HAW HAMBURG

Bachelorthesis

Simon Karl Eduard Rindelaub

Signalverarbeitung für magnetoresistive Sensor-Arrays mit Controller und Einplatinen-Computer

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Simon Karl Eduard Rindelaub

Signalverarbeitung für magnetoresistive Sensor-Arrays mit Controller und Einplatinen-Computer

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

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Klaus Jünemann

Abgegeben am 26. September 2018

Simon Karl Eduard Rindelaub

Thema der Bachelorarbeit

Signalverarbeitung für magnetoresistive Sensor-Arrays mit Controller und Einplatinen-Computer

Stichworte

TMR-Sensor, tunnel-magnetoresistiv, Sensor-Array, Tiva-Board TM4C1294, Fourier-Transformation, CORDIC, Raspberry Pi, Python, Softwareentwicklung, GUI, PyQt5, Matplotlib

Kurzzusammenfassung

Angefangen bei der Ideenfindung, wird der Entwicklungsprozess der Soft- und Hardware beleuchtet und abschließend das Gesamtsystem in Betrieb genommen und analysiert. Bei dem System handelt es sich um ein Sensor-Array mit Mikrocontroller, welches mit einem Raspberry Pi verbunden ist. Auf dem Raspberry Pi wurde eine Benutzeroberfläche (GUI) entwickelt, mit der die Signalverarbeitung auf dem Mikrocontroller sowie die grafische Ausgabe der Messwerte gesteuert werden kann. Das System soll für Vorführungen genutzt werden und ist daher kompakt gehalten und transportabel.

Simon Karl Eduard Rindelaub

Title of the bachelor thesis

Signal processing for magnetoresistive sensor arrays with controller and single-board computer

Keywords

TMR sensor, tunneling magnetoresistance, sensor array, tiva board TM4C1294, fourier transformation, CORDIC, raspberry pi, python, software development, GUI, PyQt5, matplotlib

Abstract

Starting with brainstorming, the development process of the software and hardware is examined and finally the entire system is put into operation and analyzed. The system is a sensor array with microcontroller connected to a Raspberry Pi. On the Raspberry Pi, a user interface (GUI) has been developed, with which the signal processing on the microcontroller and the graphical output of the measured values can be controlled. The system will be used for demonstrations and is therefore compact and portable.

Vorwort

Die Forschungen zur vorliegenden Arbeit wurden in der Zeit von April 2018 bis September 2018 in dem Forschungsverbundprojekt ISAR an der Hochschule für Angewandte Wissenschaften Hamburg durchgeführt.

Ein großer Dank gilt Prof. Dr.-Ing. K.-R. Riemschneider, den ich sehr für den Umgang mit seinen Studenten und Mitarbeitern schätze. Dank der tollen Projektleitung habe ich mich in dem Team sehr wohl und gut aufgehoben gefühlt.

Weiteren Dank spreche ich meinem Zweitprüfer Prof. Dr. K. Jünemann für die Übernahme der Prüfung und das Interesse an meinem Thema aus.

Besondere Unterstützung habe ich von M.Sc. T. Schüthe erhalten. Er hat viele Stunden investiert und mir stets mit Rat und Tat zur Seite gestanden. Dafür bin ich ihm von ganzem Herzen dankbar.

Herrn G. Müller möchte ich hier ebenso erwähnen und für die Zeit und Mühe bei den Korrekturen danken. Die kritischen und ehrlichen Hinweise und Gespräche haben mir sehr geholfen.

Mit sehr viel Freude habe ich in der Projektgruppe gearbeitet und meine Kommilitonen sehr ins Herz geschlossen. Auch ihnen spreche ich meine Dankbarkeit für die tolle Arbeitsatmosphäre und Unterstützung aus.

Unendlichen Dank empfinde ich für meine Mutter K. Rindelaub und für meine Freundin J. Jankowski. Meine Mutter hat mir stets den Rücken für das Studium freigehalten und mich durch alle Lebenslagen getragen. Meine Freundin gab mir stets halt und hatte immer ein offenes Ohr und offene Arme für mich. Beiden danke ich außerdem sehr für die Zeit und Mühe bei den Korrekturen meiner Arbeit. Ohne die Beiden wäre vieles nicht möglich gewesen.

Begrenzt ist das Leben – unendlich die Erinnerung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

AMR	Anisotropic Magneto-Resistance
CCS	Code Composer Studio v6
DFT	Diskrete Fourier-Transformation
FFTW	Fastest Fourier Transform in the West
GMR	Giant Magneto-Resistance
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDFT	inverse Diskrete Fourier-Transformation
ISAR	Integrated Sensor-Array
KFZ	Kraftfahrzeug
KQ-Methode	Methode der kleinsten Quadrate
MC	Mikrocontroller
MIT	Massachusetts Institute of Technology
TI	Texas Instruments
TMR	Tunneling MagnetoResistance

1 Einleitung

1.1 Stand der Technik

Der Stand der Technik lässt sich sehr gut an dem Beispiel festhalten, dass der PC, der vor einigen Jahrzehnten noch ein ganzes Zimmer füllte, heute in eine Hosentasche passt und dabei noch weitaus mehr Funktionen mit sich bringt. Was dabei deutlich wird: Die Bauteile werden immer kleiner, damit aber auch komplexer. Um ein System von komplexen Bauteilen zu steuern, benötigt es eine Überwachungsmöglichkeit. Als einfaches Überwachungsgerät wird in diesem Fall ein Sensor verwendet. Die Sensoren von früher wurden heute um eine komplexe Steuerung, teils sogar mit Messwertauswertung, erweitert. Das große Problem, welches es in den Griff zu bekommen gilt, ist die Störsicherheit. Der Sensor soll in rauen Umgebungen weiterhin zuverlässige Messwerte liefern.

1.2 Ziel dieser Arbeit

Im Verlauf dieser Bachelorarbeit wird ein Demonstrationskoffer für das Forschungsprojekt Integrated Sensor-Array (ISAR) entwickelt. Eingesetzt wird ein Sensor-Array mit Tunneling MagnetoResistance (TMR) Sensoren zur Winkelbestimmung von sich über dem Sensorfeld drehender Gebermagnete. Das Gesamtsystem soll die Möglichkeit bieten, den Magneten zu drehen und die Messungen in verschiedenen Diagrammen auf einem Monitor auszugeben. Das Menü und die Graphen werden in einer GUI dargestellt. Die Ansteuerung des Sensorboards wird über das Tiva-Board TM4C1294 geregelt. Ebenfalls werden komplexe Rechnungen wie die zweidimensionale Diskrete Fourier-Transformation (DFT), Funktionsapproximationen und der CORDIC-Algorithmus auf diesem Board implementiert.

1.3 Motivation

Die in dieser Arbeit verwendeten Sensoren sollen in Zukunft in Automobilen eingesetzt werden. Die Anforderungen an die bereits erwähnte Störsicherheit sind in dieser Branche besonders hoch. Erste Prototypen des Sensor-Arrays sind bereits entworfen und am Ende soll ein fertiger Chip hergestellt werden können. Um die in dem Projekt bereits gewonnenen Erkenntnisse und Erfahrungen auf Messen und Veranstaltungen vorführen zu können, bedarf es eines kleinen, tragbaren Demonstrationssystems. Dieses soll die Vorteile des Sensor-Arrays den Interessenten verständlich veranschaulichen und zur eigenständigen Bedienung und zu Versuchen einladen.

2 Grundlagen

In diesem Kapitel wird auf die mathematischen und technischen Grundlagen eingegangen, die für das Verständnis dieser Arbeit benötigt werden. Die mathematischen Grundlagen beinhalten Formeln aus der Signalverarbeitung und Statistik sowie Rechenoperationen. Die technischen Grundlagen beziehen sich auf das Sensor-Array, die einzelnen Sensoren und deren Funktionsweise. Es werden zwei Sensortypen näher beleuchtet, die in der Projektentwicklung von ISAR eine wichtige Rolle spielen.

2.1 Methoden der Signalverarbeitung

In diesem Abschnitt werden die wichtigsten Formeln und Methoden zur Signalverarbeitung erläutert und warum sie in der jeweiligen Form umgesetzt wurden. In Kapitel 3.3 wird auf diese Methoden bezüglich der Implementierung eingegangen und wie sie auf dem Board realisiert werden.

2.1.1 Diskrete Fouriertransformation

Die DFT ist eine Transformation aus dem Bereich der Fourier-Analysis. Ein zeitdiskretes endliches Signal wird auf ein diskretes, periodisches Frequenzspektrum (Bildbereich) abgebildet. Die DFT besitzt in der digitalen Signal- und Bildverarbeitung zur Analyse eine große Bedeutung und kommt vielfältig zur Anwendung [23].

Die DFT ist in der Signal- und Bildverarbeitung ein weit verbreitetes Werkzeug. Geläufige Anwendungen sind die Bestimmung der Frequenzanteile aus einem abgetasteten Signal oder die Zuordnung von Frequenzen und Amplituden. In dieser Arbeit wird die DFT angewandt, um die Implementierung digitaler Filter zu vereinfachen. Nach der Transformation der Datenwerte vom Zeit- in den Bildbereich, wird die Signalfilterung durch eine Matrixmultiplikation realisiert. Eine rechenaufwendige Faltung ist nicht mehr notwendig. Nachfolgend eine Darstellung der Integration der DFT mit der Ausgangsgleichung (2.1).

$$\hat{g}_{k,l} = \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} g_{m,n} \cdot \exp\left(-2\pi \mathrm{i} \cdot \frac{mk}{M}\right) \right) \exp\left(-2\pi \mathrm{i} \cdot \frac{nl}{N}\right) \tag{2.1}$$

Um diese Funktion nun auf einem Mikrocontroller (MC) zu implementieren, werden die

Exponentialfunktionen vereinfacht.

$$\exp\left(\frac{-2\,\pi\,\mathrm{i}\,m\,k}{M}\right) = {W_M}^{-mk} \qquad \exp\left(\frac{-2\,\pi\,\mathrm{i}\,n\,k}{N}\right) = {W_N}^{-nl} \tag{2.2}$$

Diese neuen Faktoren aus Gleichung (2.2) werden Twiddle-Faktoren genannt und in der Twiddle-Matrix zusammengefasst. Die Vereinfachung besteht darin, die Eingangssignalwertematrix G_{MN} mit den Twiddle-Matrizen W_M^{-mk} und W_N^{-nl} zu multiplizieren, um so weitere rechenaufwendige Multiplikationen zu vermeiden.

$$\hat{G} = {W_M}^{-mk} \cdot G \cdot {W_N}^{-nl} \tag{2.3}$$

Die Twiddle-Matrizen können bereits zum Programmstart initialisiert werden und müssen nicht im laufenden Prozess durchgehend neu berechnet werden. Nach der Filterung der Eingangswerte (siehe Kap. 2.1.4) wird die Matrix mittels inverser diskreter Fourier-Transformation(IDFT) wieder in den Zeitbereich transformiert. Analog gilt die Gleichung (2.4), wobei nun der Exponent der e-Terme positiv ist. Diese Gleichung lässt sich ebenfalls mit Hilfe der Twiddle-Faktoren vereinfachen (2.5) , um anschließend die Matrizenmultiplikationen durchzuführen zu können (2.6). Auch diese inversen Twiddle-Matrizen werden zum Programmstart initialisiert.

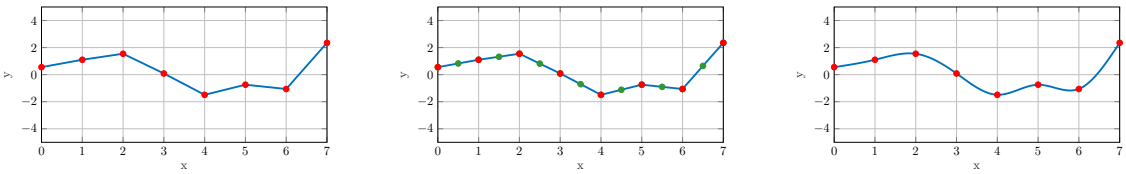
$$\hat{g}_{k,l} = \sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1} g_{m,n} \cdot \exp\left(2\pi\mathrm{i}\cdot\frac{mk}{M}\right)\right)\exp\left(2\pi\mathrm{i}\cdot\frac{nl}{N}\right) \tag{2.4}$$

$$\exp\left(\frac{2\,\pi\,\mathrm{i}\,m\,k}{M}\right) = {W_M}^{mk} \qquad \exp\left(\frac{2\,\pi\,\mathrm{i}\,n\,k}{N}\right) = {W_N}^{nl} \tag{2.5}$$

$$G = {W_M}^{mk} \cdot \hat{G} \cdot {W_N}^{nl} \tag{2.6}$$

2.1.2 Interpolation

Die Interpolation beschreibt in der Mathematik das Bilden einer kontinuierlichen Funktion anhand von diskreten Werten. Ziel ist, dass auch angenäherte Werte, die zwischen



(a) Zu interpolierende Punkte. (b) Lineare Interpolation. (c) Spline Interpolation.

Abbildung 2.1: Vergleich zweier Interpolationsmöglichkeiten Anhand diskreter Werte.

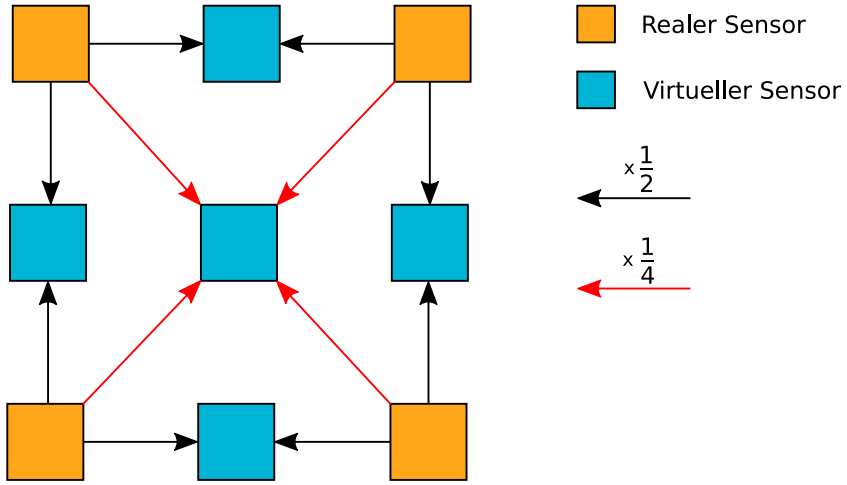


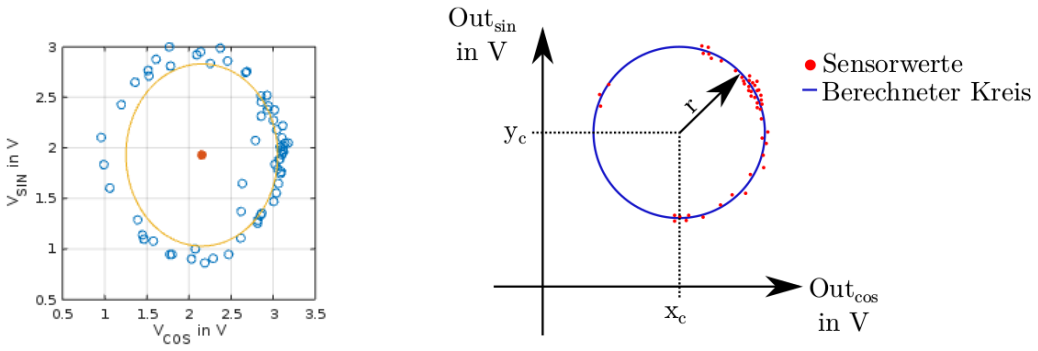
Abbildung 2.2: Berechnungsprinzip der Interpolation.

den Messpunkten liegen, verfügbar sein sollen. Am häufigsten wird in der Praxis die lineare Interpolation angewendet, da diese einfach zu implementieren und berechnen ist und bereits eine gute Approximation der Werte liefert. Die nachfolgende Abbildung 2.1 zeigt schematisch, wie diskrete Werte 2.1(a) einmal linear interpoliert 2.1(b), oder durch eine Funktion angenähert werden 2.1(c).

Die Interpolation spielt für dieses Projekt eine tragende Rolle, da in Zukunft statt der 8×8-Sensormatrix eine 15×15-Sensormatrix verwendet werden soll. Somit können schon vor der Hardwareentwicklung erste Simulationen durchgeführt werden. Eine Matrix mit 15×15 Sensoren bietet die Vorteile einer höheren Störsicherheit und Genauigkeit. Das Berechnungsprinzip der virtuellen Sensoren wird mit Hilfe der linearen Interpolation in Abbildung 2.2 verdeutlicht. Je nach Position des virtuellen Sensors werden entweder je $\frac{1}{4}$ der diagonal umliegenden Sensorwerte als Einfluss einbezogen oder je $\frac{1}{2}$ der horizontal oder vertikal benachbarten Sensorwerte.

2.1.3 Offsetkompensation

Der Offset- oder auch Nullpunktfehler ist ein Problem realer Messsysteme. Dabei hat nicht jedes gemessene Signal der einzelnen Sensoren der Sensor-Matrix den selben Offset. Zur Lösung wird die Methode der kleinsten Quadrate (KQ-Methode) angewendet. Die aufgenommenen Messwerte werden, wie in Abbildung 2.3(a) abgebildet, kreisförmig im Koordinatensystem angeordnet. Zur Ausgleichsrechnung wird eine Kreisfunktion verwendet (2.9), um die Residuen zu minimieren. Für die Berechnungen müssen die 8×8 Sinus- und Cosinus-Signal-Wert-Matrizen zu je einem 1×64 Vektor transformiert werden (Gleichungen (2.7) und (2.8)).



(a) Methode der kleinsten Quadrate mit Kreisplott.

(b) Kreisplott mit Offsetwerten y_c (Imaginärteil) und x_c (Realteil).

Abbildung 2.3: Grafische Darstellung der Offsetberechnung.

$$\mathbf{x} = (V_{COS11} \quad V_{COS12} \quad \dots \quad V_{COS21} \quad \dots \quad V_{COS88})^T \tag{2.7}$$

$$\mathbf{y} = (V_{SIN11} \quad V_{SIN12} \quad \dots \quad V_{SIN21} \quad \dots \quad V_{SIN88})^T \tag{2.8}$$

Aus den beiden Vektoren wird eine Matrix $\mathbf{A} = (\mathbf{x}, \mathbf{y}, 1)$ erstellt. Diese hat die Dimension 64×3. Nun können der Kreisfunktionsvektor $\mathbf{b}$ und der Offsetvektor $\mathbf{c}$ berechnet werden.

$$\mathbf{b} = (-(\mathbf{x}^2 + \mathbf{y}^2)) \tag{2.9}$$

$$\mathbf{c} = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T \cdot \mathbf{b} \tag{2.10}$$

Der Offsetvektor $\mathbf{c}$ (2.10) ist ein 3×1 Vektor. Die Werte c_1, c_2 und c_3 werden genutzt, um den mittleren Offset der Messdaten zu bestimmen (2.11). Diese Werte sind in Abbildung 2.3(b) grafisch veranschaulicht.

$$x_c = -\frac{c_1}{2} \qquad y_c = -\frac{c_2}{2} \qquad r = \sqrt{\frac{c_1^2 + c_2^2}{4} - c_3} \tag{2.11}$$

2.1.4 Digitale Filter

Allgemein dient ein Filter in der Signalverarbeitung zum Entfernen unerwünschter Messsignalanteile, also gewisser Frequenzbereiche. Dazu zählt zum Beispiel hochfrequentes, (weißes) Rauschen oder das 50Hz Netzbrummen. Ein besonderer Vorzug des in dieser Arbeit entwickelten Testsystems ist, dass die Filterkoeffizienten-Matrizen einfach implementiert werden können. Diese können zuvor mit Hilfe eines anderen Programms, wie MATLAB, berechnet werden. Neben den Digitalfiltern gibt es noch die Analogfilter, welche aus passiven Bauelementen, wie Widerständen, Spulen und Kondensatoren oder

aktiv mit Operationsverstärkern aufgebaut werden. Digitalfilter hingegen werden mit Logikbausteinen wie FPGAs oder in Form von Programmcode auf einem Signalprozessor realisiert (detailliertere Informationen finden Sie in der Literatur [22]).

Der erste Schritt des Filterentwurfs umfasst das Bestimmen des relevanten Frequenzbereichs. Erst im Anschluss können Grenzfrequenzen berechnet und ein Filter entwickelt werden. Zur Verfügung stehen ideale Filterfunktionen von Tiefpass, Hochpass, Bandpass und Bandsperre. Für die Filterung der Eingangssignal-Matrixen werden entsprechend große Filtermatrizen benötigt. Für die einfache Berechnung einer solcher Matrix gilt

$$\mathbf{h_{2i}} = \mathbf{h_{1i}} \cdot \mathbf{w_i} \tag{2.12}$$

wobei $\mathbf{h_{2i}}$ die Filterterkoeffizienten-Matrix, $\mathbf{h_{1i}}$ eine ideale Filterkoeffizienten-Matrix und $\mathbf{w_i}$ eine Fensterfunktions-Matrix ist [19]. Das Beispiel aus Gleichung (2.14) bescheibt ein Hammingfenster.

$$\Omega_C = \frac{\omega_C}{f_S} \qquad n : 0...N \qquad N : AnzahlSchritte$$

$$\mathbf{h_1}(i) = \begin{cases} \dfrac{\Omega_C}{\pi} & n = \dfrac{N-1}{2} \\ \dfrac{\sin\left(\left(n - \dfrac{N-1}{2}\right) \cdot \Omega_C\right)}{\left(n - \dfrac{N-1}{2}\right) \cdot \pi} & sonst \end{cases} \tag{2.13}$$

$$\mathbf{w}(i) = 0.54 + 0.46 \cdot \cos\left(\frac{2 \cdot \pi \cdot \left(i - \dfrac{N-1}{2}\right)}{N}\right) \tag{2.14}$$

2.1.5 CORDIC-Algorithmus

Bereits zu Beginn der Entwicklung erster umfangreicher, digitaler Rechensysteme wurde sich Gedanken über die Implementierung mathematischer Funktionen gemacht. Komplexe Funktionen, wie Exponential-, Wurzel-, trigonometrische Funktionen oder Logarithmen, waren Anfangs nicht realisierbar. Einige aufwändig konstruierte mechanische Rechenmaschinen, wie die Curta I (1948), konnten Wurzeln ziehen [3]. Als ab Mitte des 20. Jahrhunderts die Produktionszahl der elektromechanischen Rechenmaschinen wuchs, waren die Geräte Aufgrund des Platzbedarfs für die Rechenwerke immer noch sehr groß. Komplexe Funktionen wurde zu der Zeit noch mittels Polynom-Approximation (Potenz-, Taylor-Reihe) oder Look-Up-Tabellen berechnet [18]. Um nun die Größe der Rechenwerke zu minimieren und die Geschwindigkeit der Maschinen zu steigern, entwickelte J.E. Volder 1959 den CORDIC-Algorithmus (COordinate Rotation DIgital Computer) [21].

Die Algorithmen arbeiten ausschließlich mit schnellen Operationen, also Additionen und Multiplikationen mit Zweier-Potenzen, die in Hardware einfach umgesetzt werden können [1]. Mit einem Berechnungsbeispiel soll die Funktionsweise des CORDIC erklärt werden.

Die Drehung eines Vektors (x_0, y_0) um einen Winkel Θ wird durchgeführt und es soll der Endpunkt (x_n, y_n) bestimmt werden. Gleichung (2.15) stellt die Berechnung mit Hilfe einer Rotationsmatrix dar. Der in Abbildung 2.4(a) dargestellte Winkel soll mit Hilfe des CORDIC bestimmt werden. Durch Drehung des Vektors $(1, 0)$, gezeigt in Gleichung (2.16), um den Winkel Θ, wie in Abbildung 2.4(b) dargestellt, können $\sin\Theta$ und $\cos\Theta$ berechnet werden.

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \tag{2.15}$$

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.16}$$

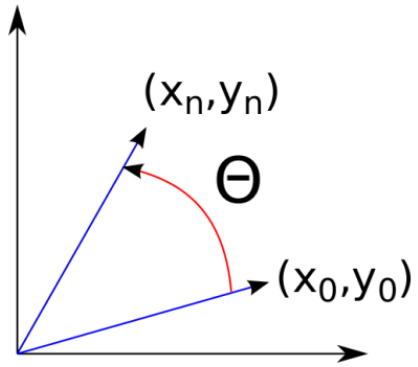
Mit der Umformung $\cos\Theta = \dfrac{1}{\sqrt{1+\tan^2\Theta}}$ ergibt sich für die Rotationsmatrix eine alleinige Abhängigkeit von der Winkelfunktion $\tan\Theta$ (2.17). Die Drehung um den Winkel Θ wird mit Teilwinkeln α_i realisiert. Diese sind vorab berechnet worden, um schnelle Operationen zu ermöglichen. Durch angepasste Vorzeichen $\sigma_i \in \{-1, 1\}$ werden die Teilwinkel entsprechend addiert oder subtrahiert (2.18). Somit wird der zu berechnende Winkel durch eine alternierende Approximation angenähert.

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \frac{1}{\sqrt{1+\tan^2\Theta}} \cdot \begin{pmatrix} 1 & -\tan\Theta \\ \tan\Theta & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \tag{2.17}$$

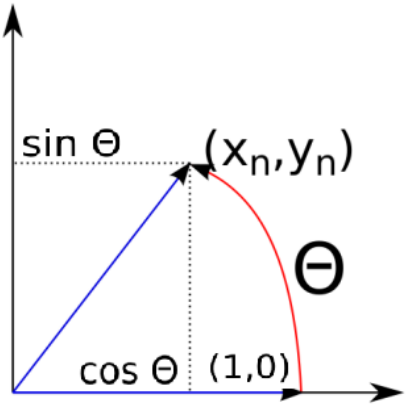
$$\Theta = \sum_{i=0}^{N-1} \sigma_i \cdot \alpha_i \tag{2.18}$$

Als Hilfsvariable zur Winkelberechnung wird z_i eingeführt. Gleichung (2.19) beschreibt, dass sich der resultierende Winkel aus den vorigen Teilwinkeln α_i, multipliziert mit dem Vorzeichenfaktor σ_i, ergibt. Dieses Verfahren wird ebenfalls in Abbildung 2.4(c) und als sukzessive Approximation in Abbildung 2.4(d) deutlich.

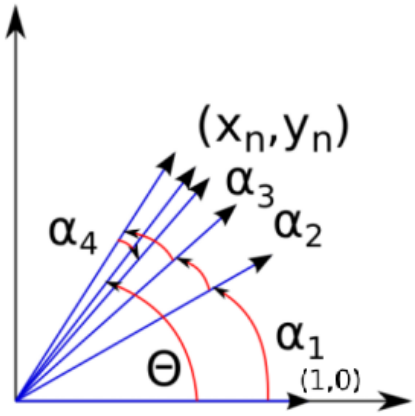
$$z_{i+1} = z_i - \sigma_i \cdot \alpha_i \qquad \sigma_i = \begin{cases} 1 & z_i \leq 0 \\ -1 & z_i \geq 0 \end{cases} \tag{2.19}$$



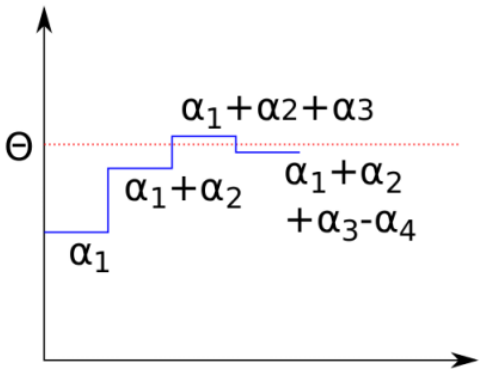
(a) Anfangswinkel, dessen Koordinaten berechnet werden sollen.



(b) Drehung des Vektors $(1,0)$ um den Winkel Θ.



(c) Iteration der Winkelberechnung.



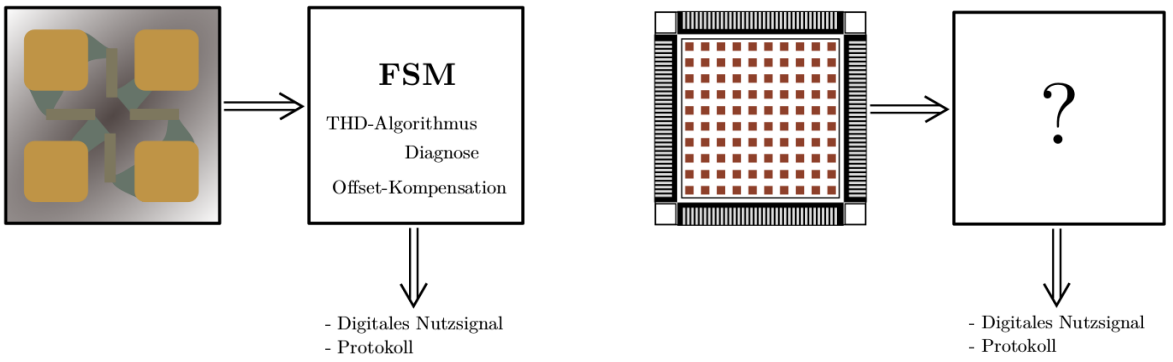
(d) Verdeutlichung der Iteration zur Winkelbestimmung.

Abbildung 2.4: Darstellung der Funktionsweise des CORDIC-Algorithmus Anhand einer Winkelbestimmung [13].

2.2 Funktion des Sensor-Arrays

Derzeitig sind in den meisten Anwendungen von Magnetsensoren, wie ABS-Sensoren oder Rotorlagegebern, nur einzelne Sensoren verbaut. Für sicherheitskritische Anwendungen ist ein einzelner Sensor jedoch nicht ausreichend, da kein Vergleich der Genauigkeit möglich ist. Das System ist deutlich anfälliger für Störungen. Um dem Problem entgegenzuwirken, sollen in diesem Projekt mehrere Sensoren zu einer $N \times N$ Matrix zusammengeschaltet werden. Einen Vergleich der bisherigen (Abbildung 2.5(a)) und der zukünftigen (Abbildung 2.5(b)) Sensoren zeigt die Abbildung 2.5.

Die Sensormatrix, die für diese Bachelorarbeit verwendet wird, ist eine 8×8 Matrix. Jeder Sensor gibt einen Sinus- und Cosinuswert aus, anhand dessen der derzeitige



(a) Einzelner Sensor mit Signalverarbeitung.

(b) Komplexes Sensor-Array mit noch undefinierter Verarbeitung.

Abbildung 2.5: Schematische Abbildung eines Sensors (derzeitige Anwendung) im Vergleich zur Sensormatrix (neue Sensorgeneration). Für die neue Generation sind die Verarbeitungsmethoden noch in den Entwicklungsprozessen.

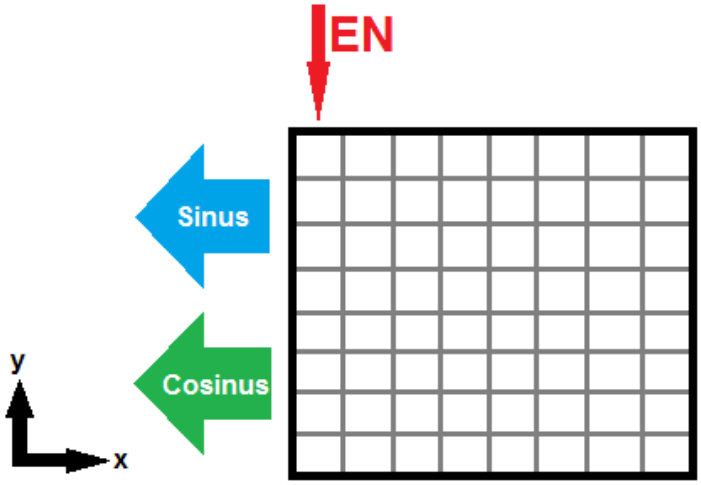


Abbildung 2.6: Schematische Darstellung der Funktionsweise der Sensormatrix.

Winkel des Gebermagnetfeldes bestimmt werden kann. Zum Einlesen der Werte werden die Enable-Leitungen vom Controller in einer Schleife von eins bis acht durchlaufend auf High-Pegel gesetzt (Y-Richtung), siehe Abbildung 2.6. Nach dem Einschwingvorgang werden die Werte (X-Richtung) vom Mikrocontroller ausgelesen und verarbeitet. Durch dieses Verfahren kann spaltenweise jeder Sensorwert ermittelt und anschließend gespeichert werden. Für die Signalverarbeitung wird anschließend, wie in Kapitel 2.1 beschrieben, die Wertematrix verwendet. Da in dieser Bachelorarbeit Winkelsensoren zum Einsatz kommen, wird ein Vergleich zwischen einem einzelnen Sensor und der Sensormatrix in nachfolgender Tabelle 2.1 festgehalten.

Tabelle 2.1: Vergleich von bisherigen und zukünftigen Sensorgenerationen an Hand von Winkelsensoren.

Sensor mit 2 oder 4 Messbrücken	Sensor-Array mit $N \times N$ Sensoren
• Winkelberechnung aus Brückensignalen (Sinus und Cosinus) • Feld des Gebermagneten möglichst homogen • Präzise Lageausrichtung Sensor/Magnet ist ergebnisrelevant • Wenig Information über Lagefehler • Unterscheidung Nutz- und Störsignal ist problematisch	• Winkelberechnung aus $N \times N$ Signalen • Unterschiedliche Magnetisierungen und Geometrien für den Gebermagneten • Weite Toleranzen beim Einbau des Sensors • Lageberechnung und Korrektur durch höheren Informationsgehalt ermöglicht • Trennung von Nutz- und Störsignal wird angestrebt

Anhand der Tabelle 2.1 wird deutlich, welche Vorteile ein Sensor-Array, besonders in den vorgesehenen Einsatzgebieten in der Automobilelektronik, bietet. Hervorzuheben sind die Toleranzen beim Einbau, die es trotz einer Verschiebung von Sensor und Magnetfeld ermöglichen, eine Lageberechnung und Fehlerkorrektur durchzuführen. Um diesen Prozess noch zu verbessern, wird auf eine genaue Trennung von Nutz- und Störsignal hingearbeitet. Abbildung 2.7 zeigt einen einzelnen, herkömmlichen Sensor und die Ausgangssignale einer 360° Drehung des Gebermagneten. Dieses Prinzip wird bereits vielfach in der Industrie angewandt. Diese Arbeit verwendet das in Abbildung 2.8 schematisch dargestellte Sensor-Array und dessen Ausgangswerte. Hierbei handelt es sich nun um ein Ausgangsfeld mit den jeweiligen Sinus- und Cosinuswerten H_x und H_y jedes Sensors.

2.2.1 Magnetische Sensoren

Die ersten magnetischen Messgeräte[1] sind bereits seit Jahrhunderten im Einsatz. Die früheste Erfindung war der Kompass. Erste magnetoresistive Effekte hingegen beschrieb

[1]Der Begriff „Sensor“ etablierte sich erst vor circa 30 Jahren, als Messfühler immer mehr in Konsumanwendungen (z.B. Kraftfahrzeug- und Haustechnik) etabliert wurden.

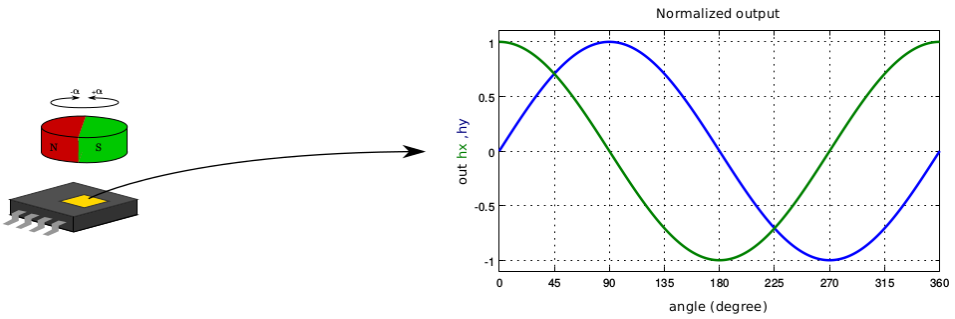


Abbildung 2.7: Ausgangssignale von Sinus und Cosinus bei einer 360° Drehung eines Gebermagneten über dem Sensor.

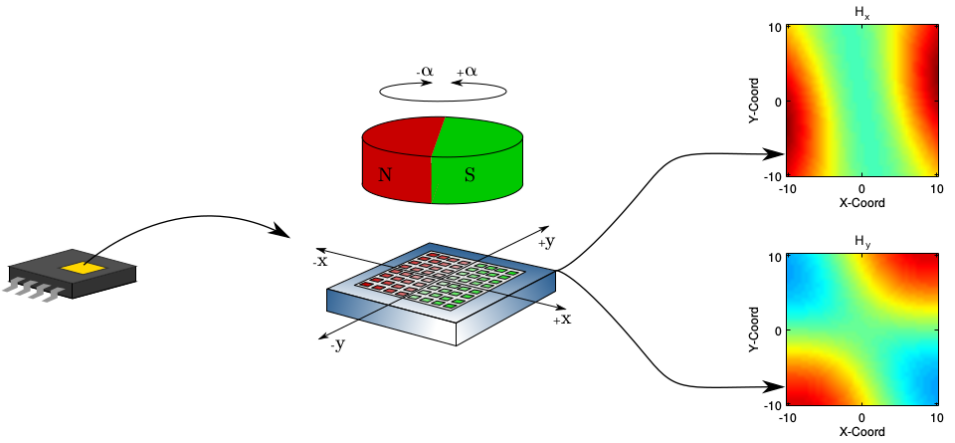


Abbildung 2.8: Ausgangssignalfeld der Sinus- und Cosinus-Signale bei einer 360° Drehung eines Gebermagneten über dem Sensor-Array.

Lord Kelvin im Jahre 1856 [24]. Damals wurden Messgeräte noch ausschließlich genutzt, um das Erdmagnetfeld zu detektieren und später auch zu messen. Die heutigen Einsatzgebiete sind um ein Vielfaches umfangreicher geworden. Jedes Einsatzgebiet stellt andere Anforderungen an Bauform, Größe des Sensors und mögliches Messprinzip. Besonders mit der Erfindung neuer Messprinzipien wurden die Sensoren stetig komplexer in ihrem Aufbau. Trotz der wachsende Komplexität und reduzierter Größe der Sensoren blieben jedoch die Anforderungen an Störanfälligkeit und Stabilität bestehen, was selbst mit heutigen Produktionsmöglichkeiten immer wieder neue Herausforderungen darstellt.

Heute werden etwa 50% aller produzierter Sensoren für die Automobilindustrie hergestellt. Dabei kommen immer häufiger sogenannte „smarte Sensoren“ zum Einsatz. Diese sind als Mikrochip entwickelt und beinhalten in ihrem Programmcode bereits eine teils sehr umfangreiche Signalverarbeitung und Fehlerkorrektur [14, S. 17]. Von den vielen Sensoren zur Erfassung von magnetischen Feldern sind Hall-Sensoren sowie Anisotropic Magneto-Resistance (AMR)-, Giant Magneto-Resistance (GMR)- und TMR-Sensoren

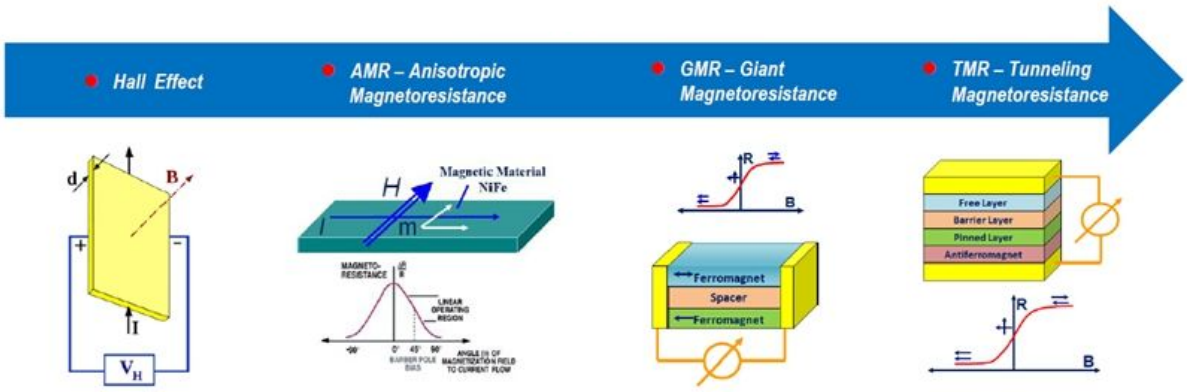


Abbildung 2.9: Entwicklungsherarchie der bekannesten Magnetfeldsensoren. Die jüngste Erfindung ist der TMR [20].

Tabelle 2.2: Vergleich von Sensorspezifikationen vier bedeutender Magnetfeldsensoren [20].

	Hall	**AMR**	**GMR**	**TMR**
Verbrauch (mA)	5~20	1~10	1~10	0.001~0.01
Fläche (mm_2)	1×1	1×1	1×2	0.5×0.5
Sensitivität ($mV/V/Oe$)	~0.05	~1	~3	~100
Dynamic Range (Oe)	~10000	~10	~100	~1000
Auflösung ($nT/Hz_{\frac{1}{2}}$)	>100	0.1~10	1~10	0.1~10
Temperaturbereich ($°C$)	<150	<150	<150	<200

aufgrund der technischen Fortschritte besonders hervorzuheben. Eine Veranschaulichung der Entwicklungsgeschichte ist in Abbildung 2.9 dargestellt. Die in diesem Projekt verwendeten magnetischen Sensoren AMR und TMR werden nachfolgend hinsichtlich ihres Aufbaus und Messprinzips erläutert.

Die Tabelle 2.2 zeigt auf, nach welchen Kriterien Sensoren ausgewählt werden. Für die spezifischen späteren Anwendungen im Kraftfahrzeug (KFZ) sind neben Stromverbrauch, die Größe und ins Besondere - wegen des Einsatzes in sicherheitskritischen Anwendungen - auch die Auflösung und Sensitivität des Sensors. Bei der Überlegung der Zusammenschaltung als 8×8 oder 15×15 Sensor-Array ist die Gesamtleistung ein sehr wichtiger Punkt. Der angegebene Wert aus der Tabelle muss demnach mit 64 oder mit 225 multipliziert werden. Erste Versuche wurde Anfangs mit dem AMR-Sensor und schließlich mit dem deutlich energieeffizienteren TMR-Sensor durchgeführt.

Ein Vorzug des TMR-Sensors wird in Abbildung 2.10 deutlich, wo die Ausgangssignale verschiedener Magnetfeldsensoren aufgezeichnet sind. Die Spannungsamplitude ist im

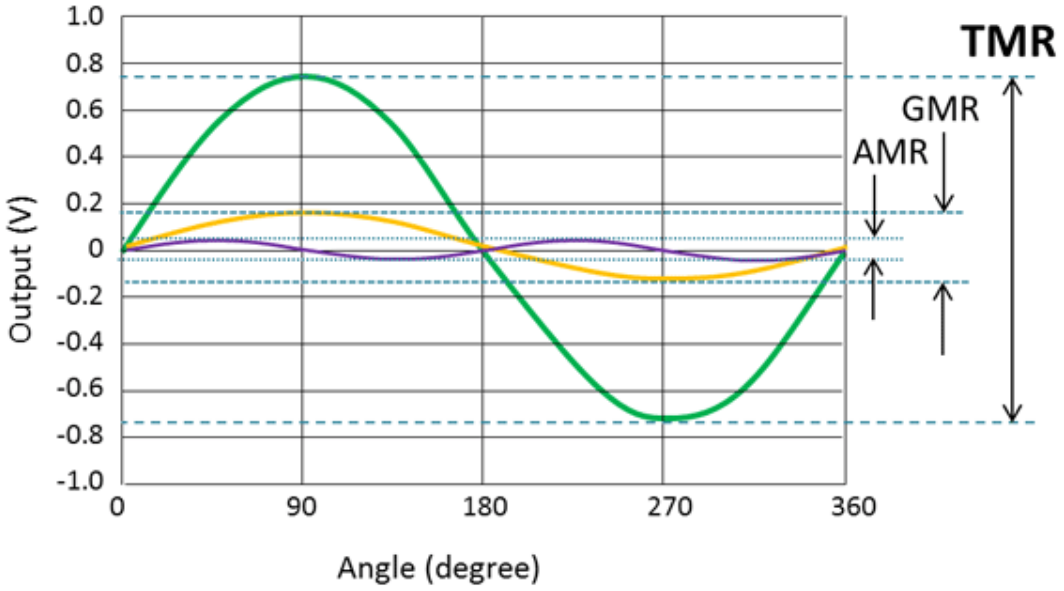


Abbildung 2.10: Ausgangssignale des TMR-, GMR- und AMR-Sensors im Laufe einer 360° Drehung des Gebermagneten [5].

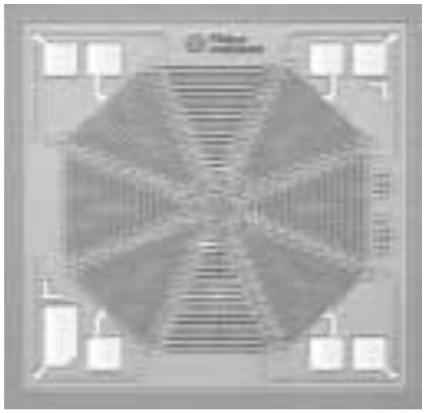
Gegensatz zum AMR-Sensor fast 20 mal größer. Dies bietet eine deutlich erhöhte Störfestigkeit, Stabilität und erfordert keine Nachverstärkung. Wie diese große Amplitude zustande kommt, wird in dem Unterkapitel 2.2.3 ausfühlich erläutert.

2.2.2 AMR Sensor

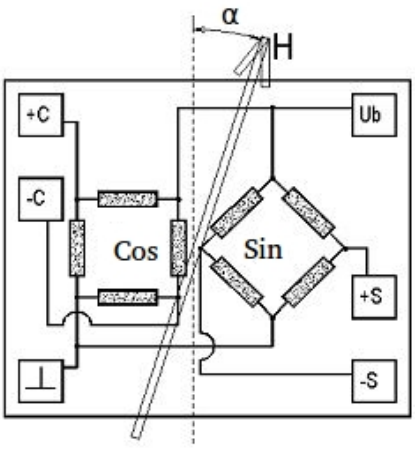
Der anisotrope magnetoresistive Effekt beruht auf anisotroper (von der Raumrichtung abhängiger) Streuung. Der Effekt lässt sich sehr gut in einer dünnen Schicht (ca. 20 nm) aus Permalloy[2] beobachten. Der elektrische Widerstand der Schicht ist vom äußeren Magnetfeld abhängig. Nur Magnetfeldkomponenten in der Schichtebene haben dabei einen merklichen Einfluss auf den Widerstand. Wenn das äußere Magnetfeld in der Stromrichtung oder gegen die Stromrichtung gerichtet ist, hat der Widerstand den größten Wert. Den kleinsten Widerstand misst man, wenn das äußere Magnetfeld senkrecht zur Stromrichtung in der Schichtebene gerichtet ist [10].

Um nun den reinen Winkeleffekt zu erhalten, muss eine AMR-Schicht bei ihrer Sättigungsfeldstärke betrieben werden, d.h. der interne Magnetisierungsvektor $\vec{m}$ richtet sich parallel zum äußeren Magnetfeld aus. In dieser Betriebsart arbeitet der Sensor als Winkelmesssystem und wird als Wheatstone-Brückenschaltung realisiert. Abbildung 2.11 zeigt das Foto 2.11(a) eines AMR-Winkelsensors und ein Schaltbild 2.11(b), jeweils bestehend aus zwei um 45° verdreht angeordneten Wheatstone-Messbrücken [4, S. 274].

[2]Eine Legierung aus Nickel (81 %) und Eisen (19 %).



(a) Röntgenbild des AMR-Sensor-Aufbaus. Die acht mäanderförmigen Wheatstone-Messbrücken sind je um 45° verdreht [4].



(b) Schaltplandesign eines AMR-Winkelsensors [16] mit Messfeld H und resultierendem Winkel α.

Abbildung 2.11: Röntgenbild und Schaltplan eines AMR-Winkelsensors zeigen den genauen Aufbau der um 45° gedrehten Wheatstonebrücken.

