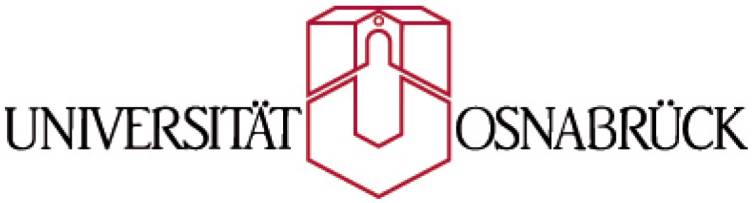


Bachelorthesis

Entwicklung einer Ansteuerungssoftware für eine magneto-optische Messapparatur

eingereicht von
Kamil Balinski - 925655

am
28. November 2010

Fachbereich Physik
AG Dünne Schichten und Grenzflächen
Prof. Dr. Joachim Wollschläger
Jun. Prof Dr.-Ing. Elke Pulvermüller

Inhaltsverzeichnis

1 Einleitung

Dünne magnetische Schichten spielen in der heutigen Technik eine immer wichtigere Rolle. Sie werden zum Beispiel in den Leseköpfen heutiger Festplatten eingesetzt (GMR-Effekt). Es werden zudem Versuche unternommen neue Speichermedien zu entwickeln, wie zum Beispiel MRAMs, die den magnetischen Tunnelwiderstand (TMR-Effekt) nutzen. Damit so eine Entwicklung Erfolg hat, ist es wichtig das Magnetisierungsverhalten der verwendeten Materialen zu kennen. An der Universität Osnabrück in der AG `Dünne Schichten und Grenzflächen` wird seit 2007 der magneto-optische Kerr-Effekt zur Untersuchung verschiedener magnetischer Materialstrukturen genutzt. Mit dem magneto-optschen Kerr-Effekt ist es möglich Ummagnetiesierungsprozesse von Materialien zu beschreiben oder Eigenschaften wie schwere und leichte Magnetisierungsachsen eines Matrials zu bestimmen.

Im Rahmen dieser Arbeit wurde eine Ansteuerungssoftware entwickelt, die die Untersuchung von magnetischen Materialien mit dem magneto-optischen Kerr-Effekt in der AG `Dünne Schichten und Grenzflächen` erleichtern soll. Die Ansteuerung sollte neben der direkten Steuerung der Messanlage auch automatische Messungen ermöglichen. Dabei sollte die Einstellung der Messung möglichst vereinfacht werden. Außerdem sollte die neue Ansteuerungssoftware über eine Benutzeroberfläche verfügen und sicher stellen, dass bei der Benutzung dieser keine Schäden durch unpassende Eingaben entstehen können.

In Kapitel 2 werden die theoretischen Grundlagen des magneto-optischen Kerr-Effekts vorgestellt. Diese beschränken sich auf den für das Programm relevanten Umfang. Die experimentellen Begebenheiten und die damit einhergehenden Probleme werden in dem Kapitel 3 erörtert. Wie die Ansteuerungssoftware realisiert werden sollte, wird in Kapitel 4 diskutiert. Schließlich wird die implementierte Ansteuerungssoftware in Kapitel 5 vorgestellt.

2 Theoretische Grundlagen

In diesem Kapitel werden die Grundlagen des magnetooptischen KERR-Effekts (MOKE) erläutert. Dabei werden nach einer allgemeinen Beschreibung (Kapitel 2.1) die verschiedenen MOKE-Arten vorgestellt (Kapitel 2.2). Die Technik vectorial magnetometry (kurz: vector-MOKE), mit der man die einzelnen Komponenten des Magnetisierungsvektors der Probe bestimmen kann, wird in Kap. 2.4 eingeführt. Anschließend werden die wichtigen Größen einer Hysterese in Kapitel 2.5 erläutert. Am Ende der theoretischen Grundlagen wird in Kapitel 3.3 das Materialsystem Fe/MgO(001) beschrieben.

2.1 Der magnetooptische KERR-Effekt

Der magnetooptische KERR-Effekt wurde nach dem schottischen Physiker JOHN KERR benannt, der den Effekt 1876 entdeckte und 1877 beschrieb[1]. Hierbei handelt es sich um die Änderung von Lichteigenschaften, die nach einer Reflexion von polarisiertem Licht an magnetisierten Proben auftritt.

Es ändert sich die Richtung und die Elliptizität der Polarisation bzw. die Intensität des Lichts in Abhängigkeit von der Magnetisierung der Probe, von der das Licht reflektiert wird.

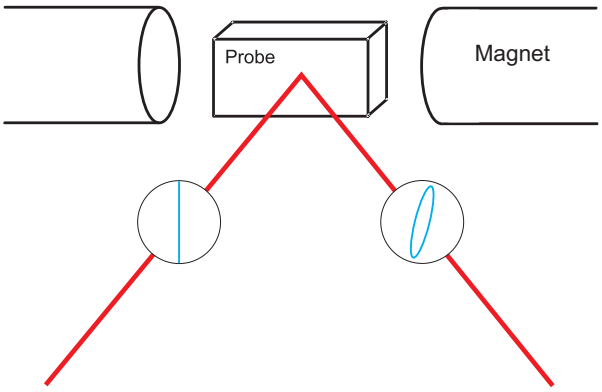


Abbildung 2.1: Skizze des KERR-Effekts. Linear polarisiertes Licht wird an der Probe reflektiert und ist danach elliptisch polarisiert.

Wird linear polarisiertes Licht auf einer magnetisierten Oberfläche reflektiert, so ist der reflektierte Lichtstrahl elliptisch polarisiert (Abbildung 2.1). Zusätzlich ist die Hauptachse der Ellipse um einen bestimmten Winkel Θ_K von der ursprünglichen Richtung der

Polarisierung gedreht.
Diese Richtungsänderung Θ_K bezeichnet man als den KERR-Winkel. Und das Verhältnis der beiden Halbachsen der Ellipse des elliptisch polarisierten Lichts wird als die KERR-Elliptizität

$$e_K \quad = \quad \tan \epsilon_K = \frac{E_{min}}{E_{max}} \tag{2.1}$$

bezeichnet (Abbildung 2.2).

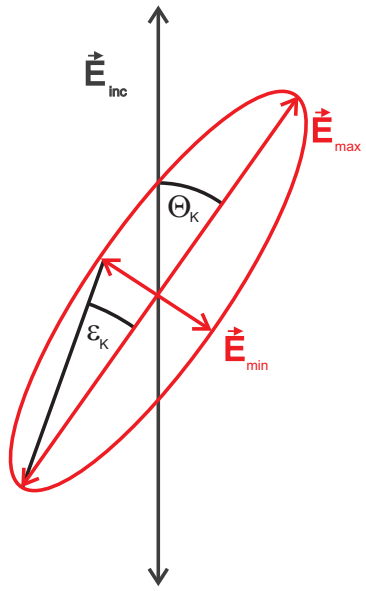


Abbildung 2.2: Polarisation und ihre Änderung beim KERR-Effekt. Θ_K: KERR-Winkel, e_K: KERR-Elliptizität, ϵ_K: Elliptizitätswinkel, E_{inc}: einfallende Lichtwelle, E_{max} und E_{min}: reflektierte Lichtwelle.

Der KERR-Winkel und die KERR-Elliptizität werden zu einer Größe zusammengefasst, die man als den komplexen KERR-Winkel Φ_K bezeichnet, er lautet

$$\Phi_K \quad = \quad \Theta_K + ie_K \ . \tag{2.2}$$

Dabei ist die KERR-Elliptizität der Imaginär- und der KERR-Winkel der Realteil dieser Größe.

Da für weitere Untersuchungen linear polarisiertes Licht senkrecht (s) und parallel (p) zur

Einfallsebene des Lichts (POI: plane of incidence) verwendet wird, betrachten wir den komplexen KERR-Winkel für diese beiden Polarisationen. Für kleine Winkeländerung Θ_K (hier in mdeg) kann man den komplexen KERR-Winkel mit der Gleichung

$$\Phi_K^s = -\frac{r_{ps}}{r_{ss}} \tag{2.3}$$

$$\Phi_K^p = \frac{r_{sp}}{r_{pp}} \tag{2.4}$$

beschreiben[14]. Die Reflextionskoeffizienten r_{sp}, r_{pp}, r_{ps}, r_{ss} sind Elemente der Reflextionsmatrix $\hat{R}$ der Probe. Diese wirkt auf die polarisation des einfallenden Lichts (bzw. Lichtvektor) wie eine Drehmatrix und lautet

$$\hat{R} = \begin{pmatrix} r_{ss} & r_{sp} \\ r_{ps} & r_{pp} \end{pmatrix} . \tag{2.5}$$

Ihre Elemente sind materialabhängig und hängen mit den Nichtdiagonalelementen[2] des Dielektrizitätstensors

$$\hat{\epsilon} = \begin{pmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{pmatrix} \tag{2.6}$$

des Materials zusammen.

Die Nichtdiagonalelemente des Dielektriezitätstensors $\hat{\epsilon}$ sind in der 1.Ordnung proportional zu der magnetischen Flussdichte B. Diese lässt sich beschreiben als

$$\vec{B} = \mu_0(\vec{H} + \vec{M}) \tag{2.7}$$

$$= \mu_0(\vec{H} + \chi\vec{H}) \tag{2.8}$$

$$= \mu_0(1 + \chi)\vec{H} \ . \tag{2.9}$$

Die magnetische Suszeptibilität χ ist bei einem Ferromagneten so hoch, dass B $\propto$ M

angenommen werden kann. Die Magnetisierung $\vec{M}$ kann dann mit Hilfe des komplexen KERR-Winkels als

$$\Phi_K = -\frac{r_{ps}}{r_{ss}}(bzw.\frac{r_{sp}}{r_{pp}}) \propto \epsilon_{ij(i\neq j)} \propto \vec{B} \propto \vec{M} \tag{2.10}$$

beschrieben werden.

2.2 Die verschiedenen Arten des MOKE

Beim MOKE unterscheidet man drei verschiedene Arten des Effekts, die in Abbildung 2.3 dargestellt sind.

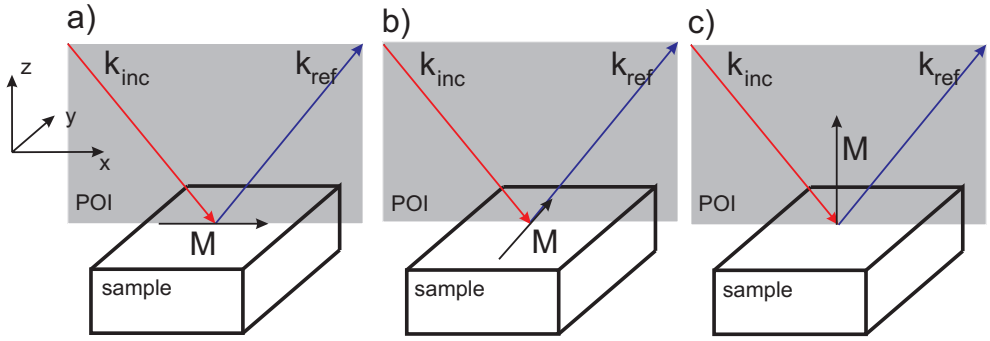


Abbildung 2.3: Drei verschiedene MOKE-Arten. ***a)*** Longitudinaler MOKE mit $\vec{M}$ parallel zur POI (plane of incidence) und Probenoberfläche. ***b)*** Transversaler MOKE mit $\vec{M}$ senkrecht zur POI und parallel zur Probenoberfläche. ***c)*** Polarer MOKE, hier steht $\vec{M}$ senkrecht zur Probenoberfläche und parallel zur POI.

Liegt der Magnetisierungsvektor $\vec{M}$ parallel zu der Einfallsebene des Lichts (plane of incidence, kurz: POI) und der Oberfläche der Probe, so spricht man von longitudinalem magnetooptischen KERR-Effekt (LMOKE, Abbildung 2.3 a)). Dieser verursacht eine Änderung der Elliptizität und eine Drehung der Polarisation des Lichts. Steht der Magnetisierungsvektor $\vec{M}$ parallel zur Probenoberfläche aber senkrecht zur POI, dann handelt es sich um transversalen magnetooptischen KERR-Effekt (TMOKE, Abbildung 2.3 b)). Der

TMOKE verursacht eine Intensitätsänderung des Lichts. Ist $\vec{M}$ senkrecht zur Probenoberfläche und damit parrallel zur POI ausgerichtet, bezeichnet man diesen als polaren magnetooptischen KERR-Effekt (PMOKE, Abbildung 2.3 c)). Der TMOKE verändert die Elliptizität und verursacht eine Drehung der Polarisationsrichtung.
Für weitere Untersuchungen wird ein kartesisches Koordinatensystem definiert, so dass die drei Komponenten des Magnetisierungsvektors $\vec{M}$ jeweils für eine MOKE-Art verantwortlich sind. Dabei verursacht die m_x-Komponente den LMOKE, die m_y-Komponente den TMOKE und die m_z-Komponente den PMOKE (vgl. Koordinatensystem in Abbildung 2.3).

