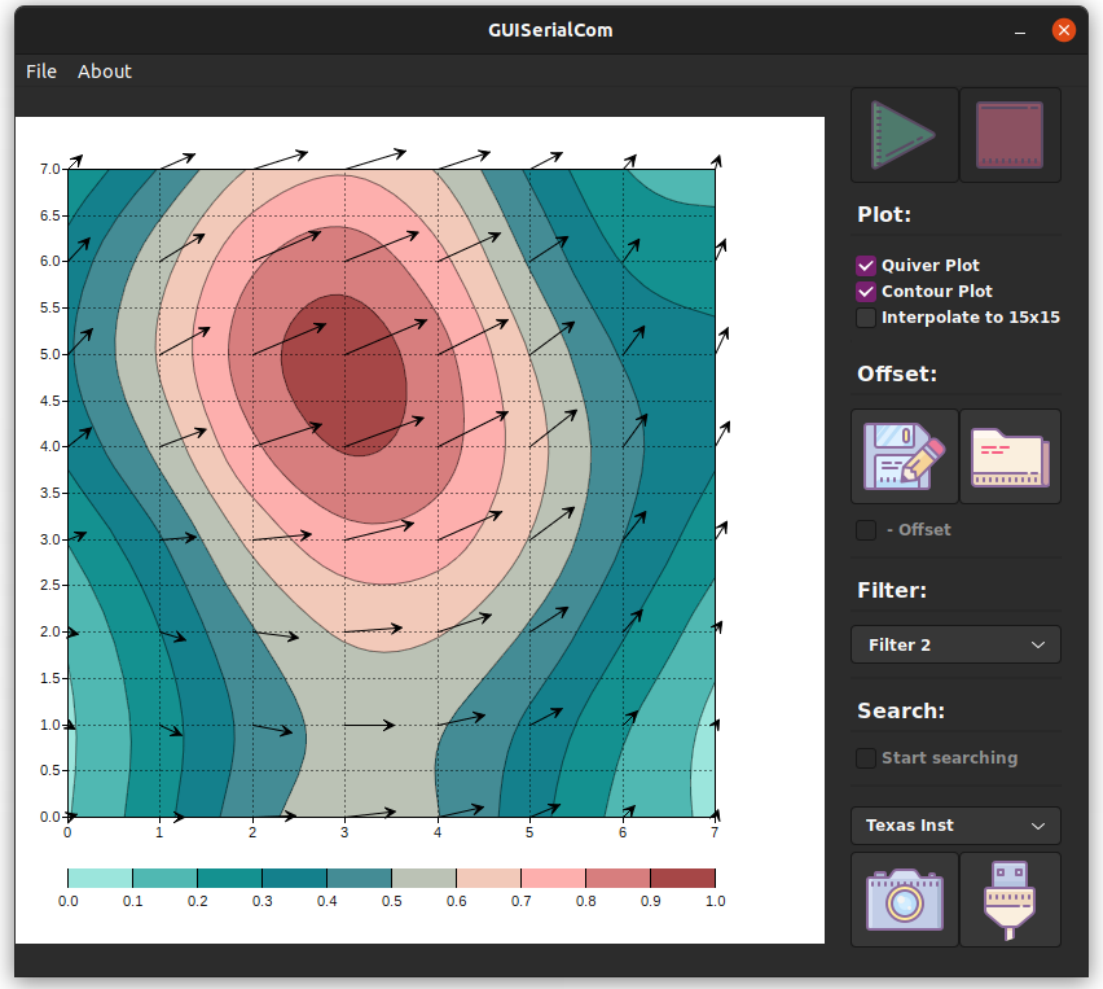


Abbildung 5.32: Hauptfenster der Anwendung unter Linux mit verschiedenen vorgenommenen Einstellungen.

Nach dem Starten der Suche öffnet sich, wie erhofft, das Unterfenster der Applikation. In diesem soll die ermittelte Magnetposition dargestellt werden (vgl. Abbildung 5.33).

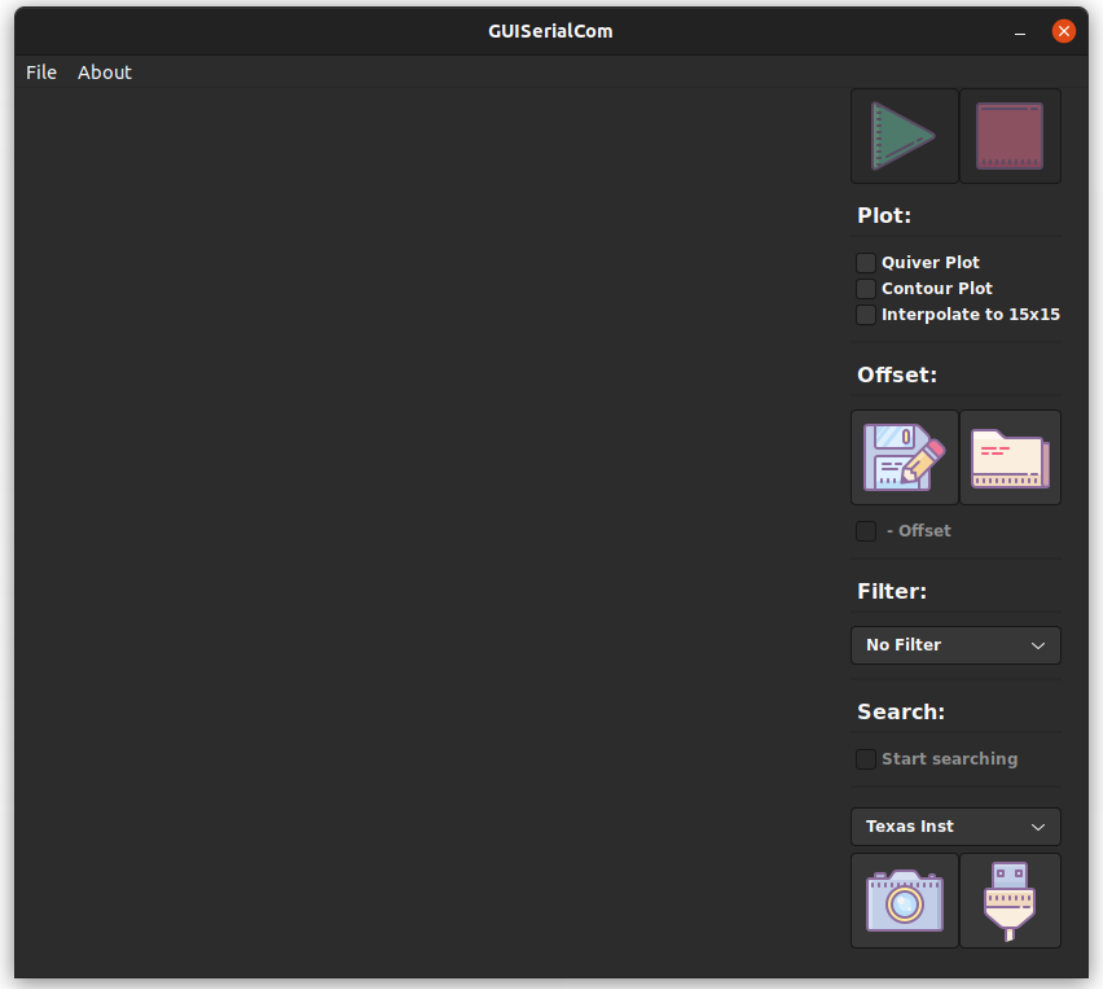


Abbildung 5.33: Unterfenster der Anwendung unter Linux.

Auch im Unterfenster arbeitet fast alles genauso wie zuvor unter Windows. In Abbildung 5.34 wurden wieder verschiedene Einstellungen vorgenommen und es ist der sich daraus ergebende Plot zu erkennen.

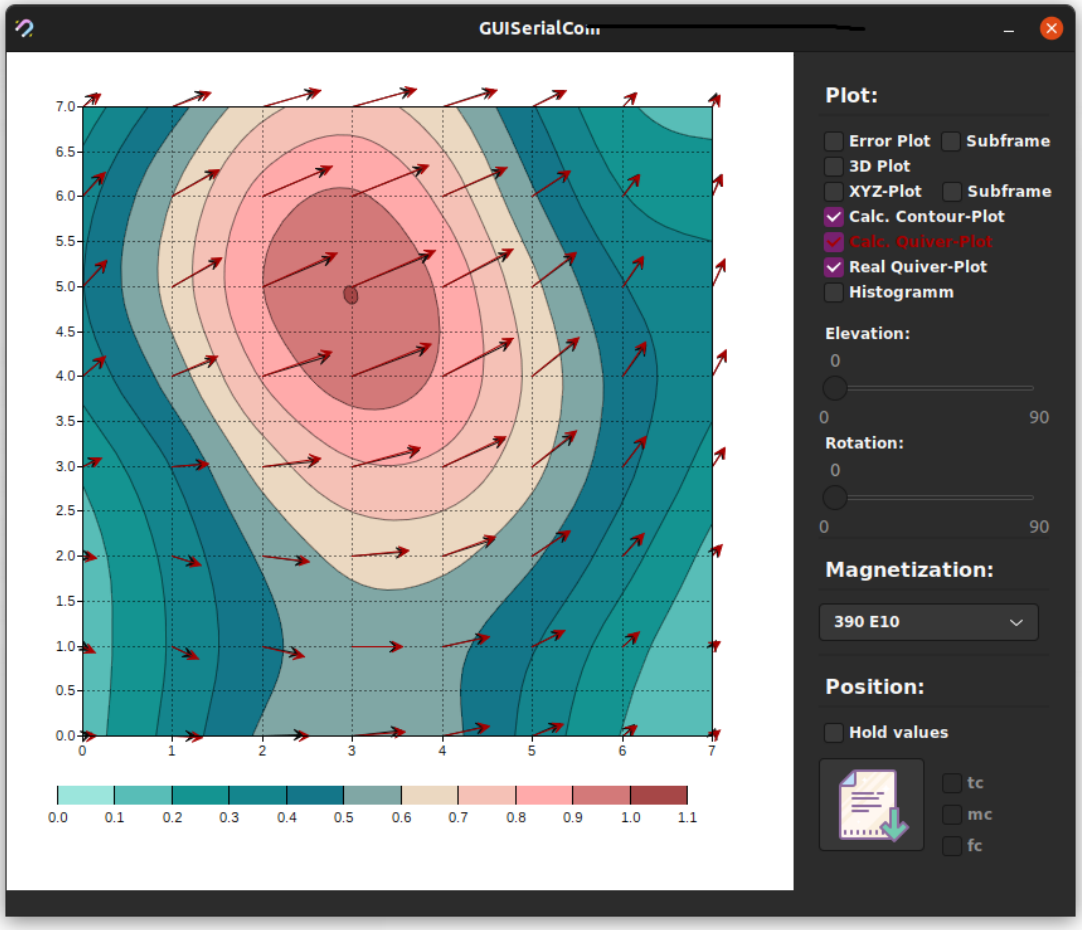


Abbildung 5.34: Unterfenster der Anwendung unter Linux mit verschiedenen vorgenommenen Einstellungen.

Die ursprüngliche Textanzeige der Position, die sich unten rechts befindet (vgl. Abschnitt 4.4.3 Abbildung 4.13 (5)), arbeitet unter Linux leider nicht wie erhofft und verursacht einen Fehler, weshalb diese entfernt wurde. Die Widgets, die wiederum über das Unterfenster gestartet werden können, funktionieren auch wie gewünscht und zeigen die Lernkurve beziehungsweise den XYZ-Plot in einem neuen Unterfenster an.
Bei längerer Verwendung der Software unter Linux ist leider wiederholt der Fehler aufgetreten, dass der aktuelle Plot nicht mehr in jedem Prozessdurchlauf aktualisiert wird. Häufig fror die Darstellung nach einiger Zeit ein und wurde erst nach Beendigung des aktuellen Threads durch Betätigung des Stopp-Buttons aktualisiert. Die genaue Ursache konnte bisher nicht ausgemacht werden, aber es liegt die Vermutung nahe, dass das wiederholte Erzeugen der Chart-Objekte (beispielsweise XYChart) in jedem Durch-

lauf Probleme verursacht. Gegebenenfalls könnte es also von Vorteil sein, die benötigten Chart-Objekte einmalig global zu erstellen.
Somit konnten zwar alle Funktionalitäten, bis auf eine, der Anwendung unter Linux nachgewiesen werden, jedoch treten noch Komplikationen in der Darstellung bei Langzeitnutzungen auf.

5.1.6 Ergänzende Tests zur Ermittlung der ungefähren Grenzen des Suchalgorithmus

Die bisherigen Tests zur Reliabilität des Algorithmus verschaffen einen groben Überblick, wo sich seine Grenzen befinden und über den Bereich, in welchem sinnvolle Resultate erzielt werden können. Um diesen Bereich etwas detaillierter darzustellen, werden weitere Tests durchgeführt. Dabei wird der Gebermagnet wieder über dem Array an verschiedenen Orten positioniert. Für diese Messungen wird der Magnet jedoch immer nur in eine Richtung (x, y oder z) verschoben und anschließend die Standardabweichung der einzelnen Koordinaten und die Gesamtstandardabweichung festgehalten. Im Anschluss lassen sich so mit Hilfe dieser ermittelten Werte Grafiken erstellen, die einen genaueren Aufschluss über den nutzbaren Bereich des Suchverfahrens geben sollen. Da dieser Abschnitt nur von dem Gebiet handelt, in dem der implementierte Algorithmus verwertbare Resultate liefert, befinden sich die dazu durchgeführten Messungen im Anhang.

Verschiebung des Gebermagneten in x-Richtung

Für diese Tests wird der Magnet in einer Höhe von 47 mm platziert und auf der x-Achse immer um 5 mm verschoben, bis das Ende des Sensor-Arrays erreicht ist. Somit ergeben sich sechs Messungen, die es durchzuführen gilt und anschließend grafisch festgehalten werden. In Abbildung 5.35 ist die Standardabweichung im Bezug zur Entfernung des Magneten zum Array dargestellt. Im gesamten Bereich des Arrays liegt die Standardabweichung jeder ermittelten Koordinate unter einem Wert von $\sigma = 1\,\text{mm}$. Wie vermutet steigt diese an, je weiter der Magnet vom Array entfernt wird. Auffällig hierbei ist, dass die Höhe eine ein wenig größere Abweichung besitzt als die anderen beiden Kenngrößen.