AMR-Sensoren als Winkelmesser - genauer Feldrichtungsmesser - werden im Sättigungsbereich betrieben. Somit ist das Messprinzip weitgehend unabhängig von Montagetoleranzen, Magnettoleranzen sowie alterungs- und temperaturbedingten Magnetänderungen. Mit den Ausgangssignalen des Sensors aus Gleichung (2.20) lässt sich der Winkel gemäß Gleichung (2.21) bestimmen. Was jedoch zu beachten ist und einen großer Unterschied zwischen den AMR- und den GMR-\TMR-Sensoren darstellt, ist der verminderte Messbereich des AMR-Sensors. Deieser ist über die 360° Drehung eines Gebermagneten doppel-periodisch, das heißt der Sensor misst nur im Bereich 0° bis 180°, was in der Abbildung 2.10 zu erkennen ist (lila Signal).

$$u_1 = \hat{u}_1 \cdot \sin(2\alpha) \qquad u_2 = \hat{u}_2 \cdot \cos(2\alpha) \tag{2.20}$$

$$\alpha = \frac{1}{2} \cdot \arctan\left(\frac{u_1}{u_2}\right) \tag{2.21}$$

2.2.3 TMR Sensor

Der tunnel-magnetoresistive Effekt ist ein Phänomen, welches ausschließlich mit der Quantenmechanik erklärt werden kann (zur Erklärung wird die Abbildung 2.12 herangezogen). Da bei Schichtdicken im Nanometerbereich Isolationsschichten für Elektronen durchlässig werden, verschwimmt hier die Grenze zwischen Leitern und Nichtleitern. Dieses Durchwandern wird als „Tunneleffekt“ bezeichnet. Oftmals ein unerwünschtes

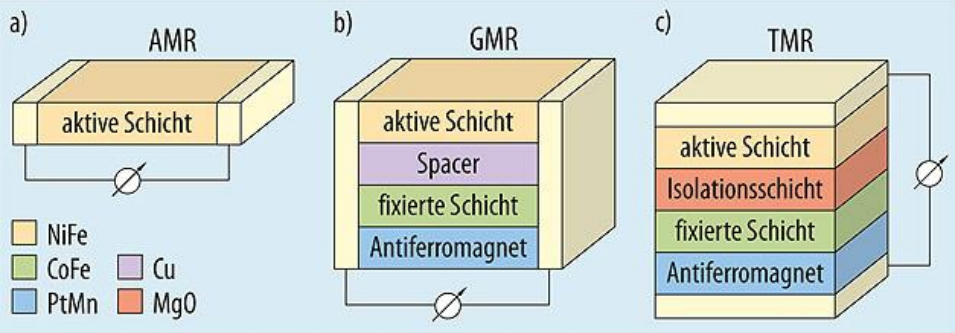


Abbildung 2.12: Physikalischer Aufbau wichtiger XMR-Sensoren und deren Betriebsart (Anschluss einer Betriebsspannung) [9].

Problem, macht es diese neue Sensorgeneration erst möglich. Es ändert sich der elektrische Widerstand, wenn der Sensor einem magnetischen Feld ausgesetzt wird. Zwischen den ferromagnetischen Schichten (gelb und grün) befindet sich die bereits erwähnte Isolationsschicht (rot). Die unterste (blaue) Antiferritschicht ist notwendig, um die Magnetisierung der untersten Ferritschicht zu fixieren. Die Obere folgt dem äußeren Feld. Die Durchlässigkeit der Isolationsschicht - und damit einhergehend der Widerstandswert - hängt nun vom Winkel zwischen den Magnetisierungsrichtungen der Ferritschichten ab. Bei paralleler Ausrichtung der Felder ist der Widerstand am kleinsten, bei antiparalleler am Größten. Die erreichbare Widerstandsänderung $\frac{\Delta R}{R}$ ist aufgrund der vertikalen Stromflussrichtung durch die Schichten um ein Vielfaches größer als beim GMR. Bei Raumtemperatur wurden Änderungen bis zu 600% gemessen. Unter Laborbedingungen und bei einer Betriebstemperatur von -170°C sind Änderungen von über 1000% gemessen worden. Dies führt zu höheren Signalamplituden (siehe Abbildung 2.10), welche somit wenig Nachverstärkung benötigen. Der Grundwiderstand lässt sich außerdem von einigen Ohm bis Megaohm variieren, wodurch der Sensor für verschiedene Anwendungen angepasst werden kann [9].

2.3 Verfügbare Hard- und Software aus Vorarbeiten

Zu Beginn der Projektentwicklung ist eine AMR-Sensormatrix entwickelt worden. Abbildung 2.13 zeigt das Tiva Board mit der angeschlossenen AMR-Matrix. Die Controlleransteuerung wurde von Herrn M. Sc. Thorben Schüthe entwickelt und implementiert. Damit gibt es die Möglichkeit über das Terminal den Befehl „*-test*“ an den Controller zu senden und anschließend die Rohdaten der 8×8 Matrizen für den Sinus- und

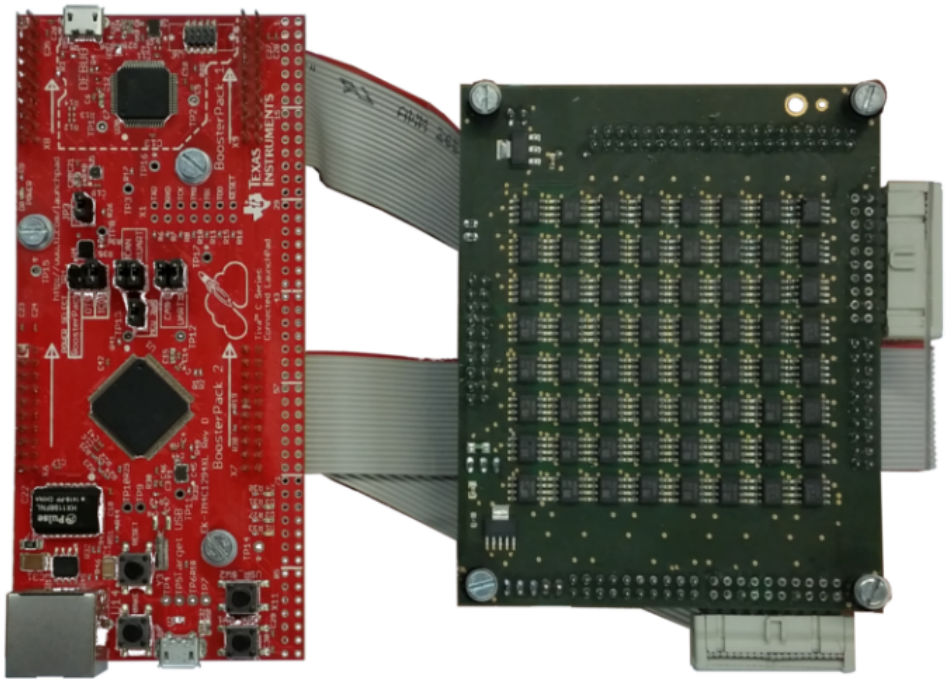


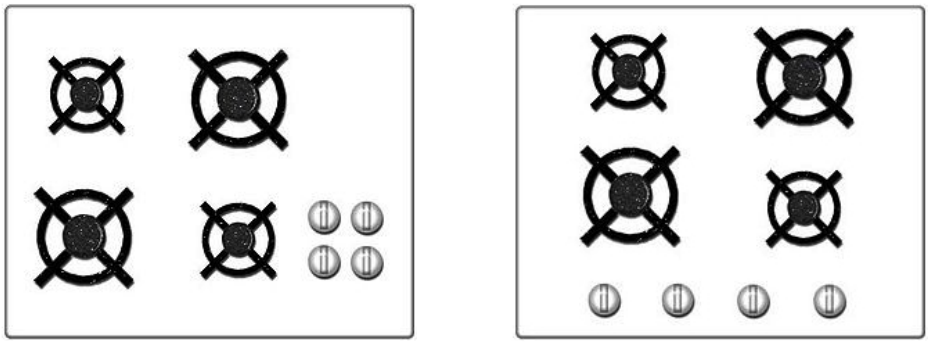
Abbildung 2.13: Ertser Hardwareaufbau des Controllerboards mit einer AMR-Sensormatrix.

Cosinusanteil zu erhalten. Diese Werte können mit Octave weiterverarbeitet und visualisiert werden. Auch hier hat Herr Schüthe erste Testskripte entworfen. Die Entwicklung einer TMR-Sensormatrix wurde in der Bachelorarbeit "Tunnel-Magnetoresistives Sensor-Array - Controllersteuerung, Platinen-Layout und Prüfstandserprobung" von Herrn B. Sc. Abraham Begic behandelt. Er entwickelte eine neue Platine mit 64 TMR-Sensoren, passte die Ansteuerung für die TMR-Sensoren an und führte erste Testmessungen durch. Diese neue TMR-Matrix mit Ansteuerung soll nun als Grundlage für diese Arbeit dienen.

2.4 Einführung in das Interfacedesign

Die Steuerung des Mikrocontrollers und die Bedienung der Ausgabeoptionen werden über eine grafische Oberfläche getätigt. Für die Entwicklung dieser Schnittstelle zwischen Mensch und Computer hat Donald Norman, Professor für Kognitionswissenschaften in Kalifornien, sechs Richtlinien aufgestellt. Von diesen sechs Richtlinien werden die für dieses Projekt besonders relevanten kurz erläutert [11]. In Kapitel 5.1 wird genauer auf die Nutzung dieser Richtlinien bei der Entwicklung der grafischen Bedienoberfläche eingegangen.

- **Visibility** ist ein wichtiges Grundprinzip, denn je sichtbarer Objekte bzw. Funktionen sind, desto schneller werden sie vom Benutzer wahrgenommen und erkannt.



(a) Gutes Mapping: Die Zuordnung von Schaltfläche und Kochstelle ist direkt zu erkennen.

(b) schlechtes Mapping: Die Zuordnung von Schaltfläche und Kochstelle ist nicht sofort ersichtlich.

Abbildung 2.14: Vergleich von gutem und schlechtem mapping [15].

Die Kernfunktionen sollten also direkt zu sehen sein, um den Benutzer mit den Bedienmöglichkeiten vertraut zu machen.

- **Mapping** bezieht sich auf die Verbindung zwischen der Nutzung und dem resultierenden Effekt. Ein Vergleich zeigt Abbildung 2.14, wobei die Zuordnung der Bedienelemente zu den vier Kochstellen bei einem guten Mapping 2.14(a) sofort erkennbar ist.
- **Affordance** beschreibt die natürliche, gewohnte Nutzung von bekannten als auch unbekannten Funktionen und Objekten. Ein einfaches Beispiel ist die Nutzung eines Button, der von einem Nutzer geklickt werden kann. Die intuitive Benutzbarkeit der implementierten Funktionen sollte möglichst hoch gehalten werden.

3 Entwicklung des Gesamtkonzepts

Im Anschluss an die Einführung in die mathematischen- und signalverarbeitungstechnischen Grundlagen wird in diesem Kapitel die konkrete Konzeptentwicklung beschrieben. Beginnend bei den ersten Entwurfsideen werden die Vor- und Nachteile der jeweiligen Systeme, hinsichtlich des späteren Nutzen als kompaktes Demonstrationssystem, beleuchtet. Dieser Entwicklungsprozess ist in Unterkapitel für die jeweiligen Hard- und Softwarebestandteile des Gesamtsystems gegliedert.

3.1 Systementwurf der Plattformen und Auswahl der Baugruppen

Zu Beginn der Entwicklung wurden die Anforderungen festgehalten, die das Sytem im Laufe der Arbeit erfüllen soll. Der Zweck des Demonstrationskoffers ist es, die in Kapitel 2.2 diskutierten Vorteile des Sensor-Arrays zu demonstrieren und anschaulich zu visualisieren. Dabei müssen die aus Vorarbeiten (siehe Kapitel 2.3) bereits vorhandenen Komponenten eingebunden werden. In Abbildung 3.1 ist der erste Systementwurf dargestellt. Nachfolgend werden die einzelnen Komponenten und die Entwicklung des Gesamtsystems im Projektverlauf erklärt.

Die Sensormatrix ist via Pinleiste mit dem Tiva-Board TM4C1294 verbunden. Eine Abbildung des Tiva-Boards findet sich im Anhang D.1. Für die grafische Ausgabe der

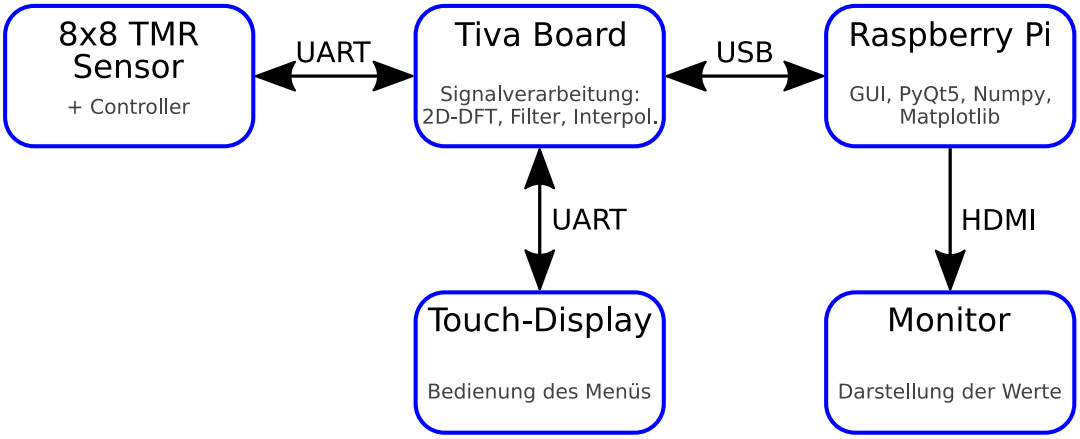


Abbildung 3.1: Erster Konzeptentwurf mit Touch-Display und Schnittstellen der einzelnen Komponenten.

Tabelle 3.1: Gegenüberstellung der Vor- und Nachteile eines Raspberry Pi.

Vorteile	Nachteile
• Viele Anschlussmöglichkeiten • Beste Anbindungsmöglichkeiten für Python-Anwendungen • Schnelles System für einfache grafische Anwendungen • Großer Arbeitsspeicher von 1GB • Quad-Core Prozessor • Kompaktes System (lässt sich überall einbauen) • Große Menge und Qualität von Internetquellen (Foren, Anleitungen, Beispiele)	• Geringe Geschwindigkeit im Vergleich zum MC bezüglich der Signalverarbeitung • Geringe Zahl an Bibliotheken (im Gegensatz zu Ubuntu) • Spätere Verfügbarkeit von Programmupdates

Messwerte wird als Plattform ein Raspberry Pi 3 genutzt. Die Vor- und Nachteile sind in Tabelle 3.1 zusammengefasst. Der Raspberry Pi bietet ausreichend Rechenleistung, Speicherplatz sowie die notwendigen Ausgänge zum Anschluss der Peripheriegeräte. Ein USB-Port wird für die Anbindung des Mikrocontrollers und zwei weitere für Maus und Tastatur benötigt. Der HDMI-Ausgang wird mit einem für den Koffer passenden Monitor verbunden. Des Weiteren bietet der Raspberry Pi mit dem Betriebssystem Raspbian und der Programmiersprache Python viele Bibliotheken zur Entwicklung einer grafischen Benutzeroberfläche und Darstellungsmöglichkeiten der Messwerte. Die in der Tabelle 3.1 aufgelisteten Nachteile fallen für dieses Projekt kaum ins Gewicht, da der Raspberry Pi nicht für Echtzeitanwendungen eingesetzt werden soll. Auf die Auswahl der Software und Schwierigkeiten mit Updates und Programmversionen wird im nachfolgenden Unterkapitel 3.2 vertiefend eingegangen.

Neben dem Sensor-Array sollte gemäß dem ursprünglichen Projektentwurf ein Touch-Display an die Pins des Mikrocontrollers angeschlossen werden. Auf diesem sollten die Bedienelemente zur Steuerung des Mikrocontrollers dargestellt werden. Der Vorteil eines Touch-Displaysist ,dass die Maus und die Tastatur als Peripheriegeräte entfallen würden. An dem Tiva-Board stehen derzeit nicht genügend Pins zur Verfügung, um ein Touch-Display anschließen zu können. Perspektisch werden noch welche frei, wenn die Arbeit

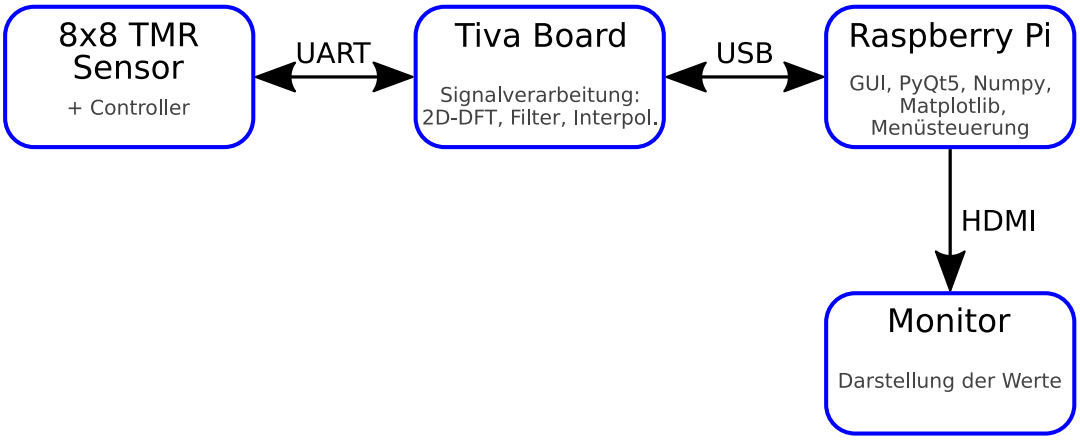


Abbildung 3.2: Finaler Konzeptentwurf ohne Touch-Display und mit Schnittstellen der einzelnen Komponenten.

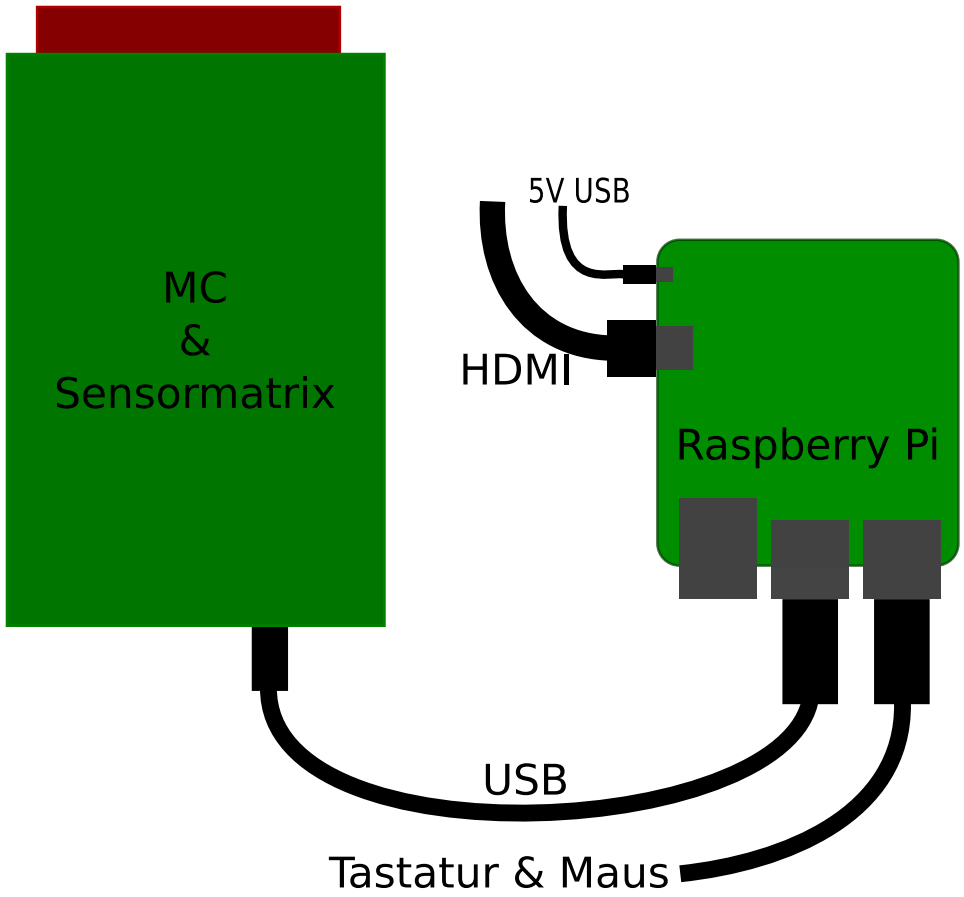


Abbildung 3.3: Schematischer Aufbau des Konzeptentwurfes mit Angabe der Verbindungen und Komponenten

von Herrn Begic zum Entwurf einer TMR-Sensor-Matrix abgeschlossen wird. Im Zweifelsfall aber eine AMR-Sensormatrix eingesetzt werden soll, kann kein Touch-Display verbaut werden. Die AMR-Matrix muss an die 20-Pin-Boosterpacks des Tiva-Boards, siehe Zeichnung D.1 im Anhang, angeschlossen werden. Die TMR-Matrix wird an das 42-Pin-Breadboard angeschlossen. Abbildung 3.2 zeigt den veränderten und finalen Entwurf des Gesamtsystems. Somit kann ein schematischer Aufbau entworfen werden, siehe Abbildung 3.3, der in die Realität umgesetzt werden kann.

3.2 Auswahl der Software-Toolchain

Unter dem Begriff *Toolchain* wird im allgemeinen Sinne die „Werkzeugkette“ bezeichnet, die benötigt wird, um Programmcode ausführbar zu machen. Eine „native“-Toolchain gibt dabei an, dass die Maschine, auf der der Code gebaut wird, dieselbe ist, auf der dieser später ausgeführt wird. In dieser Arbeit kommt die „cross“-Toolchain zur Anwendung, wo das Target eine andere Maschine ist [2]. Für die Subsysteme MC und Raspberry Pi sind in Abbildung 3.4 die Toolchains aufgezeigt. Auf dem Raspberry Pi wird der Code geschrieben, compiliert und ausgeführt. Der Code für den MC hingegen wird auf einem separaten Computer entwickelt, dort compiliert und zum Laufen auf den MC geflasht. An erster Stelle der Toolchain-Entwicklung steht die Wahl einer geeigneten Programmiersprache. Zur Auswahl stehen unzählige Programme, Softwarepakete und Programmiersprachen mit Fokus auf bestimmte Anwendungsgebiete. Zur Signalverarbeitung eignet sich in besonderem Maße die Programmiersprache C. Mit C lässt

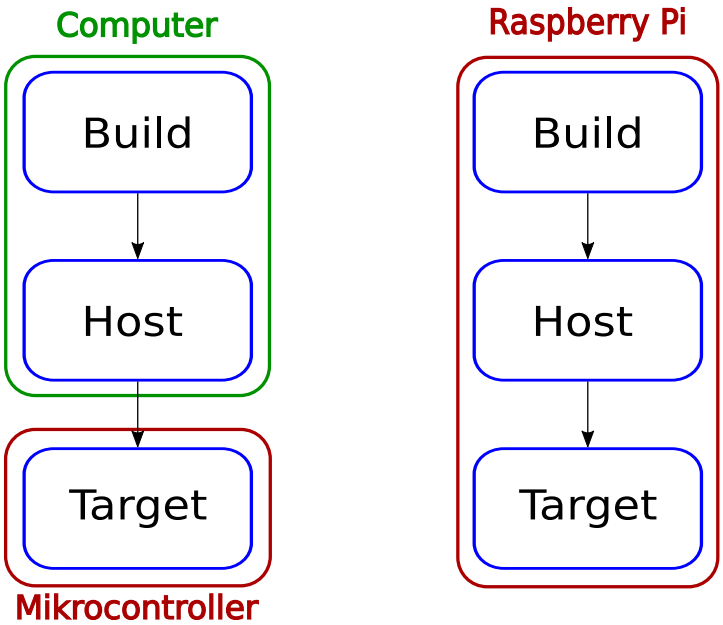


Abbildung 3.4: Toolchain für die Entwicklung der Mikrocontroller-Steuerung (links) und Toolchain für die Entwicklung der GUI (rechts).

sich hardwarenah auf den meisten Mikrocontrollern programmieren und es bietet viele Bibliotheken mit spezifischen Funktionen. Außerdem ist die Ansteuerung der Sensor-Matrix bereits in C programmiert. Dieser Code wird im Laufe dieser Arbeit erweitert (siehe Kapitel 3.3.1). Ein weiterer Punkt, der für C spricht, ist die lange Existenz. C wurde bereits 1972 von Dennis Ritchie entwickelt und verbreitete sich fortan über die ganze Welt [17]. Daher gibt es Unmengen an Programmbeispielen und Hilfen zur Entwicklung im Internet. Als Entwicklungsumgebung wird das Code Composer Studio v6 (CCS) eingesetzt. Dieses ist ein Integrated Development Environment (IDE) aus dem Hause Texas Instruments (TI), um Anwendungen für die eingebetteten Prozessoren zu entwickeln. Für das Projekt kann der eingesetzte MC in den Einstellungen ausgewählt werden, um passende Bibliotheken und Hardwareinformationen in das Projekt zu implementieren. Der fertige Programmcode kann compiliert werden, anschließend auf das Board geflasht und ausgeführt werden.

Anfänglich sollte GNU Octave, eine MATLAB nahe open-source Software, auf dem Raspberry Pi installiert werden, um den bereits vorhandenen Code zum empfangen von Sensorwerten zu übernehmen. Für den Raspberry Pi gab es jedoch nicht die aktuellste Version von Octave und damit nicht alle benötigten Pakete[1]. Die aktuelle Version von Octave gibt es derzeit nur für eine instabile Version des Betriebssystems Raspbian. Für die Aktivierung werden in der Datei `sources.list` im Ordner `/etc/apt/` die nachfolgende Änderungen vorgenommen. Der mit `# Stable` markierte Text wird auskommentiert.

```
# Stable
deb http://ftp.debian.org/debian stable main contrib non-free
deb-src http://ftp.debian.org/debian stable main contrib non-free

# Unstable
deb http://ftp.debian.org/debian unstable main contrib non-free
deb-src http://ftp.debian.org/debian unstable main contrib non-free
```

Nach einem Neustart des Gerätes werden beim Update die Pakete nun von dem Unstable-Verzeichnis bezogen. Der Begriff „unstable“ bezeichnet den Status der Betriebssystemversion - in diesem Fall noch in der Entwicklung. Auf der Website von Debian wird die Version für Entwickler zur Verfügung gestellt und ist nicht für den „normalen“Gebrauch gedacht [12]. Auch unter der instabilen Version stehen die nötigen Pakete nicht zur Verfügung. Daher wurde die Installation zurückgesetzt.

Eine weiter Möglichkeit zur Programmierung der GUI und Diagrammausgabe ist Python. Die benötigten Pakete werden auf den Raspberry Pi heruntergeladen. Nachfolgend wird in der Liste die Erklärung zum Bedarf des jeweiligen Pakets ausgeführt:

- **Serial:** In diesem Paket sind alle Funktionen für den Aufbau einer seriellen Verbindung enthalten. Dazu gehören sowohl Funktionen wie das Definieren der genauen Verbindungsdetails, als auch die Befehle zum Schreiben und Lesen.

[1] Octave wird durch einzelne Pakete erweitert. Mit dem Octave-Paket-Manager kann nach benötigten Paketen im Internet gesucht und diese installiert werden

- **Matplotlib:** Diese Bibliothek ermöglicht die Visualisierung von Werten. Auf der Website werden unzählige Diagramme zum kostenlosen Download angeboten. Eine Vielzahl an Darstellungen ist bereits als Standard enthalten.
- **Numpy:** Abgeleitet von „numeric-python“ stellt dieses Paket neben vielen mathematischen Grundfunktionen noch einen großen Umfang an Vektor- und Matrixoperationen zur Verfügung. Dazu gibt es einen eigenen Array-Datentyp, der auch in dieser Arbeit eingesetzt wird.
- **PyQt5[2]:** Mit diesem Qt-Paket eröffnen sich zahlreiche Möglichkeiten der Anwendungsentwicklung, von NFC und Bluetooth Anbindungen bis hin zu chromiumbasierten Webbrowsern, angebunden an Python. Die Bibliothek bietet alle gängigen GUI Module und einen Designer, den QtCreater, um die Fenster mit Hilfe von „drag-and-drop“ der Elemente zu entwerfen.

Als Entwicklungsumgebung wird mit dem Texteditor Geany, der das GTK3 Toolkit nutzt, gearbeitet. Der Editor läuft flüssig und schnell auf dem Raspberry Pi und öffnet beim Compilieren und Bauen neben der Anwendung auch ein Terminal, in dem Fehlermeldungen und weitere Hinweise ausgegeben werden können. Der Ablauf für den Raspberry Pi sieht wie folgt aus. Es werden die Fenster mit dem QtCreater entworfen und anschließend in Python importiert. Im Python Skript können nun die Schaltflächen der Fenster mit Events verknüpft werden. Diese Anwendung lässt sich anschließend auf dem Raspberry Pi starten und ausführen.

3.3 Entwicklung des Programmes

Zur Entwicklung des gesamten Programmes zählt an erster Stelle die Auswahl der benötigten Funktionen. Im folgenden werden sie den vorhandenen Plattformen zur Implementierung zugeordnet werden. Dann wird ein konkreter Programmablauf erstellt. Diese Entwicklungsschritte werden in den folgenden Unterkapiteln beschrieben.

3.3.1 Auswahl und Verteilung der Grundfunktionen

Um die Grundfunktionen, die das finale System erfüllen soll, festzulegen, wird ein schematischer Ablaufplan erarbeitet (siehe Abbildung 3.5). Für die Signalaufnahme ist die Grundlage, eine funktionierende TMR-Sensor-Matrix, bereits geschaffen. Die Messwertgewinnung ist teilweise ebenfalls auf dem Tiva-Board implementiert. Da das System auch eine 15×15 Matrix simulieren soll und es allerdings nur 8×8 Messwerte gibt, wird die in Kapitel 2.1.2 eingeführte Methode der Interpolation angewendet. Die Interpolation von einer 8×8 Matrix auf eine 15×15 Matrix kann durch einfache Matrixmultiplikationen realisiert werden und dann für weitere Signalverarbeitungszwecke zur Verfügung stehen.

[2] PyQt5 ist ©Riverbank Computing Limited und wird unter der GPL v3 Lizenz für nicht-kommerzielle Zwecke verwendet.

Somit wird diese Methode in C-Code auf dem Tiva-Board implementiert. Nach der Gewinnung der Messwertmatrizen müssten die Messwerte noch weiter verarbeitet werden. Vor der Vorverarbeitung, wie eine Filterung, muss eine Offsetkompensation durchgeführt werden. Die Umsetzung mit der KQ-Methode, wie in Kapitel 2.1.3 eingeführt, wird ebenfalls auf dem Tiva-Board implementiert. Diese Kompensation soll jedes Mal bei Programmstart durchgeführt werden können, oder es soll die Möglichkeit geben, die Ergebnisse der letzten Kompensation für den Betrieb zu verwenden. Die oben erwähnte Filterung wird bei den Größen der Matrizen, mit jeweils 64 oder 225 Werten, ohne Hilfe eine Fourier-Transformation eine sehr rechenintensive und damit zeitaufwändige Aufgabe. Aus dem Grund wird zuerst eine 2D-DFT implementiert. Eine umfangreiche Bibliothek mit einer sehr schnellen Transformation, der Fastest Fourier Transform in the West (FFTW), haben Matteo Frigo und Steven G. Johnson am Massachusetts Institute of Technology (MIT) entwickelt. Informationen zur Funktionsweise finden sich in der Literatur [6]. Aus Gründen der Geschwindigkeit soll die DFT, die Filterung sowie die inverse Diskrete Fourier-Transformation (IDFT) auch auf dem Tiva-Board implementiert werden. Bei einem Test der FFTW wird jedoch festgestellt, dass der Code nicht auf das Tiva-Board gespielt werden kann. Mit einer Fehlermeldung bricht der Prozess ab, da die Bibliotheken nicht für das Board geeignet seien. Aus diesem Grund werden die Formeln für die DFT manuell in C implementiert. Die letzte Funktion, die noch auf dem MC implementiert werden soll ist der CORDIC-Algorithmus zur Winkelbestimmung. Aufgrund des Rechenaufwands ist der Raspberry Pi dafür ungeeignet. Der Drehwinkel und die entsprechenden Werte sollen vom Mikrocontroller an den Raspberry Pi übermittelt werden.

Beim fünften Punkt aus Abbildung 3.5 geht es um die Darstellung der Messwerte und der Ergebnisse. Die grafischen Funktionen sollen auf dem Raspberry Pi mit Python umgesetzt werden. Vor der eigentlichen Visualisierung der Messwerte, müssen in der Bedienoberfläche einige Einstellungen vorgenommen werden. In dem Hauptfenster soll die Filterung angepasst, die Interpolation ein- und ausgeschaltet und ein Diagramm ausgewählt werden können. Mit Betätigung der Schaltfläche „Diagramm erstellen“ sollen die Informationen an den Mikrocontroller übermittelt werden und das Diagrammfenster geöffnet werden. Die Messwerte sollen in der vom Benutzer gewünschten Form dargestellt werden. Für genauere Feinabstimmungen können die Filterkoeffizienten noch einzeln vom Benutzer verändert werden.

3.3.2 Entwurf des Programmablaufs

Um einen Überblick über den Programmablauf zu bekommen und in nachfolgenden Kapiteln einen Bezugspunkt zu haben, wird in diesem Unterkapitel der Entwurf des gesamten Programmes erläutert. Die Entwicklung und genaue Implementierung der einzelnen Funktionen ist in Kapitel 4 zum Mikrocontroller und in Kapitel 5 zum Raspberry Pi dargestellt.

Gestartet wird die Software auf dem Raspberry Pi. Dort soll entweder über eine `.exe`

Abbildung 3.5: Schematischer Arbeitsablauf, der von einer Signalverarbeitung und Bedienoberfläche umgesetzt werden muss.

oder über eine IDE, wie Geany, die Main-Funktion des Python-Skriptes ausgeführt werden. Zum Programmstart wird das Kalibrierungsfenster geöffnet. Hier werden dem Nutzer die Möglichkeiten einer neuen Kalibrierung sowie das Laden von vorhandenen Daten oder das Überspringen der Kalibrierung zur Auswahl angezeigt. Wenn eine neue Kalibrierung gestartet wird, wird diese Information an den MC gesendet und eine Kalibrierfahrt von 360° durchgeführt. Dabei wird alle 15° eine Messung durchgeführt und am Ende über alle 24 Messungen gemittelt. Außerdem soll eine schnelle „one-shot“-Kalibrierung mit einen Quadrupol durchgeführt werden können. Der Winkel wird dabei kontinuierlich an den Raspberry Pi übermittelt, um dem Nutzer über einen Ladebalken den Fortschritt der Kalibrierung anzuzeigen. Nach Abschluss wird das Fenster geschlossen und das Hauptfenster geöffnet.

Im Hauptfenster sollen alle Einstellungsmöglichkeiten zur Darstellung und Vorverarbeitung der Messwerte zur Verfügung stehen, sowie ein Buttons zum Aufrufen der Bedienungsanleitung. Die erste Einstellungsmöglichkeit ist das Ein- und Ausschalten der Interpolation. In der nächsten Option kann die Filterung eingestellt werden. Das Betätigen dieses Buttons soll ein entsprechendes Fenster mit den Einstellungsmöglichkeiten öffnen. Es gibt die Möglichkeit, einen Filter auszuwählen und, wenn gewünscht, die Koeffizienten des 8×8 Filters manuell anzupassen. Dafür wird der linke obere Quadrant der Filtermatrix angezeigt. Da die Matrix für die X- und Y-Achse achsensymmetrisch ist, kann durch diese Reduktion Platz gespart und die Übersichtlichkeit verbessert werden. Diese Matrix ist entweder 4×4 oder 8×8 groß. Durch Betätigen des „Speichern“ Buttons werden die vorgenommenen Einstellungen übernommen und das Fenster wieder geschlossen. Als letztes wird ein Diagramm ausgewählt und über die Schaltfläche „Diagramm erstellen“ zu bestätigt. Nach Betätigung werden die Einstellungen gespeichert und ebenfalls an den Mikrocontroller gesendet.

Bis zu vier Daten-Streams für vier Diagramm-Fenster sollen parallel laufen können. Dazu gibt es auf dem Raspberry Pi einen Thread, der die Einstellungen kontinuierlich an den Mikrocontroller sendet und Daten von dort empfängt und in globalen Variablen speichert. Von dort werden auch die Werte für die Diagramme bezogen. Vom Mikrocontroller werden die Einstellungen empfangen und die Sensorwerte entsprechend verarbeitet. Gesendet wird jeweils die bearbeitete Sensor-Matrix im 15×15 Format und der zugehörige

Winkel. Auf dem Raspberry Pi ist für die erhaltenen Werte hinterlegt, ob entweder 8×8 Werte aus der Matrix ausgelesen werden sollen oder die gesamte Matrix gespeichert werden muss. Die Ausgabe in ein Diagramm erfolgt kontinuierlich indem die neuen Werte jeweils aus den globalen Variablen ausgelesen und in dem gewählten Diagramm dargestellt werden. Erst wenn von den vier Fenstern wieder eines geschlossen wird, kann mit neuen Einstellungen ein weiteres Fenster geöffnet werden. Mehr Fenster oder Diagramme kann der Raspberry Pi nicht gleichzeitig verarbeiten. Es würde zu Verzögerungen in der Darstellung kommen.

3.4 Konzept des mechanischen Aufbaus

Für den Transport und die Präsentation des Sensor-Arrays benötigt es ein Vorführungssystem. Mit diesem System soll ein Magnet über der Sensorplatine elektronisch gedreht werden können. Des weiteren sollen verschiedene Störfelder montierbar sein. Ein erster Entwurf, der in einem Koffer transportiert werden kann ist in Abbildung 3.6 dargestellt. an der rechten Kofferseite befinden sich die Anschlüsse für die Stromversorgung, die Anschlüsse für USB-Geräte und einen Monitor.

Im Laufe der Arbeit wurde die Fertigung eines ersten Prototyp für eine Halterung an Herrn Grejdieru übergeben. In enger Zusammenarbeit wurde sich über die Fertigung ausgetauscht. Die Materialien sollten leicht sein, also entweder aus Holz oder Plastik oder einer Kombination aus beidem. Es dürfen außerdem keine magnetischen Bauteile

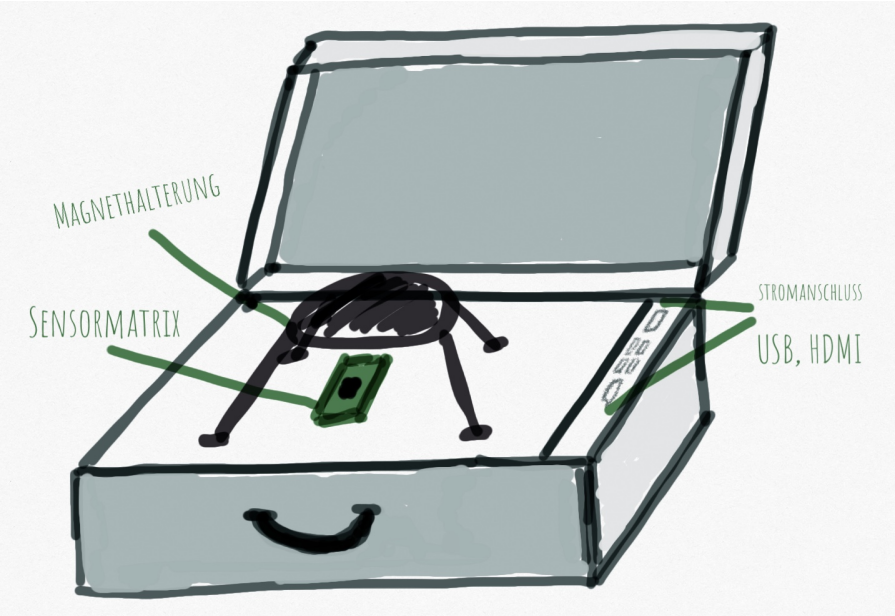
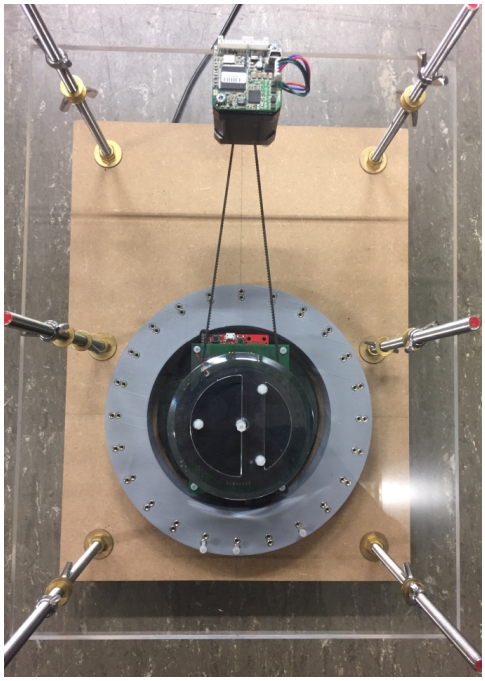


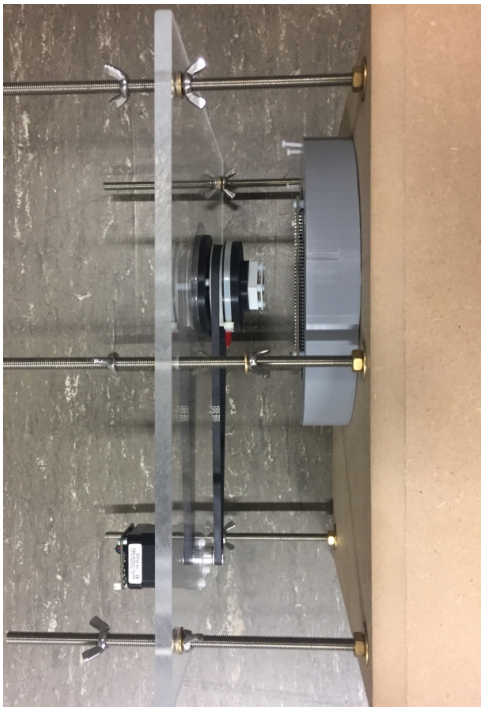
Abbildung 3.6: Erster Aufbau eines Transportkonzeptes.

verbaut werden, da diese die Messungen beeinflussen könnten. Das System muss einfach aufzubauen sein und die Halterung in dem Koffer verstaut werden können.

Es wird sich für ein Holzgestell als Fundament geeinigt, da dieses die nötige Stabilität für weitere Aufbauten liefert. Die Halterung für Störfelder wird mit einem 3D-Drucker hergestellt. Die Störfeldhalterung wird so auf dem Holzteil montiert, dass in der Mitte das Tiva-Board mit Sensorplatine montiert werden kann und das Störfeld horizontal durch die Sensoren verläuft. Des weiteren wird der Aufbau einer Drehvorrichtung für einen Magneten entwickelt. Diese soll mit einem Schrittmotor der Firma Trinamic realisiert werden. Der Motor kann via USB angesteuert werden. Da der Motor von den Magneten beeinflussbar ist, darf dieser nicht zu nah an der Messvorrichtung montiert werden. Die Lösung ist ein Riemen, damit die Messung für den Magneten aus einiger Entfernung (etwa $20cm$) durchgeführt werden kann. Die Drehvorrichtung sowie der Motor werden an einer $10mm$ dicken Plexiglasplatte befestigt. Dadurch bleibt ein Überblick über den Messplatz bestehen. Am Rand des Holzfundaments werden je drei Gewindestangen aus Edelstahl befestigt. Durch die Plexiglasplatte werden sechs Löcher gebohrt und Flügelmuttern auf die Gewindestangen gedreht. Die Plexiglasplatte kann damit über dem Sensor-Arrays in ihrer Hohe variiert werden. Trotz des hohen Gewichts des Holzes ist die Halterung immer transportabel.



(a) Darstellung der Draufsicht der Halterung.



(b) Darstellung der Seitenansicht der Halterung.

Abbildung 3.7: Darstellung des fertigen Konzepts der Magnethalterung.

4 Software auf dem Mikrocontroller

Aufbauend auf der Verteilung der Grundfunktionen im Unterkapitel 3.3.1 steht nun die Implementierung der Algorithmen im Fokus. Der auf dem MC bereits vorhandene Code muss für die nötige Kommunikation und Verarbeitung angepasst werden. Diese Anpassungen sowie die Kozeptionierung der Kommunikation des Mikrocontrollers mit dem Raspberry Pi wird nachfolgend beschrieben.

4.1 Modifikationen von Vorarbeiten

Von den vorhandenen Dateien müssen nur wenige bearbeitet werden, da hauptsächlich neue Dateien mit den Funktionen zur Signalverarbeitung hinzugefügt werden. Im Folgenden werden die Änderungen an den einzelnen Dateien beschrieben.

vals_to_terminal.c: In dieser Datei, dargestellt in Listing A.9, sind die Funktion zum Senden verschiedener Daten an das UART-Terminal implementiert. Da statt zwei Arrays für die Datenspeicherung ein Struct-Array verwendet wird, wurde dahingehend die Sendefunktion `vals_to_terminal8()` angepasst. Weiterhin wurden zwei Funktionen zum Senden der Winkel implementiert. Da es sich hier bei dem Winkel-Array in der Funktion `print_angle8()` um ein Double-Array handelt, beziehungsweise beim Senden in der Funktion `calibration_to_terminal()` des Mittelwertes um einen einzelnen Winkel, mussten dafür neue Funktionen implementiert werden.

main.c: Wie bereits beschrieben wurde in dieser Datei die komplette Empfangsroutine hinzugefügt. Hier gab es anfangs nur eine `while(1)` Schleife und der Interruptbefehl „*-test*“ wurde im Interrupt-Handler verarbeitet, woraufhin die unbearbeiteten Sensordaten gesendet wurden. Wenn das Flag `buffer_full` von der Interrupt-Routine gesetzt ist, wird die Verarbeitung durchgeführt. Die Switch-Case, mit der die Signalverarbeitung über den ersten empfangenen Charakter maßgeblich geregelt wird, ist in Listing A.8 einzusehen. Die Empfangsroutine gibt durch den Befehl vom Raspberry Pi den Verlauf der Signalverarbeitung vor. Je nachdem, welcher Kalibrierbefehl zum Programmstart vom MC gesendet wird, wird ein globales Flag `kurzOffset` gesetzt. Wenn beispielsweise eine kurze Kalibrierung durchgeführt wird, wird `kurzOffset = 1` gesetzt und in der Offset-Subtraktion der Offset-Mittelwert von jedem Sensor abgezogen. Die Möglichkeiten weiterer Verarbeitungsabläufe sind in der Abbildung 4.1 angegeben. Die Sensor-Daten werden für jeden eingehenden Befehl entsprechend verarbeitet und das gewünschte Resultat wieder zum Raspberry Pi gesendet. Die ersten Cases behandeln die Initialisierung

der Twiddle-Matrizen und das Bestimmen der Sensor-Offsets. Weitere Cases beinhalten die Verarbeitung für angeforderte Winkel-Matrizen. Die Verarbeitung unterscheidet sich ja nachdem, ob interpoliert, gefiltert oder nur die Winkel ohne Vorverarbeitung angefordert wurden. Gleiches gilt für das Senden von Sinus- und Cosinus-Werten. Die Switch-Case bietet noch Platz für weitere Befehle.

Interrupt_Handler.c: Hier gab es ursprünglich eine `while(1)` Schleife, die den Input-Buffer ausgelesen hat. Da in einem Interrupt-Handler aber keine umfangreichere Verarbeitung der eingehenden Befehle stattfinden darf, wird hier lediglich das Speichern des eingehenden `char` geregelt. Eingehende `Strings` sind mit einem `"r"` terminiert. Sobald das `"r"` registriert wird, wird das Flag `buffer_full` gesetzt, dass der Befehl fertig eingelesen wurde. Die Speicherung erfolgt in zwei globale Arrays. Ein Array beinhaltet den eingegangenen Befehl (die ersten vier Zeichen) und das Andere die Filterkoeffizienten (Ziffer fünf bis zur Terminierung). Nach dem Setzen des Flags kann die Verarbeitung in der Main-Funktion beginnen. Die Optimierung des Interrupt-Handlers ist in Listing A.10 einzusehen.