2.3 Trennung von LinMOKE und QMOKE

Eine aufgenommene Magnetisierungskurve wird für weitere Untersuchungen in den quadratischen und linearen MOKE zerlegt. Der lineare Anteil wird dann mit Hilfe der Vekor-Moke-Technik weiter untersucht. Dies ist erforderlich, falls asymmetrische Magnetisierungskurven und damit quadratische MOKE Anteile auftreten. Eine Magnetisierungskurve kann nach [3] mit den Gleichungen

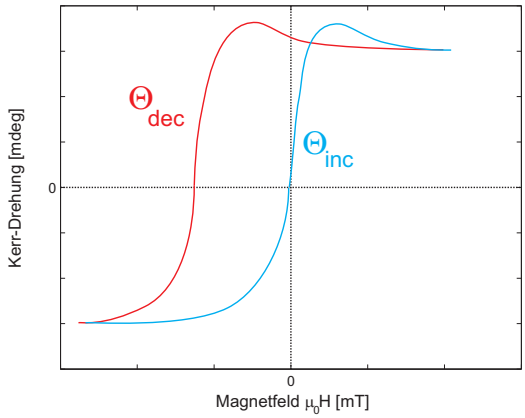


Abbildung 2.4: Eine typische asymmetrische Magnetisierungskurve. Θ_{inc}: der aufsteigende Ast der Kurve. Θ_{dec}: der abfallende Ast der Kurve.

$$\Theta_{sym} = \frac{\Theta_{inc}(H) - \Theta_{dec}(-H)}{2} \tag{2.11}$$

$$\Theta_{asym} = \frac{\Theta_{inc}(H) + \Theta_{dec}(-H)}{2} \tag{2.12}$$

in ihren symmetrischen Anteil Θ_{sym} und ihren asymmetrischen Anteil Θ_{asym} zerlegt werden. Der symmetrische Anteil ist der lineare MOKE (LinMOKE), während der asymetrische Anteil dem quadratischen MOKE entspricht. Dabei bezeichnet Θ_{dec} den abfallenden und Θ_{inc} den aufsteigenden Ast der Magnetisierungskurve. Hier werden die KERR-Drehungen bei gleicher magnetischer Feldstärke unterschiedlichen Vorzeichens verrechnet und man separiert LinMOKE und QMOKE.

2.4 Vektor-MOKE

Für den linearen Anteil des MOKE gilt

$$\Phi_{K,lin}^{s/p} = \pm B_{s/p} K M_x - A_{s/p} K M_z. \tag{2.13}$$

Mit der Gleichung 2.13 und unter Vernachlässigung experimenteller Fehler, kann für die Koeffiziennten $A_{s/p}$ und $B_{s/p}$ angenommen werden, dass

$$B_s = B_p = B \tag{2.14}$$

$$A_s = A_p = A \tag{2.15}$$

gilt und nach [12] können die Komponenten m_x und m_z mit

$$\Theta_{m_x} = \frac{\Theta_s - \Theta_p}{2} \tag{2.16}$$

$$\Theta_{m_z} = -\frac{\Theta_s + \Theta_p}{2} \tag{2.17}$$

berechnen. Dann folgt für m_z und m_x

$$\Theta_{m_z} = A K m_z \tag{2.18}$$

$$\Theta_{m_x} = B K m_x. \tag{2.19}$$

Da nur der LMOKE und PMOKE eine KERR-Drehung verursachen, können nur diese von der Messanlage detektiert werden. Um die m_y-Komponente bestimmen zu können, wird der Magnet und die Probe um 90° in die selbe Richtung gedreht. Damit ist die Anlage im Stande den TMOKE zu detektieren. Die Bestimmung verläuft analog zu der m_x-Komponente. Insgesamt werden vier Messungen benötigt um alle Komponenten des einer Magnetisierung zu beschreiben. Dabei muss mit parallel und senkrecht polarisier-

tem Licht gemessen werden bei sekrechter und paralleler Magnetstellung.
Die gesamte Herleitung kann nachgelesen werden in [6] und [8].

2.5 Die Magnetisierungskurve

Eine typische Magnetisierungskurve ist in Abbildung 2.5 dargestellt. Es ist eine idealisierte Magnetisierungskurve und hat die Form einer Hysterese.

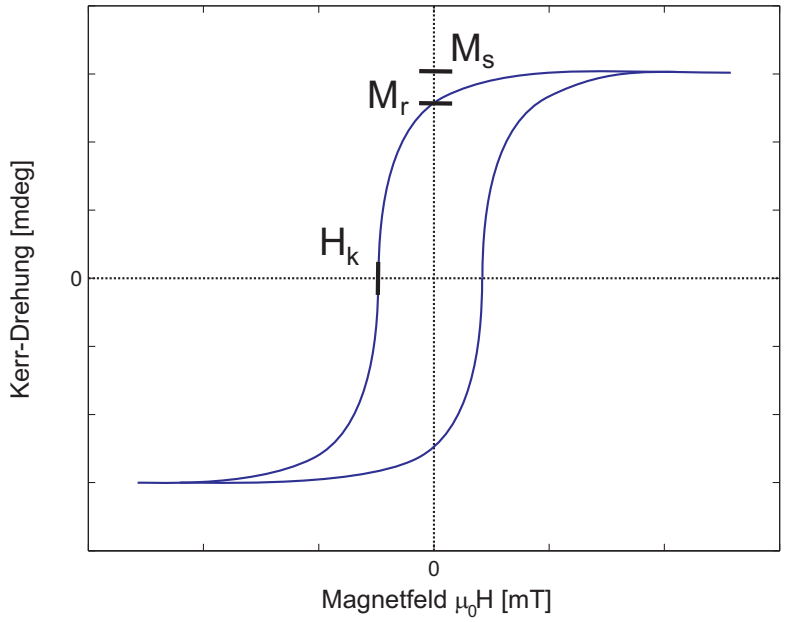


Abbildung 2.5: Idealisierte Skizze einer typischen Magnetisierungskurve. M_s: Magnetisierung in der Sättigung. M_r: Remanenz (Restmagnetisierung bei magnetischer Feldstärke 0). H_k: Koerzitivfeld (magnetische Feldstärke, bei der $\vec{M} = 0$).

Zu den wichtigen Größen der Hysterese gehören die Remanenz M_r, die Sättigungsmagnetisierung M_s und das Koerzitivfeld H_k. Als Remanenz M_r wird die Magnetisierung bei einer magnetischen Feldstärke gleich null bezeichnet, nachdem ein Magnetfeld auf die Probe gewirkt hat. Die Sättigungsmagnetisierung M_s bezeichnet einen Zustand, bei dem die Magnetisierung konstant bleibt trotz weiterer Erhöhung der Magnetfeldstärke. Alle magnetischen Momente in der Probe sind dann parallel zum äußeren magnetischen Feld ausgerichtet. Das Koerzitivfeld H_k ist die Magnetfeldstärke, die benötigt wird, um

eine schon vorhandene Magnetisierung der Probe auf null zu bringen.

2.6 Das Materialsystem Fe/MgO(001)

Als Substrat für die Untersuchung von Eisenschichten werden Magnesiumoxidsubstrate benutzt. Diese sind so präpariert, dass Eisen auf der (001)-Oberfläche des Substrats wächst. Die Gitterkonstante von MgO beträgt 4,212 Å und es liegt in der NaCL-Struktur vor. Das Magnesiumoxid ist zudem diamagnetisch, was bei dieser Substanz bedeutet, dass sie keinen MOKE hervorruft. Zudem ist MgO für die am MOKE zur Untersuchung benutzten Lichtfrequenzen durchlässig.

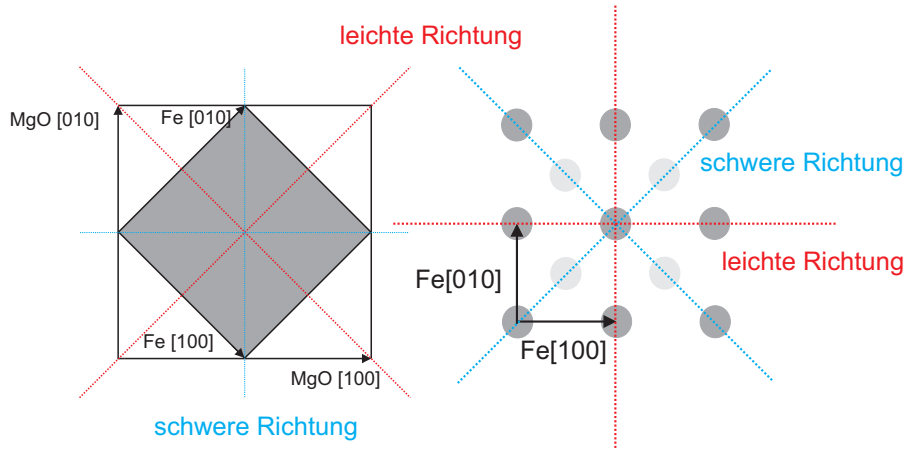


Abbildung 2.6: Links: Eisen auf MgO um 45° versetzt. Rechts: Eisen Atom mit gekennzeichneter magnetisch schwerer und leichter Richtung.

Das Eisen wird auf das MgO aufgedampft. Dabei wächst es in einer α-Fe Struktur (bcc Gitter) um 45° versetzt auf dem Magnesium auf (Abbildung 2.6). Mit einer Gitterkonstanten von 2,866 Å beträgt die Gitterfehlanpssung gegenüber MgO bei diesem Wachstum ca. 3,5 %. Eisen ist ferromagnetisch, weshalb ein MOKE beobachtet werden kann. Seine Curie-Temperatur liegt bei ca. 770°C. Die magnetisch leichten Richtungen liegen bei der Struktur des Eisens entlang der [100], [010], [100] und [010] Richtungen und die magnetisch schweren Richtungen entlang der [110], [1$\bar{1}$0], [$\bar{1}$10] und [$\bar{1}\bar{1}$0] Richtungen. Mit magnetisch leichter Richtung bezeichnet man eine Raumrichtung, in die sich

die Magnetisierung bevorzugt ausbildet. In eine magnetisch schwere Richtung bildet sich die Megnetisierung nur unter dem Einfluß eines äußeren Magnetfeldes aus.

Um eine Verunreinigung mit der umgebenden Luft und ein Durchoxidieren der Eisenschicht zu verhindern wird bei der Probenherstellung die Oberfläche der Probe mit Silizium versiegelt.

3 Messaufbau

In diesem Kapitel werden die experimentellen Begebenheiten an der MOKE-Anlage in der AG `Dünne Schichten und Grenzflächen` beschrieben. Dabei wird der Messaufbau der Anlage in Kapitel 3.1 vorgestellt. Kapitel 3.2 beschäftigt sich mit möglichen Problemen, die sich bei diesem Messaufbau ergeben könnten im Bezug auf das Erstellen einer Software. Am Ende der experimenteller Beschreibungen wird in Kapitel 3.3 das Prinzip der Kalibrierung des Messsignals erläutert.

3.1 Aufbau allgemein

Zur Vermessung des MOKE in der AG `Dünne Schichten und Grenzflächen` an der Universität Osnabrück wird ein Helium-Neon-Laser mit einer Wellenlänge $\lambda = 632{,}8$ nm benutzt. Das Laserlicht fällt auf die Probenoberfläche unter einem Winkel von 45°. Die Probe selbst befindet sich zwischen den Polschuhen eines Magneten, mit dem sie einem äußeren Magnetfeld ausgesetzt wird. Die senkrechte bzw. parallele Polarisation des Lichts wird mit Hilfe eines Polarisators erreicht, vor dem Polarisator ist eine $\frac{\lambda}{2}$-Platte aufgebaut mit der man die Lichtintensität regeln kann,beide Komponenten befinden sich zwischen dem Laser under der Probe. Die Intensität des Lichts kann dabei von der werden. Nach der Reflexion von der Probenoberfläche durchläuft das Licht ein System aus photoelastischem Modulator (PEM) und Analysator bis es auf eine Photodiode trifft. Vor diese ist ein Rotfilter angebracht, um das detektierte Licht auf die passende Wellenlänge zu beschränken.
Mit Hilfe des photoelastischen Modulators wird das reflektierte Licht mit einer Modulationsfrequenz f moduliert, was eine periodische Änderung der Lichtpolarisation bewirkt. Die Modulationsfrequenz f wird gleichzeitig an den Lock-In-Verstärker weitergegeben und dient diesem als Referenzfrequenz. Das so modulierte Licht geht dann durch den Analysator, welcher das Polarisationssignal in ein Intensitätssignal umwandelt und somit an der Lichtdiode ein Intensitätssignal gemessen wird. Aus diesem Signal kann der Lock-In-Verstärker mit Hilfe der doppelten Referenzfrequenz ($2f$) die Kerr-Drehung herausfiltern [6].
Am Probenhalter ist zusätzlich ein Motor und ein Winkelmesser angebracht, der es ermöglicht die Probe um einen beliebigen Winkel zu drehen (Abbildung 3.1). Direkt vor der Probe ist eine Hallsonde, mit der man die angelegte Magnetfeldstärke messen kann. Die ganze Messapparatur kann über das Ansteuerungsgerät M76 angesteuert werden.