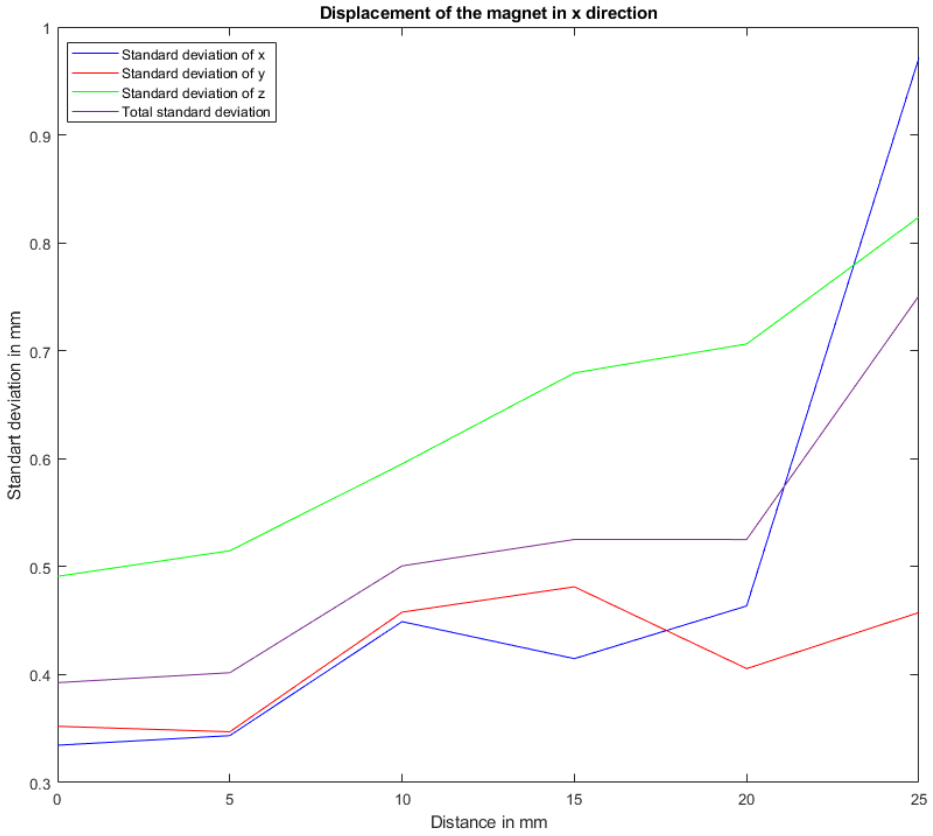


Abbildung 5.35: Standardabweichung im Bezug zur Entfernung des Magneten zum Array bei Verschiebung in x-Richtung.

Verschiebung des Gebermagneten in y-Richtung

Der Magnet wird wieder in der selben Höhe platziert, allerdings wird der Permanentmagnet in diesem Abschnitt in y-Richtung verschoben. In Abbildung 5.36 ist wieder die Standardabweichung im Bezug zur Entfernung des Magneten zum Array dargestellt. Auch hier liegt die Standardabweichung jeder ermittelten Koordinate unter einem Wert von $\sigma = 1\,\text{mm}$ und besitzt im Vergleich zur vorherigen Messung einen weniger steilen Anstieg.

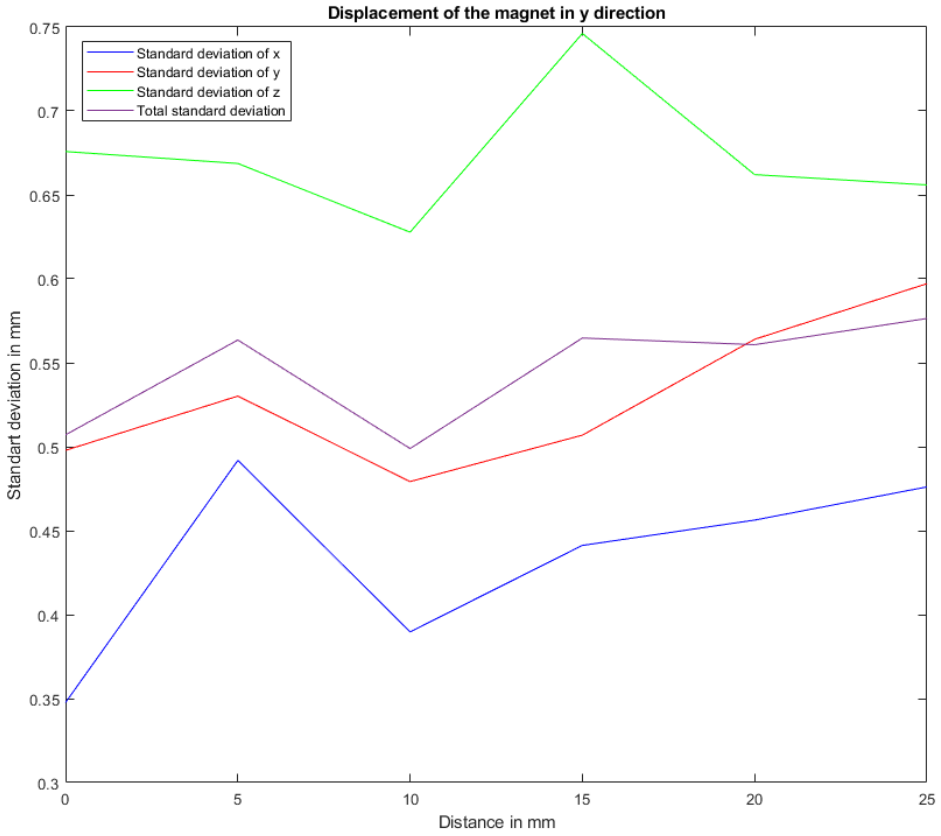


Abbildung 5.36: Standardabweichung im Bezug zur Entfernung des Magneten zum Array bei Verschiebung in y-Richtung.

Verschiebung des Gebermagneten in z-Richtung

In dieser Messreihe wird der Magnet immer in der Mitte des Arrays, dafür jedoch in verschiedenen Höhen positioniert. In Abbildung 5.37 ist deutlich erkennbar, wo sich die ausschlaggebende Grenze des Suchalgorithmus befindet. In den beiden Messungen zuvor konnte die Standardabweichung über dem gesamten Array (in einer Höhe von 47 mm) unter $\sigma \leq 1\,\text{mm}$ gehalten werden. In diesem Fall trifft das nur bis zu einer Entfernung von ungefähr 57 mm zu. Danach steigt die Abweichung zwischen tatsächlicher und ermittelter Position exponentiell an. Allerdings kann nicht daraus gefolgert werden, dass die Grenze bei 57 mm liegt, da der nächste Test, aufgrund des nicht idealen Messaufbaus, erst in

einer Höhe von 67 mm erfolgt ist. Aber es lässt sich schlussfolgern, dass sich die Grenze zwischen diesen beiden Werten befinden muss.

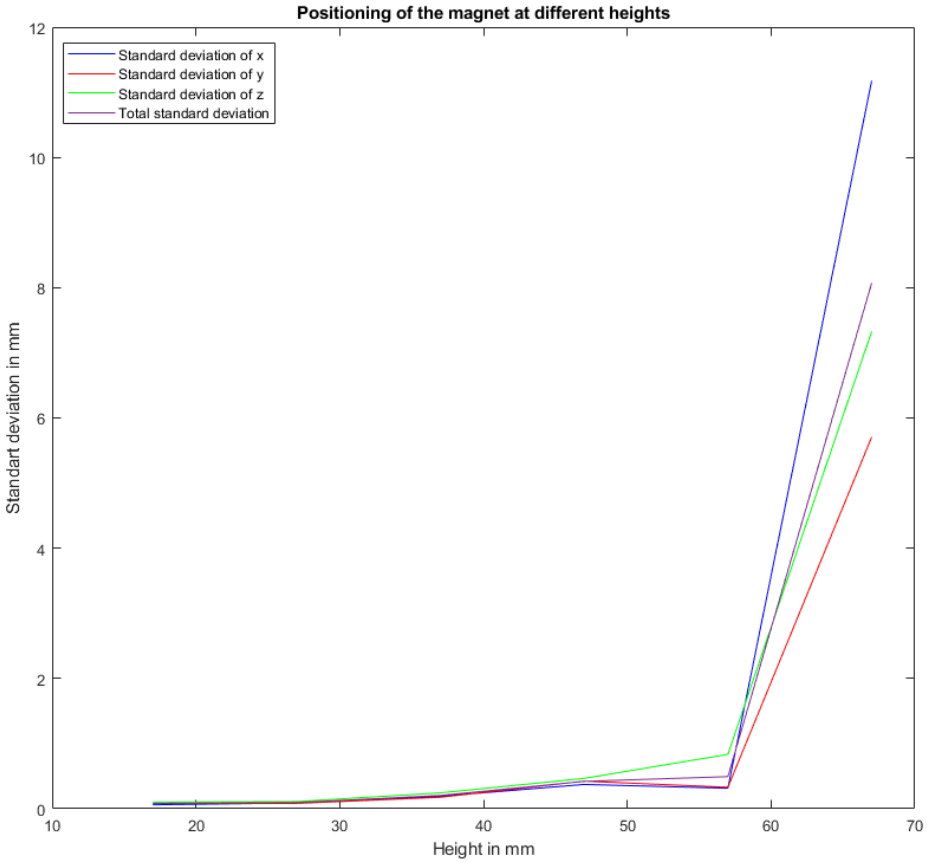


Abbildung 5.37: Standardabweichung im Bezug zur Entfernung des Magneten zum Array bei Verschiebung in z-Richtung.

Überblick über die Grenzen des Algorithmus und Erläuterung ihrer Ursachen

Um den ermittelten nutzbaren Bereich noch einmal zu versinnbildlichen, wurde mit Hilfe der errechneten Gesamtstandardabweichungen jeder Messung eine Grafik angefertigt. Diese soll einen ungefähren Überblick geben, an welchen Orten über dem Array verwertbare Ergebnisse erzielt werden können. Es ist allerdings anzumerken, dass die Messungen

in x- und y-Richtung immer auf derselben Höhe durchgeführt wurden und die in z-Richtung immer in der Mitte des Arrays. Die Gesamtstandardabweichung jedes Tests wurde als Grenze ihrer jeweiligen Richtung aufgefasst und diese anschließend zu einem Quader verbunden. Somit kann es natürlich vorkommen, dass Werte über $\sigma = 1\,\mathrm{mm}$ an Stellen auftreten, wo keine Messungen durchgeführt worden sind. Diese Grafik soll aber auch nur eine Übersicht über die noch wahrscheinlich sinnvoll erzielbaren Resultate verschaffen. Andernfalls hätten Messungen für jede mögliche Position des Magneten über dem Array durchgeführt werden müssen, wofür sich ein Robotermessplatz anbieten würde.

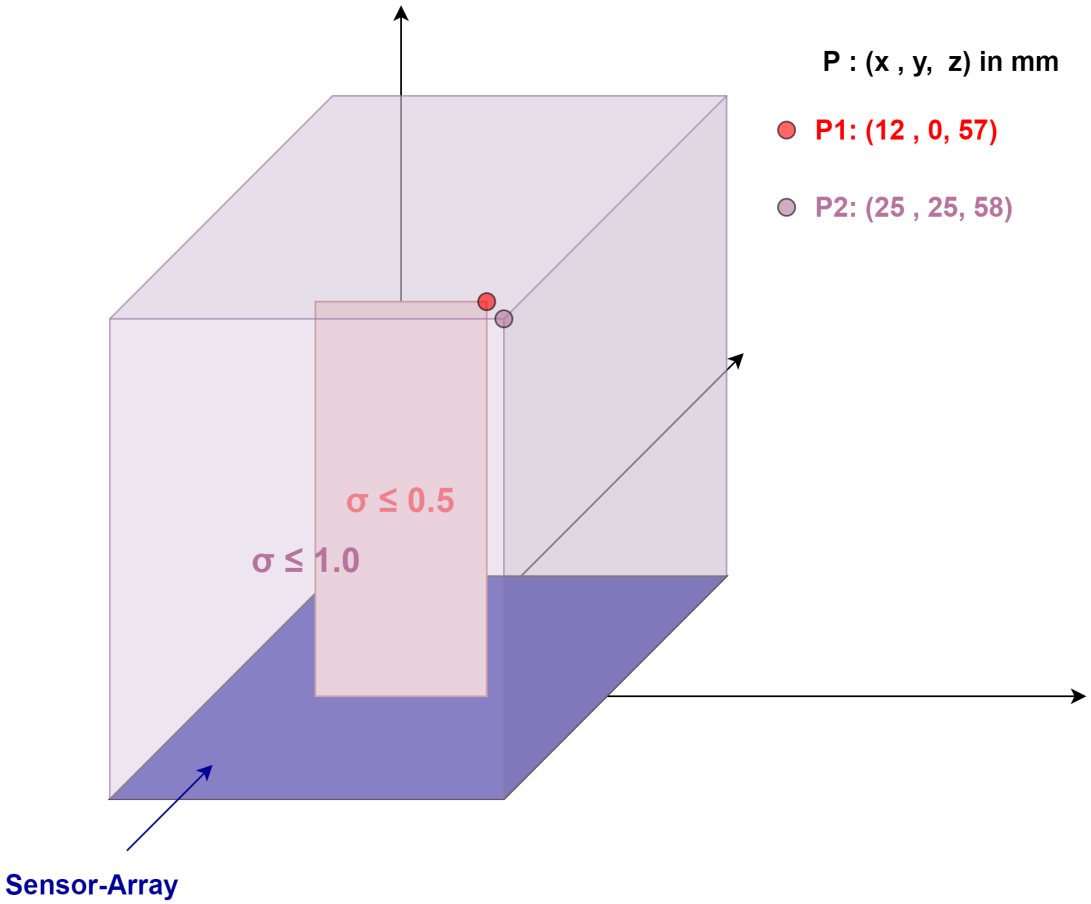


Abbildung 5.38: Grobe Übersicht über die Grenzen des Suchalgorithmus.

