DIPLOMARBEIT

zur Erlangung des akademischen Titels
Diplom-Informatiker

Vom Reverse Engineering zur Programmerweiterung: Automatische Justage für ein Röntgentopographie-Steuerprogramm

von
Sebastian Freund
Derrick Hepp

April 2001

1.Gutachter: Prof. Dr. Klaus Bothe
2.Gutachter: Prof. Dr. Rolf Köhler

Institut für Informatik
Math.-Naturwiss. Fakultät II
Lehrstuhl für Softwaretechnik
Rudower Chaussee 25
Humboldt-Universität zu Berlin

Selbständigkeitserklärung

Hiermit versichern wir, daß wir die vorliegende Diplomarbeit selbständig und nur unter Zurhilfenahme der angegebenen Quellen erstellt haben.

Berlin, 23. April 2001 Derrick Hepp Sebastian Freund

Einverständniserklärung

Hiermit erklären wir unser Einverständnis mit der öffentlichen Aufstellung unserer Diplomarbeit in der Bibliothek des Instituts für Informatik.

Berlin, 23. April 2001 Derrick Hepp Sebastian Freund

Danksagungen

Unser besonderer Dank gilt Herrn Prof. Bothe und Herrn Prof. Köhler sowie Frau Richter und Herrn Sacklowski, die für alle fachlichen Fragen immer ein offenes Ohr hatten und uns beratend zur Seite standen.

Außerdem möchten wir Gerda Hepp und Dr. Friedrich-Karl Hecht für die sehr gründlichen Korrekturen, in die sie viel Zeit investiert haben, danken. Durch sie konnten viele sprachliche „Schnitzer" und grammatikalische Verrenkungen vermieden werden.

Des weiteren möchten wir unseren Eltern und unseren Lebensgefährtinnen für ihre Unterstützung während des Studiums danken.

Zusammenfassung

Die vorliegende Diplomarbeit entstand im Rahmen des Projektseminars „Software-Sanierung" am Lehrstuhl Software-Technik des Instituts für Informatik der Humboldt-Universität zu Berlin.

Das Projektseminar wurde Sommer 1998 durch Prof. Bothe auf eine Anfrage von Prof. Köhler vom Institut für Physik ins Leben gerufen. Das Hauptanliegen des Seminars besteht darin, die Kenntnisse, die in der Lehrveranstaltung „Einführung in das Software-Engineering" vermittelt wurden, praktisch auf ein reales Softwareprojekt anzuwenden.

Ein Steuerprogramm für eine physikalische Apparatur zur Untersuchung von Halbleiterkristallen bildet den Gegenstand des Projekts. Während des Seminars wurden mehrere Aufgabengebiete abgesteckt und an Gruppen von ein bis vier Studenten verteilt. Die Aufgaben bestehen zum einen aus der Analyse der vorhandenen Quelltexte und der Dokumentation der Software und zum anderen aus der Lokalisierung und Beseitigung von Programmfehlern sowie der Implementation neuer Programmfunktionen.

Das Hauptaugenmerk dieser Diplomarbeit liegt auf der Kombination von Verfahren des Reverse Engineering und des Forward Engineering. Erkenntnisse, die aus dem Reverse Engineering Prozeß gewonnen wurden, sind dazu verwendet worden, eine automatische Justagefunktion für die Röntgentopographie in das bestehende Programm zu integrieren.

Dazu werden die physikalischen Grundlagen der Röntgentopographie von Halbleiterkristallen erarbeitet und die Anforderungen an die Funktion in einem Pflichtenheft zusammengefaßt. Ausgehend vom Prozeß der manuellen Justage wird ein für das Optimierungsproblem geeigneter Algorithmus formuliert.

Für die programmtechnische Umsetzung werden die relevanten Schnittstellen des bestehenden Programms analysiert und dokumentiert. Darauf aufbauend wird eine Erläuterung der Einbettung der automatischen Funktion in die bestehende Software gegeben.

Die Vorgehensweise während der Testphase zur Überprüfung der korrekten Funktionalität wird schwerpunktmäßig dargestellt. Außerdem beinhaltet die Diplomarbeit eine Dokumentation der neu implementierten Programm-

funktion in Form eines Handbuchs für den Anwender.

Die im Rahmen dieser Diplomarbeit programmierte Funktion „Automatische Justage" dient den Mitarbeitern der Arbeitsgruppe Prof. Köhlers zur Arbeitsoptimierung bei der Einstellung der Halbleiterkristallproben. Die gewonnenen Erkenntnisse aus dem Reverse Engineering können von den am Projekt beteiligten Studenten zum weiteren Verständnis der Software verwendet werden.

Überblick über die Kapitel

Das erste Kapitel führt in den physikalischen Sachverhalt ein. Dazu werden grundlegende Begriffe, die zum weiteren Verständnis der Arbeit nötig sind, geklärt.

Das zweite Kapitel gibt einen Überblick über die Steuersoftware. Dabei wird der Ausgangszustand der Software, wie sie dem Lehrstuhl Softwaretechnik übergeben wurde, zu Grunde gelegt. Das heißt, der IST-Zustand der Software wird beschrieben und bewertet. Außerdem wird die benutzte Hardware bezglich der Topographie und deren Einbindung erläutert.

Das dritte Kapitel liefert eine Beschreibung, wie eine Justage einer Meßprobe manuell mit dem Steuerprogramm und die anschließende Topographie durchgeführt werden.

Für eine klare Abgrenzung der neu zu implementierenden Funktion „Automatische Justage" wird eine Anforderungsdefinition benötigt. Diese erfolgt im vierten Kapitel mit Hilfe eines Pflichtenheftes.

Im fünften Kapitel wird ein Algorithmus zum Lösen des Problems der automatischen Justierung einer Meßprobe erarbeitet. Es erfolgt eine Beschreibung des mathematischen Problems, und es werden Verfahren für die Lösung dieser Art von Problemen vorgestellt.

Das sechste Kapitel widmet sich dem Design der Softwareerweiterung. Der Ablauf der einzelnen Prozesse, die während der „Automatischen Justage" abgearbeitet werden, wird beschrieben. Außerdem erfolgt eine Auflistung der relevanten Dateien und Funktionen sowie eine statische Beschreibung der Softwareerweiterung mittels UML-Klassendiagrammen. Des weiteren wird die Einbettung in die bestehende Programmstruktur erläutert.

Im siebten Kapitel werden Probleme, die in der Implementationsphase auftraten, besprochen. Einen großen Teil in diesem Kapitel nimmt die Dialogprogrammierung ein.

Ausgiebige Tests der Justagefunktion werden im achten Kapitel ausgewertet. Dazu wird untersucht, wie gut die „Automatische Justage" bei unterschiedlichen Proben und unterschiedlichen Dialogeinstellungen funktioniert.

Das neunte Kapitel stellt eine Nutzerdokumentation der neuen Funktion „Automatische Justage" zur Verfügung. Diese soll in erster Linie den Nutzern als Handbuch für die Arbeit mit dem Steuerprogramm im Kontext der automatischen Justage einer Meßprobe dienen.

Den Abschluß bildet das zehnte Kapitel. Die in dieser Diplomarbeit gelöste Aufgabenstellung wird kurz zusammengefaßt und hinsichtlich ihres Einsatzes bewertet. Außerdem werden Ideen zur Verbesserung und Erweiterung der hier vorgestellten Lösung, die sich während der Bearbeitung des Themas ergeben haben, vorgeschlagen.

Den Kapiteln folgt ein Anhang. Den größten Teil des Anhangs nimmt eine ausführliche und komplette Beschreibung des C-Interfaces zum Ansteuern der Motorenhardware ein. Den Abschluß bilden eine tabellarische Auflistung der absolvierten Testfälle mit probenabhängigen Parametern und die neu hinzugekommenen Quelltexte zur Realisierung der „Automatischen Justage".

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Motivation . . . 1
- 1.2 Physikalischer Hintergrund . . . 2
  - 1.2.1 Versuchsverfahren . . . 2
  - 1.2.2 Bragg-Reflexion . . . 3
  - 1.2.3 Zwei-Kristall-Röntgentopographie . . . 6
  - 1.2.4 Gekrümmter Kollimator . . . 8
- 1.3 Der Versuchsaufbau . . . 9

2 Das Steuerprogramm **13**
- 2.1 Aufgabe der Software . . . 13
- 2.2 Allgemeine Softwarestruktur . . . 14
  - 2.2.1 Programmdateien . . . 14
  - 2.2.2 Die Initialisierungsdatei . . . 14
  - 2.2.3 Quelltext und Entwicklungsumgebung . . . 15
- 2.3 Hardware . . . 19
  - 2.3.1 Motoren . . . 19
  - 2.3.2 Detektoren . . . 21
- 2.4 Bewertung der Software . . . 21
  - 2.4.1 Korrektheit, Robustheit und Zuverlässigkeit . . . 21
  - 2.4.2 Dokumentation . . . 23
  - 2.4.3 Wartbarkeit . . . 23
  - 2.4.4 Benutzungsfreundlichkeit . . . 24
  - 2.4.5 Zusammenfassung der Bewertung . . . 24

3 Topographie mit dem Steuerprogramm **27**
- 3.1 Manuelle Justage . . . 27
- 3.2 Topographievorgang . . . 30

4 Anforderungsdefinition „Automatische Justage” **35**
- 4.1 Ausgangspunkt . . . 35
- 4.2 Problemdarstellung . . . 35

4.3 Anforderungsspezifikation 36
4.4 Pflichtenheft 37

5 Entwicklung eines Justagealgorithmus 45
5.1 Vorbetrachtung 45
5.1.1 Mathematische Verfahren 46
5.1.2 Eigener Ansatz 47
5.2 Algorithmus „Automatische Justage" 48
5.3 Goldener Schnitt 50
5.4 Koordinatensystemtransformation 51
5.4.1 Geometrische Transformationen 51
5.4.2 Homogene Koordinaten 53

6 Design 59
6.1 Ablauf der „Automatischen Justage" 59
6.1.1 Maximumsuche auf den Achsen 62
6.2 Einbettung in die bestehende Software 64
6.2.1 Architektur 64
6.2.2 Benutzte Funktionen und Klassen 66
6.2.3 Relevante Quelltexte 68
6.3 Statische Struktur der „Automatischen Justage" 69
6.3.1 Klassendiagramm der Dialogklasse 70
6.3.2 Klassendiagramm der Transformationsklasse 71
6.3.3 Mathematische Hilfsklassen 73

7 Probleme der Implementation 77
7.1 Integration 77
7.2 Dialogprogrammierung 79
7.2.1 Windowsnachrichtenkonzept 85
7.2.2 Timerprogrammierung 87
7.2.3 Konsequenzen für die „Automatische Justage" 92
7.3 Implementation der Funktionalität 93
7.3.1 Hinweise zur Detektoransteuerung 94

8 Test 97
8.1 Testziele 97
8.2 Teststrategie 98
8.3 Test des Dialoges ohne Proben 99
8.4 Testfälle mit Proben 102
8.4.1 Test der probenunabhängigen Einstellungen 103
8.4.2 Bewertung der Tests mit probenunabhängigen Einstellungen 110

8.4.3 Test der probenabhängigen Einstellungen 113
8.4.4 Bewertung der Tests mit probenabhängigen Einstellungen 117
8.5 Zusammenfassung des Tests 118

9 Nutzerdokumentation 121
9.1 Einführung 121
9.2 Voraussetzungen für eine „Automatische Justage” 121
9.2.1 Softwarevoraussetzungen 121
9.2.2 Hardwarevoraussetzungen 122
9.3 Voreinstellungen für die „Automatische Justage” 123
9.3.1 Manuelle Justage des Freiheitsgrades „Azimutale Rotation” 123
9.3.2 Einstellungen für den Detektor 125
9.4 Durchführung der automatischen Justage 126
9.4.1 Das Dialogfenster und seine Einstellungen 126
9.4.2 Start des Justagevorganges 129
9.4.3 Ende des Justagevorganges 129

10 Zusammenfassung und weiterführende Probleme 131
10.1 Möglichkeiten und Grenzen der Funktion „Automatische Justage” 132
10.2 Ausblick 133

A Schnittstellenbeschreibung der Motorenansteuerung 137
A.1 Interface der Motorlisten-Funktionen 138
A.1.1 Dialoge zur Motorsteuerung 147
A.2 Interface der Motor-Funktionen 150

B Testfälle mit probenabhängigen Parametern 169
B.1 Testreihe mit Probe $CuAsSe_2$ auf GaAs 169
B.2 Testreihe mit Probe SiGe auf Si-Schicht (PF916324/8/8) . . . 171
B.3 Testreihe mit Probe PF916324/04/20 174
B.4 Testreihe mit Probe PF916324/8/7 176
B.5 Testreihe mit Probe PF916324/08/5 178
B.6 Testreihe mit Kollimatorprobe 181

C Kommentierte Quelltexte 185
C.1 Dialogklasse TAutomaticAngleControl 185
C.1.1 `m_justag.h` 185
C.1.2 `m_justag.cpp` 187
C.2 Algorithmus und Kernfunktionalität: TransformationClass . . 201

C.2.1 `transfrm.h` 201
C.2.2 `transfrm.cpp` 204
C.3 Mathematische Hilfsklassen 214
C.3.1 `matrix.h` 214
C.3.2 `matrix.cpp` 218

Abbildungsverzeichnis

1.1 Bragg-Reflexion an zwei Netzebenen eines Kristalls 4
1.2 Rockingkurve 5
1.3 Strahlung der Röntgenquelle 6
1.4 Kollimierte Röntgenstrahlung 7
1.5 gekrümmter Kollimator 8
1.6 Schema des Strahlengangs 9
1.7 Probenhalter 10
1.8 Topographie-Meßplatz am Institut für Physik 11
1.9 Nahaufnahme des Probenhalters 12

3.1 Dialogfenster „Manuelle Justage" 28
3.2 Dialogfenster „Zählerkonfiguration" 29
3.3 Dialogfenster zum Starten des Topographievorgangs 31
3.4 Dialogfenster zum Einstellen der Topographieparameter 32
3.5 In einem Schichtsystem erzeugte Inseln 33

4.1 Ablaufschema der manuellen Justage 40
4.2 Dialogfenster „Automatische Justage" 43

5.1 Beispiel zur Koordinatensystemtransformation 55
5.2 Visualisierung des vorangegangenen Beispiels 57

6.1 Flußdiagramm der automatischen Justage 61
6.2 Flußdiagramm der Maximumsuche auf einer Achse 63
6.3 Architektur der Programmerweiterung 65
6.4 Klassendiagramm der neuen Dialogklasse 70
6.5 Klassendiagramm der Transformationsklasse 71
6.6 Klassendiagramm der Matrixklasse 74

7.1 Einfügen neuer Quelltextdateien in das Softwareprojekt 78
7.2 Anwendung des „Borland Resource Workshop" 80
7.3 Enwurf des Dialogfensters „Automatische Justage" 81
7.4 Menüpunkt für die neue Programmfunktion 84

7.5 Kontroll- und Nachrichtenfluß der „Automatischen Justage” . 86

9.1 Dialogfenster „Manuelle Justage” 124
9.2 Dialogfenster „Zählerkonfiguration” 125
9.3 Dialogfenster „Automatische Justage” 126

Tabellenverzeichnis

2.1 Dateien zur Ausführung des Steuerprogrammes 14

6.1 Überblick über benötigte Schnittstellenfunktionen 67
6.2 Überblick über genutzte Quelltexte 68

7.1 Überblick über die neuen Programmdateien 78
7.2 Nachrichtenbehandlung im RTK-Steuerprogramm 88

Kapitel 1

Einführung

1.1 Motivation

Im Sommer 1998 trat die Arbeitsgruppe „Röntgenbeugung an dünnen Schichten" des Instituts für Physik unter Leitung von Prof. Köhler an den Lehrstuhl „Softwaretechnik" von Prof. Bothe am Institut für Informatik mit der Bitte heran, sie bei der Restrukturierung, Dokumentierung und Erweiterung ihrer Steuersoftware zu unterstützen [10].

Bei der Software handelt es sich um ein Anwendungsprogramm, mit dessen Hilfe die Versuchsaufbauanlagen gesteuert werden. Diese Anlagen werden zur Untersuchung der kristallinen Struktur von Halbleitern benutzt. Dadurch kann man Rückschlüsse auf die Güte und den Aufbau eines Halbleiters ziehen. Bei der Erstellung der Software durch einen ehemaligen Mitarbeiter der Arbeitsgruppe wurde auf wichtige Aspekte des Software Engineering verzichtet oder nur ansatzweise eingegangen. Die strukturierte Kommentierung der Softwarequellen, die Dokumentierung der Software aus Anwendersicht und die Erstellung eines Pflichtenheftes sind nur einige von vielen Gesichtspunkten die im Software-Engineering-Prozeß berücksichtigt werden müssen.

Forschungsarbeiten am Lehrstuhl „Softwaretechnik" auf dem Gebiet des Software Engineering und Reverse Engineering [3, 8] führten zur Zusammenarbeit mit der Arbeitgruppe von Prof. Köhler. Im daraus entwickelten Projektseminar „Softwaresanierung" wurden den Studenten Teilaufgaben zugewiesen, an denen sie ihre theoretisch erlangten Kentnisse auf dem Gebiet des Software Engineering in die Praxis umsetzen konnten. Diese Diplomarbeit ist aus der Bearbeitung einer dieser Teilaufgaben aus dem Seminar hervorgegangen. Die zentrale Aufgabe der vorliegenden Arbeit ist es, die Steuersoftware um die Topographiefunktion „Automatische Justage" zu erweitern.

1.2 Physikalischer Hintergrund

1.2.1 Versuchsverfahren

Halbleiterkristalle gehören zu den strukturell perfektesten Materialien, die heute hergestellt werden können. Dennoch finden sich in diesen Kristallen Defekte. Diese Abweichungen von der Idealstruktur können im Kristall auch im Zuge von Prozeßschritten entstehen. Zur Untersuchung der Realstruktur von Halbleitern gibt es viele Methoden. Ein Teil dieser Verfahren arbeitet mit Röntgenstrahlung und wird insbesondere für Halbleiter-Schichtsysteme[1] eingesetzt.

In der Arbeitsgruppe „Röntgenbeugung an dünnen Schichten" des Instituts für Physik konzentriert man sich auf drei Untersuchungsverfahren:

1. Röntgendiffraktometrie
   In der Röntgendiffraktometrie werden Untersuchungen einzelner Ebenen im Kristall (der Abstand der Netzebenen[2] beträgt ca. 0,1 nm) vorgenommen. Die Meßprobe wird in unterschiedlichen räumlichen Orientierungen durch einen Röntgenstrahl systematisch abgetastet. Der Einfallswinkel variiert hierbei typischerweise zwischen 10 bis 90 Grad. Interferenzen, durch die Netzebenenabstände hervorgerufen, werden von Detektoren als Diffraktometriekurven ermittelt. Die Kurven lassen Aussagen z.B. zur Glattheit oder Rauhheit einzelner Netzebenen zu.

2. Röntgenreflektometrie
   Die Röntgenreflektometrie wird zur Untersuchung einzelner Schichten im Kristall (die Stapeldicke beträgt normalerweise ca. 100 nm) eingesetzt. Bei der Röntgenreflektometrie wird, wie bei der Diffraktometrie, die Meßprobe in verschiedenen Orientierungen abgetastet. Typischerweise arbeitet man mit Einfallswinkeln zwischen 0,1 und 3 Grad. Die Interferenzen, die durch die Phasendifferenz von Grenzfläche zu Grenzfläche, d.h. durch die Dicke der Schichten, entstehen, werden von Detektoren gemessen und in Reflektometriekurven abgebildet. Diese Kurven erlauben Aussagen zu Schichtdicken chemisch unterschiedlicher Schichten im Kristall.

3. Röntgentopographie
   Die Röntgentopographie wird zur Untersuchung von Versetzungen im Kristall eingesetzt. Diese können in Schichtsystemen entstehen, wenn

[1] Man spricht von einem Schichtsystem, wenn eine Schicht eines Halbleiters auf einen anderen Halbleiter aufgebracht wird.

[2] Netzebenen sind periodisch und dicht mit Atomen besetzte Ebenen im Kristall.

Verspannungen zwischen Schichten mit unterschiedlichen Netzebenenabständen ausgeglichen werden. Diese Verspannungen werden dadurch hervorgerufen, daß die Atome der einen Schicht keine Bindung mit den Atomen der anderen Schicht eingehen. Zur Erfassung von Versetzungen nutzt man aus, daß die Reflexionskurve[3] perfekter Kristalle bei fester Wellenlänge der einfallenden Strahlung und festem Netzebenenabstand sehr schmal ist (Halbwertsbreite typischerweise nur einige Winkelsekunden). Gitterdeformationen um einen Kristalldefekt rufen lokale Intensitätsänderungen hervor und werden so im Topographiebild sichtbar.

Im folgenden sollen die Grundlagen für die Zwei-Kristall-Röntgentopographie erläutert werden.

1.2.2 Bragg-Reflexion

Die physikalische Grundlage der Meßmethode zur Untersuchung von Kristallstrukturen ist die Beugung von Röntgenstrahlung an den Kristallgitterbausteinen. Diese Beugung kann auch als Reflexion der Röntgenstrahlen an den atomaren Punktladungen beschrieben werden. Durch die Kristallgitterbausteine lassen sich in verschiedene Richtungen äquidistante[4] Ebenenscharen legen. Diese Ebenenscharen werden als Netzebenen bezeichnet.

Eine wichtige physikalische Eigenschaft, die man für Röntgenbeugungsuntersuchungen, z.B. die Zwei-Kristall-Röntgentopographie, benutzt, ist die Bragg-Reflexion.

Einfallende Strahlung wird an den Netzebenen eines Kristalls reflektiert. Dabei gilt, daß der Einfallswinkel gleich dem Ausfallswinkel ist. Des weiteren wird nur ein geringer Teil je Netzebene reflektiert. Wird nun der Einfallswinkel verändert, so stellt man fest, daß nur bei bestimmten Einfallsrichtungen die reflektierte Strahlung eine meßbare Intensität, sogenannte Bragg-Reflexe, aufweist. Es sind genau die Richtungen, bei denen der Gangunterschied der an den Netzebenen reflektierten Strahlung ein Vielfaches der Wellenlänge beträgt. Die Abb. 1.1 veranschaulicht den Gangunterschied zwischen den an den Netzebenen (1) und (2) reflektierten Strahlen in einem Kristall und erleichtert das Verständnis der anschließenden Bragg-Gleichung.

[3] Die Reflexionskurve stellt die idealisiert-'gemessene' charakteristische Kurve der Röntgenstrahlungsintensität in Abhängigkeit vom Einfallswinkel der monochromatischen Planwelle dar.

[4] Die einzelnen Ebenen sind gleich von einander entfernt.

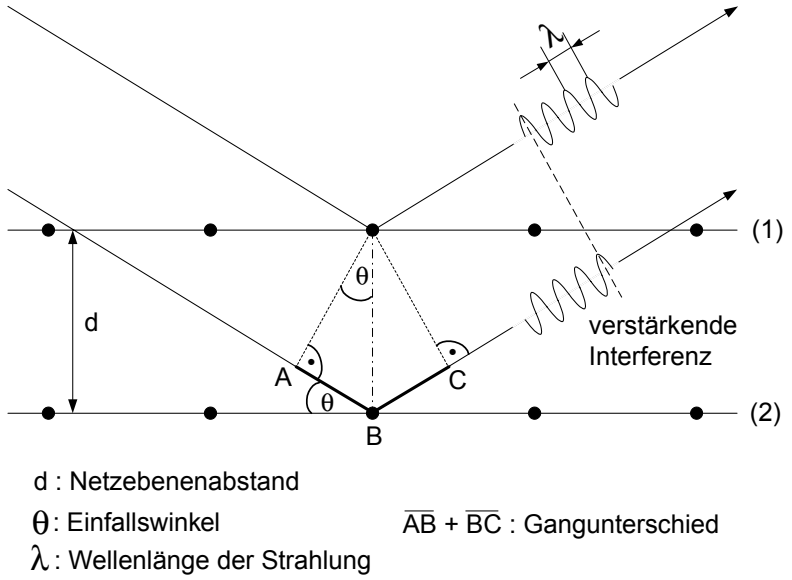


Abbildung 1.1: Bragg-Reflexion an zwei Netzebenen eines Kristalls

Die Bragg-Gleichung:

$$2d \sin\theta = n\lambda \tag{1.1}$$

mit d:	Netzebenenabstand	θ:	Einfallswinkel
n:	1,2,3,...	λ:	Wellenlänge.

Bei Erfüllung der Bragg-Gleichung wird die Strahlung durch verstärkende Interferenz komplett reflektiert. Verstärkende Interferenz bedeutet, daß die sich überlagernden Röntgenwellen[5], die auf unterschiedlichen Strahlwegen an verschiedenen Netzebenen[6] reflektiert werden, in gleicher Phase befinden. Dadurch treffen Wellenmaxima auf Wellenmaxima (siehe Abb. 1.1) und es erfolgt eine Erhöhung der Strahlungsintensität. Im Gegensatz dazu kommt es bei Phasenunterschieden zur Schwächung der reflektierten Strahlung, die gemessene Intensität nimmt ab. Treffen Wellenmaxima auf Wellenminima, so kommt es sogar zur Auslöschung der Strahlung.

Man spricht vom Bragg-Reflex, wenn eine maximale Reflexion endlicher Breite (wenige Winkelsekunden) erfolgt. Stellt man eine Meßprobe auf den Bragg-Reflex und verkippt sie über einen kleinen Bereich in Richtung Röntgenstrahl, so entsteht eine charakteristische Kurve der reflektierten Rönt-

[5]Strahlung wird auch als Welle aufgefaßt.

[6]In diesem Fall sind Netzebenen gleicher Orientierung gemeint.

genstrahlintensität in Abhängigkeit von der Winkelstellung, die sogenannte Rockingkurve[7] (Abb. 1.2). Sie ist ein Hinweis darauf, daß auch für einen Winkelbereich in der Umgebung des Bragg-Winkels θ Reflexion auftritt. Wenn die Gleichphasigkeit der Röntgenwellen, aus denen der Bragg-Reflex gebildet wird, nicht mehr gegeben ist, kommt es zur Verbreiterung der Rockingkurve. Sie entsteht, wenn die Kristallperfektion durch Defekte gestört ist [7, S.372f.].

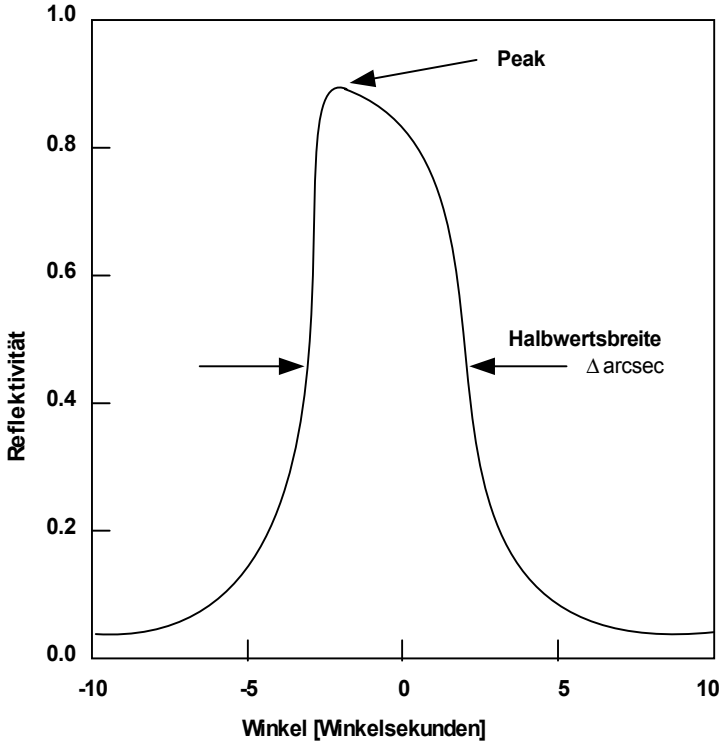


Abbildung 1.2: Rockingkurve

Aus der Rockingkurve lassen sich wichtige Kenngrößen ablesen, dazu gehören der Peak und die Halbwertsbreite. Die Halbwertsbreite gibt die Breite der Rockingkurve bei 50% der Maximalintensität in Winkelsekunden an. Je geringer die Halbwertsbreite ist, desto besser wurde der Bragg-Reflex getroffen.

[7]Die Rockingkurve stellt die real gemessene Variante der Reflexionskurve dar. Das heißt, in diesem Fall wird nicht von einem perfekten Kristall und einer perfekt monochromatischen Planwelle ausgegangen.

1.2.3 Zwei-Kristall-Röntgentopographie

Die Voraussetzung für Beugungserscheinungen an Kristallgittern ist, wie im vorangegangenen Abschnitt erläutert, daß die Wellenlänge der Strahlung kleiner als der doppelte Abstand der Netzebenen sein muß. Um dieser Voraussetzung gerecht zu werden, verwendet man Röntgenstrahlung.

Als Strahlungsquelle benutzt man eine Röntgenröhre. In dieser treffen Elektronen auf eine Kupfer-Anode. Im Ergebnis dieses Prozesses wird Strahlung emittiert, welche zum einen aus Bremsstrahlung und zum anderen aus der charakteristischen Strahlung besteht. Bei der Bremsstrahlung geben abgebremste Elektronen Röntgenstrahlung in breitem Spektrum ab, so daß weiße Strahlung entsteht.