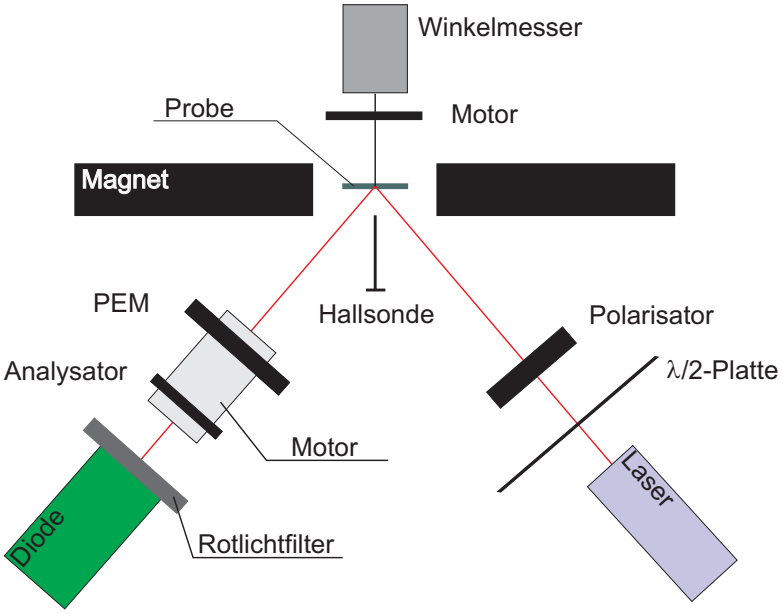


Abbildung 3.1: Aufbau einer magneto-optischer Messanlage.

Das M76 ist über USB-Anschlüsse an einen Rechner angeschlossen und kann mit Hilfe einer Emulationssoftware wie über eine RS232-Schnittstelle (COM/LTP) angesprochen werden. Ansteuerbar sind damit der Motor der Probe, der Motor des PEMs, der Magnet (mit seiner eigenen Steuereinheit) und die Hallsonde. Der an den Probenmotor angebrachte Winkelmesser ist direkt mit dem PC über die RS232-Schnittstelle verbunden.

3.2 Geräte und Verbindung mit dem PC

Wie in Kap. 3.1 beschrieben, sind alle Geräte an das M76 angeschlossen. Dabei ist es wichig zu wissen, dass fast alle Befehle über eine einzige der drei zur Verfügung stehenden Anschlüße ausgeführt bzw. an das M76 übermittelt werden können. Die Ausnahme bildet dabei das Auslesen des Winkels im Winkelmesser und das Auslesen des Messbereichs der Hallsonde.
Bei dieser Anordnung ensteht ein Engpass für die Ansteuerung und das Auslesen der Daten, weil über die RS232-Schnittstelle gleichzeitig nur ein Befehl verschickt werden kann. Dies führt zu Problemen, sobald man gleichzeitig zwei Geräte ansteuern möchte. Zum Beispiel ist es nicht möglich, bei dem Magneten die Magnetfeldstärke zuerhöhen und gleichzeitig mit der Hallsonde die magnetische Feldstärke auszulesen.

3.3 Kalibrierung des gemessenen Signals

Um Messungen von verschiedenen Proben untereinander vergleichen zu können, ist ein absolter Wert des KERR-Winkels notwendig. Damit man diesen bekommt, muss das Messsignal kalibriert werden. Das Prinzip der Kalibrierung ist in den Abbildungen 3.2 und 3.3 skizziert und wird nun erläutert.

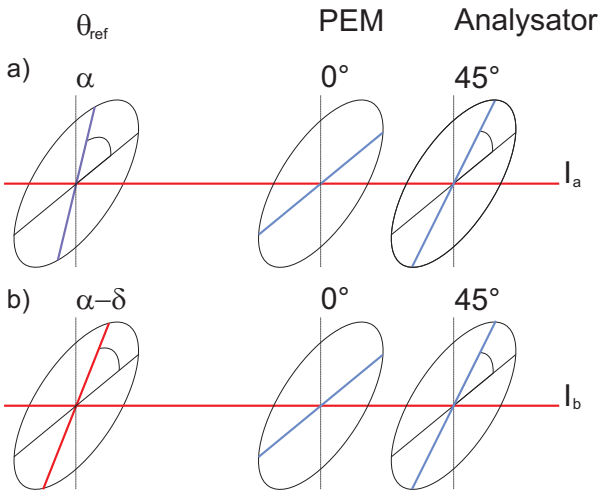


Abbildung 3.2: Kalibrierung des Signals Teil 1. a) Licht bei einer Polarisationsrichtung α geht durch den PEM und Analysator. Dabei wird eine Intensität I_a gemessen. b) Die Polarisationsrichtung ändert sich um einen Winkel $-\delta$. Dabei wird die Intensität I_b gemessen.

Das reflektierete Licht mit einer Polarisationsrichtung α geht durch den PEM und Analysator durch. An der Diode wird eine Intensität I_a gemessen (Abbildung 3.2a)). Findet eine Drehung der Polarisation des Lichts um den Winkel $-\delta$ statt, so wird an der Diode die Intensität I_b gemessen (Abbildung 3.2b)).

Wird bei der Polarisation $\alpha - \delta$ das System aus PEM und Analysator um $-\delta$ gedreht, so misst man wieder die Lichtintensität I_a (Abbildung 3.3a)). Dreht man wiederum den PEM und Analysator um den Winkel $+\delta$ bei einer Polarisationsrichtung des Lichts wie aus Abbildung 3.2a), dann wird an der Diode die Intensität I_b gemessen (Abbildung 3.3b)).

Diese Beispiele verdeutlichen, dass eine KERR-Drehung der Polarisation wie in Abbil-

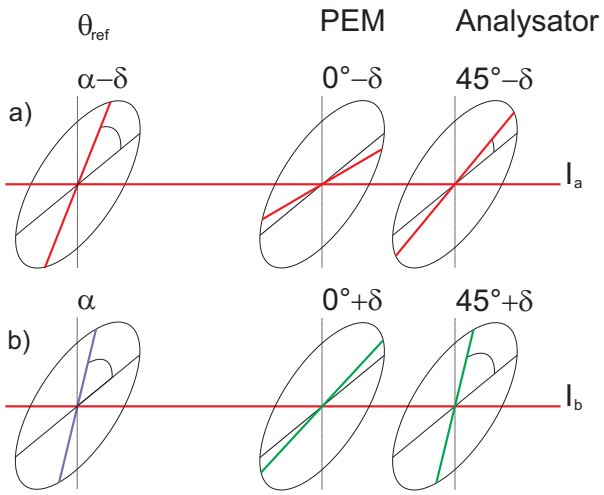


Abbildung 3.3: Kalibrierung des Signals Teil 2. a) Bei der Polarisation $\alpha - \delta$ wird das PEM mit dem Analysator um den Winkel $-\delta$ gedreht. Daraus resultiert eine Intensität I_a. b) Bei der Polarisation α wird das PEM und der Analysator um den Winkel $+\delta$ gedreht. Die intensität I_b wird gemessen.

dung 3.2b) simuliert werden kann, indem man stattdessen PEM und Analysator dreht wie in Abbildung 3.3b)

Misst man also verschiedene Intensitäten I_1 und I_2 bei zwei verschiedenen Winkeln α_1 und α_2 von PEM und Analysator, so ist dies, als wenn man zwei unterschiedliche KERR-Drehungen vorliegen hat. Bildet man den Quotienten $\frac{\alpha_1 - \alpha_2}{I_1 - I_2}$, so bekommt man einen Skalierungsfaktor, mit dem es möglich ist einen absoluten KERR-Winkel aus der Intensitätsänderung zu bestimmen.

Der PEM und der Analysator sind dazu an einen Motor angeschlossen. Dieser ist im Stande den PEM und den Analysator für eine Kalibrierung um einen beliebigen Winkel zu drehen. Man hat festgestellt, dass dieser Motor bei schweren Lasten, bei großen Winkeln oder zu schneller Fahrt ungenau arbeitet. Deshalb wird bei der Kalibrierung PEM und Analysator nur über einen Winkelbereich von 200 mdeg in relativ langsamen Schritten gedreht. Es ist sehr wichtig, dass der Motor zuverlässig und sehr genau arbeitet, damit der Kalibrierungsfaktor so exakt wie möglich ist. Denn eine typische KERR-Drehung beträgt nur wenige mdeg. Das gesamte Prinzip der Kalibrierung und Signalermittlung kann in [6] und [8] genauer nachgelesen werden.

4 Vorüberlegung

In diesem Kapitel sind die allgemeinen Überlegungen aus der Enwicklung der Ansteuerung wieder gegeben. In Kapitel 4.1 wird die Wahl der Programmiersprache erörtert. Auf welche Art die Geräte angesteuert werden sollten, wird in Kapitel 4.2 besprochen. Ziele und Anforderungen an die Ansteuerungssoftware werden in Kapitel 4.3 wiedergegeben.

4.1 Sprache

Als erstes stellt sich die Frage, welche der verfügbaren Programmiersprachen sich am besten eignen, um eine Ansteuerung der Messvorrichtung zu realisieren. Das zum Anfang dieser Arbeit benutzte Programm zur Messung des MOKE war in MATLAB geschrieben und verfügte über keine GUI (Graphical User Interface). MATLAB ist keine richtige Programmiersprache, sondern lediglich eine Software, die hauptsächlich für nummerische Berechnungen entwickelt wurde. Deshalb wurde eine andere Programmiersprache gesucht.
Eine der Anforderungen für die Ansteuerungssoftware ist eine leichte Erweiterbarkeit und Wartbarkeit. Das führt zur der Überlegung, dass sich eine Sprache dazu am besten eignen würde, die das Konzept der Objektorientierung verinnerlicht. Damit fällt die Sprache C aus der engeren Auswahl raus, weil diese für eine prozedurale Programmierung konzipiert wurde. In engere Auswahl kommen die Sprachen C++ und Java. Die Sprache C++ unterstützt das Paradigma der objektorientierten Programmierung und eignet sich gut zur Realisierung einer Ansteuerung. Die Wahl der Programiersprache fiel aber schließlich auf Java. Für Java sprechen folgende Gründe. Es ist eine objektorientierte und eine recht einfach zu erlerndende Programmiersprache, was eine spätere Modifikation der Ansteuerungssoftware durch Dritte wesentlich erleichtert. Zudem sind mit den Java Bibliotheken die wichtigsten Komponenten zu Realisierung einer GUI mitgeliefert. Darüber hinaus wurde schon in der AG `Dünne Schichten und Grenzflächen` eine Software zur Analyse von magnetooptischen Daten von Robin Schubert in Java entwickelt[7]. Es entsprach deshalb dem Wunsch der Mitarbeiter, die am MOKE gearbeitet haben, die Programmiersprache konsistent zu halten.