Es wurde im Abschnitt zuvor bereits angesprochen, dass die Höhe die ausschlaggebende Größe für die Genauigkeit des Suchalgorithmus ist. Das hat verschiedene Gründe, die in den folgenden Punkten erläutert werden sollen:

Numerische Ungenauigkeiten Wie bereits im Grundlagenteil erläutert wurde, nimmt die magnetische Feldstärke die von den Sensoren gemessen wird, mit einer Größenordnung von r^3 ab, je weiter der Gebermagnet vom Array entfernt wird. Das ist auch der Grund dafür, warum die Höhe die maßgebende Größe für die Genauigkeit des Suchalgorithmus ist und ab einer gewissen Entfernung keine verwertbaren Resultate mehr erzielt werden können. Des Weiteren steigt mit der Höhe auch die Signifikanz von numerischen Ungenauigkeiten, welche immer dann auftreten, wenn die zur Verfügung stehenden Bits des Analog-Digital-Konverters nicht ausreichen um einen gemessenen Wert exakt darzustellen. Der Grund dafür ist, dass nicht mehr die gesamte Anzahl der zur Verfügung stehenden Bits genutzt wird, um kleine Zahlen darzustellen. Und da mit Hilfe eines AD-Wandlers Spannungen nur in bestimmten Abstufungen realisiert werden können, auch Quantisierungsintervalle genannt, reichen diese nicht mehr aus um kleine Werte genau nachzubilden. Je kleiner also der gemessene Ausgangswert eines Sensor ist, desto signifikanter werden die Rundungsfehler, die gemacht werden müssen, um die Zahl einer Abstufung zuzuordnen. Die Differenz zwischen tatsächlichem und vom ADC dargestellten Wert wird auch Quantisierungsfehler genannt.

Rauschen Mit größerer Entfernung des Magneten zum Sensor-Array nimmt auch die Signifikanz des Rauschens zu. Je weiter dieser vom Array positioniert wird, desto mehr gelangen die Sensorausgangswerte in den Wertebereich des Rauschens, weshalb der Einfluss auf die erzielten Resultate zunimmt. Das Erdmagnetfeld oder ähnliche Störeinflüsse überlagern somit die eigentlich gemessene magnetische Feldstärke der TMR-Sensoren. Und da diese mit der Größenordnung r^3 abnimmt, steigt die Signifikanz des Rauschens in einem ähnlichen Verhältnis.

Um den Einfluss der Entfernung zu verdeutlichen, sind die Sensorausgangswerte des Arrays nochmal in zwei verschiedenen Höhenlagen in Matrixschreibweise festgehalten:

$$cos_{17mm} = \begin{pmatrix} -9.1 & 6.3 & 42.2 & 78.3 & 83.5 & 51.360 & 18.2 & -8.1 \\ -29.3 & -11.8 & 53.0 & 137.0 & 144.8 & 73.4 & 0.9 & -25.1 \\ -57.4 & -48.1 & 40.2 & 229.5 & 254.7 & 193.0 & -32.4 & -55.1 \\ -84.3 & -95.7 & 0.6 & 332.9 & 389.7 & 214.8 & 59.0 & -82.5 \\ -85.7 & -100.5 & -2.2 & 355.1 & 392.8 & 225.2 & -66.9 & -87.7 \\ -58.1 & -52.1 & 43.0 & 267.5 & 283.4 & 181.1 & -30.1 & -62.3 \\ -25.6 & -4.7 & 67.9 & 165.0 & 165.0 & 66.5 & -6.3 & -29.1 \\ -7.1 & 13.8 & 50.8 & 94.8 & 92.9 & 53.1 & 16.8 & -8.5 \end{pmatrix}$$

$$sin_{17mm} = \begin{pmatrix} -42.2 & -49.1 & -54.5 & -28.7 & 17.8 & 54.1 & 56.7 & 47.9 \\ -49.7 & -80.0 & -96.1 & -56.0 & 38.1 & 94.2 & 86.7 & 63.3 \\ -47.8 & -90.0 & -138.8 & -94.3 & 70.1 & 240.0 & 102.9 & 60.6 \\ -20.3 & -45.1 & -93.4 & -59.7 & 71.1 & 217.0 & 15.2 & 33.5 \\ 24.8 & 44.7 & 77.5 & 67.6 & -9.7 & -182.1 & -31.9 & -13.2 \\ 53.9 & 97.2 & 159.0 & 117.4 & -60.9 & -254.7 & -118.7 & -46.1 \\ 57.7 & 91.6 & 111.0 & 66.2 & -47.5 & -102.4 & -82.2 & -53.5 \\ 49.1 & 61.0 & 64.9 & 32.8 & -18.6 & -55.9 & -57.5 & -43.2 \end{pmatrix}$$

$$cos_{57mm} = \begin{pmatrix} 2.0 & 5.5 & 8.9 & 12.4 & 12.3 & 8.3 & 3.76 & 0.9 \\ 3.4 & 5.7 & 10.8 & 13.7 & 12.5 & 7.3 & 5.1 & 3.3 \\ 0.4 & 5.8 & 9.1 & 12.3 & 11.5 & 8.6 & 5.7 & 2.3 \\ -0.4 & 4.9 & 10.1 & 9.0 & 8.8 & 10.3 & 8.6 & 3.4 \\ -1.1 & 3.3 & 5.4 & 8.0 & 10.3 & 9.2 & 12.1 & 8.1 \\ -1.6 & 1.4 & 4.4 & 7.0 & 7.4 & 9.0 & 9.9 & 8.2 \\ -3.9 & -0.4 & 1.3 & 4.0 & 9.3 & 9.3 & 9.6 & 9.5 \\ -4.2 & -2.5 & -1.3 & 0.8 & 9.3 & 12.3 & 11.1 & 7.7 \end{pmatrix}$$

$$sin_{57mm} = \begin{pmatrix} -2.3 & -2.5 & -1.8 & -0.3 & 3.4 & 4.4 & 2.4 & 1.0 \\ -4.2 & -4.6 & -0.0 & 1.6 & 3.0 & 3.3 & 2.3 & 0.5 \\ -4.1 & -4.1 & -2.4 & -2.4 & 0.6 & -0.2 & -1.6 & -2.5 \\ -3.0 & -2.4 & -2.7 & -0.0 & -2.2 & -3.9 & -2.9 & -4.0 \\ -2.2 & -0.0 & -1.1 & 0.1 & -0.4 & -1.6 & -5.3 & -3.6 \\ 0.2 & 1.5 & 0.9 & -2.7 & -0.2 & -2.7 & -4.5 & -4.0 \\ 2.9 & 2.4 & 2.0 & 0.2 & 1.1 & -2.6 & -2.4 & -3.7 \\ 5.5 & 6.4 & 3.5 & 8.1 & 7.1 & 2.6 & -0.1 & -1.9 \end{pmatrix}$$

Die gemessenen Ausgangswerte bei einer Höhe des Magneten von 17 mm haben im Gegensatz zu einer Höhe von 57 mm deutlich abgenommen. Dabei ist in den vorherigen Abschnitten 57 mm als ungefähre Grenze für die Höhe ermittelt worden und somit spiegeln sich in den entsprechenden Matrizen die Werte wieder, die gerade noch als verwertbare Resultate gewertet werden können. Wird der Gebermagnet noch weiter vom Array entfernt, verkleinern sich auch die gemessenen Größen für die magnetischen Feldstärken nochmal deutlich und es wird klar, dass die Höhenlage den wichtigsten Einflussfaktor

für die Reliabilität des Algorithmus darstellt. Aber auch andere Faktoren besitzen einen Einfluss auf die Genauigkeit des Suchalgorithmus, die nicht direkt mit der Entfernung des Permanentmagneten zum Sensor-Array zu tun haben:

Startposition der Suche Die im Programmcode angegebene Startposition (0, 0, -40) besitzt auch Einfluss auf die Reliabilität des Algorithmus, wenn auch nicht im selben Maß, wie die beiden zuvor genannten Punkte. Je weiter der Permanentmagnet also vom Array entfernt wird, desto weiter weg befindet sich auch die eigentliche Position von der angegebenen Startposition des Magneten, was die Anzahl der benötigten Suchschritte für die Ermittlung der tatsächlichen Position zwangsläufig erhöht. Gegebenenfalls reicht die eingestellte Anzahl an Suchdurchläufen nicht aus, um den Ort des Magneten präzise zu bestimmen.

Lokale Minima Im dritten Kapitel **Algorithmusentwicklung** wurde bereits angesprochen, dass das Gradientenabstiegsverfahren Probleme aufweist, wenn zwischen einem lokalen und dem globalen Minimum unterschieden werden soll. Das bedeutet konkret, dass je nach eingestellter Schrittweite und Startposition zwar ein Minimum aufgespürt werden kann, jedoch kann es sich hierbei um ein lokales statt dem globalen handeln, aus welchem der Algorithmus nicht mehr hinausfindet. Dies würde sich auf die errechnete Position der Suche auswirken und die Resultate negativ beeinflussen.

Fehlermaß Das verwendete Fehlervergleichsmaß (**Mittlere quadratische Abweichung**) ist möglicherweise nicht immer ideal. Es ist anfällig für starke Ausreißer in den Messdaten und wirkt damit verfälschend auf die Ergebnisse. Also hängt die Verlässlichkeit des Fehlermaßes indirekt auch von der Höhe und den anderen aufgeführten Ungenauigkeiten ab, da je größer ein möglicher Ausreißer ist, der durch diese Ungenauigkeiten verursacht wurde, desto größer ist der Einfluss auf die sich am Ende der Suche ergebenden Resultate.

Modellierung des Magneten Im Grundlagenteil wurde erläutert, dass der Gebermagnet im Suchalgorithmus als Dipol angenommen wird. Diese Vereinfachung stellt an sich bereits eine Ungenauigkeit im Algorithmus dar und wurde getroffen, um die Implementation zu vereinfachen und den Rechenaufwand zu minimieren. Denn je nach verwendetem Magneten, seiner Größe und Beschaffenheit, stimmt diese Vereinfachung mal mehr und

mal weniger überein. Wird beispielsweise ein besonders großer Magnet sehr nah am Array verwendet, stimmen die Feldverläufe von diesem nicht mehr ansatzweise so gut mit denen eines Dipols überein, wie beispielsweise bei Verwendung eines kleineren Magneten, welcher sich weiter weg vom Sensor-Array befindet.

Empirisch ermittelte Magnetisierungsstärke des Magneten Unter Umständen ist die genaue Magnetisierungsstärke eines zu untersuchenden Magneten nicht immer exakt bekannt. In der implementierten Applikation besteht die Möglichkeit diese händisch über eine Auswahlliste einzustellen, oder automatisch ermitteln zu lassen, indem diese Kenngröße als Lernparameter mit in den Algorithmus aufgenommen wird. Dabei ist nicht garantiert, dass weder die händisch eingestellte noch die vom Algorithmus ermittelte Magnetisierungsstärke mit der tatsächlichen des Magneten übereinstimmen. Dies kann sich negativ auf die gewonnenen Resultate auswirken und somit verfälschend auf die ermittelte Position.

6 Schlussfolgerungen

Im letzten Kapitel dieser Arbeit soll nochmals aufgeführt werden, welche Ziele es zu erreichen galt und welche auch erreicht werden konnten. Die noch ungeklärten Eigenschaften des Systems werden erläutert und es soll ein Ausblick darauf gegeben werden, wie anschließende Arbeiten auf dieser aufgebaut werden könnten.

6.1 Zusammenfassung

Das Hauptziel dieser Arbeit war das Implementieren einer Applikation, welche in der Lage ist, die Position eines Permanentmagneten über einem Array aus magnetischen Sensoren zu bestimmen. Grundlage dieser Arbeit war einerseits das aus Vorarbeiten bestehende Sensor-Array [16] und andererseits ein Beispiel für ein Suchverfahren, welches in Matlab implementiert wurde.

Dafür galt es folgende Punkte zu bewältigen:

1. Einarbeitung Um zu verstehen, wie eine solche Software umgesetzt werden kann, müssen Recherchearbeiten geleistet werden, die im Grundlagenteil nachzulesen sind. Dazu gehören unter anderem Vorkenntnisse zu den eingesetzten TMR-Sensoren und Wissen zu Magnetfeldberechnungen.

2. Umsetzung der grafischen Oberfläche Zu einer solchen Anwendung gehört natürlich auch eine grafische Oberfläche, mit deren Hilfe der Anwender mit der Software interagieren kann. Es galt dafür ein ansprechendes Layout zu entwerfen, auf dem sich Steuerungsmöglichkeiten wie Buttons, Kontrollkästchen und Drop-Down-Menüs befinden. Sinnvoll für Messungen ist, diese festhalten zu können, wie durch das Speichern auf dem verwendeten Computer.

3. Umsetzung der Signalverarbeitung Um die gemessenen Signale des Arrays zum Beispiel von Rauschen befreien zu können, sollten Signalverarbeitungsmethoden implementiert werden, wie die Fouriertransformation oder die Offsetkompensation. Dazu zählt aber auch der entwickelte Suchalgorithmus und seine Funktionsweise sowie die dazugehörige Fehlerberechnung.

4. Test und Optimierung Um die Reliabilität des entworfenen Algorithmus zu überprüfen, sollten Messungen durchgeführt werden, welche aufzeigen, wo sich dem Anschein nach die Grenzen des Suchverfahrens befinden.

5. Versuchsbetrieb und Auswertung Es galt natürlich nicht nur den Suchalgorithmus zu testen, sondern auch die dazugehörige grafische Benutzeroberfläche. Dafür sollten im Anschluss Messungen für verschiedene Positionen und Lagen des verwendeten Permanentmagneten durchgeführt und bewertet werden. Auch eine plattformübergreifende Funktionalität galt es zu beweisen.