Get_Values.c: Die empfangenen Signalwerte der Sensoren wurden bisher in einem Sinus- und einem Cosinus-Array als `unsigned int` gespeichert. Durch die Implementierung des `struct complex` Arrays, müssen die Sensorwerte zu `double` gecastet werden. Diese Werte werden jeweils in dem Real- oder Imaginärteil des Arrays `inputData` gespeichert, siehe Listing A.7.

init_prototype.h: Die Implementierung der neuen Methoden für die Signalverarbeitung und das Senden der Daten erfolgt objektorientiert. Deshalb müssen die neuen Funktionen in dieser Datei als Prototypen initialisiert werden, damit der Linker diese im Gesamtsystem verknüpfen kann. Die Funktionsprototypen wurden bezüglich ihrer Dateizugehörigkeit sortiert. Somit herrscht eine bessere Übersicht, wenn zukünftig Änderungen an dem Programmcode vorgenommen werden. Die Änderungen an der Header-Datei sind in Listing A.12 nachzulesen.

include.h: In dieser Include-Datei, dargestellt in Listing A.11, wurden weitere globale Variablen deklariert. Dies geschieht, damit diese für alle Funktionen zugänglich sind.

4.2 Implementierung der Signalverarbeitungsmethoden

Die Grundlage einer Implementierung von Signalverarbeitungsmethoden ist meistens die Formel eines Algorithmus. Solche Formeln lassen sich in Programmcode umwandeln, jedoch muss diese Formel komplett aufgeschlüsselt berechnet werden. Wenn zum Beispiel in der Formel zwei Matrizen $\mathbf{A}$ und $\mathbf{B}$ multipliziert werden, müssen die einzelnen Elemente der Matrizen in einer Schleife multipliziert werden. Wenn solche Operationen

häufiger vorkommen, bietet es sich an, eine eigene Funktion zu schreiben, die zwei übergebene Matrizen multipliziert und das Ergebnis zurückgibt. Solche Maßnahmen bringen Übersichtlichkeit und vereinfachen das Debugging des Codes.

Abbildung 4.1 zeigt den Ablauf der einzelnen Funktionen auf dem Mikrocontroller. Die grünen Pfeile geben die Verbindung mit dem Raspberry Pi an. Entweder sind es eingehende Befehle, die den Programmablauf bestimmen, oder es sind Daten, die an den Raspberry Pi gesendet werden. Die nachfolgenden Unterkapitel sind entsprechend der Reihenfolge des Programmablaufs gegliedert. Es wird jeweils der Code erläutert und wie die Algorithmen umgesetzt wurden. Für die meisten Funktionen gibt es jeweils eine Funktion, die Matrizen mit 8×8 Werten und eine Funktion, die die interpolierten Werte (15×15) verarbeitet. Da auf dem Mikrocontroller ausreichend Speicherplatz zur Verfügung steht, wurde dies für eine bessere Übersicht und zum besseren Debugging so implementiert. In den nachfolgenden Beschreibungen werden fast ausschließlich die Funktionen für die 8×8-Berechnungen aufgeführt. Es gibt jedoch immer ein Gegenstück für die interpolierte Matrix[1]. Dies wird in Abbildung 4.1 deutlich.

Offsetkompensation: Vor der eigentlichen Datenverarbeitung, müssen die Messwerte von den Sensoren ausgelesen und gespeichert werden, wie in Kapitel 4.1 in der Funktion `get_Values.c` beschrieben. Im Anschluss an das Einlesen und Speichern der Daten in dem `struct complex InputData` wird der Offset berechnet. Wenn die schnelle Kalibrierung durchgeführt wird, wird das Array an die Funktion `calculate_offset_8()` in der Listing A.1 übergeben. In dieser Funktion wird der Offset der einzelnen Sensoren, nach dem in den Grundlagen 2.1.4 eingeführten Algorithmus, berechnet und gemittelt. Gespeichert wird der Real- und Imaginärteil des Offsets in dem globalen Array `resultOffset`. Somit kann nach dem Einlesen weiterer Daten jeweils der Sensor-Offset subtrahiert werden. Mit der Durchführung der Langzeitkalibrierung sendet der Mikrocontroller 24 mal die Rohwerte an den Raspberry Pi. Nach Abschluss der Messung werden zwei Matrizen vom Raspberry Pi an den Mikrocontroller zurückgesendet. Dabei handelt es sich um 64 Cosinus- und 64 Sinuswerte. Diese werden in der Matrix `offsetMatrix` gespeichert. Die Subtraktion wird in der Funktion `sub_offset8()` durchgeführt. Je nach gesetzter, globaler Variable `kurzOffset` wird der Mittelwert oder die Matrix zur Subtraktion verwendet. Der Funktion werden im laufenden Betrieb die Eingangsdaten übergeben und es wird von Real- und Imaginärteil der Sensor-Offset subtrahiert.

Interpolation: Wenn vom Raspberry Pi der Befehl kommt, dass interpolierte Daten gesendet werden sollen, wird das `inputData` Array an die Funktion `interpolieren` in Listing A.2 übergeben. Zuerst werden die Messwerte in dem globalen Array `interpoliert15` so gespeichert, dass jeder zweite Wert in der Matrix beschrieben wird. Die Zwischenwerte bleiben Null und werden im nächsten Schritt aus den Umgebungswerten interpoliert (siehe Abbildung 2.2). In einer doppelten For-Schleife werden daher die Nachbarwerte

[1] Beispiel: die Funktionen *calculate_offset_8()* und *calculate_offset_15()*.

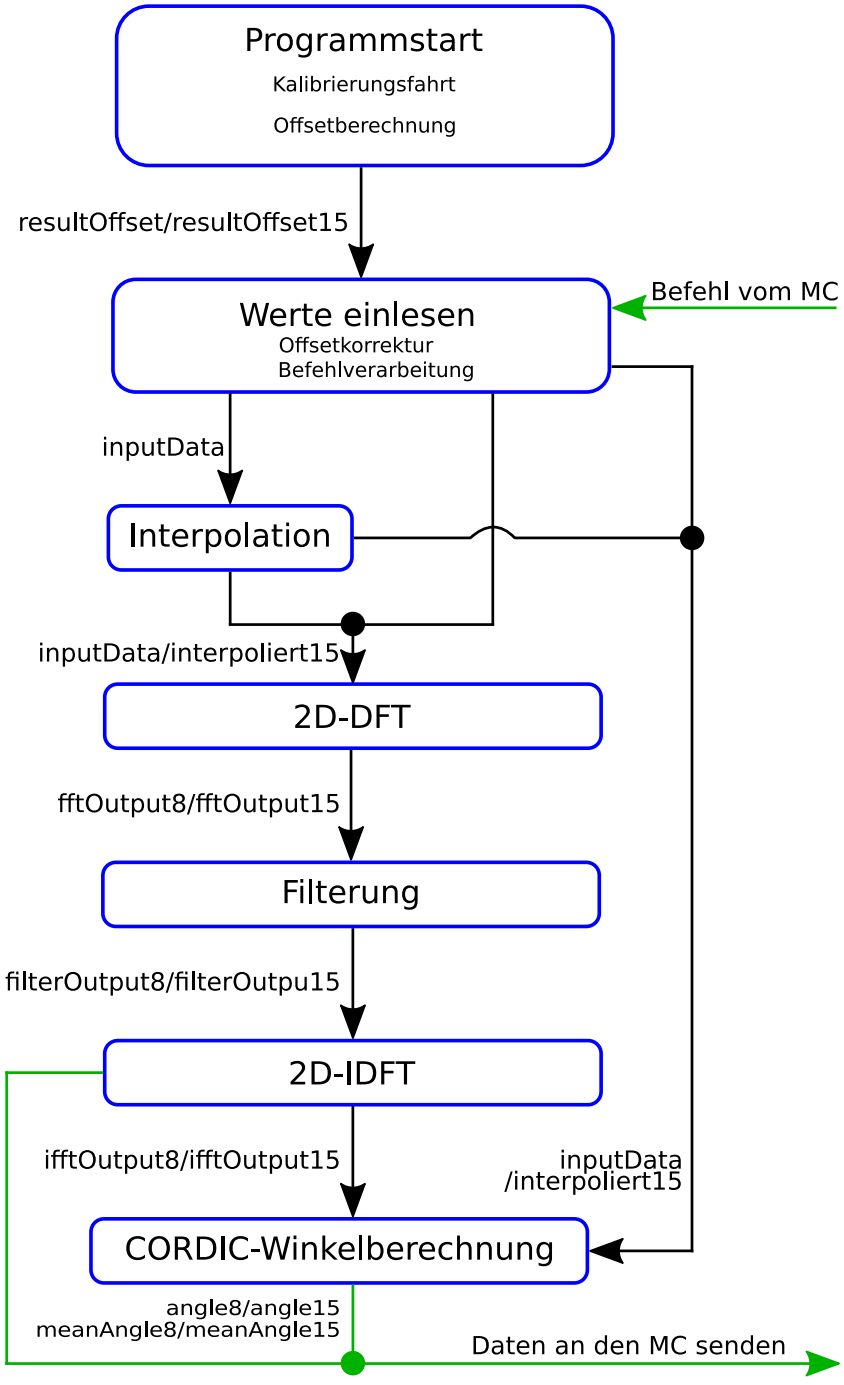
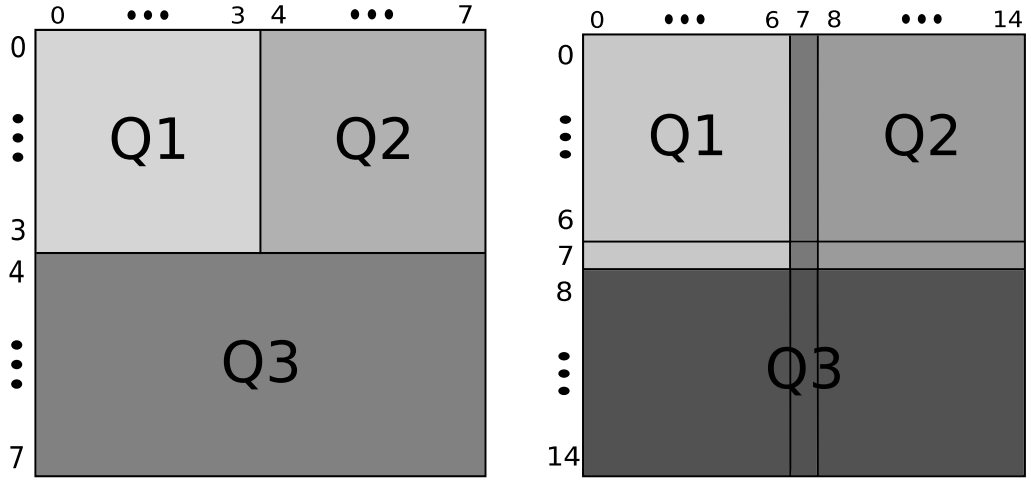


Abbildung 4.1: Programmablauf auf dem Mikrocontroller mit ein- und ausgehender Verbindung zum Raspberry Pi.

zu je einem Halb- oder einem Viertel-Teil addiert und gespeichert. Des weiteren soll mit der Interpolation die Offsetmatrix der Langzeitkalibrierung in eine 15×15 Matrix übertragen werden.

Fourier-Transformation: Die Berechnung der Fouriertransformation, dargestellt in Listing A.3, wird bereits zum Programmstart vorbereitet. Wie aus dem Grundlagenteil hervorgeht, müssen die Twiddle-Matrizen nur einmal zum Programmstart berechnet und gespeichert werden. Der Funktion `fft()` müssen keine Parameter übergeben werden. In der Berechnung wird direkt das Array `inputData` verwendet. Für den ersten Schritt wird eine komplexe Multiplikation der Matrizen durchgeführt und in einem Buffer gespeichert. Die zweite Multiplikation mit der Twiddle-Matrix muss ebenfalls als komplexe Multiplikation realisiert werden. Das Ergebnis wird in dem globalen Array `fftOutput8` gespeichert. Die inverse Fourier-Transformation wird im Anschluss an die Filterung durchgeführt, um die Werte vom Bild- zurück in den Zeitbereich zu transformieren. Hierfür werden die gefilterten Werte im Array `filterOutput8` komplex mit der inversen Twiddle-Matrix multipliziert und in einem Buffer zwischengespeichert. Das Ergebnis der zweiten komplexen Matrixmultiplikation von Buffer und inverser Twiddle-Matrix wird, nach der Normierung mit $\frac{1}{64}$ (der Anzahl Matrix-Elemente) in dem globalen Array `ifftOutput8` gespeichert.

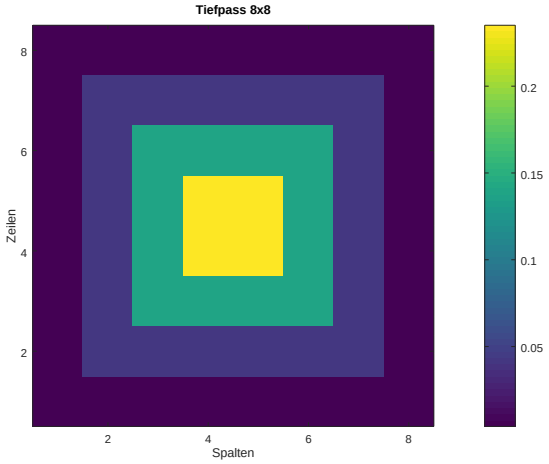
Filterung: Für die Filterung der Messwerte sind bereits zwei Filter vorbereitet, ein Tiefpass und ein Bandpass, gespeichert in globalen Arrays (Listing A.5). Die Berechnung der Filterkoeffizienten ist mit Octave im Vorfeld durchgeführt worden und in Listing C.1



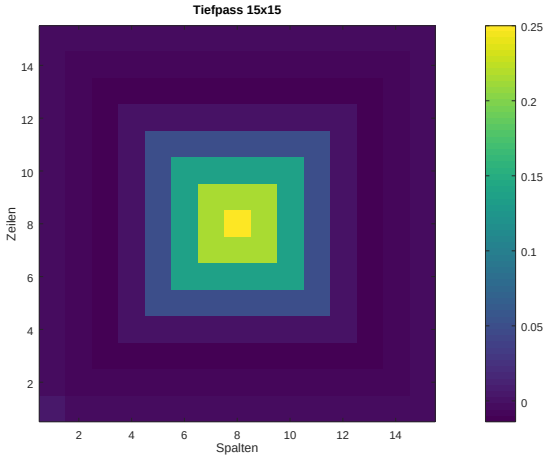
(a) Darstellung des Füllens der 8×8 Custom-Filtermatrix.

(b) Darstellung des Füllens der 15×15 Custom-Filtermatrix.

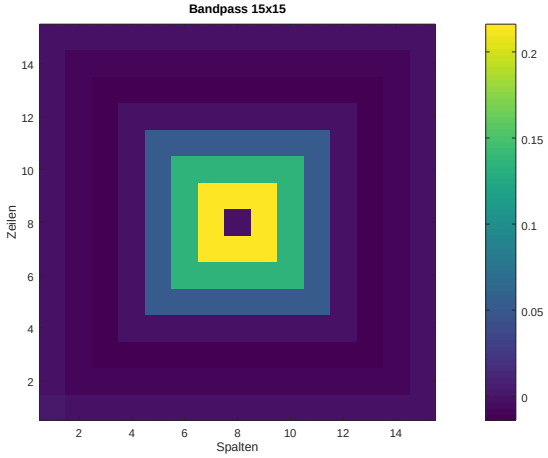
Abbildung 4.2: Schematische Darstellung des Algorithmus zum Füllen der Filtermatrizen.



(a) Tiefpasskoeffizienten 8×8



(b) Tiefpasskoeffizienten 15×15



(c) Bandpasskoeffizienten 15×15

Abbildung 4.3: Grafische Darstellung der Tief- und Bandpassfilter.

dargestellt. Eine Visualisierung der Filter ist in Abbildung 4.3 dargestellt. Für die 8×8 Matrix wird kein Bandpass vorgegeben, da kein konkreter Mittelpunkt gegeben ist, der auf Null gesetzt werden kann, um den Offsetanteil aus dem Spektrum zu filtern. Wenn in dem Befehl vom Raspberry Pi eine Filterung angeordnet wird, wird der Funktion `filter8()` ein Filtertyp und die Matrix `fftOutput8` übergeben. Falls jedoch die Filterkoeffizienten vom Benutzer verändert wurden, müssen die übertragenen Koeffizienten eingelesen und anschließend zur Filterung verwendet werden. Übergeben bekommt der Mikrocontroller nur die Werte des ersten Quadranten, da die Matrizen achsensymmetrisch sind. In Abbildung 4.2 sind die Algorithmen der Speicherung schematisch dargestellt. Für eine 8×8 Matrix wird der erste Quadrant mit den Koeffizienten gefüllt und anschließend gespiegelt in den zweiten Quadranten kopiert. Quadrant eins und zwei werden nun im letzten Schritt wiederum gespiegelt in Quadrant drei kopiert. Bei der 15×15

Matrix müssen die Kopiervorgänge angepasst werden, da aufgrund der ungeraden Zeilen- und Spaltenzahl, die Symmetrieachsen nicht zwischen zwei Zeilen\Spalten liegen sonder auf der mittleren Zeile\Spalte. Der erste Schritt ist wieder das Speichern der Koeffizienten im ersten Quadranten der Größe 8×8. Im zweiten Schritt wird in Quadrant zwei kopiert, wobei die Spalte sieben überschrieben wird. Die Werte bleiben dieselben, aber es muss kein gesonderter Code erstellt werden, um die Werte beim Kopieren zu überspringen. Das Kopieren der Quadranten eins und zwei, dekrementiert von Zeile sechs bis null, in den Quadranten drei, inkrementiert von Zeile acht bis 14, wird ohne die siebte Zeile vorgenommen.

Zum Speichern der Werte stehen zwei Arrays, `CustomFilter_8` und `CustomFilter_15`, zur Verfügung. In der Funktion `filter8()` wird in einer Switch-Case geprüft, welches Filter genutzt werden soll und diese Werte werden an die Funktion `multiply8` übertragen. Die Matrizen werden im Anschluss multipliziert und das Ergebnis in der Matrix `filterOutput8` gespeichert und der IDFT-Funktion übergeben.

CORDIC-Algorithmus: Die Winkelberechnung ist in zwei Funktionen gegliedert, die in Listing A.6 aufgeführt sind. Die Funktion `cordic()` bekommt zwei double-Werte als Input, einen X- und einen Y-Wert. Der Winkel wird iterativ mit Hilfe vorgegebener Tangens-Werte berechnet. Dieser Algorithmus funktioniert nur mit positiven Eingangswerten. Daher müssen die übergebenen X und Y-Werte immer in den ersten Quadranten des Koordinatensystems transformiert werden, bevor sie an die Funktion `cordic()` emittiert werden. Diese Hin- und Rücktransformation findet in einer weiteren Funktion, der `calcAngle8()`, statt. Der Ausgangspunkt wird hinterlegt und nach der Winkelberechnung in den entsprechenden Quadranten zurück transformiert. Außerdem werden die einzelnen Winkel der Sensors in der Matrix `angle8` gespeichert. Im Anschluss aller Berechnungen wird der Mittelwert aller Matrixelemente gebildet. Hier liefert das Randwertproblem beim Übergang von Quadrant vier zu Quadrant Eins ein falsches Ergebniss für den Mittelwert. Wenn beispielsweise ein Sensor 3° misst und ein anderer Sensor misst 357°, ergibt das einen mittleren Winkel von 180°. Um diesen Fall zu detektieren werden Variablen gesetzt, die angeben, ob für eine berechnete Winkel-Matrix die Winkel sowohl im ersten als auch im vierten Quadranten lagen. Bei Eintritt dieses Falls wird zu allen Winkeln, die unter 180° liegen, 360° addiert. Somit werden die Winkel miteinander vergleichbar und aus den 3° würde zum Beispiel 363° . Der Mittelwert aus 363° und 357° ergibt sich zu 360°. Wenn der Mittelwert über 360° liegt, muss davon 360° subtrahiert werden, da der Mittelwert dann im ersten Quadranten liegt.

4.3 Kommunikation mit übergeordneten Systemkomponenten

Eine Kommunikation zwischen Systemen muss nur dann eingerichtet werden, wenn bestimmte Informationen, die das eine System besitzt, auch auf dem anderen zur Verfügung

Tabelle 4.1: Gegenüberstellung verfügbarer Informationen auf dem Mikrocontroller und dem Raspberry Pi.

Mikrocontroller	Raspberry Pi
• Filterdaten für einen Tief- und einen Bandpass (Matrix) • Sensordaten (Matrix mit komplexen Zahlen)	• Filterdaten für einen Tief- und einen Bandpass (Liste) • Diagrammtyp (Integer/String) • Interpolation (Bool) • Filtertyp (Integer/String) • Koeffizienten angepasst (Bool)

stehen sollen. Zu Beginn wird daher das Gesamtsystem beleuchtet, wie in Abbildung 3.2 dargestellt ist, um zu bestimmen, welche Informationen auf welchem System zur Verfügung stehen. Diese Angaben sind in Tabelle 4.1 zusammengefasst.

Aus dieser Tabelle und der Abbildung des Gesamtsystems geht nun hervor, welche Informationen zwischen den Systemen ausgetauscht werden müssen. Daraus wird folgendes Schema 4.4 abgeleitet, welche Daten zwischen den Systemen übertragen werden

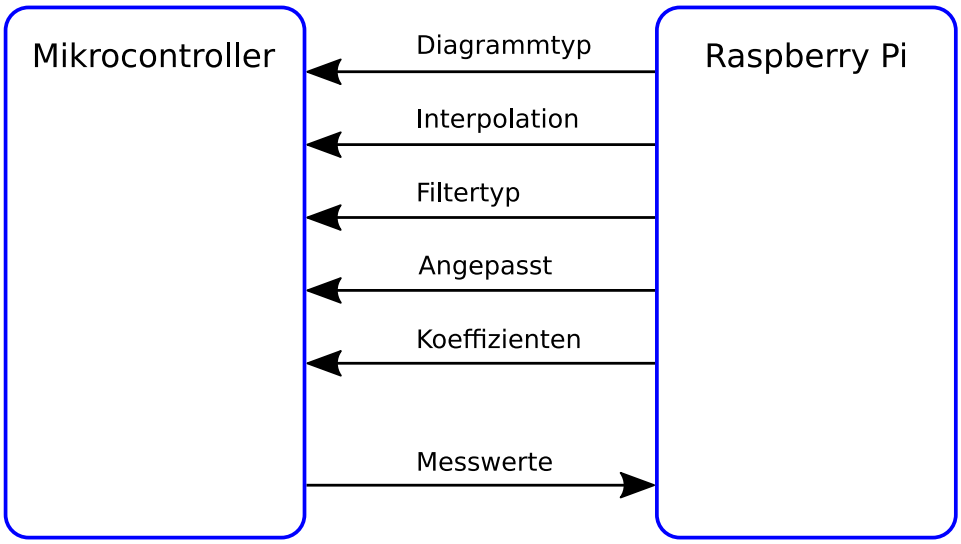


Abbildung 4.4: Darstellung der Informationen, die zwischen den Systemen ausgetauscht werden müssen.

Tabelle 4.2: Befehlsauflistung der Kommunikation zwischen Rasperry Pi und MC mit erläuterung der Nutzphase und Verarbeitung.

Befehl	Phase	Verarbeitung
„0000r“	Initialisierung	Initialisierung der Twiddle-Matrizen
„1000r“	Offsetberechnung	Speichern des aktuellen Offsets (Schnellkalibrierung)
„2000r“	Messphase	Senden einer Winkelmatrix der Sensoren
„3000r“	Messphase	Senden der Sinus- und Cosinus-Matrix
„4000r“	Messphase	Senden des Winkelmittelwertes
„5000r“	Offsetberechnung	Offsetmessung und senden der Sinus- und Cosinuswerte
„6000r“	Offsetberechnung	Speichern der empfangenen Offsetmatrix

müssen. Ohne die Eingabeinformationen vom Nutzer können auf dem Mikrocontroller nicht die gewünschten Daten berechnet werden. Die meisten Werte, die zwischen den Geräten gesendet werden sind Zahlen, bis auf zwei `bool` Werte. Diese lassen sich aber mit `True = 1` und `False = 0` in ein Zahlenformat transformieren. Diese Informationen sollen nun als ein Befehl zusammengeführt werden. Dem Befehl sollen außerdem die angepassten Filterkoeffizienten angehängt werden. Wenn das Anpassen-Flag gesetzt ist, werden diese Koeffizienten mit eingelesen und für die Filterung angewendet.

$$\{int\ diagrammtyp, int\ interpolation, int\ filtertyp, int\ angepasst, double\ filterwerte[]\}$$

Da insgesamt vier Diagrammfenster mit Messwerten versorgt werden sollen, empfiehlt es sich, eine Kommunikations-Routine einzuführen. Nach dem Call-and-Response-Prinzip spielt der Mikrocontroller die untergeordnete Rolle und der maßgebliche Teil findet auf dem Raspberry Pi statt. Die Erläuterungen dazu finden sich in Kapitel 5.3. Die eingehenden Befehle werden Anhand des ersten Wertes verarbeitet. Die Werte sind in Tabelle 4.2 aufgelistet. Es wird jeweils kurz erläutert in welcher Arbeitsphase sie gesendet werden und welche Verarbeitung sie nach sich ziehen.

5 Software auf dem Einplatinen-Computer

Die Software auf dem Einplatinen-Computer, dem Raspberry Pi, umfasst die Bereiche Front- und Backend. Dies bedeutet, es wird eine Oberfläche für Benutzer der Software eingerichtet, mit Hilfe dessen alle nötigen Einstellungen für die Darstellung der Messwerte geregelt werden können. Das Backend umfasst dabei die Verarbeitung der eingehenden Befehle vom Benutzer sowie die Beschaffung, Speicherung und Vorverarbeitung für die Darstellung der Messwerte. Im ersten Teil dieses Kapitels wird der Entwurf der Fenster und im späteren Verlauf die Verarbeitungsroutinen erläutert.

5.1 Entwicklung der Bedienoberfläche

Für den Entwurf der Bedienoberfläche werden die in den Grundlagen zum Interfacedesign aus Kapitel 2.4 beschriebenen Designkriterien herangezogen. Das Prinzip der Visibility bezieht sich demnach nicht nur auf ein einzelnes Fenster, sondern auch auf das Gesamte Programm. Wenn eine Software verschiedene Features bietet, wäre die gemeinsame Darstellung der Einstellmöglichkeiten für alle Features in einem Fenster unübersichtlich. Die hier entwickelte Software bietet sowohl Einstellungen zur Filterung, als auch Einstellungen zur Messwertdarstellung.

Im Anschluss an die Auswahl und Verteilung der Grundfunktionen in Kapitel 3.3.1 wurde Abbildung 5.1 entworfen, um eine Struktur in die Darstellung und den Ablauf der Fensteraufrufe zu bringen. Dazu wurden die Funktionen der Vorkalibrierung der Sensoroffsets (Kalibrierung), die allgemeinen Einstellungen zur Darstellung (Hauptfenster), die genaueren Einstellungen zur Filterung (Filtereinstellung) und die Darstellung der Grafen (Diagramm X[1]) jeweils einem neuen Fenster zugeordnet. Diese Aufteilung erhöht die Übersicht über das gesamte Programm.

Nach dem Programmstart wird das Kalibrierfenster, siehe Abbildung D.2, geöffnet. Das Menü dieses Fensters steuert die Kalibrierung der Sensormatrix. Mit dem ersten Button wird eine genaue Offsetmessung durchgeführt. Mit dem Button „Langzeitkalibrierung“ ist diese Auswahl möglich und es wird ein Dialogfenster geöffnet, welches den Benutzer durch das Kalibrierverfahren führt. Diese wird mit einem Halbacharray, siehe Abbildung D.7, durchgeführt. Eine zweite Wahl bietet die Möglichkeit der Durchführung einer schnellen Kalibrierung. Der Fortschritt wird dem Nutzer mit einem Fortschrittsbalken angezeigt. Diese Messung wird mit einem Quadrupol durchgeführt, wie er in

[1]Das „X“ steht in den folgenden Kapiteln bei den Fenstern für 1,2,3 oder 4 und bei den Matrizen für 8 oder 15.

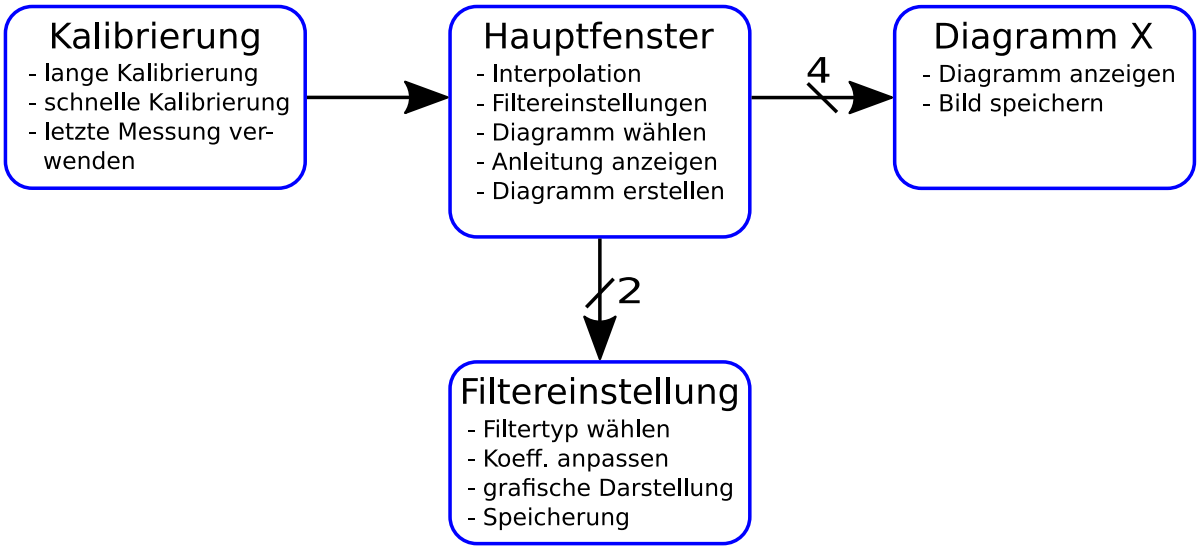


Abbildung 5.1: Aufteilung der Funktionen auf verschiedene Fenster und Skizzierung des Nutzungsablauf (X: 1, 2, 3, 4).

Abbildung D.8 dargestellt ist. Wenn keine der beiden Möglichkeiten genutzt werden soll, gibt es die Möglichkeit direkt in das Hauptfenster zu wechseln. Wenn Messwerte einer vergangenen Langzeitkalibrierung vorliegen, werden diese an den Mikrocontroller gesendet. Anschließend wird das Fenster geschlossen und das Hauptfenster geöffnet.

Das nächste Fenster, dargestellt in Abbildung D.3, bietet dem Nutzer alle Möglichkeiten zur Einstellung der Signalverarbeitung und Darstellung der Messwerte. Das Einschalten der Interpolation wird über das Setzen einer Checkbox getätigt. Es ist demnach direkt ersichtlich, ob interpoliert wird. Das Fenster zur genaueren Einstellung der Filterung kann über einen Button aufgerufen werden. Hier kommt der Punkt der Affordance zum Tragen, wodurch es für den Nutzer direkt ersichtlich ist, wie die Flächen zu bedienen sind. Über ein Drop-Down-Menü kann das gewünschte Diagramm gewählt werden. Dem Nutzer wird nur die eigene Auswahl angezeigt, um auf einen Blick die getätigten Einstellungen prüfen zu können. Eine Schaltfläche, die erst im späteren Verlauf der Entwicklung hinzugekommen ist, ist ein Button zum Aufrufen der Bedienungsanleitung. Durch klicken wird ein PDF-Dokument geöffnet, das Informationen über das Programm und sämtliche Einstellmöglichkeiten liefert. Als letzten Button gibt es die Schaltfläche zum Erstellen des Diagramms. Nach Betätigung wird ein neues Fenster, mit der Darstellung der Messwerte im Diagramm geöffnet.

Das Diagrammfenster gibt die Messwerte wieder und bietet die Möglichkeit zur Speicherung des Diagramms als Bild. Insgesamt vier Fenster können gleichzeitig aktiv sein. Andernfalls wird ein Fenster mit dem entsprechenden Hinweis geöffnet. Jegliche Bedienfunktionen werden über die Toolbar der Matplotlib Bibliothek zur Verfügung gestellt.

Ein Fenster, ohne dargestellten Plot, ist in Abbildung D.4 im Anhang abgebildet.

Das letzte Fenster, welches durch das Hauptfenster geöffnet werden kann, ist das Fenster zur Filtereinstellung. Hier ändert sich die Darstellung zum anpassen der Koeffizienten, wenn im Hauptfenster die Interpolation eingeschaltet wurde, siehe Abbildungen D.5 und D.6. Die Hauptschaltflächen für die Filtereinstellung befinden sich an der Oberkante des Fensters. Hier kann in einem Drop-Down-Menü der grundlegende Filtertyp ausgewählt werden. Zur Auswahl stehen ein Tief- und ein Bandpass. Das restliche Fenster ist mit Spinboxen, dargestellt in Matrixform, gefüllt. Wie in Kapitel 4.2 beschrieben, ist diese Matrix entweder 4×4 oder 8×8 groß. Hier kommt der Punkt des mappings zum tragen, da für den Benutzer die Matrixdarstellung eine direkte Verknüpfung mit der Visualisierung zur Koeffizientenmatrix darstellt. Je nach Auswahl der Filter, werden die Werte der Boxen mit den Werten des ersten Quadranten der Filtermatrix gesetzt. Neben dem Drop-Down-Menü für die Filter befindet sich eine Checkbox, mit der ausgewählt werden kann, ob die Filterkoeffizienten manuell angepasst werden. Erst durch das Betätigen der Checkbox werden die Spinboxen für die Anpassung freigeschaltet. Vorher konnten diese vom Nutzer nicht bearbeitet werden. Zur Überprüfung der angepassten Koeffizienten oder der bereitgestellten Filterkoeffizienten gibt es einen Button zur Visualisierung der Koeffizientenmatrix. Ähnlich zu Abbildung 4.3 werden die Werte für den Nutzer dargestellt. Die zwei letzten Schaltflächen, „Speichern“ und „Abbrechen“, schließen das Fenster. Mit dem Button Speichern, werden die eingegebenen Werte gesichert.

5.2 Auswahl und Implementierung der Darstellungsfunktionen

Bevor die Implementierung der Diagrammfenster beginnen kann, müssen auch hier die Anforderungen festgehalten werden. Jedes Diagramm soll in einem separaten Fenster dargestellt werden und es sollen maximal vier Fenster parallel aktiv sein. Dazu braucht es eine Prüfmöglichkeit, ob ein Fenster geöffnet, also sichtbar ist, oder nicht. Normalerweise stellen Bibliotheken die Funktionen zur Verfügung, um den Plot in einem Fenster zu öffnen. Diese Fenster haben jedoch nur eine Verbindung zu den Plot-Funktionen und nicht zum Hauptfenster; können also nicht auf ihre Sichtbarkeit überprüft werden. Für die Bibliothek Matplotlib gibt es eine Zusatzbibliothek, die es ermöglicht, den Plot in einem, mit der Software PyQt5 erstellten, Fenster darzustellen. Dazu wird die Bibliothek `matplotlib.backends.backend_qt5agg` in das Projekt mit eingebunden. Nun können die unzähligen Grafen, die Matplotlib mit sich bringt, in den PyQt5 Fenstern verwendet werden. Eine weitere Anforderung an den Plot ist die kontinuierliche Darstellung der sich ändernden Daten. Auch hier bietet Matplotlib ein Zusatzpaket an, das `matplotlib.animation` Paket. Der Funktion wird unter anderem ein Zeiger auf die Figur - also das Diagramm - und ein Funktionszeiger übergeben. Dadurch wird kontinuierlich eine Funktion aufgerufen und die darin geänderten Daten an die Figur weitergegeben, wodurch eine Animation entsteht.

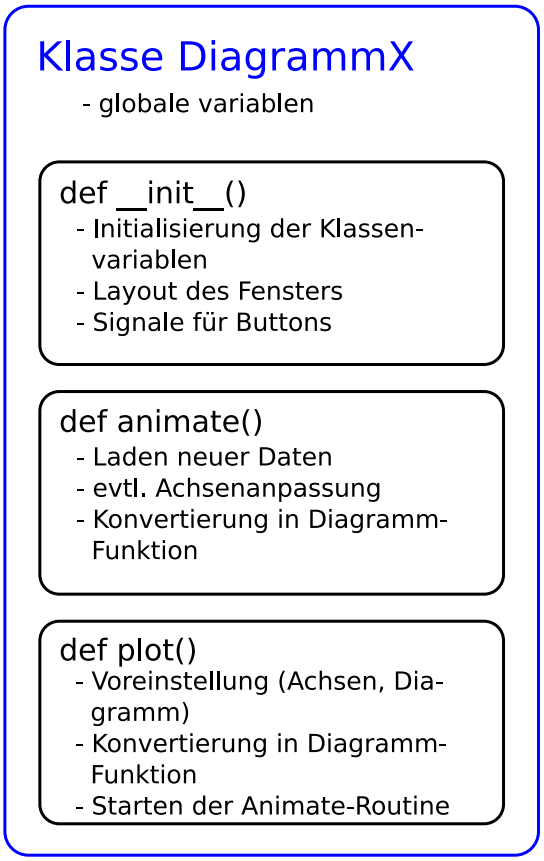
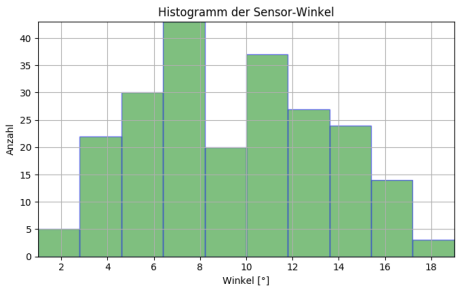


Abbildung 5.2: Grundgerüst der Klasse DiagrammX eines PyQt5 Fensters zur Darstellung eines Matplotlibgraphen.

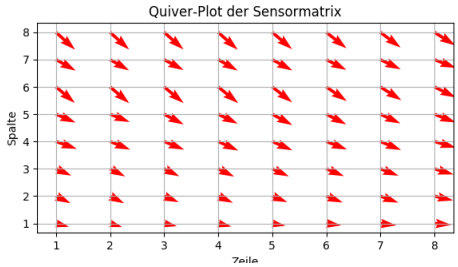
Der schematische Gesamtaufbau eines Fensters ist in Abbildung 5.2 zusammengefasst. In der `_init_()`-Funktion einer Klasse werden die Variablen initialisiert, die für alle Klassen-Funktionen zur Verfügung stehen. In der `plot()`-Funktion findet die Auswahl des Diagramms und die Initialisierung der Figurparameter mit Hilfe erster Messwerte statt. Die Verarbeitung der Messwerte wird mit der Python-Bibliothek Numpy durchgeführt. Die Messdaten werden für die Diagramme vom `list`-Format in das `array`-Format überführt. Dieser Datentyp lässt sich in Matplotlib besser verarbeiten. In der `animate()`-Funktion wird kontinuierlich der globale Messwertspeicher ausgelesen, die Figur-Achsen angepasst und das Diagramm aktualisiert. Der konkrete Aufbau der Fenster ist in Listing B.4 dargestellt. Die vier Diagrammfenster-Klassen sind, bis auf den Zugriff auf die globalen Variablen, identisch. Es wird Das Layout erstellt und nach Aufruf der `plot()`-Funktion wird, je nach gewähltem Diagramm, die Vorverarbeitung und Messwertdarstellung ausgeführt.

Nachfolgend werden die Diagramme eingeführt, die zur Datenvisualisierung verwendet werden. Es wird erläutert, wie die Messwerte vor-verarbeitet werden müssen und welchen

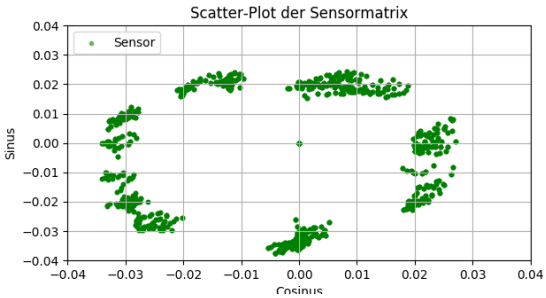
Nutzen das jeweilige Diagramm mit sich bringt. Abschließend wird auf wichtige Punkte bei der Implementierung eingegangenen. Der Programmcode für die einzelnen Plots und die jeweiligen `animate()`-Funktion ist ebenfalls in Listing B.4 dargestellt.



(a) Beispielplot eines Histogramms vom Raspberry Pi.



(b) Beispielplot eines Quiver-Plots vom Raspberry Pi.



(c) Beispielplot eines Scatter-Plots vom Raspberry Pi.

Abbildung 5.3: Ausschnitt der Messungen für den Scatter-Plot.

Histogramm: Ein Histogramm gibt in einem Balkendiagramm die relative Anzahl konkreter Messwerte im Datensatz an. Diese Darstellung ist besonders für die Erfassung der Sensorwinkel relevant. Es wird angegeben, wie viele Sensoren den jeweiligen Winkel gemessen haben. So bietet diese Darstellung einen Überblick über die gesamte Streuung der Messung. Auffälligkeiten sind sofort zu erkennen, falls ein Winkel weit außerhalb der Hauptgruppe liegen sollte. Die übermittelte Matrix der Sensorwinkel wird vom `listen`-Format in das Numpy `array`-Format überführt. Anschließend werden die Daten und die Anzahl der X-Achsenabschnitte der Numpy Funktion `histogram` übergeben. Diese teilt den Datensatz in entsprechende Abschnitte. Aus diesen Daten werden der niedrigste und der höchste Messwert extrahiert, für das dynamische Anpassen der X-Achse verwendet werden. Die Variable `bottom` wird mit Nullen initialisiert und gibt damit die Unterkante der Balken an. Mit der variable `top` werden aus dem Datensatz für die Balken die Anzahl der vorhandenen Messwerte angegeben. Aus diesen beiden Variablen und zwei weiteren, `left` und `right`, werden die Balken gebildet. Die Funktion `FuncAnimation` ruft die Funktion `aniHistogram` kontinuierlich auf. In dieser müssen die neuen Daten aus der

globalen variable `wX_angles` übersetzt, die Balken gebildet und die Achsen aktualisiert werden. Ein Beispiel aus der Testphase ist im Abbildung 5.3(a) dargestellt.

Quiver-Plot: Mit einem Quiver-Plot können Feldlinien und Feldstärken als Vektorfeld dargestellt werden. Die Länge jedes Vektors ist proportional zur Signalstärke und die Richtung gibt den Winkel des Magnetfeldes an. Diese Informationen werden aus den Messdaten der Sinus- und Cosinussignale gewonnen. Je nach Matrixgröße werden je ein Sinus- und Cosinuswert aus den Arrays einander zugeordnet und mit dem `meshgrid()` auf ein 8×8 oder 15×15 Feld aufgeteilt. Die Messdaten werden aus den Variablen `wX_sinus` und `wX_cosinus` ausgelesen. In der Funktion `FuncAnimation` wird `aniQuiver` aufgerufen, um den Plot mit neuen Daten zu aktualisieren. Ein Beispiel aus der Testphase in in Abbildung 5.3(b) dargestellt.

Kreis-Plot: Der Kreis-Plot wird als Synonym für einen Scatter Plot benutzt. Der Scatter Plot stellt für zwei übergebene Arrays die index-gleichen Werte in einem XY-Diagramm als Punkte dar. Diese Punkte-Wolken sollen sich nach einem vorigen Offsetausgleich auf einer Kreisbahn um den Nullpunkt bewegen. Jeder Sensor wird für den jeweiligen Messzeitpunkt als Punkt in dem Koordinatensystem dargestellt. Somit können auffällige Streuungen der Sensoren oder eines einzelnen Sensors festgestellt werden. Wenn die Bahn um den Nullpunkt nicht annähernd kreisförmig verläuft, ist der Offsetausgleich fehlerhaft. Zur Vorverarbeitung des Plots werden die Sinus- und Cosinuswerte in Numpy-Arrays hinterlegt. Die Arrays können jeweils 15500 Werte fassen. Somit entsteht ein dynamischer Plot, dessen Kreis sich immer weiter zeichnet. Wenn die Arrays gefüllt sind, wird der Zähler auf Null gesetzt und die neuen Werte werden, wie in einer FIFO, nachgeschoben. Die `scatter(X-Array,Y-Array,color,marker)`-Funktion wird nur Anfangs in der Plot-Funktion initialisiert. In der `animate()`-Funktion wird die Funktion `scatter.set_offsets(X-Arrays,Y-Array)` aufgerufen, die die Werte für den Scatter-Plot aktualisiert. Ein Beispiel des Plots ist in Abbildung 5.3(c) veranschaulicht. Die Kreisbahn ist deutlich zu erkennen.

5.3 Datenverarbeitung im Backend

Die Beschreibung der Datenverarbeitung ist in Unterkapitel für die einzelnen Fenster gegliedert. Es wird einführend eine Grafik zur Funktionsweise und zum Umfang der Funktionen erläutert. Im weiteren Verlauf wird auf maßgebliche Änderungen in dem Programmcode hingewiesen und Verweise zum Code gegeben.

5.3.1 Kalibrierfenster

Das Kalibrierfenster wird direkt nach dem Programmstart geöffnet, nachdem die Importe, die Initialisierung der globalen Variablen und das Durchlaufen der `main` von Statten gegangen sind. Dieser Vorgang ist im Anhang in Listing B.5 dargestellt.

Der Programmcode für das Kalibrierfenster und die zugehörigen Threads sind in Listig B.1 ausgeführt. An erster Stelle eines GUI-Elements steht dessen Initialisierung. In dieser Funktion werden Variablen gesetzt, Events verbunden und gegebenenfalls weitere Klassen initialisiert. Um mit dem Kalibrierungsfenster nun Aktionen zu erzeugen, müssen die verfügbaren Buttons mit Events verknüpft werden. Dies geschieht über sogenannte Signale. Diese verbinden einen Sender und einen Empfänger. Der Button (Sender) wird mit einer Funktion (Empfänger) über den Befehl `connect` verbunden. Wenn der Button nun vom Nutzer betätigt wird, wird im Backend die verbundene Funktion aufgerufen. Der Button „Langzeitkalibrierung“ triggert die Funktion `startLongCali()` und dort wird der `longCaliThread` gestartet. Dadurch wird die `run()`-Methode des Threads aufgerufen. Diese beinhaltet die Routine für eine Langzeitkalibrierung. Nachdem die Serielle Verbindung zum Mikrocontroller hergestellt ist, wird das Signal `Connected` ausgelöst. Dieses Signal gibt dem Kalibrierfenster an ein Dialogfenster zu öffnen. Das Dialogfenster dient dem Nutzer dazu die Kalibrierung durchzuführen. Es werden Anweisungen gegeben, wenn eine Kalibriermessung stattfindet. Die hier verwendeten Signale und Slots sind ein notwendiger Bestandteil der GUI-Programmierung. Jedes QDialog- oder QMainWindow-Fenster darf nur auf seine eigenen Funktionen zugreifen. Wenn beispielsweise der Thread die Progress-Bar des QDialog-Fensters ändert, wird die Anwendung vom System unterbrochen und die Fenster sind „eingefroren“. Somit ist der Weg über ein Signal zu wählen, das die verbundene Klasse dazu anweist, die Änderung eigenständig vorzunehmen. Aus Abbildung 5.4 geht hervor, dass vom Kalibrierfenster aus zwei Threads gestartet werden können. Der `longCaliThread` und der `progressbarThread`. Die eingezeichneten Pfeile symbolisieren ebenfalls Signale, die von den Threads an die Klasse gesendet werden. Die Klammern sind entweder leer, oder geben einen Datentyp an, der mit dem Signal an die Klasse gesendet wird. Nach der Initialisierung der Threads können die Signale mit entsprechenden Funktionen in der Klasse verbunden werden.