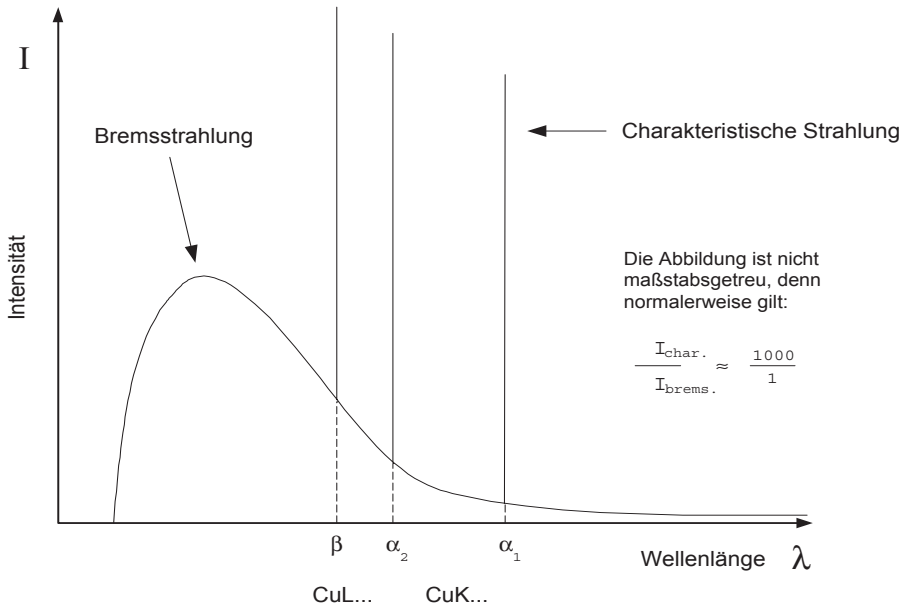


Abbildung 1.3: Strahlung der Röntgenquelle

Die charakteristische Strahlung wird erzeugt, weil die Elektronen der Anodenatome auf höhere Energieniveaus angehoben werden. Sie emittieren, im gegebenen Beispiel für Kupfer, beim Zurückfallen Röntgenstrahlung in charakteristischen Wellenlängen.

Im allgemeinen muß die erzeugte Röntgenstrahlung (siehe Abb. 1.3) zusätzlich monochromatisiert werden. Diese Funktion übernimmt im hier diskutierten Verfahren ein hochperfekter Kollimatorkristall - im Folgenden nur noch als Kollimator bezeichnet. Um eine Monochromatisierung zu erreichen, wird

der Kollimator mit der in der Röntgenröhre erzeugten Strahlung bestrahlt. Die Röntgenstrahlung unterschiedlicher Wellenlänge wird vom Kollimator in unterschiedliche Richtungen gestreut. Bei genügend großem Abstand zwischen dem Kollimator und der zu untersuchenden Probe wird dann genau eine bestimmte Wellenlänge herausgegriffen, im gegebenen Fall die CuK α_1-Linie.

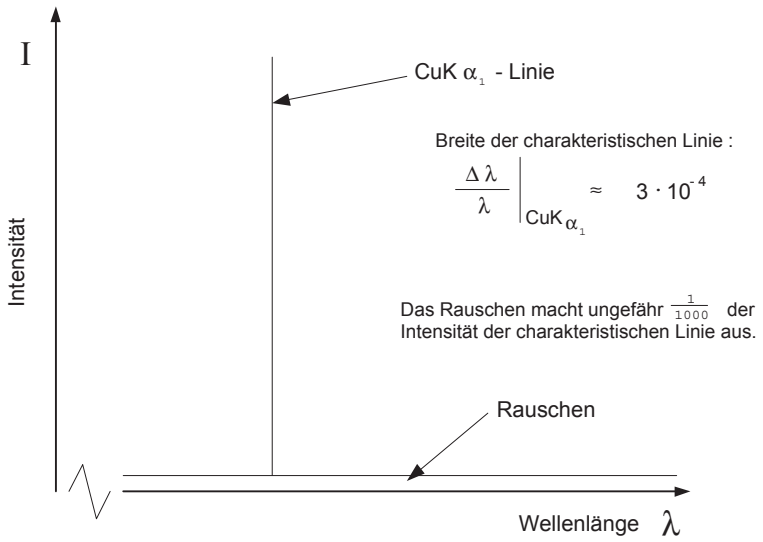


Abbildung 1.4: Kollimierte Röntgenstrahlung

Der Kollimator kann so eingestellt werden, daß man eine sehr schmale charakterische Linie erhält (siehe Abb. 1.4). Die Divergenz der Strahlung in der Einfallsebene beträgt dadurch nur noch ca. $\frac{1}{2}$ Winkelsekunde.

Die hier vorgestellte Methode zur Erzeugung von Röntgenstrahlung mit einer bestimmten Wellenlänge ist die Grundvoraussetzung für die Zwei-Kristall-Röntgentopographie. Diese kann wie folgt beschrieben werden:

Ein idealer Kollimator erzeugt aus dem von der Röntgenröhre ausgehenden schmalen Strahlenbündel durch Bragg-Reflexion ein breites, nahezu paralleles und monochromatisches Strahlenbündel, das auf die Meßprobe trifft. Von der Meßprobe erhält man dann durch Bragg-Reflexion ein Abbild, unter der Voraussetzung, daß lokal für jeden Punkt der Probe die Bragg-Bedingung erfüllt ist. Zur Erfassung des Abbildes dienen eine Fotoplatte, ein Film oder ein 2-dimensionaler Detektor, die jeweils in den von der Meßprobe reflektierten Strahl gebracht werden. Abweichungen von der Idealstruktur führen zu Abweichungen von der Bragg-Bedingung und damit zu einer kontrastreichen Abbildung, die der Realstruktur entspricht.

Aus der Vielzahl der Netzebenen des Kristalls kann ein Ausgangspunkt für die Untersuchung ausgewählt werden. Die Auswahl erfolgt durch eine entsprechende räumliche Orientierung des Kristalls bezüglich der Ebene Kollimator - Meßprobe - Detektor.

Das bei der Topographie erfaßte Raumsegment hat eine Fläche von etwa 8mm x 8mm und eine Tiefe von maximal 20 μm. Dies entspricht etwa 10^5 Netzebenen. Die Eindringtiefe hängt vom Einfallswinkel ab. Im allgemeinen gilt: je steiler dieser ist, desto tiefer liegende Netzebenen können untersucht werden.

1.2.4 Gekrümmter Kollimator

Da es vorkommt, daß die Meßproben nicht immer absolut eben sind, wurde die Möglichkeit der Krümmung des Kollimators eingeführt. Wird die Krümmung des Kollimators nicht an die Probenkrümmung angepaßt, dann tritt das Phänomen auf, daß die Bragg-Bedingung nur in einem schmalen Bereich erfüllt ist (Abb. 1.5a.). Um die Bragg-Bedingung global über der Probe zu erfüllen, muß der Kollimator entsprechend dem Krümmungsradius der Meßprobe verformt sein(Abb. 1.5b.).

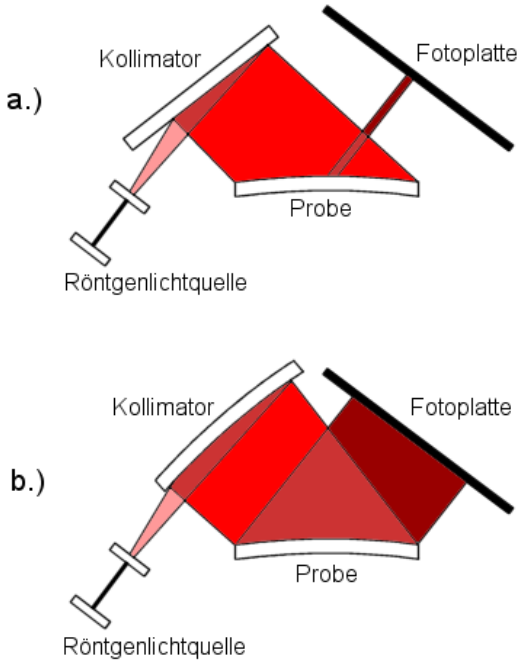


Abbildung 1.5: a.) normaler und b.) gekrümmter Kollimator

Es kann aber auch vorkommen, daß die Meßprobe zu stark bzw. unregelmäßig gekrümmt ist. Wenn dieser Fall eintritt, ist es nicht mehr möglich nur mit Krümmung des Kollimators diese Unebenheiten auszugleichen. Darum wird in diesem Fall die Probe bei maximaler Kollimatorkrümmung mehrmals belichtet, wobei in jedem Schritt die Probe ein Stück gedreht wird.

1.3 Der Versuchsaufbau

Der Topographie-Meßplatz hat folgenden Aufbau: Er besteht aus einer Röntgenquelle, einem Kollimator, einem Probenhalter zusammen mit einer Halterung für die Fotoplatte und einem Detektor, der zur Erfassung der Strahlungsintensität verwendet wird. Die Abb. 1.6 veranschaulicht den Strahlengang von der Strahlungsquelle bis zum Detektor bzw. zur Fotoplatte.

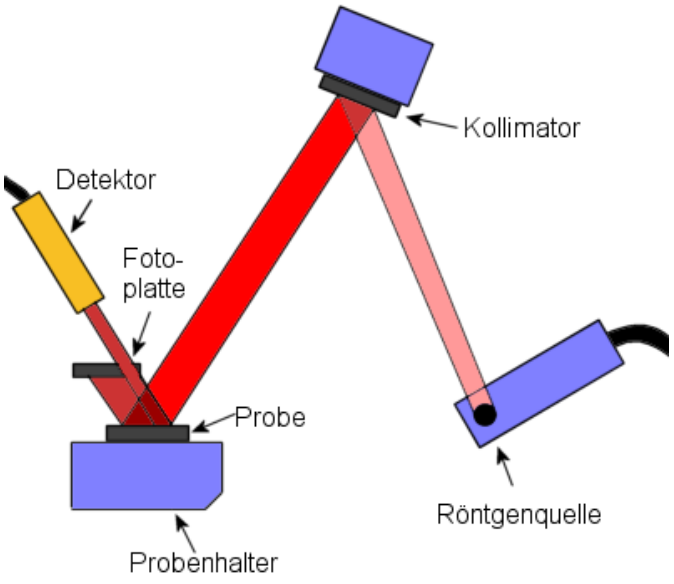


Abbildung 1.6: Schema des Strahlengangs

Jeder Meßplatz ist mit einem PC verbunden. Dieser ist für die Steuerung des Probentellers[8] und die Einstellung der Kollimatorkrümmung verantwortlich. Des weiteren wird über den Rechner die gemessene Strahlungsintensität des angeschlossenen Detektors ausgelesen.

[8]Der Probenhalter selbst ist starr, wohingegen der Probeteller im Probenhalter beweglich gelagert ist und durch die am Probenhalter angeschlossenen Motoren gesteuert wird.

Bei den Motoren, die für die Bewegung des Probentellers und des Kollimators zuständig sind, handelt es sich um Gleichstrommotoren mit Encodern, die durch Untersetzung (Getriebe und Hebelgetriebe) geringste Bewegungen zulassen. So erreicht man Drehbewegungen mit einer Genauigkeit von 0.02 Bogensekunden, was einem $\frac{1}{180000}$ Grad entspricht.

Der Probenteller verfügt über 3 Freiheitsgrade zur räumlichen Positionierung der Probe. Die Freiheitsgrade haben die folgenden Bezeichnungen: „Tilt"(TL), „Azimutale Rotation"(AR) und „Beugung". Für den Freiheitsgrad „Beugung", der in der Ebene Kollimator-Probe-Detektor liegt, wird noch eine Unterscheidung eingeführt - „Beugung fein"(DF) und „Beugung grob"(DC). Das bedeutet, daß zur Ansteuerung des Probentellers 4 Motoren benötigt werden.

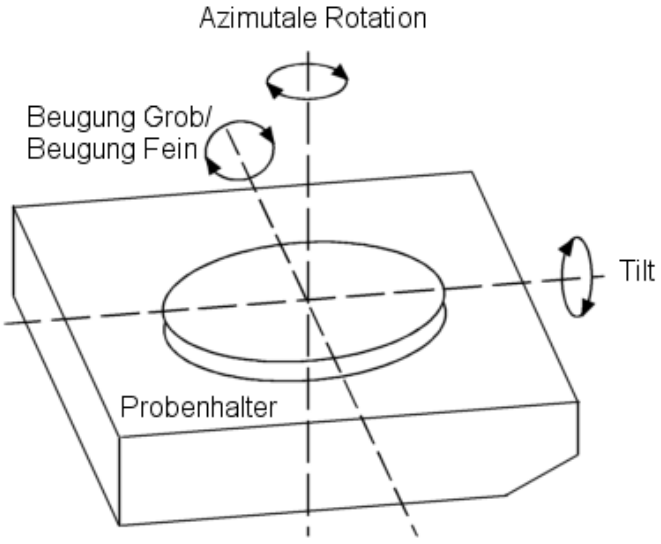


Abbildung 1.7: Freiheitsgrade der Probe bei der Topographie

Die Freiheitsgrade(Abb. 1.7) sollen im weiteren als Bezeichnung für die entsprechenden Motoren dienen. Ein weiterer Motor übernimmt die Aufgabe der Krümmung des Kollimatorkristalls. Dieser Motor hat die Bezeichnung „Kollimator" bzw. „CC".

Für das Messen der Röntgenstrahlung wird ein 0-dimensionaler Detektor[9] eingesetzt. Dieses Gerät mißt den Photonenfluß, der auf eine bestimmte Querschnittsfläche einwirkt. Das Ergebnis ist eine Zählrate, die angibt, wieviele Impulse in einem bestimmten Zeitfenster erfaßt wurden.

[9]Es gibt noch 1- und 2-dimensionale Detektoren, die zusätzlich die Positionen der Einzelphotonen in einer Dimension ermitteln.

Die Abb. 1.8 und 1.9 zeigen Fotos eines Versuchsaufbaus für die Zwei-Kristall-Röntgentopographie, wie er beim Institut für Physik in der Arbeitsgruppe „Röntgenbeugung an dünnen Schichten" verwendet wird.

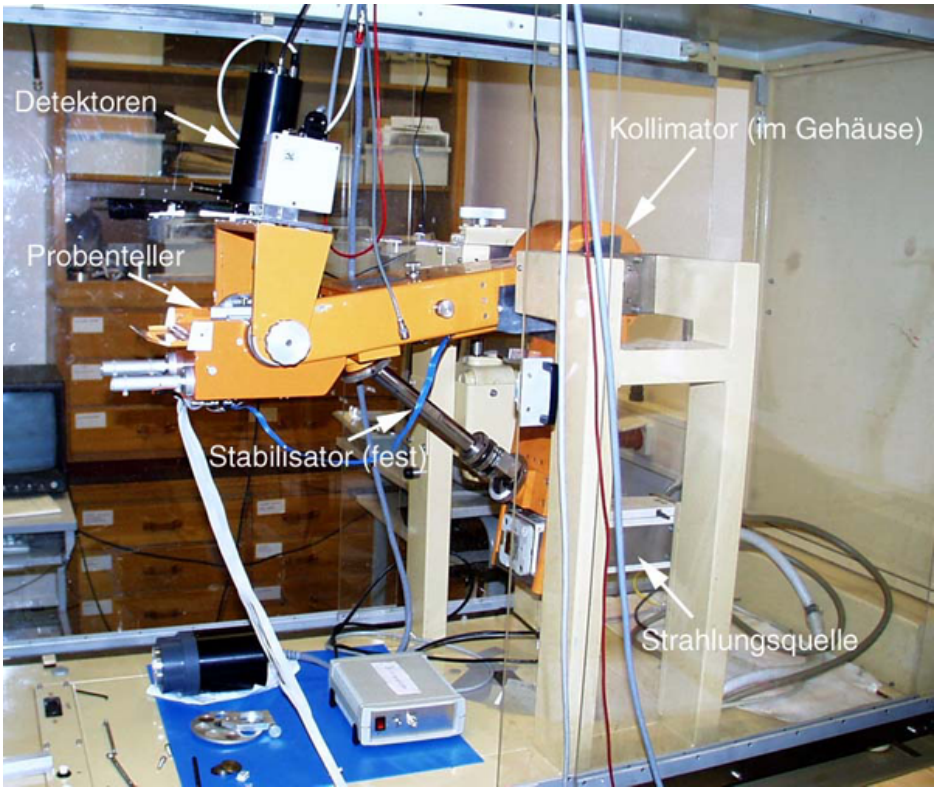


Abbildung 1.8: Topographie-Meßplatz am Institut für Physik

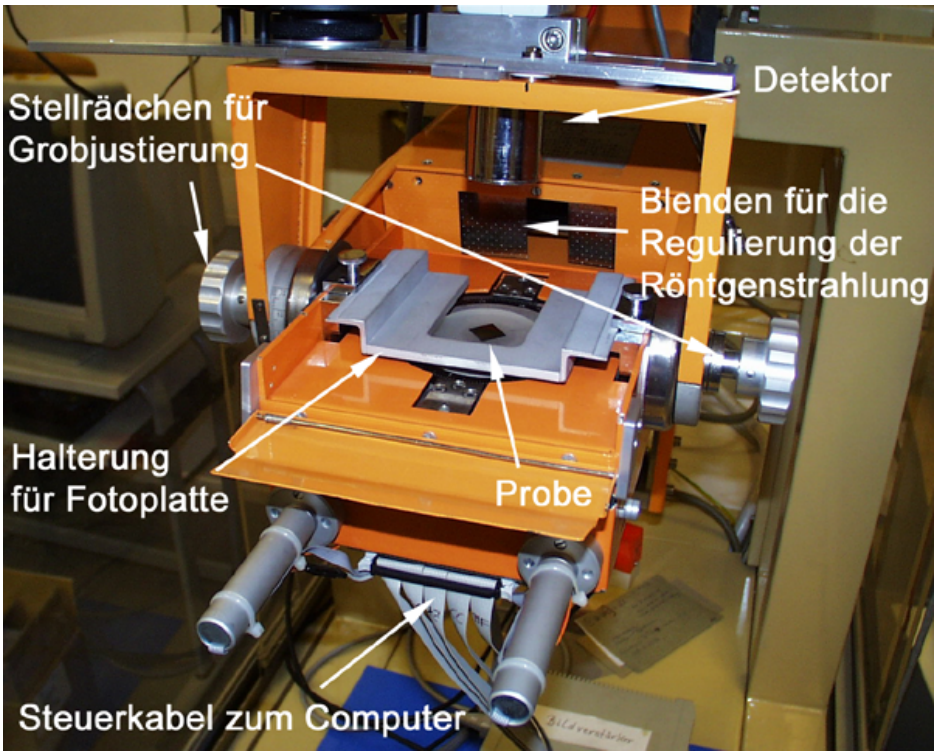


Abbildung 1.9: Nahaufnahme des Probenhalters

Kapitel 2

Das Steuerprogramm

2.1 Aufgabe der Software

Die Steuerung bzw. Nutzung der Versuchsapparatur funktioniert nur mit Hilfe eines Computers und einer Steuersoftware. Hierzu steht eine Applikation, die unter dem Betriebssystem Windows 3.1 läuft, zur Verfügung. Die Software wurde von einem ehemaligen Mitarbeiter des Instituts für Physik entwickelt und ist Gegenstand der Softwaresanierung. Im Folgenden soll der Name „RTK-Steuerprogramm"[1] als Bezeichnung der Steuersoftware verwendet werden.

Das Steuerprogramm wird für die Durchführung der Topographie, der Reflektometrie und der Diffraktometrie benötigt. Diese drei Hauptaufgaben schließen wiederum eine Anzahl von Basisaufgaben ein, darunter eine der wichtigsten Aufgaben des RTK-Programmes, die Ansteuerung der Motoren und Detektoren. Dazu gehört die Möglichkeit, Einstellungen für diese Geräte über das Steuerprogramm tätigen zu können. Eine weitere wichtige Aufgabe ist die Darstellung der Meßwerte der Diffraktometrie und der Reflektometrie.

Um die Topographie ausführen zu können, muß zuvor die Meßprobe justiert werden. Die Qualität der Justage der Meßprobe läßt sich durch die Ermittlung der Halbwertsbreite bestimmen. Auch diese Aufgaben werden mit Hilfe des Steuerprogrammes erledigt. Am Ende dieses Kapitels wird die Ausführung der Topographie und der manuellen Justage mit Hilfe des Steuerprogrammes genauer erläutert.

[1] RTK steht für Röntgen-Topographie-Kamera.

2.2 Allgemeine Softwarestruktur

2.2.1 Programmdateien

Für die Ausführung und die korrekte Arbeitsweise des Steuerprogrammes werden folgende Dateien benötigt:

`steerng.exe`	ausführbares Steuerprogramm
`steerng.ini`	Initialisierungsdatei
`motors.dll`	Dynamische Laufzeitbibliothek zur Ansteuerung und Verwaltung der einzelnen Antriebe
`counters.dll`	Dynamische Laufzeitbibliothek zur Ansteuerung der unterschiedlichen Detektoren
`splib.dll`	Dynamische Laufzeitbibliothek zum Verwalten von Intensitäts- und Diffraktometriekurven
`win488.dll`	16-Bit Treiber für C-812 Motorsteuerkarte, dieser kann wahlweise über die Initialisierungsdatei eingebunden werden und muß nicht zwingend installiert sein. Grund: Eigener implementierter Treiber für C-812 befindet sich in der `motors.dll`.
`sphelp.hlp`	Hilfebibliothek, die während des Programmlaufs, aufgerufen werden kann
`asa.dll`	16-Bit Treiber für die Ansteuerung des BraunPSD Detektors
`bc450rtl.dll`	Dynamische Laufzeitbibliothek von Borland
`standard.mak`	Makroscript für die Steuerung von Basisaufgaben

Tabelle 2.1: Dateien zur Ausführung des Steuerprogrammes

Die Applikation wurde unter Borland C++ 4.0 entwickelt und wird unter dem Betriebssystem Windows 3.1 eingesetzt.

2.2.2 Die Initialisierungsdatei

Obwohl es sich bei den Arbeitsplätzen der Topographie, der Reflektometrie und der Diffraktometrie um unterschiedliche Versuchsanlagen handelt, wird für alle Aufgaben ein und dieselbe Applikation verwendet. Um dies zu ermöglichen, wird der jeweilige Arbeitsplatz über eine Initialisierungsdatei korrekt konfiguriert. Jede Initialisierungsdatei stellt ein Unikat dar, das nur für einen bestimmten Arbeitsplatz vorgesehen ist. Der Grund dafür liegt darin, daß die Versuchsanlagen hochpräzise gesteuert werden müssen.

Dazu wurde jede Anlage mittels Lasertechnik vermessen. Die im Ergebnis erhaltenen Werte für die Konfigurierung der Motoren wurden in die jeweilige Initialisierungsdatei eingetragen. Werden also die Initialisierungsdateien zweier Topographiearbeitsplätze ausgetauscht, so ist die korrekte Steuerung der Anlage nicht mehr gewährleistet.

2.2.3 Quelltext und Entwicklungsumgebung

Der Quelltext besteht aus ca. 27000 Zeilen Programmcode. Als Programmiersprachen wurden C++ und C eingesetzt. Die ausführbare Datei und die dynamischen Laufzeitbibliotheken wurden mit einem 16-Bit Compiler von Borland übersetzt. Ungewiß war, welchen Bearbeitungsstatus die Projektquelltexte hatten, da es vom Entwickler keinerlei Dokumente gab, die darüber hätten Aufschluß geben können.

Den Ausgangspunkt bilden die vom Institut für Physik übergebenen Projektdateien, die in einem Paket mit der Bezeichnung „`XCTL9702`" zur Verfügung gestellt wurden.

Die Datei `xcontrol.ide` ist die Projektdatei für die Borland-Entwicklungsumgebung. In ihr sind alle administrativen Parameter wie Compilerschalter, Übersetzungsparameter etc. für die Projektstruktur in der Entwicklungsumgebung gespeichert. Die Projektquelltexte sind in den vier Modulen `splib.dll`, `motors.dll`, `counters.dll` und `develop.exe` zusammengefaßt. Die folgende Tabelle zeigt die Quelltextstruktur:

splib.dll

Quelltextdateien	*Beschreibung*	*inkludierte Dateien*
`dlg_tpl.cpp`	selbstdefinierte Templateklassen für Dialogobjekte inklusive der Methoden	`rc_def.h`, `comhead.h`
`l_layer.cpp`	Allgemeine Hilfsfunktionen	`rc_def.h`, `comhead.h`
`m_curve.cpp`	Klassen und Instanzen für die Datenhaltung	`rc_def.h`, `comhead.h`
`splib.rc`	Dialogressource für About-Dialog	`l_layer.h`

motors.dll

Quelltextdateien	*Beschreibung*	*inkludierte Dateien*
`motors.cpp`	Klassen zur Motorenansteuerung	`rc_def.h`, `comhead.h`, `m_motcom.h`, `m_mothw.h`, `m_layer.h`, `ieee.h`
`m_layer.cpp`	C-Schnittstelle für die Motorenansteuerung	`rc_def.h`, `comhead.h`, `m_motcom.h`, `m_mothw.h`, `m_layer.h`
`dlg_tpl.cpp`	selbstdefinierte Templateklassen für Dialogobjekte inklusive der Methoden	`rc_def.h`, `comhead.h`
`motors.rc`	Dialogressourcen für Motoren	`rc_def.h`

counters.dll

Quelltextdateien	*Beschreibung*	*inkludierte Dateien*
`counters.cpp`	Klassen für die Ansteuerung der Detektoren	`rc_def.h`, `comhead.h`, `m_devcom.h`, `m_devhw.h`, `c_layer.h`, `m_layer.h`
`c_layer.cpp`	C-Schnittstelle für die Ansteuerung der Detektoren	`rc_def.h`, `comhead.h`, `m_devcom.h`, `m_devhw.h`, `c_layer.h`, `m_layer.h`, `dfkisl.h`
`am9513.cpp`	Klassen für die Ansteuerung der Zählerkarte Am9513A	`rc_def.h`, `comhead.h`, `am9513a.h`
`braunpsd.cpp`	Klassen zur Ansteuerung des BraunPSD	`rc_def.h`, `comhead.h`, `m_devcom.h`, `m_psd.h`
`kisl1.c`	Radicon SCSCS Treiber	`dfkisl.h`, `prkmpt1.h`, `radicon.h`
`kmpt1.c`	Funktionen für Kommunikation mit der Radicon Controllerkarte	`dfkisl.h`, `prkmpt1.h`

dlg_tpl.cpp	selbstdefinierte Templateklassen für Dialogobjekte inklusive der Methoden	rc_def.h, comhead.h
counters.rc	Dialogressourcen für die Dialoge „Einstellungen für SCS" & „Zählerkonfiguration"	rc_def.h
develop.exe		
Quelltextdateien	*Beschreibung*	*inkludierte Dateien*
m_data.cpp	Modul für die Daten-Präsentation	rc_def.h, comhead.h, m_devcom.h, m_steerg.h, m_data.h
arscan.cpp	Modul für die Ausführung des AreaScans mit dem PSD und SLD	rc_def.h, comhead.h, m_devcom.h, m_layer.h, m_steerg.h, m_dlg.h, m_xscan.h, c_layer.h
m_scan.cpp	Klassen für die Diffraktometrie und Dialoge für die Scanparameter	rc_def.h, comhead.h, m_devcom.h, m_steerg.h, m_xscan.h, c_layer.h, m_layer.h
m_steerg.cpp	Klassen für die Ablaufsteuerung (Makros)	rc_def.h, comhead.h, m_devcom.h, m_steerg.h, m_xscan.h, c_layer.h, m_layer.h, l_layer.h, m_dlg.h
m_topo.cpp	Klassen für die Topographie	rc_def.h, comhead.h, m_devcom.h, m_layer.h, m_steerg.h, m_topo.h
m_device.cpp	Klasse bzw. Methoden für das Zählerfenster, Klasse: TCounterWindow	rc_def.h, comhead.h, m_devcom.h

m_main.cpp	Hauptfunktion für das Programm, Initialisierung der Bibliotheken, Windows-Nachrichtenschleife (Event Handler)	rc_def.h, comhead.h, m_devcom.h, m_steerg.h, m_topo.h, m_xscan.h, m_dlg.h, help_def.h, st_layer.h, l_layer.h, m_layer.h, c_layer.h
m_dlg.cpp	Implementation verschiedener Dialogklassen unter anderem „Manuelle Justage"	rc_def.h, comhead.h, m_devcom.h, m_steerg.h, m_layer.h, c_layer.h, m_dlg.h
dlg_tpl.cpp	selbstdefinierte Templateklassen für Dialogobjekte inklusive der Methoden	rc_def.h, comhead.h
main.rc	Ressourcen für die Dialoge des Steuerprogrammes	rc_def.h

Da die Entwicklungsumgebung Borland C++ 4.0 nicht mehr zur Verfügung steht, werden jetzt die Versionen Borland C++ 4.5 respektive 5.02 eingesetzt. Zur Compilierung unter Borland 5.02 müssen die Bibliotheken der Version 4.5 benutzt werden. Dazu ist es notwendig, in der Entwicklungsumgebung unter dem Menüpunkt *Optionen/Projekt/Verzeichnisse* den Pfad für die Bibliotheken richtig zu setzen z.B.: `D:\BC4\Lib`. Unter der Version Borland 4.5 sind diesbezüglich keine Änderungen notwendig.