6. Zusammenfassung und Ausblick Schlussendlich sollten die gewonnenen Ergebnisse und Kenntnisse präsentiert werden, offene Punkte diskutiert und Ansätze zur Weiterführung aufgezeigt werden, was genau die Thematik dieses abschließenden Kapitels darstellt.

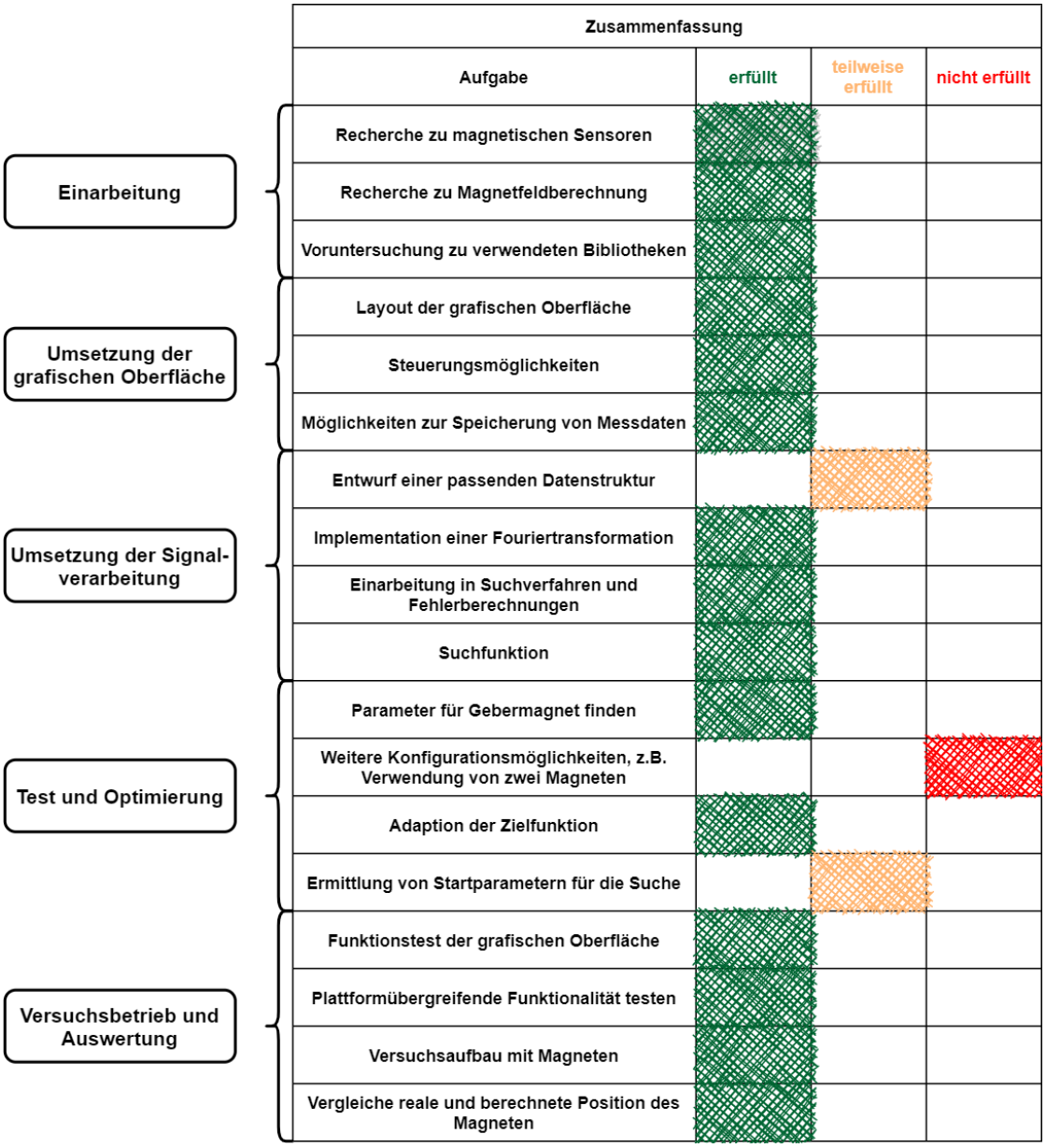


Abbildung 6.1: Übersicht zu den gestellten Aufgaben und ob diese erfüllt werden konnten.

In Abbildung 6.1 sind die zuvor angesprochen Punkte und dazugehörige Aufgaben festgehalten. Darüber hinaus wird gezeigt, welche Aufgaben erfüllt, teilweise erfüllt oder nicht erfüllt werden konnten. Anschließend wird im Kapitel **Offene Punkte** bei auffälligen Punkten diskutiert, wo die Schwächen oder Herausforderungen saßen und es wird unter anderem ausgeführt, wie man diese möglicherweise beheben kann.

6.2 Offene Punkte

Einige der im vorherigen Abschnitt aufgeführten Aufgaben konnten nur teilweise oder gar nicht erfüllt werden. Der Entwurf einer passenden Datenstruktur für die Sensorausgangswerte wurde nicht weiter verfolgt, da diese in zwei separaten Arrays gespeichert werden. Unter C++ besteht die Möglichkeit über `struct` eine neue Struktur anzufertigen, die beide Arrays übergeordnet zusammenfassen würde. Da dieser Punkt niedriger priorisiert wurde als andere und die Lösung die Implementation nicht grundlegend vereinfachen würde, blieb diese Aufgabe nur teilweise erfüllt. Aus Zeitgründen wurde zunächst auf weitere Konfigurationsmöglichkeiten, wie zum Beispiel die Verwendung von zwei Magneten, verzichtet. Die Ermittlung der Startparameter für die Suche funktioniert für die Ermittlung der Parameter des Gebermagneten, jedoch ist die Startposition des Magneten immer in der Mitte des Arrays in einer Höhe von 4 cm angegeben.

6.3 Ausblick

Nicht nur die Punkte, die bei der Ausarbeitung offen geblieben sind, bieten Raum für Verbesserung. Auch an anderen Stellen können Sensor-Array und die implementierte Anwendung weiterentwickelt werden und die nachfolgenden Themen sollen Anregungen geben, um auf dieser Arbeit aufzubauen und das vorgestellte Konzept weiterzuführen:

6.3.1 Ansätze für die äußeren Umstände

Messaufbau zur Auswertung des Algorithmus Durch den nicht idealen Messaufbau waren in den Messungen von Beginn an Ungenauigkeiten zu erwarten, weshalb eine detailliertere Untersuchung des Suchalgorithmus beziehungsweise der Software nicht möglich war. Die Verwendung eines Robotermessplatzes, der den Magneten präzise an jede mögliche Position setzen könnte, würde genaueren Aufschluss über die von der Software ermittelten Resultate liefern.

Anforderungen an die Umgebung Die Umgebung, in welchem das Sensor-Array verwendet wird, hat auch Auswirkungen auf die vom Algorithmus ermittelten Resultate. Empfehlenswert ist es dabei eine Umgebung zu wählen, welche möglichst frei von Störquellen ist, um das Rauschen, welches die Ausgangswerte der Sensoren beeinflusst, zu

minimieren. Selbstverständlich ist dies nicht für jedes Anwendungsgebiet möglich und es können nur schwer alle Störeinflüsse, wie zum Beispiels das Erdmagnetfeld, entfernt werden. Es sollte jedoch zumindest darauf geachtet werden, mögliche Störeinflüsse aus der unmittelbaren Nähe des Arrays zu entfernen.

6.3.2 Ansätze für das Sensor-Array

Verwendete TMR-Sensoren Mittlerweile stehen auch genauere TMR-Sensoren zur Verfügung als die, die auf diesem Array verbaut sind. Das Entwerfen eines solchen Arrays mit diesen Sensoren hätte vermutlich auch eine Steigerung der Reliabilität und Präzision der Anwendung zur Folge, da diese die magnetische Feldstärke genauer erfassen.

Remanenz Auf Grund von Restmagnetisierung, welche TMR-Sensoren in der freien Schicht besitzen können, entstehen Messungenauigkeiten. Diese Remanenz entsteht beispielsweise, wenn vor einer Messung bereits ein Magnetfeld an das Sensor-Array angelegt wurde. Dabei ist irrelevant, ob es sich dabei um das Feld eines Gebermagneten oder das, welches möglicherweise durch Rauschen verursacht wird, handelt. Um dem entgegenzuwirken, und somit die Messergebnisse zu verbessern, könnten Techniken genutzt werden, welche diese Restmagnetisierung aufheben. Beispielweise werden bei AMR-Sensor-Arrays bereits zwei Spulen dafür eingesetzt, dass der Arbeitspunkt der verwendeten Sensoren in den linearen Bereich verschoben wird. Mehr zu diesem Thema kann in „Parasitic Capacitances, Inductive Coupling, and High-Frequency Behavior of AMR Sensors“ von Neoclis G. Hadjigeorgiou und Paul P. Sotiriadis [10] nachgelesen werden.

Quantisierungsfehler In dem Kapitel **Evaluation** wurde im Abschnitt **Überblick über die Grenzen des Algorithmus und Erläuterung ihrer Ursachen** bereits angesprochen, dass bei steigendem Abstand auch numerische Ungenauigkeiten zunehmen. Dem zu Grunde liegt vor allem die 12 Bit Auflösung des Analog-Digital-Konverters. Werden die gemessenen Sensorausgangswerte zu klein, reicht diese aufgrund zu weniger Quantisierungsintervalle nicht mehr aus, um die Zahlen exakt nachzubilden. Zwei mögliche Lösungsansätze wären entweder die Nutzung eines ADCs mit höherer Auflösung oder die Verwendung eines Vorverstärkers für die gemessenen Sensorausgangswerte, damit wieder die vollständige Anzahl der Bits ausgenutzt wird, um die Zahlen darzustellen.

6.3.3 Ansätze für die Darstellung

Grafische Benutzeroberfläche Die grafische Benutzeroberfläche kann natürlich auch noch angepasst und um Funktionen erweitert werden, die den Bedürfnissen des jeweiligen Anwendungsbereiches dadurch besser gerecht werden würde. Somit könnten zusätzliche Bedienelemente mit neuen Funktionen implementiert werden, wie mehr Plotdarstellungen oder Speichermöglichkeiten für erhaltende Messergebnisse. Das Entfernen von Funktionalitäten oder Algorithmen ist genauso möglich, um beispielsweise mehr Übersicht zu schaffen oder auch die Verarbeitungsgeschwindigkeit zu erhöhen.

Plotdarstellung Es wurde zuvor bereits angeschnitten, dass Plotdarstellungen angepasst werden könnten, um beispielsweise die Resultate dem Anwendungsgebiet entsprechend besser darzustellen. Eine weitere Idee ist das Ergänzen einer Hold-Funktion, welche mehrere erstellte Grafiken übereinander plottet. So könnten mehrere Magnetpositionen in einer Darstellung miteinander verglichen werden; anbieten würde sich dies vor allem für den XYZ-Plot. Für den Error-Plot wurde dies schon einmal in einfacher Ausführung versucht. ChartDirector stellt außerdem die Möglichkeit zu Verfügung, Echtzeitgrafiken zu erstellen, bei welchen die Anzeigegeschwindigkeit eingestellt werden kann. Unter Umständen würde die Implementation dieser Darstellungsart eine Verbesserung für die Untersuchung der ermittelten Ergebnisse ermöglichen.

Fenstergröße anpassbar gestalten Die Fenstergrößen der entwickelten Applikation sind nicht veränderbar, weshalb zum Beispiel kein Vollbildmodus zur Verfügung steht. Um die Grafiken näher zu untersuchen, wäre es gegebenenfalls von Vorteil, wenn diese Funktionalität noch ergänzt werden würde.

6.3.4 Ansätze für die Struktur

Softwarestruktur Eine Überarbeitung der genutzten Softwarestruktur könnte Reliabilität und Präzision der Suche positiv beeinflussen. Bisher werden die Funktionen zum Auslesen des Arrays und für die Positionsbestimmung in einem Thread aufeinanderfolgend ausgeführt. Zuerst werden die Sensorausgangswerte ausgelesen und anschließend wird die Suche für diese Werte gestartet. Das bedeutet, dass erst wenn die erste Suche abgeschlossen wurde, die nächste gestartet werden kann. Durch Nutzung mehrerer

Threads könnten die beiden Funktionen parallelisiert und somit die Prozessgeschwindigkeit erhöht werden. Es könnten auch mehrere Suchdurchläufe auf der Basis von den selben Ausgangswerten gestartet und miteinander verglichen werden. Anschließend würden die besten Resultate als Endergebnis gewertet werden und mehr Aufschluss über die tatsächliche Position des Magneten geben. Dabei muss auf das **Erzeuger-Verbraucher-Problem** geachtet werden, welches die Problemstellung bei Prozesssynchronisation darstellt. Dabei würde das Auslesen des Arrays dem Erzeuger von Daten und die Suche den Verbraucher, welcher diese Daten verarbeiten soll, entsprechen. Durch die Verwendung mehrerer Threads, welche alle auf dieselben Speicher zugreifen, entsteht das Problem der Zugriffsreihenfolge, welches es zu regeln gilt. Die Verwendung von Semaphoren oder Mutexen sind mögliche Ansätze um das Erzeuger-Verbraucher-Problem zu lösen.

6.3.5 Ansätze für den Suchalgorithmus

Startposition der Suche In dieser entworfenen Software wurde als Startposition für die Suche immer derselbe Ort (0, 0, -40) verwendet. Das Implementieren einer Auswahl von verschiedenen Position, an welchen der Algorithmus zu suchen beginnt, könnte eine Verbesserung der am Ende ermittelten Ergebnisse zur Folge haben. Je nachdem, wie weit der Magnet nämlich von dieser Position entfernt ist, könnte das Einfluss auf die benötigte Anzahl der Suchschritte besitzen, welche gegebenenfalls nicht mehr ausreichen um ein zufriedenstellendes Ergebnis zu erzielen. Um dieses Problem zu lösen, könnte eine Auswahl von verschiedenen Anfangspositionen eingebettet werden, aus denen der Anwender, sofern eine Vermutung über den ungefähren Ort des Magneten vorliegt, auswählen kann. Auch mehrere Suchdurchläufe, welche alle unterschiedliche Anfangspositionen nutzen, ist ein möglicher Ansatz, der verfolgt werden könnte.