Für die Kalibrierung wird ein Halbachmagnet (vgl. Abbildung D.7) in 15° Schritten über dem Sensor gedreht und es wird jeweils eine Messung durchgeführt. Nun laufen zwei Prozesse parallel. Im Dialogfenster läuft ein Count-Down von drei Sekunden und dann wird das Wort „Messung“ ausgegeben. Diese Änderungen des Anzeigetextes werden durch den Thread gesteuert. Mit dem Signal `updateSec(int)` wird die Sekunde angegeben und das Signal `messung()` ändert die Anzeige zu „Messung“. Anschließend hat der Nutzer drei Sekunden Zeit das Hallbacharray zu drehen. Währenddessen wird im Thread in einer Schleife der Befehl „5000r“ gesendet. Der Befehl ist in Tabelle 4.2 erklärt. Die empfangenen Messwerte der Sinus- und Cosinussignale werden in einem Array gespeichert. Nach Abschluss der Messung werden die Array-Werte durch 24 dividiert, um auf einen Mittelwert über alle 24 Messungen zu kommen. Die Matrizen für die Cosinus- und Sinus-Werte werden an den Mikrocontroller gesendet, um dort gespeichert zu werden. Dies geschieht mit dem Befehl „6000 + Cosinuswerte + r“ und „6000 + Sinuswerte + r“. Auf dem MC werden die Strings in dieser Reihenfolge verarbeitet, damit keine Vermischung der Werte stattfinden kann. Als zweites werden die Werte in ein `.txt`-File geschrieben, um für spätere Zwecke wiederverwendet werden zu können.

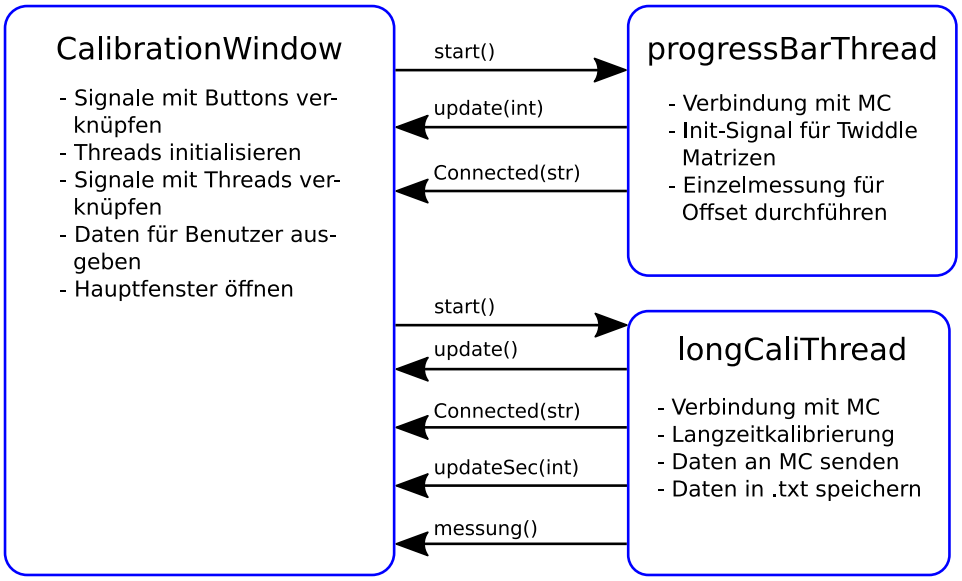


Abbildung 5.4: Grundgerüst der Klasse CalibrationWindow zur Steuerung der Sensorkalibrierung.

Der nächste Button „schnelle Kalibrierung“ startet den `progressbarThread`. Dieser sendet, nachdem die serielle Verbindung mit dem MC hergestellt wurde, „0000r“ und anschließend „1000r“, um eine einzelne Messung durchzuführen. Diese sollte mit einem Quadrupol durchgeführt werden. Ein Quadrupol besitzt vier Pole, was in der Abbildung D.9 dank der Metallspanfolie zu erkennen ist. Die markierten Linien deuten die Übergänge zwischen den Polen an. Ein Bild des Magneten ist im Anhang in der Abbildung D.8 dargestellt. Der Magnet wird verwendet, um eine kreisförmige Streuung der Offsetwerte um den Nullpunkt zu erreichen. Dabei wird das Signal `update()` an den Mikrocontroller gesendet, um den Ladebalken zu aktualisieren.

Mit der letzten Schaltfläche des Fensters kann direkt zum Hauptfenster gewechselt werden. Vorher wird jedoch geprüft, ob eventuell noch eine Kalibrierung durchgeführt wird. Dies wird mit der Abfrage `self.threadX.isRunning()` getätigt. Bei `TRUE` ist diese Aktion noch nicht möglich. Ansonsten muss sichergegangen werden, dass Werte für eine Offsetkorrektur auf dem Mikrocontroller zur Verfügung stehen. Für diesen Fall, da weder eine schnelle noch eine lange Kalibrierung vorgenommen wurde, werden die Werte aus den Textdateien verwendet. Diese werden geöffnet, eingelesen und, wie im vorigen Abschnitt beschrieben, mit dem Befehl „6000r“ an den MC gesendet. Im Anschluss wird das Hauptfenster initialisiert und geöffnet, während mit `self.close()` das Kalibrierfenster geschlossen wird.

5.3.2 Hauptfenster

In der Initialisierungsfunktion des Hauptfensters werden lokale Variablen zum Speichern der Einstellungen initialisiert und weitere Fenster und Signale verknüpft. Zu den Fenster gehören die vier Diagrammfenster und die beiden Fenster für die Filtereinstellungen. Von den Filterfenstern wird je ein Signal mit einer Funktion zum Speichern der Filtereinstellungen verbunden. Des weiteren gibt es zwei Buttons und eine CheckBox, die mit Funktionen des Hauptfensters verbunden werden. Als letzter Punkt der Initialisierung wird der `SendRecieveThread` gestartet, wie in Abbildung 5.5 durch das `start()`-Signal angedeutet. Dieser läuft autark und sorgt für die Kommunikation zwischen dem MC und dem Raspberry Pi.

Den Schaltflächen „Interpolation“ und „Diagrammauswahl“ sind keine Events zugeordnet. Das nächste Event wird getriggert, wenn der Button „Filter anpassen“ betätigt wird. Dadurch wird das Fenster zum Einstellen der Signalfilterung geöffnet. Vor dem öffnen wird jedoch überprüft, ob die CheckBox „Interpolation“ gesetzt ist. Davon ist abhängig, ob das Einstellfenster für eine 8×8 oder 15×15 Matrix geöffnet wird. Mit dem Button „Hilfe“ kann eine PDF geöffnet werden. Dies geschieht über das Ausgeben eines Terminalbefehls `os.system(qpdfview anleitung.pdf)` und öffnet die Bedienungsanleitung.

Die Schaltfläche, die ein neues Diagramm öffnet, ist der Button „Diagramm Erstellen“. In der Funktion `changeSettings()` werden die globalen Variablen verknüpft. Geschieht dies nicht, kann nur lesend und nicht schreibend auf diese zugegriffen werden. Im Anschluss folgen vier If-Abfragen. Es wird nacheinander überprüft, ob das jeweilige Fenster sichtbar ist. Bei vier offenen Fenstern wird der Nutzer darauf hingewiesen, dass bereits die maximale Fensteranzahl offen ist und erst wieder eines geschlossen werden muss. Wenn diese Bedingung nicht zutrifft werden die lokalen Einstellungen des Hauptfensters auf die globalen Fenstervariablen geschrieben. Dazu gehört das Diagramm, der gewählte Filter aus dem Filterfenster, die Information, ob die Filterwerte manuell angepasst wurden, die Koeffizienten aus der Filtermatrix und ob die Interpolation gesetzt ist. Anschließend wird der globale Befehl für das Fenster, wie in Kapitel 4.3 beschrieben, zusammengestellt. Als letzter Schritt wird das Diagrammfenster angezeigt und der Nutzer kann den „Plot“ Button betätigen.

Der größte und umfangreichste Thread ist der zum Senden der Befehle und zum Empfangen der gesendeten Daten vom MC. In der Initialisierungsfunktion werden lokale Listen erstellt. Diese dienen als Zwischenspeicher für die empfangenen Werte und überschreiben nach Empfang und Verarbeitung die globale Variable. In der `run()`-Funktion müssen ebenfalls die globalen Variablen verknüpft werden. Im Anschluss kann die serielle Verbindung mit dem Mikrocontroller hergestellt werden. In einer `while`-Schleife werden nun kontinuierlich die Befehle für die Fenster gesendet und anschließend die empfangenen Daten ausgelesen. Beim Senden wurde Anfangs der Befehl `serial.readlines()`

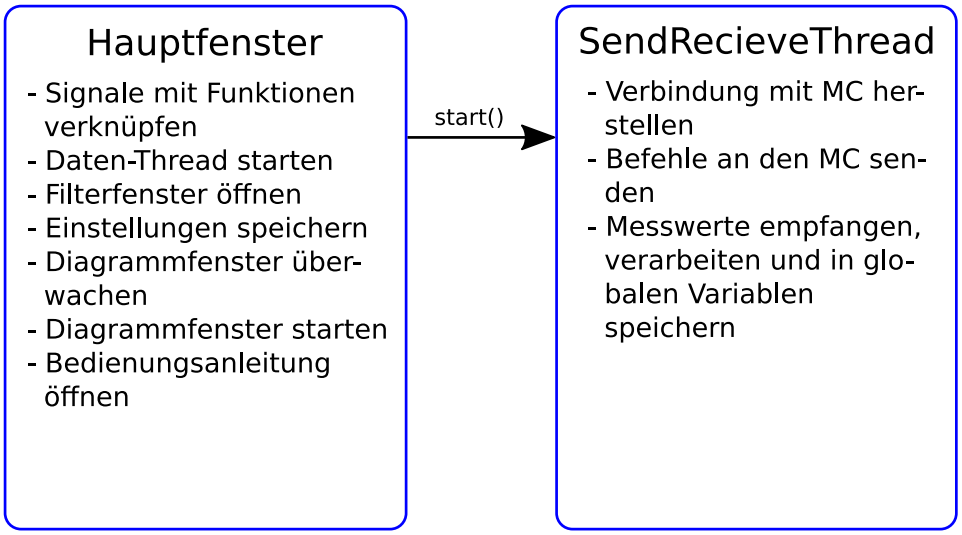


Abbildung 5.5: Grundgerüst der Klasse MainWindow sowie des SendRecieveThread zum Senden und Empfangen jeglicher Messdaten.

verwendet, um den gesamten Buffer auszulesen. Diese Routine hat aber viel Verarbeitungszeit in Anspruch genommen, da erst durch ein Timeout das Einlesen beendet wurde. Aus diesem Grund wurde der Befehl durch `serial.readline()` ersetzt, musste daher aber in einer Schleife acht bis sechzehn mal aufgerufen werden. Seitdem kommt es zu Problemen beim konvertieren der empfangenen Werte vom `String` zum `float` Format. Teilweise wurde nichts eingelesen - der Buffer war leer. Um diesen Fehler zu umgehen wird eine Abfrage nach der Anzahl an verfügbaren Daten durchgeführt. Der Fehler lag am mehrfachen Bufferzugriff. Somit wurde die Funktion `serial.read()` eingesetzt, mit der eine feste Anzahl an Bytes vom Buffer eingelesen wird. Jedoch war der Buffer teilweise leer. Vom Terminal des Raspberry Pi wurden die Befehle zu Testzwecken manuell gesendet. Es kam heraus, dass nur auf jeden zweiten gesendeten Befehl eine Antwort vom MC kam. Dies konnte auf dem PC mit angeschlossenem Mikrocontroller nicht verifiziert werden. Es lag somit am Raspberry Pi. Die Lösung ist das mehrfache Senden des Befehls mit anzuschließender Überprüfung, ob der Buffer die erwartete Anzahl an Werten aufweist - also entweder 128 oder 450 Werte. Nach erfolgreichem Einlesen muss zur Speicherung der Werte auf die globalen Variablen zugegriffen werden. Je nach gewähltem Diagramm und Interpolation passt sich die Speicherroutine dynamisch an. Eine weitere wichtige Änderung im Programmcode für die Speicherroutine war das übertragen des globalen Interpolationswertes an eine lokale, statische Variable. Falls es während der Speicherroutine somit zur Änderung des globalen Wertes kommt, bleibt die Routine für den Zyklus unbeeinflusst und es kommt nicht zu Speicherüberläufen bei den Arrays. Quellcode 5.1 zeigt den Verlauf der Datenverarbeitung vom `byte` zum `float` Format. Es wird nach dem Empfangen erst `decode()` ausgeführt, mit den Dekodierungsstyle „ascii“. Anschließend werden die Separatoren „\x00“ mit Semikola ersetzt. Anhand dieser kann

der String nun in die einzelnen Zahlenwerte separiert werden. Diese werden mit dem Befehl `append(float(value))` wird die zahl in das Float-Format gecastet und an die Liste angehängt. Als letzter Punkt werden die Buffer in die globalen Listen kopiert und dann geleert. Dadurch bleiben die globalen Speicher nie leer und es kommt zu keinen Ausfällen in den Diagrammen.

Quellcode 5.1: Ablauf der Verarbeitung empfangener Messwerte.

```
# read() <byte-item>
b'\x00-29.23\x00-27.23\x00-8.23\x00-24.23\x00-11.23\x006.77\x0011.77\x0034.77\x00'

# decode("ascii") <string-item>
"\x00-29.23\x00-27.23\x00-8.23\x00-24.23\x00-11.23\x006.77\x0011.77\x0034.77\x00"

# replace("\x00", ";") <string-item>
";-29.23;-27.23;-8.23;-24.23;-11.23;6.77;11.77;34.77;"

# split(";") <string-item>
"-29.23", "-27.23", "-8.23", "-24.23", "-11.23", "6.77", "11.77", "34.77"
```

5.3.3 Filtereinstellungen

Bei den Fenstern zur Filtereinstellung gibt es die Besonderheit, dass es ein Fenster für die Einstellungen zur 8×8 und ein Fenster für die Einstellungen zur 15×15 Matrix gibt. Beide werden in dem Hauptfenster initialisiert. In der Initialisierung werden das Layout, lokale Variablen und die listen mit den Werten des ersten Quadranten der Filtermatrix initialisiert. Des weiteren werden ebenso die klasseninternen und -externen Signalverknüpfungen eingerichtet. Nach dem öffnen des Fensters kann mit dem Filterauswahlmenü ein Tiefpass oder ein Bandpass ausgewählt werden. Wenn eine der Auswahlmöglichkeiten getroffen wurde, werden direkt die Werte in der Pin-Box-Matrix mit den entsprechenden Koeffizienten aus den lokalen Listen gefüllt. Die Änderung der Combo-Box triggert ein Signal, das die Funktion `showKoeffs()` aufruft. Hier werden die Werte der Spin-Boxen in einer Schleife mit den Werten der Listen gesetzt. Die nächste Auswahlfläche, die Check-Box „anpassen“ ist nicht nur Teil des Sendebefehls an den MC, sonder löst ebenfalls bei Betätigung ein internes Signal aus. Dieses führt die Funktion `costumCoeffsEnable()` aus, in der die Group-Box, in der sich die Spin-Boxen befinden, zur Bedienung freigeschaltet wird.

Der nächste Button ist mit der Funktion `plotKoeffs()` verbunden. Durch klicken wird diese Funktion aufgerufen und liest zu erst die aktuellen Werte der Spin-Boxen in eine Liste ein. Anschließend muss eine Filtermatrix erstellt werden. Der Kopiervorgang ist ähnlich dem auf dem Mikrocontroller, beschrieben in Kapitel 4.2 - Filterung. Anschließend wird die Matrix mit der Funktion `plt.imshow()` und einer Farbskala geplottet und der Benutzer kann die gesamte Filtermatrix prüfen. Die „Speichern“ Schaltfläche löst die Funktion `saveFilterXSettings()` aus. Nun werden jegliche Daten des Fensters gespeichert. Dazu gehört, welcher Filter ausgewählt wurde, ob die Koeffizienten angepasst wurden und wenn dies der Fall ist, werden die Koeffizienten ebenfalls gespeichert.

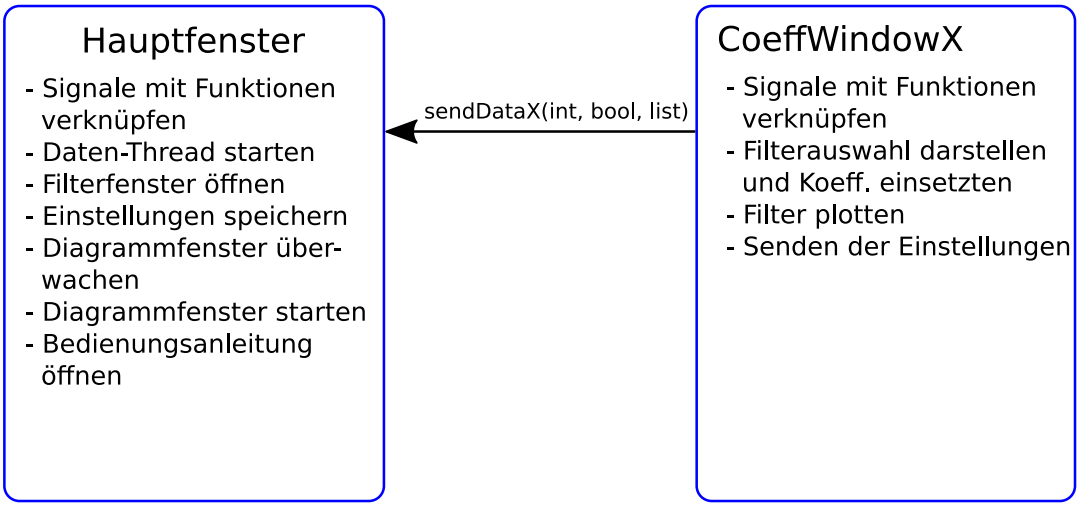


Abbildung 5.6: Grundgerüst der Klasse MainWindow sowie des CoeffWindowX zum einstellen der Filterung.

Das Signal `sendDataX(int, bool, list)`, wie in Abbildung 5.6 einzusehen, sendet die Daten an das Hauptfenster. Abschließend schließt sich das Fenster.

6 Evaluation

In dem vorletzten Kapitel dieser Arbeit wird im Anschluss an die Programmierung der Test des Programms durchgeführt. Dabei sollen sämtliche Funktionen überprüft werden. Die Inbetriebnahme dient außerdem der Betrachtung der Benutzbarkeit des Gesamtsystems. Anhand dessen kann im Anschluss eine Bedienungsanleitung angefertigt werden, die es fachfremden Benutzern erlauben soll, das System zu bedienen.

6.1 Inbetriebnahme des Gesamtsystems

Bevor das Gesamtsystem in Betrieb genommen werden kann, wird eine Liste mit den zu überprüfenden Funktionen erstellt. Dazu gehören spezifische Punkte, wie die Funktionstauglichkeit der Diagramme mit der Ausgabe sinngemäßer Messwerte, aber auch allgemeinere Anforderungen, wie das flüssige Darstellen der gesamten GUI Anwendung. In Tabelle 6.1 sind aus diesem Grund die wichtigsten Punkte der Prüfung zusammengefasst. Die Ursachenforschung für Fehler und eine Ergebnisbewertung erfolgt in Kapitel 7.1.

Zu Beginn der Inbetriebnahme wird festgestellt, dass auf dem Raspberry Pi keine Werte empfangen werden. Die Kommunikation zwischen dem PC und dem Tiva-Board funktionierte jedoch einwandfrei. Damit muss der Fehler bei dem Raspberry Pi liegen. Die Einstellungen für den seriellen Port werden überprüft und es wird festgestellt, dass nach dem Programmstart auf dem Raspberry Pi die Standarteinstellungen verändert werden. Die Anzahl der minimal einzulesenden Charakter wird auf Null gesetzt (Standard war eins). Dies wird gelöst, indem im Programmcode der Befehl `os.system('stty -icanon min 1 eof ^A -F /dev/ttyACM0')` eingefügt wird. Der Befehl wird aufgerufen, nachdem die serielle Verbindung mit dem Tiva-Board hergestellt ist. So können die nötigen Einstellungen eingerichtet und die Inbetriebnahme durchgeführt werden.

Die Langzeitkalibrierung kann erfolgreich getestet werden. Die Sekunden werden gleichmäßig runtergezählt, wobei die Anzeige zur Messung aus bleibt. Danach beginnt der Count-Down erneut. Zum Ende der 24 Messungen wird außerdem angegeben, dass die Kalibrierung beendet ist. Der Wert des Ladebalkens ist äquivalent zum Fortschritt der Messung. Die Textdateien, in denen die Werte gespeichert werden, sind mit je 64 Werten beschrieben. Die Ausgabe an den Mikrocontroller wird ebenfalls getätigt. Dies wird überprüft, indem die Befehle an dem MC auf der Konsole ausgegeben werden. Nachfolgend werden die einzelnen Diagramme getestet. Diese Tests werden mit der schnellen Kalibrierung durchgeführt, um diese ebenfalls zu testen. Von den Diagrammen werden

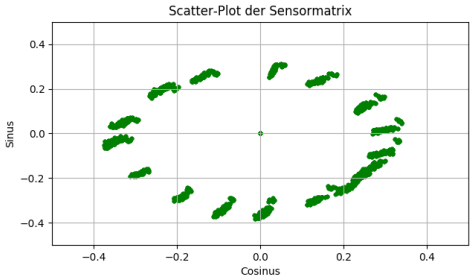
Tabelle 6.1: Stichpunkte, die bei der Inbetriebnahme des Gesamtsystems zu beachten sind.

Überprüfungsstichpunkte zur Inbetriebnahme	
• Durchführung einer Langzeitkalibrierung • Prüfung Scatter-Plot • Prüfung Scatter-Plot mit Tiefpass (TP) • Prüfung Scatter-Plot mit Bandpass (BP) • Prüfung Quiver-Plot • Prüfung Quiver-Plot mit TP • Prüfung Quiver-Plot mit BP	• Prüfung Histogramm • Prüfung Histogramm mit TP • Prüfung Histogramm mit BP • Mehrere Fenster geöffnet • Mehrere Fenster mit unterschiedlichen Plots geöffnet • Prüfung der Koeffizientenspeicherung • Allgemeiner Bedienablauf

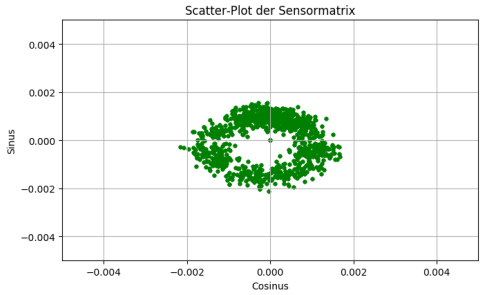
in der Beschreibung jeweils vier der fünf Messungen herangezogen. Die restlichen Bilder der Diagramme finden sich im Anhang D. Zu den letzten drei Punkten der Tabelle wird im Anschluss an den Diagrammtest Stellung genommen.

Scatter-Plot: Der erste Test beinhaltet den Plot für eine 8×8 Matrix ohne Filterung. Die Messung verläuft flüssig und die Werte bewegen sich bei einer 360° Messung in einer Kreisbahn um den Nullpunkt. Es ist eine Verschiebung in negativer X-Richtung ($-0.05V$) und in negativer Y-Richtung ($-0.02V$) in Abbildung 6.1(a) zu erkennen. Die Messung für die ungefilterte 15×15 Matrix verlaufen nicht kreisförmig, sondern in einer steigenden Geraden. In Abbildung 6.1(c) ist diese deutlich zu erkennen. Die Messung wird für gefilterte, interpolierte Matrizen wiederholt. Für die Messung der tiefpassgefilterten 8×8 Matrix ist die Kreisbahn geblieben. Der Wertebereich verringert sich und liegt statt bei `0.4V` bei `0.004V` (Abbildung 6.1(b)). Die Aufnahmen für den gefilterten Scatter-Plot der 15×15 Matrizen sind im Anhang **??** und **??**. Diese sind ebenfalls kleiner in der Achsenskalierung und liegen im Bereich $\pm 0.001V$.

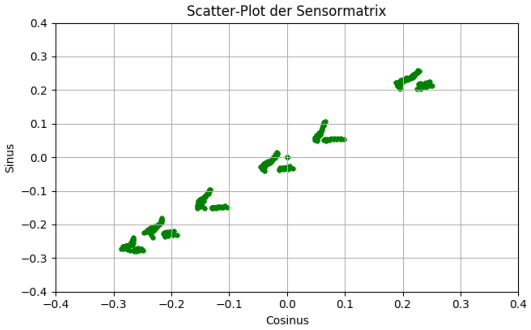
Quiver-Plot: Auch für den Quiver-Plot werden erst die ungefilterten Signale getestet. Für die 8×8 Matrix verlaufen die Pfeile parallel und mit der gleichen Länge, wie in Abbildung 6.2(a) zu erkennen ist. Bei der 15×15 Matrix ist eine leichte Krümmung (siehe Abbildung 6.2(c)) des Gesamtverlaufs zu beobachten. Besonders auffällig ist die Messung



(a) 8×8 Matrix mit ungefilterten Werten.



(b) 8×8 Matrix mit tiefpassgefilterten Werten.

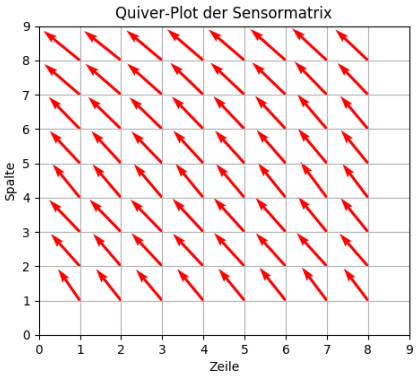


(c) 15×15 Matrix mit ungefilterten Werten.

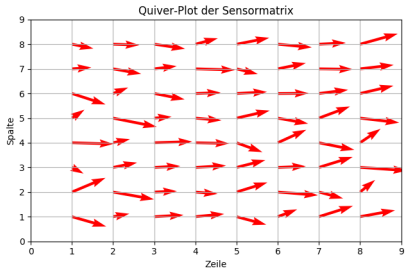
Abbildung 6.1: Ausschnitt der Messungen für den Scatter-Plot.

der tiefpassgefilterten 8×8 Matrix. Wie in Abbildung 6.2(b) einzusehen, verlaufen die Vektoren nicht mehr einheitlich parallel und weisen deutliche Differenzen in ihren Längen auf. Die Messungen für die 15×15 Matrizen sind im Anhang in Abbildung D.16 und Abbildung D.15 dargestellt. Diese weisen einen linearen Verlauf auf.

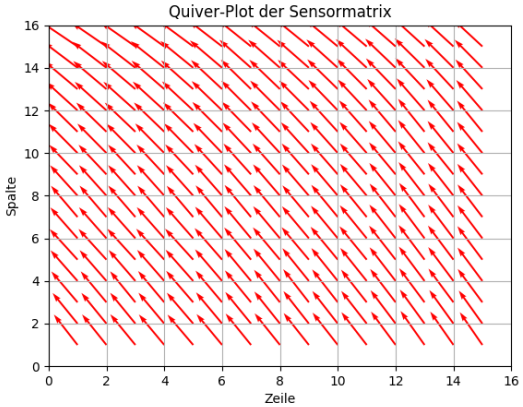
Histogramm: Für die Histogramm-Test fällt direkt auf, dass die Plot-Funktion mit autark skalierenden Achsen funktioniert. Mit der Drehung über 360° wandert die Skalierung der X-Achse mit. Beim Übergang von 360° wieder zu 0° wird die gesamte Achsenauflösung angezeigt und es gibt nur ein bis zwei Balken an jedem Ende der Achse (siehe Anhang D.10). In Abbildung 6.3(a) ist zu erkennen, dass die Winkel der ungefilterten 8×8 Matrix eine geringe Streuung aufweisen. Lediglich 8° liegen zwischen dem höchsten und dem niedrigsten Wert. Das Histogramm erinnert nur leicht an eine Gauß-Verteilung, da die 64 Werte auf nur sieben Balken aufgeteilt werden, aber die meisten Werte um den Erwartungswert liegen. Die Gauß-Verteilung wird besonders in Abbildung 6.3(c), der tiefpassgefilterten 15×15 Matrix, deutlich. Hier liegt die Streuung aller Werte bei circa 20°. Am auffälligsten ist die Verteilung der tiefpassgefilterten 8×8 Matrix. Das in Abbildung 6.3(b) dargestellte Histogramm zeigt eine hohe Streuung von etwa 70°. Ein weiterer Punkt ist das Einfrieren des Diagramms nach unbestimm-



(a) 8×8 Matrix mit ungefilterten Werten.



(b) 8×8 Matrix mit tiefpassgefilterten Werten.

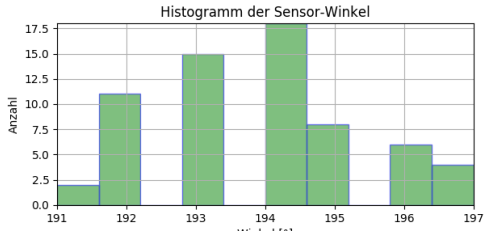


(c) 15×15 Matrix mit ungefilterten Werten.

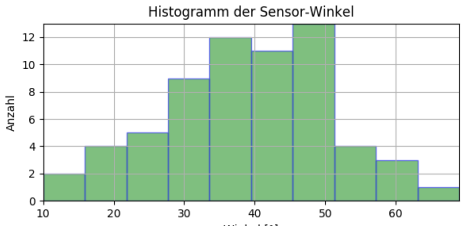
Abbildung 6.2: Ausschnitt der Messungen für den Quiver-Plot.

ter Zeit. Es wird eine Fehlermeldung im Terminal ausgegeben, dass die Verbindung mit dem `matplotlib_backends_pyqt5` nicht mehr besteht. Die Messungen für die restlichen 15×15 Matrizen sind im Anhang unter Abbildung D.11 und mit Bandpassfilterung in Abbildung D.12 einzusehen. Die Wertebereiche liegen zwischen 10° (ohne Filterung)und 20°(mit Bandpassfilterung).

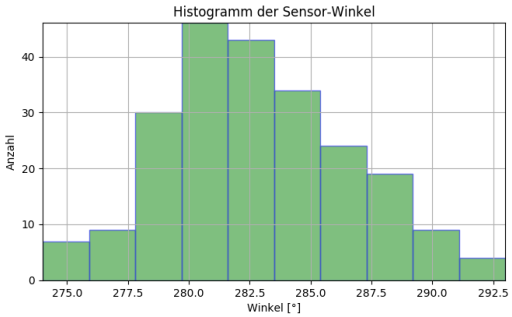
Allgemeine Beobachtungen: Zur Bedienbarkeit des Programmes lässt sich festhalten, dass die Übersicht über die Funktionen schnell erlangt wird und die Einstellungen mit wenigen Klicks vorgenommen werden können. Nachdem ein Diagramm geöffnet und der Plot gestartet wird, werden die Eingaben und Betätigungen deutlich verzögert. Je mehr Fenster geöffnet werden, so wie in Abbildung 6.4 dargestellt, desto langsamer wird das System. Einige Fenster frieren Aufgrund des oben erwähnten Verbindungsproblems nach einiger Zeit ein. Ein Neustart der Fenster bewirkt keine Änderung. Der Plot wird nicht



(a) 8×8 Matrix mit ungefilterten Werten.



(b) 8×8 Matrix mit tiefpassgefilterten Werten.



(c) 15×15 Matrix mit tiefpassgefilterten Werten.

Abbildung 6.3: Ausschnitt der Messungen für den Quiver-Plot.

dargestellt. Bei vier geöffneten Fenstern und dem Versuch ein fünftes zu öffnen, wird wie erwartet, ein Hinweis ausgegeben, dass bereits vier von vier Fenstern geöffnet sind. Aufgrund des Performance-Einbruchs die Bedienbarkeit stark gesunken. Weitere Test für die Diagramme werden nach einem Neustart des Programms durchgeführt, um andere Fehlerquellen und Einflüsse zu minimieren.

Beim öffnen zweier Quiver-Plots mit unterschiedlichen Matrixgrößen werden die Achsen nicht angepasst. In Abbildung 6.4 ist zu sehen, dass der Plot für die 15×15 Matrix die selbe Skalierung wie der Plot für die 8×8 Matrix hat. Somit können nicht alle Werte dargestellt werden. Beim Laden von zwei Histogrammen kommt es sehr schnell zum Einfrieren der Plots.

Die letzte Funktion, die getestet wurde war die Visualisierung der Filterkoeffizienten. Beim betätigen des Buttons öffnen sich vier Plot-Fenster. Eines zeigt die Koeffizienten im `imagesc()` Plot mit Farbskala und die anderen sind leer. Die dargestellten Koeffizienten entsprechen den eingestellten Werten und das Kopieren der Quadranten hat ebenfalls funktioniert, da das Diagramm für X und Y Achsensymmetrie aufweist.

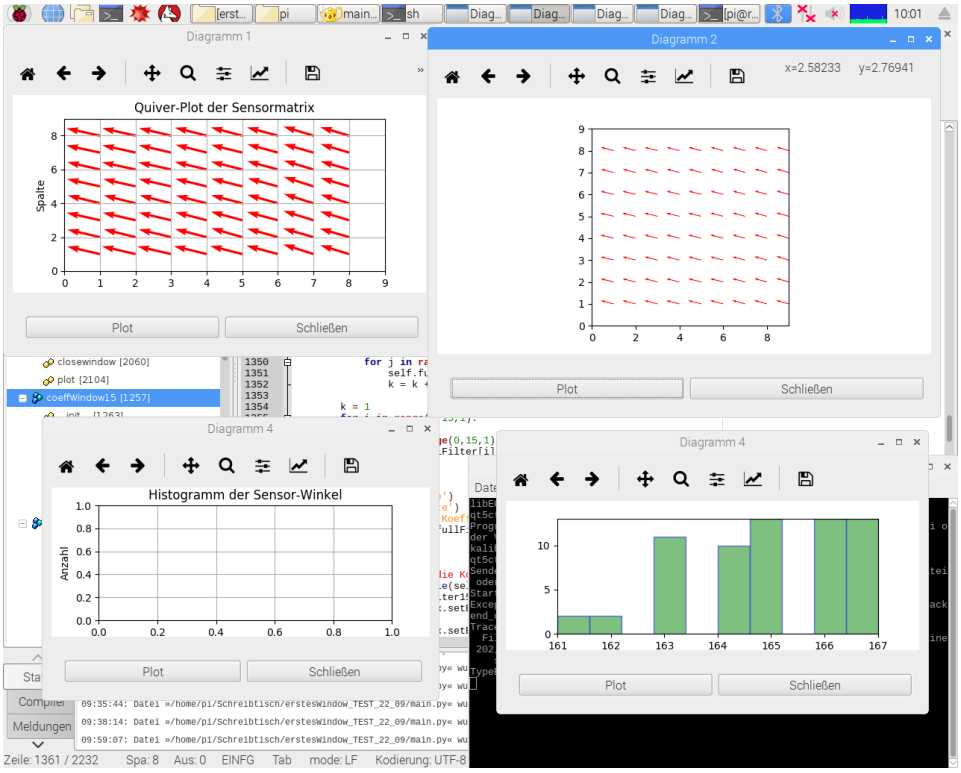


Abbildung 6.4: Darstellung des Test vom öffnen von vier Fenstern.

6.2 Entwicklung eines Benutzerhandbuches

Für die Entwicklung eines Benutzerhandbuches muss der Entwickler sein System aus der Perspektive eines unwissenden Nutzers betrachten. Der Entwickler hat das System sehr ausgiebig in der Entwurfs- und Testphase bedient und ist in den Zugriffen routiniert. Somit muss jeder Schritt beschrieben werden, um das System in Betrieb zu nehmen und zu bedienen. Des weiteren sollte eine Anleitung zum Verständnis der verwendeten Funktionen und der Diagramme gegeben werden. Nachfolgend wird eine kurze Erläuterung zu den Kapiteln gegeben und welche Informationen jeweils zu nennen sind. Für Bilder und Darstellungen wird auf die Bilder zurückgegriffen, die in dieser Arbeit verwendet werden. Dies vermindert den Aufwand der Erstellung und bietet einen höheren Wiedererkennungswert für den Benutzer, der sich vertiefend über Funktionsweisen informieren möchte.

Zu Beginn der Anleitung wird eine Einführung gegeben, um was für ein Programm es sich handelt. Was kann die Soft- und Hardware und wozu wird sie eingesetzt. Anschließend werden die Vorraussetzungen beschrieben. Dazu gehören einerseits die Software auf den Geräten und andererseits die Geräte selbst und was benötigt wird, um diese anzuschließen. Für den Softwarebereich umfasst dies zum Beispiel die benötigten Pakete, die für den Raspberry Pi heruntergeladen werden müssen, als auch eine passende IDE, mit

der der Sourcecode ausgeführt werden kann.

Die nachfolgenden Kapitel beziehen sich auf die Bedienung des Programmes und Informationen zu den verwendeten Diagrammen. Beginnend bei dem Programmstart und dem Aufbau der Geräte werden die einzelnen Fenster erläutert. Für ausführliche Informationen zur Funktionsweise des Programmes kann auf diese Arbeit verwiesen werden. Bei dem Kalibrierfenster ist es wichtig auf die beiden Kalibrierverfahren detailliert hinzuweisen. Es soll verständlich erklärt werden, was der Vorteil der beiden Möglichkeiten ist und wie die Kalibrierung durchgeführt wird. Für das Hauptfenster müssen die einzelnen Bedienfelder erläutert werden. Ein weiterer Abschnitt beschreibt das Filtereinstellungsfenster. Hier muss deutlich werden, wieso ein Viertel der erwarteten Matrix dargestellt wird. Es soll klar werden, wie die Matrix mit Werten zu füllen ist. Für die Diagrammfenster wird das Starten des Plots durch klicken des Buttons einmal explizit erwähnt. Außerdem wird die Nutzung der eigenen Matplotlib Toolbar und das Lesen der wählbaren Diagramme erklärt.

Insgesamt soll durch die Anleitung ein „roter Faden“ verlaufen, der vom Erhalt und Aufbau der Geräte bis zum Deuten des ersten Plots durch die einzelnen Schritte führt. Die Anleitung soll alle Aspekte zur Bedienung beinhalten, wodurch ein rasches Einarbeitet in die Oberfläche möglich ist. Die Bedienungsanleitung ist im Anhang im Kapitel E einzusehen.

7 Zusammenfassung und Ausblick

Nachdem im vorangegangenen Kapitel das Gesamtsystem in Betrieb genommen wurde und Auffälligkeiten beschrieben wurden, geht es nun daran, diese zu analysieren und zu bewerten. Des Weiteren werden Möglichkeiten zur Erweiterung oder Änderung des Projekts diskutiert und diese Arbeit in den Kontext des Gesamtprojekts ISAR eingeordnet. Abschließend werden die Entwicklungen und Ergebnisse in einem Fazit zusammengefasst.

7.1 Bewertung der Ergebnisse

Nachfolgend wird zu den Auffälligkeiten während der Inbetriebnahme in sofern Stellung genommen, wie sie das Programm in größerem Maße negativ beeinflussen. Lösungen zu den Problemen und Ideen zur Weiterführung werden im nachfolgenden Kapitel 7.2 stichpunktartig diskutiert.

Während der Langzeitkalibrierung wurde der Befehl zum messen nicht auf dem Dialogfenster angezeigt. Da die Messungen dennoch einwandfrei durchgeführt werden konnten und der Timer funktioniert hat, ist dieser Fehler zu vernachlässigen. Es könnten dennoch die Timings zur Anzeigezeit angepasst werden.

Bei dem Test der Diagramme sind einige Funktionsstörungen aufgefallen. Die marginale Nullpunkt-Verschiebung des Scatter-Plots für die ungefilterte 8×8 Matrix lässt sich vernachlässigen, da mit der „one-shot“-Methode kalibriert wurde. Es könnte dennoch ein Test mit den Daten der Langzeitkalibrierung zur Verifizierung durchgeführt werden. Der Fehler bei der 15×15 Matrix des Scatter-Plots konnte direkt ausfindig gemacht werden. Es wurde getestet, ob es an den interpolierten Daten oder an einer Funktion auf dem Raspberry Pi liegt. Dabei konnte festgestellt werden, dass bei Auswahl eines „Kreis-Plots“die Variable für die Interpolation auf Null gesetzt wurde. Dadurch wurden auf dem Raspberry Pi bei dem Einlesen lediglich 128 Werte aus den 225 Werten des Cosinus-Arrays ausgelesen. Dieser Umstand wurde direkt behoben und eine Testmessung durchgeführt. Diese Messung ist in Abbildung 7.1 dargestellt und weist eine leichte Verschiebung des Mittelpunktes auf der Y-Achse von etwa $-0.07V$ auf. Die Form ist kreisförmig und liegt im erwarteten Wertebereich. Die Wertebereiche der gefilterten 15×15 Matrizen sind ebenfalls sehr klein (im Bereich $\pm 0.001V$), was auf die angewandten Filter zurückzuführen ist. Eine Normierung auf Eins wäre hier nötig.

Eine Beobachtung, die sich durch alle Diagrammtests gezogen hat, ist eine Unstimmigkeit bei der Visualisierung der tiefpassgefilterten 8×8 Matrix. Hier gab es mit jeden der drei Plots Probleme, wo hingegen die ungefilterten Werte richtig dargestellt wurden.

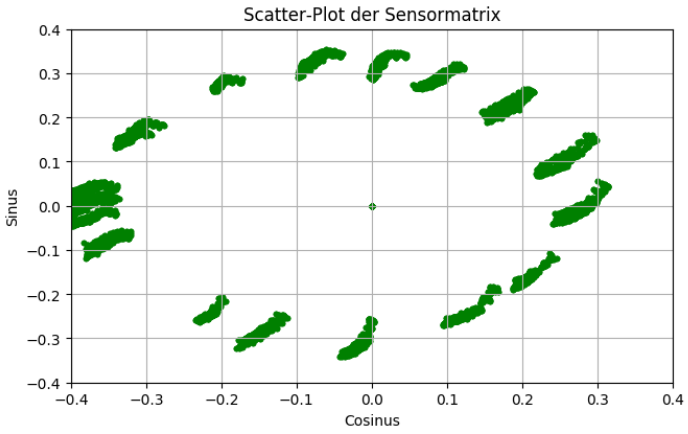


Abbildung 7.1: 15×15 Matrix mit ungefilterten Werten im Scatter-Plot.

Diese Fehler liegen an der Matrixform. Der Vorteil einer 8×8 Matrix ist die schnelle Berechnung der DFT aufgrund der Zweierpotenz. Der Nachteil ist jedoch deutlich gravierender, da der Gleichanteil durch eine Filterung nicht direkt aus der Matrix zu entfernt wird. Dieser kann nämlich durch das Null-setzten des Matrixmittelpunktes im Bildbereich entfernt werden. Bei einer 8×8 Matrix gibt es jedoch keinen einzelnen Mittelpunkt, wie bei einer 15×15 Matrix. Das entfernen von allen vier inneren Werten würde zu viele Werte der Matrix löschen. Des weiteren ist das Tiefpassfilter bei allen äußeren Matrixwerten auf Null gesetzt. Diese Maßnahme löscht ebenfalls 28 Werte aus der Matrix, wodurch nur noch 36 zur weiteren Verarbeitung und Aussage der Messung herangezogen werden können.

Weitere Auffälligkeiten sind beim Quiver-Plot in der Darstellung der 15×15 Matrix beschrieben. Die Krümmung der Feldlinien ist in einem 8×8 Plot nicht zu sehen, da die Auflösung zu gering ist. Da die Messung mit einem Würfelmagneten von $10mm \times 10mm$ durchgeführt wurde, kann die Krümmung durch eine unsaubere manuelle Haltung über dem Sensor-Array entstanden sein. Für größere Magneten fehlt an dem mechanischen Aufbau eine entsprechende Halterung, um den Magneten bei einer Messung immer in der gleichen Position und Höhe zu halten. Ein Problem mit der Achsenskalierung ist bei der gleichzeitigen Darstellung von zwei Quiver-Plots aufgefallen. Die Skalierung des zu erst geöffneten Plots wurde auch für den zweiten Plot verwendet. Dieses Problem liegt an der Variablenverwaltung von Matplotlib, da immer, zwar lokal, die selben Variablennamen verwendet wurden. Zum Test könnten andere Variablennamen vergeben werden, beispielsweise mit einer Nummerierung. Behoben werden muss der Fehler unbedingt, da eine Darstellung der gesamten Messwerte sonst nicht möglich ist. Ebenso diesem Umstand der Variablenverwaltung zuzuschreiben, ist das Öffnen von vier Fenstern für die Visualisierung der Filterkoeffizienten. Hier werden die selben Namen für die Funktions- und Achsenpointer verwendet.

Das letzte Problem, was im Zusammenhang mit den Diagrammen aufgetreten ist, war die Fehlermeldung zum Verbindungsproblem mit anschließendem einfrieren des Plots. Da dies ein sporadisch auftretender Fehler ist, ist es um so schwerer die Ursache auszumachen. Die genaue Fehlermeldung ist in Listing 7.1 angegeben. Im Internet wird beschrieben, dass dies ein Bug in der Python-Version 3.5 ist. Der `DataCursor`, der Zeiger zwischen der Figur und dem Speicherbereich der Daten für das Diagramm, wird irgendwann vom `Garbage-Collector` gelöscht [8]. Damit entsteht ein Time-Out-Error, weil keine Daten in die Figur geladen werden können.

Quellcode 7.1: Fehlermeldung zum Timout-Error von Matplotlib.

```
Exception ignored in: <bound method TimerQT.__del__ of <matplotlib.backends.
backend_qt5.TimerQT object at 0x67b2d110>>      Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/matplotlib/backends/backend_qt5.py", line 202,
in __del__self._timer.timeout.disconnect(self._on_timer)
```

Das allgegenwärtige Problem ist die Geschwindigkeit des gesamten Programms. Die Diagramme wurden nur sehr langsam neu geladen und sobald ein Diagrammfenster geöffnet wurde, hat sich die Bedienung der Oberfläche deutlich verzögert. Dies ist zum einen ein Problem, da nach dem schließen eines Fenster und erneuten Öffnen die Geschwindigkeit noch weiter abnimmt, zum anderen wird die flüssige Demonstration des Sensor-Arrays dadurch erschwert. Der eigentliche Zweck des Projektes ist damit nicht mehr umsetzbar. Dafür gibt es mehrere Ursachen. Die Erste ist der Raspberry Pi selbst, da dieser nur begrenzte Rechenkapazität aufbringt und aufwändigere grafische Anwendungen nicht flüssig darstellen kann. Des Weiteren ist die Übertragungsgeschwindigkeit zwischen den Systemen noch relativ gering. Möglich wären Übertragungsraten von über 1Mio. Baud. Anschließend könnte auch das Timing auf dem Raspberry Pi angepasst werden.

7.2 Ansätze zur Weiterführung

Die Ansätze zur Weiterführung und Ideen zur Optimierung des Programms werden nachfolgend in Stichpunkten festgehalten. Dies bietet eine bessere Übersicht und vereinfacht es den zukünftigen Nutzern, in das Projekt einzusteigen.

- **Geschwindigkeitsoptimierung:** Die Geschwindigkeit des Systems könnte nur durch ein Systemupgrade deutlich gesteigert werden. Mit größerem Kapital könnte ein älteres ThinkPad von Lenovo zum Einsatz kommen. Dies gibt es für 100 bis 150 Euro auf Ebay. Es bietet bis zu 8GB RAM und eine mehr als doppelt so hohe Taktfrequenz der 4-Kerner-CPU. Für ein Demonstrationssystem macht es keinen Sinn, dass die Diagramme nicht flüssig dargestellt werden können. Darauf ließe sich ebenso ein vollwertiges Ubuntu Betriebssystem installieren, welches deutlich mehr Möglichkeiten als das schlanke Raspbian bietet. Wenn die Plattform Raspberry Pi jedoch bestehen bleiben soll gibt es auch die Möglichkeit, die Baudraten der

Systeme zu erhöhen. Diese liegen derzeit bei 115200 und bei USB 2.0 sind Raten bis über 1Mio. Baud möglich. Anschließend könnten die Wartezeiten im Empfangs -Thread auf dem Raspberry Pi deutlich verringert werden. Ein Problem bleibt die Taktfrequenz des Raspberry Pi von standardmäßig 1,2GHz im „Performance Mode“, sowie der Arbeitsspeicher von 1GB mit einer Frequenz von 450MHz. Diese Größen lassen sich nicht verändern, außer durch eine Übertaktung. Dies erfordert aber gutes technisches Verständnis und es kann keine sichtbare Verbesserung versprochen werden. Tests zeigen, dass ein Übertakten auf 1,4GHz stabil laufen kann. Der Speichertakt kann auf bis zu 500MHz angehoben werden [25]. Im Vergleich, die CPU des ThinkPads läuft standardmäßig bei 2,6GHz und der Arbeitsspeicher bei 1600MHz.