Die Version Borland 4.5 hat außerdem den Vorteil, daß im Gegensatz zu Borland 5.02 ein 16-Bit Debugger in die Entwicklungsumgebung integriert ist. Dies stellt eine enorme Erleichterung bei der Analyse der Quelltexte dar, da man mittels „Breakpoints" das Programm schrittweise ausführen kann. Außerdem können Fehler bei einer Neuimplementation von Funktionalität schneller lokalisiert werden.

Des weiteren dürfen die Dialogressourcen nur mit der Version 4.5 bearbeitet werden. Der Grund dafür liegt darin, daß das Format für die Ressourcendatei in der Version 5.02 geändert wurde. Auf die Dialogerstellung wird in Kapitel 7 näher eingegangen.

Zur fehlerfreien Compilierung mußten an drei Stellen des Quelltextes Änderungen vorgenommen werden. Es genügt, wenn die entsprechenden

Zeilen auskommentiert werden. Es handelt sich um folgende Zeilen in der Datei `m_main.cpp`:

- Zeile 251:
  `Ctl3dRegister((const HINSTANCE__ near*)Main.hWndFrame);`
- Zeile 252:
  `Ctl3dAutoSubclass((const HINSTANCE__ near*)Main.hWndFrame);`
- Zeile 1059:
  `Ctl3dUnregister((const HINSTANCE__ near*)Main.hWndFrame);`

2.3 Hardware

In diesem Abschnitt wird kurz auf die Hardware und ihre Ansteuerung eingegangen.

Aus der Sicht des Programms wurde die Hardwareansteuerung abgekapselt und in eigenständigen Laufzeitbibliotheken (`motors.dll`, `counters.dll`) realisiert. Im Anhang A dieser Arbeit befindet sich eine komplette Beschreibung des Hardwareinterface zur Motorenansteuerung. Diese Schnittstelle wird vom RTK-Steuerprogramm für die Ansteuerung der Motoren verwendet.

Im weiteren soll vorgestellt werden, welche Hardware benutzt und wie sie ins System eingebunden wird, welche Möglichkeiten zur Kommunikation existieren und welche vom Steuerprogramm genutzt werden.

2.3.1 Motoren

Die RTK-Arbeitsplatzrechner sind jeweils mit einer Motoren-Controllerkarte vom Typ C-812 und einer Karte vom Typ C-832 ausgerüstet. Beide Kartentypen sind für den ISA-Bus[2] vorgesehen. Die C-812-Karte kann vier Gleichstrom-Motoren und die C-832-Karte 2 Gleichstrom-Motoren ansteuern. Die Kommunikation zwischen Hostrechner und Controllerkarte kann unterschiedlich erfolgen.

Bei der C-812 Controllerkarte hat man die Wahl zwischen RS232-[3], PC-Bus- und IEEE488-Schnittstelle. Die Kommunikation über die RS232-Schnittstelle funktioniert ähnlich wie die bei einem Modem. Die Steuerbefehle

[2] Abk. für Industrial Standard Architecture. ISA definiert die Busstruktur, die Architektur von CPU und Supportchips sowie die Taktfrequenz des ISA-Bus.

[3] Ein Standard für serielle Schnittstellen, der die Signalpegel, die Signalbedeutung, die Steckerbelegung und die Prozedur für den Aufbau einer Verbindung definiert.

können mit jedem beliebigen Terminalprogramm an die Controllerkarte gesendet werden, wobei die Controllerkarte nicht unbedingt im ISA-Slot der Hauptplatine gesteckt sein muß. Diese Art der Kommunikation wurde im RTK-Steuerprogramm nicht realisiert.

Die Kommunikation mit der PC-Bus-Schnittstelle erfolgt über einen Dual-Port-RAM. Der Dual-Port-RAM ist ein Speicherbaustein, der zwei voneinander unabhängige Zugänge zu den Speicherzellen aufweist. Dadurch können zwei Einheiten gleichzeitig auf die Informationen im RAM zugreifen, ohne sich zu behindern. In diesem Fall wird ein 1 KByte großer Speicherbereich oberhalb von 640 KByte sowohl von der Controllerkarte als auch vom Host-Rechner für Schreib-/Leseoperationen benutzt. Innerhalb dieses Bereichs werden zwei Register zur Kommunikation verwendet. Das Timing wird über ein Handshakeflag realisiert. Ist dieses Handshakeflag[4] gesetzt, liegen Daten im Schreib- bzw. Leseregister an. Für die Kommunikation bedeutet das, daß die Steuerbefehle nacheinander ins Schreibregister geschrieben oder aus dem Leseregister gelesen werden. Dabei muß ständig der Status des Handshakeflags abgefragt werden.

Der Befehlssatz umfaßt ca. 80 Befehle, wobei jeder Befehl aus zwei ASCII-Zeichen besteht. Die zwei Zeichen stehen für eine Abkürzung des auszuführenden Befehls, z.B. Move Absolute=MA. Eine vorangestellte Ziffer bezeichnet den entsprechenden Antrieb, für den der Befehl ausgeführt werden soll. Der nachfolgende Wert steht für eine spezifizierte Kenngrösse wie Geschwindigkeit, anzufahrende Position oder Beschleunigung. Abschließend wird an das Ende des Befehls ein Return in Form des ASCII-Wertes 13 angefügt. Der vollständige Befehl für „bewege Motor 3 zur Position 55000" würde folgende Form haben: „3MA55000<Return>".

Bei der dritten Möglichkeit der Kommunikation, der IEEE-488 Schnittstelle, wird vom Hersteller eine 16-Bit DLL (`win488.dll`) zur Verfügung gestellt. Sie bietet eine strukturierte Schnittstelle zur Ansteuerung der C-812 Motorensteuerkarte. Man kann diese Bibliothek auch als 16-Bit-Treiber bezeichnen. Diese Art der Kommunikation ist ebenfalls im RTK-Steuerprogramm benutzt worden. Die Auswahl der jeweiligen Kommunikation wird in der Initialisierungdatei festgelegt. Unter dem Eintrag `[MotorX]` muß beim Parameter `Type` als Wert „C-812GIPB" eingetragen werden, um den 16-Bit Treiber zu benutzen. Analog ist für die Kommunikation über den PC-Bus der Wert „C-812ISA" anzugeben.

Die Controllerkarte C-832 bietet nur eine Möglichkeit zur Kommunikation. Mittels Port-Befehlen wird auf einen I/O-Adressbereich zugegriffen. Um Kommandos zur Karte zu senden oder von ihr zu lesen, sind folgende zwei

[4]Ein Handshake ist die Einleitung einer Daten- oder Signalübertragung, die durch ein Request-Signal und die Bestätigung durch ein Acknowledge-Signal gekennzeichnet ist.

Schritte notwendig. Als erstes wird ein *Select* Byte in das interne Adressregister geschrieben. Danach wird das Kommando in den gewählten Kanal, der den jeweiligen Motor bezeichnet, geschrieben oder ausgelesen. Der Status der Kommunikation wird über ein entsprechendes Register abgefragt.

Die zentralen Funktionen befinden sich in der Datei `motors.cpp`. Sie realisieren den endgültigen Zugriff auf die Hardware. Tiefergehende Informationen bieten die Dokumente [12] und [13].

2.3.2 Detektoren

Das RTK-Steuerprogramm arbeitet mit unterschiedlichen Arten von Detektoren. Abhängig vom Arbeitsplatz werden die einzelnen Detektoren benutzt, dabei wird zwischen 0- ,1- und 2-dimensionalen Detektoren unterschieden.

Am Topographiearbeitsplatz wird ein einfaches Zählrohr, welches zur Klasse der 0-dimensionalen Detektoren zählt, eingesetzt. Der Detektor wird mittels einer Schnittstellenkarte der Firma Radicon eingebunden. Genau wie bei der C-832 Motorencontrollerkarte wird die Kommunikation über Port-Befehle ausgeführt. Dabei stehen wieder zwei Portadressen zur Verfügung, eine für die Daten und eine für die Kontrollcodes. Die Portadressen werden durch Einstellen von Schaltern auf der Karte festgelegt. Genauere Informationen zur Einbindung und Ansteuerung des Radicon-Zählers befinden sich unter [14].

2.4 Bewertung der Software

Im Softwareengineering wird ein Softwareprodukt anhand von Qualitätskriterien bewertet. Je nachdem, wie gut oder wie schlecht diese Kriterien erfüllt werden, besteht die Möglichkeit der Bewertung der Software. Als Orientierung für die Qualitätsbewertung des RTK-Steuerprogrammes sollen hier die Kriterien, wie sie in [16] beschrieben wurden, dienen. Die Bewertung der Software soll sich auf die Version beziehen, die von der Physik zur Verfügung gestellt wurde.

2.4.1 Korrektheit, Robustheit und Zuverlässigkeit

Korrektheit wird in der Literatur als das Kriterium bezeichnet, welches Aussagen darüber gibt, wie gut die Spezifikation aller Anforderungen an ein Softwareprodukt erfüllt wurde. Entsprechend ist ein Softwaresystem genau dann korrekt, wenn es eine vollständige und fehlerfreie programmtechnische Umsetzung der vorliegenden Aufgabenstellung darstellt.

Das zentrale Problem ist, daß die Frage der Korrektheit im Normalfall nicht entschieden werden kann, schon gar nicht, wenn es wie bei diesem Softwareprojekt keine Produktdefinition gibt. Damit entfällt auch die Programmverifikation auf Basis einer formalen Spezifikation[5]. Als Möglichkeit der Bewertung der Korrektheit kommt also nur noch der Test aller Programmzustände in Frage. Da dies schon für kleinere Projekte äußerst schwierig ist, kann man bei großer Software nie völlige Korrektheit nachweisen. Korrektheit ist also kein praktisch verwertbares Kriterium, sondern eher ein theoretischer Anspruch. Allerdings deckt die Betrachtung der Korrektheit des RTK-Softwareprojektes gravierende Schwächen auf, insbesondere die fehlende Produktspezifikation in Form eines Pflichtenheftes.

Eine Zielsetzung des Softwaresanierungsprojektes war es, diese Mängel auszumerzen. Damit sollte eine Grundbasis geschaffen werden, auf der man aufbauend Teststrategien entwickeln bzw. das Verständnis für das Programm erhöhen kann.

Ein Softwareprodukt ist robust, wenn es bei jeder Form der externen Kommunikation sinnvoll reagiert. Einfache Fehlermeldungen oder Aufforderungen zu einer erneuten Eingabe wären zum Beispiel sinnvolle Reaktionen eines Softwareprodukts auf fehlerhafte Eingaben. Im Fall des RTK-Steuerprogramms kann man die Robustheit als befriedigend bewerten. Auf fehlerhafte Eingaben in den Dialogen wird meistens mit „0"-Werten reagiert. Eine Aufforderung oder einen Hinweis auf eine fehlerhafte Eingabe gibt es allerdings nicht.

Es existieren auch Eingaben, die das Programm zum Absturz bringen. Als Beispiel sollen hier zwei Fehler genannt werden:

1. Wenn man ein Makro, wie zum Beispiel „Halbwertsbreite messen", startet und dann das Makro bei der Bearbeitung unterbricht, kommt es zum Absturz.

2. Ist in dem Verzeichnis von `steerng.exe` keine Datei `standard.mak` vorhanden, dann stürzt das Programm beim Aufruf der Topographie über den Menüpunkt *Ausführen/Topographie...* ab. Dieser Fehler ist inzwischen im Softwaresanierungsprojekt korrigiert worden.

Ein weiteres wichtiges Kriterium von Software ist die Fehleranfälligkeit. Man geht von der Erkenntnis aus, daß jede Software Fehler enthält, also daß in keinem Fall die Existenz von Fehlern ausgeschlossen werden kann. Man spricht von zuverlässiger Software, wenn Fehler selten auftreten und nur geringe Auswirkungen haben.

[5]Die formale Spezifikation kann man als die Weiterführung der Problembeschreibung ansehen.

Danach kann das RTK-Steuerprogramm als zuverlässig betrachtet werden. Es gibt allerdings einige undokumentierte Fehler, die von Mitarbeitern des Instituts für Physik registriert wurden. Der Autor des RTK - Steuerprogramms hatte selbst versucht, einige dieser Fehler zu analysieren und zu beheben, sie waren aber oft nicht reproduzierbar. Er war der Meinung, diese Fehler wären auf eine fehlerhafte Bedienung der Nutzer zurückzuführen, hervorgerufen durch falsche Eintragungen in der Initialisierungsdatei. Dies wäre aber ein Indiz dafür, daß das Programm in Bezug auf die Initialisierungsdatei nicht robust sei.

Eigene Erfahrungen haben in der Tat bestätigt, daß eine große Fehleranfälligkeit von der Initialisierungsdatei ausgeht.

2.4.2 Dokumentation

Man unterscheidet zwei Arten von Dokumentationen:

1. Die *Systemdokumentation* umfaßt alle inneren Aspekte der Software, d.h. operationale und systemnahe Eigenschaften vom Beginn der Entwicklung an bis zur aktuellen Version. Die Systemdokumentation bildet die Grundlage aller Wartungsaktivitäten.

2. In der *Benutzerdokumentation* wird das Verhalten der Software nach außen beschrieben. Dazu gehört eine Übersicht der verfügbaren Funktionalität bzw. ein Handbuch für den Nutzer.

Sind beide Arten von Dokumenten vorhanden und konsistent zur aktuell eingesetzten Version, dann spricht man von einer gut dokumentierten Software.

Zum RTK-Steuerprogramm existierten nur wenige unvollständige Dokumente. Ein Dokument beschreibt einen Teil der Schnittstelle zu den Motoren und Auszüge zu den Parametern der Initialisierungsdatei. Ein weiteres Dokument geht kurz auf die Anwendung des Dialoges „Manuelle Justage" ein. Da weitere Dokumentationen nicht verfügbar waren, wurde die Einarbeitung in die Quellen erheblich erschwert. Hinzu kam, daß die Quelltexte kaum dokumentiert waren. Vorhandene Kommentare gaben wenig Aufschluß für das Verständnis der Programminterna.

2.4.3 Wartbarkeit

Das Kriterium Wartbarkeit faßt alle inneren Eigenschaften von Software zusammen, die das Suchen und Beheben von Fehlern, die Portierung und die Verbesserung der Software unterstützen.

Bezüglich der Wartbarkeit ist das RTK-Steuerprogramm unzureichend. Begründet wird das damit, daß keine Dokumentation der Quellen und des Systems vorhanden waren. Positiv ist anzumerken, daß eine erkennbare Architektur mit Schnittstellen zur Hardware und zur Oberfläche existiert. Die Software besteht aus den drei Schichten „Hardwareansteuerung", „Kernfunktionalität" und „grafische Oberfläche". Die unzureichende Bewertung ergibt sich unter anderem deshalb, weil die Trennung der beiden Schichten „Kernfunktionalität" und „grafische Oberfläche" zu unscharf gestaltet ist. Gerade an dieser Stelle wäre es wünschenswert, wenn es eine schärfere Trennung geben würde, da dies im Hinblick auf die Portabilität enorme Erleichterungen bringt. Mit Portabilität ist dabei nicht unbedingt die Portierung auf das Betriebssystem Linux gemeint, obwohl diese schon einmal ins Auge gefaßt wurde, sondern allein schon die Übertragung in eine andere Entwicklungsumgebung. Wäre diese Schichtentrennung besser gelöst worden, könnte das einige Entwicklungsarbeit sparen.

Irritierend ist außerdem die Verwendung von C++ und C. Obwohl der Großteil der Software in C++ programmiert ist, wurde für die Schnittstellen zur Hardware C als Programiersprache verwendet. Dadurch wurde das Programm unübersichtlich und unnötig aufgebläht - sprich die Lesbarkeit des Quelltextes wurde verschlechtert.

2.4.4 Benutzungsfreundlichkeit

Unter Benutzerfreundlichkeit versteht man, daß ein Softwareprodukt Benutzern eine komfortable Bedienung ermöglicht. Wichtige Kriterien, die an eine benutzerfreundliche Software gestellt werden, sind eine ergonomische Benutzeroberfläche, die permanente Verfügbarkeit von Hilfefunktionen und die Anpassung der Art der Interaktionen an den Kenntnisstand des Nutzers.

Das RTK-Steuerprogramm besitzt keine permanente Hilfefunktion. Lediglich zum Dialog „Manuelle Justage" gibt es eine kleine Hilfe, die die Benutzung der Tastaturkürzel erklärt. Die Benutzeroberfläche ist nicht in allen Dialogfenstern ergonomisch gestaltet. Oft wurden Schalter mit gleicher Funktionalität, aber unterschiedlicher Bezeichnung verwendet, z.B. „Abbruch" und „Beenden". Des weiteren gibt es Dialogfenster, in denen Funktionalität, die nicht existiert, assoziiert wird. Als Beispiel ist hier der Schieberegler auf der Laufleiste im Dialog „Manuelle Justage" zu nennen.

2.4.5 Zusammenfassung der Bewertung

Die Bewertung des RTK-Steuerprogrammes mit Hilfe der Qualitätskriterien deckt gravierende Mängel auf. Die folgende Übersicht faßt die Mängel zu-

sammen:

- fehlende Dokumente:
  - Produktdefinition
  - Pflichtenheft
  - Systemdokumentation
  - Benutzerhandbuch
  - unvollständige Beschreibung der Initialisierungsdatei
- Mängel am Quelltext:
  - keine bzw. schlechte Kommentierung der Quellen
  - ungünstige Strukturierung der Quellen
  - keine durchgängige objektorientierte Implementierung
- Mängel aus Benutzersicht:
  - keine permanente Hilfe
  - Fehleranfälligkeit
  - unzureichende Ergonomie der Oberfläche.

Im Rahmen des Softwaresanierungsprojektes wurden bereits einige dieser Mängel beseitigt. Dabei standen die Aufgaben Analyse vorhandener Dokumente, Kommentierung der Quellen sowie die Erstellung von Entwicklerdokumenten im Vordergrund. Entstanden sind Pflichtenhefte einzelner Aufgabenbereiche, die später zu einem Pflichtenheft zusammengefaßt werden sollen.

Ziel des Softwaresanierungsprojektes ist es, sämtliche Mängel zu beseitigen und das Steuerprogramm um Funktionalität zu erweitern. Die dringlichsten Aufgaben bezüglich der Erweiterung der Software sind die Einbindung von neuen Detektoren und die Automatisierung der manuellen Justage der Meßprobe.

Kapitel 3

Topographie mit dem Steuerprogramm

Schwerpunkt in Bezug auf eine erweiterte Funktionalität des Steuerprogrammes ist die Möglichkeit der automatischen Justage der Meßprobe. Diese Erweiterung ist der zentrale Aspekt dieser Arbeit. Voraussetzung dazu ist die Kenntnis, wie die manuelle Justage bzw. der Topographievorgang mit Hilfe des Steuerprogrammes durchgeführt wird.

3.1 Manuelle Justage

Die Topographie umfaßt das Justieren der Meßprobe und den Topographievorgang. Die Justage der Meßprobe ist eine komplexe Aufgabe. Um einen Röntgentopographie-Meßvorgang durchzuführen, muß die Meßprobe so justiert werden, daß die Strahlungsintensität, gemessen am Detektor, am höchsten ist. Ist dies geschehen, hat man den Bragg-Reflex optimal getroffen. Allerdings gibt es keine Grenzwerte, anhand derer man bestimmen kann, ob die maximale Strahlungsintensität gefunden wurde. Beim manuellen Einstellen der Meßprobe kann man nur von Erfahrungswerten für die Höhe der Strahlungsintensität ausgehen. Als Qualitätskriterium dient die Halbwertsbreite der Rockingkurve nach der Justage. Da die Halbwertsbreiten für die einzelnen Meßproben vorher schon bekannt sind, kann mit ziemlicher Sicherheit festgestellt werden, ob der Bragg-Reflex korrekt eingestellt wurde.

Die Schwierigkeit der Justage der Meßprobe liegt darin, daß die Probe frei im Raum gedreht werden muß, um den Bragg-Reflex optimal einzustellen. Hinzu kommt die Einstellung für die Krümmung des Kollimators. Da so gut wie jede Probe eine leichte Krümmung aufweist, ist die Einstellung des Kollimators unerläßlich. Damit hat man 4 Freiheitsgrade, die korrekt eingestellt werden müssen.

Bisher muß die Justage der Meßprobe mit Hilfe des Steuerprogramms manuell durchgeführt werden. Dazu steht im Steuerprogramm ein Dialog zur Verfügung. Anhand dieses Dialogs soll erläutert werden, wie die manuelle Justage der Probe funktioniert.

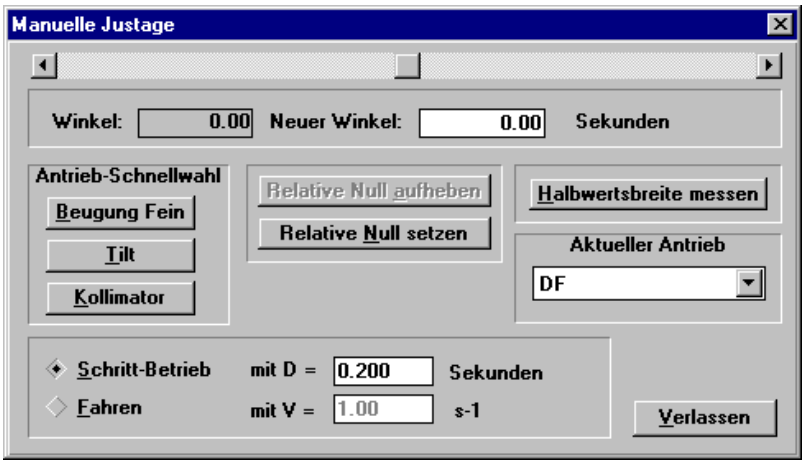


Abbildung 3.1: Dialogfenster „Manuelle Justage"

Über das Popup-Menü *Ausführen/Manuelle Justage...* im Hauptfenster der Anwendung wird der Dialog „Manuelle Justage" (Abb. 3.1) ausgewählt. Mit Hilfe dieses Dialoges werden die Antriebe für den Probenteller und den Kollimator angesteuert. Auf eine Erklärung der gesamten Funktionalität des Dialoges wird hier verzichtet, da das Pflichtenheft zur Funktion „Probe und Kollimator manuell justieren" [4] genaue Informationen dazu enthält.

Als erstes wird die Probe auf den Probenteller gelegt. Für die richtige Positionierung auf dem Probenteller werden die Informationen, die zu jeder Meßprobe existieren, benutzt. Dabei ist die Probe schon durch bestimmte feste Parameter wie die Röntgenwellenlänge, die Gitterparameter und die Netzebene, vorbestimmt. Danach wird anhand des manuell berechneten Einfallswinkels der Detektor per Hand eingestellt. Nun wird der Detektor eingeschaltet und fängt an zu zählen. Das Zählen des Detektors kann auch akustisch wahrgenommen werden, da die Kontrollerkarte des Detektors mit einem Piezolautsprecher ausgerüstet ist. Dieser wird über den Dialog (Abb. 3.2), zu erreichen unter dem Menüpunkt *Einstellungen/Detektor/Detektoren...*, aktiviert.

Des weiteren muß beachtet werden, daß der Antrieb Tilt(Verkippung) die Position TL=0 hat. Dazu wird im Dialog „Manuelle Justage" der Antrieb Tilt im Pulldown-Menü *„Aktueller Antrieb"* ausgewählt und im Direktbe-

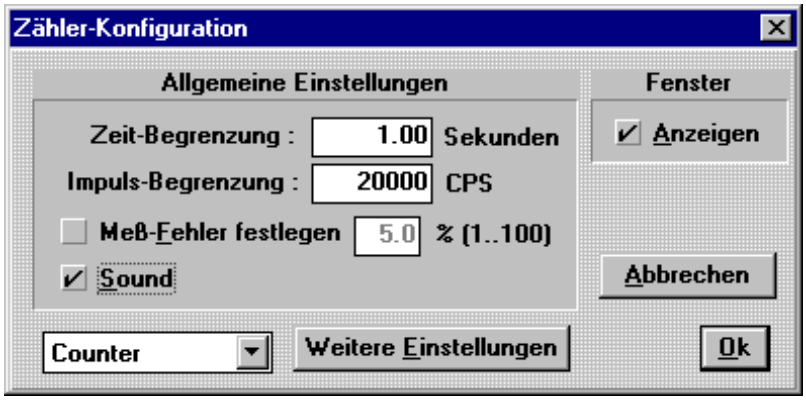


Abbildung 3.2: Dialogfenster „Zählerkonfiguration"

trieb[1] an die Position 0 gefahren. Als nächstes wird der Freiheitsgrad für die Beugung mit zwei Stellrädchen (Abb. 1.9) um 2 bis 3 Grad verändert, so daß ein kurzes lautes Schnarren im Detektor vernehmbar ist. Das ist das charakteristische Geräusch, wenn der Detektor den Bragg-Reflex erfaßt. Allerdings braucht zu diesem Zeitpunkt das Rauschen nicht permanent vernehmbar zu sein. Es reicht aus, wenn man das Schnarren kurz wahrnimmt. Man befindet sich dann in der näheren Umgebung des Bragg-Reflexes. Genaueres Positionieren gelingt per Hand nicht. Jetzt kann man nur noch mit den Antrieben des Probentellers arbeiten.

Im Dialog „Manuelle Justage" wählt man nun den Antrieb „Azimutale Rotation" aus. Dieser wird im Fahrbetrieb[2] mit höchster Geschwindigkeit in eine Richtung bewegt, wobei nach kurzer Zeit das Zählen des Detektors schneller werden und bis hin zum Rauschen führen muß. Ist dies nicht der Fall, muß die andere Richtung abgesucht werden. Wurde ein Peak gefunden, wird der zweite Peak bestimmt. Das heißt, daß der Antrieb „Azimutale Rotation" weiter bewegt werden muß. Findet man in der einen Richtung den zweiten Peak nicht, wird die Richtung geändert. Ist er gefunden, ermittelt man den Mittelpunkt zwischen beiden Peaks, indem die relative Null im Dialog „Manuelle Justage" auf einen Peak gesetzt wird und dann zum zweiten Peak gefahren wird. Dann fährt man mit dem Antrieb „Azimutale Rotation" den halben Abstand zwischen den beiden Peaks an und schließt damit die Justage für den Antrieb „Azimutale Rotation" ab.

[1]Mit Direktbetrieb ist gemeint, daß unter „neuer Winkel" eine Position eingegeben wird und anschließend nach Bestätigung mit <Enter> der Motor diese Position anfährt.

[2]Der Fahrbetrieb bezeichnet den Betriebsmodus, bei dem durch Drücken der Pfeiltasten der entsprechende Motor bewegt wird, wobei im Dialog „Manuelle Justage" die Checkbox „Fahren" ausgewählt sein muß.

Als nächstes regelt man mit dem zuvor ausgewählten Antrieb „Beugung grob" nach. Dabei muß wiederum auf die maximal mögliche Intensität gestellt werden, wobei auf das Geräusch vom Lautsprecher der Zählerkarte geachtet werden sollte, denn die Einstellungen nach Gehör gestalten sich in diesem Stadium der Justage einfacher als die Einstellung nach den Zählerwerten im Zählerfenster.

Nun beginnt die Feinjustage. Sie nimmt den größten Teil der Zeit bei der manuellen Justage der Meßprobe ein. Dabei kann man von einem iterativen Prozeß sprechen, da es sich um immer zu wiederholende Ablaufschritte handelt.

Zuerst wird der Antrieb „Beugung fein" nach dem Detektorwert eingestellt. Hat man die maximale Intensität erreicht, wird auf den Antrieb „Kollimator" gewechselt. Der Kollimator wird so verstellt, daß man wiederum eine Steigerung der Strahlungsintensität erreicht. Danach stellt man mit dem Antrieb „Beugung fein" nach. Genauso wie beim Kollimator verfährt man beim Freiheitsgrad Tilt. Der Antrieb Tilt wird ausgewählt und dann so verstellt, daß die Strahlungsintensität wieder maximal ist. Mit „Beugung fein" regelt man dann wieder nach. Dieser Prozeß wird ungefähr 20-100 Mal wiederholt. Zwischendurch wird bei fast optimaler Justage zur Kontrolle die Halbwertsbreite bestimmt, wobei man dort von Erfahrungswerten ausgeht. Ziel der manuellen Justage ist es, eine maximale Strahlungsintensität bei minimaler Halbwertsbreite zu erreichen.

Ist das geschehen, kann der Topographievorgang gestartet werden.

3.2 Topographievorgang

Die Meßprobe ist nun so genau auf den Bragg-Reflex eingestellt, daß mit der Belichtung der Fotoplatte begonnen werden kann. Dazu wird eine Fotoplatte in die dafür vorgesehene Vorrichtung eingelegt (Abb. 1.9).

Der Topographievorgang wird mit Hilfe des Dialoges „Topographie", zu finden im Hauptmenü unter *Ausführen/Topographie...* , gesteuert.