Modellierung des Magneten Es wurde bereits mehrfach angesprochen, dass der Gebermagnet in dieser Arbeit als Dipol angenommen wird, um die Berechnung der magnetischen Feldstärken zu vereinfachen. Das beeinflusst, je nach verwendetem Magneten und dessen Entfernung, die erzielten Resultate natürlich in unterschiedlichem Ausmaß. Im Kapitel **Grundlagen** wurde bereits das Verfahren der Superposition vorgestellt, mit denen es möglich ist, die Verläufe des Magnetfeldes präziser zu bestimmen. Dabei steigt der benötigte Rechenaufwand, je nachdem wie viele Dipole verwendet werden, um den Permanenmagneten nachzumodellieren. Eine andere Möglichkeit wäre die Verwendung der entsprechenden Formel zur Bestimmung der Feldstärken eines Magneten. So könnte bei

der Verwendung eines Hohlzylindermagneten beispielsweise dessen zugehörige Gleichung als Funktion in der Anwendung implementiert werden. Eine Auswahl für verschiedene Magnettypen würde sich anbieten.

Verwendete Parameter im Suchalgorithmus Momentan werden sechs beziehungsweise sieben Parameter in der Suche verwendet, wenn die Magnetisierungsstärke als Parameter im Lernalgorithmus mit aufgenommen wird. Durch eine allgemeine Verbesserung des Suchverfahrens könnte die Rotation des Magneten unter Umständen vernachlässigt und die Rotationsmatrizen aus dem Programmcode entfernt werden. Dadurch würde sich die Anzahl der nötigen Kenngrößen auf drei beziehungsweise vier verringern.

Fehlermaß Bisher wurde nur eine Methode zur Bestimmung des Fehlers zwischen realen und errechneten Sensorausgangswerten verwendet. Diese ist anfällig für starke Ausreißer in den Messdaten und unter Umständen somit nicht immer am besten geeignet. Das Untersuchen des Algorithmus mit anderen Fehlervergleichsmethoden bietet sich an, um zu überprüfen, ob diese vielleicht eine Verbesserung der Genauigkeit mit sich bringen.

6.4 Mögliche Anwendungsgebiete

Um diese Arbeit abzuschließen sollen noch einmal mögliche Anwendungsgebiete aufgeführt werden, in welchen TMR-Sensor-Arrays bereits Anwendung finden. Hierbei soll es sich aber nur um einen kleinen Einblick aus einer Vielzahl von möglichen Anwendungsgebieten handeln.

Magnetic Marker Monitoring In der heutigen Medizin werden operative Eingriffe oft von Computern unterstützt und chirurgische Instrumente navigieren zum Teil mit Hilfe eines Ortungssystemes [5]. Durch das Einbringen eines Magneten können interne Strukturen des menschlichen Körpers über ein MR-Sensor-Array aufgezeichnet und untersucht werden. Dafür werden die Daten des Magnetfeldes gemessen und mit simulierten Werten für eine Feldverteilung verglichen.

Winkelmessung In moderner Technologie ist es Stand der Technik, magnetische Sensoren für Winkelmessungen zu nutzen, wie zum Beispiel im Bereich der Automobilindustrie. Dort werden diese beispielsweise eingesetzt, um den Rotationswinkel eines Motors oder den eines Lenkrades zu bestimmen [28]. Durch das Nutzen eines MR-Sensor-Arrays statt nur eines Sensors ist es möglich, eine höhere Präzision und Sicherheit der gemessenen Werte zu erzielen.

Lichtbogenmessung Für die Verbesserung von Niederspannungsschaltern ist die Analyse der Bewegung des Kontaktstücks und des entstehenden Lichtbogens essentiell, welche bisher weitgehend auf Basis optischer Beobachtung beruhte [24]. Da die zu untersuchenden Schalter dafür im Aufbau modifiziert werden müssen, um die Beobachtung möglich zu machen, wird in Quelle „**Development of High Resolution Magnetic Field Measurements for Switching Arc Diagnosis of Low-Voltage Switchgear**“ ein neuer Ansatz vorgestellt. Es sollen über das magnetische Feld außerhalb des Schalters durch inverse Berechnung der Stromdichte Informationen über den Lichtbogen und die Bewegung des Kontaktstücks gewonnen werden.

Erweiterte Sensordiagnose für Raddrehzahlsensoren MR-Sensoren können laut „**Development of advanced diagnostic functions in very high volume automotive sensor applications**“ [12] auch dazu genutzt werden, um Drehzahlsensoren, welche zum Beispiel beim Antiblockiersystem (ABS) oder der Fahrdynamikregelung (ESP) genutzt werden, zu untersuchen. Bisher geschah dies über das Beobachten der magnetischen Amplitude, in Quelle [12] soll zusätzlich das elektrische Signal, das der Drehzahlsensor erzeugt, überwacht werden, um Informationen über das gesamte Sensorsystem zu erhalten.

Magnetfeldkamera Das MR-Sensor-Array kann auch als Magnetfeldkamera eingesetzt werden, um zum Beispiel transiente Stromverteilung in nicht idealen elektromagnetischen Umgebungen aufzuspüren [23]. Darunter fallen beispielsweise die Strompfade innerhalb eines geschlossenen Gehäuses, wie das von Niederspannungs-Schalteinrichtungen.

In Abbildung 6.2 sind nochmal die zuvor aufgeführten Anwendungsgebiete mit ihren benötigten Parametern, dem benötigten Rechenaufwand und den mechanischen Toleranzen abgebildet.

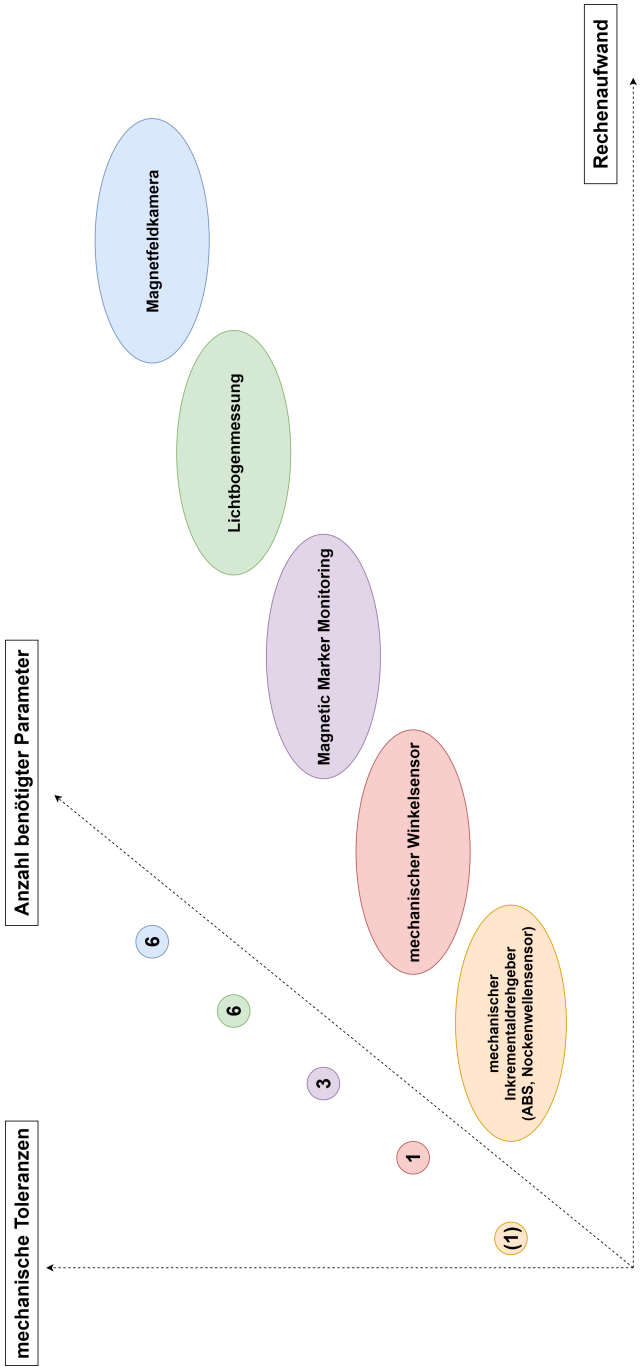


Abbildung 6.2: Übersicht über die Anzahl der benötigten Parameter und dem benötigten Rechenaufwand für verschiedene Anwendungsmöglichkeiten von MR-Sensor-Arrays.

Literaturverzeichnis

[1] *Tiva^{TM} C Series TM4C1294 Connected LaunchPad Evaluation Kit EK-TM4C1294XLUser's Guide. 2014.*, 2014

[2] https://de.wikipedia.org/wiki/Gradiometrie. (2017)

[3] https://de.wikipedia.org/wiki/Drehmatrix. (2021)

[4] ADVSOFTENG: Chartdirector. In: *https://www.advsofteng.com/index.html*

[5] BIEN, Dipl.-Ing.Tomasz: *Modellbasierte Kalibrierung, Detektion und Kompensation der Störungen des elektromagnetischen Trackingsystems für minimalinvasive Eingriffe*, Otto-von-Guericke-Universität Magdeburg, Dissertation, 2015

[6] BREDIES, Kristian ; LORENZ, Dirk: *Mathematische Bildverarbeitung*. Vieweg +Teubner, 2011

[7] BURGER, Wilhelm ; BURGE, Mark J.: *Digitale Bildverarbeitung - Eine algorithmische Einführung mit Java*. Springer Verlag, 2015

[8] DOMS, Marco ; PAUL, Johannes ; LEHNDORFF, Ronald ; GRIMM, Hubert: AMR vs. GMR vs. TMR - Eigenschaften, Unterschiede, Anwendungen. In: *5. Mikrosystemtechnik-Kongress, Aachen* (2013)

[9] GINSBERG, Daniel: *Lösungsstrategien für dünnbesetzte inverse Problemeim Bereich der Strukturüberwachung*, Universität Siegen, Dissertation, 2019

[10] HADJIGEORGIOU, Neoclis G. ; SOTIRIADIS, Paul P.: Parasitic Capacitances, Inductive Coupling, and High-Frequency Behavior of AMR Sensors. In: *IEEE SENSORS JOURNAL, VOL. 20, NO. 5, MARCH 1* (2020)

[11] HOLTIJ, Thomas ; SLATTER, Rolf: *Hochintegrierte Stromsensoren für Elektrofahrzeuge*. T.Tille, 2020

[12] Krey, Martin ; Sabotta, Daniel ; Zahn, Fabian ; Riemschneider, Karl-Ragmar ; Rettig, Rasmus: DEVELOPMENT OF ADVANCED DIAGNOSTIC FUNCTIONS IN VERY HIGH VOLUME AUTOMOTIVE SENSOR APPLICATIONS. In: *IEEE* (2013)

[13] Kruse, Rudolf: *Computational Intelligence*. Springer Verlag, 2015

[14] Lattmann, Thomas: Chipimplementation einer zweidimensionalen Fouriertransformation für die Auswertung eines Sensor-Arrays. (2018)

[15] Lückehe, Daniel: *Hybride Optimierung für Dimensionsreduktion*. Springer Verlag, 2015

[16] Mehm, T.: Schaltungsentwurf und Mikrocontrollersteuerung für ein Tunnel-Magnetoresistives Sensor-Array. (2019)

[17] Müller-Gronbach, Thomas ; Erich Novak·Klaus, Ritter: *Monte Carlo-Algorithmen*. Springer Verlag, 2012

[18] Nischwitz, Alfred ; Fischer, Max ; Haberäcker, Peter ; Socher, Gudrun: *Computergrafik und Bildverarbeitung Band II: Bildverarbeitung*. Vieweg +Teubner, 2004

[19] Pape, H.: Simulation und Auswertung von Permanentmagneten für magnetoresistive Sensor-Arrays. (2017)

[20] Petrak, Oleg: *Signal processing algorithms for magnetic sensor arrays*, HAW Hamburg, Diplomarbeit, 2018

[21] Raberg, W.: Magnetsensorschaltungen und -systeme und Verfahren zum Bilden von Magnetsensorschaltungen. (2017)

[22] Raschka, Sebastian ; Mirjalili, Vahid ; Lorenzen, Knut: *Machine Learning mit Python und Keras, TensorFlow 2 und Scikit-learn : Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. mitp, 2021

[23] Reil, Christian ; Meier, Matthias ; Schmidt, Hans-Peter: A High-Speed Magnetic Camera for Harsh Electromagnetic Environments. In: *IEEE* (2019)

[24] Reil, Christian ; Schmidt, Hans-Peter: Development of High Resolution Magnetic Field Measurements for Switching Arc Diagnosis of Low-Voltage Switchgear. In: *AFRICON Conference* (2017)

[25] Rindelaub, Simon Karl E.: Signalverarbeitung für magnetoresistive Sensor-Arrays mit Controller und Einplatinen-Computer. (2018)

[26] Schüthe, T. ; Riemschneider, K.-R. ; Petrak, O. ; Jünemann, K.: *Positionserfassung mittels Sensor-Array aus Tunnel-Magnetoresistiven Vortex-Dots und lernender Signalverarbeitung.* T.Tille, 2020

[27] Schüthe, Thorben: Abbildung TMR-Sensor Vergleich: linear und vortex. (2019)

[28] Schüthe, Thorben ; Jünemann, Klaus ; Riemschneider, Karl-Ragmar: Tolerance Compensation based on Gaussian Processes for Angle Measurements with Magnetic Sensor Arrays. In: *IEEE* (2020)

[29] Slatter, Dr. R. ; Manteuffel, Glenn von: *Magnetoresistive Sensoren für Weg-, Winkel-, Strom- und Feldmessung im Automobil.* T.Tille, 2016