- **Code auf dem Mikrocontroller:** Auf dem Mikrocontroller sind viele Funktionen doppelt implementiert, damit Matrizen verschiedener Größen verarbeitet werden können. Dieses wurde zur Verbesserung der Programmübersicht implementiert. Zur Steigerung der Performance und zur Minimierung des Speicherbedarfs, könnten die Funktionen dynamisch gestaltet werden. Das bedeutet die Funktionen bekommen einen Pointer zur Matrix und die Matrixgröße übermittelt.

- **Code auf dem Raspberry Pi:** Da die Programmiersprache für dieses Projekt erst erlernt worden ist, ist es durchaus zu empfehlen, den Code von bereits erfahrenen Python-Programmierern prüfen zu lassen. Es gibt viele Möglichkeiten, den Code hinsichtlich der Struktur und der Geschwindigkeit zu optimieren. Außerdem sollte, zur besseren Darstellung der Werte, eine Normierung der Filterkoeffizienten durchgeführt werden. Diese wäre für die vorhandenen, als auch für die vom Nutzer eingegebenen nötig.

- **Offsetkalibrierung:** Die Offsetkalibrierung findet derzeit als „one-shot“-Methode oder als Mittlung über eine 360° Drehung statt. Dies spiegelt aber keinen genauen Werte der Sensoren wieder, da sich der Offset eines Sensors, je nach Messbereich, ändern kann. Dafür müsste eine Messung jedes einzelnen Sensors über eine 360° Drehung aufgenommen werden. Anschließend kann die Variation der Werte begutachtet werden. Wenn es zu deutlichen Differenzen kommt, bietet es sich an, eine Korrektur je nach Quadrant durchzuführen. Dazu müssten mehrere Matrizen, mindestens eine für jeden Quadranten, berechnet werden. Der Funktion `sub_Offset()` wird dann ein Pointer zur Matrix und der derzeitige Winkel übergeben, Je nach Winkel wird der Offset subtrahiert.

- **Kommunikationsoptimierung:** Bei dem Raspberry Pi gibt es das Problem, dass die Befehle teilweise doppelt gesendet werden, bis ein Antwort vom Mikrocontroller wahrgenommen wird. Für Funktionen, bei denen mit einem Rückgabewert vom Mikrocontroller zu rechnen ist, ist diese Methode praktikabel. In der Kalibrierungsphase jedoch, beim übermitteln der berechneten Offsetmatrix, wird diese an

den Mikrocontroller gesendet, ohne zu wissen, ob die Werte auch empfangen wurden. Hier sollte ein „Handshake-Verfahren“ eingebaut werden, bei dem der MC eine Rückmeldung darüber sendet, ob die Werte empfangen wurden. Ein einfacher Vorschlag wäre, die identischen Werte zurückzusenden. Dann kann auf dem Mikrocontroller ein Test auf Gleichheit durchgeführt werden.

- **Diagrammanpassungen:** Um das Übersprechen von verschiedenen Variablen zu vermeiden, sollte jede Fensterklasse, sowie die Klassen zu den Filterfenstern, neue Namen für die Figur-Variablen bekommen. Ansonsten kann nicht ausgeschlossen werden, dass es mit der Kombination der `matplotlib.backends` Bibliothek und der Pythonversion zusammen hängt, die auch für den Fehler mit dem `DataCursor` verantwortlich ist.

- **Pythonversion ändern:** Da auf der Internetseite vorgeschlagen wird eine neue Pythonversion auf den Raspberry Pi zu spielen, wird empfohlen eine neuere Version zu wählen. Dies wäre zum Beispiel Python 3.6. Dieser Aufwand kann allerdings nicht abgeschätzt werden, da zusätzlich zu der Versionsänderung eventuell diverse Programmcodeanpassung durchzuführen sind.

7.3 Fazit und Beitrag zum Gesamtprojekt

In dieser Arbeit wurde nach der Einarbeitung in das Thema damit begonnen, eine Projektstruktur bezüglich der Anforderungen und verfügbaren Hard- und Software zu erstellen. Anschließend wurden erste Signalverarbeitungsmethoden implementiert und getestet. Die getesteten Funktionen konnten in den verfügbaren Code eingebettet werden. Neben der Entwicklung erster Verarbeitungsmethoden wurde das Frontend, die Bedienoberfläche für den Benutzer, entworfen. Die Anforderungen wurden festgehalten und schließlich mit Funktionen verknüpft, um diese verschiedene Bedienungsoberflächen zuzuordnen. Dafür wurden einige Interface-Designkriterien herangezogen. Somit war die Oberfläche entworfen, jedoch ohne jegliche Funktionen. Diese wurden im nächsten Schritt implementiert. Jeder Eingabe des Benutzers folgt eine Reaktion, ein Event, des Systems, um diese zu verarbeiten. Der nächste große Schritt war das Zusammenführen der beiden Systeme, das herstellen der Kommunikation und dessen Verarbeitung. Dazu wurden auf beiden Systeme weitere umfangreiche Subroutinen implementiert. Die Informationen mussten zwischen den Systemen geteilt werden, damit jeder Teil für sich damit arbeiten kann. Als letzter Schritt wurden die Diagrammfenster implementiert. Dies war nur möglich, nachdem die Kommunikation zwischen den Systemen einwandfrei funktioniert. Das heißt, der Nutzer kann seine Eingaben tätigen, diese werden zum Mikrocontroller gesendet, dort werden die Messwerte beschafft und verarbeitet sowie zum Raspberry Pi zurückgesendet, um dort für den Nutzer in einem Diagramm angezeigt werden zu können. Dieser Kreislauf konnte als letzter Arbeitsschritt getestet und analysiert werden.

Ein besonderes Merkmal dieser Arbeit ist, dass ein Projekt von der Idee bis hin zur

fertigen Umsetzung durchgearbeitet wurde. Dies bietet nicht nur einen Einblick in den gesamten Entwicklungsprozess, sondern auch in die Inbetriebnahme und Analyse des abgeschlossenen Projekts. Die Arbeit soll daher auch dazu dienen, sich einen Überblick über den Arbeitsumfang, den Aufwand und die Zeitplanung eines solchen Projekts zu verschaffen, um eigene Projekte besser strukturieren und planen zu können.

Das System, das in dieser Projektarbeit entworfen wurde, ist sehr kompakt und eignet sich sehr gut zum Transport. Mit diesem Aufbau lassen sich Versuche durchführen und die Interessenten auf Messen werden dazu eingeladen, selbst mit dem Programm zu arbeiten und kleine Versuche durchzuführen, um das Sensor-Array kenne zu lernen. Damit kann die Projektarbeit der Forschungsgruppe veranschaulicht und vorgeführt werden.

Die Programmiersprache Python eignet sich eben so gut für die Visualisierung von Messdaten, als auch für das Entwerfen von grafischen Bedienoberflächen. Jedoch wird für größere Projekte empfohlen, bereits erste Erfahrungen mit dieser Programmiersprache gesammelt zu haben. Diese erst in einer Projektarbeit zu erlernen erfordert Zeit und es ist aufwändig den richtigen Codingstyle anzuwenden, den Code zu Strukturieren und bereits geschwindikgeitsoptimiert zu programmieren. Daher gibt es für den hier entstandenen Pythoncode einen Optimierungsbedarf.

Abschließend ist festzuhalten, dass bei Geschwindigkeitsoptimierung durch bessere Hardware, dieses Projekt für die Forschungsgruppe ISAR eingesetzt werden kann. Ein Grundstein ist gelegt, neuer Code zur Visualisierung entwickelt und das System getestet worden. Dies bietet eine sehr gute Grundlage zur weiteren Entwicklung und zur letzten Optimierung. Das Projekt ist so strukturiert, dass an jeder Stelle leicht angesetzt werden kann, um Änderungen und Erweiterungen zu implementieren.

Literatur

[1] Marcel Beuler. „CORDIC-Algorithmus zur Auswertung elementarer Funktionen in Hardware“. In: (2008).

[2] crosstool-NG. *Toolchain Types*. Englisch. Zugriff am 18. august 2018. URL: https://crosstool-ng.github.io/docs/toolchain-types/.

[3] Curta.org. *Curta*. Englisch. Zugriff am 29. August 2018. URL: http://curta.org/wiki/CurtaAlgorithms.

[4] Klaus CJ Dietmayer. „Magnetische Sensoren auf Basis des AMR-Effekts (Magnetic sensors based on the AMR-effect)“. In: *tm Technisches Messen Plattform für Methoden, Systeme und Anwendungen der Messtechnik* 6/2001 (2001).

[5] ElectronicDesign. *What's the Difference Between TMR and GMR Sensors?* Englisch. Zugriff am 31. Juni 2018. URL: https://www.electronicdesign.com/industrial-automation/what-s-difference-between-tmr-and-gmr-sensors.

[6] Matteo Frigo und Steven G. Johnson. *Fastest Fourier Transform in the West*. Englisch. Zugriff am 03. Juli 2018. URL: http://www.fftw.org/.

[7] Texas Instruments. *Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit*. Englisch. Zugriff am 17. September 2018. 2016. URL: http://www.ti.com/lit/ug/spmu365c/spmu365c.pdf.

[8] joferkington. *Failure with Qt5*. English. Zugriff am 17. September 2018. URL: https://github.com/joferkington/mpldatacursor/issues/50.

[9] Helmuth Lemme. *Messung durch den Tunnel*. Deutsch. Zugriff am 01. August 2018. 2016. URL: https://www.elektroniknet.de/elektronik/messen-testen/messung-durch-den-tunnel-133265.html.

[10] Uwe Loreit. *Der AMR-Effekt*. Deutsch. Zugriff am 31. Juni 2018. URL: http://www.mr-sensor.de/SitesGer/AMR1.htm.

[11] Donald Norman. *The design of everyday things: Revised and expanded edition*. Constellation, 2013.

[12] Software in the Public Interest (SPI). *Debian Releases*. Englisch. Zugriff am 18. august 2018. URL: https://www.debian.org/releases/.

[13] Michael Reichenbach Marc und Schmidt. „VHDL - CORDIC Verfahren“. In: (2012).

[14] Konrad Reif. *Sensoren im Kraftfahrzeug*. Springer, 2010, S. 10–33.

[15] Sachin Rekhi. *Don Norman's Principles of Interaction Design*. Englisch. Zugriff am 28. Juni 2018. URL: https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33.

[16] Produktmanagement SENSiTEC. *AMR Sensordesigns*. Deutsch. Zugriff am 01. August 2018. URL: https://sensitec.wordpress.com/2013/06/15/amr-sensordesigns/.

[17] c-buch Sommergut. *Die Entstehung von C*. Deutsch. Zugriff am 18. august 2018. URL: http://c-buch.sommergut.de/Kapitel1/Die-Entstehung-von-C.shtml.

[18] Peter Stingl. *Mathematik für Fachhochschulen: Technik und Informatik*. Hanser Verlag, 2004.

[19] G. Krucker - Hochschule für Technik und Architektur Bern. *Tiefpassfunktion*. Deutsch. Zugriff am 20. Juni 2018. URL: http://www.krucker.ch/Skripten-Uebungen/DSPUeb/dsp_ueb2000-4.pdf.

[20] MultiDimension Technology. *MR Sensor Technology*. Englisch. Zugriff am 30. Juni 2018. URL: http://www.dowaytech.com/en/1776.html.

[21] Jack E Volder. „The CORDIC trigonometric computing technique“. In: *IRE Transactions on electronic computers* 3 (1959), S. 330–334.

[22] Wikipedia. *Digitales Filter*. Deutsch. Zugriff am 20. Juni 2018. URL: https://de.wikipedia.org/wiki/Digitales_Filter.

[23] Wikipedia. *Diskrete Fourier-Transformation*. Deutsch. Zugriff am 08. Juni 2018. URL: https://de.wikipedia.org/wiki/Diskrete_Fourier-Transformation.

[24] Wikipedia. *Magnetoresistiver Effekt*. Deutsch. Zugriff am 31. Juni 2018. URL: https://de.wikipedia.org/wiki/Magnetoresistiver_Effekt.

[25] Jack Zimmermann. *Raspberry Pi 3 Overclocking*. Englisch. Zugriff am 25. September 2018. 2016. URL: https://www.jackenhack.com/raspberry-pi-3-overclocking.

Anhang

A Quellcode auf dem Mikrocontroller

Quellcode A.1: Berechnung des Signaloffsets.

```
/**
 * In dieser Funktion wird der Offset einer Sensor-Matrix berechnet.
 * Mit Hilfe der Methode der kleinsten Quadrate werden die
 * kürzesten Abstände der Messwerte zum Ausgleichkreis berechnet
 * und der Mittelwert gebildet.
 *
 *        Name: offset_calc.c
 *  Created on: 21.08.2018
 *      Author: Simon Rindelaub
 */

#include "include.h"
#include "init_prototyp.h"

/**
 * @brief calculate_offset_8
 * @param input
 * Berechnung des Offsetvektors: c = (A^T * A)^T * A^T * b
 * Yc = -(C2/2)     <- Sinusoffset
 * Xc = -(C1/2)     <- Cosinusoffset
 * r = sqrt((C1^2 + C2^2) / 4) - C3)    <- Kreisradius
 */
void calculate_offset_8(struct complex input[8][8]){

    int k = 0, i = 0, j = 0;
    double sum = 0;
    double det_B = 0;

    // Zurücksetzten des Offsetwertes
    resultOffset[0][0] = 0;
    resultOffset[1][0] = 0;
    resultOffset[2][0] = 0;

    // Input in die 64x3 Matrix übertragen
    for (i = 0; i < 8; i++) {
        for(j = 0; j < 8; j++){
            transponiert1[k][0] = input[i][j].real; // Relteil
            transponiert1[k][1] = input[i][j].imag; // Imaginärteil
            transponiert1[k][2] = 1;                // Mit Einsen auffüllen
            k++;
        }
    }

    // Transponierte der Eingangsmatrix erstellen
    for(i = 0; i < 64; i++){
        for(k = 0; k < 3; k++){
            transponiert2[k][i] = transponiert1[i][k];
        }
    }

    // Matrixmultiplikation 64x3 * 3x64 => 3x3
    for(i = 0; i < 3; i++){
```

```
            for(j = 0; j < 3; j++){
54              for(k = 0; k < 64; k++){
                    sum += transponiert2[j][k] * transponiert1[k][i];
56              }
                B[i][j] = sum;
58              sum = 0;
            }
60      }

62      // Adjungierte von B berechnen
        adjungieren(B);
64
        // Determinate von B
66      det_B = det3x3(B);

68      // Die Determinate darf nicht Null sein
        if(det_B != 0){
70
            // Inverse von berechnen: Inv_B = 1/det(B) * adj(B)
72          for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++) {
74                  Inv_B[i][j] = (1/det_B) * Adj[i][j];
                }
76          }

78          // Matrixmultiplikation 3x3 * 3x64 => 3x64
            sum = 0;
80          for(i = 0; i < 3; i++){
                for(j = 0; j < 64; j++){
82                  for(k = 0; k < 3; k++){
                        sum += Inv_B[i][k] * transponiert2[k][j];
84                  }
                    C[i][j] = sum;
86                  sum = 0;
                }
88          }

90          // Vektor b berechnen
            for(j = 0; j < 64; j++){
92              b[j][0] = (-1) * (pow(transponiert1[j][0],2) + pow(transponiert1[j
                    ][1],2));
            }
94
            //Ergebnis berechnung des "Offsetvektors"
96          for(k = 0; k < 3; k++){
                for(j = 0; j < 64; j++){
98                  resultOffset[k][0] += C[k][j] * b[j][0];
                }
100         }

102     }
    }
104
    /**
106  * @brief calculate_offset_15
     * @param input
108  * Berechnen des Offsetvektor: c = (A^T * A)^T * A^T * b
     * Yc = -(C2/2)     <- Sinusoffset
110  * Xc = -(C1/2)     <- Cosinusoffset
     * r = sqrt((C1^2 + C2^2) / 4) - C3)    <- Kreisradius
112  */
    void calculate_offset_15(struct complex input[15][15]){
114
        int k = 0, i = 0, j = 0;
116     double sum = 0;
```

```
        double det_B = 0;
118
        // Zurücksetzten des Offsetwertes
120     resultOffset15[0][0] = 0;
        resultOffset15[1][0] = 0;
122     resultOffset15[2][0] = 0;

124     // Input in die 225x3 Matrix übertragen
        for (i = 0; i < 15; i++) {
126         for(j = 0; j < 15; j++){
                transponiert115[k][0] = input[i][j].real;    // realteil
128             transponiert115[k][1] = input[i][j].imag;    // Imaginärteil
                transponiert115[k][2] = 1;                   // Mit Einsen auffüllen
130             k++;
            }
132     }

134     // Transponierte der Eingangsmatrix erstellen
        for(i = 0; i < 225; i++){
136         for(k = 0; k < 3; k++){
                transponiert215[k][i] = transponiert115[i][k];
138         }
        }
140
        // Matrixmultiplikation 225x3 * 3x225 => 3x3
142     for(i = 0; i < 3; i++){
            for(j = 0; j < 3; j++){
144             for(k = 0; k < 225; k++){
                    sum += transponiert215[j][k] * transponiert115[k][i];
146             }
                B15[i][j] = sum;
148             sum = 0;
            }
150     }

152     // Adjungierte von B berechen
        adjungieren(B15);
154
        // Determinate von B
156     det_B = det3x3(B15);

158     // Die Determinate darf nicht Null sein
        if(det_B != 0){
160
            // Inverse von B berechnen: B_Inv = 1/det(B) * adj(B)
162         for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++) {
164                 Inv_B15[i][j] = (1/det_B) * Adj[i][j];
                }
166         }

168         // Matrixmultiplikation 3x3 * 3x225 => 3x225
            sum = 0;
170         for(i = 0; i < 3; i++){
                for(j = 0; j < 225; j++){
172                 for(k = 0; k < 3; k++){
                        sum += Inv_B15[i][k] * transponiert215[k][j];
174                 }
                    C15[i][j] = sum;
176                 sum = 0;
                }
178         }

180         // Kreisfunktions-Vektor B berechnen
            for(j = 0; j < 225; j++){
```

```
182             b15[j][0] = (-1) * (pow(transponiert115[j][0],2) + pow(transponiert115[j
                    ][1],2));
            }
184
            // Ergebnisberechnung des Offsetvektors 3x1
186         for(k = 0; k < 3; k++){
                for(j = 0; j < 225; j++){
188                 resultOffset15[k][0] += C15[k][j] * b15[j][0];
                }
190         }
        }
192 }

194 /**
     * @brief sub_offset8
196  * @param input
     * Zieht den berechneten Offset von  einer
198  * übergebenen 8x8 Messwertmatrix ab.
     */
200 void sub_offset8(struct complex input[8][8]){

202     int i=0, j=0;

204     if (kurzOffset == 1){
            for(i=0;i<8;i++){
206             for(j=0;j<8;j++){
                    input[i][j].real -= (-resultOffset[0][0] / 2);  // Xc = -(C1/2)
208                 input[i][j].imag -= (-resultOffset[1][0] / 2);  // Yc = -(C2/2)
                }
210         }
        }else{
212         for(i=0;i<8;i++){
                for(j=0;j<8;j++){
214                 input[i][j].real -= offsetMatrix[i][j].real;
                    input[i][j].imag -= offsetMatrix[i][j].imag;
216             }
            }
218     }

220 }

222 /**
     * @brief sub_offset15
224  * @param input
     * Zieht den berechneten Offset von einer
226  * übergebenen 15x15 Messwertmatrix ab.
     */
228 void sub_offset15(struct complex input[15][15]){

230     int i=0, j=0;

232     if (kurzOffset == 1){
            for(i=0;i<15;i++){
234             for(j=0;j<15;j++){
                    input[i][j].real -= (-resultOffset15[0][0] / 2);  // Xc = -(C1/2)
236                 input[i][j].imag -= (-resultOffset15[1][0] / 2);  // Yc = -(C2/2)
                }
238         }
        }else{
240         for(i=0;i<15;i++){
                for(j=0;j<15;j++){
242                 input[i][j].real -= offsetMatrix[i][j].real;
                    input[i][j].imag -= offsetMatrix[i][j].imag;
244             }
            }
```

```
246         }
    }
248
    /**
250  * @brief adjungieren
     * @param input
252  * Bildet für das Input-Array die Adjungierte.
     */
254 void adjungieren(double input[3][3]){

256     Adj[0][0] = (input[1][1]*input[2][2]) - (input[1][2]*input[2][1]);  //A(2,2)*A
            (3,3)-A(2,3)*A(3,2)
        Adj[0][1] = (input[0][2]*input[2][1]) - (input[0][1]*input[2][2]);  //A(1,3)*A
            (3,2)-A(1,2)*A(3,3)
258     Adj[0][2] = (input[0][1]*input[1][2]) - (input[0][2]*input[1][1]);  //A(1,2)*A
            (2,3)-A(1,3)*A(2,2)

260     Adj[1][0] = (input[1][2]*input[2][0]) - (input[1][0]*input[2][2]);  //A(2,3)*A
            (3,1)-A(2,1)*A(3,3)
        Adj[1][1] = (input[0][0]*input[2][2]) - (input[0][2]*input[2][0]);  //A(1,1)*A
            (3,3)-A(1,3)*A(3,1)
262     Adj[1][2] = (input[0][2]*input[1][0]) - (input[0][0]*input[1][2]);  //A(1,3)*A
            (2,1)-A(1,1)*A(2,3)

264     Adj[2][0] = (input[1][0]*input[2][1]) - (input[1][1]*input[2][0]);  //A(2,1)*A
            (3,2)-A(2,2)*A(3,1)
        Adj[2][1] = (input[0][1]*input[2][0]) - (input[0][0]*input[2][1]);  //A(1,2)*A
            (3,1)-A(1,1)*A(3,2)
266     Adj[2][2] = (input[0][0]*input[1][1]) - (input[0][1]*input[1][0]);  //A(1,1)*A
            (2,2)-A(1,2)*A(2,1)

268 }

270 /**
     * @brief det3x3
272  * @param input
     * @return det
274  * Determinante einer 3x3 Matrix berechnen.
     */
276 double det3x3(double input[3][3]){
        double det = 0;
278     det = (input[0][0]*input[1][1]*input[2][2]) + (input[0][1]*input[1][2]*input
            [2][0]) + (input[0][2]*input[1][0]*input[2][1]) - (input[2][0]*input[1][1]*
            input[0][2]) - (input[2][1]*input[1][2]*input[0][0]) - (input[2][2]*input
            [1][0]*input[0][1]);

280     return det;
    }
```

Quellcode A.2: Berechnung der Interpolation auf 15×15.

```
/**
 * Die Interpolation von einer 8x8 auf eine 15x15 Matrix
 * wird mit Hilfe der interpolieren()-Funktion
 * durchgeführt. genutzt wird eine lineare Interpolation.
 *
 *        Name: interpolation.c
 *  Created on: 21.08.2018
 *      Author: Simon Rindelaub
 */

#include "include.h"
#include "init_prototyp.h"


/**
 * @brief interpolieren
 * @param input
 * Interpoliert eine Matrix von 8x8 auf 15x15
 */
void interpolieren(struct complex input[8][8]){

    int i = 0, j = 0;

    // InputData wird auf jeden geraden Wert der Interpolierten geschrieben
    for(i=0;i<8;i++){

        for(j=0;j<8;j++){
            interpoliert15[i*2][j*2].real = input[i][j].real;
            interpoliert15[i*2][j*2].imag = input[i][j].imag;
        }
    }

    // Berechnung der Zwischenwerte der interpolierten Matrix
    for (i = 0; i < 15; i++) {
        for (j = 0; j < 15; j++) {

            if(i%2 == 0){       // gerade Zeile
                if(j%2 == 0){   // gerade Spalte

                }else{
                    // Je ein Halb der benachbarten Werte addiert sich zum neuen virtuellen Sensorwert
                    interpoliert15[i][j].real = (0.5 * interpoliert15[i][j-1].real) + (0.5 * interpoliert15[i][j+1].real);
                    interpoliert15[i][j].imag = (0.5 * interpoliert15[i][j-1].imag) + (0.5 * interpoliert15[i][j+1].imag);
                }
            }else{              // ungerade Zeile
                if(j%2 == 0){   // gerade Spalte
                    // Je ein Halb der benachbarten Werte addiert sich zum neuen virtuellen Sensorwert
                    interpoliert15[i][j].real = (0.5 * interpoliert15[i-1][j].real) + (0.5 * interpoliert15[i+1][j].real);
                    interpoliert15[i][j].imag = (0.5 * interpoliert15[i-1][j].imag) + (0.5 * interpoliert15[i+1][j].imag);
                }else{
                    // Ein Halb des benachbarten Wertes und ein Viertel der schrägen nachbarn addiert sich zum neuen virtuellen Sensorwert
                    interpoliert15[i][j].real = (0.5 * interpoliert15[i][j-1].real) + (0.25 * interpoliert15[i-1][j+1].real) + (0.25 * interpoliert15[i+1][j+1].real);
                    interpoliert15[i][j].imag = (0.5 * interpoliert15[i][j-1].imag)
```

```
                            + (0.25 * interpoliert15[i-1][j+1].imag) + (0.25 *
                            interpoliert15[i+1][j+1].imag);
54                  }
                }
56          }
        }
58  }

60  /**
     * @brief interpolierenOffset
62   * @param input
     * Interpoliert eine Offset-Matrix von 8x8 auf 15x15
64   */
    void interpolierenOffset(struct complex input[8][8]){
66
        int i = 0, j = 0;
68
        // InputData wird auf jeden geraden Wert der Interpolierten geschrieben
70      for(i=0;i<8;i++){

72          for(j=0;j<8;j++){
                offsetMatrix15[i*2][j*2].real = input[i][j].real;
74              offsetMatrix15[i*2][j*2].imag = input[i][j].imag;
            }
76      }

78      // Berechnung der Zwischenwerte der interpolierten Matrix
        for (i = 0; i < 15; i++) {
80          for (j = 0; j < 15; j++) {

82              if(i%2 == 0){       // gerade Zeile
                    if(j%2 == 0){   // gerade Spalte
84
                    }else{
86                      // Je ein Halb der benachbarten Werte addiert sich zum neuen
                            virtuellen Sensorwert
                        offsetMatrix15[i][j].real = (0.5 * offsetMatrix15[i][j-1].real)
                            + (0.5 * offsetMatrix15[i][j+1].real);
88                      offsetMatrix15[i][j].imag = (0.5 * offsetMatrix15[i][j-1].imag)
                            + (0.5 * offsetMatrix15[i][j+1].imag);
                    }
90              }else{              // ungerade Zeile
                    if(j%2 == 0){   // gerade Spalte
92                      // Je ein Halb der benachbarten Werte addiert sich zum neuen
                            virtuellen Sensorwert
                        offsetMatrix15[i][j].real = (0.5 * offsetMatrix15[i-1][j].real)
                            + (0.5 * offsetMatrix15[i+1][j].real);
94                      offsetMatrix15[i][j].imag = (0.5 * offsetMatrix15[i-1][j].imag)
                            + (0.5 * offsetMatrix15[i+1][j].imag);
                    }else{
96                      // Ein Halb des benachbarten Wertes und ein Viertel der schrä
                            gen nachbarn addiert sich zum neuen virtuellen Sensorwert
                        offsetMatrix15[i][j].real = (0.5 * offsetMatrix15[i][j-1].real)
                            + (0.25 * offsetMatrix15[i-1][j+1].real) + (0.25 *
                            offsetMatrix15[i+1][j+1].real);
98                      offsetMatrix15[i][j].imag = (0.5 * offsetMatrix15[i][j-1].imag)
                            + (0.25 * offsetMatrix15[i-1][j+1].imag) + (0.25 *
                            offsetMatrix15[i+1][j+1].imag);
                    }
100             }
            }
102     }
    }
```

Quellcode A.3: Berechnung der Fourier-Transformation und ihrer Inversen.

```
/**
 * In dieser Funktion wird die Fourier-Transformation durchgeführt.
 * Es werden außerdem die Twiddle-Matrizen und deren Inverse
 * berechnet. Zum Schluss wird noch die inverse Fourier-
 * Transformation berechnet, die nach der Filterung an-
 * gewendet werden soll.
 *
 *  - Twiddle-matrizen: Zeile 23 ff.
 *  - DFTs            : Zeile 96 ff.
 *
 *        Name: fourier_transform.c
 *  Created on: 27.08.2018
 *      Author: Simon Rindelaub
 */

#include "include.h"
#include "init_prototyp.h"


//////////////////////////////////////////
//    Berechnung der Twiddle-Matrizen   //
//////////////////////////////////////////

/**
 * @brief twiddle
 * Berechnung der Twiddle Matrix für 8x8
 */
void twiddle(void){

    int i = 0, j = 0;

    for(i=0;i<8;i++){
        for(j=0;j<8;j++){
            // Berechnung der einzelnen Twiddle-Faktoren
            W[i][j].real = cos(-2 * PI * i * (j / sqrt(64)));
            W[i][j].imag = sin(-2 * PI * (i / sqrt(64)* j));
        }
    }
}

/**
 * @brief InvTwiddle
 * Berechnung der inversen Twiddle Matrix für 8x8
 */
void InvTwiddle(void){

    int i = 0, j = 0;

    for(i=0;i<8;i++){
        for(j=0;j<8;j++){
            // Berechnung der einzelnen inversen Twiddle-Faktoren
            InvW[i][j].real = cos(2 * PI * i * (j / sqrt(64)));
            InvW[i][j].imag = sin(2 * PI * (i / sqrt(64)* j));
        }
    }
}

/**
 * @brief twiddle
 * Berechnung der Twiddle Matrix für 15x15
 */
void twiddle15(void){
```

```
64      int i = 0, j = 0;

66      for(i=0;i<15;i++){
            for(j=0;j<15;j++){
68              // Berechnung der einzelnen Twiddle-Faktoren
                W15[i][j].real = cos(-2 * PI * i * (j / sqrt(15*15)));
70              W15[i][j].imag = sin(-2 * PI * (i / sqrt(15*15)* j));
            }
72      }
    }
74
    /**
76   * @brief InvTwiddle
     * Berechnung der inversen Twiddle Matrix für 15x15
78   */
    void InvTwiddle15(void){
80
        int i = 0, j = 0;
82
        for(i=0;i<15;i++){
84          for(j=0;j<15;j++){
                // Berechnung der einzelnen Twiddle-Faktoren
86              InvW15[i][j].real = cos(2 * PI * i * (j / sqrt(15*15)));
                InvW15[i][j].imag = sin(2 * PI * (i / sqrt(15*15)* j));
88          }
        }
90  }

92  ////////////////////////////////////////
    //           Berechnung der DFTs        //
94  ////////////////////////////////////////

96  /**
     * @brief fft
98   * Berechnung der 2D-DTF mit Hilfe der Twiddle Matrix
     * Output = (W * Input) * W
100  * Aufgeteilt in zwei schleifen: g1 = W * Input, Output = g1 * W
     */
102 void fft(void){

104     int c = 0, d = 0, k = 0;
        double isum = 0, rsum = 0;
106     // Buffer für den ersten Schritt der komplexen Matrixmultiplikation
        double rBuffer[8][8], iBuffer[8][8];
108
        // Erste Schleife g1 = W * input
110     for (c = 0; c < 8; c++) {
            for (d = 0; d < 8; d++) {
112             for (k = 0; k < 8; k++) {
                    // Komplexe Multipl. Addition zum Buffer
114                 isum = isum + (W[c][k].real * inputData[k][d].imag) + (W[c][k].imag
                        * inputData[k][d].real);
                    rsum = rsum + (W[c][k].real * inputData[k][d].real) - (W[c][k].imag
                        * inputData[k][d].imag);
116             }
                iBuffer[c][d] = isum;
118             rBuffer[c][d] = rsum;
                // Zurücksetzten der Buffer
120             isum = 0;
                rsum = 0;
122         }
        }
124
        // Zurücksetzten der Buffer
```

```
126         isum = 0;
            rsum = 0;
128
            // Zweite Schleife out = g1 * W
130         for (c = 0; c < 8; c++) {
                for (d = 0; d < 8; d++) {
132                 for (k = 0; k < 8; k++) {
                        // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
                            Real: (a1*b1)-(a2*b2)
134                     // und Addition zum Buffer
                        isum = isum + (iBuffer[c][k] * W[k][d].real) + (rBuffer[c][k] * W[k
                            ][d].imag);
136                     rsum = rsum + (rBuffer[c][k] * W[k][d].real) - (iBuffer[c][k] * W[k
                            ][d].imag);
                    }
138
                    // Speichern der Werte in der DFT-Output-Matrix
140                 fftOutput8[c][d].imag = isum;
                    fftOutput8[c][d].real = rsum;
142                 // Zurücksetzten der Buffer
                    isum = 0;
144                 rsum = 0;
                }
146         }
        }
148
        /**
150      * @brief ifft
         * Berechnung der 2D IFFT mit Hilfe der inversen Twiddle-Matrix.
152      */
        void ifft(void){
154
            int c = 0, d = 0, k = 0;
156         double isum = 0, rsum = 0;
            double rBuffer[8][8], iBuffer[8][8];
158
            // Erste Schleife g1 = InvW * DFT-Output
160         for (c = 0; c < 8; c++) {
                for (d = 0; d < 8; d++) {
162                 for (k = 0; k < 8; k++) {
                        // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
                            Real: (a1*b1)-(a2*b2)
164                     // und Addition zum Buffer
                        isum = isum + (InvW[c][k].real * filterOutput8[k][d].imag) + (InvW[
                            c][k].imag * filterOutput8[k][d].real);
166                     rsum = rsum + (InvW[c][k].real * filterOutput8[k][d].real) - (InvW[
                            c][k].imag * filterOutput8[k][d].imag);
                    }
168                 iBuffer[c][d] = isum;
                    rBuffer[c][d] = rsum;
170                 // Zurücksetzten der Buffer
                    isum = 0;
172                 rsum = 0;
                }
174         }

176         // Zurücksetzten der Buffer
            isum = 0;
178         rsum = 0;

180         // Zweite Schleife out = g1 * InvW
            for (c = 0; c < 8; c++) {
182             for (d = 0; d < 8; d++) {
                    for (k = 0; k < 8; k++) {
184                     // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
```

```
                    Real: (a1*b1)-(a2*b2)
                // und Addition zum Buffer
186             isum = isum + (iBuffer[c][k] * InvW[k][d].real) + (rBuffer[c][k] *
                    InvW[k][d].imag);
                rsum = rsum + (rBuffer[c][k] * InvW[k][d].real) - (iBuffer[c][k] *
                    InvW[k][d].imag);
188         }
            // Speichern der IFFT Werte in der Output-Matrix und Normierung auf
                Matrixgröße
190         ifftOutput8[c][d].imag = isum/(8*8);
            ifftOutput8[c][d].real = rsum/(8*8);
192         // Zurücksetzten der Buffer
            isum = 0;
194         rsum = 0;
        }
196     }
    }
198
    /**
200  * @brief fft15
     * Berechnung der 2D-DTF mit Hilfe der Twiddle Matrix
202  * für ein 15x15 Array
     * Output = (W * Input) * W
204  * Aufgeteilt in zwei Schleifen: g1 = W * Input, Output = g1 * W
     */
206 void fft15(void){

208     int c,d,k;
        double isum = 0, rsum = 0;
210     double rBuffer15[15][15], iBuffer15[15][15];

212     // erste Schleife g1 = W * in
        for (c = 0; c < 15; c++) {
214         for (d = 0; d < 15; d++) {
                for (k = 0; k < 15; k++) {
216                 // Komplexe Multipl. und Addition zum Buffer
                    isum = isum + (W15[c][k].real * interpoliert15[k][d].imag) + (W15[c
                        ][k].imag * interpoliert15[k][d].real);
218                 rsum = rsum + (W15[c][k].real * interpoliert15[k][d].real) - (W15[c
                        ][k].imag * interpoliert15[k][d].imag);
                }
220             iBuffer15[c][d] = isum;
                rBuffer15[c][d] = rsum;
222             // Zurücksetzten der Buffer
                isum = 0;
224             rsum = 0;
            }
226     }

228     // Zurücksetzten der Buffer
        isum = 0;
230     rsum = 0;

232     // Zweite Schleife out = g1 * W
        for (c = 0; c < 15; c++) {
234         for (d = 0; d < 15; d++) {
                for (k = 0; k < 15; k++) {
236                 // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
                        Real: (a1*b1)-(a2*b2)
                    // und Addition zum Buffer
238                 isum = isum + (iBuffer15[c][k] * W15[k][d].real) + (rBuffer15[c][k]
                        * W15[k][d].imag);
                    rsum = rsum + (rBuffer15[c][k] * W15[k][d].real) - (iBuffer15[c][k]
                        * W15[k][d].imag);
240             }
```

```
                fftOutput15[c][d].imag = isum;
242             fftOutput15[c][d].real = rsum;
                // Zurücksetzten der Buffer
244             isum = 0;
                rsum = 0;
246         }
        }
248 }

250
    /**
252  * @brief ifft15
     * Berechnung der 2D-IFFT mit Hilfe der inversen Twiddle-Matrix
254  * für ein 15x15 Array
     */
256 void ifft15(void){

258     int c,d,k;
        double isum = 0, rsum = 0;
260     double rBuffer15[15][15], iBuffer15[15][15];

262     // Erste Schleife g1 = InvW * fft15
        for (c = 0; c < 15; c++) {
264         for (d = 0; d < 15; d++) {
                for (k = 0; k < 15; k++) {
266                 // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
                        Real: (a1*b1)-(a2*b2)
                    // und Addition zum Buffer
268                 isum = isum + (InvW15[c][k].real * filterOutput15[k][d].imag) + (
                        InvW15[c][k].imag * filterOutput15[k][d].real);
                    rsum = rsum + (InvW15[c][k].real * filterOutput15[k][d].real) - (
                        InvW15[c][k].imag * filterOutput15[k][d].imag);
270             }
                iBuffer15[c][d] = isum;
272             rBuffer15[c][d] = rsum;
                // Zurücksetzten der Buffer
274             isum = 0;
                rsum = 0;
276         }
        }
278
        // Zurücksetzten der Buffer
280     isum = 0;
        rsum = 0;
282
        // Zweite Schleife out = g1 * InvW
284     for (c = 0; c < 15; c++) {
            for (d = 0; d < 15; d++) {
286             for (k = 0; k < 15; k++) {
                    // Multiplikation zwei komplexer Zahlen - Imag: (j*a1*b2)+(j*a2*b1)
                        Real: (a1*b1)-(a2*b2)
288                 // und Addition zum Buffer
                    isum = isum + (iBuffer15[c][k] * InvW15[k][d].real) + (rBuffer15[c
                        ][k] * InvW15[k][d].imag);
290                 rsum = rsum + (rBuffer15[c][k] * InvW15[k][d].real) - (iBuffer15[c
                        ][k] * InvW15[k][d].imag);
                }
292             // Speichern der IFFT Werte in der Output-Matrix und Normierung auf
                    Matrixgröße
                ifftOutput15[c][d].imag = isum/(15*15);
294             ifftOutput15[c][d].real = rsum/(15*15);
                // Zurücksetzten der Buffer
296             isum = 0;
                rsum = 0;
298         }
```

```
            }
300     }
```

Quellcode A.4: Filterung der Messwert-Matrizen.

```
/**
 * In dieser Funktion werden die übergebenen Matrizen mit einem
 * beliebigen Filter nach Wahl gefiltert. Die vorhandenen Filter
 * werden aus dem Headerfile "filter.h" bezogen.
 *
 *        Name: digital_filter.c
 *  Created on: 21.08.2018
 *      Author: Simon Rindelaub
 */

#include "include.h"
#include "init_prototyp.h"
#include <filter.h>     // Einbinden der berechneten Filter-Matritzen

/**
 * @brief filter8
 * @param filtertyp
 * @param angepasst
 * @param input
 * Filtert dftOutput mit einem Filter nach Wahl
 */
void filter8(char filtertyp, char angepasst, struct complex input[8][8]){

    if(angepasst == '1'){
        multiply8(input, CustomFilter_8);
    }else{
        switch(filtertyp){
        case '0':               // Kein Filter ausgewählt
            multiply8(input, Bandpass_8);   // Bandpass alles Einsen
            break;
        case '1':               // TP Hammingwindow
            multiply8(input, Hamming_TP_8);
            break;
        case '2':               // Bandpassfilter
            multiply8(input, Bandpass_8);
            break;
        default:
            break;
        }
    }
}

/**
 * @brief multiply8
 * @param input
 * @param filter
 * Multiplizerit skalar zwei 8x8 Matrizen zur Filterung
 */
void multiply8(struct complex input[8][8], double filter[8][8]){

    int c = 0, d = 0;

    for (c = 0; c < 8; c++) {
      for (d = 0; d < 8; d++) {
          filterOutput8[c][d].imag = (input[c][d].imag * filter[c][d]);
          filterOutput8[c][d].real = (input[c][d].real * filter[c][d]);
      }
    }
}

/**
 * @brief filter15
```

```
     * @param filtertyp
64   * @param angepasst
     * @param input
66   * Filtert DFTOutput mit einem Filter nach Wahl
     */
68  void filter15(char filtertyp, char angepasst, struct complex input[15][15]){

70      if(angepasst == '1'){
            multiply15(input, CustomFilter_15);
72      }else{
            switch(filtertyp){
74          case '0':               // Kein Filter ausgewählt
                break;
76          case '1':               // TP Hammingwindow
                multiply15(input, Hamming_TP_15);
78              break;
            case '2':               // Bandpassfilter
80              multiply15(input, Bandpass_15);
                break;
82          default:
                break;
84          }
        }
86  }

88  /**
     * @brief multiply15
90   * @param input
     * @param filter
92   * Multiplizerit skalar zwei 15x15 Matrizen zur Filterung
     */
94  void multiply15(struct complex input[15][15], double filter[15][15]){

96      int c = 0, d = 0;

98      for (c = 0; c < 15; c++) {
          for (d = 0; d < 15; d++) {
100           filterOutput15[c][d].imag = (input[c][d].imag * filter[c][d]);
              filterOutput15[c][d].real = (input[c][d].real * filter[c][d]);
102       }
        }
104 }
```

Quellcode A.5: Filterkoeffizienten zur Filterung.

```
/**
 * Diese Headerfile beinhaltet alle Filtermatrizen, die dem Nutzer
 * zur Filterung der Eingangswerte angebotenwerden. Diese Koeffi-
 * zienten wurden im Vorfeld mit Octave berechnet.
 *
 *        Name: filter.h
 *  Created on: 21.08.2018
 *      Author: Simon Rindelaub
 */


#ifndef FILTER_H
#define FILTER_H

#endif // FILTER_H

////////////////////////////// 8x8
    ////////////////////////////////////////////////////////

double Hamming_TP_8[8][8]={
    {0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029,
        0.0040029},
    {0.0040029, 0.0428140, 0.0428140, 0.0428140, 0.0428140, 0.0428140, 0.0428140,
        0.0040029},
    {0.0040029, 0.0428140, 0.1403809, 0.1403809, 0.1403809, 0.1403809, 0.0428140,
        0.0040029},
    {0.0040029, 0.0428140, 0.1403809, 0.2350932, 0.2350932, 0.1403809, 0.0428140,
        0.0040029},
    {0.0040029, 0.0428140, 0.1403809, 0.2350932, 0.2350932, 0.1403809, 0.0428140,
        0.0040029},
    {0.0040029, 0.0428140, 0.1403809, 0.1403809, 0.1403809, 0.1403809, 0.0428140,
        0.0040029},
    {0.0040029, 0.0428140, 0.0428140, 0.0428140, 0.0428140, 0.0428140, 0.0428140,
        0.0040029},
    {0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029, 0.0040029,
        0.0040029}
};

double Bandpass_8[8][8]={
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00},
    {1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00}
};

////////////////////////////// 15x15
    ////////////////////////////////////////////////////////

double Hamming_TP_15[15][15]={
    {-0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
        -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
        -0.0028955, -0.0028955, -0.0028955},
    {-0.0028955, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
        -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
        -0.0089048, -0.0089048, -0.0028955},
    {-0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
        -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
```

```
            -0.0139550, -0.0089048, -0.0028955},
48      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.00000000, 0.00000000,
            0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
50      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.13493000, 0.13493000, 0.13493000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.21613000, 0.21613000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
52      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.21613000, 0.25000000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.21613000, 0.21613000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
54      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.13493000, 0.13493000, 0.13493000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
56      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.00000000, 0.00000000,
            0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0089048, -0.0028955},
58      {-0.0028955, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0028955},
        {-0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955}
60  };

62  double Bandpass_15[15][15]={
        {-0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955},
64      {-0.0028955, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0089048, -0.0028955},
66      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.00000000, 0.00000000,
            0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
68      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.13493000, 0.13493000, 0.13493000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.21613000, 0.21613000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
70      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.21613000, 0.00000000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
```

```
            0.21613000, 0.21613000, 0.21613000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
72      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
            0.13493000, 0.13493000, 0.13493000, 0.13493000, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.05117900, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
74      {-0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.00000000, 0.00000000,
            0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
            -0.0139550, -0.0089048, -0.0028955},
        {-0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550, -0.0139550,
            -0.0139550, -0.0089048, -0.0028955},
76      {-0.0028955, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0089048, -0.0028955},
        {-0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955}
78  };
```