4.2 sequentielle vs. parallele Ausführung

Mit Java ist es möglich ein Programmablauf zu parallelisieren. Die Parallelisierung kann in Java mit Threads bzw. mit Klassen, die das Interface Runable implementieren, rea-

lisiert werden. Damit kann ein Programm so gestaltet werden, dass möglichst viele auszuführende Aufgaben und Berechnungen parallel (bzw. bei Single-Core-Prozessoren quasi-parallel) ausgeführt werden. Damit ist es möglich einen Programmablauf so zu beschleunigen, dass eine Berechnung nur ein Bruchteil der Zeit dauert, als dies bei einem sequenziellen Programmablauf der Fall wäre.
Parallelisierung von Programmabschnitten hat aber zwei wesentliche Nachteile in der Entwicklung einer Software. Zum einem ist es oft so, dass die parallelisierten Abschnitte (sog. Threads) oft gemeinsame Ressourcen nutzen. Es muss dann sichergestellt werden, dass der Zugriff auf diese Ressourcen in einer vom Programmierer gewollten Reihenfolge geschieht. Dies kann schon bei einer kleinen Anzahl von Threads sehr kompliziert werden. Zum anderen ist die Gefahr für die Entstehung sogenannter Deadlocks sehr groß. Als Deadlock wird der Zustand eines Programms bezeichnet, bei dem Threads auf Ressourcen warten, die von anderen Threads bearbeitet werden, diese wiederum aber umgekehrt auf die Ressourcen der ersten Threads warten, um ihre Ressourcen freizugeben. Dies hat zur Folge, dass das Programm nie zu Ende ausgeführt werden kann. Deadlocks können von dem ausführenden Programm nicht erkannt werden. Sollte sich also ein Deadlock bei der Programmierung einschleichen, ist die Beseitigung sehr aufwendig bzw. nicht durch eine einfache Fehlerbehandlung möglich.
Zieht man die experimentellen Begebenheiten (Kapitel 3.2) in die Überlegung mit ein, so ist die Parallelisierung der Ansteuerung nicht sinnvoll. Dies liegt daran, dass die Messung in einer bestimmten Reihenfolge durchgeführt werden muss. Es bringt zum Beispiel nichts die magnetische Feldstärke auszulesen, während der Motor der Probe gedreht oder das Magnetfeld erhöht wird. Der Zeitgewinn durch eine Parallelisierung wurde daher als gering eingeschätzt, da das Programm durch diese Bedingungen ohnehin quasi sequentiell arbeiten müsste. Eine Parallelisierung war daher mit dem damit verbundenen Programmieraufwand (incl. der Beseitigung von Bugs und Deadlocks) und der dazu zur Verfügung stehenden Zeit nicht mehr als sinnvoll zu betrachten.

4.3 Was sollte das Programm können?

Neben dem Zugriff sollte noch folgendes in der Ansteuerung realisiert werden.

- Das Programm sollte wie oben schon erwähnt über eine GUI verfügen.
- Es sollte unmöglich sein, durch bloße Benutzung der Software die Messgeräte zu beschädigen.

- Die Funktionalität des Matlab Programms sollte übernommen ggf. verbessert werden.
- Die falsche Stellung der Hallsonde sollte vom Programm abgefangen werden.
- Das Programm sollte selbstständig den passenden Sensitivitätsbereich der Hallsonde einstellen.

Die Sicherheit beinhaltet:

- Das Umpolen des Magneten bei angelegtem Magnetfeld durch den User sollte nicht möglich sein.
- Ein Drehen des PEMs und Analysators nur über kleinen Winkelbereich und in relativ langsamen Schritten sollte realisiert werden. Diese Einstellungen sollen dem User nicht zugänglich sein, um exakte Kalibrierungswerte zu garantieren.
- Der Probenmotor sollte nicht abrupt gestoppt werden können.

Bisher war jede Änderung der Messparameter mit der Änderung des Matlab Codes verbunden. Dies war oft sehr umständlich und verursachte viele Probleme durch Fehler beim Ändern des Codes oder ähnliches. Dies sollte jetzt in die GUI ausgelagert werden. Darüber hinaus sollte folgendes realisiert werden:

- Man sollte die Probe um einen beliebigen Winkel drehen können. Diese sollte immer den kürzesten Weg zu der gewünschten Winkelstellung nehmen.
- Die Magnetfeldstärke sollte vom User in mTesla eingestellt werden können.
- Die Kalibrierung des Intensitätssignals des Lichts sollte möglich sein.
- Der User sollte die Möglichkeit haben zu bestimmen, über wie viele Wertepaare das Messsignal gemittelt werden soll.
- Nach der Kalibrierung sollten die Messssignale, über die gemittelt wurde, einsehbar sein.
- Der Skalierungsfaktor sollte vom User eingestellt werden können.
- Das Lock-In-Signal sollte sichtbar sein.
- Das kalibrierte Lock-In-Signal sollte sichtbar sein.
- Die Stellung des Magneten ($\parallel$ oder $\perp$) und des Polarisationsfilters ($\parallel$ oder $\perp$) sollte eingetragen werden können.
- Der Modus, in dem gemessen wird, sollte eingetragen werden können (f oder $2f$).
- Es sollte möglich sein eine Schnellhysterese aufzunehmen, diese zu sehen und zu speichern.

- Es sollte möglich sein die Stellung der Hallsonde zu überprüfen.
- Der Name der Probe sollte für die Messung eingegeben werden können.
- Für die Messung sollte der Winkelbereich einstellbar sein, sowie die Schrittweite.
- Die Magnetfeldstärke, die bei der Messung abgefahren wird, sollte vom User in mTesla eingestellt werden können.
- Alle Einstellungen und Änderungen, die gewünscht sind für eine automatische Messung, sollten in Form eines Formulars eingegeben werden können.

5 Das Programm

In diesem Kapitel wird die Ansteuerungsoftware vorgestellt. Dabei wird in Kapitel 5.1 beschrieben, wie der Zugriff auf die Messapparatur realisiert wurde. In Kapitel 5.2 wird die gesamte Programmstruktur erklärt. Dabei werden alle Java-Klassen vorgestellt, die hier verwendet wurden. Die ersten Erfahrungen mit der Ansteuerungssoftware werden Kapitel 5.3 beschrieben.

5.1 Zugriff auf RS232 über `Comunicator.java`

Bei der hier entwickelten Software wurde der Zugriff auf die Messvorrichtung über die Klasse `Comunicator.java` realisiert. Die Klasse `Comunicator.java` verwendet dazu eine native Bibliothek, die unter dem Namen RxTx und der GNU LGPL (GNU Lesser General Public License) entwickelt wird. RxTx stellt dabei verschiedene Klassen zu Verfügung, die den Zugriff auf Serielle- und Parallelschnittstellen eines Rechners ermöglichen. Bei der im Rahmen dieser Arbeit entwickelten Software ist `Comunicator.java` das Verbindungsglied zwischen den realen Messgeräten und ihren digitalen Pendants, die als Objekte in der Programmstruktur existieren.

5.1.1 Die Klasse Comunicator

Um den Zugriff möglichst schnell zu machen und die Kommunikation an dem in Kapitel 3.2 beschriebenen Engpass zu organisieren, beschränkt sich diese Klasse nur auf die Übermittlung der gesendeten Befehle und der Antwort auf diese. Ob ein Befehl von dem Gerät angenommen werden kann oder eine Antwort, die zurückgeliefert wurde, die richtige ist, muss ausserhalb dieser Klasse überprüft werden.
`Comunicator.java` stellt lediglich folgende zwei Methoden und einen Konstruktor zur Verfügung.

- `Comunicator()` (Konstruktor)
- `sendOrder(String order, String comPort, int awaitingAnswerLength)`
- `plzDie()`

Die restlichen Methoden in dieser Klasse sind als `private` deklariert und daher von außen nicht sichtbar. Sie dienen hauptsächlich dazu den Code in der Klasse selbst übersichtlicher und einfacher zu gestalten und sollten von außen nicht erreichbar sein.

Der Konstruktor Comunicator(), der beim Erzeugen der Klasse Comunicator.java mittels **new** aufgerufen wird, initialisiert alle Variablen. Außerdem öffnet und konfiguriert er alle COM-Ports, die für die Ansteuerung benötigt werden.
Die Methode plzDie() schließt alle geöffneten Ports und sollte aufgerufen werden, sobald die Ports nicht mehr benötigt werden. Die Methode sendOrder(String order, String comPort, int awaitingAnswerLength) ist für die Kommunikation zwischen dem Rechner und den Messgeräten zuständig und wird daher etwas ausführlicher im Unterkapitel 5.1.2 beschrieben.

5.1.2 Methode sendOrder(String order, String comPort, int awaitingAnswerLength)

Die methode protected byte[] sendOrder(String order, String comPort, int awaitingAnswerLength) ermöglicht das Senden der Befehle an die Messgeräte. Das erste Argument ist der versendete Befehl, der als ein String vorliegen muss. Hierbei wird das zum Beenden eines Befehls nötige CR (carriage return) nicht übergeben. Das zweite Argument bezeichnet die Schnittstelle, über die der Befehl versendet werden soll (z.B. COM1). Dies muss erneut ein String sein. Das dritte Argument ist die Länge der Antwort, die erwartet wird. Diese wird als eine Integer Zahl übergeben, die die Anzahl der Bytes benennt.
Die Antwort, die von den Messgeräten zurück kommt, ist in ASCII (American Standard Code for Information Interchange) codiert und wird als ein Byte-Array zurückgeliefert. Wird die Methode sendOrder(order, comPort, awaitingAnswerLength) aufgerufen, so wird zunächst überprüft, über welchen COM-Port die Nachricht gesendet werden soll. Dieser wird dann einem Objekt vom Typ SerialPort zugeordnet. Mit Hilfe verschiedener Methoden, die als private deklariert sind, wird ein OutputStream für den entsprechenden COM-Port eingerichtet. Der Befehl kann mittels der Methode write(byte[] argument) des OutputStreams verschickt werden, sobald dieser in ein Byte-Array umgewandelt wurde. Hierbei wird das CR, das in ASCII als eine dreizehn codiert ist, automatisch von Comunicator.java ans Befehlsende angehängt. Wurde der Befehl verschickt, wartet Comunicator.java auf eine Antwort der Geräte mittels serialPortEventListener. Der serialPortEventListener ist als eine innere Klasse von Comunicator.java deklariert und implementiert das Interface SerialPortEventListener, bei dem die Methode public void serialEvent(SerialPortEvent event) implementiert wurde.
Dabei wird das Warten auf das SerialPort Objekt (COM-Port) synchronisiert, über des-

sen Datenstrom (OutputStream) der Befehl gesendet wurde und mit `tmpPort.wait(2500)` gestartet (Java-Code 1).

```
1  synchronized(tmpPort)
2  {
3      try {
4          tmpPort.wait(2500);
5      } catch (InterruptedException e) {
6          System.out.println("Irgendetwas unterbricht das
               Warten");
7          e.printStackTrace();
8      }
9
10 }
```

Java-Code 1: Synchronisierung auf das SerialPort Objekt (tmpPort), über den der Befehl gesendet wurde.

Die Wartezeit ist auf 2500 ms festgelegt. Dies ist nötig, um das Blockieren des Programms zu verhindern, falls keine bzw. eine zu kurze Antwort von den Messgeräten geliefert wird.

Ist eine Antwort des Messgeräts verfügbar, so wird ein SerialPortEvent ausgelöst und der `serialPortEventListener` arbeitet seine `serialEvent(SerialPortEvent event)`-Methode ab.

```
2  if("//./COM3".contentEquals(event.getSource().toString()))
3  {
4      Comunicator.outDataM76=recieveData(M76);
5      if(Comunicator.outDataM76[answerLength]!=0)
6      synchronized (M76)
7      {
8          answerLength=0;
9          M76.notify();
10     }
11 }
```

Java-Code 2: Ausschnitt aus `serialEvent(SerialPortEvent event)`-Methode. Die Nachricht wird mit der Methode `recieveData(M76)` empfangen und in das Datenfeld `outDataM76` der Klasse `Comunicator.java` geschrieben.

Der Java-Code 2 stellt den wesentlichen Teil der Methode `serialEvent` dar. Nachdem festgestellt wurde, an welchem Port das SerialPortEvent ausgelöst wurde (Zeile 2), wird die Nachricht mit der Methode `recieveData (M76)` empfangen und in einem Byte-Array gespeichert. Danach wird eine Referenz auf das Array dem Datenfeld `outDataM76` der Klasse `Comunicator.java` übergeben (Zeile 4). Eine if-Abfrage in Zeile 5 überprüft, ob die Nachricht mindestens die erwartete Länge besitzt. Ist dies der Fall, wird die Klasse `Comunicator.java` mittels `M76.notify()` benachrichtigt, dass sie nicht weiter warten soll (Zeile 9).
Wenn dies geschieht oder aber 2500 ms Wartezeit vorbei sind, wird mittels `return` das jeweilige Byte-Array, in dem sich die Antwort befindet, zurück an den Absender des ursprunglichen Befehls zurück geliefert.

5.2 Das Ansteuerungsprogramm

Das Ansteuerungsprogramm besteht im wesentlichen aus zehn Klassen, fünf Hilfsklassen und zwei Interfaces, die auf die Java-Pakete guiMOKE, MOKE, und supportingClasses verteilt sind. Diese werden im folgendem vorgestellt, wobei hier nur die wichtigsten Eingenschaften erwähnt werden.