[30] Suess, D. ; Brückl, H. ; Pruegl, K. ; Satz, A. ; Raberg, W.: Magnetsensorbauelement und Magneterfassungsverfahren. (2016)

[31] Suhl, Leena ; Mellouli, Taïeb: *Optimierungssysteme.* Springer Verlag, 2009

[32] Tarantola, Albert: *Inverse Problem Theory and Methods for Model Parameter Estimation.* Siam, 2005

[33] Tönnis, K. D.: *Grundlagen der Bildverarbeitung.* Pearson, 2005

[34] Vollmer, Jürgen: Kapitel 4 - Die Diskrete Fourier Transformation (DFT). In: *Signale und Systeme Skript* (2019)

[35] Woodall, William: Serial Library. In: *http://wjwwood.io/serial/doc/1.1.0/index.html*

[36] Wurft, T. ; Raberg, W.: Magnetsensorbauelement und Verfahren für ein Magnetsensorbauelement mit einer magnetoresistiven Struktur. (2015)

[37] WxWidgets: Overview. In: *https://www.wxwidgets.org/about/* (2021)

A Ergänzende Tests zur Ermittlung der ungefähren Grenzen des Suchalgorithmus

A.0.1 Verschiebung des Gebermagneten in x-Richtung (0, 0, -47)

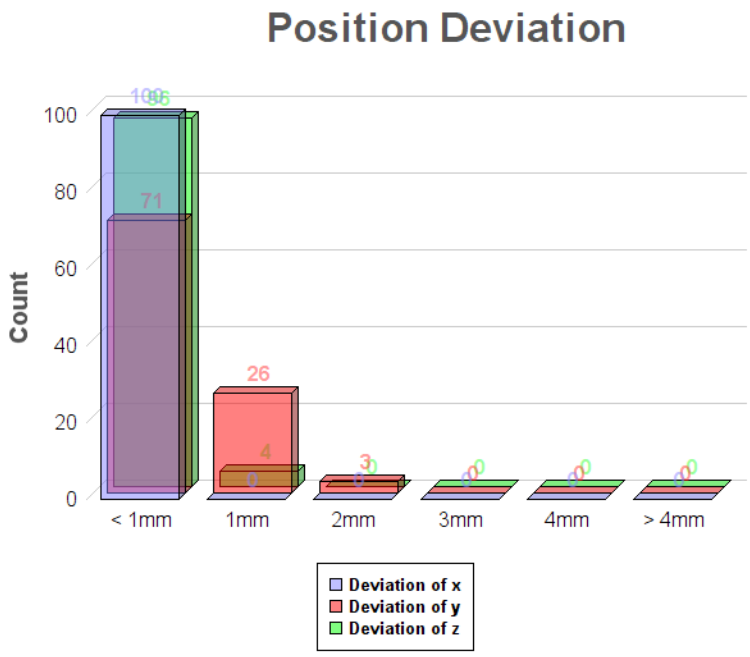


Abbildung A.1: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	100	0	0	0	0	0
y	71	26	3	0	0	0
z	96	4	0	0	0	0

Tabelle A.1: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.247	0.112	0.335	
y	0.793	0.124	0.352	0.393
z	-44.893	0.241	0.491	

Tabelle A.2: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -47).

A.0.2 Verschiebung des Gebermagneten in x-Richtung (5, 0, -47)

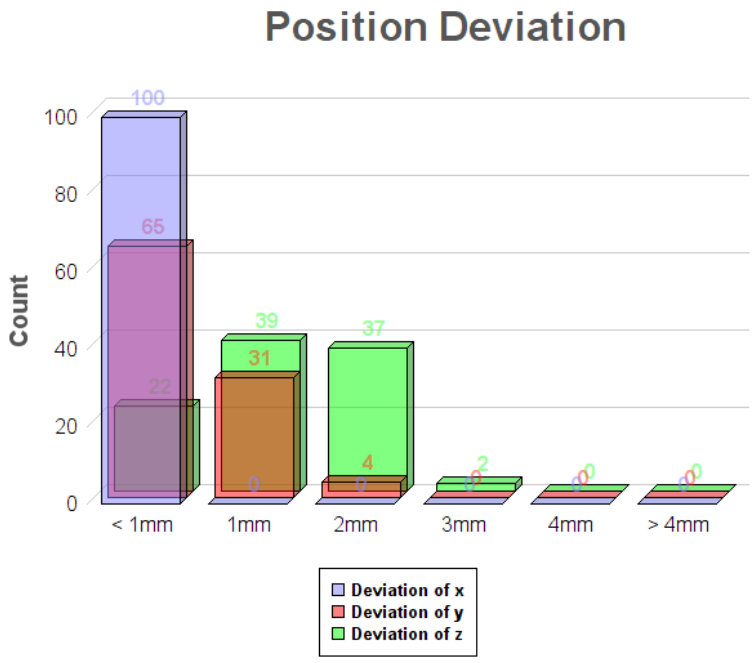


Abbildung A.2: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (5, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	100	0	0	0	0	0
y	65	31	4	0	0	0
z	22	39	37	2	0	0

Tabelle A.3: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (5, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	5.082	0.118	0.344	
y	0.916	0.120	0.347	0.402
z	-46.337	0.265	0.515	

Tabelle A.4: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (5, 0, -47).

A.0.3 Verschiebung des Gebermagneten in x-Richtung (10, 0, -47)

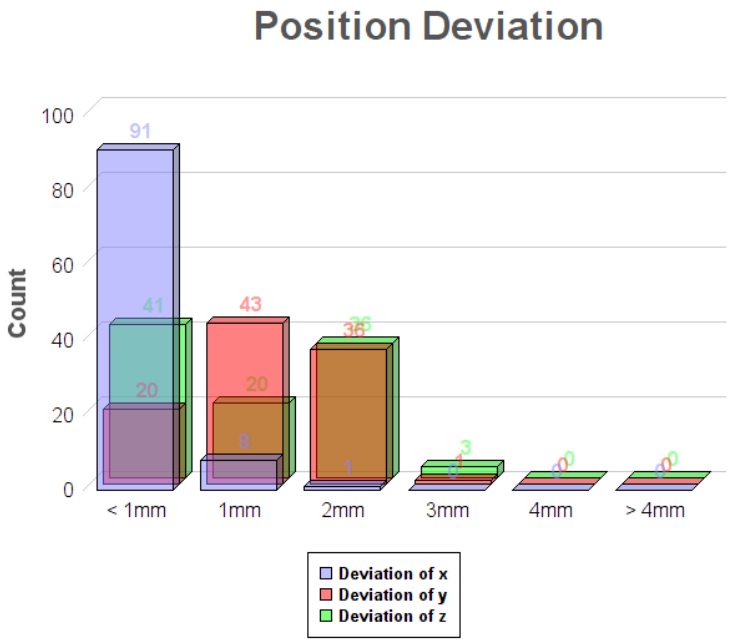


Abbildung A.3: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (10, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	91	8	1	0	0	0
y	20	43	36	1	0	0
z	41	20	37	36	3	0

Tabelle A.5: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (10, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	9.622	0.202	0.449	
y	1.371	0.210	0.458	0.501
z	-43.725	0.354	0.595	

Tabelle A.6: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (10, 0, -47).

A.0.4 Verschiebung des Gebermagneten in x-Richtung (15, 0, -47)

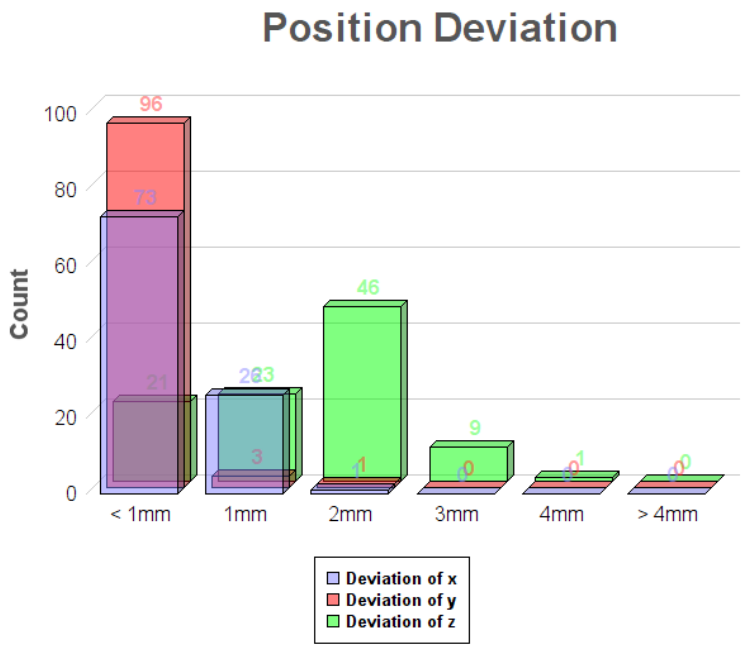


Abbildung A.4: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (15, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	73	26	1	0	0	0
y	96	3	1	0	0	0
z	21	23	46	9	1	0

Tabelle A.7: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (15, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	14.319	0.172	0.415	
y	0.0491	0.232	0.481	0.525
z	-43.398	0.462	0.680	

Tabelle A.8: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (15, 0, -47).

A.0.5 Verschiebung des Gebermagneten in x-Richtung (20, 0, -47)

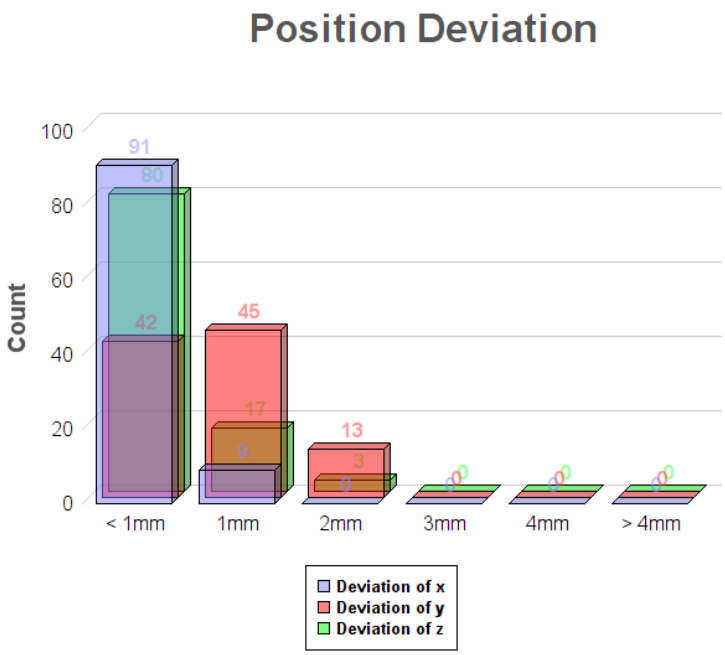


Abbildung A.5: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (20, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	91	9	0	0	0	0
y	42	45	13	0	0	0
z	80	17	3	0	0	0

Tabelle A.9: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (20, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	19.915	0.215	0.464	
y	1.011	0.164	0.406	0.529
z	-45.478	0.499	0.706	

Tabelle A.10: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (20, 0, -47).

A.0.6 Verschiebung des Gebermagneten in x-Richtung (25, 0, -47)

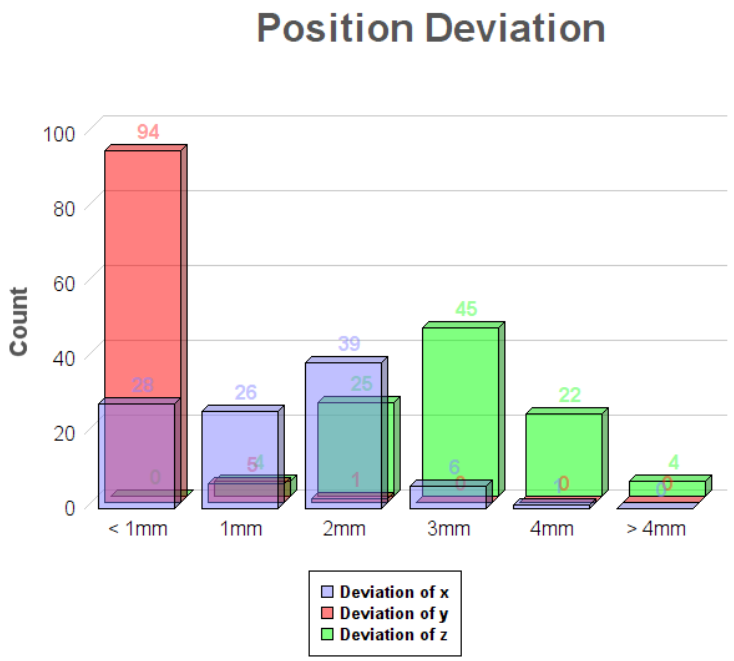


Abbildung A.6: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (25, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	28	26	39	6	1	0
y	94	5	1	0	0	0
z	0	4	25	45	22	4

Tabelle A.11: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (25, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	25.370	0.944	0.971	
y	0.177	0.209	0.457	0.751
z	-47.492	0.679	0.824	

Tabelle A.12: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (25, 0, -47).