Von diesem Dialog(Abb. 3.3) können weitere vorbereitende Maßnahmen, die für die Belichtung wichtig sind, ausgeführt werden. Zunächst muß mit dem Antrieb „Beugung fein" der Arbeitspunkt, angefahren werden. Dieser liegt bei ca. 60 Prozent auf der steileren der beiden Flanken der Rockingkurve(Abb. 1.2). Der Grund dafür, warum man nicht den Peak der Rockingkurve als Position für die Topographie beibehält, liegt darin, daß thermische Einflüsse auf die Meßapparatur einwirken. Es kommt zu Deformationen, wobei der Meßpunkt vom Peak wegwandert und dadurch eine optimale Einstellung nicht mehr gewährleistet ist.

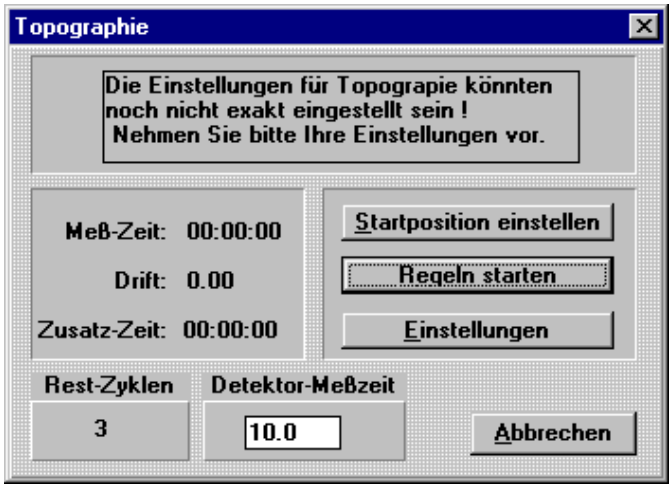


Abbildung 3.3: Dialogfenster zum Starten des Topographievorgangs

Da bei geringer Manipulation der Position um den Peak keine große Intensitätsänderung erkennbar ist, würden kleine Positionsabweichungen vom Peak nicht ausreichend schnell erfaßt werden. Das kann sehr gut anhand der Abbildung der Rockingkurve (Abb. 1.2) nachvollzogen werden[3]. Außerdem ist nicht ersichtlich, in welche Richtung der Arbeitspunkt wandert. Um mit Hilfe des Detektors angemessen auf die Veränderungen während der Belichtung zu reagieren, wählt man den Arbeitspunkt an der steileren Flanke der Rockingkurve. In der Regel ist das die linke Flanke. Kleinste Veränderungen der Position werden sofort von der Steuersoftware mittels des Detektors erkannt. Durch schnelles Nachregeln der Position wird angemessen reagiert, so daß keine Abweichungen von der optimalen Einstellung der Meßprobe erfolgen. Außerdem kann die Steuersoftware schnell erkennen, in welche Richtung der Arbeitspunkt wandert, was eine wichtige Information im Hinblick auf die zu erfolgende Reaktion des Programms auf diesen Umstand darstellt.

Da sich der Arbeitspunkt nicht auf dem Peak befindet, bedeutet dies allerdings eine höhere Belichtungszeit der Fotoplatte, da die Intensität der reflektierten Röntgenstrahlung von der Meßprobe nun viel geringer ist als auf dem Peak. Bei einem Arbeitspunkt bei 60 Prozent der Höhe der Rockingkurve wirken also nur noch 60 Prozent der Intensität der Röntgenstrahlung des Peaks.

Der Arbeitspunkt wird über Anwählen der Schaltfläche „Startposition einstellen" angefahren.

Über die Schaltfläche „Einstellungen" erreicht man den Dialog(Abb. 3.4),

[3]Im Bereich der höchsten Intensität ist die Kurve flach. Man spricht von einem Plateau.

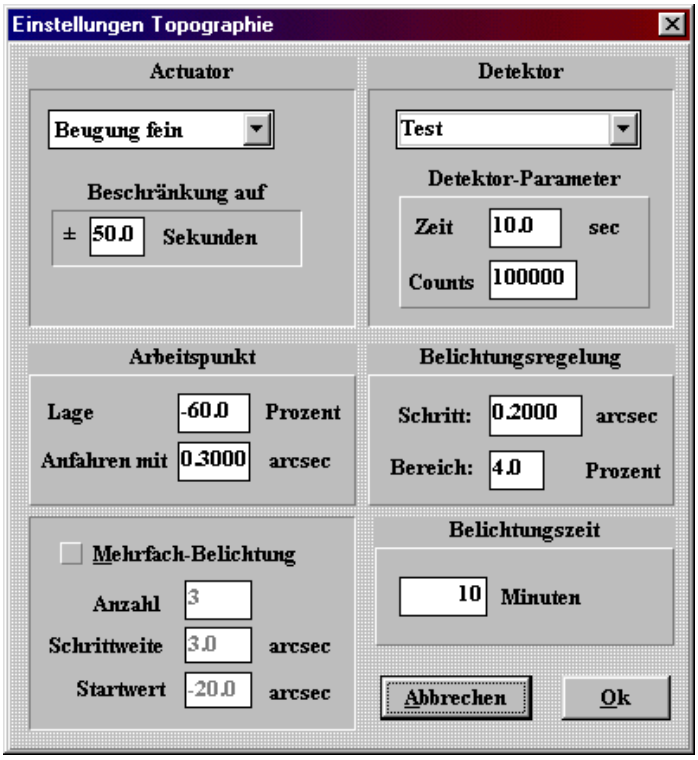


Abbildung 3.4: Dialogfenster zum Einstellen der Topographieparameter

in dem alle relevanten Parameter für den Topographievorgang eingestellt werden. Unter Arbeitspunkt wird angegeben, bei wieviel Prozent der Höhe der Rockingkurve der Arbeitspunkt liegen und mit welcher Schrittweite(in Winkelsekunden) er angefahren werden soll.

Der „Actuator" bezeichnet den Motor, der den Freiheitsgrad „Beugung fein" verändert. In der Regel ist das immer der Antrieb „Beugung fein". Zusätzlich dazu wird ein Bereich angegeben, den der Arbeitspunkt nicht verlassen darf, ansonsten wird der Meßvorgang abgebrochen.

Unter „Detektor" wird der jeweilige Zähler ausgewählt und die Zählzeit sowie die maximale Impulsrate festgelegt. Bei „Belichtungsregelung" wird die Schrittweite angegeben, mit der der Antrieb „Beugung fein" bei einer Veränderung der Intensität korrigiert werden soll. Dazu wird ein Toleranzbereich(in Prozent) vorgegeben, in dessen Grenzen eine Veränderung der Intensität noch nicht als ein Wandern des Arbeitspunktes interpretiert wird.

Für stark gekrümmte Meßproben muß eine Mehrfachbelichtung durchgeführt werden. Unter dem Auswahlkästchen „Mehrfachbelichtung" wird diese Art der Topographie ausgewählt. Zusätzlich werden die Anzahl der Mehr-

fachbelichtungen und der Abstand zwischen den jeweiligen Arbeitspunkten sowie ein Startwert angegeben.

Entgegen der Information, die sich aus der Abb. 3.4 unter dem Punkt „Belichtungszeit" entnehmen läßt, dauert eine Topographieaufnahme ca. 8 bis 12 Stunden. Allerdings kommt es darauf an, was für eine Probe benutzt wird, wie hoch die Peak-Intensität ist oder welche Belichtungsart verwendet wird.

Vom Dialog „Topographie" kann nun der Belichtungsvorgang durch Anwählen des Schalters „Regelung starten" begonnen werden. Dabei beginnt die Meßzeit zu zählen. Der Wert „Drift" gibt an, wie weit sich der Arbeitspunkt von der Ausgangsposition entfernt hat. Die „Zusatz-Zeit" zeigt die über die vorgegebene Meßzeit verstrichene zusätzliche Zeit an. Wird die Mehrfachbelichtung benutzt, kann unter „Rest-Zyklen" die Anzahl der noch zu erfolgenden Belichtungen abgelesen werden. Des weiteren wird die Detektor-Meßzeit ausgegeben.

Nach Ablauf des Topographievorganges ist die Fotoplatte korrekt belichtet, und es können Schwarz-Weiß-Vergrößerungen von der Aufnahme angefertigt werden (Abb. 3.5). Diese Vergrößerungen, auf denen die Struktur der Meßprobe abgebildet ist, werden dann ausgewertet.

Abbildung 3.5: In einem Schichtsystem erzeugte Inseln

Kapitel 4

Anforderungsdefinition „Automatische Justage”

4.1 Ausgangspunkt

Im April 1999 sind von Prof. Dr. Köhler in Zusammenarbeit mit seinen Mitarbeitern Anforderungen an das RTK-Steuerprogramm spezifiziert worden [9]. Demnach besteht die Hauptforderung in der Stabilität und Funktionstüchtigkeit der bestehenden Software. Einen hohen Stellenwert nimmt weiterhin die Definition und Dokumentation einer geeigneten Hardwareschnittstelle zur Erleichterung der Einbindung neuer Geräte wie Motorsteuerkarten oder Detektoren ein. Bei der Erweiterung der Funktionalität wurde die Implementierung einer automatischen Optimierung (Justage) im Rahmen der Topographie priorisiert. Das Hauptaugenmerk der Softwaresanierung sollte somit auf die Quelltextanalyse mit Fehlersuche gelegt werden, zusätzlich aber auch auf die Neuprogrammierung der gewünschten Funktion gerichtet sein.

4.2 Problemdarstellung

Bei der bisherigen Einstellung der Probe mittels des Dialogs „Manuelle Justage” handelt es sich um eine zeitaufwendige Routineaufgabe, die auch Erfahrung und Expertenwissen voraussetzt. Es sind nacheinander die Antriebe der verschiedenen Achsen so einzustellen, daß sich die Probe im Bereich der Maximalintensität befindet. Dazu müssen die Antriebe „Beugung grob” und „Beugung fein”, „Azimutale Rotation”, „Verkippung” und „Kollimator” angesteuert werden und gleichzeitig muß auf die Intensitätsänderungen der Röntgenstrahlung geachtet werden, um so das Maximum zu finden.

Dieser Prozeß ist dafür prädestiniert, von einer automatischen Programm-

funktion übernommen zu werden, da nach der Grobeinstellung des Bragg-Reflexes - wie in der Beschreibung der manuellen Justage in Kapitel 3 ausgeführt - zwischen 50 bis 100 Mal die selben Schritte wiederholt werden, bis das Strahlungsmaximum erreicht ist. Ein Mitarbeiter der Physik-Arbeitsgruppe benötigt im Durchschnitt für diese Aufgabe 15-25 Minuten, die er auch mit anspruchsvollerer Arbeit füllen könnte.

4.3 Anforderungsspezifikation

Es wird also gefordert, daß der Prozeß der Meßprobeneinstellung durch eine neu zu implementierende Funktion des Steuerprogramms abzuarbeiten ist.

Diese Funktion muß nach einer groben manuellen Einregelung des Bragg-Reflexes vollautomatisch die für die Feinjustage benötigten Antriebe ansteuern und unter Berücksichtigung der Röntgenstrahlenintensität die Probe so positionieren, daß die Strahlungsintensität, gemessen am 0-dimensionalen Detektor, einen maximalen Wert annimmt. Der Nutzer kann nach Beendigung der automatischen Justage anhand des Wertes der Halbwertsbreite die Güte der Einstellung beurteilen.

Diese groben verbalen Anforderungen müssen nun in einem Pflichtenheft verfeinert und gegliedert werden. Das Pflichtenheft dient dazu, die fachlichen Anforderungen an die zu entwickelnde Software-Funktion aus der Sicht des Auftraggebers zusammenzufassen. Es wird zur Einschätzung des fertigen Programms herangezogen und sollte deshalb die Funktionalität in umfassender Weise beschreiben [1, Seite 104f.].

Das Pflichtenheft sollte gemäß den Richtlinien des *ANSI/IEEE Standard 830-1984* zu diesem Zweck die folgenden Informationen in übersichtlicher Weise enthalten (nach [1, Seite 105, Tab. 2.3-1]):

- Funktionale Anforderungen,
- Leistungsanforderungen,
- Entwurfsrestriktionen,
- Qualitätsmerkmale,
- Externe Schnittstellen-Anforderungen.

Der folgende Abschnitt befaßt sich mit dem Pflichtenheft, in dem die Anforderungen an die Programmfunktion „Automatische Justage” detailliert und vollständig beschrieben sind.

4.4 Pflichtenheft

Die Gliederung des Pflichtenhefts folgt einer eigenen Numerierung, die nicht die Einteilung der Diplomarbeit beeinflußt.

Pflichtenheft: RTK-Steuerprogramm
Funktion: „Automatische Justage"
Dokumentversion: 1.1 (Mai 2000)
Dokumentstatus: abgeschlossen

Die automatische Justage soll Teil des Steuerprogramms zur Röntgentopographie werden. Sie ist als eigenständige Funktion zu implementieren und muß als Menüpunkt aufrufbar sein.

1 Aufgabe

Mit der Steuerprogrammfunktion „Automatische Justage" soll der Vorgang der Probenjustierung für die Topographie von der Steuersoftware durchgeführt werden. Nachdem vom Benutzer über den Dialog „Manuelle Justage" die Probe soweit eingestellt wurde, daß der Bragg-Reflex für einen Bereich der Probe gefunden wurde, wird der weitere Ablauf der Justage vom RTK-Programm übernommen.

Die automatische Justage muß den Probenteller mit der Kristallprobe durch Ansteuerung der Antriebsachsen „Tilt" und „Beugung fein" so positionieren, daß die Röntgenstrahlung, die mittels der Motorachse „Kollimator" in ihrer Ablenkung beeinflußt wird, die Probe komplett mit der höchstmöglichen Strahlungsintensität ausleuchtet. In diesem Fall ist die Bragg-Bedingung global über der gesamten Meßprobe erfüllt und der Peak eingestellt.

Die charakteristische Halbwertsbreite für die zu justierende Probe sollte dann möglichst klein sein. Die Güte der Justage läßt sich durch diese Größe angeben.

1.1 Mußkriterien

Die Funktion „Automatische Justage" muß als gesonderter Menüpunkt unter dem Menü *Ausführen* erscheinen. Die Aktivierung des Menüpunktes ruft einen Dialog auf, der zur Steuerung der automatischen Justage dient.

Vor dem Start der automatischen Funktion muß geprüft werden, ob die Antriebe, die für den Justageprozeß angesteuert werden müssen, korrekt vom System eingerichtet wurden. Außerdem muß kontrolliert werden, ob der 0-dimensionale Röntgendetektor funktionsbereit ist.

Es wird gefordert, daß die Justage durch die neu implementierte Programmfunktion zuverlässig und in angemessener Zeit ein Ergebnis liefert. Als Zeitrahmen wird ein Richtwert von 30 Minuten angesetzt.

Zur Beeinflussung des Justageverhaltens sollen dem Nutzer Parameter angeboten werden, die unter anderem angemessene Abbruchkriterien darstellen. Diese sollen verhindern, daß das Steuerprogramm auch dann noch läuft, wenn durch weitere Änderung der Probenstellung oder der Kollimatorkrümmung keine Verbesserung des Justageergebnisses absehbar ist.

Es wird weiterhin verlangt, daß im Dialogfenster die anzusteuernden Intervalle der am Justageprozeß beteiligten Achsen eingrenzbar sein sollen. Werden diese vom Benutzer nicht verändert, müssen angemessene Werte als Voreinstellung eingesetzt werden.

Bei der Ansteuerung der Motoren sind die Softwareschranken einzuhalten. Das bedeutet, daß die automatische Justage nur im Rahmen der durch den Meßplatz vorgegebenen Winkelintervalle durchgeführt werden darf. Andernfalls könnte die Apparatur durch das Anfahren ungültiger Postionen beschädigt werden.

Während der automatischen Justage ist von der Programmfunktion ein Statusfenster darzustellen, in dem der Benutzer darüber informiert wird, in welchem Stadium sich der Prozeß befindet. Da der 0-dimensionale Detektor nur maximal 100000 Röntgenstrahlungsimpulse pro Zeiteinheit registrieren kann, muß durch die Programmfunktion bei Überschreitung der Zählrate ein entsprechender Warnhinweis mit der Aufforderung, die Leistung des Röntgenapparats zu verringern, ausgegeben werden. Die Justage muß an dieser Stelle abgebrochen werden.

Zum Dialog der automatischen Justage soll eine Hilfefunktion vorhanden sein, die die zu verändernden Parameter erläutert.

1.2 Wunschkriterien

Es sollte die Möglichkeit bestehen, daß Parameter, die z.B. Eigenschaften der Probe betreffen, vom Benutzer an die Programmfunktion übergeben werden können.

Hier wären allgemeine Angaben zur Krümmung der Probe (konvex oder konkav), aber auch genaue Werte der Probenkrümmung denkbar. Diese Informationen könnten während der automatischen Justage zur Verkürzung des Optimierungsprozesses herangezogen werden und auch dazu dienen, den Kollimatorkristall schon von vornherein nur in die erforderliche Richtung zu krümmen. Dadurch könnte dann verhindert werden, daß der Halbleiter durch Anfahren positiver und negativer Krümmungswinkel übermäßig beansprucht wird.

- *Dieses Kriterium ist in der Programmfunktion **nicht** realisiert worden. Das Prinzip des entwickelten Justagealgorithmus' kommt ohne Angaben über Probeneigenschaften aus und bietet deshalb dafür keine Einflußmöglichkeiten.*

Das Protokollieren des Justagevorgangs sollte ebenfalls möglich sein. Die einzelnen angefahrenen Motorpositionen mit den dazugehörigen gemessenen Röntgenstrahlungsintensitäten werden dazu in einer Protokolldatei abgespeichert. Der Benutzer bestimmt selbst, ob die Optimierung protokolliert wird.

- *Dieses Kriterium ist in der Programmfunktion realisiert worden. Die Protokollierung kann innerhalb des Dialogs aktiviert werden.*

2 Anwendungsszenario

Es folgt eine Darstellung des Funktionsverhaltens aus Benutzersicht.

Der Anwender wählt aus dem Hauptmenü des RTK-Steuerprogramms den Punkt *Ausführen/Automatische Justage...* aus. Dadurch wird der Dialog „Automatische Justage" gestartet. Wenn die für die automatische Justage notwendigen Antriebe oder der 0-dimensionale Detektor vom Hauptprogramm nicht korrekt initialisiert wurden, wird im Textfeld des Dialogs eine entsprechende Meldung ausgegeben und der „Start"-Button deaktiviert. In diesem Fall ist es nicht möglich, eine Justage durchzuführen.

Andernfalls können die Voreinstellungswerte der Dialogparameter verändert werden. So ist es möglich, Abbruchkriterien wie die Anzahl der Wiederholungen festzulegen, die Such-Intervalle für die Justageachsen einzustellen oder zu bestimmen, ob die Justage in einer Log-Datei protokolliert werden soll. Um eine Erläuterung zu den einzelnen Parametern zu erhalten, kann der Hilfe-Button gedrückt werden.

Durch Aktivierung des Start-Buttons wird der Vorgang der automatischen Justage gestartet. Der Prozeß läuft selbständig ab, so daß der Anwender während der Optimierung nicht mehr einzugreifen braucht. Die Justage ist bei Erreichen des Maximums der Strahlungsintensität, im Fall des Eintretens einer vom Benutzer festgelegten Abbruchbedingung (z.B. der Anzahl der Wiederholungen) oder bei Überschreitung der zulässigen Zählimpulse für die Strahlung beendet.

Der Dialog kann durch Drücken des „Beenden"-Buttons verlassen werden. Wurde die Justage erfolgreich abgeschlossen, läßt sich die Güte der Optimierung anhand des Wertes der Halbwertsbreite beurteilen. Die Halbwertsbreite kann über den Menüpunkt *Ausführen/Manuelle Justage...* und anschließendes Drücken des Steuerknopfes „Halbwertsbreite messen" bestimmt werden.

3 Funktionsbeschreibung

Um die Funktionsweise der automatischen Justage zu erläutern, ist es notwendig, daß die Vorgehensweise der manuellen Justage noch einmal kurz rekapituliert wird. In Abb. 4.1 ist der Vorgang schematisch dargestellt.

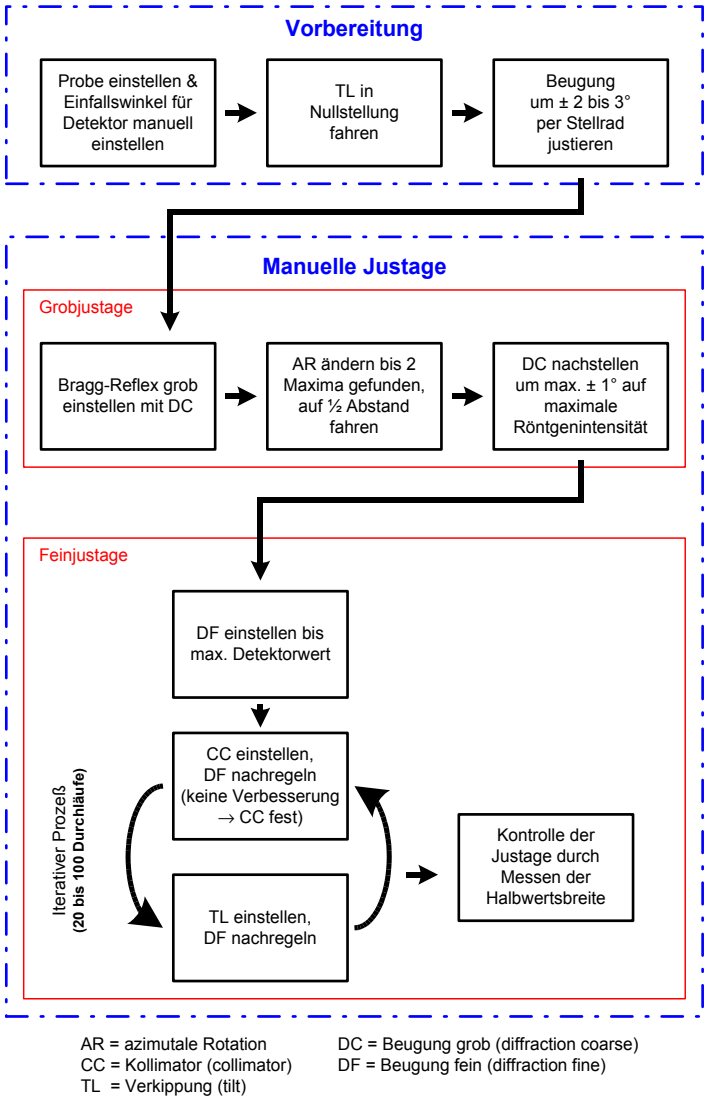


Abbildung 4.1: Ablaufschema der manuellen Justage

Die automatische Justage umfaßt nur die Schritte des iterativen Justageprozesses, also diejenigen Schritte, die die Achsen „Tilt”, „Beugung fein” und „Kollimator” verändern.

Nach Eingabe der Dialogparameter optimiert die Funktion automatisch die Lage der Kristallprobe zur Röntgenstrahlung. Dazu wird zuerst der Antrieb „Kollimator" so justiert, daß ein maximaler Intensitätswert im Röntgendetektor gemessen wird. Mit „Beugung fein" wird so nachgeregelt, daß die Strahlungsintensität nicht abfällt. Im darauf folgenden Optimierungsschritt wird die Achse „Tilt" verstellt, bis die Strahlung ein weiteres Maximum erreicht. Der Antrieb „Beugung fein" dient wiederum zur Nachregelung.

Der gesamte Prozeß besteht aus der Wiederholung dieser beiden Schritte. Nach jedem Durchlauf werden die Abbruchkriterien getestet und die Justage wird gegebenenfalls beendet. Im Fehlerfall wird eine entsprechende Nachricht für den Nutzer generiert.

4 Eingaben

Der Benutzer kann im Dialog „Automatische Justage" folgende Parameter zur Steuerung der Funktion verändern:

- Anzahl der Wiederholungen der Schritte des iterativen Prozesses,
- Maximale Röntgenintensitätsdifferenz,
- Suchintervalle für die Achsen „Beugung fein", „Tilt" und „Kollimator".

Außerdem hat der Anwender die Möglichkeit, durch einen Schalter anzugeben, ob die Justage protokolliert werden soll oder nicht.
Zum besseren Nachvollziehen der Gestaltung des Dialogfensters sei auf den Anhang dieses Pflichtenheftes verwiesen, in dem eine Abbildung des Dialogfensters zu finden ist, das anhand der anschließenden Anforderungen entworfen wurde.
Die einzelnen Optionen werden im folgenden näher erläutert.

4.1 Protokolldatei

Bei Auswahl dieser Option werden alle wichtigen Informationen der automatischen Justage in die Protokolldatei `justage.log` geschrieben. Diese Datei befindet sich im selben Verzeichnis wie das RTK-Programm. Sollte sie noch nicht existieren, wird die Datei neu erstellt. Ansonsten wird die Logdatei mit jeder Neuausführung der automatischen Justage sukzessive verlängert, wobei jede Justage mit Datum und Uhrzeit protokolliert wird. Das Log enthält die von den Antrieben angefahrenen Positionen mit den an diesen Stellen gemessenen Strahlungswerten und Angaben über das jeweilige Stadium des Justageprozesses. Im Dialogfenster wurde diese Option mit „Logdatei" benannt (siehe Anhang dieses Pflichtenheftes).
Ist die Checkbox aktiviert, wird die Protokolldatei geschrieben.

4.2 Wiederholungen

Die Anzahl der Wiederholungen gibt an, wie oft die Schritte des iterativen Prozesses durchlaufen werden. Da es keinen geeigneten Nachweis dafür gibt, ob es sich bei einer bestimmten Kristallprobenstellung um den Peak handelt, müssen andere Abbruchverfahren angewendet werden. Eine Möglichkeit ist die Annahme, daß nach einer bestimmten Anzahl von Durchläufen der Peak erreicht wird bzw. sich in unmittelbarer Umgebung befindet. Ist die Justage nach Erreichen der Wiederholungen nicht erfolgreich, ist vom Anwender ein höherer Wert zu wählen. Im Dialogfenster wurde für diesen Parameter die Bezeichnung „Durchläufe" gewählt (siehe Anhang dieses Pflichtenheftes).

Angemessene Werte für den Parameter können erst später durch entsprechende Testläufe in der Praxis festgelegt werden.

4.3 Maximale Intensitätsdifferenz

Die maximale Intensitätsdifferenz gibt an, um wieviel kleiner die Intensität im Vergleich zum vorherigen Durchlauf sein darf. Ist diese Intensitätsschranke unterschritten, wird zum letzten maximalen Intensitätspunkt aus dem vorherigen Durchlauf zurückgekehrt und die automatische Justage beendet. Damit existiert ein weiteres Abbruchkriterium. Darüber hinaus kann es passieren, daß der Algorithmus zwischenzeitlich durch Meßfehler verursachte Abweichungen von der Maximalrichtung ansteuert. Diese „Ausreißer" sollen durch die Prüfung der Intensitätsdifferenz erkannt werden.

Die Prüfung der maximalen Intensitätsdifferenz kann separat aktiviert werden. Eine Angabe für den Wertebereich der Differenz wird beim praktischen Funktionstest ermittelt.

4.4 Suchbereich

Der Suchbereich gibt an, in welchen Intervallgrenzen der Peak auf den einzelnen Achsen gesucht werden soll und bezieht sich auf die Stellung der Motoren zu Beginn der automatischen Justage. Das Programm testet, ob diese Such-Intervalle die Softwareschranken der Motoren überschreiten. Ist dies der Fall, werden die Grenzen intern angepaßt, so daß die Motoren nicht über die Schranken gefahren werden. Die Werte des Suchbereichs geben die Grenzen zu beiden Richtungen an und sind für jeden an der Justage beteiligten Antrieb („Beugung fein", „Tilt", „Kollimator") anzugeben.

Die Wertebereiche müssen beim späteren Test der Funktion anhand verschiedener Proben entsprechend ermittelt werden.

5 Voreinstellungen

Die Voreinstellungen der Dialogparameter (Defaultwerte) müssen in einer ausgedehnten Testreihe am RTK-Arbeitsplatz ermittelt werden. Dazu sind mehrere Proben aus unterschiedlichen Probenklassen zu untersuchen, um ausreichende Erfahrungswerte zu sammeln. Die Forderung besteht darin, Werte zu ermitteln, bei denen die automatische Justage ohne Änderung der Parameter gute Ergebnisse für einen Großteil der untersuchten Halbleiterkristalle liefert.

Anhang

Zur Veranschaulichung soll hier bereits das Dialogfenster, wie es als Produkt der Anforderungen entworfen wurde, gezeigt werden. Gemäß den Richtlinien zur Erstellung eines Pflichtenheftes [1] sind grafische Entwürfe von Benutzerschnittstellen normalerweise nicht vorgesehen.