5.2.1 Package guiMOKE

Für die Erstellung einer GUI bietet Java gleich zwei Bibliotheken. Das AWT (Abstract Window Toolkit) und die Bibliothek Swing. Die Klassen, die von Swing mitgeliefert werden verfügen meist über mehr Funktionalität als die des AWT. Außerdem werden mehr Klassen von Swing zur Verfügung gestellt. Für die Ansteuerungssoftware bietet sich deshalb Swing an, um die Benutzeroberfläche zu implementieren. Diese ist zu dem Paket guiMOKE zusammengefasst und in Abbildung 5.1 zu sehen. Das Paket besteht aus den beiden Klassen `MainFrameMOKE.java` und `MeasurementFrame.java`, die beide von JFrame erben. Es sind die einzigen für den Anwender sichtbaren Klassen. Die Klasse `MainFrameMOKE.java` ist für die direkte Bedienung der Messanlage zuständig und enthält eine Referenz auf das Objekt der Klasse `MOKEMeasurementDevice.java` aus dem Java-

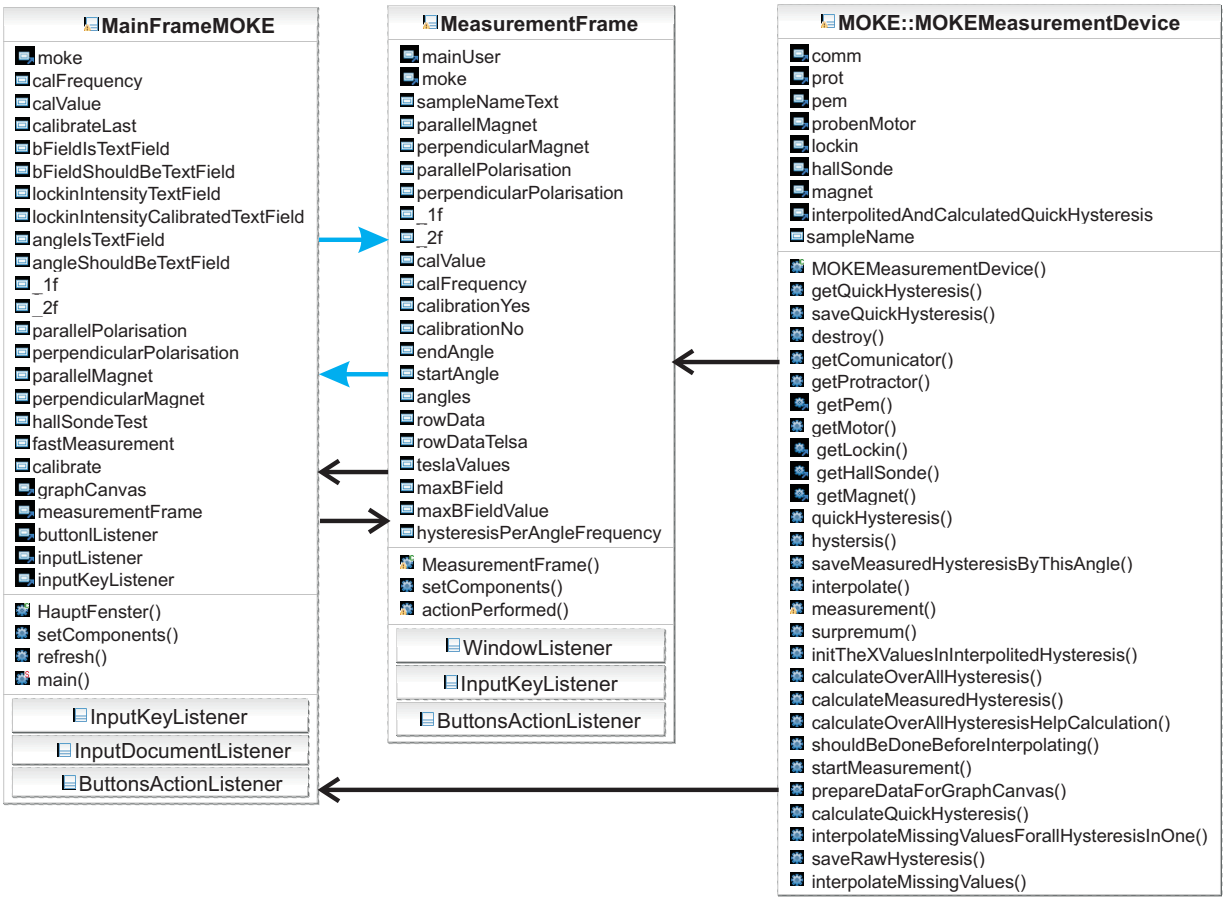


Abbildung 5.1: Das Java-Paket guiMOKE bestehend aus den beiden Klassen `MainFrameMOKE.java` und `MeasurementFrame.java`. Schwarze Pfeile: Die Referenz eines Objektes dieser Klasse ist in den aufzeigenden Klassen enthalten. Blaue Pfeile sollen andeuten, dass es dem Anwender möglich ist zwischen den beiden Klassen hin und her zu wechseln.

Paket MOKE. Mit der Klasse `MeasurementFrame.java` ist es möglich die gewünschten Einstellungen für eine Messung vorzunehmen und diese dann zu starten.

Die Abbildung 5.2 zeigt die Benutzeroberfläche, die von der Klasse `MainFrameMOKE.java` erstellt wird. Rechts ist eine GraphCanvas eingebunden, die Robin Schubert im Rahmen von [7] entwickelt wurde. Diese soll in der GUI die im Schnellmessverfahren aufgenommenen Hysteresen sichtbar machen.

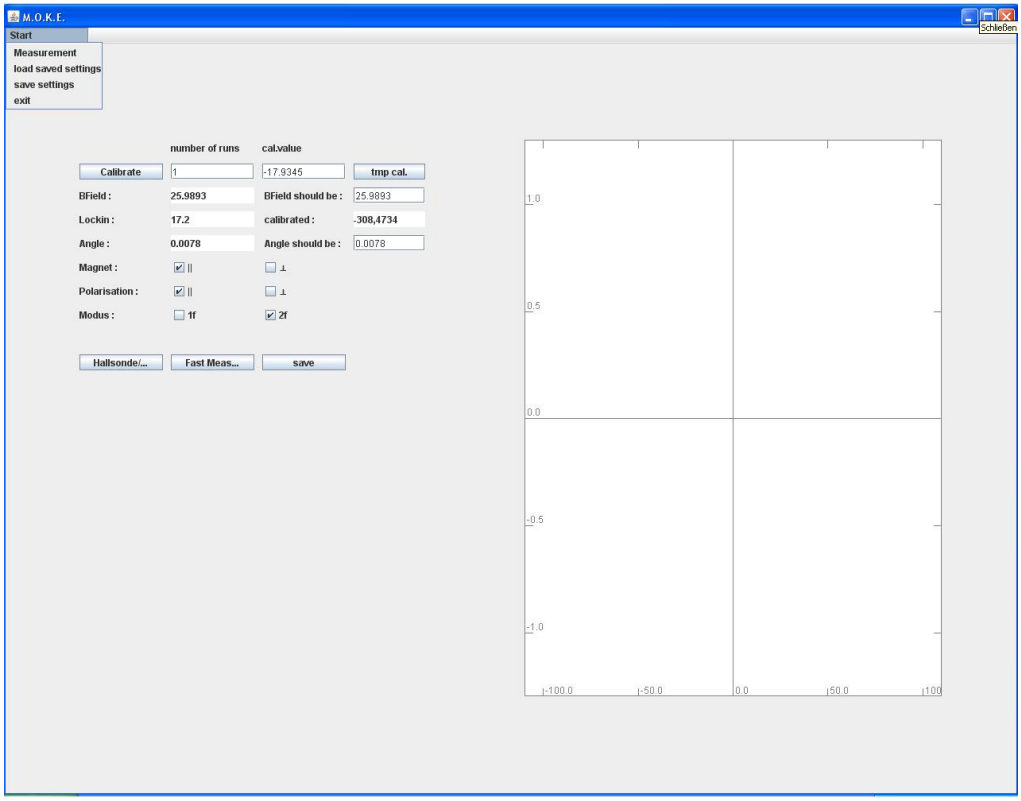


Abbildung 5.2: Die Gesamte GUI für die direkte Bedienung der Messanlage. Weitere Erläuterungen im Text.

In der Abbildung 5.3 sind die Schaltflächen und Eingabefelder der Klasse `MainFrameMOKE.java` nochmal vergößert dargestellt. Es ist dem Anwender möglich mit der Schaltfläche Calibrate die Kalibrierung des Lock-In-Signals vorzunehmen. Rechts daneben befindet sich ein Eingabefeld, mit dem bestimmt werden kann, über wieviele Wertepaare die Kalibrierung vorgenommen werden soll. Dem Anwender ist es auch möglich den Skalierungsfaktor direkt einzugeben (Eingabefeld unter cal.value). Die Schaltfläche mit der Aufschrift `tmp cal.` (temporary calibration values) ermöglicht es die Einzelwerte einzusehen, über die das Kalibrierungssignal ermittelt wurde. Direkt darunter befinden sich die Eingabefelder `BField should be` und `Angle should be`. Mit `BField should be` kann man den Magneten über die Feldstärke steuern. `Angle should be` steuert den Probenmotor. Das Lock-In-Signal wird im Feld `Lockin` angezeigt. Während das kalibrierte Messsignal im Feld `calibrated` zu sehen ist.
Unter `Magnet` ist es möglich die Position des Magneten (parallel/senkrecht) dem Programm mitzuteilen. Diese steht beim Starten der Anwendung auf parallel. Von dieser

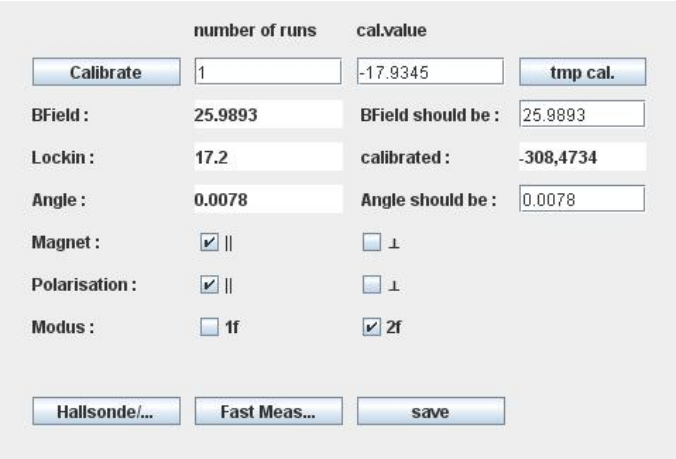


Abbildung 5.3: Eingabefelder aus Abb. 5.1 vergrößert, über die die MOKE-Anlage gesteuert werden kann. Weitere Erläuterungen im Text.

Einstellung hängt auch ab, wie die ausgelesenen Magnetfeldstärken von der Software interpretiert und verrechnet werden. Die Anwendung ist im Stande die Fehlstellung der Hallsonde mit einer Verrechnung zu korrigieren. Ist eines dieser Kontrollkästchen ausgewählt, dann wird der Speichername einer Messung durch den Zusatz `pmag` bzw. `smag` erweitert.
Eine Zeile weiter kann man dem Programm mitteilen, welche Polarisation des Lichtes eingestellt wurde. Wurde hier eine Auswahl getroffen, dann erweitert sich der Speichername einer Messung um `ppol` bzw. `spol`.
Vollständigkeitshalber kann man auch den Messmodus auswählen ($1f$ oder $2f$) (vgl. Kapitel 3.3). Weiter unten sind die Schaltflächen `Fast Measurement` und `save` für eine Schnellmessung und Speicherung angebracht. Beim Speichern wird der Benutzer nach dem Namen der Datei gefragt. Die Datei wird dann in dem Ordner gespeichert, von dem aus die Anwendung ausgeführt wurde. Und ist in dem Ordner mit dem Namen `QuickHysteresis` zu finden. Mit der Schaltfläche `Hallsonde/Test` kann man die Stellung der Hallsonde überprüfen. Ist die Hallsonde falsch eingestellt, so wird das von der Software abgefangen und das Messsignal wird mit -1 multipliziert.

Die GUI besitzt auch eine Menüleiste (Abbildung 5.4). Diese enthält die Schaltflächen `Measurement` und `exit`. Mit `exit` kann man die gesamte Anwendung kontrolliert beenden. Die Schaltfläche `Measurement` schließt das Fenster der normalen Bedienung und öffnet ein Fenster zur Einstellung einer automatischen Messung (Abbildung 5.5). Dabei

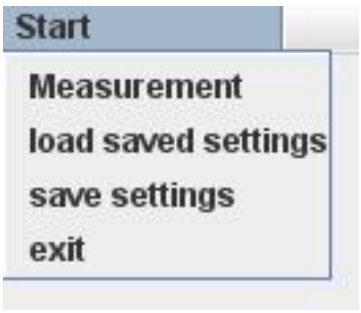


Abbildung 5.4: Die Tabs mit den Schaltflächen `Measurement`, um zu der GUI für eine automatische Messung zu gelangen, und `exit`, um kontrolliert die Anwendung zu beenden.

werden die Einstellungen des Bedienungsfensters übernommen.