A.0.7 Verschiebung des Gebermagneten in y-Richtung (0, 0, -47)

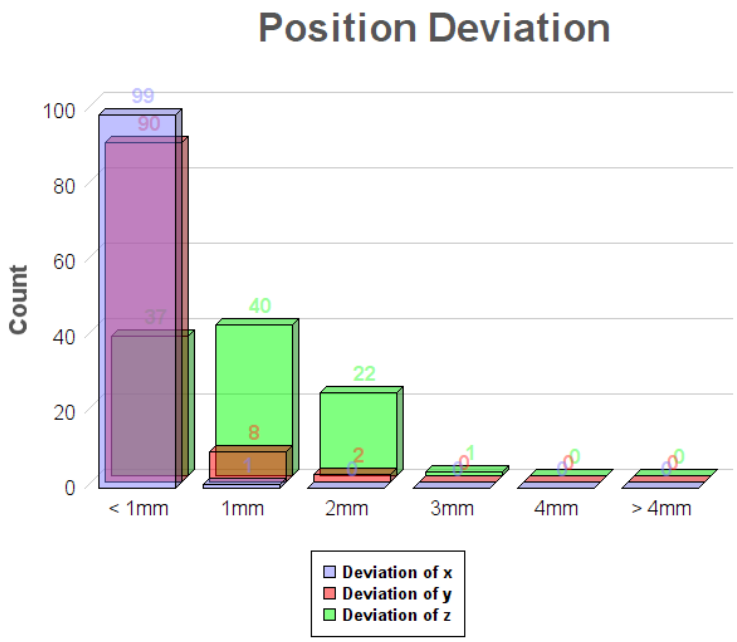


Abbildung A.7: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -47).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	99	1	0	0	0	0
y	90	8	2	0	0	0
z	37	40	22	1	0	0

Tabelle A.13: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.099	0.121	0.348	
y	0.324	0.248	0.499	0.508
z	-43.843	0.457	0.676	

Tabelle A.14: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -47).

A.0.8 Verschiebung des Gebermagneten in y-Richtung (0, 5, -47)

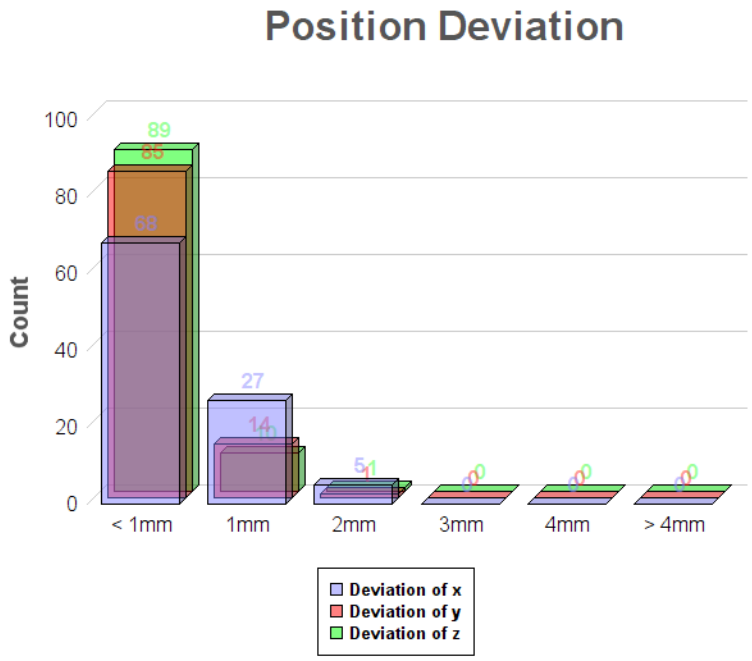


Abbildung A.8: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 5, -47).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	69	27	5	0	0	0
y	85	14	1	0	0	0
z	89	10	1	0	0	0

Tabelle A.15: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 5, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.788	0.242	0.492	
y	4.617	0.281	0.530	0.563
z	-44.785	0.447	0.668	

Tabelle A.16: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 5, -47).

A.0.9 Verschiebung des Gebermagneten in y-Richtung (0, 10, -47)

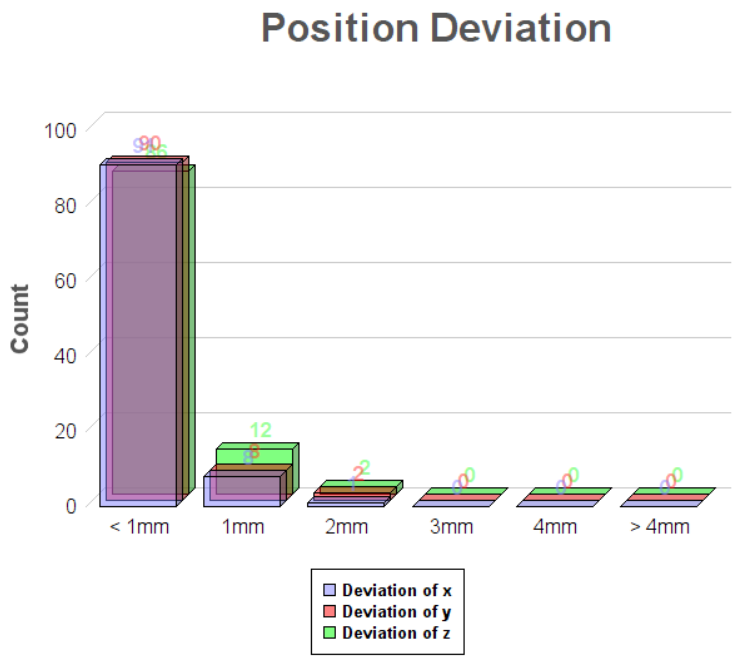


Abbildung A.9: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 10, -47).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	91	8	1	0	0	0
y	90	8	2	0	0	0
z	86	12	2	0	0	0

Tabelle A.17: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 10, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.532	0.152	0.390	
y	10.448	0.230	0.479	0.499
z	-45.080	0.479	0.628	

Tabelle A.18: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 10, -47).

A.0.10 Verschiebung des Gebermagneten in y-Richtung (0, 15, -47)

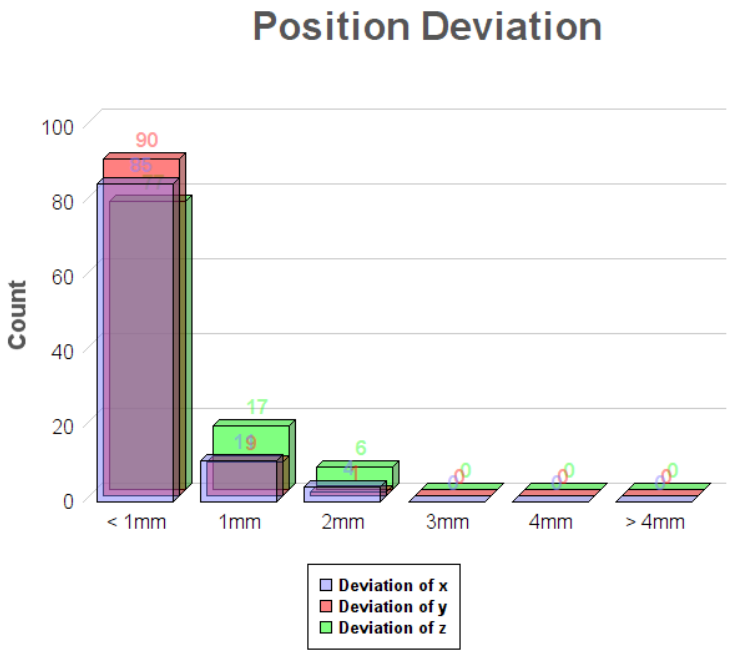


Abbildung A.10: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 15, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	85	11	4	0	0	0
y	90	9	1	0	0	0
z	77	17	6	0	0	0

Tabelle A.19: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 15, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.622	0.195	0.441	
y	15.312	0.257	0.507	0.565
z	-44.716	0.557	0.746	

Tabelle A.20: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 15, -47).

A.0.11 Verschiebung des Gebermagneten in y-Richtung (0, 20, -47)

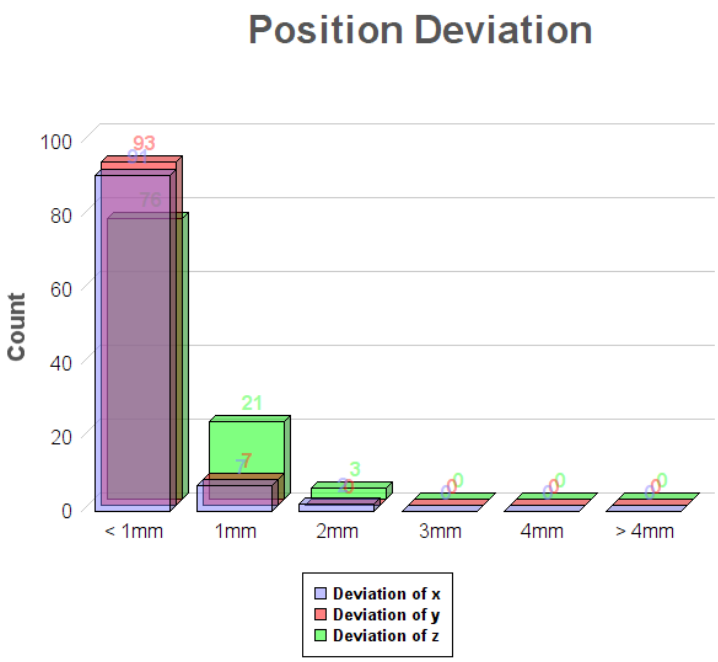


Abbildung A.11: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 20, -47).

Abweichung in mm	$<$ 1mm	1mm	2mm	3mm	4mm	$>$ 4mm
x	91	7	2	0	0	0
y	93	7	0	0	0	0
z	76	21	3	0	0	0

Tabelle A.21: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 20, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.309	0.208	0.456	
y	20.161	0.318	0.564	0.561
z	-44.523	0.438	0.662	

Tabelle A.22: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 20, -47).

A.0.12 Verschiebung des Gebermagneten in y-Richtung (0, 25, -47)

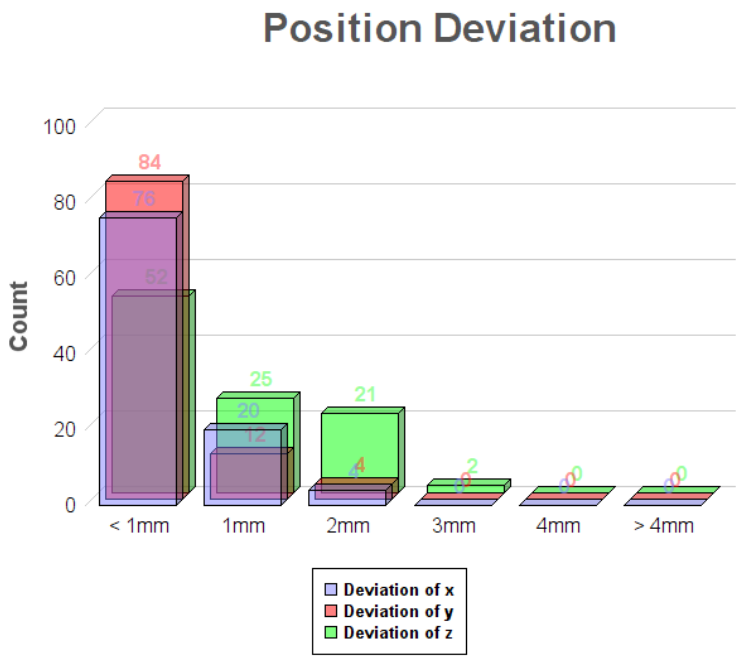


Abbildung A.12: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 25, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	76	20	4	0	0	0
y	84	12	4	0	0	0
z	52	25	21	2	0	0

Tabelle A.23: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 25, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.589	0.227	0.476	
y	25.329	0.357	0.597	0.576
z	-43.920	0.430	0.656	

Tabelle A.24: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 25, -47).

A.0.13 Verschiebung des Gebermagneten in z-Richtung (0, 0, -17)

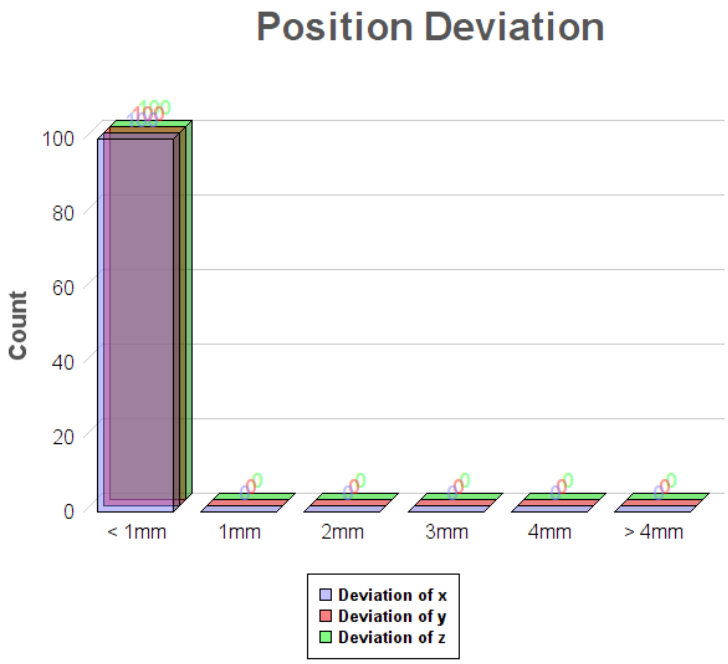


Abbildung A.13: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -17).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	100	0	0	0	0	0
y	100	0	0	0	0	0
z	100	0	0	0	0	0

Tabelle A.25: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -17).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.198	0.004	0.061	
y	0.302	0.007	0.082	0.082
z	-16.797	0.010	0.102	

Tabelle A.26: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -17).