Quellcode A.6: Berechnung des CORDIC-Algorithmus.

```
/**
 * In dieser Funktion findet die Berechnung des Drehwinkels statt.
 * In einer Teilfunktoin werden die Sensorwerte an den CORDIC-
 * Algorithmus übergeben. Der CORDIC liefert einen Winkel
 * zurück, der in einer Winkel-Matrix gespeichert wird.
 *
 *        Name: calculate_angle.c
 *  Created on: 21.08.2018
 *      Author: Simon Rindelaub
 */

#include "include.h"
#include "init_prototyp.h"

/**
 * @brief cordic
 * @param real, imag
 * Funktion berechnet Anhand eines X und Y Wertes
 * itterativ mit dem CORDIC Algorithmus den arctan(Y/X)
 *
 * Angeleht an den Code von Marcel Beuler der FH Giessen/Friedberg - Juni 2008
 */
double cordic(double real, double imag)
{
    // Tangenswerte für die Berechnung des Winkels im CORDIC
    double aTan[19] = {0.78540, 0.46365, 0.24498, 0.12435, 0.062419, 0.031240,
        0.015624, 0.0078123, 0.0039062, 0.0019531, 0.00097656, 0.00048828,
        0.00024414, 0.00012207, 0.000061035, 0.000030518, 0.000015259,
        0.0000076294, 0.0000038147};

    int n = 0;

    double tmpx, tmpy, VZ;
    double y = imag, x = real, z = 0.0, delta = 1;
    do{
        // Prüfung, ob der Wert sichOberhalb oder Unterhalb der X-Achse befindet
        // und entsprechende Anpassung des Vorzeichens des algorithmus
        if(y <= 0){
            VZ = 1;
        }else{
            VZ = -1;
        }
        tmpx = x - VZ*delta*y;  // berechnung des neuen X-Wertes
        tmpy = y + VZ*delta*x;  // Berechnung des neuen Y-Wertes
        x = tmpx;
        y = tmpy;
        z = z - VZ * aTan[n];   // Addition der neuen Drehung zum Gesamtwinkel
        delta = delta / 2;
        n++;
    }while (n <= 16);           // Mit 16 Schritten Genauigkeit von 0,01 Grad

    return z;
}

/**
 * @brief calcAngle8
 * @param input
 * Berechnet für jeden Sensor den Winkel
 * und einen Mittelwert über alle 64 Sensoren
 */
void calcAngle8(struct complex input[8][8]){
```

```
60      int i=0, j=0;
        int Q1 = 0, Q4 = 0;      // Quadranten zurücksetzen
62      meanAngle8 = 0;          // Winkel zurücksetzen

64
        for(i=0;i<8;i++){
66          for(j=0;j<8;j++){

68              // Wert befindet sich in Q1 (+,+)
                if(input[i][j].real >= 0 && input[i][j].imag >= 0){
70                  // Berechnung des Winkel mit dem CORDIC
                    angle8[i][j] = cordic(input[i][j].real, input[i][j].imag) * (180/PI
                        );
72                  Q1 = 1;
                }
74              // Wert befindet sich in Q2 (-,+)
                else if(input[i][j].real < 0 && input[i][j].imag >= 0){
76                  // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                        ursprünglichen Quadranten
                    angle8[i][j] = 180 - (cordic((input[i][j].real * (-1)), input[i][j
                        ].imag) * (180/PI));
78              }
                // Wert befindet sich in Q3 (-,-)
80              else if(input[i][j].real < 0 && input[i][j].imag < 0){
                    // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                        ursprünglichen Quadranten
82                  angle8[i][j] = 180 + (cordic((input[i][j].real * (-1)), (input[i][j
                        ].imag * (-1))) * (180/PI));
                }
84              // Wert befindet sich in Q4 (+,-)
                else if(input[i][j].real >= 0 && input[i][j].imag < 0){
86                  // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                        ursprünglichen Quadranten
                    angle8[i][j] = 360 - (cordic(input[i][j].real, (input[i][j].imag *
                        (-1))) * (180/PI));
88                  Q4 = 1;
                }
90          }
        }
92
        // Test, ob die Werte im ersten als auch im zweiten Quadrenten liegen
94      // Umrechnung der Werte aus dem 1.Quadrenten in den 4.
        if (Q4 == 1 && Q1 == 1){
96          for(i=0;i<8;i++){
                for(j=0;j<8;j++){
98                  if(angle8[i][j] < 180){
                        angle8[i][j] += 360;
100                 }
                }
102         }
        }
104
        // Zurücksetzen der Quadranten-Werte
106     Q1 = 0;
        Q4 = 0;
108     // Addition der Winkel
        for(i=0;i<8;i++){
110         for(j=0;j<8;j++){
                meanAngle8 += angle8[i][j];
112         }
        }
114
        // Wenn der Wert über 360 Grad liegt, muss er zwangsläufig
116     // im ersten Quadranten liegen
        meanAngle8 = meanAngle8 / 64;
```

```
118         if(meanAngle8 >= 360){
                meanAngle8 -= 360;
120         }
        }
122
        /**
124      * @brief calcAngle15
         * @param input
126      * Berechnet für jeden Sensor den Winkel
         * und einen Mittelwert über alle 225 Sensoren
128      */
        void calcAngle15(struct complex input[15][15]){
130
            int i=0, j=0;
132         int Q1 = 0, Q4 = 0;      // Quadranten zurücksetzen
            meanAngle15 = 0;         // Winkel zurücksetzen
134
            for(i=0;i<15;i++){
136             for(j=0;j<15;j++){

138                 // Wert befindet sich in Q1 (+,+)
                    if(input[i][j].real >= 0 && input[i][j].imag >= 0){
140                     // Berechnung des Winkel mit dem CORDIC
                        angle15[i][j] = cordic(input[i][j].real, input[i][j].imag) * (180/
                            PI);
142                     Q1 = 1;
                    }
144                 // Wert befindet sich in Q2 (-,+)
                    else if(input[i][j].real < 0 && input[i][j].imag >= 0){
146                     // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                            ursprünglichen Quadranten
                        angle15[i][j] = 180 - (cordic((input[i][j].real * (-1)), input[i][j
                            ].imag) * (180/PI));
148                 }
                    // Wert befindet sich in Q3 (-,-)
150                 else if(input[i][j].real < 0 && input[i][j].imag < 0){
                        // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                            ursprünglichen Quadranten
152                     angle15[i][j] = 180 + (cordic((input[i][j].real * (-1)), (input[i][
                            j].imag * (-1))) * (180/PI));
                    }
154                 // Wert befindet sich in Q4 (+,-)
                    else{
156                     // Berechnung des Winkel mit dem CORDIC und Verschiebung in den
                            ursprünglichen Quadranten
                        angle15[i][j] = 360 - (cordic(input[i][j].real, (input[i][j].imag *
                             (-1))) * (180/PI));
158                     Q4 = 1;
                    }
160                 // Addition der Winkel zur Berechnung des Mittelwertes
                    meanAngle15 += angle15[i][j];
162             }
            }
164         // Test, ob die Werte im ersten als auch im zweiten Quadrenten liegen
            // Umrechnung der Werte aus dem 1.Quadrenten in den 4.
166         if (Q4 == 1 && Q1 == 1){
                for(i=0;i<15;i++){
168                 for(j=0;j<15;j++){
                        if(angle15[i][j] < 180){
170                         angle15[i][j] += 360;
                        }
172                 }
                }
174         }
```

```
176         // Zurücksetzen der Quadranten-Werte
            Q1 = 0;
178         Q4 = 0;
            // Addition der Winkel
180         for(i=0;i<15;i++){
                for(j=0;j<15;j++){
182                 meanAngle15 += angle8[i][j];
                }
184         }

186         // Wenn der Wert über 360 Grad liegt, muss er zwangsläufig
            // im ersten Quadranten liegen
188         meanAngle15 = meanAngle15 / 225;
            if(meanAngle15 >= 360){
190             meanAngle15 -= 360;
            }
192     }
```

Quellcode A.7: Speichern der Eingangswerte von den Sensoren.

```
/**
 *        Name: Get_VAlues.c
 *  Created on: 04.04.2018
 *      Author: abegic
 *  Bearbeitet: Simon Rindelaub - Ändern der Speicher-Matrizen
 *                                und Optimierung des Codes.
 */

#include "include.h"
#include "init_prototyp.h"


/**
 * @param Get_Values
*  Verarbeiten der Sensorsignale und speichern
*  der Werte im Daten-Array für die Signalver-
*  arbeitung.
*/
void Get_Values(void)
{
    int run = 0;

    for(run=0;run<=7;run++)
    {
        enable_set_high(run);
        // Wartezeit
        SysCtlDelay(20000);
        // Triggern des ADC-Processors => Starten des Abtastvorgangs
        ADCProcessorTrigger(ADC0_BASE, 0);
        ADCProcessorTrigger(ADC1_BASE, 1);
        ADCProcessorTrigger(ADC1_BASE, 2);

        while(!ADCIntStatus(ADC1_BASE, 2, false))
        {
            // Warten bis alle Sample_Sequencer fertig sind und ADC-Werte vorliegen
        }
        // Quittierung des Interrupts
        ADCIntClear(ADC1_BASE, 2);

        enable_set_low();
        // Wartezeit
        SysCtlDelay(5000);
        // Auslesen der abgetasteten ADC-Werte und Speicherung in den globalen
            Variablen
        ADCSequenceDataGet(ADC0_BASE, 0, ADC_Values_SS0);
        ADCSequenceDataGet(ADC1_BASE, 1, ADC_Values_SS1);
        ADCSequenceDataGet(ADC1_BASE, 2, ADC_Values_SS2);

        // Schreiben der gesammelten ADC-Werte in das Daten-Array
        inputData[run][0].imag      = (double)ADC_Values_SS0[0];
        inputData[run][1].imag      = (double)ADC_Values_SS0[1];
        inputData[run][2].imag      = (double)ADC_Values_SS0[2];
        inputData[run][3].imag      = (double)ADC_Values_SS0[3];
        inputData[run][4].imag      = (double)ADC_Values_SS0[4];
        inputData[run][5].imag      = (double)ADC_Values_SS0[5];
        inputData[run][6].imag      = (double)ADC_Values_SS0[6];
        inputData[run][7].imag      = (double)ADC_Values_SS0[7];

        inputData[run][0].real      = (double)ADC_Values_SS1[0];
        inputData[run][1].real      = (double)ADC_Values_SS1[1];
        inputData[run][2].real      = (double)ADC_Values_SS1[2];
        inputData[run][3].real      = (double)ADC_Values_SS1[3];
```

```
62              inputData[run][4].real      = (double)ADC_Values_SS2[0];
                inputData[run][5].real      = (double)ADC_Values_SS2[1];
64              inputData[run][6].real      = (double)ADC_Values_SS2[2];
                inputData[run][7].real      = (double)ADC_Values_SS2[3];
66          }
        }
```

Quellcode A.8: Main-Funktion mit Befel-Verarbeitung.

```
/**
 * In der Main-Funktion wird die System-Clock initialisiert und
 * nach weiteren Initialisierungen in eine while(1) Loop
 * gewechselt. Dort wird stetig überprüft, ob im Buffer ein
 * neuer String vorliegt. Wenn der Interrupt-handler das
 * entsprechnede Flag gesetzt hat, wird der Befehl ausgeführt
 * und die angeforderten Messwerte zurückgesendet.
 *
 *      Autor: Simon Rindelaub
 *      Datum: 04.09.2018
 *      Name: main.c
 *
 **/

#include "include.h"
#include "init_prototyp.h"


 int main(void) {

     uint32_t sysClock = 0;
     int i,j,k,ptr_count, round = 0;
     double filters[64];
     char * ptr;


     // globale Variablen initialisieren
     buffer_full = 0;
     index = 0;

    // set processor to 80 MHz (intern PLL with 320 MHz)
    sysClock = SysCtlClockFreqSet(  SYSCTL_OSC_INT |
                                    SYSCTL_USE_PLL |
                                    SYSCTL_CFG_VCO_320,
                                    80000000-1);

    IntMasterDisable(); // turn off master
    enable_init_gpios();
    gpio_set_input();
    Init_ADCs();
    Init_UART(sysClock);
    IntMasterEnable();  // turn on master

    while(1)
    {

        if (buffer_full == 1){

            buffer_full = 0;

            // Verarbeitung des eingehenden Befehls vom Raspberry Pi:
            // rcv_char2[0] = Befehl            <- 0-9 (wird in SC verarbeitet)
            // rcv_char2[1] = Interpolation     <- 1 = True, 0 = False
            // rcv_char2[2] = Filter            <- 0 = kein Filter, 1 = TP, 2 = BP
            // rcv_char2[3] = Anpassen          <- 0 = False, 1 = True (Die
                Filterkoeff. wurden verändert)
            // filterValues[_] = Filterkoeffizienten vom Nutzer (4x4 oder 8x8)
            switch(rcv_char2[0]){
            case '0':                               // Initialisierung jeglicher
                Daten
                IntDisable(INT_UART0);

```

```
                twiddle();                              // Twiddlematrizen
                    initialisieren
62              InvTwiddle();
                twiddle15();
64              InvTwiddle15();

66              IntEnable(INT_UART0);
                break;
68          case '1':                                   // Kalibrierung wird ausgeführt
                IntDisable(INT_UART0);
70
                Get_Values();
72              kurzOffset = 1;                         // Es soll der kurze Offset
                    verwendet werden
                calculate_offset_8(inputData);          // berechnet Offsetvektor
74              interpolieren(inputData);
                calculate_offset_15(interpoliert15);// berechnet Offsetvektor
76
                IntEnable(INT_UART0);
78              break;
    //-----------------Winkel-Matrix angefordern---------------------//
80          case '2':
                    IntDisable(INT_UART0);
82
                    Get_Values();
84                  //Interpolation == False
                    if(rcv_char2[1] == '0'){
86                      // Offset abziehen
                        sub_offset8(inputData);
88
                        if(rcv_char2[3] == '1' || rcv_char2[2] != '0'){
90
                            // Aus dem String filterValues die Zahlen filtern
92                          ptr = strtok(filterValues, ";");
                            ptr_count = 0;
94                          while(ptr != NULL)
                            {
96                              filters[ptr_count] = strtod(ptr, NULL);
                                ptr = strtok(NULL, ";");
98                              ptr_count++;
                            }
100                         if(rcv_char2[3] == '1'){
                                // Quadranten in Filtermatrix kopieren
102                             k = 0;
                                for(i=0;i<4;i++){
104                                 for(j=0;j<4;j++){
                                        CustomFilter_8[i][j] = filters[k];
106                                     k++;
                                    }
108                             }
                                k = 0;
110                             for(i=0;i<4;i++){
                                    for(j=7;j>3;j--){
112                                     CustomFilter_8[i][j] = filters[k];
                                        k++;
114                                 }
                                }
116                             k = 0;
                                for(i=4;i<8;i++){
118                                 k++;
                                    for(j=0;j<8;j++){
120                                     CustomFilter_8[i][j] = CustomFilter_8[i-k][
                                            j];
                                    }
122                                 k++;
```

```
                    }
124                 }
                    // Eingangsmatrix fourier-transformieren
126                 fft();
                    // FFT-Output filtern
128                 filter8(rcv_char2[2], rcv_char2[3], fftOutput8);
                    // Filteroutput zurücktransformieren
130                 ifft();
                    // Winkel für 8x8 berechnen
132                 calcAngle8(ifftOutput8);
                    // Winkel-Matrix ausgeben
134                 print_angles8();
                }else{
136                 // ungefilterte Werte zur Winkelberechnung nutzen
                    // Winkel für 8x8 berechnen
138                 calcAngle8(inputData);
                    // Winkel-Matrix ausgeben
140                 print_angles8();
                }
142         }else{
                //Interpolation == True
144             // inputData auf 15x15 interpolieren
                interpolieren(inputData);
146             // Offset abziehen
                sub_offset15(interpoliert15);
148
                // Anpassen == 1 oder Filter != 0
150             if(rcv_char2[3] == '1' || rcv_char2[2] != '0'){
                    // Aus dem String filterValues die Zahlen filtern
152                 ptr = strtok(filterValues, ";");
                    ptr_count = 0;
154                 while(ptr != NULL)
                    {
156                     filters[ptr_count] = strtod(ptr, NULL);
                        ptr = strtok(NULL, ";");
158                     ptr_count++;
                    }
160                 if(rcv_char2[3] == '1'){
                        // Quadranten in Filtermatrix kopieren
162                     k = 0;
                        for(i=0;i<8;i++){
164                         for(j=0;j<8;j++){
                                CustomFilter_15[i][j] = filters[k];
166                             k++;
                            }
168                     }
                        k = 0;
170                     for(i=0;i<8;i++){
                            for(j=14;j>=7;j--){
172                             CustomFilter_15[i][j] = filters[k];
                                k++;
174                         }
                        }
176                     k = 1;
                        for(i=8;i<15;i++){
178                         k++;
                            for(j=0;j<15;j++){
180                             CustomFilter_15[i][j] = CustomFilter_8[i-k
                                    ][j];
                            }
182                         k++;
                        }
184                 }

186                 // Eingangsmatrix fourier-transformieren
```

```
                            fft15();
188                         // FFT-Output filtern
                            filter15(rcv_char2[2], rcv_char2[3], fftOutput15);
190                         // Filteroutput zurücktransformieren
                            ifft15();
192                         // Winkel für 15x15 berechnen
                            calcAngle15(ifftOutput15);
194                         // Winkel-Matrix ausgeben
                            print_angles15();
196                     }else{
                            // Winkel für 15x15 berechnen
198                         calcAngle15(interpoliert15);
                            // Winkel-Matrix ausgeben
200                         print_angles15();
                        }
202                 }
                    IntEnable(INT_UART0);
204                 break;
    //--------------------Sinus und Cosinus Matrizen anfordern------------------//
206             case '3':
                    IntDisable(INT_UART0);
208                 Get_Values();
                    if(rcv_char2[1] == '0'){//nicht interpolieren
210                        // Offset abziehen
                           sub_offset8(inputData);
212
                           if(rcv_char2[3] == '1' || rcv_char2[2] != '0'){
214                            // Aus dem String filterValues die Zahlen filtern
                               ptr = strtok(filterValues, ";");
216                            ptr_count = 0;
                               while(ptr != NULL)
218                            {
                                   filters[ptr_count] = strtod(ptr, NULL);
220                                ptr = strtok(NULL, ";");
                                   ptr_count++;
222                            }
                               // Quadranten in Filtermatrix kopieren
224                            if(rcv_char2[3] == '1'){
                                   k = 0;
226                                for(i=0;i<4;i++){
                                       for(j=0;j<4;j++){
228                                        CustomFilter_8[i][j] = filters[k];
                                           k++;
230                                    }
                                   }
232                                k = 0;
                                   for(i=0;i<4;i++){
234                                    for(j=7;j>3;j--){
                                           CustomFilter_8[i][j] = filters[k];
236                                        k++;
                                       }
238                                }
                                   k = 0;
240                                for(i=4;i<8;i++){
                                       k++;
242                                    for(j=0;j<8;j++){
                                           CustomFilter_8[i][j] = CustomFilter_8[i-
                                               k][j];
244                                    }
                                       k++;
246                                }
                               }
248
                               // Eingangsmatrix fourier-transformieren
250                            fft();
```

```
                    // FFT-Output filtern
252                 filter8(rcv_char2[2], rcv_char2[3], fftOutput8);
                    // Filteroutput zurücktransformieren
254                 ifft();
                    // gefilterte Werte zum Raspberry Pi senden
256                 vals_to_terminal8(ifftOutput8);
                }else{
258                 // ungefilterte Werte zum Raspberry Pi senden
                vals_to_terminal8(inputData);
260             }
            }else{//Interpolation == True
262             // inputData auf 15x15 interpolieren
                interpolieren(inputData);
264             // Offset abziehen
                sub_offset15(interpoliert15);
266
                // Anpassen == 1 oder Filter != 0
268             if(rcv_char2[3] == '1' || rcv_char2[2] != '0'){
                    // Aus dem String filterValues die Zahlen filtern
270                 ptr = strtok(filterValues, ";");
                    ptr_count = 0;
272                 while(ptr != NULL)
                    {
274                     filters[ptr_count] = strtod(ptr, NULL);
                        ptr = strtok(NULL, ";");
276                     ptr_count++;
                    }
278                 // Quadranten in Filtermatrix kopieren
                    if(rcv_char2[3] == '1'){
280                     k = 0;
                        for(i=0;i<8;i++){
282                         for(j=0;j<8;j++){
                                CustomFilter_15[i][j] = filters[k];
284                             k++;
                            }
286                     }
                        k = 0;
288                     for(i=0;i<8;i++){
                            for(j=14;j>=7;j--){
290                             CustomFilter_15[i][j] = filters[k];
                                k++;
292                         }
                        }
294                     k = 1;
                        for(i=8;i<15;i++){
296                         k++;
                            for(j=0;j<15;j++){
298                             CustomFilter_15[i][j] = CustomFilter_15[
                                    i-k][j];
                            }
300                         k++;
                        }
302                 }

304                 // Eingangsmatrix fourier-transformieren
                    fft15();
306                 // FFT-Output filtern
                    filter15(rcv_char2[2], rcv_char2[3], fftOutput15);
308                 // Filteroutput zurücktransformieren
                    ifft15();
310                 // gefilterte Werte zum Raspberry Pi senden
                    vals_to_terminal15(ifftOutput15);
312             }else{
                    // Rohe Daten zum Raspberry Pi senden
314                 vals_to_terminal15(interpoliert15);
```

```
                        }
316                     }
                        IntEnable(INT_UART0);
318                     break;
    //--------------------Einzelnen Winkel senden-------------------------//
320                 case '4':
                        IntDisable(INT_UART0);
322                     Get_Values();
                        sub_offset8(inputData);
324                     // Mittelwert des Winkels berechnen
                        calcAngle8(inputData);
326                     // Winkel zum Pi senden
                        calibration_to_terminal(meanAngle8);
328                     IntEnable(INT_UART0);
                        break;
330 //----------------Langzeitkalibrierung-----------------------------------//
                    case '5':
332                     IntDisable(INT_UART0);
                        Get_Values();
334                     // Rohdaten zum Raspberry senden
                        vals_to_terminal8(inputData);
336                     IntEnable(INT_UART0);
                        break;
338 //--------------------Kalibrierungsmatrix vom RasPi empfangen--------------------//
                    case '6':
340                     IntDisable(INT_UART0);
                        kurzOffset = 0;
342                     round += 1;
                        ptr = strtok(filterValues, ";");
344                     ptr_count = 0;
                        // einlesen und filtern der Zahlen aus dem String
346                     while(ptr != NULL)
                        {
348                         filters[ptr_count] = strtod(ptr, NULL);
                            ptr = strtok(NULL, ";");
350                         ptr_count++;
                        }
352                     if(round==1){
                            // speichern realteil
354                         k=0;
                            for(i=0;i<8;i++){
356                             for(j=0;j<8;j++){
                                    offsetMatrix[i][j].real = filters[k];
358                                 k++;
                                }
360                         }
                        }else{
362                         // speichern imagteil
                            k=0;
364                         for(i=0;i<8;i++){
                                for(j=0;j<8;j++){
366                                 offsetMatrix[i][j].imag = filters[k];
                                    k++;
368                             }
                            }
370                     }
                        // speichern der Koeffs
372                     IntEnable(INT_UART0);
                        break;
374                 default:
                        break;
376                 }
            // leeren der Arrays
378         memset(rcv_char2, 0, sizeof(rcv_char2));
            memset(filterValues, 0, sizeof(filterValues));
```

```
380                 }
            }
382     }
```

Quellcode A.9: Ausgabe der Berechnungswerte an das UART-Terminal.

```
/**
 *        Name: vals_to_terminal.c
 *  Created on: 04.04.2018
 *      Author: abegic
 *  Bearbeitet: Simon Rindelaub - Hinzufügen neuer Funktionen:
 *                                - vals_to_terminal15
 *                                - clibration_to_terminal
 *                                - print_angles8
 *                                - print_angles15
 */

#include "include.h"
#include "init_prototyp.h"

 /**
  * @brief vals_to_terminal8
  * @param input
  * Sendet die übergebene 8x8 Matrix an das UART Terminal
  */
void vals_to_terminal8(struct complex input[8][8])
{
    // Zwischenspeicher für printf-Funktionen
    char buffer [100];
    int len = 0;
    int i = 0;
    int j = 0;

    for(i = 0 ; i<=7; i++)
    {
        for(j = 7 ; j>=0; j--)
        {
            if(j>0)
            {
                len = sprintf(buffer,"%4.2f", input[i][j].real);
                UARTSend((uint8_t*)buffer,len);
                SysCtlDelay(10);
            }
            else
            {
                // Mit "\n" wird der Abschluss einer Line signalisiert
                // Dies wird für die Serial.readline() Funktion benötigt
                len = sprintf(buffer,"%4.2f", input[i][j].real);
                UARTSend((uint8_t*)buffer,len);
                SysCtlDelay(10);
            }
        }
    }

    for(i = 0 ; i<=7; i++)
    {
        for(j = 7 ; j>=0; j--)
        {
            if(j>0)
            {
                len = sprintf(buffer,"%4.2f", input[i][j].imag);
                UARTSend((uint8_t*)buffer,len);
                SysCtlDelay(10);
            }
            else
            {
                // Mit "\n" wird der Abschluss einer Line signalisiert
                // Dies wird für die Serial.readline() Funktion benötigt
```

```
                len = sprintf(buffer,"%4.2f", input[i][j].imag);
64              UARTSend((uint8_t*)buffer,len);
                SysCtlDelay(10);
66          }
        }
68  }
70  /**
72   * @brief vals_to_terminal15
     * @param input
74   * Sendet die übergebene 15x15 Matrix an das UART Terminal
     */
76  void vals_to_terminal15(struct complex input[15][15])
    {
78      // Zwischenspeicher für printf-Funktionen
        char buffer [100];
80      int len = 0;
        int i = 0;
82      int j = 0;

84      for(i = 0 ; i<=14; i++)
        {
86          for(j = 14 ; j>=0; j--)
            {
88              if(j>0)
                {
90                  len = sprintf(buffer,"%4.2f", input[i][j].real);
                    UARTSend((uint8_t*)buffer,len);
92                  SysCtlDelay(10);
                }
94              else
                {
96                  // Mit "\n" wird der Abschluss einer Line signalisiert
                    // Dies wird für die Serial.readline() Funktion benötigt
98                  len = sprintf(buffer,"%4.2f", input[i][j].real);
                    UARTSend((uint8_t*)buffer,len);
100                 SysCtlDelay(10);
                }
102         }
        }
104
        for(i = 0 ; i<=14; i++)
106     {
            for(j = 14 ; j>=0; j--)
108         {
                if(j>0)
110             {
                    len = sprintf(buffer,"%4.2f", input[i][j].imag);
112                 UARTSend((uint8_t*)buffer,len);
                    SysCtlDelay(10);
114             }
                else
116             {
                    // Mit "\n" wird der Abschluss einer Line signalisiert
118                 // Dies wird für die Serial.readline() Funktion benötigt
                    len = sprintf(buffer,"%4.2f", input[i][j].imag);
120                 UARTSend((uint8_t*)buffer,len);
                    SysCtlDelay(10);
122             }
            }
124     }
    }
126
    /**
```

```
128  * @brief calibration_to_terminal
     * @param angle
130  * Sendet die übergebenen winkel an das UART Terminal
     */
132 void calibration_to_terminal(double angle){
        // Zwischenspeicher für printf-Funktionen
134     char buffer [10];
        int len = 0;
136
        len = sprintf(buffer,"%3.1f␣", angle);
138     UARTSend((uint8_t*)buffer,len);
        SysCtlDelay(10);
140
    }
142
    /**
144  * @brief print_angles8
     * Gibt die Winkel-Matrix für das 8x8 Sensorfeld aus
146  */
    void print_angles8(void){
148
        char buffer [100];
150     int len = 0;
        int i = 0;
152     int j = 0;

154     for(i = 0 ; i<8; i++)
        {
156         for(j = 0 ; j<8; j++)
            {
158             if(j<7)
                {
160                 len = sprintf(buffer,"%3.2f", angle8[i][j]);
                    UARTSend((uint8_t*)buffer,len);
162                 SysCtlDelay(50);
                }
164             else
                {
166                 // Mit "\n" wird der Abschluss einer Line signalisiert
                    // Dies wird für die Serial.readline() Funktion benötigt
168                 len = sprintf(buffer,"%3.2f", angle8[i][j]);
                    UARTSend((uint8_t*)buffer,len);
170                 SysCtlDelay(50);
                }
172         }
        }
174 }

176 /**
     * @brief print_angles15
178  * Gibt die Winkel-Matrix für das 15x15 Sensorfeld aus
     */
180 void print_angles15(void){

182     char buffer [100];
        int len = 0;
184     int i = 0;
        int j = 0;
186
        for(i = 0 ; i<=14; i++)
188     {
            for(j = 0 ; j<=14; j++)
190         {
                if(j<14)
192             {
```

```
                    len = sprintf(buffer,"%3.2f", angle15[i][j]);
194                 UARTSend((uint8_t*)buffer,len);
                    SysCtlDelay(10);
196             }
                else
198             {
                    // Mit "\n" wird der Abschluss einer Line signalisiert
200                 // Dies wird für die Serial.readline() Funktion benötigt
                    len = sprintf(buffer,"%3.2f", angle15[i][j]);
202                 UARTSend((uint8_t*)buffer,len);
                    SysCtlDelay(10);
204             }
            }
206     }
    }
```

Quellcode A.10: Optimierung des Interrupt-Handlers

```
/*
 *        Name: Interrupt_Handler.c
 *  Created on: 04.04.2018
 *      Author: abegic
 *  Bearbeitet: Simon Rindelaub - Entfernen der while-Schleife und
 *                                Umbau auf einzelnen Char-Empfang.
 */

#include "include.h"
#include "init_prototyp.h"

/**
 * @brief UARTIntHandler
 * Verarbeitet die eingehenden Befehle und setzt ein
 * Flag, sobal der empfangene String fertig eingelesen
 * wurde. Damit kann in der Main die Verarbeitung
 * beginnen.
 */
void UARTIntHandler(void)
{

    uint32_t ui32Status;
    int32_t rcv_char;
    char empfang;

    // Get the interrrupt status.
    ui32Status = UARTIntStatus(UART0_BASE, true);
    // Clear the asserted interrupts.
    UARTIntClear(UART0_BASE, ui32Status);

    #if 0
        ROM_UARTCharPutNonBlocking(UART0_BASE,
        ROM_UARTCharGetNonBlocking(UART0_BASE));
    #else
        rcv_char = UARTCharGetNonBlocking(UART0_BASE);
        empfang = (unsigned char) rcv_char;
    #endif

    if (empfang == 'r'){            // Abbruchbedingung ein "r" am Ende des Stings
       rcv_char2[4] = 0;            // Terminieren mit \0
       filterValues[index-4] = 0;   // Terminieren mit \0
       buffer_full = 1;             // Flag für die main zum Start der Verarbeitung
       index = 0;
    }else if(index < 4){            // Speichern des Befehls
        rcv_char2[index] = empfang;
        index++;
       // Speichern der Filterkoeffizienten
    }else if(index > 3 && (index-4) < (sizeof(filterValues)-1)){
        filterValues[index-4] = empfang;
        index++;
    }
}


/**
 * @brief Init_UART
 * Initiiert die UART-Verbindung und Einstellungen
 */
void Init_UART(uint32_t takt)
{

    // Enable the GPIO Peripheral used by the UART.
```

```
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
64      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

66      // Configure GPIO Pins for UART mode.
        GPIOPinConfigure(GPIO_PA0_U0RX);
68      GPIOPinConfigure(GPIO_PA1_U0TX);
        GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
70
        // Initialize the UART for console I/O.
72      UARTConfigSetExpClk(UART0_BASE, takt, 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
74                           UART_CONFIG_PAR_NONE));
        // Enable the UART interrupt.
76      IntEnable(INT_UART0);
        UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
78      UARTIntRegister(UART0_BASE, UARTIntHandler);
    }
```

Quellcode A.11: Include der Bibliotheken und globaler Variablen.

```
/*
 * include.h
 *
 *  Created on: 03.04.2018
 *      Author: abegic
 *  Bearbeitet: Simon Rindelaub
 */

#ifndef INCLUDE_H_
#define INCLUDE_H_

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "inc/hw_timer.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include <string.h>
#include "driverlib/fpu.h"
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/adc.h"
#include "inc/hw_gpio.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "math.h"
#include "string.h"
#include "stdio.h"

#define PI 3.14159265
#define VDD 3.3
#define NBITS 1
#define RAD_TO_ANGLE 57.29577950560105 //  =>  180.0 / PI

#define SIN_1 ADC_CTL_CH9
#define SIN_2 ADC_CTL_CH10
#define SIN_3 ADC_CTL_CH11
#define SIN_4 ADC_CTL_CH13
#define SIN_5 ADC_CTL_CH16
#define SIN_6 ADC_CTL_CH17
#define SIN_7 ADC_CTL_CH18
```

```
#define SIN_8 ADC_CTL_CH19

#define COS_1 ADC_CTL_CH0
#define COS_2 ADC_CTL_CH1
#define COS_3 ADC_CTL_CH2
#define COS_4 ADC_CTL_CH3
#define COS_5 ADC_CTL_CH4
#define COS_6 ADC_CTL_CH6
#define COS_7 ADC_CTL_CH7
#define COS_8 ADC_CTL_CH8

#define HIGH 0xFF
#define LOW  0x00

// Arrays für die ADC-Werte
unsigned int ADC_Values_SS0[8];
unsigned int ADC_Values_SS1[4];
unsigned int ADC_Values_SS2[4];

struct complex{                                 // komplexe Zahl
    double real;
    double imag;
};

struct complex inputData[8][8];                 // Matrix mit 8x8 komplexen Zahlen

double meanAngle8;                              // Mittelwert der 8x8
    Sensorenwinkel
double meanAngle15;                             // Mittelwert der 15x15
    Sensorewinkel
double angle8[8][8];                            // Winkel der 8x8 Matrix Sensoren
double angle15[15][15];                         // Winkel der 15x15 Matrix Sensoren

double transponiert2[3][64];                    // 3x64 Matrix
double transponiert1[64][3];                    // 64x3 Matrix
double transponiert215[3][225];                 // 3x225 Matrix
double transponiert115[225][3];                 // 225x3 Matrix
double Adj[3][3];
double B[3][3], B15[3][3], Inv_B[3][3], Inv_B15[3][3], C[3][64], C15[3][225], b
    [64][1], b15[225][1], resultOffset[3][1],resultOffset15[3][1]; // Matrizen zur
    Offsetberechnung

struct complex W[8][8];                         // Twiddle Matrix 8x8
struct complex InvW[8][8];                      // Inverse Twiddle Matrix 8x8
struct complex W15[15][15];                     // Twiddle Matrix 15x15
struct complex InvW15[15][15];                  // Inverse Twiddle Matrix 15x15

struct complex fftOutput8[8][8];                // Output Daten der FFT 8x8
struct complex ifftOutput8[8][8];               // Output Daten der IFFT 8x8
struct complex fftOutput15[15][15];             // Output Daten der FFT 15x15
struct complex ifftOutput15[15][15];            // Output Daten der IFFT 15x15

struct complex filterOutput8[8][8];             // Output der Filterung im 8x8
    Bildbereich
struct complex filterOutput15[15][15];          // Output der Filterung im 15x15
    Bildbereich

struct complex interpoliert15[15][15];          // Interpolierte Matrix -> 15x15

double CustomFilter_8[8][8];                    // Filter-Matrizen
double CustomFilter_15[15][15];

volatile uint8_t buffer_full;                   // Gibt an, ob der Inputbuffer
    fertig gefüllt ist
int index;                                      // globaler Index für Kommunikation
```

```
        zwischen Interrupt und Main
    char rcv_char2[5];
122 char filterValues[550];

124 int kurzOffset;                                 // Gibt an, ob mit schnellem Offset
        gerechnet wird
    int langOffset;                                 // Gibt an, ob mit langem Offset
        gerechnet wird
126
    struct complex offsetMatrix[8][8];              // Offsetmatrix aus
        Langzeitkalibrierung
128 struct complex offsetMatrix15[15][15];          // Offsetmatrix aus
        Langzeitkalibrierung

130 #endif /* INCLUDE_H_ */
```

Quellcode A.12: Include der Funktions-Prototypen.

```
/**
 *        Name: init_prototyp.h
 *  Created on: 04.04.2018
 *      Author: abegic
 *  Bearbeitet: Simon Rindelaub - Ordnen der Prototypen zu den C-Datein
 *                                und hinzufügen neuer Prototypen.
 */

#ifndef INIT_PROTOTYP_H_
#define INIT_PROTOTYP_H_

//----------Controller-Steuerungs-Methoden----------------//
void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count);
void enable_init_gpios(void);
void gpio_set_input(void);
void enable_set_low(void);
void Init_UART(uint32_t takt);
void Init_ADCs(void);
void enable_set_high(int column);
void Get_Values(void);
void vals_to_terminal(void);
void UARTIntHandler(void);

//--------------vals_to_terminal.c---------------------//
// Sendet Winkel-Matrix zum Terminal
void print_angles8(void);
// Sendet Winkel-Matrix 15x15 zum Terminal
void print_angles15(void);
// Sendet den Kalibrierungswinkel zum Terminal
void calibration_to_terminal(double angle);
// Sendet übergebenes 8x8 zum Terminal
void vals_to_terminal8(struct complex input[8][8]);
// Sendet übergebenes 15x15 zum Terminal
void vals_to_terminal15(struct complex input[15][15]);

//------------calculate_angle.c-----------------//
// CORDIC Algorithmus
double cordic(double real, double imag);
// Winkelrechnung für 8x8 Matrix
void calcAngle8(struct complex input[8][8]);
// Winkelrechnung für 15x15 Matrix
void calcAngle15(struct complex input[15][15]);

//------------offset_calc.c-----------------//
// Zieht den Berechneten Offset von 8x8 Messwertmatrix ab
void sub_offset8(struct complex input[8][8]);
// Zieht den Berechneten Offset von 15x15 Messwertmatrix ab
void sub_offset15(struct complex input[15][15]);
// Adjungiert die übergebene Matrix
void adjungieren(double input[3][3]);
// Bildet die Determinate der übergebenen matrix
double det3x3(double input[3][3]);
// Berechnet den Offsetvektor einer Matrix 8x8
void calculate_offset_8(struct complex input[8][8]);
// Berechnet den Offsetvektor einer Matrix 15x15
void calculate_offset_15(struct complex input[15][15]);

//----------fourier_transform.c---------------//
// Berechnung der Twiddle Matrix für 8x8
void twiddle(void);
// Berechnung der inversen Twiddle Matrix für 8x8
void InvTwiddle(void);
```

```
    // Berechnung der Twiddle Matrix für 15x15
64  void twiddle15(void);
    // Berechnung der inversen Twiddle Matrix für 15x15
66  void InvTwiddle15(void);
    // Berechnung der 2D-FFT für 8x8
68  void fft(void);
    // Berechnung der 2D-IFFT für 8x8
70  void ifft(void);
    // Berechnung der 2D-FFT für 15x15
72  void fft15(void);
    // Berechnung der 2D-IFFT für 15x15
74  void ifft15(void);

76  //----------interpolation.c---------------//
    // Interpoliert eine Matrix von 8x8 auf 15x15
78  void interpolieren(struct complex input[8][8]);

80  //----------digital_filter.c---------------//
    // Verarbeitet die Filterung und wählt passenden 8x8 Filter
82  void filter8(char filtertyp, char anpassen, struct complex input[8][8]);
    // Multiplizerit skalar zwei 8x8 Matrizen zur Filterung
84  void multiply8(struct complex input[8][8], double filter[8][8]);
    // Verarbeitet die Filterung und wählt passenden 15x15 Filter
86  void filter15(char filtertyp, char anpassen, struct complex input[15][15]);
    // Multiplizerit skalar zwei 15x15 Matrizen zur Filterung
88  void multiply15(struct complex input[15][15], double filter[15][15]);

90
    #endif /* INIT_PROTOTYP_H_ */
```

B Quellcode auf dem Raspberry Pi

Quellcode B.1: Backend zum Kalibrierungsfenster und den zugehörigen Threads.

```
    '''
2       Diese Klasse ist das erste Fenster, dass der
        Benutzer zu sehen bekommt. Es sorgt fürm die
4       Einstellung des Offsetausgleichs. Es kann eine
        kurze und eine lange kalibrierung durchgeführt
6       werden.
    '''
8   class CalibrationWindow(QtWidgets.QDialog):

10      # Initialisierung
        def __init__(self):
12          QtWidgets.QDialog.__init__(self)
            self.ui = uic.loadUi("kalibrierung.ui", self)
14          self.setWindowTitle("Kalibrierung")
            self.setFixedSize(230, 160)
16          self.move(500, 500)
            self.progressCali.setMaximum(360)
18          self.progressCali.setValue(0)
            self.count = 0
20
            self.kurzkali = False
22
            self.offsetReal = []
24          self.offsetImag = []

26          # Initialisierung von den Threads
            self.thread1 = progressBarThread(1)
28          self.thread2 = longCaliThread(2)

30          # Verbindung der Threadsignale
            self.thread1.update.connect(self.updateProgressBar)
32          self.thread1.Connected.connect(self.showDialog1)
            self.thread1.finished.connect(self.fin1)
34
            self.thread2.update.connect(self.updateProgressBar2)
36          self.thread2.Connected.connect(self.showDialog2)
            self.thread2.updateSec.connect(self.updateMessung)
38          self.thread2.messung.connect(self.messung)
            self.thread2.finished.connect(self.fin2)
40
            # Anpassung der Button Namen
42          self.startLastCali.setText("Langzeitkalibrierung")
            self.startNewCali.setText("schnelle␣Kalibrierung")
44          self.withoutCali.setText("Langzeitkalib.␣verwenden")

46          # Signale einrichten zu den Buttons
            self.ui.startNewCali.clicked.connect(self.startfastCalibration)
48          self.ui.withoutCali.clicked.connect(self.noCalibration)
            self.ui.startLastCali.clicked.connect(self.startLongCali)
50
        # Diese Funktion wird beim klicken "Zum Hauptfenster" aufgerufen
52      def noCalibration(self):
```

```
            if self.thread1.isRunning() == True:
54              print("Es wird noch kalibiert!")
            elif self.thread2.isRunning() == True:
56              print("Es wird noch kalibiert!")
            else:
58              if(self.kurzkali == False):
                    # Real und Imag Datei öffnen und einlesen
60                  fCos = open("OffsetCosinus.txt", "r")
                    Cosdata = fCos.readlines()
62                  fCos.close()

64                  for item in Cosdata:
                        self.offsetReal.append(float(item.rstrip()))
66
                    CosinusWerte = "6000" + ';'.join(map(str,self.offsetReal)) + "r"
68
                    fSin = open("OffsetSinus.txt", "r")
70                  Sindata = fSin.readlines()
                    fSin.close()
72
                    for item in Sindata:
74                      self.offsetImag.append(float(item.rstrip()))

76                  SinusWerte = "6000" + ';'.join(map(str,self.offsetImag)) + "r"

78                  print(SinusWerte, CosinusWerte)

80                  # serialle Verbindung einrichten
                    try:
82                      self.ser = serial.Serial('/dev/ttyACM0', baudrate = 115200,
                            timeout = 0.3)
                    except Exception as e:
84                      print(e)

86                  try:
                        self.ser = serial.Serial('/dev/ttyACM1', baudrate = 115200,
                            timeout = 0.3)
88                  except Exception as e:
                        print("Kalibrierungsfenster:",e)
90
                    os.system('stty -icanon min 1 eof ^A -F /dev/ttyACM0')
92
                    # Offsetwerte an den MC senden
94
                    self.ser.flushOutput()
96                  self.ser.flushInput()
                    time.sleep(0.2)
98                  self.ser.write(CosinusWerte.encode("ascii"))
                    time.sleep(0.2)
100
                    self.ser.flushOutput()
102                 self.ser.flushInput()
                    time.sleep(0.2)
104                 self.ser.write(SinusWerte.encode("ascii"))
                    time.sleep(0.2)
106
                # weiter zum Hauptfensters
108                 self.ser.close()
                    self.close()
110                 mainWindow = MainWindow()
                    mainWindow.show()
112             else:
                    self.close()
114                 mainWindow = MainWindow()
                    mainWindow.show()
```

```
116
        # Start der Langzeitkalibrierung
118     def startLongCali(self):
            if (self.thread2.isRunning() == False):
120             self.withoutCali.setText("Bitte warten...")
                self.progressCali.setValue(0)
122             self.kurzkali = False
                self.thread2.start()
124         else:
                print("Kalibierung wird bereits durchgeführt!")
126
        # Thread 1 ist fertig
128     def fin1(self):
            self.thread1.quit()
130
        # Thread 2 ist fertig
132     def fin2(self):
            self.thread2.quit()
134         self.msg2.setText("Messung abgeschlossen!")

136     # Schnelle Kalibrierung durchführen
        def startfastCalibration(self):
138
            if (self.thread1.isRunning() == False):
140             self.withoutCali.setText("Bitte warten...")
                self.progressCali.setValue(0)
142             self.kurzkali = True
                self.thread1.start()
144         else:
                print("Kalibierung wird bereits durchgeführt!")
146
        # Werte auf der ProgressBar aktualisieren(Thread 1)
148     def updateProgressBar(self, updateNum):

150         if (int(updateNum) < 359):
                self.progressCali.setValue(self.progressCali.value() + 1)
152         elif (int(updateNum) == 360):
                self.progressCali.setValue(self.progressCali.value() + 1)
154             print("kalibrierung abgeschlossen!")
                self.withoutCali.setText("Weiter zum Hauptmenü")
156
        # Werte auf der ProgressBar aktualisieren(Thread 2)
158     def updateProgressBar2(self):
            self.count  = self.count + 1
160         self.progressCali.setValue(self.progressCali.value() + 15)
            if (self.count == 24):
162             #self.progressCali.setValue(self.progressCali.value() + 4)
                print("kalibrierung abgeschlossen!")
164             self.withoutCali.setText("Weiter zum Hauptmenü")

166     # Dialogfenster Thread 1
        def showDialog1(self, info):
168         self.serialInfo = info
            self.msg = QMessageBox()
170         self.msg.setIcon(QMessageBox.Warning)
            self.msg.setWindowTitle("Serial-Connection Info")
172         self.msg.setText(self.serialInfo)

174         self.msg.setStandardButtons(QMessageBox.Ok)
            self.msg.buttonClicked.connect(self.msg.close)
176         self.msg.exec_()

178     # Dialogfenster Thread 2
        def showDialog2(self, info):
180         self.serialInfo = info
```

```
            self.msg2 = QMessageBox()
182         self.msg2.setIcon(QMessageBox.Warning)
            self.msg2.setWindowTitle("Langzeitmessung")
184
            self.msg2.setStandardButtons(QMessageBox.Ok)
186         self.msg2.buttonClicked.connect(self.msg2.close)
            time.sleep(0.4) # Wartezeit im Thread
188         self.msg2.exec_()

190     # Dialogfenster 2 Text anpassen mit Count-Down
        def updateMessung(self, sec):
192         self.msg2.setText("Zeit zur nächsten Messung: "+str(sec)+" Sekunde(n)")

194     # Dialogfenster 2 text anpassen mit Messung
        def messung(self):
196         self.msg2.setText("Zeit zur nächsten Messung:                Jetzt!")

198
    '''
200     Diese Klasse sorgt für das Durchführen einer
        schnellen Kalibrierung. Die Kommunikation mit
202     dem MC findet seriell via UART statt.
    '''
204 class progressBarThread(QThread):

206     # Signale zum Kalibrierungsfenster
        update = pyqtSignal(int)
208     Connected = pyqtSignal(str)

210     # Initialisierung der Klasse
        def __init__(self, threadID):
212         QThread.__init__(self)
            self.threadID = threadID
214         self.connected = False

216     # Wartefunktion des Threads
        def __del__(self):
218         self.wait()

220     # Run-Methode wird nach start() des Threads ausgeführt
        def run(self):
222
            # Verbindung mit dem MC herstellen
224         try:
                self.ser = serial.Serial('/dev/ttyACM0', baudrate = 115200, timeout =
                    1)
226         except Exception as e:
                print(e)
228
            try:
230             self.ser = serial.Serial('/dev/ttyACM1', baudrate = 115200, timeout =
                    1)
            except Exception as e:
232             print("ProgressBarThread:",e)

234         try:
                self.serialInfo = "Verbindung hergestelt mit: " + str(self.ser.name)
236             self.connected = True
            except Exception as e:
238             self.serialInfo = "Es konnte keine Verbindung hergestellt werden."
                self.connected = False
240
            # Signal an das Kalib.-Fenster, dass die Verbindung zum MC steht
242         self.Connected.emit(self.serialInfo)
```

```
244         if (self.connected == True):

246                 self.ser.flushOutput()
                    self.ser.flushInput()
248                 self.sleep(0.1)
                    # initialisierung auf dem Tiva Board
250                 self.ser.write("0000r".encode("ascii"))
                    self.sleep(0.1)
252                 self.ser.flushOutput()
                    self.ser.flushInput()
254                 self.sleep(0.1)
                    # initialisierung auf dem Tiva Board
256                 self.ser.write("0000r".encode("ascii"))
                    self.sleep(0.1)
258
                    self.ser.flushOutput()
260                 self.ser.flushInput()
                    self.sleep(0.1)
262                 # Offset auf dem Tiva Board berechnen
                    self.ser.write("1000r".encode("ascii"))
264                 self.sleep(0.1)
                    self.ser.flushOutput()
266                 self.ser.flushInput()
                    self.sleep(0.1)
268                 # Offset auf dem Tiva Board berechnen
                    self.ser.write("1000r".encode("ascii"))
270                 self.sleep(0.1)

272                 # Simulation des Ladebalkens
                    for i in range(361):
274                     self.update.emit(i)
                        self.sleep(0.5)
276
            # schließt zum Abschluss die serielle Verbindung
278         self.ser.close()

280 '''
        Diese Klasse ist für die Durchführung einer Lang-
282     zeitkalibrierung zuständig. Es wird alle drei
        Sekunden eine Messung bei 15 Grad durchgeführt.
284 '''
    class longCaliThread(QThread):
286
        # Signale zum Kalibrierungsfenster
288     update = pyqtSignal()
        Connected = pyqtSignal(str)
290     updateSec = pyqtSignal(int)
        messung = pyqtSignal()
292
        # Initialisierung der Klasse
294     def __init__(self, threadID):
            QThread.__init__(self)
296         self.threadID = threadID
            self.connected = False
298         self.OffsetBufferReal = np.zeros(shape=(8,8))
            self.OffsetBufferImag = np.zeros(shape=(8,8))
300         self.finish = ""
            self.updateInt = 0
302         self.cosBuffer = []
            self.sinBuffer = []
304
        # Wartefunktion des Threads
306     def __del__(self):
            self.wait()
308
```

```
        # Run-Methode wird nach start() des Threads ausgeführt
310     def run(self):

312         # Verbindung mit dem MC herstellen
            try:
314             self.ser = serial.Serial('/dev/ttyACM0', baudrate = 115200, timeout =
                    1)
            except Exception as e:
316             print(e)

318         try:
                self.ser = serial.Serial('/dev/ttyACM1', baudrate = 115200, timeout =
                    1)
320         except Exception as e:
                print(e)
322
            try:
324             self.serialInfo = "Verbindung hergestelt mit: " + str(self.ser.name)
                self.connected = True
326         except Exception as e:
                self.serialInfo = "Es konnte keine Verbindung hergestellt werden."
328             self.connected = False

330         os.system('stty -icanon min 1 eof ^A -F /dev/ttyACM0')

332         # Signal an das Kalib.-Fenster, dass die Verbindung zum MC steht
            self.Connected.emit(self.serialInfo)
334         print("Langzeitkalibrierung gestartet\n")

336         DATNEU_FLOAT = []

338         if (self.connected == True):

340             # Initialisierung auf dem Tiva Board

342             self.ser.write("0000r".encode("ascii"))
                self.ser.flushInput()
344             time.sleep(0.02)

346             for cycle in range(1,25,1):

348                 for i in range(3):
                        self.updateSec.emit(3-i)
350                     time.sleep(1)

352                 # Senden des Befehls zur Wertbeschaffung

354                 self.ser.write("5000r".encode("ascii"))
                    time.sleep(0.02)
356                 DAT = self.ser.read(4000)
                    DATNEU_FLOAT.clear()
358                 DATNEU = DAT.decode("ascii").replace("\x00", ";")
                    for value in DATNEU.split(';'):
360                     if(value is not ''):
                            DATNEU_FLOAT.append(float(value))
362
                    # Falls Buffer zu klein, Senden wiederholen
364                 if (len(DATNEU_FLOAT) != 128):
                        self.ser.flushInput()
366                     time.sleep(0.02)
                        self.ser.write("5000r".encode("ascii"))
368                     time.sleep(0.02)
                        DAT = self.ser.read(4000)
370                     DATNEU_FLOAT.clear()
                        DATNEU = DAT.decode("ascii").replace("\x00", ";")
```

```
372                     for value in DATNEU.split(';'):
                            if(value is not ''):
374                             DATNEU_FLOAT.append(float(value))

376                 # Zuordnen der Werte aus dem Buffer zu Sinus und Cosinus
                    for i in range(0, 64, 1):
378                     self.cosBuffer.append(DATNEU_FLOAT[i])
                    for i in range(64, 128, 1):
380                     self.sinBuffer.append(DATNEU_FLOAT[i])

382                 DATNEU_FLOAT.clear()

384                 # Speichern des Buffers in den Matrizen
                    k = 0
386                 for i in range(8):
                        for j in range(8):
388                         self.OffsetBufferReal[i][j] = self.OffsetBufferReal[i][j] +
                                self.cosBuffer[k]
                            self.OffsetBufferImag[i][j] = self.OffsetBufferImag[i][j] +
                                self.sinBuffer[k]
390                         k = k + 1

392                 # Signal an das Fenster zur ProgressBaraktualisierung
                    self.update.emit()
394
                # Mittelwert über die Messung bilden
396             for i in range(8):
                    for j in range(8):
398                     self.OffsetBufferReal[i][j] = self.OffsetBufferReal[i][j] / 24
                        self.OffsetBufferImag[i][j] = self.OffsetBufferImag[i][j] / 24
400
                # Senden der Messwerte Cosinuswerte
402             self.ser.flushOutput()
                self.ser.flushInput()
404             self.sleep(0.1)
                self.finish = "6000" + str(self.OffsetBufferReal) + "r"
406             self.ser.write(self.finish.encode("ascii"))
                self.sleep(0.1)
408
                # Senden der Messwerte Sinuswerte
410             self.ser.flushOutput()
                self.ser.flushInput()
412             self.sleep(0.1)
                self.finish = "6000" + str(self.OffsetBufferImag) + "r"
414             self.ser.write(self.finish.encode("ascii"))
                self.sleep(0.1)
416             #print("Cosinus:", self.OffsetBufferReal)

418             # Speichern der Messwerte im .txt File
                f = open("OffsetCosinus.txt", "w+")
420             k = 0
                for i in range(8):
422                 for j in range(8):
                        f.write("%f\r" % self.OffsetBufferReal[i][j])
424             f.close()

426             f = open("OffsetSinus.txt", "w+")
                for i in range(8):
428                 for j in range(8):
                        f.write("%f\r" % self.OffsetBufferImag[i][j])
430             f.close()

432         self.ser.close()
```