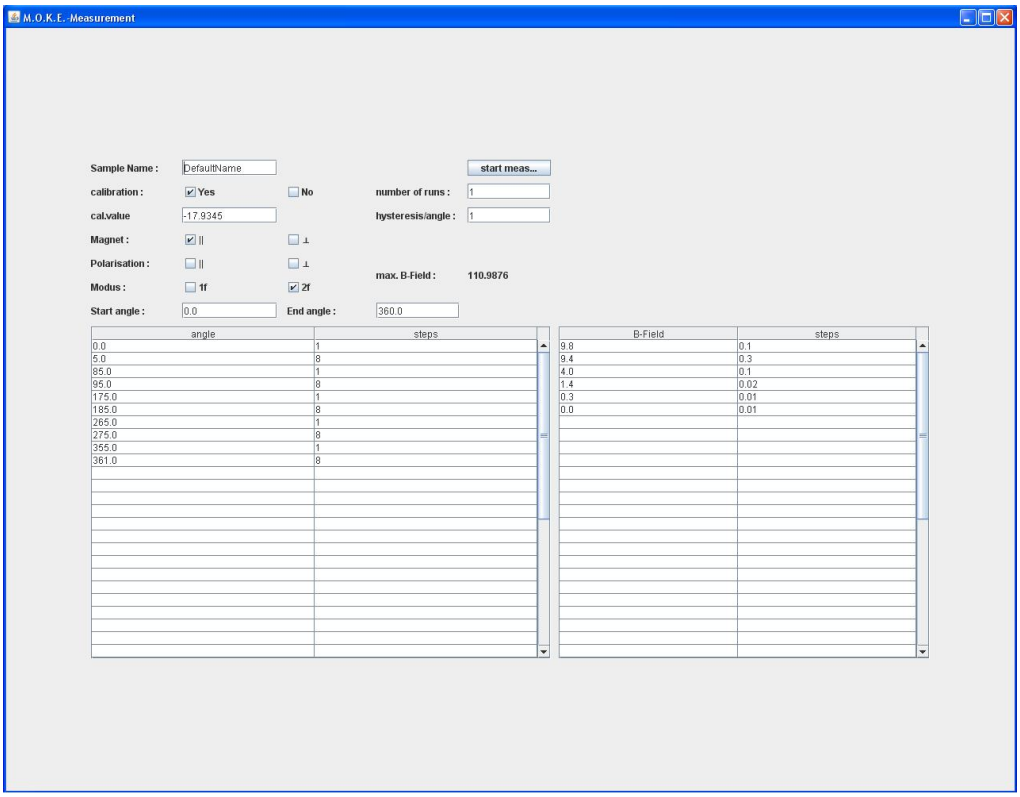


Abbildung 5.5: GUI zur Einstellung der automatischen Messung. Weitere Erläuterungen im Text.

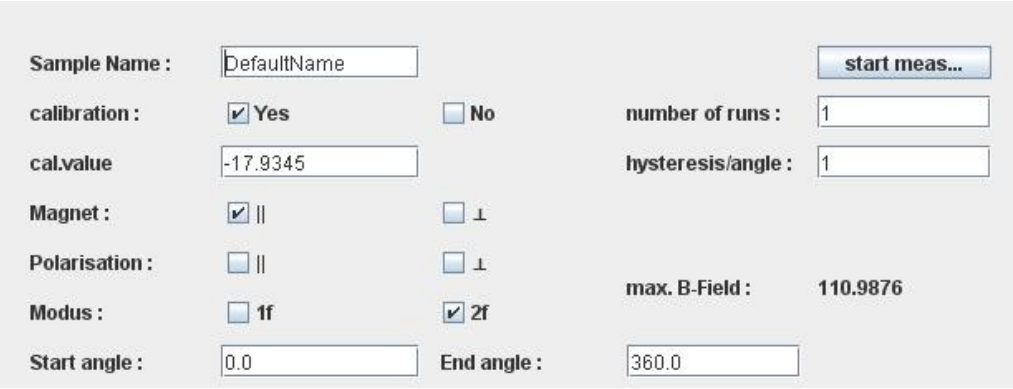
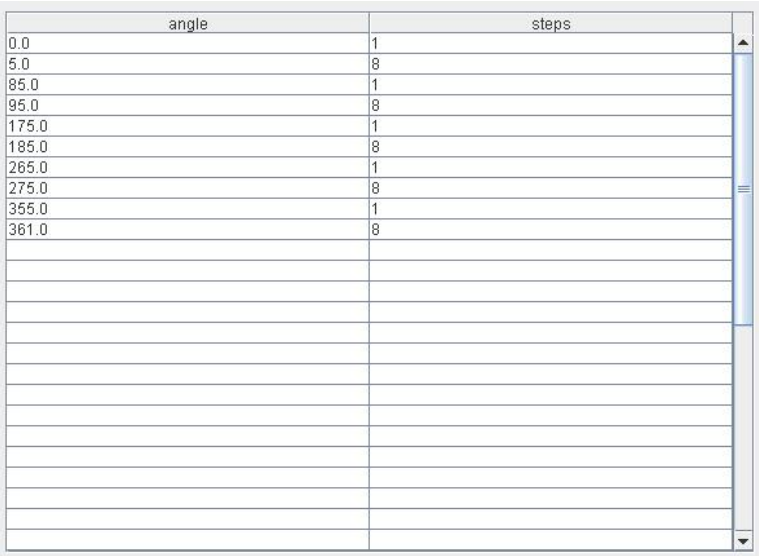


Abbildung 5.6: Schaltflächen zur Einstellung der Messung. Ausschnitt vergrößert aus Abbildung 5.5. Weitere Erläuterungen im Text.

Von diesem Fenster aus ist es dem Anwender nicht mehr möglich das Messssignal direkt zu Kalibrieren, die Magnetfeldstärke zu ändern oder die Probe zu drehen. Es können nur die gewünschten Einstellungen vorgenommen werden, die für eine automatische Messung benötigt werden. In der Zeile `calibration` wird festgelegt, ob eine Kalibrierung des Messsignals vor der Messung und wie genau sie stattfinden soll. Eine Zeile darunter befindet sich das Feld `hysteresis/angle`, in dem angegeben werden kann, wie viele Magnetisierungskurven pro Winkel aufgenommen werden. Unter `Sample Name` kann der Name der Messprobe eingegeben werden, unter dem die Messung gespeichert wird.

Die gewünschten Winkel, bei denen die Magnetisierungskurven aufgenommen werden sollten, können in der unteren linken Tabelle eingegeben werden. Unter `angle` wird zunächst ein Winkelbereich festgelegt. In der Spalte `steps` in welcher Schrittweite dieser Winkelbereich abgemessen werden soll (Abbildung 5.7). In der Tabelle rechts kann der Verlauf des Magnetfeldes festgelegt werden. Die Eingaben werden, im Gegensatz zu dem Matlab Programm, in mT vorgenommen und verlaufen analog zu den Winkeleingaben. Diese müssen nur den positiven Bereich des Magnetfeldes abdecken, da die Software diese selbständig symmetrisiert. Als Orientierungshilfe zeigt `max. B-Field` die maximal erreichbare Magnetfeldstärke an.

Mit der Schaltfläche `start measurement` kann die Messung gestartet werden.



angle	steps
0.0	1
5.0	8
85.0	1
95.0	8
175.0	1
185.0	8
265.0	1
275.0	8
355.0	1
361.0	8

Abbildung 5.7: JTable, in der die Winkelbereiche, bei denen die Magnetisierungskurven aufgenommen werden sollten, eingegeben werden können .

5.2.2 Package MOKE

Das Java-Paket MOKE besteht aus acht Klassen. Die Abbildung 5.8 ist eine vereinfachte Darstellung dieses Paketes und skizziert, in welcher Beziehung die Klassen zueinander stehen. Hier enthalten ist die Klasse `Comunicator.java` die, wie in Kapitel 5.1 schon erwähnt wurde, das Bindeglied zwischen den realen Geräten und ihren digitalen Abbildungen darstellt. Bis auf die Klasse `MOKEMeasurementDevice.java` bilden die restlichen sechs Klassen die Geräte und ihre Funktionen ab. Die Abhängigkeiten der Klassen untereinander sind ähnlich wie die der realen Geräte. Diese sind mit den dickeren schwarzen Pfeilen in der Abbildung 5.8 angedeutet. Dabei enthalten alle instanzierten Objekte dieser Klassen eine Referenz auf das Objekt der Klasse `Comunicator` (dünnere schwarze Pfeile). Die blauen Pfeile geben die Zugriffe der Klassen untereinander über die Methoden wieder.

Die Klasse Protractor

Die Klasse Protractor bildet den Winkelmesser des Probenmotors ab. Sie enthält Methoden, die das Auslesen des Winkels aus dem Gerät ermöglichen (`getCurrentAngle()`). Intern werden die aus dem Gerät ausgelesenen Daten, die als ein Byte-Array ankommen, zunächst in einen String umgewandelt (vgl. Kapitel 5.1.2) und dann als double

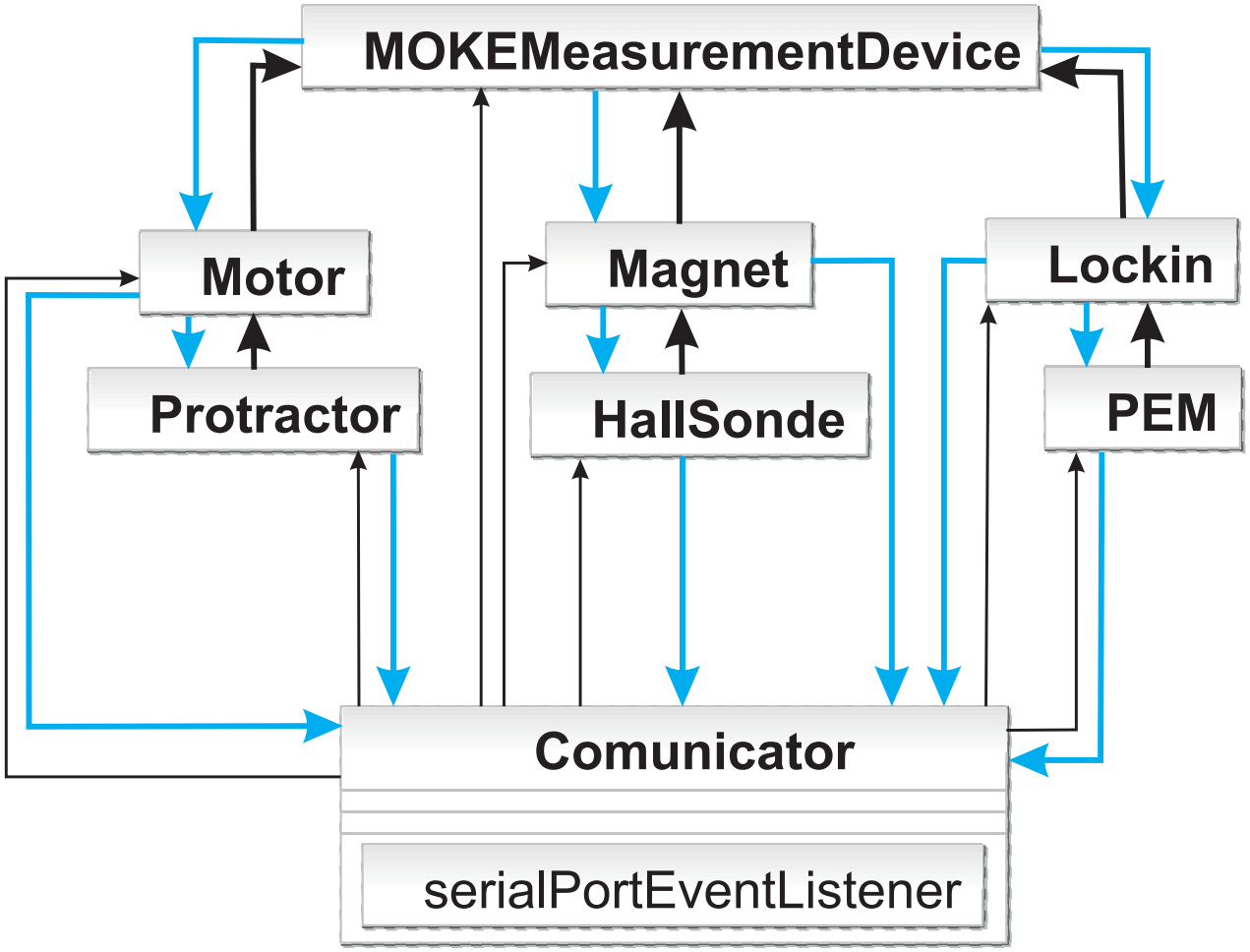


Abbildung 5.8: Vereinfachte Darstellung der Java-Paketes MOKE. Schwarze dünne Pfeile: Eine Referenz dieser Klasse ist enthalten in. Schwarze dicke Pfeile: ist enthalten in. Blaue Pfeile beschreiben die möglichen Zugriffe unter den Klassen

Wert zurückgeliefert.