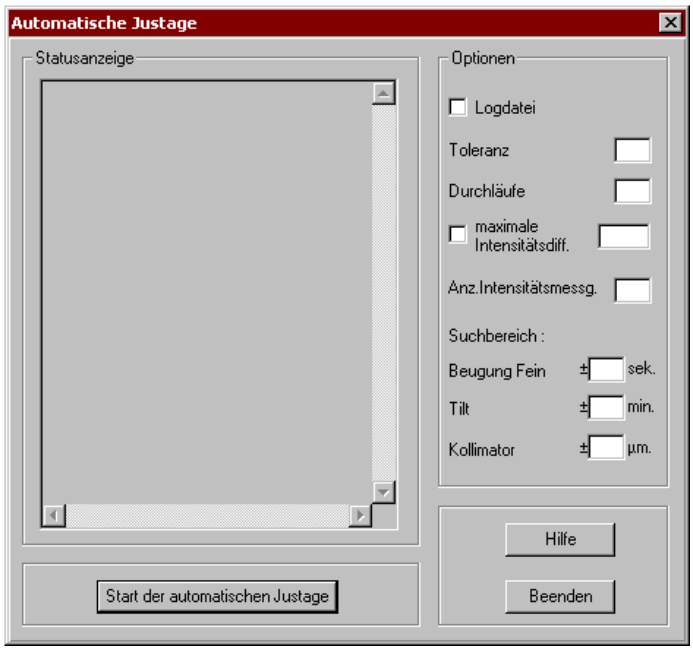


Abbildung 4.2: Enwurf des Dialogfensters der Programmfunktion „Automatische Justage" gemäß Pflichtenheft

Kapitel 5

Entwicklung eines Justagealgorithmus

5.1 Vorbetrachtung

Für die automatische Justage werden die Antriebe „Beugung fein", „Tilt" und „CC" verwendet. Zusätzlich werden die Intensitätswerte des Detektors benötigt. Veränderungen der Antriebe wirken sich auf die Intensitätswerte des Detektors aus. Ziel ist es, die am Detektor gemessenen Intensitätswerte zu maximieren.

Speziell auf das Problem bezogen, stellt sich die Aufgabe mittels dieser nichtlinearen Zielfunktion wie folgt dar:

$$f : DF \times TL \times CC \rightarrow Intensität \tag{5.1}$$

$$mit \;\; DF, TL, CC, Intensität \in \mathbb{R}.$$

Es handelt sich bei der Justage der Meßprobe um eine nichtlineare Optimierungsaufgabe mit drei Veränderlichen, bei der die Zielfunktion maximiert werden muß.

$$f(\vec{x}) = max! \qquad mit \;\; \vec{x} \in \mathbb{R}^3. \tag{5.2}$$

5.1.1 Mathematische Verfahren

In der Mathematik gibt es viele verschiedene iterative Verfahren zum Lösen nichtlinearer Optimierungsaufgaben. Dabei gehen die meisten nach folgendem Schema vor (angelehnt an [18, S.241 ff]):

Man geht von einem zulässigen Punkt $\vec{x}_1 \in M$ mit $M \subseteq \mathbb{R}^n$ aus, wobei M der Suchbereich ist.

(1) Zuerst wird eine Suchrichtung $\vec{d_1} \in \mathbb{R}^n$ ermittelt.

(2) Als nächstes wird das Problem $f(\vec{x}_1 + \lambda\vec{d_1}) \rightarrow max!$ mit $\vec{x}_1 + \lambda\vec{d_1} \in M$ gelöst.

Man optimiert also die Funktion $\phi(\lambda) = f(\vec{x}_1 + \lambda\vec{d_1})$ der Variablen $\lambda \in \mathbb{R}$, so daß die Forderung $\vec{x}_1 + \lambda\vec{d_1} \in M$ weiterhin erfüllt wird. Diese Bedingung drückt nichts anderes aus, als daß ein λ in einem Intervall I gesucht wird, also $\lambda \in I$ mit $I \subseteq \mathbb{R}$. Die Lösung dieses Problems nennt man auch eindimensionale Suche. Man erhält ein λ_1, welches einen neuen Punkt $\vec{x}_2 = \vec{x}_1 + \lambda_1\vec{d_1} \in M$ bestimmt, wobei gilt: $f(\vec{x}_1) < f(\vec{x}_2)$. Nun werden die Schritte (1) und (2) für $\vec{x}_2$ wiederholt. So erhält man sukzessive eine Folge von Punkten $\vec{x}_1, \ldots, \vec{x}_k$, die die Zielfunktion schrittweise maximieren. Diese werden solange erzeugt, bis ein entsprechendes Abbruchkriterium greift.

Diese Art von Optimierungsverfahren nennt man Anstiegsverfahren. Zur Veranschaulichung dieser Verfahrensart soll folgende verbale Beschreibung dienen: Eine Funktion zweier Variabler kann als „Gebirge" über der Koordinatenebene dieser beiden Variablen aufgefaßt werden. Um zu einem lokal höchsten Punkt des Gebirges zu kommen, wählt man im Ausgangspunkt eine „brauchbare" Richtung. Damit ist gemeint, daß man sich eine Richtung sucht, in der es zunächst nur „bergauf" geht. Dieser Richtung folgt man solange, wie es noch „bergauf" geht und gelangt so zum nächsten Ausgangspunkt.

Beim bekanntesten Anstiegsverfahren, dem Gradientenverfahren, wird auf der Grundlage einer stetig differenzierbaren Funktion mittels partieller Ableitungen der Gradient $\nabla f(\vec{x})$ bestimmt. Dieser gibt die Richtung an, in der die Funktion am steilsten ansteigt. Im zuvor beschriebenen allgemeinen Verfahren wäre dies der Punkt (1) - die Bestimmung der Suchrichtung. Auf dieser Richtung wird anschließend das eindimensionale Teilproblem, maximiere $\phi(\lambda) = f(\vec{x}_1 + \lambda\nabla f(\vec{x}))$, gelöst.

Ein weiteres Verfahren, welches auch für die nichtlineare Optimierung benutzt wird, ist das Verfahren der konjugierten Richtungen. Dort wird die Zielfunktion in der Nähe ihres Maximums durch die Bildung einer quadratischen Funktion approximiert. Das geschieht durch die Bildung einer

Hesse-Matrix, wiederum auf der Grundlage partieller Ableitungen. Die konjugierten Richtungen, die aufgrund dieser Matrix bestimmt werden können, bilden die Suchrichtungen. Beide Verfahren, das Gradientenverfahren und das Verfahren der konjugierten Richtungen, setzen als Zielfunktion eine stetige differenzierbare Funktion voraus.

Im Fall der automatischen Justage sind für die Intensitätsverteilung im Raum keine partiellen Ableitungen zur Bildung des Gradienten bestimmbar. Der Grund dafür liegt im Fehlen einer mathematischen Zielfunktion, die die Intensitätsverteilung beschreiben würde. Allerdings wird davon ausgegangen, daß die Intensitätsverteilung auf einer stetigen Funktion beruht, die aber nicht bestimmbar ist. Deshalb muß ein anderes Verfahren für die Bestimmung der Suchrichtung verwendet werden, das ohne die Bildung von Ableitungen auskommt.

5.1.2 Eigener Ansatz

Das hier vorgestellte Verfahren basiert auf dem Anstieg in Koordinatenrichtung, auch Relaxation genannt. Allerdings ist dieses Verfahren, wie es in der Literatur[17] vorgestellt wird, für das Problem der automatischen Justage allein nicht ausreichend. Deshalb mußte das Verfahren leicht abgeändert werden.

Definition 5.1.1. Gegeben ist $f(x)$ als zu maximierende Funktion. $\vec{d}$ heißt *brauchbare Richtung* im Punkt x_1, wenn mit einem $\lambda^* > 0$ gilt:

$$f(x_1 + \lambda\vec{d}) > f(x_1) \text{ für alle } 0 < \lambda \leq \lambda^*.$$

Das Extremum $\overline{\lambda}$ aller dieser λ^* heißt *Brauchbarkeitsgrenze*.

Beim Verfahren der Relaxation werden nacheinander in zyklischer Reihenfolge die Richtungen jeder Koordinatenachse auf Brauchbarkeit getestet. Wenn sie brauchbar sind, werden sie als Anstiegsrichtung verwendet.

Da im Fall der Topographie die Intensitätsverteilung ellipsoidförmig schief im 3-dimensionalen Raum liegt, würde das Verfahren der Relaxation nur sehr langsam zu einem Ergebnis kommen. Der Optimierungsalgorithmus würde sich nicht schnell genug in Richtung Maximalstelle bewegen. Sinnbildlich kann man sich das Annähern an die Maximalstelle in Form einer Treppe vorstellen. Um also das Verfahren zu beschleunigen, wurden zusätzlich Koordinatensystemtransformationen eingebaut.

Für den Algorithmus bedeutet das, daß nun die Richtungen nicht nur „gerade" die einzelnen Ebenen hinaufzeigen, sondern daß jetzt auch die Anstiege schräg über die Ebenen verlaufen. Wie das zustande kommt, soll der folgende Ansatz für ein entsprechendes Verfahren zeigen.

Als erstes wird entlang einer Achse ein Maximum gesucht. Dies kann zum Beispiel mit dem Verfahren des Goldenen Schnitts durchgeführt werden. Danach wird an der entsprechenden Stelle ein Maximum auf der nächsten Achse gesucht. Man erhält einen Punkt. Der Vektor vom Koordinatenursprung zum jetzt erhaltenen Punkt bildet den Transformationsvektor. Dieser ist die Grundlage für die durchzuführende Koordinatensystemtransformation. Das heißt, das Koordinatensystem wird zuerst gedreht und zwar so, daß die erste untersuchte Achse auf dem zuvor erhaltenen Vektor liegt. Dann wird das Koordinatensystem in den Punkt verschoben, zu dem der Transformationsvektor zeigt. Als nächstes wird das andere Paar Achsen betrachtet, mit denen genauso verfahren wird. Im folgenden Abschnitt wird das Verfahren für die „Automatische Justage" konkret erläutert.

5.2 Algorithmus „Automatische Justage"

Um einen Transformationsvektor zu erhalten, werden immer zwei Achsen betrachtet. Das ist darin begründet, daß auch in der manuellen Justierung der Probe entweder „Tilt"(TL) oder der Kollimator(CC) verändert werden und danach immer mit „Beugung fein"(DF) nachgeregelt wird. Dieses Prinzip wurde als Ausgangspunkt für den Algorithmus der „Automatischen Justage" benutzt. Des weiteren wird vorausgesetzt, daß sich die Probe schon im Reflex aufhält.

Nun zur formalen Beschreibung des Algorithmus:

1. Achse $CC^{(n)}$ des Koordinatensystems auf maximale Intensität stellen.

2. Achse $DF^{(n)}$ des Koordinatensystems danach ebenfalls auf maximale Intensität stellen.

3. Der Vektor $\vec{v}_n$ vom Koordinatenursprung $O^{(n)}$ zum Punkt $P_n[max(CC^{(n)}), max(DF^{(n)})]$ bestimmt die neue Richtung.
   Transformation des Koordinatensystems, so daß der Vektor $\vec{v}_n$ die neue Achse mit der Bezeichnung $CC^{(n+1)}$ im neuen Koordinatensystem $K^{(n+1)}$ bildet.
   Der Punkt $P_n[max(CC^{(n)}), max(DF^{(n)})]$ bildet nun den neuen Koordinatenursprung $O^{(n+1)}$

4. $CC^{(n+1)}$ bleibt fest $\rightarrow$ Optimierung für $DF^{(n+1)}$ und $TL^{(n+1)}$
   $TL^{(n+1)}$ auf Maximalintensität stellen

5. $DF^{(n+1)}$ auf Maximalintensität stellen

6. Der Vektor $\vec{v}_{n+1}$ vom Koordinatensystemursprung $O^{(n+1)}$ zum Punkt $P_{n+1}[max(TL^{(n+1)}), max(DF^{(n+1)})]$ ist neuer Richtungsvektor.
   Transformation des Koordinatensystems $K^{(n+1)}$ zu $K^{(n+2)}$, wobei der Vektor $\vec{v}_{n+1}$ die neue Achse $TL^{(n+2)}$ in $K^{(n+2)}$ wird.
   Der neue Koordinatenursprung $O^{(n+2)}$ liegt jetzt im Punkt $P_{n+1}[max(TL^{(n+1)}), max(DF^{(n+1)})]$.

Jetzt iteriert der ganze Algorithmus und würde im ersten Punkt fortfahren. Der Algorithmus wird so lange durchlaufen, bis eine Abbruchbedingung greift.

Wichtig ist zu sagen, daß nur die Achsen $CC^{(1)}$, $TL^{(1)}$ und $DF^{(1)}$ den Antrieben im System entsprechen. Danach werden die Koordinatensystemachsen so verdreht, das alle Antriebe beim Anfahren eines Punktes auf einer Koordinatensystemachse benutzt werden. Die Bezeichnungen $CC^{(n)}$, $TL^{(n)}$ und $DF^{(n)}$ haben in diesem Fall also nur den symbolischen Zweck der Beschriftung der Koordinatensystemachsen.

Die Ermittlung des Maximums auf einer Koordinatensystemachse, wie im ersten und zweiten sowie im vierten und fünften Punkt des Algorithmus, entspricht einer 1-dimensionalen Suche. Dazu ist es notwendig, daß man mit Meßpunkten auf der Achse arbeitet. Theoretisch könnte man die Achse auch abfahren und dabei die Intensität messen und auf das Maximum stellen. Aber es gibt zwei Gründe die dagegen sprechen:

1. Da durch die Transformationen des Koordinatensystems die Achsen nicht mehr nur einem Antrieb entsprechen, müsste man mehrere Antriebe zugleich betreiben. Dies würde jedoch einen exakten Synchronbetrieb der Motorenhardware erfordern. Das wird aber nicht vom gegebenen System ermöglicht.

2. Für eine genaue Messung an einem Punkt muß für einen kurzen Augenblick gestoppt werden (cirka 1 Sekunde). Der Grund ist der, daß der Detektor nur in einem bestimmten Zeitfenster Meßwerte liefert.

Für die 1-dimensionale Suche gibt es zwei Strategien. Eine wäre die Verwendung von simultanen Verfahren, in denen die Meßpunkte zur Ermittlung des Maximums von vornherein festgelegt werden. Die andere Möglichkeit ist die Verwendung von sequentiellen Suchverfahren, in deren Verlauf die Meßpunkte erst bestimmt werden.

Fest steht, daß bei der „Automatischen Justage” die Zeitkomplexität des Verfahrens eindeutig von der Anzahl der Meßpunkte abhängt. Das heißt, je weniger Meßpunkte, desto schneller der Ablauf der „Automatischen Justage”. Es bietet sich in diesem Fall die Verwendung eines sequentiellen Verfahrens

an. Dort unterscheidet man zwei Arten von Verfahren. Zum einen die 1-dimensionale Suche ohne Verwendung von Ableitungen und zum anderen die 1-dimensionale Suche unter Verwendung von Ableitungen. Verfahren wie die dichotome Suche, der Goldene Schnitt oder die Fibonacci-Suche gehören zu der Sorte, die ohne Ableitungen auskommen. Das Newton-Verfahren ist ein typisches Verfahren für die Klasse, die mit Ableitungen arbeitet.

Im Fall der „Automatischen Justage" scheint das Verfahren des Goldenen Schnitts am besten geeignet zu sein. Der folgende Abschnitt soll dieses Verfahren kurz erläutern und die obige Annahme bestätigen.

5.3 Goldener Schnitt

Der Goldene Schnitt wird auch als „stetige Teilung" bezeichnet, weil die Verkürzung des Suchintervalls stets in den gleichen Proportionen erfolgt. Das Ausgangsintervall wird in jedem Schritt um ca. ein Drittel kleiner. Da stetige Teilung in der Natur und Kunst häufig vorkommt, wurde der Begriff „Goldener Schnitt" geprägt.

Der Vorteil des Goldenen Schnitts gegenüber der dichotomen Suche ist der, daß hier nur eine neue Messung pro Iteration nötig ist. Dies ist wichtig, da bei der „Automatischen Justage" die Anzahl der Meßpunkte darüber entscheidet, wieviel Zeit der Suchalgorithmus braucht.

Die Erläuterung des Goldenen Schnitt-Verfahrens hält sich an das Schema von [18, S.245].

Eingabe: Toleranzlänge $\ell > 0$; Intervall $[a, b]$; unimodale Fkt. φ

Ausgabe: Intervall $[a_k, b_k]$ mit Länge kleiner als ℓ, das die Lösung enthält.

Benutzt: Konstante $\alpha = \frac{\sqrt{5}-1}{2} \approx 0,618$

Initialisierung: setze $k = 1$, $[a_1, b_1] := [a, b]$, $\lambda_1 = \alpha a_1 + (1-\alpha) b_1$ und $\mu_1 = (1-\alpha) a_1 + \alpha b_1$

Iteration: 1.) wenn $b_k - a_k < \ell$, STOP $\rightarrow$ Lsg. in $[a_k, b_k]$

2.) falls $\varphi(\lambda_k) > \varphi(\mu_k)$, setze $[a_{k+1}, b_{k+1}] := [a_k, \mu_k]$
$\lambda_{k+1} := \alpha a_k + (1-\alpha) b_k$
$\mu_{k+1} := \mu_k$ und berechne $\varphi(\lambda_{k+1})$

$$\begin{array}{ll} \text{falls } \varphi(\lambda_k) \leq \varphi(\mu_k), & \text{setze } [a_{k+1}, b_{k+1}] := [\lambda_k, b_k] \\ & \lambda k + 1 := \mu_k \\ & \mu_{k+1} := (1-\alpha)a_k + \alpha b_k \\ & \text{und berechne } \varphi(\mu_{k+1}) \end{array}$$

ersetze k durch $k+1$ und gehe zu 1.).

Ein noch besseres Ergebnis als der Goldene Schnitt liefert das Fibonacci-Verfahren. Dort wird eine Verkleinerung des Suchintervalls um fast die Hälfte erreicht. Dadurch braucht man pro Suche einen Schritt weniger. Die zu geringe Zeitersparnis würde den Aufwand für die Umsetzung des Fibonacci-Verfahrens jedoch nicht rechtfertigen. Deshalb ist das Verfahren des Goldenen Schnitts im Fall der automatischen Justage vorzuziehen.

5.4 Koordinatensystemtransformation

Dieser Abschnitt erläutert die Koordinatensystemtransformationen, die im Algorithmus der automatischen Justage durchgeführt werden. Notwendig wird dies, weil von einem Weltkoordinatensystem ausgegangen wird, welches die realen Positionen der Antriebe repräsentiert. Die virtuellen Koordinatensysteme des Algorithmus werden für die Suche verwendet. Es wird eine Punktmenge in zwei unterschiedlichen Koordinatensystemen betrachtet. Um die Koordinaten eines Punktes von einem Koordinatensystem in ein anderes umzurechnen, sind geometrische Transformationen notwendig. Die folgenden Ausführungen sind angelehnt an die Erläuterungen aus [6, 5].

5.4.1 Geometrische Transformationen

Zu den geometrischen Transformationen gehören die Translation, die Skalierung und die Rotation. Für eine Transformation zum Wechsel des Koordinatensystems werden nur die Translation und die Rotation benötigt.

Die Translation ist eine Verschiebung um einen gewünschten Betrag. Will man einen Punkt $P(x, y, z)$ um d_x, d_y und d_z Einheiten parallel zu den drei Achsen verschieben, so erhält man einen neuen Punkt $P'(x', y', z')$. Dies kann in folgender Schreibweise festgehalten werden:

$$x' = x + d_x, \qquad y' = y + d_y, \qquad z' = z + d_z.$$

In Form von Spaltenvektoren definiert,

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}, \qquad T = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix},$$

kann man eine kompaktere Schreibweise wählen:

$$P' = P + T.$$

Eine Translation in die Rückrichtung erhält man ganz einfach durch die Multiplikation des Vektors T mit -1.

Für die Rotation eines Punktes $P(x, y, z)$ im Raum muß man zunächst die Rotationen um die einzelnen Achsen berechnen. Angenommen, man will einen Punkt um die z-Achse mit dem Winkel θ rotieren lassen, so wird die Drehung mathematisch definiert durch die Gleichungen

$$x' = x \cdot \cos\theta - y \cdot \sin\theta, \qquad y' = x \cdot \sin\theta + y \cdot \cos\theta, \qquad z' = 1.$$

In Matrizenschreibweise erhält man

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

oder $P' = R_z \cdot P$, wobei R_z die Drehungsmatrix um die z-Achse ist. Die Drehungsmatrix für eine Rotation um die x-Achse lautet:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}.$$

Die Drehungsmatrix für eine Rotation um die y-Achse hat die folgende Form:

$$R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}.$$

Eine beliebige Rotation um den Koordinatenursprung wird aus den Rotationen um die einzelnen Hauptachsen zusammengesetzt. Durch Multiplikation der Drehungsmatrizen erhält man eine Rotationmatrix für eine Drehung um den Koordinatenursprung:

$$R = R_z \cdot R_y \cdot R_x.$$

Für eine Umkehrung der Drehung verwendet man die Inversen der Rotationsmatrizen R_z, R_y und R_x. Diese erhält man durch das Einsetzen ihrer Winkel mit negativen Vorzeichen. Bei der Bildung der umgekehrten Rotationsmatrix R für eine Drehung um den Ursprung muß beachtet werden, daß die Multiplikation der Rotationsmatrizen um die einzelnen Hauptachsen mit negativen Winkeln in umgekehrter Weise zu erfolgen hat. Der Grund liegt in

der Nichtkommutativität der Matrixmultiplikation. So hat die Inverse Rotationsmatrix die folgende Form:

$$R^{-1} = (R_z R_y R_x)^{-1} = R_x^{-1} R_y^{-1} R_z^{-1}.$$

Eine weitere Transformation ist die Skalierung. Auf die Erläuterung wird an dieser Stelle verzichtet, da die Skalierung für den Algorithmus der automatischen Justage nicht von Bedeutung ist.

Die Matrizendarstellungen für die Translation und die Rotation lauten wie folgt:

$$P' = T + P,$$

$$P' = R \cdot P.$$

Man sieht, daß die Darstellung der Translation anders als die der Rotation ist. Vorteilhaft wäre es, wenn man beide Transformationen einheitlich behandeln und damit leichter kombinieren könnte.

5.4.2 Homogene Koordinaten

Mit Hilfe der homogenen Koordinaten besteht die Möglichkeit zur einheitlichen Beschreibung von geometrischen Transformationen durch Matrixmultiplikation. In einer 4x4-Matrix können alle geometrischen Transformationen zusammengefaßt werden:

$$\left(\begin{array}{c|c} \begin{array}{c} Rotation \\ Skalierung \end{array} & Translation \\ \hline \begin{array}{c} Perspektivische \\ Transformation \end{array} & Gesamtskalierung \end{array}\right).$$

Der Vorteil liegt darin, daß anstelle der Hintereinanderausführung mehrerer Transformationen auf Punkte des 3-dimensionalen Raumes einmalig eine Gesamttransformationsmatrix berechnet werden kann, die anschließend mit den homogenen Koordinaten der Punkte multipliziert wird.

Für eine Schreibweise der homogenen Koordinaten fügt man jedem Punkt des 3-dimensionalen Raumes eine vierte Koordinate hinzu. Bisher wurde ein Punkt in kartesischer Schreibweise notiert:

$$P = (x, y, z) \quad mit \quad x, y, z \in \mathbb{R}.$$

Der entsprechende Punkt in homogenen Koordinaten hat folgende Form:

$$P_H = (h \cdot x, h \cdot y, h \cdot z, h) \quad h \in \mathbb{R}, h \neq 0 \quad beliebig.$$

Die einfachste Möglichkeit einen Punkt zu homogenisieren ist die, daß man $h = 1$ setzt. Um einen Punkt aus dem 3-dimensionalen Raum in homogener Schreibweise zu erhalten, muß man ihm in der vierten Koordinate eine 1 anfügen.

Die Translation kann nun mit Hilfe der homogenen Koordinatenschreibweise auch als multiplikative Matrixoperation beschrieben werden. In Matrixdarstellung hat die Translation eines Punktes $P(x, y, z)$ um den Vektor (T_x, T_y, T_z) folgendes Aussehen:

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}.$$

Analog wie im vorhergehenden Abschnitt gestaltet sich die Rotation.
Rotation eines Punktes um die x-Achse mit dem Winkel α_x:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_x) & -\sin(\alpha_x) & 0 \\ 0 & \sin(\alpha_x) & \cos(\alpha_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \cdot \cos(\alpha_x) - z \cdot \sin(\alpha_x) \\ y \cdot \sin(\alpha_x) + z \cdot \cos(\alpha_x) \\ 1 \end{pmatrix}.$$

Rotation eines Punktes um die y-Achse mit dem Winkel α_y:

$$\begin{pmatrix} \cos(\alpha_y) & 0 & \sin(\alpha_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha_y) & 0 & \cos(\alpha_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\alpha_y) + z \cdot \sin(\alpha_y) \\ y \\ -x \cdot \sin(\alpha_y) + z \cdot \cos(\alpha_y) \\ 1 \end{pmatrix}.$$

Rotation eines Punktes um die z-Achse mit dem Winkel α_z:

$$\begin{pmatrix} \cos(\alpha_z) & -\sin(\alpha_z) & 0 & 0 \\ \sin(\alpha_z) & \cos(\alpha_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\alpha_z) - y \cdot \sin(\alpha_z) \\ x \cdot \sin(\alpha_y) + y \cdot \cos(\alpha_y) \\ z \\ 1 \end{pmatrix}.$$

Zur Bestimmung der Lage eines Objektes in einem anderen Koordinatensystem erfolgt ein Wechsel des Koordinatensystems. Beim Suchalgorithmus gestaltet sich das Problem so, daß ein Punkt im Koordinatensystem K gegeben ist und die Koordinaten desselben Punktes im Koordinatensystem K' gesucht werden. Das Ganze soll auch in umgekehrter Reihenfolge möglich sein.

Die Translation eines Koordinatensystems um den Vektor (T_x, T_y, T_z) entspricht einer Verschiebung des geometrischen Objektes um den Vektor $(-T_x, -T_y, -T_z)$.

Die Rotation eines Koordinatensystems erfolgt analog wie die Rotation eines Punktes. Allerdings ist zu beachten, daß die Rotation eines Koordinatensystems um den Winkel α einer Drehung eines geometrischen Objektes um den Winkel $-\alpha$ entspricht.

Zum Abschluß soll das folgende Beispiel eine Koordinatentransformation unter Verwendung homogener Koordinaten mittels geometrischer Transformationen erläutern.

Angenommen, der Vektor $v_1^T = (0, 5, 0)$ soll die x-Achse des neuen Koordinatensystems bilden. Der Punkt, auf den der Vektor zeigt, soll der Ursprung vom neuen Koordinatensystem sein. Das neue Koordinatensystem erzeugt man durch eine Translation in Richtung v_1 und einer Rotation um die z-Achse des alten Koordinatensystems.

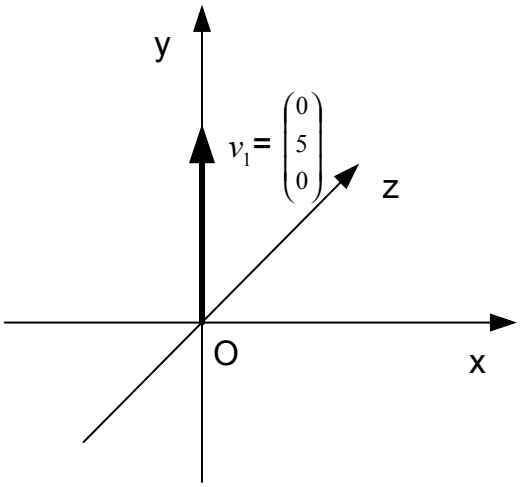


Abbildung 5.1: Beispiel zur Koordinatensystemtransformation

Zunächst muß der entsprechende Winkel zwischen dem Vektor v_1 und der alten x-Achse berechnet werden. Die alte x-Achse wird repräsentiert durch den Einheitsvektor $e_1^T = (1, 0, 0)$. Der Winkel α zwischen beiden Vektoren kann mit Hilfe des Skalarproduktes berechnet werden.

$$\cos\alpha = \left(\frac{v_1 \cdot e_1}{\|v_1\| \cdot \|e_1\|}\right)$$

$$\cos\alpha = \frac{0 \cdot 1 + 5 \cdot 0 + 0 \cdot 0}{\sqrt{0^2 + 5^2 + 0^2} \cdot \sqrt{1^2 + 0^2 + 0^2}}$$

$$\cos\alpha = 0$$

$$\alpha = 90°.$$

Um die entsprechende Transformationsmatrix zum neuen Koordinatensystem zu erhalten, muß man den negativen Wert des Winkels und den umgekehrten Translationsvektor verwenden. Mit diesen bildet man die jeweilige Rotationsmatrix bzw. Translationsmatrix und multipliziert sie anschließend. Die Berechnung wird wie folgt durchgeführt:

$$R_{hin} = R_z \cdot T$$

$$R_{hin} = \begin{pmatrix} \cos(-90°) & -\sin(-90°) & 0 & 0 \\ \sin(-90°) & \cos(-90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_{hin} = \begin{pmatrix} 0 & 1 & 0 & -5 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Um die Inverse Transformationsmatrix zu berechnen, die den Übergang vom neuen zum alten Koordinatensystem darstellt, muß mit einem positiven Winkel und dem originalen Translationsvektor gearbeitet werden. Außerdem ist zu beachten, daß die Matrixmultiplikation nicht kommutativ ist, dementsprechend werden Translations- und Rotationsmatrix vertauscht.

$$R_{rück} = T \cdot R_z$$

$$R_{rück} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & \cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_{rück} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Als nächstes ist der Punkt $P = (0, 4, 0)$ im neuen Koordinatensystem gegeben. Nun möchte man die Weltkoordinaten dieses Punktes berechnen. Das geschieht folgendermaßen:

$$P_{welt} = R_{rück} \cdot P$$

$$P_{welt} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 4 \\ 0 \\ 1 \end{pmatrix}$$

$$P_{welt} = \begin{pmatrix} -4 \\ 5 \\ 0 \\ 1 \end{pmatrix}.$$

Dieses einfache Beispiel wurde deshalb gewählt, weil das Ergebnis schnell überprüft werden kann.