Quellcode B.2: Backend zum Hauptfensterfenster mit dem Send-Recieve-Thread.

```
    '''
2       Klasse des Hauptfensters. Diese wird aufgerufen nach der Kalibrierung.
        Aus der Klasse werden alle weiteren Einstellungen gesteuert. Mit der
4       Initialisierung der Klasse wird auch der Daten-Thread gestartet, der
        zur Kommunikation mit dem Mikrocontroller dient.
6   '''
    class MainWindow(QtWidgets.QMainWindow):
8
        startWindow1 = pyqtSignal()
10
        '''
12          Initialisierung des Hauptfensters. Dazu gehören auch die Ini-
            tialisierungen der einzelnen Dialog-Fenster (diagramme, filtersetting)
14      '''
        def __init__(self):
16
            QtWidgets.QMainWindow.__init__(self)
18          self.ui = uic.loadUi("MainWindow.ui", self)
            self.move(500, 500)
20          self.setFixedSize(280, 210)
            self.filterType = 0
22          self.customType = False
            self.diagramm = 0
24          self.interpol = False
            self.filterValues = []
26
            # Initialisierung der Diagrammfenster
28          self.diagram1 = Window1()
            self.diagram2 = Window2()
30          self.diagram3 = Window3()
            self.diagram4 = Window4()
32
            # Slots zu den Buttons einrichten
34          self.ui.setCoeff.clicked.connect(self.openCoeff)
            self.ui.setSettings.clicked.connect(self.changeSettings)
36          self.ui.callHelp.clicked.connect(self.callHelpPDF)

38          # Filtersetting 8x8 initialisieren und Signal verbinden
            self.coeff8Window = CoeffWindow8()
40          self.coeff8Window.sendData8.connect(self.saveFilterData)

42          # Filtersetting 15x15 initialisieren und Signal verbinden
            self.coeff15Window = coeffWindow15()
44          self.coeff15Window.sendData15.connect(self.saveFilterData)

46          # Data-Send-Recieve-Thread initialisieren und starten
            self.dataThread = SendRecieveThread()
48          #self.dataThread.notConnected.connect(self.showDialog)
            self.dataThread.start()
50
        '''
52          Wenn die Einstellungen vorgenommen worden sind und der Button
            "Diagramm erstellen" betätigt wird, wird diese Funktoin zur
54          Speicherung der Werte und zum öffnen und verwalten der Diagramm-
            fenster aufgerufen.
56      '''
        def changeSettings(self):
58
            # Globale Variablen, die in dieser Funktion benötigt werden
60          global w1_diagramm
            global w2_diagramm
62          global w3_diagramm
```

```
        global w4_diagramm
64
        global w1_interpolation
66      global w2_interpolation
        global w3_interpolation
68      global w4_interpolation

70      global w1_filtertype
        global w2_filtertype
72      global w3_filtertype
        global w4_filtertype
74
        global w1_befehl
76      global w2_befehl
        global w3_befehl
78      global w4_befehl

80      global w1_anpassen
        global w2_anpassen
82      global w3_anpassen
        global w4_anpassen
84
        global w1_filter
86      global w2_filter
        global w3_filter
88      global w4_filter

90      global w1_diaName
        global w2_diaName
92      global w3_diaName
        global w4_diaName
94

96      ##############################
        #    Fenster 1 einrichten    #
98      ##############################
        if (self.diagram1.isVisible() == False):
100
            # Welches Diagramm wurde gewählt?
102         if self.ui.chooseDiag.currentText() == "Histogramm":
                w1_diagramm = 2     # Befehl für Interrupt_Handler
104         else:
                w1_diagramm = 3     #Sonst werden immer Sinus und Cosinus Werte ben
                    ötigt
106         # Speichern des Filtertyps
            w1_filtertype = self.filterType
108         # Speichern, ob die koeffizienten angepasst wurde oder nicht
            if (self.customType == True):
110             w1_anpassen = 1
            else:
112             w1_anpassen = 0
            # Speichern der Filterwerte
114         w1_filter = self.filterValues
            w1_diaName = self.ui.chooseDiag.currentText()
116         # Speichern, ob interpoliert werden soll oder nicht
            if (self.setInterpol.isChecked()):
118             w1_interpolation = 1
            else:
120             w1_interpolation = 0
            # Informationen als String in "w1_befehl" schreiben
122         w1_befehl = str(w1_diagramm) + str(w1_interpolation) + str(
                w1_filtertype) + str(w1_anpassen) + ';'.join(map(str,w1_filter)) +
                "r"
            #print(w1_befehl)
124         self.diagram1.show()
```

```
126
            ##############################
128         #    Fenster 2 einrichten    #
            ##############################
130         elif (self.diagram2.isVisible() == False):

132             # Welches Diagramm wurde gewählt?
                if self.ui.chooseDiag.currentText() == "Histogramm":
134                 w2_diagramm = 2     # Befehl für Interrupt_Handler
                else:
136                 w2_diagramm = 3     #Sonst werden immer Sinus und Cosinus Werte ben
                        ötigt
                # Speichern des Filtertyps
138             w2_filtertype = self.filterType
                # Speichern, ob die koeffizienten angepasst wurde oder nicht
140             if (self.customType == True):
                    w2_anpassen = 1
142             else:
                    w2_anpassen = 0
144             # Speichern der Filterwerte
                w2_filter = self.filterValues
146             w2_diaName = self.ui.chooseDiag.currentText()
                # Speichern, ob interpoliert werden soll oder nicht
148             if (self.setInterpol.isChecked()):
                    w2_interpolation = 1
150             else:
                    w2_interpolation = 0
152             # Informationen als String in "w2_befehl" schreiben
                w2_befehl = str(w2_diagramm) + str(w2_interpolation) + str(
                    w2_filtertype) + str(w2_anpassen) + ';'.join(map(str,w2_filter)) +
                    "r"
154
                self.diagram2.show()
156

158         ##############################
            #    Fenster 3 einrichten    #
160         ##############################
            elif (self.diagram3.isVisible() == False):
162
                # Welches Diagramm wurde gewählt?
164             if self.ui.chooseDiag.currentText() == "Histogramm":
                    w3_diagramm = 2     # Befehl für Interrupt_Handler
166             else:
                    w3_diagramm = 3     #Sonst werden immer Sinus und Cosinus Werte ben
                        ötigt
168             # Speichern des Filtertyps
                w3_filtertype = self.filterType
170             # Speichern, ob die koeffizienten angepasst wurde oder nicht
                if (self.customType == True):
172                 w3_anpassen = 1
                else:
174                 w3_anpassen = 0
                # Speichern der Filterwerte
176             w3_filter = self.filterValues
                w3_diaName = self.ui.chooseDiag.currentText()
178             # Speichern, ob interpoliert werden soll oder nicht
                if (self.setInterpol.isChecked()):
180                 w3_interpolation = 1
                else:
182                 w3_interpolation = 0
                # Informationen als String in "w3_befehl" schreiben
184             w3_befehl = str(w3_diagramm) + str(w3_interpolation) + str(
                    w3_filtertype) + str(w3_anpassen) + ';'.join(map(str,w3_filter)) +
```

```
                "r"

186             self.diagram3.show()

188
            ##############################
190         #    Fenster 4 einrichten    #
            ##############################
192         elif (self.diagram4.isVisible() == False):

194             # Welches Diagramm wurde gewählt?
                if self.ui.chooseDiag.currentText() == "Histogramm":
196                 w4_diagramm = 2     # Befehl für Interrupt_Handler
                else:
198                 w4_diagramm = 3     #Sonst werden immer Sinus und Cosinus Werte ben
                        ötigt
                # Speichern des Filtertyps
200             w4_filtertype = self.filterType
                # Speichern, ob die koeffizienten angepasst wurde oder nicht
202             if (self.customType == True):
                    w4_anpassen = 1
204             else:
                    w4_anpassen = 0
206             # Speichern der Filterwerte
                w4_filter = self.filterValues
208             w4_diaName = self.ui.chooseDiag.currentText()
                # Speichern, ob interpoliert werden soll oder nicht
210             if (self.setInterpol.isChecked()):
                    w4_interpolation = 1
212             else:
                    w4_interpolation = 0
214             # Informationen als String in "w4_befehl" schreiben
                w4_befehl = str(w4_diagramm) + str(w4_interpolation) + str(
                    w4_filtertype) + str(w4_anpassen) + ';'.join(map(str,w4_filter)) +
                    "r"
216
                self.diagram4.show()
218
            ##############################
220         #    alle Fenster offen    #
            ##############################
222         else:
                self.showDialog()
224
        '''
226         Dialogfenster, welches angezeigt wird,
            wenn bereits 4/4 Fenster Geöffnet sind.
228     '''
        def showDialog(self):
230         self.msg = QMessageBox()
            self.msg.setIcon(QMessageBox.Warning)
232         self.msg.setWindowTitle("Window␣Counter␣Warning")
            self.msg.setText("Es␣sind␣bereits␣4␣Fenster␣geöffnet.")
234
            self.msg.setStandardButtons(QMessageBox.Ok)
236         self.msg.buttonClicked.connect(self.msg.close)
            self.msg.show()
238
        '''
240         Funktion, die durch das Signal von den Koeffizienten-
            fenster aufgerufen wird. Die empfangenen Daten werden in lokale
242         Variablen gespeichert.
        '''
244     def saveFilterData(self, filterType, customType, filterValues):
            self.filterType = filterType
```

```
246             self.customType = customType
                self.filterValues = filterValues
248
        '''
250             Funktion, die aufgeruen wird, wenn ein Filtereinstellungs-
                fenster geöffnet werden soll. Es wird Anhand Interpolation
252             (True, False) entschieden.
        '''
254         def openCoeff(self):
                if self.setInterpol.isChecked():
256                 # Interpolation ist geckecked
                    self.coeff15Window.exec_()
258             else:
                    # es soll nicht interpoliert werden
260                 self.coeff8Window.exec_()
        '''
262             Öfnet eine PDF. Diese soll Anleitungen zur Programmnutzung
                enthalten.
264         '''
            def callHelpPDF(self):
266             self.cmd = 'qpdfview anleitung.pdf'
                help = os.system(self.cmd)
268 '''
        Die Klasse SendRecieveThread() ist ein Thread-Objekt,
270     welches koninuierlich die Fensterbefehle an den MC
        sendet und die Werte empfängt, verarbeitet und in
272     den Fenstervariablen speichert.
    '''
274 class SendRecieveThread(QThread):

276     # Initialisierung
        def __init__(self):
278         QThread.__init__(self)
            # Lokale Buffer
280         self.connected = False
            self.IOBuffer1 = []
282         self.IOBuffer2 = []
            self.IOBuffer3 = []
284         self.IOBuffer4 = []
            self.errorcount = 0
286         self.count = 0

288         self.sinBuffer = []
            self.cosBuffer = []
290

292     # Wartefunktion
        def __del__(self):
294         self.wait()

296     # Wird durch Thread.start() aufgerufen
        def run(self):
298
            # globale Variabeln
300         global w1_befehl
            global w2_befehl
302         global w3_befehl
            global w4_befehl
304
            global w1_sinus
306         global w1_cosinus
            global w2_sinus
308         global w2_cosinus
            global w3_sinus
310         global w3_cosinus
```

```
            global w4_sinus
312         global w4_cosinus

314         global w1_angles
            global w2_angles
316         global w3_angles
            global w4_angles
318
            global w1_filter
320         global w2_filter
            global w3_filter
322         global w4_filter

324         global w1_diagramm
            global w2_diagramm
326         global w3_diagramm
            global w4_diagramm
328
            global w1_interpolation
330         global w2_interpolation
            global w3_interpolation
332         global w4_interpolation

334         global angle

336         global w1_kreisDataCount
            global w2_kreisDataCount
338         global w3_kreisDataCount
            global w4_kreisDataCount
340
           #help = os.system('stty -icanon -F /dev/ttyACM0')
342
           # Serielle Verbindung herstellen
344        try:
               self.ser = serial.Serial('/dev/ttyACM0', baudrate = 115200, timeout =
                   0.4)
346        except Exception as e:
               print(e)
348
           try:
350            self.ser = serial.Serial('/dev/ttyACM1', baudrate = 115200, timeout =
                   0.4)
           except Exception as e:
352            print("SendeEmpfangsThread:", e)

354        try:
               self.serialInfo = "Verbindung hergestelt mit: " + str(self.ser.name)
356            self.connected = True
           except Exception as e:
358            self.serialInfo = "Es konnte keine Verbindung hergestellt werden."
               print(self.serialInfo)
360
           # Konsolenausgabe
362        print("Starting Data-Thread")

364        os.system('stty -icanon min 1 eof ^A -F /dev/ttyACM0')

366        DATNEU_FLOAT = []
           while(True):
368

370            ########################
               #   Senden befehl 1    #
372            ########################
```

```
374             # Damit keine Änderung nach dem Einlesen kommen kann

376
                self.ser.flushInput()
378             time.sleep(0.02)
                self.ser.write(w1_befehl.encode("ascii"))
380             time.sleep(1)
                DAT = self.ser.read(4000)
382             #print(len(DAT))
                DATNEU_FLOAT.clear()
384             DATNEU = DAT.decode("ascii").replace("\x00", ";")
                for value in DATNEU.split(';'):
386                 if(value is not ''):
                        DATNEU_FLOAT.append(float(value))
388             staticInterpol = w1_interpolation
                # Wiederholtes Senden, falls nichts/zu wenig eingelesen wurde
390             if (len(DATNEU_FLOAT) != (128 + (staticInterpol * 322))):
                    self.ser.flushInput()
392                 time.sleep(0.02)
                    self.ser.write(w1_befehl.encode("ascii"))
394                 time.sleep(1)
                    DAT = self.ser.read(4000)
396                 DATNEU_FLOAT.clear()
                    DATNEU = DAT.decode("ascii").replace("\x00", ";")
398                 for value in DATNEU.split(';'):
                        if(value is not ''):
400                         DATNEU_FLOAT.append(float(value))

402             #rint(len(DATNEU_FLOAT))
                # Speichern der Antowrt für ein Histogramm
404             if(w1_diagramm == 2):
                    for index in range(0, 64 + (staticInterpol * 161), 1):
406                     if (DATNEU_FLOAT[index] > 360.0):
                            self.IOBuffer1.append(int(DATNEU_FLOAT[index] - 360.0))
408                     else:
                            self.IOBuffer1.append(int(DATNEU_FLOAT[index]))
410
                # Speichern der Sinus- und Cosinuswerte
412             if(w1_diagramm == 3):
                    #if (w1_diaName == "Kreisplot"):
414                 #   w1_interpolation = 0
                    #   staticInterpol = 0
416                 for i in range(0, 64 + (staticInterpol * 161), 1):
                        self.cosBuffer.append(DATNEU_FLOAT[i])
418                 for i in range(64 + (staticInterpol * 161), 128 + (staticInterpol *
                         322), 1):
                        self.sinBuffer.append(DATNEU_FLOAT[i])
420
                # Werte übertragen und Buffer leeren
422             w1_angles = self.IOBuffer1[:]
                self.IOBuffer1.clear()
424             w1_cosinus = self.cosBuffer[:]
                self.cosBuffer.clear()
426             w1_sinus = self.sinBuffer[:]
                self.sinBuffer.clear()
428
                ########################
430             #   Senden befehl 2    #
                ########################
432
                # Damit keine Änderung nach dem Einlesen kommen kann
434             staticInterpol = w2_interpolation

436             self.ser.write(w2_befehl.encode("ascii"))
                time.sleep(0.04)
```

```
438             DAT = self.ser.read(4000)
                DATNEU_FLOAT.clear()
440             DATNEU = DAT.decode("ascii").replace("\x00", ";")
                for value in DATNEU.split(';'):
442                 if(value is not ''):
                        DATNEU_FLOAT.append(float(value))
444
                # Wiederholtes Senden, falls nichts/zu wenig eingelesen wurde
446             if (len(DATNEU_FLOAT) != (128 + (staticInterpol * 322))):
                    self.ser.flushInput()
448                 time.sleep(0.02)
                    self.ser.write(w2_befehl.encode("ascii"))
450                 time.sleep(0.02)
                    DAT = self.ser.read(4000)
452                 DATNEU_FLOAT.clear()
                    DATNEU = DAT.decode("ascii").replace("\x00", ";")
454                 for value in DATNEU.split(';'):
                        if(value is not ''):
456                         DATNEU_FLOAT.append(float(value))

458
                # Speichern der Antwort für ein Histogramm
460             if (w2_diagramm == 2):
                    for index in range(0, 64 + (staticInterpol * 161), 1):
462                     if (DATNEU_FLOAT[index] > 360.0):
                            self.IOBuffer2.append(int(DATNEU_FLOAT[index] - 360.0))
464                     else:
                            self.IOBuffer2.append(int(DATNEU_FLOAT[index]))
466
                # Speichern der Sinus- und Cosinuswerte
468             if(w2_diagramm == 3):
                    #if (w2_diaName == "Kreisplot"):
470                 #   w2_interpolation = 0
                    #   staticInterpol = 0
472                 for i in range(0, 64 + (staticInterpol * 161), 1):
                        self.cosBuffer.append(DATNEU_FLOAT[i])
474                 for i in range(64 + (staticInterpol * 161), 128 + (staticInterpol *
                     322), 1):
                        self.sinBuffer.append(DATNEU_FLOAT[i])
476
                # Werte übertragen und Buffer leeren
478             self.IOBuffer2.clear()
                w2_cosinus = self.cosBuffer[:]
480             self.cosBuffer.clear()
                w2_sinus = self.sinBuffer[:]
482             self.sinBuffer.clear()

484             ########################
                #   Senden befehl 3    #
486             ########################

488             # Damit keine Änderung nach dem Einlesen kommen kann
                staticInterpol = w3_interpolation
490
                self.ser.write(w3_befehl.encode("ascii"))
492             time.sleep(0.04)
                DAT = self.ser.read(4000)
494             DATNEU_FLOAT.clear()
                DATNEU = DAT.decode("ascii").replace("\x00", ";")
496             for value in DATNEU.split(';'):
                    if(value is not ''):
498                     DATNEU_FLOAT.append(float(value))

500             # Wiederholtes Senden, falls nichts/zu wenig eingelesen wurde
                if (len(DATNEU_FLOAT) != (128 + (staticInterpol * 322))):
```

```
502                 self.ser.flushInput()
                    time.sleep(0.02)
504                 self.ser.write(w3_befehl.encode("ascii"))
                    time.sleep(0.02)
506                 DAT = self.ser.read(4000)
                    DATNEU_FLOAT.clear()
508                 DATNEU = DAT.decode("ascii").replace("\x00", ";")
                    for value in DATNEU.split(';'):
510                     if(value is not ''):
                            DATNEU_FLOAT.append(float(value))
512
514             # Speichern der Antowrt für ein Histogramm
                if (w3_diagramm == 2):
516                 for index in range(0, 64 + (staticInterpol * 161), 1):
                        if (DATNEU_FLOAT[index] > 360.0):
518                         self.IOBuffer3.append(int(DATNEU_FLOAT[index] - 360.0))
                        else:
520                         self.IOBuffer3.append(int(DATNEU_FLOAT[index]))

522             # Speichern der Sinus- und Cosinuswerte
                if(w3_diagramm == 3):
524                 #if (w3_diaName == "Kreisplot"):
                    #   w3_interpolation = 0
526                 #   staticInterpol = 0
                    for i in range(0, 64 + (staticInterpol * 161), 1):
528                     self.cosBuffer.append(DATNEU_FLOAT[i])
                    for i in range(64 + (staticInterpol * 161), 128 + (staticInterpol *
                         322), 1):
530                     self.sinBuffer.append(DATNEU_FLOAT[i])

532             # Werte übertragen und Buffer leeren
                w3_angles = self.IOBuffer3[:]
534             self.IOBuffer3.clear()
                w3_cosinus = self.cosBuffer[:]
536             self.cosBuffer.clear()
                w3_sinus = self.sinBuffer[:]
538             self.sinBuffer.clear()

540
                ########################
542             #   Senden befehl 4    #
                ########################
544
                # Damit keine Änderung nach dem Einlesen kommen kann
546             staticInterpol = w4_interpolation

548             self.ser.write(w4_befehl.encode("ascii"))
                time.sleep(0.04)
550             DAT = self.ser.read(4000)
                DATNEU_FLOAT.clear()
552             DATNEU = DAT.decode("ascii").replace("\x00", ";")
                for value in DATNEU.split(';'):
554                 if(value is not ''):
                        DATNEU_FLOAT.append(float(value))
556
                # Wiederholtes Senden, falls nichts/zu wenig eingelesen wurde
558             if (len(DATNEU_FLOAT) != (128 + (staticInterpol * 322))):
                    self.ser.flushInput()
560                 time.sleep(0.02)
                    self.ser.write(w4_befehl.encode("ascii"))
562                 time.sleep(0.02)
                    DAT = self.ser.read(4000)
564                 DATNEU_FLOAT.clear()
                    DATNEU = DAT.decode("ascii").replace("\x00", ";")
```

```
566                 for value in DATNEU.split(';'):
                        if(value is not ''):
568                         DATNEU_FLOAT.append(float(value))

570
                # Speichern der Antowrt für ein Histogramm
572             if (w4_diagramm == 2):
                    for index in range(0, 64 + (staticInterpol * 161), 1):
574                     if (DATNEU_FLOAT[index] > 360.0):
                            self.IOBuffer4.append(int(DATNEU_FLOAT[index] - 360.0))
576                     else:
                            self.IOBuffer4.append(int(DATNEU_FLOAT[index]))
578
                # Speichern der Sinus- und Cosinuswerte
580             if(w4_diagramm == 3):
                    #if (w4_diaName == "Kreisplot"):
582                 #   w4_interpolation = 0
                    #   staticInterpol = 0
584                 for i in range(0, 64 + (staticInterpol * 161), 1):
                        self.cosBuffer.append(DATNEU_FLOAT[i])
586                 for i in range(64 + (staticInterpol * 161), 128 + (staticInterpol *
                     322), 1):
                        self.sinBuffer.append(DATNEU_FLOAT[i])
588
                # Werte übertragen und Buffer leeren
590             w4_angles = self.IOBuffer4[:]
                self.IOBuffer4.clear()
592             w4_cosinus = self.cosBuffer[:]
                self.cosBuffer.clear()
594             w4_sinus = self.sinBuffer[:]
                self.sinBuffer.clear()
```

Quellcode B.3: Backend zu den Filtereinstellungsfenstern.

```
    '''
2       Diese Klasse wird von dem Main-Window aufgerufen und ermöglicht
        die Einstellungen der Filter für eine 8x8 Matrix vorzunehmen.
4       Beim Speichern der Werte werden diese an das Main-Window gesendet.
    '''
6   class CoeffWindow8(QtWidgets.QDialog):

8       # Signal zum Main-Window zum übermitteln der Filtereinstellungen
        sendData8 = pyqtSignal(int, bool, list)
10
        #Initialisierung des Filterfensters
12      def __init__(self):
            QtWidgets.QDialog.__init__(self)
14          self.ui = uic.loadUi("Filtersettings_8.ui", self)
            self.setFixedSize(530, 245)
16          self.groupBox.setEnabled(False)
            self.filterValues8 = []
18          # Vorberechnete Filterwerte
            self.tiefpass = [0.004,0.004,0.004,0.004,0.043,0.043,0.043,
20                  0.004,0.043,0.140,0.140,0.004,0.043,0.14,0.235]
            self.bandpass = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
22          self.fullFilter = np.zeros(shape=(8,8))

24          # Signale der Buttons
            self.ui.cancelFilter8.clicked.connect(self.close)
26          self.ui.saveFilter8.clicked.connect(self.saveFilter8Settings)
            self.ui.customFilter8.stateChanged.connect(self.costumCoeffsEnable)
28          self.ui.chooseFilter8.currentIndexChanged.connect(self.showKoeffs)
            self.ui.showFilter8.clicked.connect(self.plotKoeffs)
30
        # Speichern der Filtereinstellungen und senden an das Hauptfenster
32      def saveFilter8Settings(self):
            self.custom = self.customFilter8.isChecked()
34          self.filterType = self.chooseFilter8.currentIndex()
            self.filterValues8.clear()
36          # Speichern der angepassten Filterwerte in Schleife von 0-3,
            # da die 8x8 Matrix Achsensymmetrisch ist
38          for i in range(4):
                for j in range(4):
40                  # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                    self.name = "spinBox_" + str(i) + str(j)
42                  label = getattr(self, self.name)
                    # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
44                  # in der Filterkoeff.-Liste
                    self.filterValues8.append(float("%.2f" % label.value()))
46          # Übermitteln der Werte
            self.sendData8.emit(self.filterType, self.custom, self.filterValues8)
48          # Schließen des Fensters
            self.close()
50
        # Passende Werte in den SpinBoxen anzeigen, je mach Filterwahl
52      def showKoeffs(self):
            if self.chooseFilter8.currentText() == "Tiefpass":
54              # Q1 der Tiefpass Koeffs anzeigen
                k = 0
56              for i in range(4):
                    for j in range(4):
58                      # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                        self.name = "spinBox_" + str(i) + str(j)
60                      self.label = getattr(self, self.name)
                        # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
62                      # in der Filterkoeff.-Liste
```

```
                    self.ui.label.setValue(self.tiefpass[k])
64                  k = k + 1
        elif self.chooseFilter8.currentText() == "Bandpass":
66          # Q1 der Bandpass Koeffs anzeigen
            k = 0
68          for i in range(4):
                for j in range(4):
70                  # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                    self.name = "spinBox_" + str(i) + str(j)
72                  self.label = getattr(self, self.name)
                    # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
74                  # in der Filterkoeff.-Liste
                    self.ui.label.setValue(self.bandpass[k])
76                  k = k + 1

78      # Plotten der derzeitig eingegebenen Koeffizienten
        def plotKoeffs(self):
80          self.filterValues8.clear()
            for i in range(4):
82              for j in range(4):
                    # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
84                  self.name = "spinBox_" + str(i) + str(j)
                    label = getattr(self, self.name)
86                  # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
                    # in der Filterkoeff.-Liste
88                  self.filterValues8.append(float("%.2f" % label.value()))

90          k = 0
            for i in range(4):
92              for j in range(4):
                    self.fullFilter[i][j] = self.filterValues8[k]
94                  k = k + 1

96          k = 0
            for i in range(4):
98              for j in range(7,3,-1):
                    self.fullFilter[i][j] = self.filterValues8[k]
100                 k = k + 1

102         k = 0
            for i in range(4,8,1):
104             k = k + 1
                for j in range(0,8,1):
106                 self.fullFilter[i][j] = self.fullFilter[i-k][j]
                k = k + 1
108
            plt.xlabel('Zeile')
110         plt.ylabel('Spalte')
            plt.title('8x8 Koeffizienten-Matrix')
112         plt.imshow(self.fullFilter)
            plt.colorbar()
114         plt.show()

116     # Eingabefelder für die Koeffs. freigeben, wenn anpassen betätigt
        def costumCoeffsEnable(self):
118         if self.customFilter8.isChecked():
                self.groupBox.setEnabled(True)
120         else:
                self.groupBox.setEnabled(False)
122

124 '''
        Diese Klasse wird von dem Main-Window aufgerufen und ermöglicht
126     die Einstellungen der Filter für eine 15x15 Matrix vorzunehmen.
        Beim Speichern der Werte werden diese an das Main-Window gesendet.
```

```
'''
class coeffWindow15(QtWidgets.QDialog):

    # Signal zum Main-Window zum übermitteln der Filtereinstellungen
    sendData15 = pyqtSignal(int, bool, list)

    #Initialisierung des Filterfensters
    def __init__(self):
        QtWidgets.QDialog.__init__(self)
        self.ui = uic.loadUi("Filtersettings_15.ui", self)
        self.setFixedSize(650, 410)
        self.groupBox.setEnabled(False)
        self.filterValues15 = []
        # Speichern der berechneten Filterwerte
        self.tiefpass = [-0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0089048,
             -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550,
             -0.0139550, -0.0139550, -0.0139550, -0.0028955, -0.0089048,
            -0.0139550, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
             -0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, -0.0028955, -0.0089048, -0.0139550,
             0.00000000, 0.05117900, 0.13493000, 0.13493000, 0.13493000,
            -0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
             0.21613000, 0.21613000, -0.0028955, -0.0089048, -0.0139550,
            0.00000000, 0.05117900, 0.13493000, 0.21613000, 0.25000000]
        self.bandpass = [-0.0028955, -0.0028955, -0.0028955, -0.0028955,
            -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0028955, -0.0089048,
             -0.0089048, -0.0089048, -0.0089048, -0.0089048, -0.0089048,
            -0.0089048, -0.0028955, -0.0089048, -0.0139550, -0.0139550, -0.0139550,
             -0.0139550, -0.0139550, -0.0139550, -0.0028955, -0.0089048,
            -0.0139550, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
             -0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900,
            0.05117900, 0.05117900, 0.05117900, -0.0028955, -0.0089048, -0.0139550,
             0.00000000, 0.05117900, 0.13493000, 0.13493000, 0.13493000,
            -0.0028955, -0.0089048, -0.0139550, 0.00000000, 0.05117900, 0.13493000,
             0.21613000, 0.21613000, -0.0028955, -0.0089048, -0.0139550,
            0.00000000, 0.05117900, 0.13493000, 0.21613000, 0.00000000]

        self.fullFilter = np.zeros(shape=(15,15))

        # Signale der Buttons
        self.ui.cancelFilter15.clicked.connect(self.close)
        self.ui.saveFilter15.clicked.connect(self.saveFilter15Settings)
        self.ui.customFilter15.stateChanged.connect(self.costumCoeffsEnable)
        self.ui.chooseFilter15.currentIndexChanged.connect(self.showKoeffs)
        self.ui.showFilter15.clicked.connect(self.plotKoeffs)

    # Speichern der Filtereinstellungen und senden an das Hauptfenster
    def saveFilter15Settings(self):
        self.custom = self.customFilter15.isChecked()
        self.filterType = self.chooseFilter15.currentIndex()
        self.filterValues15.clear()
        # Speichern der angepassten Filterwerte
        # Schleife geht von 0-7, da die Matrix nicht ganz Achsen-
        # symmetrisch ist
        for i in range(8):
            for j in range(8):
                # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                self.name = "spinBox_" + str(i) + str(j)
                label = getattr(self, self.name)
                # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
                # in der Filterkoeff.-Liste
                self.filterValues15.append(float("%.3f" % label.value()))
        # Übermitteln der Werte
```

```
            self.sendData15.emit(self.filterType, self.custom, self.filterValues15)
172         # Schließen des Fensters
            self.close()
174
        # Passende Werte in den SpinBoxen anzeigen, je mach Filterwahl
176     def showKoeffs(self):
            if self.chooseFilter15.currentText() == "Tiefpass":
178             # Q1 der Tiefpass Koeffs anzeigen
                k = 0
180             for i in range(8):
                    for j in range(8):
182                     # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                        self.name = "spinBox_" + str(i) + str(j)
184                     self.label = getattr(self, self.name)
                        # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
186                     # in der Filterkoeff.-Liste
                        self.ui.label.setValue(self.tiefpass[k])
188                     k = k + 1
            elif self.chooseFilter15.currentText() == "Bandpass":
190             # Q1 der Bandpass Koeffs anzeigen
                k = 0
192             for i in range(8):
                    for j in range(8):
194                     # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
                        self.name = "spinBox_" + str(i) + str(j)
196                     self.label = getattr(self, self.name)
                        # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
198                     # in der Filterkoeff.-Liste
                        self.ui.label.setValue(self.bandpass[k])
200                     k = k + 1

202     # Plotten der derzeitig eingegebenen Koeffizienten
        def plotKoeffs(self):
204         self.filterValues15.clear()
            for i in range(8):
206             for j in range(8):
                    # erstellen der SpinBox Widget Namen zum Zugriff auf den Wert
208                 self.name = "spinBox_" + str(i) + str(j)
                    label = getattr(self, self.name)
210                 # Speichern des Wertes mit 2 Nachkommastellen (float-Fehler)
                    # in der Filterkoeff.-Liste
212                 self.filterValues15.append(float("%.3f" % label.value()))

214         k = 0
            for i in range(8):
216             for j in range(8):
                    self.fullFilter[i][j] = self.filterValues15[k]
218                 k = k + 1

220         k = 0
            for i in range(8):
222             for j in range(14,6,-1):
                    self.fullFilter[i][j] = self.filterValues15[k]
224                 k = k + 1

226         k = 1
            for i in range(8,15,1):
228             k = k + 1
                for j in range(0,15,1):
230                 self.fullFilter[i][j] = self.fullFilter[i-k][j]
                k = k + 1
232
            plt.xlabel('Zeile')
234         plt.ylabel('Spalte')
            plt.title('15x15 Koeffizienten-Matrix')
```

```
236             plt.imshow(self.fullFilter)
                plt.colorbar()
238             plt.show()

240         # Eingabefelder für die Koeffs. freigeben, wenn anpassen betätigt
            def costumCoeffsEnable(self):
242             if self.customFilter15.isChecked():
                    self.groupBox.setEnabled(True)
244             else:
                    self.groupBox.setEnabled(False)
```

Quellcode B.4: Backend zum Diagrammfenster und den Plot-Funktionen.

```
    '''
2       Die Klasse Window2 stellt das zweite von vier Fenstern
        dar, im dem Diagramme dargestellt werden können.
4   '''
    class Window2(QtWidgets.QDialog):
6
        # Initialisierung des Fensters
8       def __init__(self, parent = None):
            super(Window2, self).__init__(parent)
10          self.ui = uic.loadUi("window2.ui", self)

12          # Klassenvariablen
            self.bottom= None
14          self.top=None
            self.left=None
16          self.right=None
            self.n=None
18          self.data=None
            self.bins=None
20          self.verts=None
            self.patches=None
22          self.X = None
            self.Y = None
24          self.U = None
            self.V = None
26          self.Q = None
            self.cosList = np.zeros(15500)
28          self.sinList = np.zeros(15500)
            self.sc = None
30          self.kreisDataCount = 0

32
            # Einrichten des Layouts
34          self.figure = plt.figure(num = 2)
            self.ax = self.figure.add_subplot(111)
36          self.canvas = FigureCanvas(self.figure)
            self.toolbar = NavigationToolbar(self.canvas, self)
38
            self.buttonPlot = QPushButton('Plot')
40          self.buttonQuit = QPushButton('Schließen')

42          self.buttonQuit.clicked.connect(self.closewindow)
            self.buttonPlot.clicked.connect(self.plot)
44
            self.horizontalGroupBox = QGroupBox()
46          layoutBtn = QHBoxLayout()
            layoutBtn.addWidget(self.buttonPlot)
48          layoutBtn.addWidget(self.buttonQuit)
            self.horizontalGroupBox.setLayout(layoutBtn)
50
            layout = QVBoxLayout()
52          layout.addWidget(self.toolbar)
            layout.addWidget(self.canvas)
54          layout.addWidget(self.horizontalGroupBox)

56          self.figure.clf()
            self.ax.cla()
58
            self.setLayout(layout)
60
        # Beim schließen des Fensters den Figurepointer löschen
62      def closewindow(self):
```

```
64          self.close()

66      # Animationsfunktion: Histogramm
        def aniHistogram(self, i):
68          self.data = np.array(w2_angles)
            self.n, self.bins = np.histogram(self.data)
70          self.left = np.array(self.bins[:-1])
            self.right = np.array(self.bins[1:])
72          self.top = self.bottom + self.n
            self.verts[0::5, 0] = self.left
74          self.verts[0::5, 1] = self.bottom
            self.verts[1::5, 0] = self.left
76          self.verts[1::5, 1] = self.top
            self.verts[2::5, 0] = self.right
78          self.verts[2::5, 1] = self.top
            self.verts[3::5, 0] = self.right
80          self.verts[3::5, 1] = self.bottom

82          self.ax.set_xlim(self.left[0], self.right[-1])
            self.ax.set_ylim(self.bottom.min(), self.top.max())
84
            return [self.patch, ]
86
        # Animationsfunktion: Quiver-Plot
88      def aniQuiver(self, i):
            self.U = np.array(w2_cosinus)
90          self.V = np.array(w2_sinus)
            self.Q.set_UVC(self.U, self.V)
92
            return self.Q,
94
        # Animationsfunktion: Scatter-Plot
96      def aniScatter(self, i):
            if(self.kreisDataCount >= 15275):
98              self.kreisDataCount = 0
            for cos, sin in zip(w2_cosinus, w2_sinus):
100             self.cosList[self.kreisDataCount] = cos/(4095) * 3.3
                self.sinList[self.kreisDataCount] = sin/(4095) * 3.3
102             self.kreisDataCount = self.kreisDataCount + 1
            self.sc.set_offsets(np.c_[self.cosList, self.sinList])
104
        # Auswahl des zu plottenden Diagramms
106     def plot(self):

108         # Lokale Variablen
            self.cosList = np.zeros(15500)
110         self.sinList = np.zeros(15500)
            self.kreisDataCount = 0
112         self.figure.clf()
            self.ax.cla()
114
            # Wenn ein Histogramm geplottet werden soll
116         if (w2_diaName == "Histogramm"):
                time.sleep(1)
118             self.ax = self.figure.add_subplot(111)
                # Daten fürs Histogramm
120             self.data = np.array(w2_angles)
                # Übergabe der Daten an die Histogramm-Funktion
122             self.n, self.bins = np.histogram(self.data)
                # Die Ecken der Rechtecke
124             self.left = np.array(self.bins[:-1])
                self.right = np.array(self.bins[1:])
126             #print(len(self.left), len(self.right))
                self.bottom = np.zeros(len(self.left))
```

```
128
                self.top = self.bottom + self.n
130             nrects = len(self.left)

132             nverts = nrects*(1 + 3 + 1)
                self.verts = np.zeros((nverts, 2))
134             codes = np.ones(nverts, int) * path.Path.LINETO
                codes[0::5] = path.Path.MOVETO
136             codes[4::5] = path.Path.CLOSEPOLY
                self.verts[0::5, 0] = self.left
138             self.verts[0::5, 1] = self.bottom
                self.verts[1::5, 0] = self.left
140             self.verts[1::5, 1] = self.top
                self.verts[2::5, 0] = self.right
142             self.verts[2::5, 1] = self.top
                self.verts[3::5, 0] = self.right
144             self.verts[3::5, 1] = self.bottom

146             barpath = path.Path(self.verts, codes)
                self.patch = patches.PathPatch(barpath, facecolor='green', edgecolor='
                    blue', alpha = 0.5)
148             self.ax.add_patch(self.patch)

150             self.ax.set_xlim(self.left[0], self.right[-1])
                self.ax.set_ylim(self.bottom.min(), self.top.max())
152
                self.ax.set_xlabel('Winkel [°]')
154             self.ax.set_ylabel('Anzahl')
                self.ax.set_title('Histogramm der Sensor-Winkel')
156             self.ax.grid(True)

158             # Starten der Animate-Funktion
                self.ani2 = animation.FuncAnimation(self.figure, self.aniHistogram)
160             self.canvas.draw()

162         # Wenn ein Quiver geplottet werden soll
            elif (w2_diaName == "Quiver-Plot"):
164             time.sleep(1)
                self.ax = self.figure.add_subplot(111)
166             #print("Länge: ", len(w1_cosinus), len(w1_sinus))
                self.X = np.arange(1,(9 + (7 * w2_interpolation)),1)
168             self.Y = np.arange(1,(9 + (7 * w2_interpolation)),1)
                self.X, self.Y = np.meshgrid(self.X,self.Y)
170             self.U = np.array(w2_cosinus)
                self.V = np.array(w2_sinus)
172
                # Füllen der Quiver-Funktion
174             self.Q = self.ax.quiver(self.X, self.Y, self.U, self.V, color='r')
                self.ax.set_xlim(0,(9 + (7 * w1_interpolation)))
176             self.ax.set_ylim(0,(9 + (7 * w1_interpolation)))
                self.ax.set_aspect('equal', 'box')
178
                self.ax.set_xlabel('Zeile')
180             self.ax.set_ylabel('Spalte')
                self.ax.set_title('Quiver-Plot der Sensormatrix')
182             self.ax.grid(True)

184             # Starten der Animate-Funktion
                self.ani2 = animation.FuncAnimation(self.figure, self.aniQuiver)
186             self.canvas.draw()

188         # Wenn ein Scatter geplottet werden soll
            elif (w2_diaName == "Kreisplot"):
190             time.sleep(1)
                self.ax = self.figure.add_subplot(111)
```

```
192                 self.kreisDataCount  = 0
                    for cos, sin in zip(w1_cosinus, w1_sinus):
194                     self.cosList[self.kreisDataCount] = cos/(4095) * 3.3
                        self.sinList[self.kreisDataCount] = sin/(4095) * 3.3
196                     self.kreisDataCount  = self.kreisDataCount  + 1

198                 # Füllen der Scatter-Funktion
                    self.sc = self.ax.scatter(self.cosList, self.sinList, c="g", alpha=0.5,
                         marker=".")
200                 self.ax.set_xlim(-0.4,0.4)
                    self.ax.set_ylim(-0.4,0.4)
202                 self.ax.set_aspect('equal', 'box')

204                 self.ax.set_xlabel('Cosinus')
                    self.ax.set_ylabel('Sinus')
206                 #self.ax.set_legend(loc='upper left')
                    self.ax.set_title('Scatter-Plot␣der␣Sensormatrix')
208                 self.ax.grid(True)

210                 # Starten der Animate-Funktion
                    self.ani2 = animation.FuncAnimation(self.figure, self.aniScatter)
212                 self.canvas.draw()
```

Quellcode B.5: Mein-Funktion, globale Variablen und Importe.

```
from __future__ import unicode_literals

import sys
import time
import os
import matplotlib
matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import matplotlib.path as path
import matplotlib.patches as patches
import random
import serial
import numpy as np

from matplotlib.pylab import *
from serial import Serial
from serial import SerialException
from numpy import arange, sin, pi
from PyQt5 import QtGui, uic, QtWidgets, QtPrintSupport
from PyQt5.QtCore import pyqtSignal, QObject, QThread, QTimer
from PyQt5.QtWidgets import *
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
    NavigationToolbar
from matplotlib.figure import Figure

# Globale Variabeln
angle               = 0.0

# Fenster 1
w1_befehl           = "3000r"
w1_sinus            = []
w1_cosinus          = []
w1_diagramm         = 0
w1_interpolation        = 0
w1_filtertype           = 0
w1_filter           = []
w1_angles           = []
w1_anpassen         = 0
w1_diaName          = ""
w1_kreisDataCount       = 1

# Fenster 2
w2_befehl           = "3000r"
w2_sinus            = []
w2_cosinus          = []
w2_diagramm         = 0
w2_interpolation        = 0
w2_filtertype           = 0
w2_filter           = []
w2_angles           = []
w2_anpassen         = 0
w2_diaName          = ""
w2_kreisDataCount       = 1

# Fenster 3
w3_befehl           = "3000r"
w3_sinus            = []
w3_cosinus          = []
w3_diagramm         = 0
w3_interpolation        = 0
```

```
62  w3_filtertype           = 0
    w3_filter           = []
64  w3_angles           = []
    w3_anpassen             = 0
66  w3_diaName          = 0
    w3_kreisDataCount       = 1
68
    # Fenster 4
70  w4_befehl           = "3000r"
    w4_sinus            = []
72  w4_cosinus          = []
    w4_diagramm         = 0
74  w4_interpolation        = 0
    w4_filtertype           = 0
76  w4_filter           = []
    w4_angles           = []
78  w4_anpassen         = 0
    w4_diaName          = ""
80  w4_kreisDataCount       = 1

82
    '''
84      main() Klasse, die für einen sauberen Programmstart aufgerufen wird
        und die QApplication initialisiert und das erste Fenster.
86  '''
    class main():
88
        app = QtWidgets.QApplication(sys.argv)
90      # initialisierung des Kalibrierungsfensters
        calib = CalibrationWindow()
92      # öffnen des Kalibrierungsfensters
        calib.show()
94
        sys.exit(app.exec_())
96

98  '''
        Überleitung in die main() Funktion
100 '''
    if __name__ == "__main__":
102     main()
```

C Berechnungen mit Octave

Quellcode C.1: Berechnung der Filterkoeffizienten.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Datum: 24.07.2018                         %
%           Autor: Simon Rindelaub                    %
%           Name : filterkoeff_berechnung.m           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
clear all
clc

tau = 100 * 10^-6;      %Timeconstant tau
fs = 80 * 10^3;         %Samplingfreq.
fc = 1/tau;             %Grenzfrequenz Durchlassbereich
Omega_C = (2 * pi  * fc)/fs;
% N=8                   % Größe der Filtermatrix 8x8
N = 15;                 % Größe der Filtermatrix 15x15
buff = (N-1)/2;

w = zeros(N,N);
h1 = zeros(N,N);

for j = 1 : N
  for i = 0 : N-1
    % Berechnung der Fensterkoeffizienten
    w(i+1,j) = 0.54 + 0.46 * cos((2 * pi * ((i)-buff)) / N);