Die Klasse Motor

Die Klasse Motor bildet den Probenmotor digital ab. Sie enthält eine Referenz auf die Klasse Protractor. Mit der Methode `turnMotorProbe(String messageWithAngles)` ist es möglich, den Probenmotor um einen beliebigen Winkel zu drehen. Dabei wird der gewünschte Winkel als String übergeben. Da dieser mit Hilfe eines Parsers in einen double Wert umgewandelt wird, ist es möglich, durch Eingabe von einem Plus- oder

Minuszeichen am Anfang des Strings auch relative Winkel zu fahren. Es ist möglich den gewünschten Winkel bis auf ein tausendstel Grad einzustellen.

Die Klasse PEM

Die Klasse PEM bildet den photoelastischen Modulator ab. Sie stellt die Methode `rotatePEM(int steps)` zur Verfügung, die es ermöglicht den PEM um einen gewünschten Winkel zu drehen. Der Methode wird die Anzahl der Schritte (ein Schritt $\widehat{=}$ 5 mdeg) als eine Integer Zahl übergeben. Ist die Zahl negativ, so dreht sich der PEM mit dem Uhrzeigersinn, ist sie positiv, so dreht sich der PEM gegen den Uhrzeigersinn. Die Anzahl der Schritte, die gefahren werden können, ist auf 50 in jede Richtung beschränkt. Dabei wird der PEM unabhängig von der gefahrenen Schrittzahl immer in Einzelschritten gefahren, um die Probleme, die in Kapitel 3.3 beschrieben sind, zu verhindern. Intern zählt die Klasse PEM die Schritte und merkt sich in welche Richtung der PEM gedreht wurde. So wird die Position intern immer mitgespeichert. Mit der Methode `pemToStartConfiguration()`, die keinen Wert zurück liefert, wird der PEM wieder auf die ursprüngliche Ausgangsstellung gefahren. Der PEM-Motor besitzt keinen Winkelmesser und kann daher nicht anders als oben beschrieben vor zu großen Drehung geschützt werden.

Die Klasse Lockin

Die Klasse Lockin stellt die Abbildung des Lock-In-Verstärkers dar. Diese Klasse enthält eine Referenz auf ein Objekt der Klasse PEM und stellt die Methoden zur Kalibrierung des Messsignals (`calibrate(int steps,int howOften)`) zur Verfügung. Es ist möglich dabei die Schrittweite anzugeben und damit sowohl den Winkel um den der PEM dabei gedreht werden soll. Desweiteren kann bestimmt werden, über wie viele Kalibrierungswerte gemittelt werden soll.
Bei der Implementierung der Kalibrierung konnte der alte Algorithmus soweit optimiert werden, dass bei gleicher Anzahl an Werten, die benötigte Zeit für die Kalibrierung auf die Hälfte gesunken ist. Die Verbesserung liegt darin, dass der jetzt verwendete Algorithmus aus allen aufeinander folgenden Messsignalen Wertepaare bildet und nicht wie bis her nur aus jedem zweiten Paar.
Darüber hinaus ist es möglich mit entsprechenden get und set Methoden die Wertepaare und den Skalierungsfaktor auszulesen sowie den Skalierungsfaktor selbst zu setzen.

Die Klasse Hallsonde

Die Klasse Hallsonde stellt die Methode `getCurrentPreciseMagneticFieldStrength()` zu Verfügung, die das Auslesen der Magnetfeldstärke ermöglicht. Diese wird als ein double Wert zurückgegeben. Dabei wurde die Methode so implementiert, dass die Klasse selbständig überprüft, in welchem Sensivitätsbereich die Hallsonde gerade die Magnetfeldstärke misst und ggf. den optimalen Sensivitätsbereich einstellt um genauere Werte zu liefern.

Die Klasse Magnet

Diese Klasse gehört zu den umfangreichsten und ist für die Steuerung des Magneten beim MOKE zuständig. Sie enthält eine Referenz auf das Objekt der Klasse Hallsonde. Die Klasse Magnet bietet unter anderem die Methoden `changeBFieldByVolt(double volt)` und `changeBFieldUser(String message)`. Die erste Methode steuert die Magnetfeldstärke, indem eine Spannung zwischen 0 und 10 Volt an den Anschluss der Steuereinheit des Magneten angelegt wird. Diese wiederum entspricht einer Stromstärke I, mit welcher der Magnet betrieben wird. Mit der zweiten Methode kann man ein gewünschtes Magnetfeld in mT angeben, das dann eingestellt wird. Übergibt man den Methoden zu hohe Werte, so werden diese auf den höchstmöglichen Wert gesetzt und die Methode wird dann ausgeführt. `changeBFieldUser(String message)` wird aufgerufen, wenn der Anwender den Magneten direkt über die in Kapitel 5.2.1 beschriebene Benutzeroberfläche steuert. Dabei ist es möglich die Magnetfeldstärke beliebig zu ändern. Es können also auch sehr große Änderungen hervorgerufen werden. Im ungünstigsten Fall benötigt der Magnet bis zu 550 ms um die gewünschte magnetische Feldstärke zu erreichen (Abbildung 5.10). Um sicher zu stellen, dass der Magnet zuverlässig arbeitet, wird nach der Übermittlung der gewünschten Feldstärke an die Geräte das Programm für 700 ms angehalten. Bei der Methode `changeBFieldByVolt(double volt)` ist eine Wartezeit von 225 ms eingestellt, weil diese Methode bei der automatischen Messung eingesetzt wird. Hier sind die Änderungen der Magnetfeldstärke meist klein.

Beide Methoden berücksichtigen dabei die Beziehungen zwischen dem definierten Koordinatensystem (vgl. Kapitel 2.2) und den möglichen Polungsrichtungen des Magneten (Abbildung 5.10). Damit ist sicher gestellt, dass die Aufnahme einer Magnetisierungskurve immer auf gleiche Weise erfolgt, unabhängig von der Stellung des Magneten. Voraussetzung dabei ist, dass der Magnet mittels der Methode `calibrateMagnet()` kalibriert

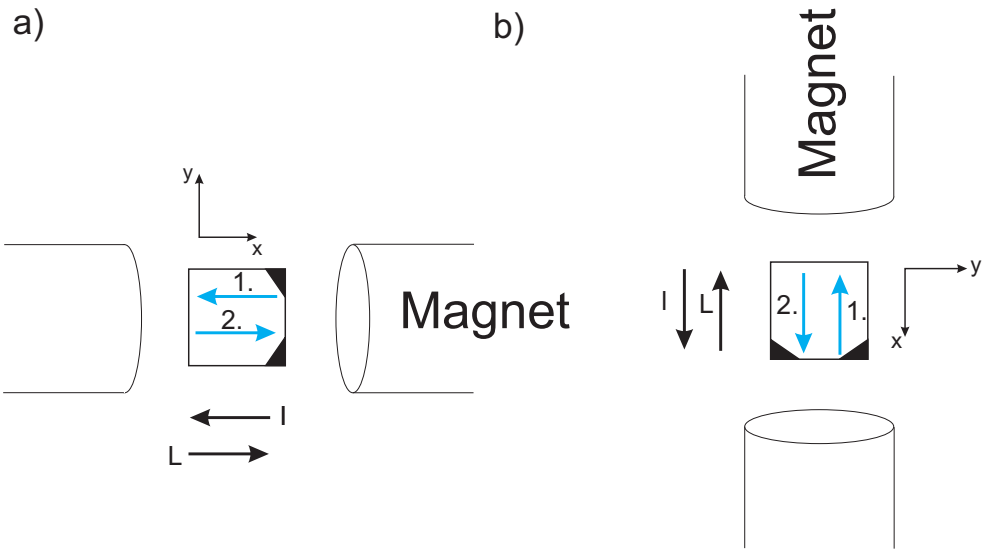


Abbildung 5.9: Beziehungen zwischen dem definierten Koordinatensystem, Lage der Probe, Polungsrichtungen des Magneten und Reihenfolge der Aufnahme einer Magnetisierungskurve. Blaue Pfeile: Richtung und Reihenfolge der Aufnahme einer Magnetisierungskurve. Schwarze Pfeile: Polrichtungen des Magneten. a) Die Einstellung l entspricht der negativen Magnetfeldstärke und L der positiven. b) L entspricht der negativen und l der positiven Magnetfeldstärke.

wurde. Sind die Eingaben bei `changeBFieldByVolt(double volt)` und `changeBFieldUser(String message)` negativ, so wird nach Definition immer ein negatives Feld angelegt, während bei positiven Werten ein positives Feld angelegt wird. Dabei polt der Magnet sich bei entsprechender Eingabe selbständig um. Die Methoden dazu sind für den Anwender nicht sichtbar und so konzipiert, dass der Magnet dabei geschont wird. Sollte

eine Umpolung des Magneten stattfinden so werden intern die Methoden `setStandardPolarity()` (Einstellung L) bzw. `setNotStandardPolarity()` (Einstellung l) dazu benutzt. Die Einstellungen L und l geben vor, in welche Richtung der Strom durch den Magneten fließst und damit in welche Richtung das Magnetfeld gerichtet ist. Die Methoden fahren die Magnetfeldstärke dabei zunächst auf null herunter, bevor die eigentliche Umpolung ausgeführt wird. Dabei wird von dem `worst-case`-Szenario ausgegangen

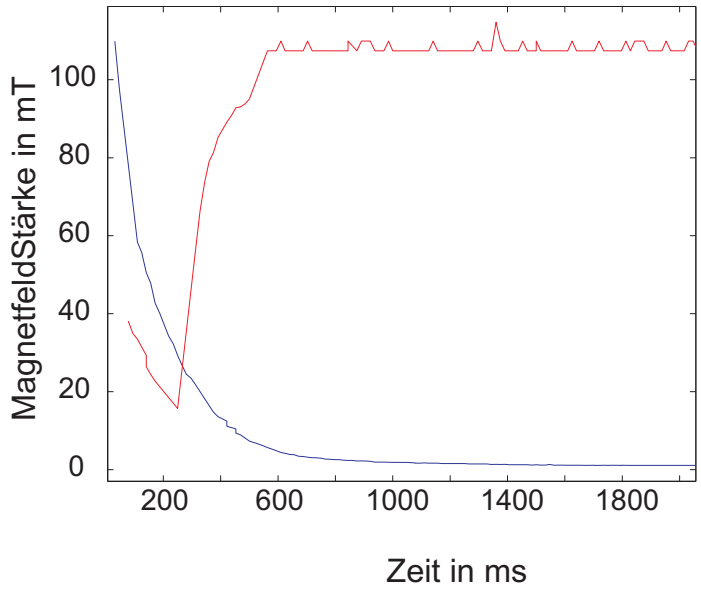


Abbildung 5.10: Der Verlauf der Magnetfeldstärke in Abhängig von der Zeit. Rote Kurve: Der Magnet wird voll aufgedreht. Blaue Kurve: Der Magnet wird bei voller Magnetfeldstärke abgeschaltet.

(der Magnet soll bei voller Magnetfeldstärke umgepolt werden) und das Programm wartet zwischen dem Abschalten des Magneten und der Umpolung 2100 ms. Weil nach dieser Zeitspanne die magnetische Feldstärke auf den Wert null abgesunken ist (Abbildung 5.10).
Eine weitere wichtige Methode dieser Klasse ist die schon erwähnte `calibrateMagnet()`. Diese sollte immer aufgerufen werden sobald eine Änderung an dem Messaufbau statt gefunden hat. Sie kalibriert alle benötigten Variablen in Abhängigkeit von der maximalen magnetischen Feldstärke und prüft dabei, ob die Hallsonde richtig steht. Gegebenenfalls wird die Fehlstellung der Hallsonde durch interne Verrechnungen mit ”-1” korrigiert. Mit `getMagneticFieldStrength()` kann die aktuelle Magnetfeldstärke abgefragt werden, wobei hier der Aufbau und die mögliche Fehlstellung der Hallsonde mit berücksichtigt werden (vgl. Kapitel 3).

Die Klasse MOKEMeasurementDevice

Diese Klasse bildet die gesamte Messanlage ab. Sie ist das Bindeglied zwischen der GUI und den implementierten Messgeräten und enthält damit auch Referenzen auf

die Objekte aller oben beschriebenen Klassen. Die Referenzen können mit entsprechenden get-Methoden aufgerufen werden. `MOKEMeasurementDevice.java` stellt die Methode `startMeasurement(String sampleName, LinkedList<Double> iValuesForMagnet, LinkedList<Double> angles, int numberOfHysteresis, String info )` zu Verfügung. Mit ihr kann eine automatische Messung an einer Probe durchgeführt werden. Dabei sind die Argumente, die diese Methode empfangen muss der Reihe nach:

- der Name der Probe,
- Liste mit Magnetfeldstärkewerten, die angefahren werden sollen,
- Liste mit Winkeln, bei denen die Probe untersucht werden soll,
- Anzahl der Magnetisierungskurven pro Winkel und
- zusätzliche Informationen, die an den Namen angehängt werden, unter dem die Dateien später gespeichert sind.