Um die Koordinaten des Ursprungs O_{welt} vom Weltkoordinatensystem im neuen Koordinatensystem zu ermitteln, multipliziert man einfach den Punkt $O_{welt}(0,0,0)$ mit der Transformationsmatrix R_{hin}:

$$O_{neu} = R_{hin} \cdot O_{welt}$$

$$O_{neu} = \begin{pmatrix} 0 & 1 & 0 & -5 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$O_{neu} = \begin{pmatrix} -5 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Auch dies läßt sich schnell nachvollziehen. Die Abb. 5.2 ist dabei hilfreich.

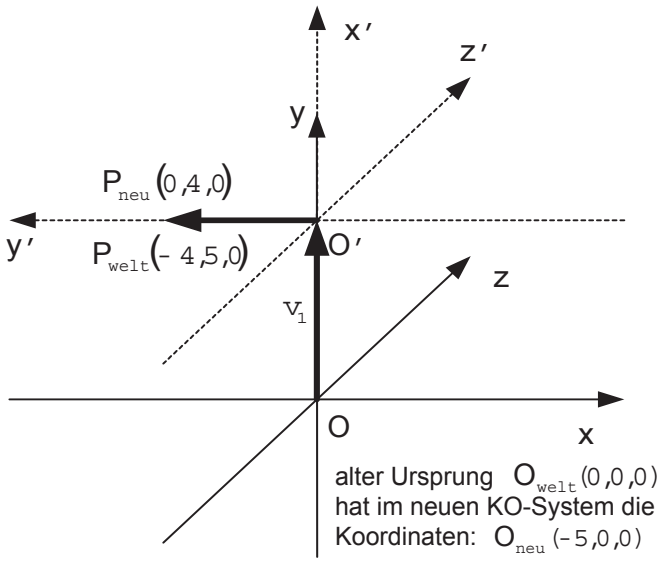


Abbildung 5.2: Visualisierung des vorangegangenen Beispiels

Kapitel 6

Design

In diesem Kapitel soll es um das Design der Programmerweiterung gehen. Dazu wird der Algorithmus für die Justage zunächst unter dem Aspekt der softwaretechnischen Integration betrachtet. Dadurch werden Teilaufgaben sichtbar, die neben der Umsetzung des mathematischen Algorithmus zusätzlich realisiert werden müssen.

6.1 Ablauf der „Automatischen Justage"

Der Entwurf des Algorithmus für die automatische Justage im Kapitel 5 ist mathematisch abstrakt gestaltet. Zudem werden durch das Pflichtenheft Forderungen gestellt, die zu Teilaufgaben führen, aber noch nicht im mathematischen Entwurf beachtet wurden. Deshalb soll der Ablauf der automatischen Justage und der damit verbundenen Teilaufgaben unter dem Aspekt der softwaretechnischen Umsetzung erläutert werden.

Zur Erläuterung und Darstellung der einzelnen Prozesse dient das Flußdiagramm in Abb. 6.1. Es soll den Arbeitsablauf der Erweiterung „Automatische Justage"dokumentieren und gleichzeitig einen einfachen Überblick bieten.

Zunächst muß beim Ausführen der „Automatischen Justage"geprüft werden, ob alle Geräte, die für eine korrekte Einstellung der Probe notwendig sind, vorhanden und ins System eingebunden sind. Folgende Geräte müssen für die Justage der Probe zur Verfügung stehen:

- Motor für die Achse „Tilt" (TL),
- Motor für die Achse „Beugung fein" (DF),
- Motor für die Krümmung des Kollimators (CC),
- 0-dimensionaler Detektor.

Die korrekte Einbindung der Geräte ist Voraussetzung für die Ausführung des Suchalgorithmus.

Als nächste Aufgabe erfolgt das Einlesen und Auswerten der Dialogparameter. Damit sind die Einstellungen gemeint, die durch den Nutzer laut Pflichtenheft vorgegeben werden können, um den Suchalgorithmus zu beeinflussen. Zu diesen Parametern gehören das Suchintervall, die maximale Intensitätsdifferenz, die Anzahl der Intensitätsmessungen, die Anzahl der Wiederholungen des Algorithmus und die Auswahl für die eventuelle Protokollierung der Justage. Auf die einzelnen Parameter wird später eingegangen.

Nachdem die Dialogparameter eingelesen wurden, werden die Motorpositionen der einzelnen Freiheitsgrade (TL, DF, CC) ermittelt. Da das Suchintervall zu diesem Zeitpunkt schon gegeben ist, kann nun überprüft werden, ob die Suche innerhalb eines für die Motoren erlaubten Bereichs erfolgt. Von der Steuersoftware sind dazu Softwareschranken vorgegeben. Durch sie soll die Ansteuerung von Positionen außerhalb der Spezifikation verhindert werden, um Hardwareschäden vorzubeugen. Sollte der Fall eintreten, daß das Suchintervall über den Fahrbereich eines Motors hinausreicht, wird das Suchintervall für die betreffende Achse an die Softwareschranke angepaßt.

Da nun der Wertebereich für die Suche im Weltkoordinatensystem[1] vorliegt, muß dieser jetzt in das aktuelle Koordinatensystem umgerechnet werden. Das heißt, es muß eine Funktion implementiert werden, die die Koordinanten vom Weltkoordinantensystem ins aktuelle Koordinatensystem umrechnet. Am Anfang des Suchalgorithmus ist, wie im Kapitel 5 beschrieben, das Weltkoordinatensystem auch das aktuelle Koordinatensystem. Erst nach der ersten Koordinatensystemtransformation wird es notwendig, diese Umrechnungen durchzuführen.

Nachdem man den Wertebereich umgerechnet hat, kann nun die Suche auf den Achsen starten. Im ersten Iterationsschritt wird zunächst die z-Achse nach einem Maximum untersucht. Im Anfangsschritt ist dies gleichzusetzen mit einer Suche auf dem Freiheitsgrad „Kollimator" (CC). Der im Flußdiagramm verzeichnete Prozeß „Max-Suche auf z-Achse" unterteilt sich in weitere wichtige Teilaufgaben, so das es notwendig wird, für diese Aufgabe ein gesondertes Diagramm mit Beschreibung zu verwenden. Im Prinzip soll auf der gegebenen Achse mit Hilfe des Goldenen Schnitts ein Maximum gesucht werden. Ergebnis dieses Prozesses ist, daß das Extremum gefunden wird und die Motoren an den entprechend umgerechneten Weltkoordinanten stehen. Äquivalent zu dieser Aufgabe sind auch die Prozesse „Max-Suche auf y-Achse" und „Max-Suche auf x-Achse".

[1]Das Weltkoordinatensystem hat als Achsen die Freiheitsgrade bzw. Motoren TL, DF und CC. Es stellt die realen Koordinaten (Motorpositionen) des Systems dar.

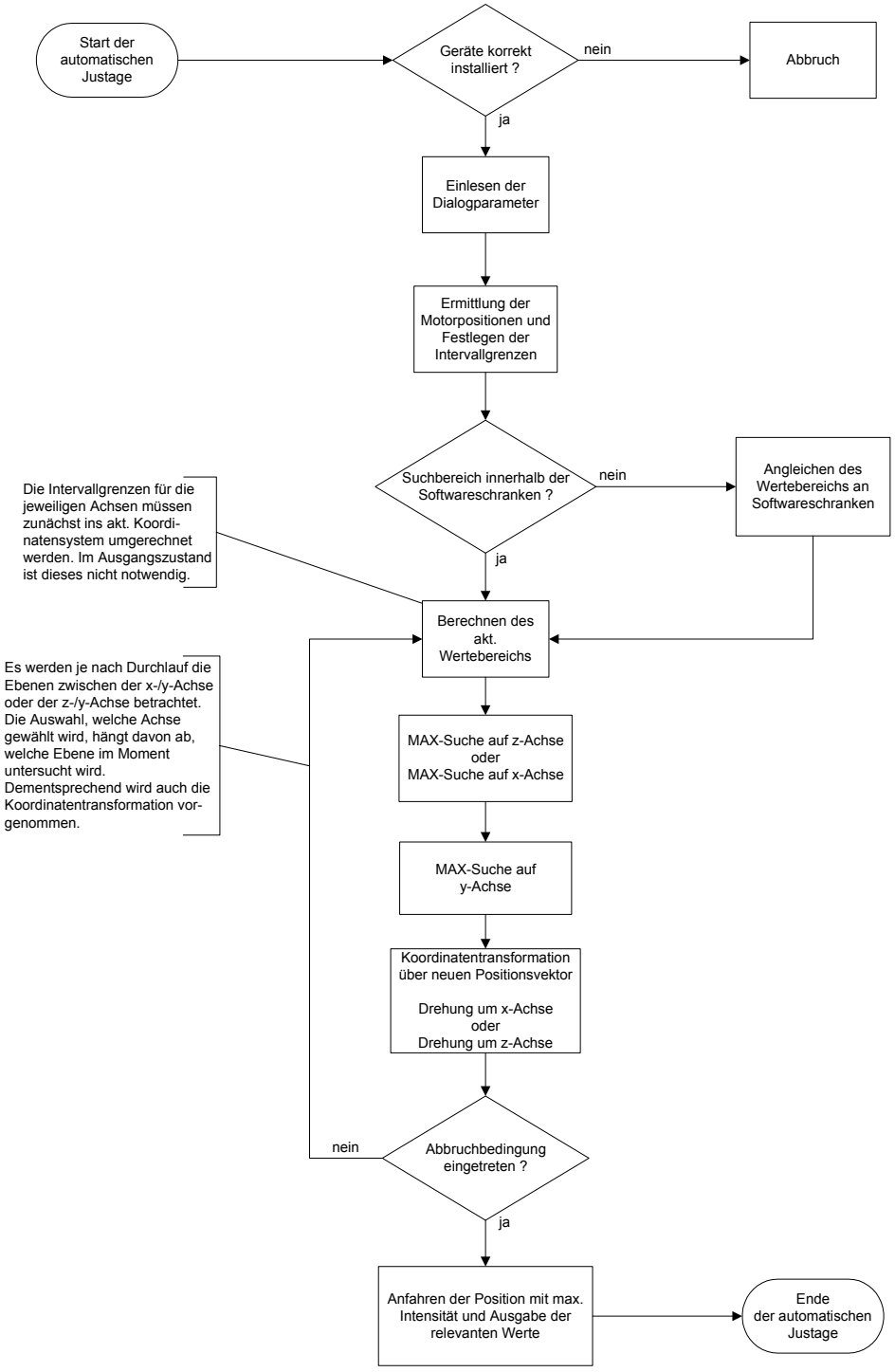


Abbildung 6.1: Flußdiagramm der automatischen Justage

Nach der Maximumsuche auf der z-Achse und auf der y-Achse hat man einen Punkt gefunden, der durch einen Vektor im aktuellen Koordinatensystem beschrieben werden kann. Dieser Vektor, im folgenden als Positionsvektor bezeichnet, bildet die Grundlage für die Transformation des Koordinatensystems. Je nachdem, welche Ebene betrachtet wurde, wird das Koordinatensystem um die x-Achse bzw. um die z-Achse gedreht und um den Positionsvektor verschoben. Der Punkt, der durch den Positionsvektor gegeben ist, bildet dann den Ursprung des neuen Koordinatensystems.

Danach werden die Abbruchbedingungen abgetestet. Dazu benutzt man die eingangs genannten Parameter, die in einem Dialog vom Nutzer eingestellt werden. Es werden mehrere Möglichkeiten für einen Abbruch des Suchalgorithmus gegeben. Da dieser Suchalgorithmus sukzessive versucht, in jedem neuen Iterationsschritt ein Intensitätsmaximum zu finden, wird zu einem bestimmten Zeitpunkt der Fall eintreten, daß die maximale Intensität erreicht ist. Allerdings ist das grundlegende Problem der nichtlinearen Optimierung, daß man nicht „den richtigen Wert" findet, sondern nur eine Näherungslösung erhält. Es kann also nicht festgestellt werden, ob das wirkliche Maximum gefunden wurde.

Eine Möglichkeit wäre zu behaupten, daß man sich nach einer bestimmten Anzahl von Durchläufen (die Schleife im Flußdiagramm ist ein Durchlauf) in der näheren Umgebung der Position aufhält, wo sich die maximale Intensität der Röntgenstrahlung befindet. Zusätzlich wird durch den Parameter „maximale Intensitätsdifferenz" vorgegeben, um wieviel kleiner die Intensität im Vergleich zum vorherigen Durchlauf sein darf. Ist diese Intensitätsschranke unterschritten, wird der Suchalgorithmus abgebrochen. Dies ist notwendig, da der Algorithmus kleine „Ausreißer", hervorgerufen durch Meßungenauigkeiten, aufweisen kann.

Wurde der Suchalgorithmus abgebrochen, wird die Position mit der maximal gefundenen Intensität angefahren. Zusätzlich werden für den Nutzer relevante Daten ausgegeben. Zu diesen Daten gehören die Intensität, die Dauer der Justage und die Position der Motoren.

6.1.1 Maximumsuche auf den Achsen

Dieser Abschnitt unterzieht die Maximumsuche auf einer Achse, wie in Abb. 6.1 aufgeführt, einer genaueren Betrachtung. Ausgangspunkt ist, daß auf einer bestimmten Achse mittels des Goldenen Schnitts ein Maximum bestimmt werden muß. An dieser Stelle soll noch einmal verdeutlicht werden, daß die Suche auf einer Achse mit imaginären Koordinaten durchgeführt wird. Allerdings sind diese Koordinaten in reale Positionen, die den Koordinaten des Weltkoordinatensystems entsprechen, umzurechnen.

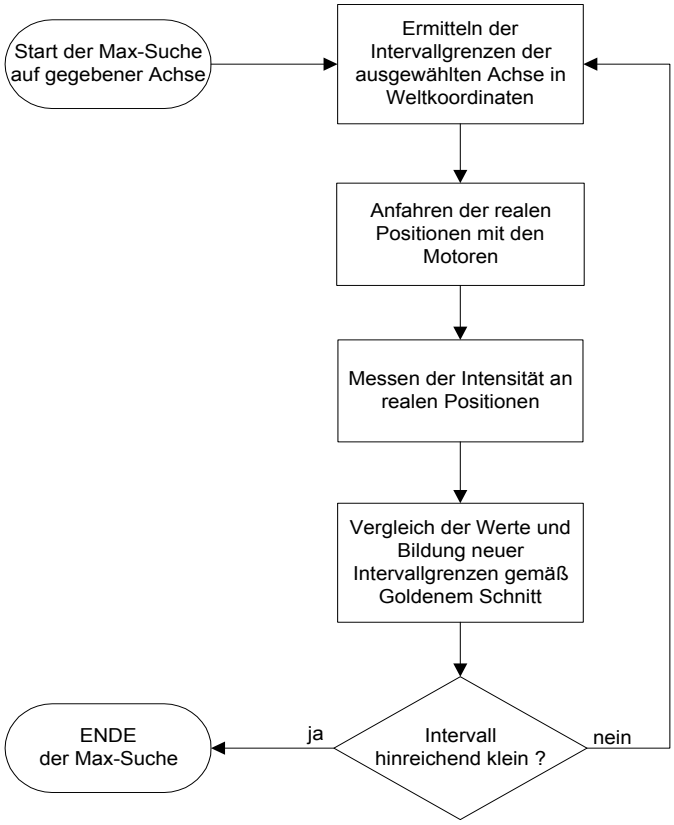


Abbildung 6.2: Flußdiagramm der Maximumsuche auf einer Achse

Eingangs muß die Achse angegeben werden, für die die eindimensionale Optimierung durchgeführt werden soll. Dann werden die Intervallgrenzen für diese Achse in reale Positionen umgerechnet und anschließend von den Motoren angefahren. Nach der ersten Koordinatensystemtransformation werden zwei Motoren für die Betrachtung auf einer Koordinatensystemachse benutzt. Das resultiert daraus, daß die Achse des aktuell betrachteten Koordinatensystems gegenüber dem Weltkoordinantensystem verdreht ist. Beim Anfahren der realen Positionen muß immer gewartet werden, bis beide Motoren stehen. Erst dann können die Intensitätswerte gemessen werden, je nachdem, wieviele Intensitätsmessungen an einer Position durchgeführt werden sollen. Bei mehreren Messungen an einer Stelle wird der Median der gemessenen Werte gebildet. Notwendig wird das, weil durch Fehler des Detektors und des Versuchsaufbaus Intensitätsschwankungen auftreten, die durch mehrfache Messungen ausgeglichen werden können.

Nachdem die Intensitäten an den Intervallgrenzen erfaßt wurden, werden sie verglichen und anschließend gemäß dem Goldenen Schnitt neu ge-

setzt. Nach dem ersten Durchlauf der Schleife wird nur noch ein Meßwert pro Durchlauf genommen. Das ist in der Besonderheit des Goldenen Schnitts begründet, daß er nur einen neuen Meßwert pro Iteration benötigt (Kap. 5.3).

Zum Schluß wird überprüft, ob das Suchintervall hinreichend klein ist. Ist das der Fall, ist die eindimensionale Suche auf der aktuellen Achse beendet. Ansonsten wird der ganze Prozeß, wie hier dargestellt, wiederholt.

Damit ist ein Gesamtüberblick über die Funktionsweise der Programmerweiterung „Automatische Justage" gegeben. Wie letztendlich die „Automatische Justage" gemäß der Objektorientierung entworfen wurde, ist in Abschnitt 6.3 nachzuvollziehen.

6.2 Einbettung in die bestehende Software

Die Funktion der automatischen Justage soll in das bestehende Programm eingebettet werden. Dazu ist es notwendig, das Software-Projekt soweit zu analysieren, bis ausgeschlossen werden kann, daß die Integration die bisherige Funktionsweise beeinträchtigt. Es müssen also Schnittstellen gesucht werden, über die eine reibungslose Kommunikation der zu implementierenden Programmfunktion mit den anderen Komponenten der Software gewährleistet ist. Die für den Algorithmus der automatischen Justage relevanten Teile des RTK-Programms müssen lokalisiert werden.

6.2.1 Architektur

Der Anwender sollte bei der automatischen Justage die Möglichkeit haben, Parameter einzustellen und zu verändern sowie Ergebnisse während des Justageprozesses am Bildschirm zu verfolgen. Dazu muß eine Interaktion mit dem Programmnutzer möglich sein, was eine Implementation eines Dialoges impliziert. Die Lage der auf dem Probenteller befindlichen Probe soll durch Schrittmotoren gesteuert werden. Die Motoren können über die Hardware zur Motoransteuerung beeinflußt werden. Programmtechnisch muß dazu auf Funktionen des Interfaces zur Motorsteuerung zugegriffen werden (siehe Anhang A). Die Intensitätswerte der Röntgenstrahlung, die ein Maß für die Optimierung der Lage der Kristallprobe liefern, werden über die Detektorsteuerkarte vom 0-dimensionalen Detektor ermittelt. Hierfür sind von der automatischen Justage Funktionen der Detektorsteuerung aufzurufen.

Damit sind die für die Integration der automatischen Justage relevanten Komponenten der RTK-Steuersoftware gefunden. Der Architekturentwurf für die zu implementierende Programmfunktion legt fest, welche Klassen und Schnittstellen genutzt und entworfen werden müssen. Dazu werden die einzelnen Funktionsbausteine anfangs als abstrakte Pakete dargestellt und in

weiteren Schritten verfeinert. Die Schnittstellen zum bestehenden Programm sind ebenso aufgeführt wie die Beziehungen der einzelnen Pakete zueinander. Im Diagramm 6.3 sind die Komponenten der „Automatische Justage" in Relation zu den relevanten Teilen des Steuerprogramms dargestellt.

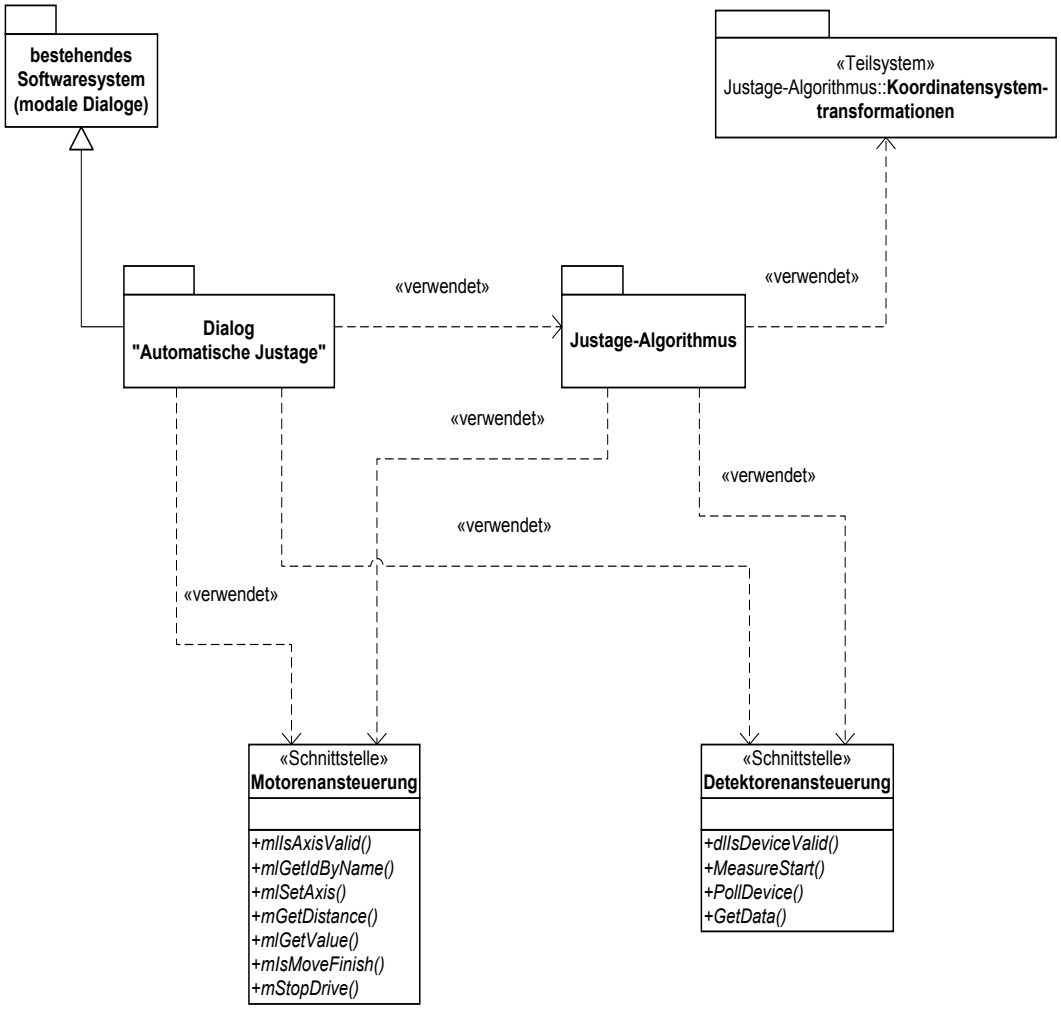


Abbildung 6.3: Architektur der Programmerweiterung

Richtet man sich nach [11, S.118ff.], gliedert sich die Architektur einer Anwendung in folgende Teilsysteme:

- Dialogsteuerung,
- Interaktionssteuerung,
- Vorgangssteuerung,
- Fundamentalklassen
- und Datenhaltung.

Für das Design der neuen Programmfunktion sind nur die ersten vier Aspekte von Bedeutung. Die Dialogsteuerung übernimmt die gesamte Kommunikation zwischen Nutzer und System, wobei die Eingaben nur syntaktisch verarbeitet werden. Der Dialog präsentiert und akkumuliert Informationen auf dem Bildschirm über Eingabefelder, Schaltflächen und weitere Dialogelemente. Eingaben des Benutzers, die nicht direkt verarbeitet werden können, wie beispielsweise das Betätigen eines Buttons, werden an die Interaktionssteuerung übergeben. Diese wiederum wertet die eingegangenen Informationen nach ihren semantischen Aspekten aus und regelt den Datenaustausch zwischen den Dialogklassen und den funktionalen Klassen, die die Steuerung der Vorgänge des Programms übernehmen. Die Vorgangssteuerung dient der Abarbeitung im fachlichen Kontext, das heißt, die für die Lösung bestimmter Aufgaben nötigen Arbeitsschritte werden von ihr initiiert. Die Fundamentalklassen stellen den eigentlichen Anwendungsbereich dar. Sie sind losgelöst von der Repräsentation der Daten in der Dialogschicht und nur für die inhaltlichen Aspekte der Anwendung zuständig.

Die Programmfunktion der automatischen Justage umfaßt neben dem Justagealgorithmus und dem dazugehörigen Subsystem der Koordinatentransformationen auch eine Komponente zur Dialogsteuerung, den Dialog „Automatische Justage". Die Dialog- und Interaktionssteuerung wird in der Dialogklasse vorgenommen, während die Justage-Vorgänge von der Algorithmusklasse gesteuert werden. Die Klassen für die Koordinatentransformation bilden die Fundamentalklassen und somit das Gerüst der Justagefunktion.

Zur näheren Betrachtung werden nun die entsprechenden Klassen, Funktionen und Programmteile lokalisiert, die eine unmittelbare Rolle bei der Programmierung der automatischen Justage spielen.

6.2.2 Benutzte Funktionen und Klassen

Das Diagramm 6.3 zeigt, daß zur Realisierung einer automatischen Justagefunktion Methoden der Klassen zur Motor- und Detektorsteuerung benötigt

werden. Für die Ansteuerung der Antriebe sind hauptsächlich Funktionen zur Auswahl des aktiven Antriebs, zur Bestimmung der Winkelposition und zum Anfahren einer bestimmten Position wichtig. Zusätzlich werden auch Funktionen gebraucht, um beispielsweise zu testen, ob ein spezieller Motor ins System eingebunden wurde oder ob sich ein Antrieb in Bewegung befindet. Um auf Detektorwerte zuzugreifen, sind Methoden zum Setzen des aktuellen Detektors, zum Starten einer Messung und zum Auslesen der Röntgenintensität vonnöten. Außerdem muß das Vorhandensein der entsprechenden Detektorsteuerkarte mittels einer Schnittstellenfunktion getestet werden können.

In der folgenden Tabelle sind die ermittelten Funktionen aufgelistet.

Aufgabe	***Schnittstellenfunktion(en)***
Motorverwaltung	`mlIsAxisValid( AchsenTyp )` `mlGetIdByName( AchsenTyp )`

Motoransteuerung	`mlSetAxis( MotorID )`
	`mIsMoveFinish()`
	`mStopDrive()`
	`mGetDistance( Winkel )`
	`mlGetValue( MotorID, MotorvariablenTyp )`
	`mMoveToDistance( Winkel )`

Detektorverwaltung	`dlIsDeviceValid( GeraeteTyp )`

Detektorsteuerung	`TDevice::MeasureStart()`
	`TDevice::PollDevice()`
	`TDevice::GetData( Intensitaet )`

Tabelle 6.1: Überblick über benötigte Schnittstellenfunktionen

Da für die Motorensteuerung ein C-Interface existiert, das die Funktionsanforderungen der Programmkomponente „Automatische Justage" erfüllt, muß nicht auf die speziellen Methoden der Motorklassen zugegriffen werden. Im Gegensatz dazu ist das Interface der Detektorsteuerung nicht ausreichend. Deshalb ist in diesem Fall eine nähere Analyse der von `TDevice` abgeleiteten Klassen nötig. Die Schwierigkeiten, die sich daraus ergeben, sind im Abschnitt 7.3 ausgeführt.

Die Steuerung der automatischen Justage erfolgt durch ein Dialogfenster,

das keine weiteren Zugriffe auf die Anwendung zulassen soll. Das bedeutet, daß die Dialogsteuerung von einer modalen[2] Dialogklasse übernommen wird. Von der bereits vorhandenen Klasse `TModalDlg` wird eine Dialogklasse für die Steuerung der automatischen Justage abgeleitet.

Die ermittelten Klassen und Methoden müssen im weiteren Verlauf auf die zugrundeliegende Quelltextstruktur zurückgeführt werden. Dieser Schritt wird im nächsten Abschnitt vollzogen.

6.2.3 Relevante Quelltexte

Bei der Analyse der bestehenden Quelltexte haben sich folgende Dateien als relevant für die Implementation der automatischen Justage herausgestellt:

Quelltextdateien	***Aufgabe***
`main.rc`	Ressourcendatei des Resource Workshops; enthält die Beschreibung der Dialoge des Hauptprogramms im ASCII-Textformat
`rc_def.h`	Headerdatei, in der u.a. die IDs der Dialogelemente und die Kommandos-IDs aller Programmdialoge verzeichnet sind
`dlg_tpl.h/cpp`	Templateklassen für modale und nichtmodale Dialoge
`m_dlg.h/cpp`	Funktionalität der Dialoge zur Motorensteuerung
`m_layer.h/cpp`	C-Schnittstelle zur Ansteuerung und Verwaltung der Motoren
`c_layer.h/cpp`	C-Interface zur Steuerung der Detektoren
`m_main.cpp`	Hauptprogramm mit Integration des Hauptmenüs und Aufruf der Dialoge entsprechend den Menüpunkten

Tabelle 6.2: Überblick über genutzte Quelltexte

Die in der Tabelle 6.2 aufgeführten Quelltexte stellen die Basis zur Einarbeitung der neuen Programmfunktion dar. Bei der Bearbeitung des Dialogfensters im zur Entwicklungsumgebung gehörigen Resource Workshop werden die Dateien `rc_def.h` und `main.rc` automatisch um die entsprechen-

[2]Modale Dialogfenster stellen die häufigste Art von Dialogen dar. Während der Anzeige eines modalen Dialogs kann der Benutzer das übergeordnete Fenster der Anwendung nicht anwählen oder verwenden. Er muß das Dialogfenster vor weiteren Aktionen erst bearbeiten und schließen.

den Einträge der Dialogelemente erweitert. Die bisherigen Implementationen von Dialogklassen der Motorsteuerung sind in den Dateien `m_dlg.h` und `m_dlg.cpp` zu finden. Die Dialogklasse der Funktion „Automatische Justage"wird aber in die gesonderten Quelltextdateien `m_justage.h` und `m_justage.cpp` ausgelagert. Bei einer späteren Portierung des Softwareprojekts ist eine Aufteilung der Klassen auf einzelne Dateien von Vorteil.