    % Berechnung der idealen Tiefpasskoeffizienten
    if i == ((N-1)/2)
      h1(i+1,j) = Omega_C/pi;
    else
      h1(i+1,j) = sin(((i) - buff) * Omega_C) / (((i) - buff) * pi);
    end
  end
end

% Multiplikation von Fenster- und Filterfunktion
h2 = h1 * w;
```

D Grafiken und Schaltpläne

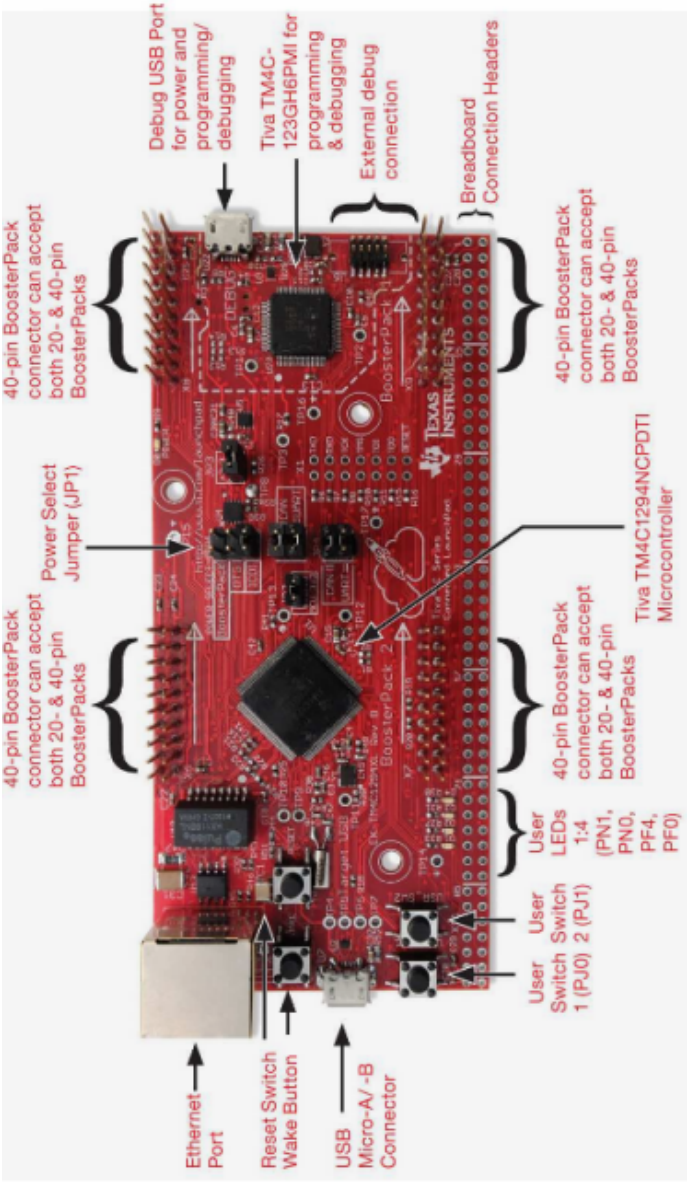


Abbildung D.1: Pinbelegung des Tiva Boards TM4C1294 [7].

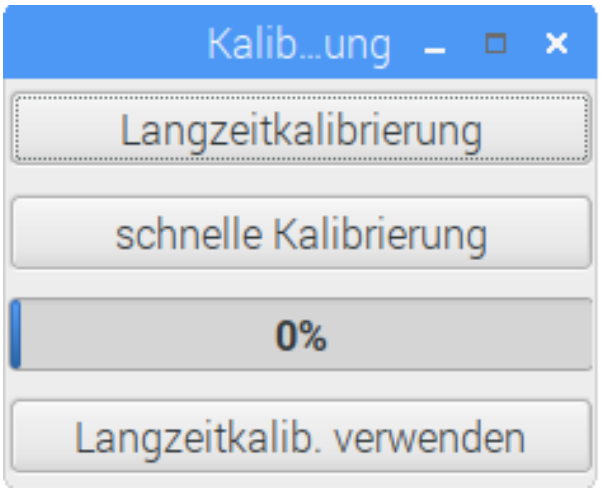


Abbildung D.2: Fenster zum Durchführen oder Laden einer Kalibrierung.

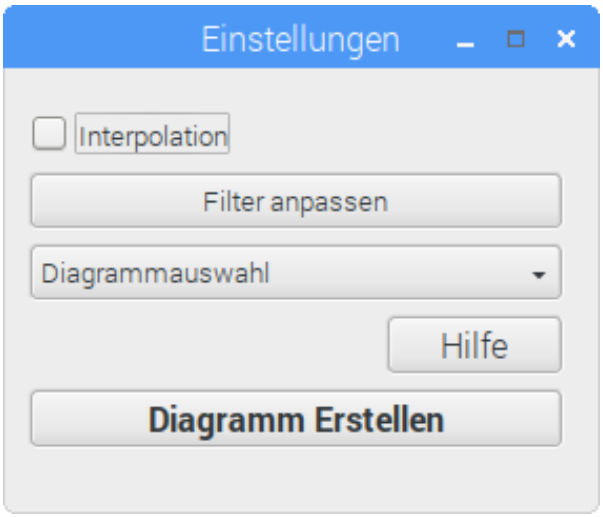


Abbildung D.3: Hauptfenster für grundlegende Einstellungsmöglichkeiten.

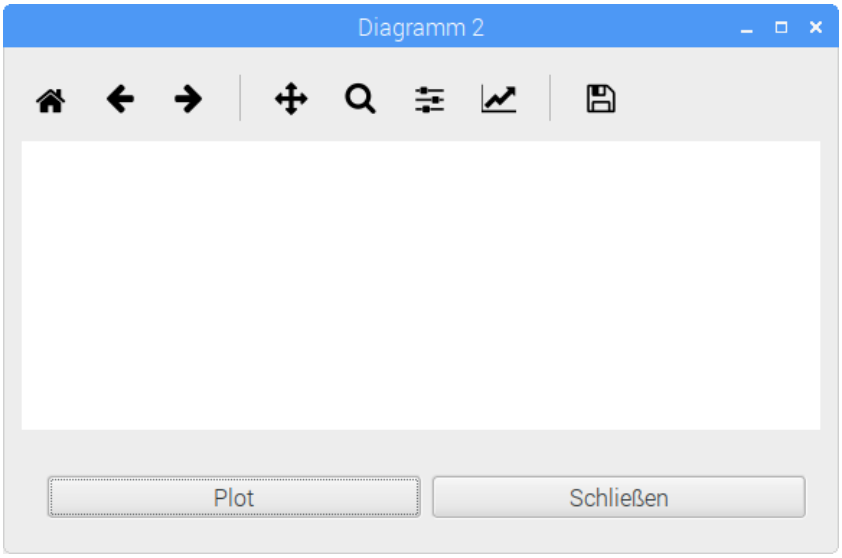


Abbildung D.4: Beispiel eines Diagrammfensters.

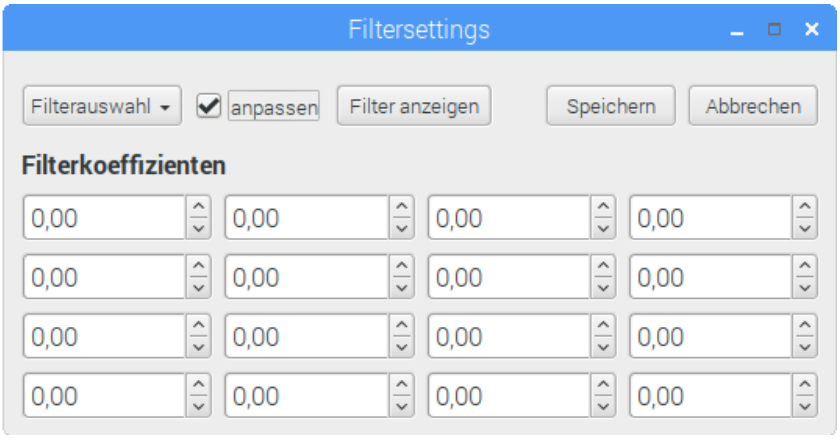


Abbildung D.5: Einstellungsfenster für eine 8×8 Filtermatrix.

Abbildung D.6: Einstellungsfenster für eine 15×15 Filtermatrix.

Abbildung D.7: 15kA/m Halbacharray zum durchführen der Langzeitmessung.

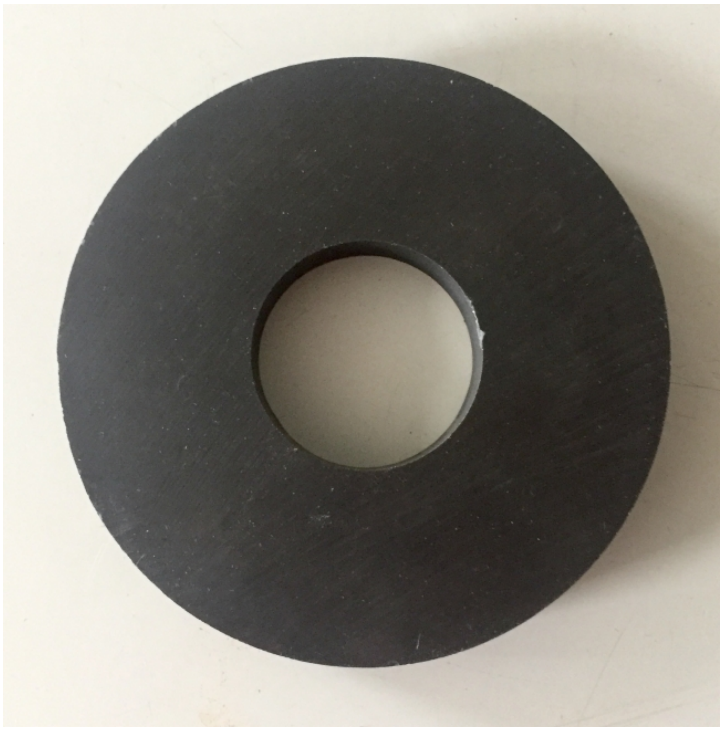


Abbildung D.8: Quadrupol zum durchführen der schnellen Kalibrierung.

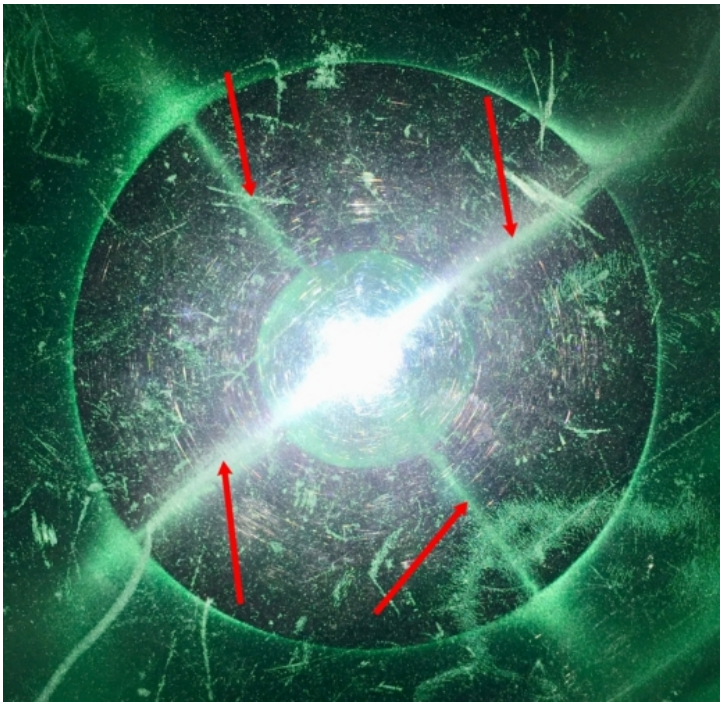


Abbildung D.9: Quadrupol mit Metallspanfolie zur Visualisierung der Feldlinien.

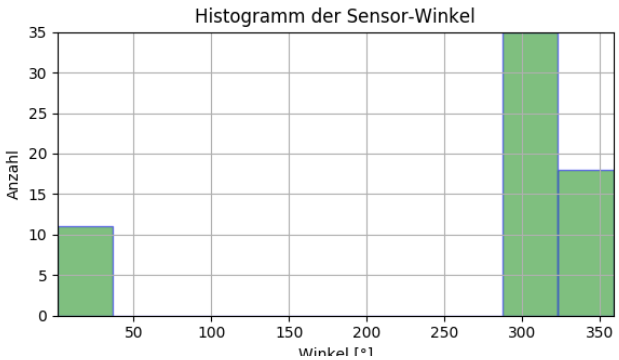


Abbildung D.10: Aussteuerung von 0° bis 360° des Histogramms für eine 8×8 Matrix.

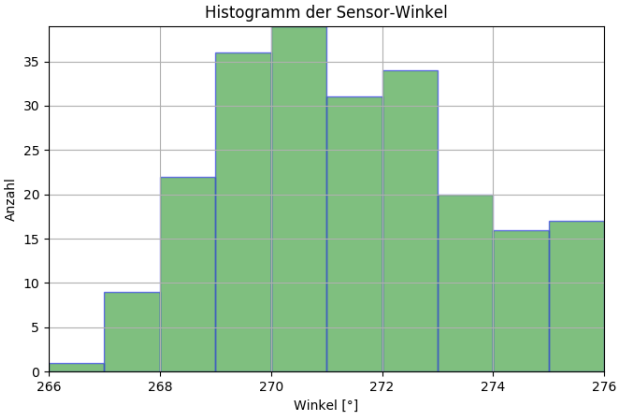


Abbildung D.11: Histogramm für eine 15×15 Matrix ohne Filterung.

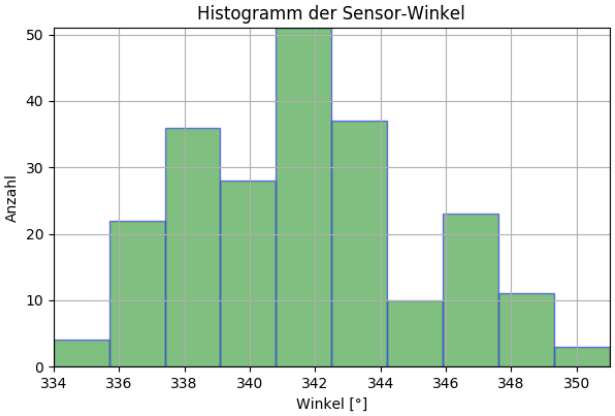


Abbildung D.12: Histogramm für eine 15×15 Matrix mit Bandpass-Filterung.

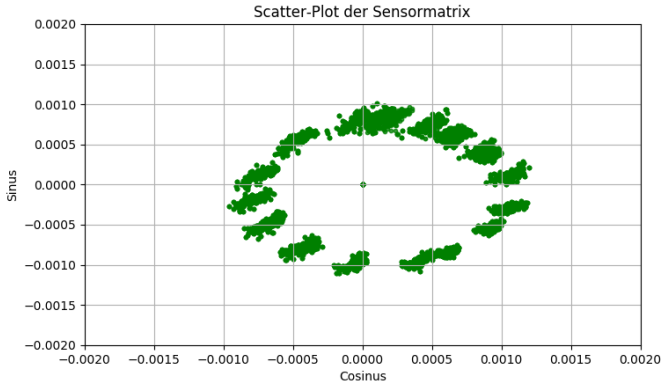


Abbildung D.13: Scatter-Plot einer 15×15 Matrix mit Bandpass.

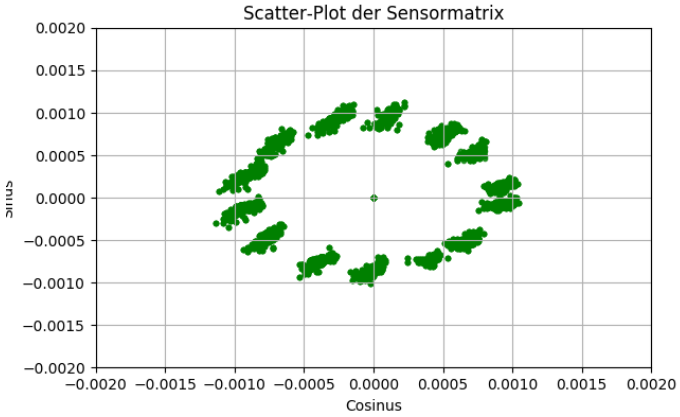


Abbildung D.14: Scatter-Plot einer 15×15 Matrix mit Tiefpass.

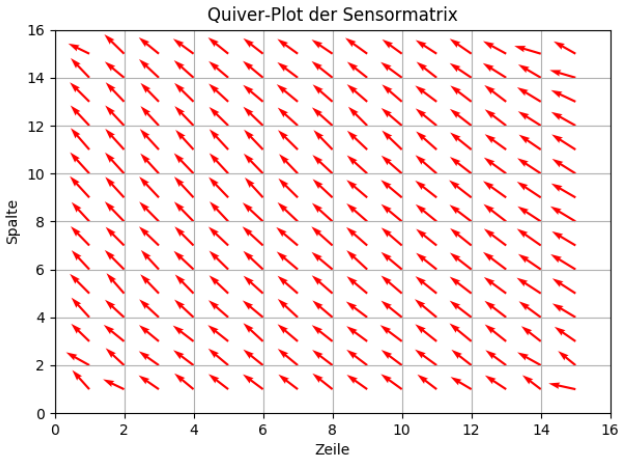


Abbildung D.15: Quiver-Plot einer 15×15 Matrix mit Tiefpass.

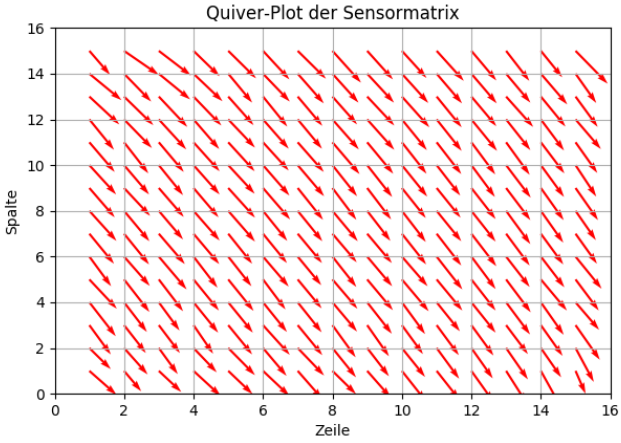
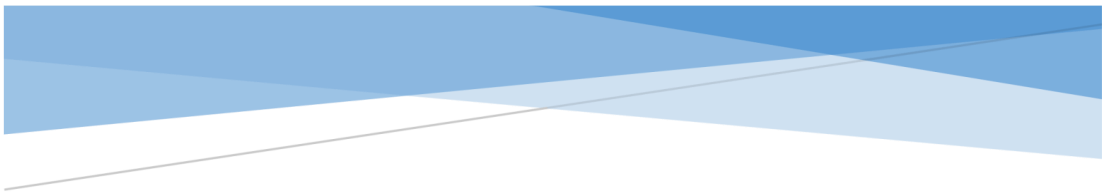


Abbildung D.16: Quiver-Plot einer 15×15 Matrix mit Bandpass.

E Bedienungsanleitung



BEDIENUNGSANLEITUNG

Für das Demonstrationssystem des

Sensor-Arrays im ISAR Projekt

Simon Rindelaub
Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Vorwort

Hamburg, den 26.09.2018

Diese Anleitung wurde im Rahmen der Bachelorarbeit „Signalverarbeitung für magnetoresistive Sensor-Arrays mit Controller und Einplatinen-Computer“ erstellt.

Es wird keine Gewährleistung für die Richtigkeit aller Angaben übernommen und der Nutzer wird dazu angehalten, diese vor der Umsetzung kritisch zu prüfen.

Das Gesamtsystem wurde lediglich in Betrieb genommen und nicht ausgiebig getestet, da es sich um einen Prototyp handelt. Es können daher noch Fehler in Soft- und Hardware vorhanden sein. Eine Liste von Verbesserungsvorschlägen und bekannten Fehlern ist in der Bachelorarbeit im Kapitel „Zusammenfassung und Ausblick“ angegeben.

Das Gesamtsystem wurde mit den in dieser Anleitung verwendeten Controllern und Softwareversionen für die Controller betrieben. Bei Abweichungen dazu kann ein Funktionieren nicht gewährleistet werden.

Inhalt

Systemvoraussetzungen

Software

- Auf dem Raspberry Pi ist Raspian in der Version XY installiert
- Die Software auf dem Mikrocontroller ist auf der CD im Ordner „Software Tiva-Board“ zu finden
- Der C-Code wurde mit dem Code Composer Studio v6 8.0.0 auf eine PC mit Ubuntu v17.04 entwickelt
- Der Python-Code befindet sich auf der CD im Ordner „Software Raspberry Pi“
- Der Python-Code wurde auf dem Raspberry Pi mit der IDE Geany 1.29 entwickelt
- Eine Liste der installierten Pakete auf dem Raspberry Pi ist im Anhang einzusehen

Hardware

- Tiva Board TM4C1294
- TMR Sensormatrix mit NVE AAT00-10E Sensoren
- Raspberry Pi 3 Model B
- Mini-USB Kabel
- Netzteil mit 5V Mini USB Versorgung
- HDMI Kabel
- USB-Maus
- USB-Tastatur
- Monitor mit HDMI Eingang
- Netzkabel für den Monitor

Aufbau des Systems

Die Sensorplatine wird mit der langen Pinleiste auf der langen Pinleiste des Tiva-Boards fixiert, sodass die Platinen sich übereinander befinden. In den „Debug“-Eingang des Tiva-Boards wird das Mini-USB Kabel eingesteckt und mit dem Raspberry Pi verbunden. An zwei weitere USB-Steckplätzen werden die Maus und die Tastatur angeschlossen. Das HDMI Kabel verbindet den Raspberry Pi und den Monitor. Es kann für den Monitor auch ein Adapter von HDMI zu VGA/DVI verwendet werden. Der Aufbau ist in Abbildung 1 dargestellt.

Zum Schluss wird der Raspberry Pi mit der Stromversorgung verbunden. Dann startet das System.

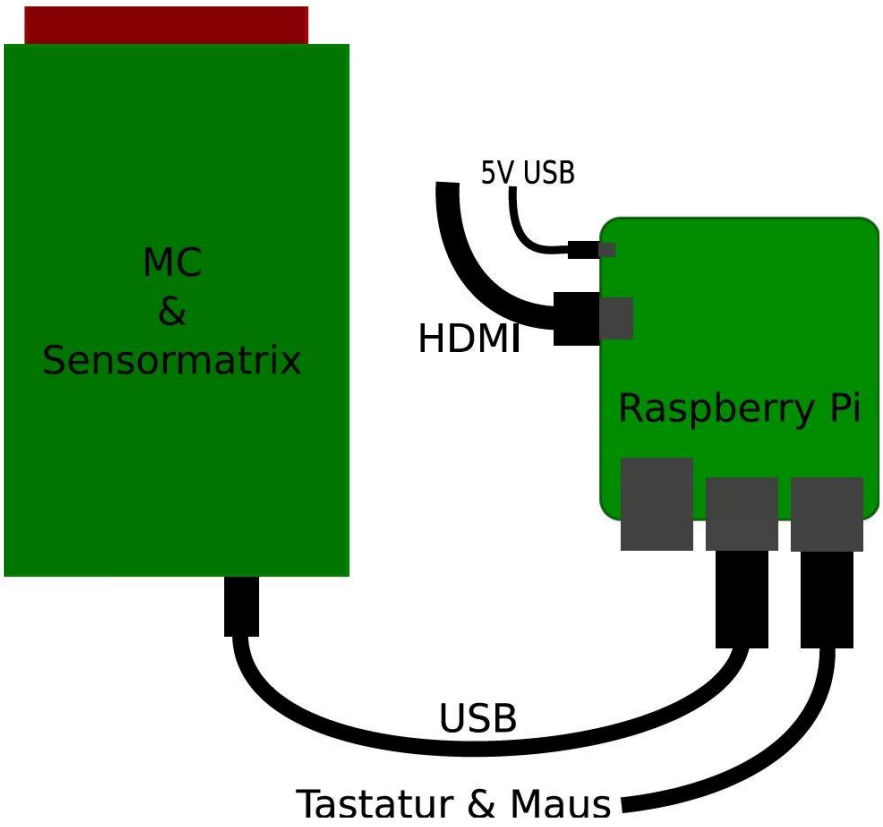


Abbildung 1: Systemaufbau des Controllers und Einplatinen-Computers

Programmstart

Nachdem der Raspberry Pi hochgefahren ist, wird eine Meldung zur SSH Verbindung angezeigt. Diese kann geschlossen werden, weil das Standartpasswort bestehen bleiben soll.

In dem Ordner „Python GUI Bachelorarbeit“ befinden sich alle notwendigen Dateien. Darunter die .txt Files für den Offset, die Layouts der Fenster und der Python Programmcode. Mit einem Rechtsklick auf die Datei „main.py“ wird ein Auswahlmenü angezeigt. Hier wird auf öffnen mit „Geany“ geklickt. Abbildung 2 zeigt den Bildschirm, den man jetzt sehen sollte.

Zum Programmstart geht man nun wie folgt vor:

1. Klicken auf den Reiter Erstellen > Kommandos zum Erstellen konfigurieren. Hier wird überprüft ob in den Felder der Befehl mit „python3“ ausgeführt wird, siehe Abbildung 3
2. Durch Schließen des Einstellungsfensters werden die Änderungen gespeichert.
3. Mit klicken auf den Button „Build“ (markiert in Abbildung 4) wird der Code ausgeführt

Nachfolgend werden die einzelnen Fenster mit ihren Auswahlmöglichkeiten erläutert.

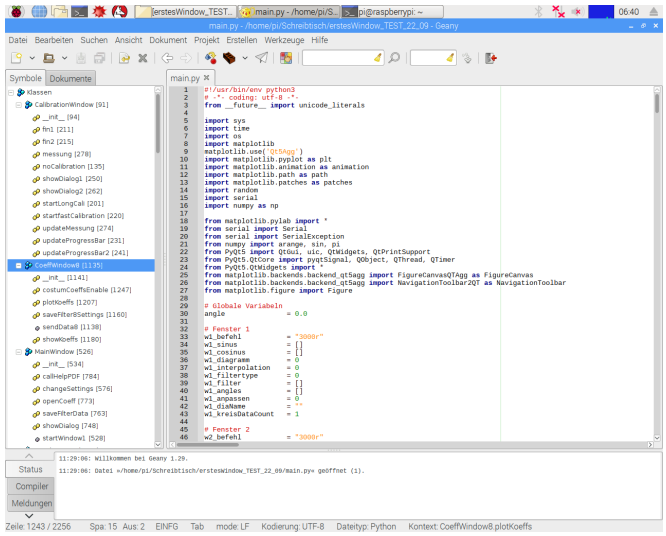


Abbildung 2: Startbildschirm nach dem Öffnen von GEANY

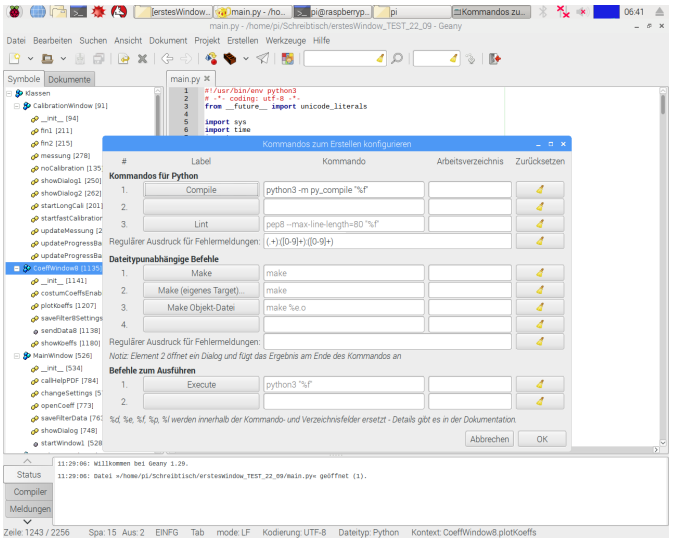


Abbildung 3: Einstellungen für ein Python3 Skript

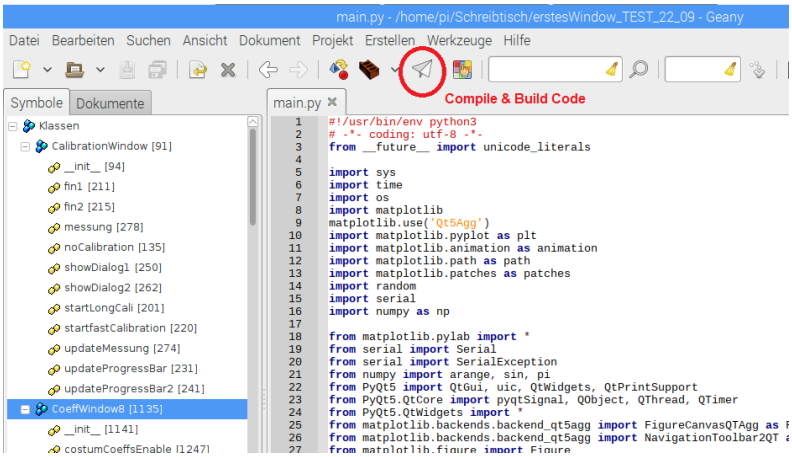


Abbildung 4: Button zum Compilieren und Starten des Programms

Kalibrierungsfenster

Das Kalibrierungsfenster dient dazu, den Offset der Sensoren zu kalibrieren. Dazu gibt es zwei Möglichkeiten. Mit der **Langzeitkalibrierung** wird alle 15° eine Messung durchgeführt und am Ende der Mittelwerte über alle Messungen gebildet. Dadurch wird für spätere Rechnungen ein individueller Offset von jedem Sensor abgezogen.

Die **schnelle Kalibrierung** ist eine „one-shot" Aufnahme und es wird mit dem Mittelwert über alle Sensoren gerechnet. Damit fallen Ausreißer deutlich mehr auf die anderen Sensoren zurück.

Mit dem Button „**Langzeitkalib. verwenden**" wird direkt zum Hauptfenster gewechselt. Vorher werden jedoch die Werte der letzten Langzeitkalibrierung aus den .txt Dateien an den Mikrocontroller gesendet.

In Abbildung 5 ist das Kalibrierungsfenster mit den Buttons dargestellt.

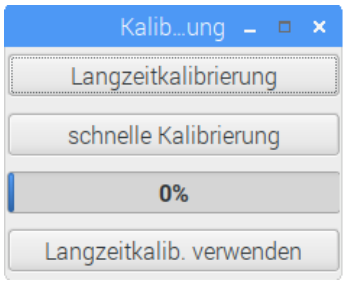


Abbildung 5: Kalibrierungsfenster mit Bedienflächen

Langzeitkalibrierung

Durch diese Kalibrierung wird der Nutzer vom Programm geführt. Es wird ein Dialog-Fenster angezeigt, wo die Anweisungen ausgegeben werden. Vor einer Messung hat man 3 Sekunden Zeit, das Halbacharray um 15° zu drehen. Dann wird eine Messung durchgeführt und der Count-Down von neuem gestartet. Das Halbacharray ist in Abbildung 6 dargestellt. Dieses wird im Laufe der Kalibrierung um 360° über der Sensormatrix rotiert. Somit finden 24 Messungen statt.

Abbildung 6: 15kA/m Halbacharray zur Kalibrierung

Schnelle Kalibrierung

Vor dem Start der Kalibrierung muss der Quadrupol, abgebildet in Bild 7, bereits über den Sensoren platziert werden. Bei betätigen des Buttons **schnelle Kalibrierung**, wird die Messung direkt durchgeführt.

Abbildung 7: Quadrupol zur "one-shot" Kalibrierung

Hauptfenster

Das Hauptfenster, dargestellt in Abbildung 8, steuert jegliche Einstellungsmöglichkeiten.

Mit der Checkbox **Interpolation** wird die eigentliche 8x8 Matrix virtuell auf eine 15x15 Matrix erweitert. Damit gibt es dann 225 Sensoren, deren Messwerte in den Diagrammen dargestellt werden können. Wenn diese Box gecheckt ist, wird bei den Filtereinstellungen ein größeres Fenster geöffnet, um die Einstellungen für die 15x15 Matrix vornehmen zu können.

Unter dem Drop-Down Menü zur **Diagrammauswahl** können drei verschiedene Diagramme ausgewählt werden. Darauf wird im Kapitel Diagrammfenster weiter eingegangen.

Eine Schaltfläche, die ein weiteres Fenster öffnet, ist der Button **Filter anpassen**. Das anschließend geöffnete Fenster wird in Kapitel „Filtereinstellungsfenster“ näher beschrieben.

Der Button **Hilfe** hat zu dieser Anleitung geführt.

Mit dem letzten Button **Diagramm Erstellen**, wird erst einmal überprüft, ob bereits vier Fenster geöffnet sind. Ist dies nicht der Fall werden die Einstellungen übernommen und das Diagrammfenster wird geöffnet.

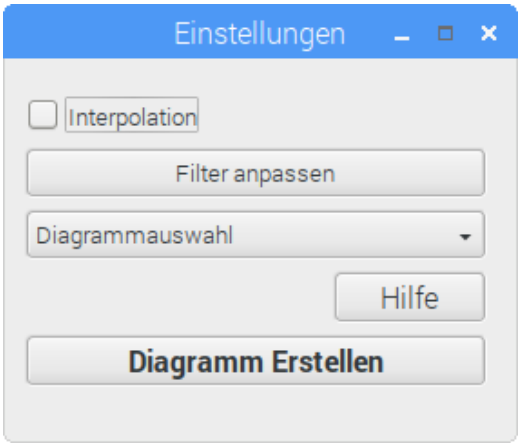


Abbildung 8: Hauptfenster mit Schaltflächen

Filtereinstellungsfenster

Dieses Fenster dient der alleinigen Einstellung für die Signalfilterung. Wenn hier keine Änderungen vorgenommen werden, werden die Messwerte lediglich offsetbereinigt.

Das Auswahlmenü **Filter wählen** bietet die Möglichkeit der Wahl zwischen Tiefpass- und Bandpassfilterung. Die Koeffizienten dafür sind bereits berechnet worden und werden in dem unteren Feld angezeigt. Bei einer 8x8 Matrix wird ein 4x4 Feld angezeigt, wie in Abbildung 9 dargestellt, da die Filtermatrix X- und Y-Achsen-Symmetrisch ist. Zur Übersicht wird daher nur ein Viertel aller Werte angezeigt. Für die Filterung werden diese Werte in die restliche Matrix übertragen. Veranschaulicht wird dies in Abbildung 11, für eine 8x8 Matrix und in Abbildung 12 für eine 15x15 Matrix. Für eine 15x15 Matrix wird hingegen ein 8x8 Feld zur Bearbeitung angezeigt, wie in Abbildung 10 dargestellt. Die siebte Zeile und die siebte Spalte kommen nur einmal mit den Werten in der Matrix vor. Dafür gilt die Symmetrie für den Rest des Quadranten. Quadrant eins wird in Quadrant zwei kopiert und anschließend beide in den Quadranten drei.

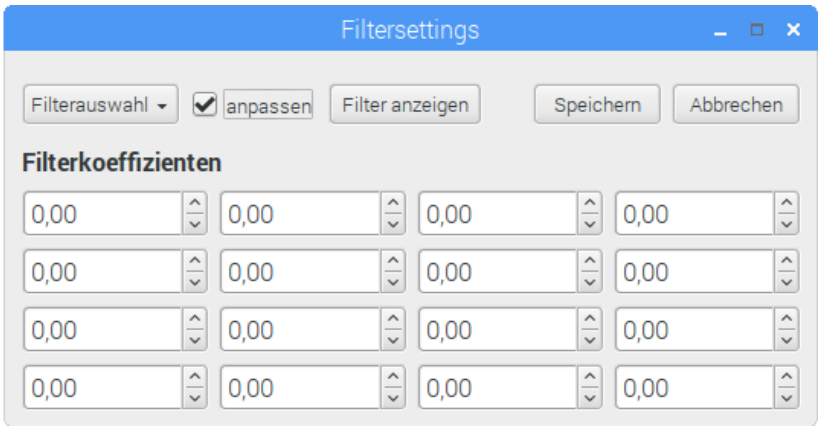


Abbildung 9: Darstellung des Einstellungsfenster für eine 8x8 Matrix

Die Checkbox **Anpassen** muss gesetzt werden, wenn die Koeffizienten manuell angepasst werden sollen. Dann werden die Felder für eine Eingabe freigegeben.

Mit dem Button **Filter anzeigen** werden die Koeffizienten in einem Plot visualisiert. Man kann direkt die getätigte Eingabe überprüfen. Mit einer Farbskala ist eine Zuordnung von Farbe und Wert sofort möglich.

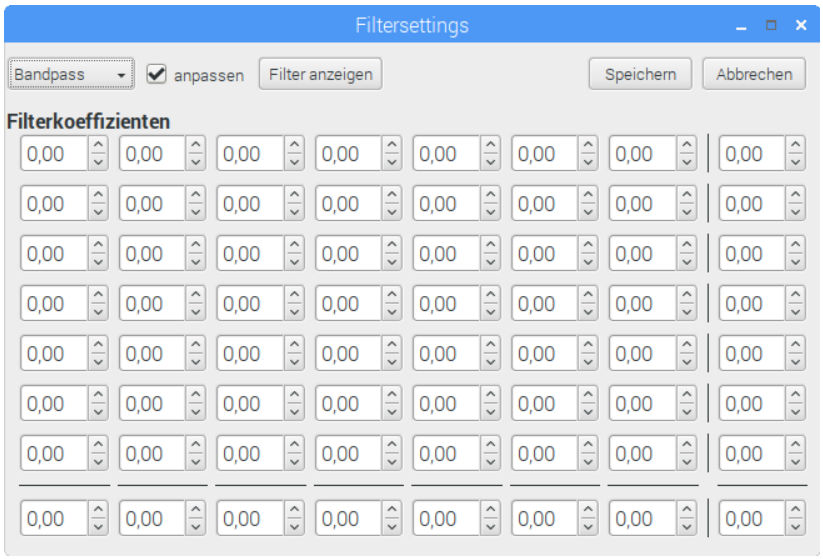


Abbildung 10: Darstellung des Einstellungsfenster für eine 15x15 Matrix

Mit der Schaltfläche **Speichern** werden die alle Einstellungen übernommen und das Fensterwird geschlossen.

Durch **Abbrechen** wird das Fenster ohne Speicherung geschlossen.

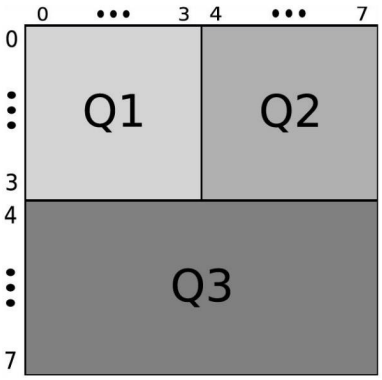


Abbildung 11: Darstellung des Kopieren der Quadranten bei einer 8x8 Matrix

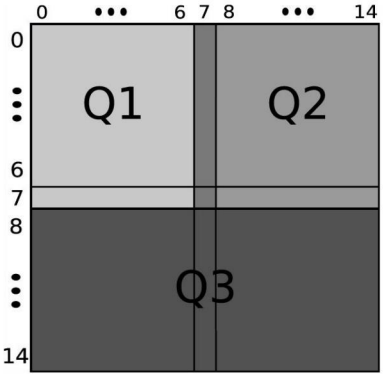


Abbildung 12: Darstellung des Kopierens der Quadranten bei einer 15x15 Matrix

Diagrammfenster

Die Diagrammfenster sind für das Darstellen des Plots zuständig. In Abbildung 13 ist zu sehen, dass das Fenster nur zwei Buttons benötigt. Mit dem Button **Plot** werden die Eingabedaten geladen und der gewünschte Plot dargestellt.

Mit der Schaltfläche **Abbrechen** wird das Fenster wieder geschlossen.

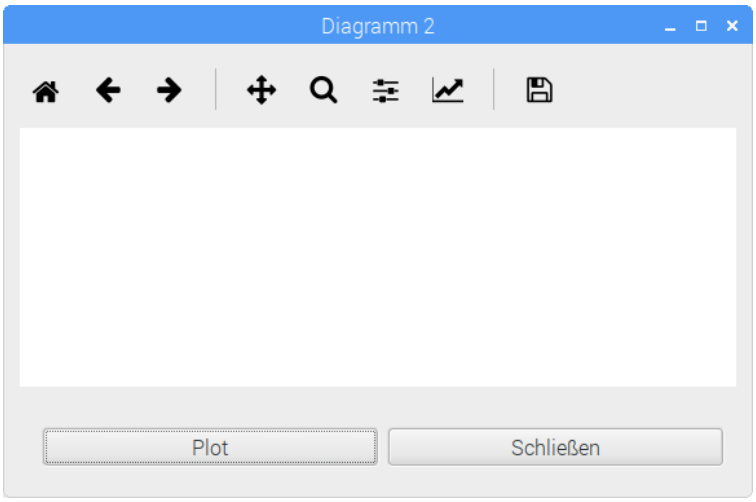


Abbildung 13: Beispiel eines Diagrammfensters

Die Toolbar von Matplotlib ist am oberen Fensterrand positioniert. Hier kann in die Grafik gezoomt werden, der Sichtbereich verschoben, Auflösungseinstellungen vorgenommen oder der Graf als Bild gespeichert werden.

Nachfolgende Diagramme sind in der Auswahl möglich. Es wird kurz erklärt was dargestellt wird und ein Beispielplot gezeigt.

Histogramm

Bei einem Histogramm wird die relative Anzahl in einem Wertebereich vorkommender zahlen angegeben. Die wird durch die Höhe des Balkens angegeben. Die Breite gibt den Werteberiech an. Bei dem Histogramm werden die mit dem CORDIC berechneten Winkel der Sensoren angegeben. Die

Streuung kann hier sehr gut beobachtet werden. Ein Beispiel von dem Raspberry Pi ist in Abbildung 14 dargestellt.

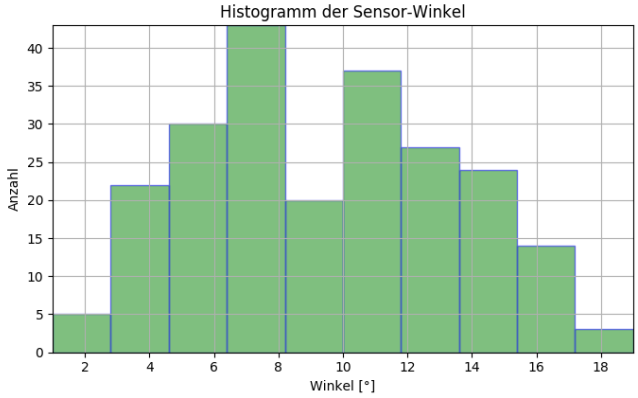


Abbildung 14: Beispiel eines Histogramm mit 15x15 Sensoren

Quiver-Plot

Mit einem Quiver-Plot können Feldlinien und Feldstärken dargestellt werden. Dieses geschieht mit Pfeilen. Die Länge des Pfeils ist proportional zur Signalstärke und die Richtung des Pfeils gibt den Winkel des Feldes an. Diese Informationen werden aus den Messdaten der Sinus- und Cosinussignale gewonnen. Ein Pfeil steht für einen Sensor.

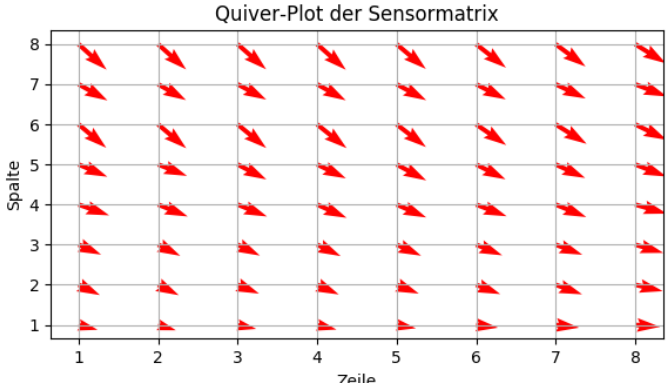


Abbildung 15: Beispiel eines Quiver-Plots mit 8x8 Sensoren

Scatter-Plot

Der Scatter Plot stellt für zwei übergebene Arrays die index-gleichen Werte in einem XY-Diagramm als Punkte dar. Diese Punkte-Wolken sollen sich nach einem vorigen Offsetausgleich auf einer Kreisbahn um den Nullpunkt bewegen. Jeder Sensor wird für den jeweiligen Messzeitpunkt als Punkt in dem Koordinatensystem dargestellt. Somit können auffällige Streuungen der Sensoren oder eines einzelnen Sensors festgestellt werden. Ein Punkt einer Wolke steht für einen Sensor.

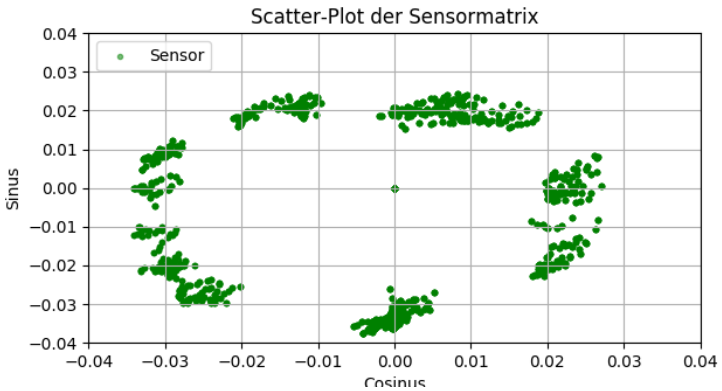


Abbildung 16: Beispiel eines Scatter-Plots mit 8x8 Sensoren

Anhang

Liste der wichtigen Python Bibliotheken:

- python3-matplotlib 2.0.0+dfsg1
- python3-numpy 1:1.12.1-3
- python3-pip 9.0.1-2+rpt2
- python3-pyqt5 5.7+dfsg-5
- python3-serial 3.2.1-1
- qt5-default 5.7.1+dfsg-3
- qt5-gtk-platfo 5.7.1+dfsg-3
- qt5-qmake:armh 5.7.1+dfsg-3
- qt5-style-plug 5.0.0+git16.

F CD

Auf der beigefügten CD befinden sich sämtlich Programme, Messdaten und Benutzerhandbücher, die für diese Arbeit erstellt bzw. verwendet worden. Abbildung F.1 zeigt die Ordnerstruktur, wie sie auf der CD vorhanden ist.

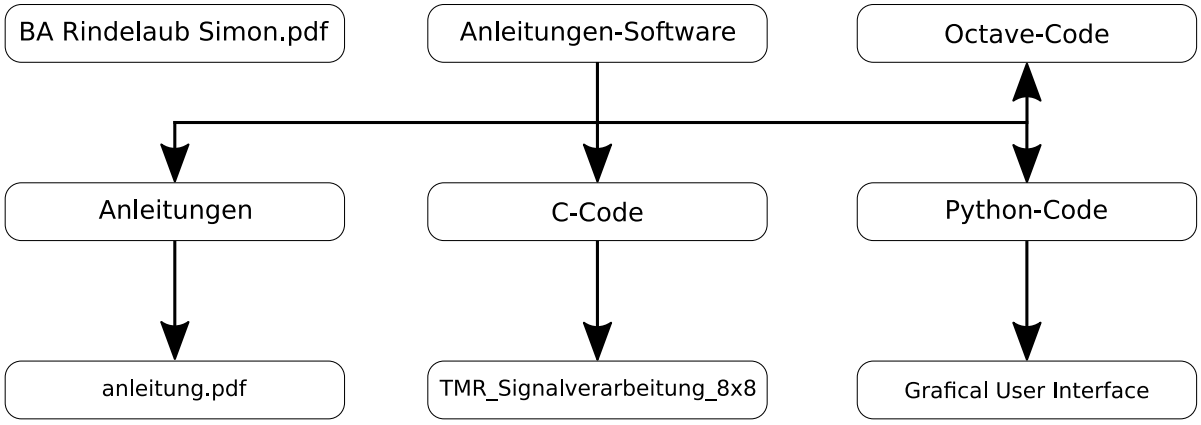


Abbildung F.1: Ordnerstruktur der beigefügten CD.

Selbstständigkeitserklärung

Hiermit versichere ich, Simon Karl Eduard Rindelaub,dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 26. September 2018