Ist die Methode gestartet, so läuft die Messung automatisch ab. Dabei werden die Magnetisierungskurven gleich nach ihrer Aufnahme und nach der Mittelung in einer Dat-Datei gespeichert. Sie befinden sich in einem nach der Probe benannten Ordner. Bei der Speicherung der Magnetisierungskurven wurde darauf geachtet, dass die gespeicherte Datei mit negativen x-Werten (das gemessene magnetische Feld) beginnen, unabhängig davon wie die Magnetisierungskurve aufgenommen wurde. Diese Maßnahme senkt den weiteren Auswertungsaufwand der gemessenen Daten, da die von Robin Schubert in [7] entwickelte Software nur Magnetisierungskurve zuverlässig verarbeiten kann, die bei negativen x-Werten beginnen und damit das Aufbereiten der Messdaten für weiter Analysen entfällt. Neben der Methode zur automatischen Messung stellt `MOKEMeasurementDevice.java` die Methode `quickHysteresis()` zur Verfügung mit der eine Schnellhysterese aufgenommen werden kann. Dabei wird die Hysterese mit einer Auflösung von 392 Messpunkten in jede Richtung aufgenommen und gemittelt. Sie liegt dann in einer Klassenvariable gespeichert und kann mit `saveQuickHysteresis(String filename)` gespeichert oder zur Visualisierung mit der Methode `prepareDataForGraphCanvas(Vector<Vector<Double» data)` als ein Objekt vom Typ `Data` (siehe [7]) aufgerufen werden.

5.2.3 Package supportingClasses

In diesem Paket werden alle Hilfsklassen und Interfaces zusammengefasst. Bei Hilfsklassen handelt es sich hier um vollwertige Klassen im Sinne von Java. Da die Aufgaben, die sie verrichten von allgemeiner Natur in Informatik sind, wurden diese Klassen aus dem Hauptprogramm ausgelagert. Dadurch wird der Code des Hauptprogramms lesbarer und

leichter verständlich. Bis auf die Klasse `Parser.java` bieten alle statische Methoden an. Dadurch muss nicht erst ein Objekt dieser Klasse erstellt werden um diese Methoden zu nutzen.

Die Klasse AppendByteDataTogether

Die Klasse `AppendByteDataTogether` stellt die Methode `conditionData(byte[] out, byte[] in)` zur Verfügung. Diese Methode dient hauptsächlich dazu zwei beschriebene Byte-Arrays an einander zu hängen. Hier wird davon ausgegangen, dass der Inhalt des Arrays ASCII kodiert ist. Daher werden die anzufügenden Daten an das Array, das als Parameter `out` übergeben wurde, gleich an der ersten Stelle angehängt, die eine 0 enthält.

Die Klasse ComplexDateFormater

Diese Klasse stellt Methoden zu Verfügung, die das aktuelle Datum als einen String liefern. Dabei hat das Datum die Form JJJJMMTT (J: Jahr, M: Monat, T: Tag des Monats). Es kann auch ein Datum mit der Uhrzeit zurückgeliefert werden mit der Form JJJJMMTT_HHmmss (H: Stunden, m: Minuten, s: Sekunden).

Die Klasse DecToHexString

`DecToHexString` bietet Methoden zur Umwandlung einer Dezimalzahl in eine Hexadezimalzahl an. Es können Zahlen umgewandelt werden, die als `double` oder `int` vorliegen. Wird eine Zahl vom Typ `double` umgewandelt, so werden die Nachkommastellen nicht berücksichtigt.
Die Zahl wird dann von der Methode als ein String zurückgegeben.

Die Klasse OrderVerification

Diese Klasse bietet Methoden zur Überprüfung einer in ASCII codierten Nachricht, die als Byte-Array vorliegt. Dabei werden die von den Geräten zurückgeschickten Antorten auf Fehler zu überprüfen. Die Methode `verifyOrder(byte[] whatWasSended,String order)` überprüft den Inhalt des Arrays auf Gleichheit mit einem String. `verifyOrder(byte[] whatWasSended,String order,int index)` überprüft die Gleichheit eines einzigen Zeichens an einer vorgegebenen Stelle.

Die Klasse Parser und die Interfaces Funktion und Operation

Die Klasse `Parser` wurde aus [10] übernommen. Sie bietet Methoden an, welche mathe-

matische Ausdrücke, die als String vorliegen, parsen und verrechnen können. Das Ergebnis liegt dann als eine `double` Zahl vor. Dieser Parser kann unter anderem einfache mathematische Operationen wie die Addition, Subtraktion, Division und Multiplikation erkennen.

5.3 Erste Praxiserfahrungen mit der Software

Durch die oben beschriebenen Maßnahmen konnte ein Programm entwickelt werden, dass die in Kapitel 4.3 genannten Anforderungen erfüllt. Das Ansteuerungsprogramm verfügt über eine Benutzeroberfläche, die eine einfache Steuerung der magneto-optischen Messanlage ermöglicht. Über diese Software können die Geräte direkt angesteuert werden ohne, dass eine Fehlbedinung Schaden an der Hardware anrichten kann. Mit der Funktion `Fast Measurement` ist es dem Benutzer möglich eine Magnetisierungskurve relativ schnell aufzunehmen. Dabei können die Ergebnisse sofort angezeigt werden. Bei den ersten Messungen konnte festgestellt werden, dass die Messzeit von 14 Stunden auf 7 Stunden reduziert wurde.

6 Zusammenfassung und Ausblick

Das Ziel dieser Bachelorarbeit war es eine komfortable und zuverlässige Anseteuerungssoftware für eine magneto-optische Messanlage zu entwickeln. Dabei wurden alle Punkte, die in dieser Arbeit gesetzt wurden, mehr als zufriedenstellend erfüllt. Es wurde nicht nur die Funktionalität des alten Programms übernommen, sondern darüberhinaus verbessert. Betrug die Messzeit davor 14 Stunden, so konnte diese auf unter 7 Stunden verkürzt werden. Darüber hinaus bietet das Programm mit der Funktion `Fast Measurement` eine bequeme und schnelle Möglichkeit sich einen groben Überblick über die zu untersuchende Probe zu verschaffen. Durch die Aufteilung des Programms in Klassen, ist es jetzt möglich den Aufbau ohne großen Programmieraufwand zu verändern.
Die Fähigkeit des Programms die falsche Stellung der Hallsonde zu korrigieren reduziert den Nachbearbeitungsaufwand der Daten enorm. Mit der jetzt zur Verfügung stehenden Benutzeroberfläche und den mit ihr verbundenen Funktionen ist eine Geräte schonende und sichere Arbeitsweise möglicht. Da unpassende Eingaben, die eine Beschädigung hervorrufen könnten abgefangen werden. War eine Änderung der Messparameter früher mit der Neukodierung des Programms verbunden, kann diese bequem in der GUI vorgenommen werden.
Um die Benutzerfreundlichkeit zu steigern, sollte die GUI so erweitert werden, dass eine Speicherung der veränderten Messparameter möglich ist. Ein automatisch angefertigtes Messprotokoll, das neben den vorgenommenen Einstellungen auch den Messverlauf in einer Datei speichern würde, wäre von großem Vorteil. Sollte eine Messung aus unbekannten Gründen unterbrochen werden, könnte man so zumindest den Verlauf der Messung nachvollziehen.
Eine weitere Möglichkeit das Programm zu erweitern wäre, die in [7] entwickelte Auswertungssoftware in die Benutzeroberfläche einzubinden so könnten die Messdaten direkt ausgewertet werden. Als zusätzlicher Schritt sollte der digitale Lock-In-Verstärker angesteuert und ausgelesen werden. Hierzu müsste eventuell ein Ansteuerungsgerät eingerichtet werden.

7 Summary

The aim of this bachelor thesis was to develop a software program to control the equipment used in experimentation making use of the magneto optical Kerr effect. This setup can be used to examine the magneto-optical properties of materials by measuring their reflection of laserlight in the presence of a magnetic field. An existing Matlab programs functions were to be retained while at the same time adding a graphical user interface that would allow the user a greater level of control over the settings. This was achieved using Java. In the new program for instance the angle of the magnet, the strength of the magnetic field and the polarization of the light can now be adapted easily without changing the code. Additionally, the program grants a greater measure of visibility of features such as the position of the hall-effect probe, which reduces the time spent in subsequent error handling, drastically. It ensures correct calibration by allowing only small changes in angle of the PEM and the analyzer. Furthermore the new program has includes features that protect the equipment. The software prevents both a pole reversal while the magnetic field is activated and the abrupt stoppage of the probe motor. The new software also has an option to run a rough analysis of the probe quickly. First tests of the program have revealed that the time necessary to run a probe could be reduced from 14 to 7 hours. Future improvements of the software could include an expansion of the GUI to include an option for saving the measurement parameters. An automatic protocol could include the course of the measurement in case the measurement is interrupted for unknown reasons. Another option to expand the software would be to embed the analysis software in the user interface or to realise the activation of the Lock-In-Enhancer.

Literatur- und Internetadressenverzeichnis

[1] **Kerr, J.**, *On rotation of the plane of polarization by reflection from the pole of a magnet.*, Phil. Mag. , 3:321-343, 1877.

[2] **Visnovsky S.**, *Magneto-optical permittivity tensor in crystals*, Czech. J. Phys., B 36:1424-1433, 1986

[3] **J. Hamrle, S. Blomeier, O. Gaier, B. Hillebrands, H. Schneider, G. Jakob, K. Postava, and C. Felser.**, *Huge quadratic magneto-optical Kerr effect and magnetization reversal in the Co_2FeSi Heusler compound.*,J. Phys. D: Appl. Phys., 40:1563-1569, 2007.

[4] **Vavassori P.**, *Polarization modulation technique for magneto-optical quantitative vector magnetometry.*Appl. Phys. Lett., 77:11:1605-1607, 2000.

[5] **Kuschel, T.**, *Aufbau einer Apparatur zur Messung des magnetooptischen KERR-Effekts*, Universität Osnabrück, Diplomarbeit, 27. November 2007

[6] **Bardenhagen, H.**, *MOKE Messungen an ferromagnetischen epitaktischen Schichten*, Diplomarbeit, Universität Osnabrück, 2009.

[7] **Schubert, R.**, *Entwicklung einer Software zur Analyse von magnetooptischen Daten*, Bachelorarbeit, Universität Osnabrück, 2009.

[8] **Wilkens, H.**, *Röntgen- und Vektor-MOKE-Untersuchung ferromagnetischer Fe-Schichten*, Diplomarbeit, Universität Osnabrück, 2009.

[9] **Vornberger, O.**, *Algorithmen*, Script, WS 2006/2007

[10] **Sigg, B.**, *Parser für mathematische Formeln*
`http://www.java-forum.org/allgemeines/12306-parser-fuer-mat\`
`discretionary{-}{}{}hematische-formeln.html`

[11] **Ullenboom, Christian**, *Java ist auch eine Insel*
`http://galactica.dyndns.org/java/book/javainsel4/javainsel_10_007.`
`htm`

[12] **Daboo, C., J. A. C. Bland, R.J. Hicken, A. J. R. Ives, M. J. Baird und M.J. Walker:**, *Vectorial magnetometry with the magneto-optical Kerr effect applied to CoCu- Co trilayer structures.*, Phys. Rev. B, 47:18:11852-11859, 1993.

[13] **v.A.**, *www.mikrocontroller.net*
`http://www.mikrocontroller.net/forum/pc-programmierung/java`

[14] **Robinson, C. C.:**, *Longitudinal Kerr Magneto-Optic Effect in Thin Films of Iron, Nickel and Permalloy.*,J. Opt. Soc. Am., 53:6:681-689, 1963.

Aufgrund der Aktualität der in dieser Bachelorarbeit eingesetzten Techniken sind zum Beleg viele Internetseiten und verlinkte pdf-Dokumente aufgeführt. Da sich die Erreichbarkeit und der Inhalt dieser verlinkten Seiten schnell ändern können, kann nicht sichergestellt werden, dass die unter den Links zu findenden Informationen noch aktuell sind. Zum Zeitpunkt der Fertigstellung dieser Bachelorarbeit am 28.11.2010 waren alle im Literatur- und Internetadressenverzeichnis aufgeführte Quellen zu finden.

Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Osnabrück, 28. November 2010 _____

Unterschrift Kamil Balinski