Der Entwickler des RTK-Steuerprogramms hat aufbauend auf der Windows 3.1 API eigene Klassen für modale und nichtmodale Anwendungsdialoge entworfen. Von den durch Borland C++ zur Verfügung gestellten ObjectWindows[3]-Dialogklassen hat er keinen Gebrauch gemacht, obwohl diese in etwa die gleiche Funktionalität zur Verfügung stellen.

Das zur Steuerung der automatischen Justage notwendige Dialogfenster wird von einer modalen Dialogfensterklasse verwaltet, die von der bereits vorhandenen Klasse `TModalDlg` abzuleiten ist. Das weitere Vorgehen bei der Dialogprogrammierung wird im Abschnitt 7.2 erläutert. Die Einbindung des neuen Menüpunktes „Automatische Justage" in das Anwendungsmenü und die Verbindung des Menüeintrages mit der Ausführung des Dialogfensters muß in der Quelldatei `m_main.cpp` vorgenommen werden.

6.3 Statische Struktur der „Automatischen Justage"

In diesem Abschnitt soll es um den Entwurf der statischen Struktur der Programmerweiterung „Automatische Justage" gehen. Die statische Struktur wird am besten durch Klassendiagramme verdeutlicht. Die Wahl der Darstellung fiel auf Klassendiagramme in der UML-Notation. Die Verwendung der Objektorientiertheit lag nahe, da daß bestehende Softwareprojekt zum Großteil eine objektorientierte Struktur aufweist und auch in der Programmiersprache C++, die dieses Konzept unterstützt, umgesetzt wurde.

Die Programmerweiterung besteht aus 3 Hauptteilen, die in der Abb. 6.3 dargestellt sind. Das wären demnach folgende Komponenten: eine Dialogklasse zur Anbindung an die bestehende Oberfläche, eine Klasse, die die Kernfunktionalität der „Automatischen Justage" enthält und als mathematische Hilfsbibliothek eine Klasse, die die geometrischen Transformationen für den Algorithmus realisiert.

[3]Borland C++ bietet mit ObjectWindows ein einfaches objektorientiertes Anwendungsgerüst zur Erstellung von Windowsprogrammen an. Die Klassenbibliothek stellt eine Kapselung der Windows API, ähnlich der Windows-Programmierschnittstelle Microsoft Foundation Classes (MFC) von Microsoft, dar.

6.3.1 Klassendiagramm der Dialogklasse

Wie bereits erwähnt, wird die Dialogklasse `TAutomaticAngleControl` von der Klasse `TModalDlg` abgeleitet. Dabei werden lediglich die wichtigsten Methoden neu überschrieben, der Rest wird geerbt. An dieser Stelle soll nur das Klassendiagramm stehen. Im Kapitel 7 wird genauer auf die Dialogprogrammierung im bestehenden Softwareprojekt eingegangen.

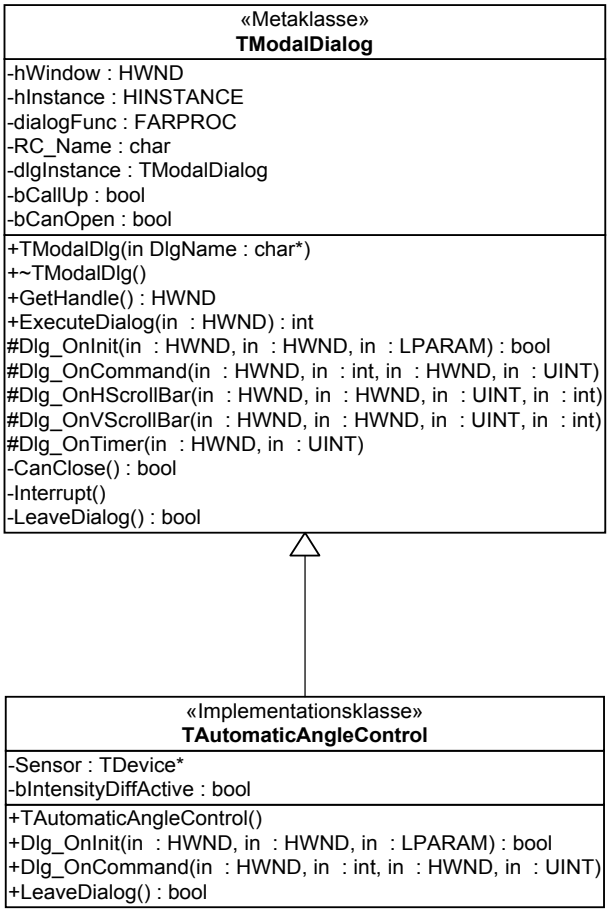


Abbildung 6.4: Klassendiagramm der neuen Dialogklasse

Das Attribut *Sensor* ist eine Instanz der Klasse `TDevice`. Es wird benötigt, um festzustellen, ob das Zählerfenster des Detektors geöffnet ist. Das Zählerfenster muß dargestellt sein, um Werte vom Detektor lesen zu können. Eine nähere Beschreibung zur Problematik des Auslesens der Werte vom Detektor erfolgt im Abschnitt 7.2.

6.3.2 Klassendiagramm der Transformationsklasse

Die Klasse `TransformationClass` ist für die Ausführung der eigentlichen Justage zuständig. Eine Instanz dieser Klasse führt den Such-Algorithmus nach dem Intensitätsmaximum aus. Aufgaben wie die Maximumsuche auf einer Achse und die Durchführung der Koordinatensystemtransformationen werden von einem Objekt dieser Klasse gesteuert.

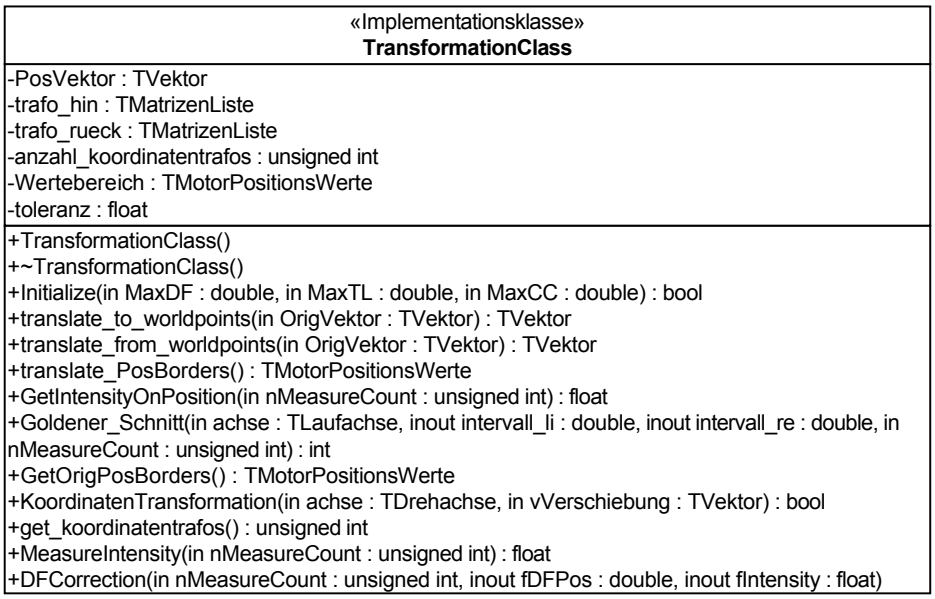


Abbildung 6.5: Klassendiagramm der Transformationsklasse

Der *PosVektor* ist dabei das wichtigste Attribut der Klasse. Dieser Vektor repräsentiert die Position während des Suchens und wird vom aktuellen Koordinatensystem zum Weltkoordinatensystem und umgekehrt transformiert.

Die Attribute *trafo_hin* und *trafo_rueck* sind Matrizenlisten, die die Transformationsmatrizen abspeichern. Diese Listen werden für die Berechnung der Transformation eines Vektors von einem Koordinatensystem zu einem anderen benutzt.

Das Attribut *anzahl_koordinatentrafos* wird als Zähler für die durchgeführten Koordinatensystemtransformationen verwendet. Über diesen Wert kann abgefragt werden, ob das Abbruchkriterium „Abbruch nach einer Anzahl von Iterationen” eingetreten ist.

Mit dem *Wertebereich* werden die ursprünglichen Motorpositionen und die Intervalle für die Suche gespeichert. Dabei handelt es sich um einen Datentypen, der die originale sowie die maximale und minimale Motorposition

enthält. Der Datentyp `TMotorPositionsWerte` speichert diese Daten für alle drei verwendeten Antriebe.

Der Wert *toleranz* gibt die Größe für das letzte Intervall beim „Goldenen Schnitt" an. Im Abschnitt 5.3, in dem der Goldene Schnitt erläutert wurde, war das die Toleranzlänge ℓ.

Die Memberfunktion `Initialize` ist, wie der Name verdeutlicht, für das Initialisieren der Transformationsklasse zuständig. Als erstes ermittelt die Funktion die aktuellen Motorpositionen. Anschließend werden die Intervalle für die Suche nach dem Peak festgelegt und die Daten im Attribut *Wertebereich* gespeichert.

Die Funktion `translate_to_worldpoints` erfüllt die Aufgabe, die Weltkoordinaten eines Punktes aus dem aktuellen Koordinatensystem zu berechnen. Um dies zu realisieren, muß zunächst der Vektor homogenisiert werden. Danach wird eine sogenannte Rücktransformation mit Hilfe der Matrizenliste *trafo_rueck* berechnet. Dies erfolgt durch die Multiplikation des homogenen Vektors mit den Transformationsmatrizen vom Ende bis zum Anfang der Matrizenliste *trafo_rueck*. Anschließend wird der homogene Vektor wieder in eine kartesische Form umgewandelt.

Analog ist die Funktion `translate_from_worldpoints` in ihrem Ablauf, nur daß hier ein Punkt, repräsentiert durch einen Vektor, vom Weltkoordinatensystem ins aktuelle Koordinatensystem transformiert wird. Zur Berechnung wird die Matrizenliste *trafo_hin* benutzt.

Die Memberfunktion `translate_PosBorders` ist für die Übersetzung der Intervalle ins aktuelle Koordinatensystem verantwortlich. Dadurch werden die Suchintervalle für die einzelnen Achsen im aktuellen Koordinatensystem festgelegt.

Die Methode `GetIntensityOnPosition` fährt die zu übersetzenden Motorpositionen des *PosVektor* an und gibt die Intensität zurück. Um die Motorpositionen zu erhalten, muß zunächst der *PosVektor* in die Koordinaten des Weltkoordinatensystems transformiert werden. Danach fahren alle Antriebe die Koordinaten an. Dabei muß darauf geachtet werden, daß alle Motoren ihre Positionen erreicht haben. Erst dann darf die Intensität an der entsprechenden Stelle gemessen werden. Der Rückgabewert soll den Wert der Intensität enthalten.

Die Bestimmung der Strahlungsintensitäten erfolgt durch Medianbildung über mehrere Messungen in der Memberfunktion `MeasureIntensity`. Die Anzahl der Messungen wird durch die Übergabe des Parameters *nMeasureCount* bestimmt. Mehrere Meßwerte verhindern, daß der Suchalgorithmus nicht negativ beeinflußt wird durch „Ausreißer" beim Messen der Intensitäten. Durch Bildung des Medians wird aus mehreren Meßwerten ein Wert bestimmt, der der Intensität an dieser Stelle am besten entspricht. Dies

erfolgt durch Bildung einer nach Größe geordneten Liste mit den einzeln gemessenen Werten. Bei einer ungeraden Anzahl von Werten wird der mittlere Wert selektiert und bei einer geraden Anzahl wird aus den beiden mittleren Werten ein Durchschnittswert gebildet und dieser als Intenstätswert ausgewählt.

Eine zentrale Stellung nimmt die Funktion `Goldener_Schnitt` in dieser Klasse ein. Die Aufgabe besteht im Ermitteln einer maximalen Intensität auf einer Achse aus dem aktuellen Koordinatensystem mit Hilfe des mathematischen Suchalgorithmus „Goldener Schnitt”. Dazu werden als Parameter die entsprechende Achse, auf der gesucht werden soll, und die Intervallgrenzen übergeben. Um die realen Positionen anzufahren, wird die Funktion `GetIntensityOnPosition` benutzt.

Die Methode `GetOrigPosBorders` dient dazu, die Wertebereiche der Motoren zurückzugeben.

Die Memberfunktion `KoordinatenTransformation` berechnet die Transformationen der Koordinatensysteme. Zunächst muß der Winkel zwischen dem gegebenen Vektor und der Drehachse berechnet werden. Dann werden über die `transformiere`-Methode der Klasse `TMatrix` die Transformationsmatrizen gebildet und in die Matrizenlisten *trafo_hin* und *trafo_rueck* eingetragen. Außerdem muß der Wert *anzahl_koordinatentrafos* um 1 erhöht werden.

Die Funktion `get_koordinatentrafos` gibt die Anzahl der Koordinatentransformationen zurück.

Wird am Ende eines Justagevorganges ein Maximum angefahren, welches während der Suche, aber nicht im letzten Suchschritt, gefunden wurde, so kann es vorkommen, daß sich an dieser Position nicht unbedingt der Intensitätswert befindet, der vorher gemessen wurde. Die Ursache dafür können unter anderem mechanische Fehler des Versuchsaufbaus sein. Es reicht allerdings eine kleine Korrektur des Antriebs „Beugung fein” aus. Diese Korrektur wird mit der Memberfunktion `DFCorrection` umgesetzt, so daß auch wirklich annähernd der Intensitätswert der Röntgenstrahlung erreicht wird, der vorher gemessen wurde.

6.3.3 Mathematische Hilfsklassen

Die Klassen `TMatrix` und `TVektor` realisieren die mathematischen Grundlagen, die notwendig sind zur Berechnung der Koordinatensystemtransformationen. Die Klasse `TMatrizenListe` ist zur Verwaltung von mehreren Matrizen gedacht. Die Funktionsweise dieser Matrizenliste ähnelt der eines Stacks.

`TMatrix` organisiert einen Datentypen, der einer Matrix im mathema-

tischen Sinne entspricht. Zusätzlich werden Operatoren zur Verfügung gestellt, die Multiplikationen, Additionen, Subtraktionen, Zuweisungen und reelle Vervielfachungen von Matrizen realisieren. Außerdem ist es möglich, mittels eines Operators eine Matrix mit einem Vektor zu multiplizieren. Dies wird für die Berechnung von geometrischen Transformationen benötigt. Die Memberfunktion `invers` berechnet die Inverse einer regulären Matrix.

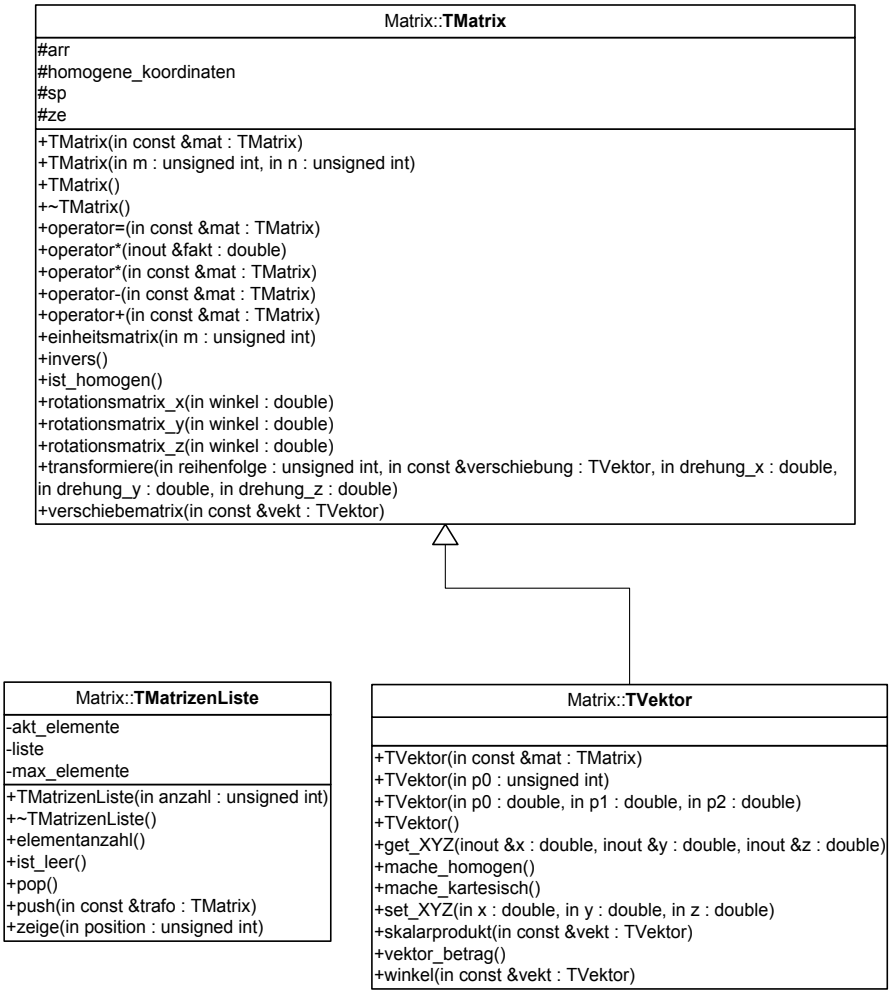


Abbildung 6.6: Klassendiagramm der Matrixklasse

Die Methode `einheitsmatrix` erstellt eine Einheitsmatrix mit Rang m. Mit `ist_homogen` wird ermittelt, ob eine Matrix bzw. ein Vektor in homogenen Koordinaten dargestellt ist. Die Funktion `verschiebematrix` berechnet eine Translationsmatrix, wobei die Verschiebung durch einen übergebenen Vektor ermittelt wird. Bei der Methode `rotationsmatrix_x` wird

eine Rotationsmatrix für die Drehung um die x-Achse mit einem bestimmten Winkel berechnet. Analog arbeiten die Funktionen `rotationsmatrix_y` und `rotationsmatrix_z`. Die Memberfunktion `transformiere` bildet eine Zusammenfassung aller Transformationen in einer Funktion. Der Parameter *Reihenfolge* bestimmt, wie die Gesamttransformationsmatrix zusammengesetzt werden soll. Dadurch ist es leicht, die Inverse einer Transformationsmatrix zu berechnen. Die weiteren Parameter geben die Verschiebung und die Winkel für die Drehung um die einzelnen Achsen an.

Die Klasse `TVektor` realisiert einen Datentypen, der einem Vektor entspricht. Da ein Vektor ein Spezialfall einer Matrix ist, wobei gilt: Spaltenanzahl=1, wurde die Klasse `TVektor` von der Klasse `TMatrix` abgeleitet. Zusätzlich zu den geerbten Methoden aus `TMatrix` wurde `TVektor` um weitere wichtige Funktionen erweitert. Dazu zählen:

- Berechnung des Skalarproduktes zweier Vektoren(`skalarprodukt`),
- Berechnung eines Winkels zwischen zwei Vektoren(`winkel`),
- Berechnung des Betrages eines Vektors(`vektor_betrag`),
- Umwandlung in einen Vektor mit homogenen Koordinaten (`mache_homogen`),
- Umwandlung in einen Vektor mit kartesischen Koordinaten (`mache_kartesisch`),
- Setzen der x,y,z-Koordinaten eines Vektors (`set_XYZ`),
- Ausgabe der x,y,z-Koordinaten eines Vektors (`get_XYZ`).

Außerdem wurde ein Operator erstellt, der einen Vektor skalieren kann.

Die Klasse `TMatrizenliste` stellt einen Datentypen zur Verfügung, mit dem es möglich ist, mehrere Matrizen ähnlich wie bei einem Stack zu verwalten. Benutzt werden Instanzen dieser Klasse zum Speichern der Transformationsmatrizen. Folgende Memberfunktionen wurden für die Arbeit mit dem Datentypen `TMatrizenliste` realisiert:

- Test, ob die Matrizenliste leer ist (`ist_leer`),
- Ermitteln der Anzahl der Listenlemente (`elementanzahl`),
- Matrix zur Liste hinzufügen (`push`),
- letzte Matrix aus Liste entfernen (`pop`),
- Element an einer bestimmten Position in der Matrizenliste ausgeben (`zeige`).

Kapitel 7

Probleme der Implementation

Die Implementation der neuen Funktion „Automatische Justage" brachte einige Schwierigkeiten mit sich, die vorwiegend in der Struktur der vorgefundenen Software begründet waren. Die anschließenden Abschnitte erläutern, was bei der Einbettung in die existierenden Quelltextdateien, bei der Entwicklung des Dialogfensters mit Borland C++ und allgemein bei der Implementation zu beachten war.

7.1 Integration

Wenn ein bestehendes Softwareprodukt im Zuge einer Anpassung an neue Anforderungen des Auftraggebers mit neuer Funktionalität versehen werden soll, ist folgendes zu beachten: Das Risiko, durch Änderungen an vorliegenden Quelltexten neue Fehler zu verursachen, ist nicht zu unterschätzen. Solche Fehler lassen sich in vielen Fällen nicht sofort als Fehler der neuen Komponente identifizieren, da sie in denselben Quelltexten, wie die ursprüngliche Programmversion, zu finden sind.

Daher sollte versucht werden, neu zu implementierende Funktionen auch in extra Quelldateien unterzubringen, um so diese Funktionalität zu kapseln. Eine spätere Wartung der Software läßt sich dadurch wesentlich vereinfachen. Im Fall der Funktion „Automatische Justage" sind getrennte Quelldateien für die Klassen zur Matrizenberechnung, die Transformationsklasse und die Dialogklasse zur automatischen Justage eingerichtet worden. Die Tabelle 7.1 zeigt die zum RTK-Projekt hinzugefügten Programmdateien und die jeweils enthaltene Funktionalität.

Die Anbindung für den Aufruf der neuen Programmkomponente muß weiterhin in den existierenden Programmcode integriert werden. Der Menüpunkt der „Automatischen Justage" wurde in die Datei `m_main.cpp` eingefügt. Die konkrete Umsetzung wird im Abschnitt 7.2 erläutert.

Quelltextdatei	*Inhalt*
`matrix.h/cpp`	mathematische Basisklassen `TMatrix`, `TVektor`, `TMatrizenListe`
`transfrm.h/cpp`	Aufzählungstypen `TDrehachse`, `TLaufachse`; Justageklasse `TranformationClass`
`m_justage.h/cpp`	Strukturen `TMotorPositionsWerte`, `TIntensityPosition`; globale Funktion `WriteToJustageLog`; Dialogklassse `TAutomaticAngleControl`

Tabelle 7.1: Überblick über die Implementationsdateien der neuen Programmfunktion „Automatische Justage"

Zur Erweiterung der Steuerungssoftware müssen die neuen Quelltextdateien in der Borland-Entwicklungsumgebung in das bestehende Projekt aufgenommen werden. Ansonsten werden die neuen Quelltexte bei der Kompilierung des Projekts nicht berücksichtigt. Dazu werden im Projektfenster die entsprechenden Datei-Knoten zum Übersetzungszweig der ausführbaren Datei hinzugefügt. Durch Drücken der rechten Maustaste auf dem Hauptknoten von `Develop.exe` wird das Projekt-Kontextmenü aufgerufen (siehe Abb 7.1). Der Menüpunkt „*Knoten hinzufügen*" öffnet einen Dateiauswahldialog, der die Auswahl der einzufügenden Datei erlaubt.

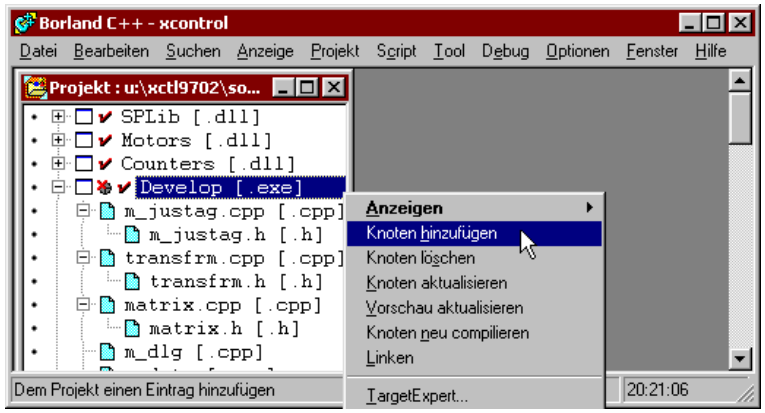


Abbildung 7.1: Einfügen neuer Quelltextdateien im Projektfenster der Borland C++ 5.02 Entwicklungsumgebung

7.2 Dialogprogrammierung

Die Bestrebung, das Softwareprojekt durch die schrittweise Ausführung im Debugmodus auf Fehler hin zu untersuchen, stellt in gewisser Weise eine Einschränkung dar. Um die Software, solange sie noch in der 16-Bit Variante vorliegt, auch in der Borland-Entwicklungsumgebung für diesen Zweck zu debuggen, ist Borland C++ 4.5 erforderlich, da nur in dieser Version ein 16-Bit Debugger Teil der Entwicklungsumgebung ist. Deshalb muß darauf geachtet werden, daß die Quelltexte unter den Versionen 4.5 *und* 5.02 kompiliert werden können. Das Beschreibungsformat der Ressourcendateien hat in Borland C++ 5.02 eine Änderung erfahren, und somit sind mit dieser Version erzeugte Dialoge nicht mehr fehlerfrei unter Borland C++ 4.5 übersetzbar. Das legt nahe, daß neue Dialoge nur mit der älteren Variante der Entwicklungsumgebung erstellt werden sollten. Die Dialogerstellung wird mit dem „Resource Workshop", der als extra Entwicklungstool in Borland C++ 4.5 enthalten ist, vorgenommen.

Die Schritte zur Erstellung des Dialogs „Automatische Justage" für das bestehende Softwareprojekt mit dem „Borland Resource Workshop" werden im weiteren erläutert.

Da der Dialog Teil des Hauptprogramms werden soll, wird dazu die Ressourcendatei `main.rc`, in der die Dialoge der Anwendung zu finden sind, über den Menüpunkt „*Datei/Projekt öffnen...*" als Projekt mit dem „Resource Workshop" geöffnet. Das Programm übersetzt die in der Datei enthaltenen Ressourcenbeschreibungen und zeigt eine Übersicht mit allen Dialogen, Bitmaps, Menüs etc. an. Ein neues Dialogfenster wird über den Menüpunkt „*Ressource/Neu...*" in das Projekt integriert. Im angezeigten Dialogfenster muß als Ressourcentyp „DIALOG" eingestellt werden. Außerdem ist festzulegen, in welchen Dateien die Beschreibungen der neuen Ressource eingefügt werden sollen. Die Dialogressourcen werden in der Datei `main.rc`, die Bezeichnerkonstanten in `rc_def.h` gespeichert (siehe Abb 7.2).

Der Entwurf des Dialogs richtet sich nach den in Abschnitt 4.4 explizit und implizit spezifizierten Dialogelementen. Die folgenden Gruppen von Steuerelementen sind im Dialogfenster anzuordnen:

- Aktionsknöpfe für Justagestart, Hilfe und Beenden,
- Eingabefelder für die Parameter zur Steuerung der automatischen Justage (Abbruchkriterien, Suchbereiche auf den Achsen, Anzahl der Intensitätsmessungen[1]),

[1]Diese Anforderung ist erst bei der Entwicklung des Justagealgorithmus entstanden (siehe Abschnitt 6.1.1).

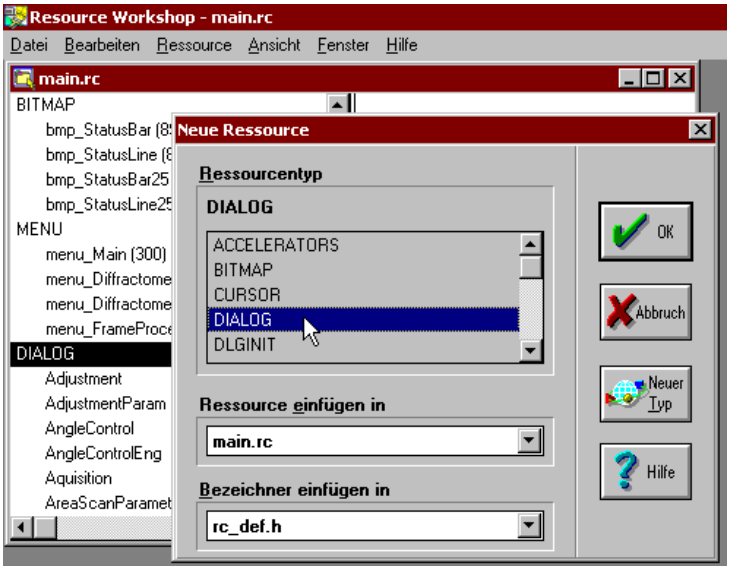


Abbildung 7.2: Erstellung einer Dialogressource mit dem „Resource Workshop" der Borland C++ 4.5 Entwicklungsumgebung

- Auswahlfeld für die Protokollierung des Justageprozesses,
- Ausgabefeld für den Status der automatischen Justage.