A.0.14 Verschiebung des Gebermagneten in z-Richtung (0, 0, -27)

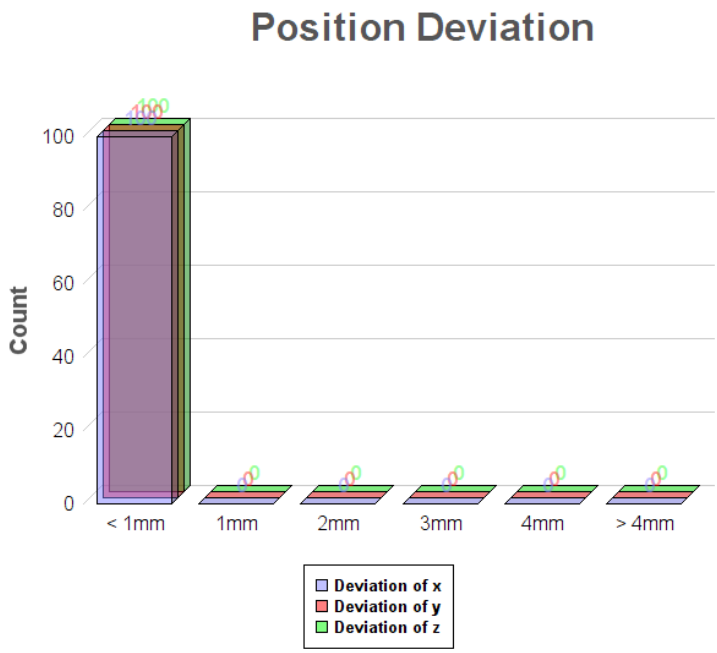


Abbildung A.14: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -27).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	100	0	0	0	0	0
y	100	0	0	0	0	0
z	100	0	0	0	0	0

Tabelle A.27: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -27).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.316	0.008	0.089	
y	-0.277	0.007	0.082	0.094
z	-26.366	0.012	0.110	

Tabelle A.28: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -27).

A.0.15 Verschiebung des Gebermagneten in z-Richtung (0, 0, -37)

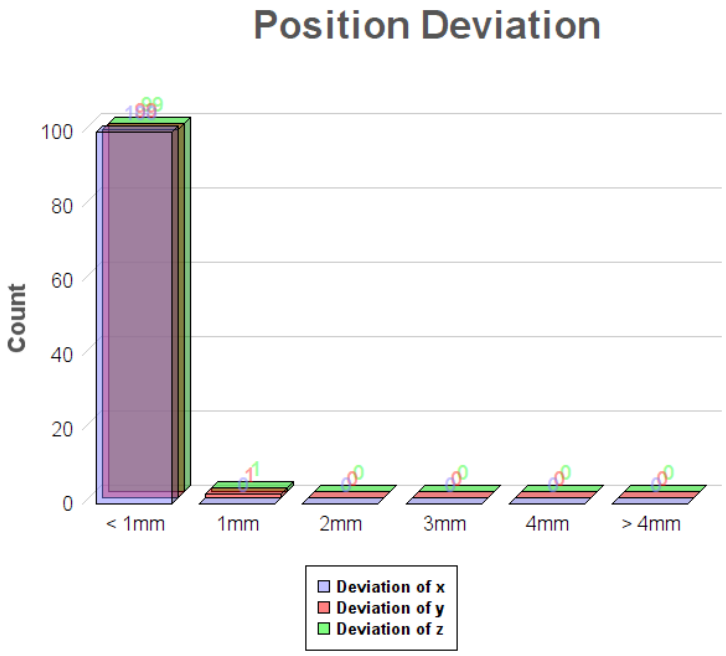


Abbildung A.15: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -37).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	100	0	0	0	0	0
y	99	1	0	0	0	0
z	99	1	0	0	0	0

Tabelle A.29: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -37).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	0.164	0.036	0.191	
y	-0.505	0.031	0.176	0.204
z	-37.356	0.060	0.244	

Tabelle A.30: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -37).

A.0.16 Verschiebung des Gebermagneten in z-Richtung (0, 0, -47)

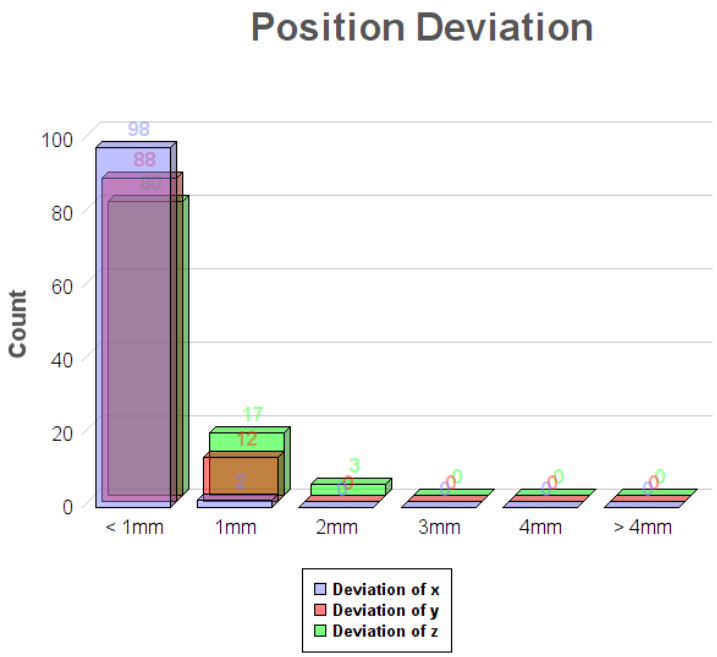


Abbildung A.16: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -47).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	98	2	0	0	0	0
y	88	12	0	0	0	0
z	80	17	3	0	0	0

Tabelle A.31: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -47).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.227	0.139	0.373	
y	0.430	0.179	0.423	0.421
z	-46.398	0.217	0.466	

Tabelle A.32: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -47).

A.0.17 Verschiebung des Gebermagneten in z-Richtung (0, 0, -57)

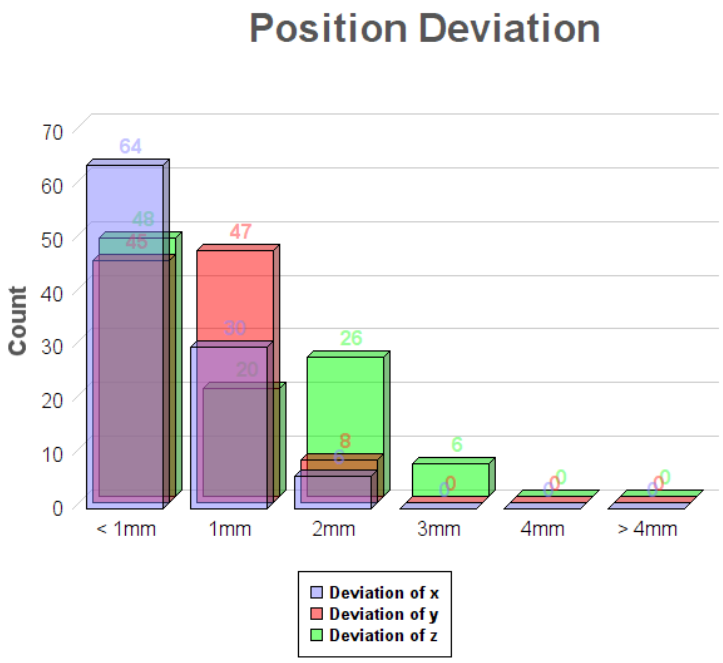


Abbildung A.17: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -57).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	64	30	6	0	0	0
y	45	47	8	0	0	0
z	48	20	26	6	0	0

Tabelle A.33: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -57).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-0.829	0.099	0.315	
y	1.086	0.109	0.330	0.493
z	-55.900	0.697	0.835	

Tabelle A.34: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -57).

A.0.18 Verschiebung des Gebermagneten in z-Richtung (0, 0, -67)

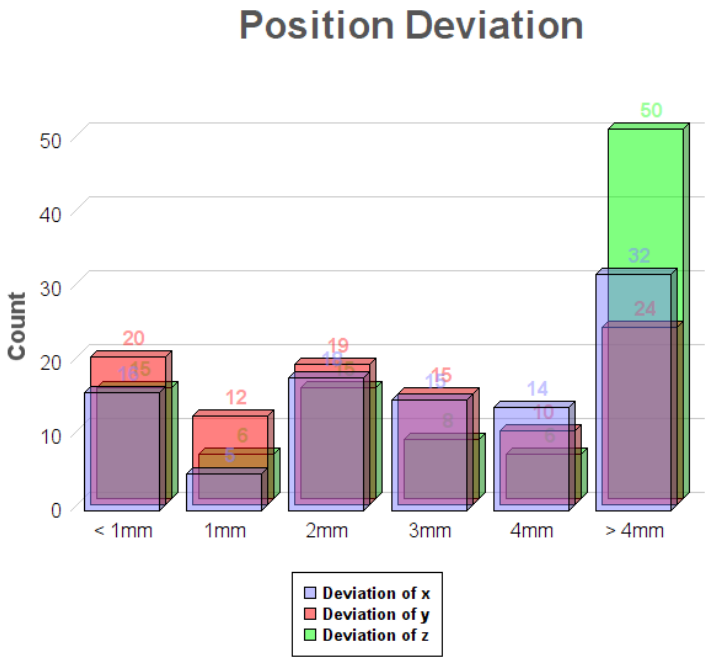


Abbildung A.18: Histogramm der Abweichung der ermittelten Position von 100 Suchdurchläufen bei einer Entfernung von (0, 0, -67).

Abweichung in mm	< 1mm	1mm	2mm	3mm	4mm	> 4mm
x	16	5	18	15	14	32
y	20	12	19	15	10	24
z	15	6	15	8	6	50

Tabelle A.35: Abweichung zwischen berechneter und tatsächlicher Position des Magneten bei einer Entfernung von (0, 0, -67).

	Mittelwert (mm)	Varianz (mm^2)	Standardabweichung (mm)	Gesamtstandardabweichung (mm)
x	-3.114	124.921	11.177	
y	0.454	32.563	5.706	8.071
z	-64.945	53.717	7.329	

Tabelle A.36: Mittelwert, Varianz und Standardabweichung der vom Suchalgorithmus ermittelten Position bei einer Entfernung von (0, 0, -67).

B Anwendungshinweise

B.0.1 Installation der nötigen Bibliotheken

Um die aufgeführten Bibliotheken zu nutzen, muss der verwendeten Entwicklungsumgebung (beispielsweise Visual Studio) entweder aufgezeigt werden, wo sich der Speicherort von diesen befindet. Dies kann entweder in der Entwicklungsumgebung selbst geschehen oder über das zu CMake zugehörige „CMakeLists.txt“. Der Einfachheit halber können Umgebungsvariablen wie beispielsweise $(wxwin) genutzt werden, um den Speicherpfad zu verkürzen. In dieser Arbeit wurden für die Bibliotheken folgende Umgebungsvariablen genutzt:

- **wxWidgets:** $(wxwin)
- **ChartDirector:** $(chartdir)
- **Serial.h:** $(serialcom)

Für weitere Informationen, wie wxWidgets unter dem jeweiligen Betriebssystem installiert wird, kann folgende Seite genutzt werden: https://wiki.wxwidgets.org/Compiling_and_getting_started

Speicherpfad angeben in der Entwicklungsumgebung

Windows - Visual Studio

- Reiter Project > Properties > C/C++ > Additional Include Directories > $(wxwin)\include\msvc; $(wxwin)\include ;$(chartdir)\include; $(serialcom)\include
- Reiter Project > Properties > C/C++ > Preprocessor > WXUSINGDLL; wxMSVC_VERSION_AUTO;

- Reiter Project > Properties > Linker > General > Additional Library Directories $(wxwin)\lib\vc142_x64_dll; $(wxwin)\lib\vc_lib; $(serialcom)\src; $(serialcom)\include
- Reiter Project > Properties > Linker > Input > Additional Dependencies $(chartdir)\lib\win64\chartdir60.lib

CMake

Eine andere Möglichkeit, das Projekt zu erstellen und die notwendigen Bibliotheken einzubinden, ist die Nutzung des Buildsystems CMake. Dafür werden die benötigten Dateien vorzugsweise in einem Ordner abgelegt und über das „CMakeLists.txt“ wird der Entwicklungsumgebung mitgeteilt, wo sich diese befinden. Für genauere Informationen, wie CMake genutzt wird, bietet sich folgende Seite an:
https://cmake.org/cmake/help/latest/guide/tutorial/index.html
Im nachfolgenden Listing B.1 ist das in dieser Arbeitfür Linux genutzte „CMakeLists.txt“ abgebildet, mit welchem sich die benötigten Bibliotheken, Klassen und Header-Files hinzufügen lassen:

Listing B.1: „CMakeLists.txt“, welches in dieser Arbeit verwendet wurde.

```
cmake_minimum_required (VERSION 3.8)
project(MY_PROJECT)
set(wxWidgets_USE_STATIC 1)
find_package(wxWidgets REQUIRED COMPONENTS net core base)
include(${wxWidgets_USE_FILE})

if(UNIX)
file(GLOB PNGFILES "img/*.png")
file(GLOB HEADRFILES "codefiles/*.h")
file(GLOB SOUCECODEFILES "codefiles/*.cpp")
file(GLOB SERIAL "serial/unix.cc" "serial/impl/unix.h"
"serial/serial.h" "serial/serial.cc" "serial/list_ports_linux.cc")
include_directories(main STATIC
chartdir_cpp_linux_64/ChartDirector/include)
link_directories(main STATIC
chartdir_cpp_linux_64/ChartDirector/lib)
add_executable (main
"cApp.cpp"
${HEADRFILES}
${SOUCECODEFILES}
${SERIAL}
"chartdir_cpp_linux_64/ChartDirector/include/chartdir.h")
endif()

target_link_libraries(main PRIVATE
${wxWidgets_LIBRARIES} chartdir)
file(COPY ${PNGFILES} DESTINATION .)
```

B.0.2 Ermitteln des COM-Ports

Windows

Geräte-Manager > Anschlüsse (COM & LPT) > Stellaris Virtual Serial Port (COM**X**)

Linux

Zur Ermittlung des verwendeten COM-Ports unter Linux kann folgender Befehl im Terminal eingegeben werden: `$ ls /dev/ttyA*`

B.0.3 Einstellen des COM-Ports

Entweder lässt sich der genutzte Port über die entworfene Software direkt einstellen oder er kann fest im Programmcode implementiert werden, um beispielsweise das Debuggen zu vereinfachen. Somit muss es nicht bei jedem Programmstart neu über die entsprechende Auswahlliste ausgewählt werden.

Windows

```
Serial* m_serialPort = new Serial("COM8", 625000, serial::Timeout::simpleTimeout(1000));
```

Linux

```
Serial* m_serialPort = new Serial(" /dev/ttyACM0 ", 625000, serial::Timeout::simpleTimeout(1000));
```

C CD

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