Die Eingabefelder der Abbruchkriterien umfassen die Parameter für

- die Anzahl der Algorithmusdurchläufe und
- die Intensitätsdifferenz, die die maximale Differenz zwischen der Röntgenstrahlungsintensität zweier aufeinanderfolgender Justageschritte angibt.

Auch die Suchbereiche des Algorithmus werden als Eingabefelder realisiert. Für jede Motorachse, die am Justageprozeß beteiligt ist, wird ein Feld bereitgestellt. Zusätzlich werden Eingabefelder zur Abfrage der Toleranz des „Goldenen Schnitt"-Algorithmus und zum Festlegen der Anzahl von Messungen zur Bestimmung der Röntgenstrahlungsintensität integriert.

Um die einzelnen vom Anwender zu manipulierenden Werte übersichtlich anzuordnen, werden Gruppen von Parametern durch Rahmen optisch von einander getrennt und mit der Gruppenkategorie beschriftet.

Abbildung 7.3 zeigt das Ergebnis des Dialogentwurfs mit Hilfe des „Resource Workshop". Nach der Dialogerstellung sind in der Datei `main.rc` die

Abbildung 7.3: Enwurf des Dialogfensters der Programmfunktion „Automatische Justage"

Beschreibungen der Dialogelemente unter dem Abschnitt *AutomaticAngleControl* zu finden.

Der folgende Ausschnitt zeigt einen Teil der Spezifikation des Dialogs in der Borland C++ 4.5 spezifischen Ressourcenbeschreibungssprache:

```
AutomaticAngleControl DIALOG 57, 18, 308, 254
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
      WS_SYSMENU
CLASS "BorDlg_Gray"
CAPTION "Automatische Justage"
FONT 8, "MS Sans Serif"
{
  .
  .
 EDITTEXT ID_Status, 14, 17, 163, 190, ES_MULTILINE |
          ES_AUTOVSCROLL | ES_AUTOHSCROLL | ES_READONLY |
          NOT WS_TABSTOP | WS_BORDER | WS_VSCROLL | WS_HSCROLL
 DEFPUSHBUTTON "Start der automatischen Justage",
          cm_start_justage, 40, 228, 110, 14
 PUSHBUTTON "Beenden", ID_CANCEL, 226, 228, 50, 14
```

```
 PUSHBUTTON "Hilfe", ID_Help, 226, 204, 50, 14
 CHECKBOX "Logdatei", ID_Logfile, 200, 24, 41, 11,
          BS_AUTOCHECKBOX | WS_TABSTOP
   .
   .
}
```

Der erste Teil eines Ressourceneintrages legt den Ressourcentyp fest (z.B. `CHECKBOX`). Danach folgt optional eine Beschriftung (im Beispiel `Logdatei`), die in Anführungsstrichen angegeben wird. Durch Kommata getrennt werden die Ressource-ID (im Beispiel `ID_Logfile`), die Abmessungen und gegebenfalls Flags zur besonderen Behandlung des Dialogelements aufgelistet (hier: `BS_AUTOCHECKBOX`, `WS_TABSTOP` - getrennt durch |).

Die Ressourcen-IDs zur Identifizierung einzelner Dialogelemente (beispielsweise `ID_Logfile` für die Checkbox zur Auswahl des Schreibens eines Protokolls) sind in der Datei `rc_def.h` in Form von *define*-Anweisungen gespeichert. Außerdem finden sich in dieser Datei auch die Kommando-IDs der Steuerkommandos, die der Dialog empfangen kann (z.B. `cm_start_justage` zum Start der automatischen Justage). Innerhalb des Projektes werden für die Dialogelemente vom Ressourcen Manager fortlaufend Nummern vergeben. Die Ressourcenbezeichnungen (IDs) dienen dazu, die Dialogelemente innerhalb des Programmtextes komfortabler und eindeutig ansprechen zu können. Im folgenden ist ein Ausschnitt der Datei `rc_def.h` dargestellt, der sich auf die Dialogelemente und Kommandos der Dialogs „Automatische Justage"bezieht.

```
   .
   .
//! AutomaticAngleControl: Kommandos
#define cm_start_justage           562
#define cm_set_parameters          574
#define cm_initialize              575
#define cm_choose_axis             576
#define cm_optimizing_TL           577
   .
   .
//! AutomaticAngleControl: Fensterelemente
#define ID_Status                  825
#define ID_Logfile                 826
#define ID_Help                    827
#define ID_CANCEL                  828
   .
```

```
  .
#define ID_Toleranz                564
#define ID_Durchlauf               565
#define ID_DF_Intervall            566
  .
  .
#define ID_IntenseDifferenz        569
#define ID_MeasureCount            570
  .
  .
```

Zur programmiertechnischen Umsetzung der Dialogsteuerung wird auf die vom ursprünglichen Programmierer der Steuersoftware zur Verfügung gestellten Dialogklassen zurückgegriffen, wobei die Dialogklasse zur Steuerung der automatischen Justage `TAutomaticAngleControl` vom Template[2] für modale Dialoge (Klasse `TModalDlg`) abgeleitet wird. Damit ist ein Grundgerüst eines modalen Dialogs vorhanden, für das die Konstruktoren und Destruktoren sowie die Memberfunktionen zur Initialisierung (DlgOnInit) und zur Nachrichtenverarbeitung (DlgOnCommand) lediglich an die eigenen Vorstellungen angepaßt werden müssen. Die Dialogtemplateklassen sind in den Dateien `dlg_tpl.h` und `dlg_tpl.cpp` definiert. In den Quelltextdateien `m_dlg.h` und `m_dlg.cpp` kann die Anwendung der Templates anhand der ursprünglichen Dialogklassen des Steuerprogramms nachvollzogen werden.

Der Programmentwickler hat die Struktur des Anwendungsmenüs des Steuerprogramms nicht mit dem Ressourcen Manager der Entwicklungsumgebung erzeugt. Stattdessen wird abhängig von den Einstellungen der ini-Datei das Programm-Menü mit Hilfe von Windows-API-Funktionen erstellt. Das geschieht in der Memberfunktion `LoadMenuBar` der Hauptprogrammklasse `TMain`. Solange diese Funktionsweise nicht geändert werden soll, müssen neue Menüpunkte in der Quelldatei `m_main.cpp` innerhalb dieser Funktion eingebunden werden.

Der Menüpunkt „*Automatische Justage...*" wird in das Untermenü des Menüpunkts „*Ausführen*" eingefügt. Die notwendigen Änderungen (Eintrag nach „`//!neu:`") in der Methode `TMain::LoadMenuBar` sind folgermaßen vorgenommen worden:

```
HMENU TMain::LoadMenuBar(void)

HMENU hmPL1,hmPL2;
```

[2]Klassen-Templates sind generische Klassen und stellen eine Schablone für die Implementierung eigener Klassen dar. Durch Vererbung können die grundlegenden Methoden übernommen und spezielle Methoden überschrieben bzw. angepaßt werden.

```
// Erzeugung eines Anwendungsmenüs
hTheMenu = CreateMenu();
// Unterpunkt: Ausführen
hmPL1 = CreatePopupMenu();
  .
  .
//!neu: Menüpkt Automat. Just.
AppendMenu(hmPL1,MF_STRING,cm_AutomaticAngleControl,
           "&Automatische Justage...");
  .
  .
// Einfügen der Menüpunkte zum Menü "Ausführen"
AppendMenu(hTheMenu,MF_POPUP,(WORD)hmPL1,"&Ausführen");
  .
  .
  .
```

In der folgenden Abbildung ist das Resultat der Menüerweiterung dargestellt.

Abbildung 7.4: Menüpunkt für die Programmfunktion „Automatische Justage"

Durch die Auswahl des Menüpunkts *Ausführen/Automatische Justage...* im Anwendungsmenü wird eine Nachricht des Typs `WM_COMMAND` mit der ID `cm_AutomaticAngleControl` erzeugt und von der Anwendung in der Hauptnachrichtenschleife verarbeitet. Die Verbindung des Menüeintrages mit der Ausführung des Dialogfensters muß in der Quelldatei `m_main.cpp` geschaffen werden. Dazu wird in der Verarbeitungsroutine für `WM_COMMAND`-Nachrichten

(Funktion `DoCommandsFrame`) ein *case*-Zweig eingefügt, der dazu dient, den Dialog der automatischen Justage zu starten.

```
LRESULT DoCommandsFrame(HWND hwnd,WPARAM wParam,LPARAM lParam)
{
  TModalDlg* dlg;

  switch(GET_WM_COMMAND_ID(wParam,lParam))
  {
     .
     .
    //!neu: Erzeugung des Dialogs der autom. Justage,
    //      Uebergabe der Steuerung an den Dialog und
    //      Rueckkehr nach Beendigung
    case cm_AutomaticAngleControl:
      dlg = (TAutomaticAngleControl *)
            new TAutomaticAngleControl();
      dlg->ExecuteDialog(hwnd);
      delete dlg;
      return TRUE;
     .
     .
  }
}
```

Danach kann die eingebundene Funktion „Automatische Justage" mit Hilfe des dazugehörigen Dialogfensters im Programm genutzt werden.

7.2.1 Windowsnachrichtenkonzept

Da die Anbindung der Funktion „Automatische Justage" an das RTK-Steuerprogramm über die Dialogprogrammierung erfolgt, ergeben sich einige Besonderheiten für die Implementation der Erweiterung.

Die Dialogerstellung und -verarbeitung erfolgt mit Hilfe des Windowsnachrichtenkonzepts. Das heißt, durch das Betätigen eines Dialogknopfes wird eine Nachricht erzeugt, die in die Nachrichtenschleife von Windows gelangt. Diese Nachricht wird in der Funktion `Dlg_OnCommand` für das jeweilige Fenster ausgewertet und der entsprechende Handler gestartet. Das bedeutet, daß der Kontrollfluß der „Automatischen Justage" mit Hilfe von Handlern realisiert werden muß. Dabei starten die Handler die jeweiligen Hauptaufgaben, die für die Durchführung der „Automatischen Justage" notwendig sind.

Diese Aufgaben wurden in Kapitel 6 beschrieben, wobei die Abb. 6.1 zum Verständnis der Abfolge hilfreich ist.

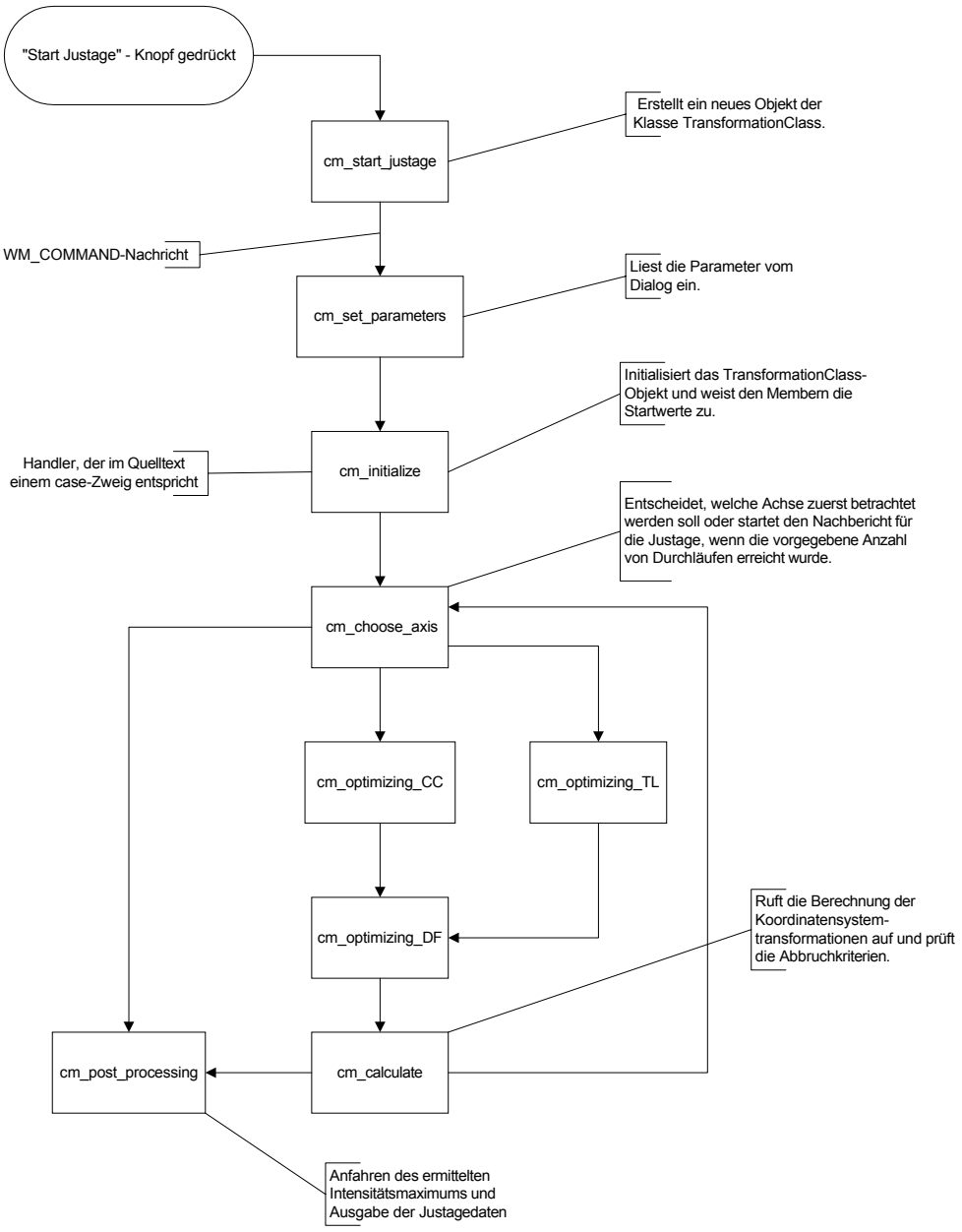


Abbildung 7.5: Kontrolle der „Automatischen Justage" durch das Senden von Nachrichten und der damit verbundenen Nacheinanderausführung der entsprechenden Handler (*case*-Zweige)

Die jeweiligen Handler werden mit Hilfe einer *switch-case*-Anweisung in der Funktion `TAutomaticAngleControl::Dlg_OnCommand` realisiert. Je nachdem welche Nachricht vorliegt, wird der entsprechende *case*-Zweig abgearbeitet und anschließend in die Nachrichtenschleife zurückgekehrt. In den *case*-Zweigen werden die Funktionen der „Automatischen Justage" aufgerufen. Um zu steuern, welche Aufgabe im nächsten Schritt der Justage durchgeführt werden soll, wird eine Nachricht mit dem Aufruf

```
FORWARD_WM_COMMAND(GetHandle(),MESSAGE,0,0,PostMessage);
```

verschickt, wodurch man erzwingt, welcher *case*-Zweig als nächstes abgearbeitet wird. Dadurch kann gemäß dem Windowsnachrichtenkonzept der Kontrollfluß der „Automatischen Justage" gesteuert werden. `MESSAGE` bezeichnet die jeweilige Nachricht, die in die Nachrichtenschleife gesandt wird. Die Abb. 7.5 veranschaulicht, wie durch das Senden von Nachrichten die „Automatische Justage" in der Funktion `TAutomaticAngleControl::Dlg_OnCommand` abgearbeitet wird.

7.2.2 Timerprogrammierung

Der Timer ist eine wichtige Komponente eines jeden PCs. Mittels eines Timerbausteins werden zeitkritische Abläufe wie zum Beispiel die Steuerung der RAM-Refreshzyklen realisiert. Innerhalb einer bestimmten Zeit löst der Timerbaustein ein Signal aus. Windows wandelt diese Art von Systemsignalen mit Hilfe des Kerneltreiber SYSTEM.DRV um. Dabei wird direkt in das Modul USER.EXE[3] verzweigt, wo mit einer Funktion aus diesem Modul entsprechend reagiert wird. Es gibt zwei Arten von Abläufen, wie auf ein Timersignal reagiert werden soll. Zum einen kann eine Nachricht WM_TIMER in der Nachrichtenschlange der Applikation abgelegt werden oder es wird eine bestimmte, vorher festgelegte Funktion beim Auslösen eines Timers abgearbeitet.

Im RTK-Steuerprogramm werden beide Arten von Timerprogrammierung verwendet. Allerdings handelt es sich hier nur um Timer, die in den Dialogen Verwendung finden. Für die zeitkritischen Prozesse zum Ansteuern der Motorenhardware wird eine ungünstige Variante in Form einer Delay-Funktion benutzt. Dabei handelt es sich um eine FOR-Schleife, die auf moderneren Rechnern schneller abgearbeitet wird.

Im File `m_dlg.cpp`, in dem der Dialog für die manuelle Justage realisiert wurde, wird die Variante des Versendens einer Nachricht mit anschließender Verarbeitung in der Nachrichtenschleife verwendet. Dadurch wird es möglich bei fahrendem Motor den aktuellen Winkel und die entsprechende Position

[3]SYSTEM.DRV und USER.EXE gehören zum Systemkern von Windows 3.1.

am horizontalen Schieberegler anzuzeigen. Der Timer wird gesetzt durch den Aufruf:

```
SetTimer(hwnd,TimerIdInformation,AskTime,NULL);
```

`hwnd` ist das Handle auf das aktuelle Dialogfenster. `TimerIdInformation` ist die ID des Timers. Darüber wird später identifiziert, um welchen Timer es sich handelt, wenn ein Timerevent ausgewertet wird. `AskTime` ist die Zeit in Millisekunden, in welchem Abstand ein Timerevent ausgelöst werden soll. Der vierte Parameter enthält die Prozedurinstanzenadresse im Fall, daß eine Bearbeitungsfunktion aufgerufen werden soll. Da in diesem Fall nur das Nachrichtenkonzept verfolgt wird, ist der Zeiger hier NULL.
Zerstört wird ein Timer durch den Aufruf:

```
KillTimer(hwnd,TimerIdInformation);
```

Bei den Parametern gelten die gleichen Beschreibungen wie oben.

Wird ein Timerevent ausgelöst, so wird dies im RTK-Steuerprogramm in der Funktion `Dlg_OnTimer` ausgewertet. Man nennt diese Nachrichten auch WM_TIMER-Nachrichten. Wie in der Tabelle 7.2 angegeben, sind für andere Nachrichten verschiedene Handler-Funktionen verantwortlich.

Nachrichtentyp	***Funktion zur Behandlung***
WM_INITDIALOG	`TModalDlg::Dlg_OnInit`
WM_COMMAND	`TModalDlg::Dlg_OnCommand`
WM_TIMER	`TModalDlg::Dlg_OnTimer`
WM_HSCROLL	`TModalDlg::Dlg_OnHScrollBar`
WM_VSCROLL	`TModalDlg::Dlg_OnVScrollBar`
WM_VSCROLL	`TModalDlg::Dlg_OnVScrollBar`

Tabelle 7.2: Nachrichtenbehandlung im RTK-Steuerprogramm

Bei einem Timerevent wird üblicherweise so vorgegangen, das zunächst der Timer mit `KillTimer` deaktiviert wird, anschließend wird die Funktionalität ausgeführt und danach der Timer wieder aktiviert. Würde der Timer

nicht deaktiviert werden, käme es zu einem Aufruf während der Abarbeitung der Timerfunktion. Dies würde die Abarbeitung nicht behindern, aber nach der Bearbeitung kommt es gleich wieder zur Abarbeitung eines Timerevents, denn an erster Stelle in der Nachrichtenschleife wäre immer eine WM_TIMER-Nachricht. Dadurch gebe es nur noch Aufrufe der `Dlg_OnTimer` Funktion. Die normale Funktionalität wäre nicht mehr möglich, da nur stets die Timerfunktionen ausgeführt würden.

Auf unterschiedliche Timer wird in der `Dlg_OnTimer` Funktion mit entsprechenden Event-Handlern reagiert. Realisiert wird dies mit Hilfe von switch-case Anweisungen, die die ID's der Timer auswerten. Da es auch andere Timerevents vom System gibt, ist diese Form der Auswertung unbedingt notwendig. Als Beispiel soll an dieser Stelle die Bearbeitungsfunktion für Timerevents aus der Dialogklasse `TAngleControl` (Dialog für die manuelle Justage) dienen. Alle 100 Millisekunden wird bei geöffnetem Dialogfenster „Manuelle Justage" ein Timer ausgelöst und eine Nachricht verschickt. Wird in der Nachrichtenschleife eine WM_TIMER-Nachricht verarbeitet, dann führt die Funktion `Dlg_OnTimer` dieser Klasse die Bearbeitung durch.

```
void TAngleControl::Dlg_OnTimer(HWND hwnd,UINT id) {
  char   buf[MaxString];
  double dDistance;

  switch(id)
  {
    case TimerIdInformation:
    KillTimer(hwnd,TimerIdInformation);
    if(mIsMoveFinish())
    {
      if(!mGetDistance(dDistance))
      {
        SetCursor(LoadCursor(NULL,IDC_WAIT));
        SetTimer(hwnd,TimerIdInformation,AskTime,NULL);
        return;
      }
      bLongMove = FALSE;
      sprintf(buf,mGetDF(),dDistance);
      SetDlgItemText(hwnd,id_NewAngle,(LPSTR)buf);
      SetDlgItemText(hwnd,id_Angle,(LPSTR)buf);
      SetScrollPos(BarHandle,SB_CTL,GetBarPos(),TRUE);
      bMoveActive = FALSE;
      SetFocus(BarHandle);
      SetCursor(LoadCursor(NULL,IDC_ARROW));
```

```
    }
    else
    {
      if(bLongMove)
      {
        sprintf(buf,mGetDF(),mGetDistanceProcess());
        SetDlgItemText(hwnd,id_Angle,buf);
      }
      SetCursor(LoadCursor(NULL,IDC_WAIT));
      SetTimer(hwnd,TimerIdInformation,AskTime,NULL);
    }
  }
};
```

In diesem Fall werden zwei Zustände getestet und entsprechend reagiert. Entweder der aktuelle Motor steht still oder er ist in Bewegung. Steht er still, wird die aktuelle Position ausgelesen. Funktioniert dies nicht, wird der Cursor in Form einer Sanduhr dargestellt und die Timerbearbeitungsfunktion beendet. Wurde die Position korrekt ausgelesen, so erfolgt eine Aktualisierung der Anzeige „Winkel" und „Neuer Winkel" mit der aktuellen Position. Zusätzlich findet ein Neuzeichnen des horizontalen Schiebereglers statt und der Cursor erhält eine neue Gestalt in Form eines Zeigers.

Im Fall daß der Motor in Bewegung ist, wird die Positon ausgelesen und in der Anzeige „Winkel" aktualisiert. Der Cursor bekommt in diesem Fall die Form einer Sanduhr.

Um zu zeigen, wie der Timer im Dialog „Manuelle Justage" funktioniert, soll das folgende Beispiel gegeben werden. Gibt man im Dialog „Manuelle Justage" einen neuen Winkel ein und bestätigt die Eingabe mit RETURN, dann wird der ausgewählte Antrieb an die entsprechende Position gefahren. Dabei kann beobachtet werden, wie sich bei fahrendem Antrieb der Wert der Anzeige „Winkel" ändert.

Zunächst wird in der Funktion `TAngleControl::Dlg_OnCommand` untersucht, in welchem Dialogelement die Eingabe getätigt wurde. Anschließend wird die Bewegung gestartet und eine Nachricht `cm_MoveButton` gesendet. Diese wird als nächstes in der Nachrichtenschleife bearbeitet. Das heißt, `Dlg_OnCommand` muß den entsprechenden Handler in Form einer Case-Anweisung aufrufen. Der wandelt den Cursor in eine Sanduhr und setzt den Timer.

```
  .
  .
case IDOK:
    if(!hDlgItem)
    {
      TModalDlg::Dlg_OnCommand(hwnd,id,hwndCtl,codeNotify);
      break;
    }
    // Der Nutzer hat mit Return eine neue Position, eine neue
    // Geschwindigkeit oder eine neue Schrittweite eingegeben
    for(;;)
    {
      .
      .
      if(hDlgItem == GetDlgItem(hwnd,id_NewAngle))
      {
        if(!bMoveActive)
        {
          GetDlgItemText(hwnd,id_NewAngle,(LPSTR)buf,10);
          bMoveActive = TRUE;
          bLongMove = TRUE;
          mMoveToDistance(atof(buf));
          FORWARD_WM_COMMAND(hwnd,cm_MoveButton,0,0,PostMessage);
        }
        break;
      }
      .
      .
    }
    hDlgItem = NULL;
    SetFocus(BarHandle);
    break;

case cm_MoveButton:
bMoveActive = TRUE;
SetCursor(LoadCursor(NULL,IDC_WAIT));
SetTimer(hwnd,TimerIdInformation,AskTime,NULL);
break;
    .
    .
```

Befindet sich das Programm in der Windowsnachrichtenschleife und werden keine weiteren Aktionen im Dialogfenster gestartet, so werden lediglich alle 100 Millisekunden Nachrichten durch den Timer ausgelöst. Durch die daraufhin abgearbeitete `Dlg_OnTimer`-Funktion wird festgestellt, ob der Motor in Bewegung ist und dementsprechend die Position im Dialogelement `id_angle` („Winkel") geändert. Hat der Antrieb seine Position erreicht, so wird die Position ausgelesen und in die Dialogelemente `id_angle` („Winkel") und `id_newangle` („Neuer Winkel") eingetragen. Bei Stillstand und erfolgreichem Auslesen der Position wird der Timer nicht mehr aktiviert (siehe `TAngleControl::Dlg_OnTimer`).

Die andere Form der Timerprogrammierung, die anfangs schon erwähnt wurde, veranlaßt Windows dazu, anstelle der Übersendung einer Nachricht direkt eine Bearbeitungsfunktion des Timerevents aufzurufen. Auch von dieser Möglichkeit wurde im RTK-Steuerprogramm Gebrauch gemacht. Initialisiert wird solch ein Timer mit folgendem Aufruf:

```
nEvent=timeSetEvent(nTimeTicks,1,(LPTIMECALLBACK)Funktion, 0,
                                                    TIME_PERIODIC);
```

Der Rückgabewert dieser Funktion vergibt die Kennziffer für das Timerevent. Dadurch kann immer ermittelt werden, um welches Timerevent es sich handelt. Der erste Parameter gibt an, wie oft ein Timersignal ausgelöst werden soll. Der zweite Parameter gibt die Kennziffer des Timers vor, das heißt, die Applikation bestimmt die Kennziffer. Wird an dieser Stelle `NULL` angeben, so weist Windows dem Timerevent automatisch eine Kennziffer zu. Der dritte Parameter der Funktion gibt die Prozedurinstanzenadresse der Funktion an, die beim Auslösen des Timerevents aufgerufen werden soll. Der vierte Parameter ist immer 0. Mit dem fünften Parameter wird angegeben, ob das Timerevent nur einmal oder immer wieder aufgerufen werden soll. Die möglichen Werte sind hier `TIME_ONESHOT` oder `TIME_PERIODIC`.
Deaktiviert wird diese Art von Timer mit dem Aufruf:

```
timeKillEvent(nEvent);
```

Der Parameter `nEvent` gibt die Kennziffer des jeweiligen Timerevents an.

7.2.3 Konsequenzen für die „Automatische Justage"

Die Möglichkeit eines Abbruchs der „Automatischen Justage" mit Hilfe eines Dialogknopfs gestaltet sich aufgrund des Windowsnachrichtenkonzeptes als äußerst schwierig zu implementieren. Da die „Automatische Justage" nach dem Start des Justagevorganges komplett die gesamte Prozeßzeit benötigt,

ist es unmöglich, überhaupt ein Dialogelement während dieses Vorganges zu betätigen. Der Grund dafür sind die Funktionen `MeasureIntensity` und `GetIntensityOnPosition` der Klasse `TransformationClass`. Dort werden Schleifen benutzt, um zu warten, bis die Messwerte vom Detektor anliegen bzw. bis die Motoren ihre anzufahrenden Positionen erreicht haben. Als Lösung dieses Problems käme die Einrichtung von Timerfunktionen in Frage. Das heißt, man würde die Schleifen entfernen und anschließend, nachdem die Motoren an ihre Positionen geschickt wurden, in die Nachrichtenschleife zurückkehren. Dort müsste ein Timer nach bestimmten Zeitintervallen abfragen, ob die Motoren alle ihre Positionen erreicht haben. Dadurch hätte man erreicht, das die Wartezeit nicht in der Schleife der Funktionen abläuft, sondern in der Nachrichtenschleife der Applikation.

Diese Art des Vorgehens würde aber gravierende Änderungen des Designs der Erweiterung „Automatische Justage" nach sich ziehen, denn die Funktionalität des Justagealgorithmus müßte komplett in die Dialogklasse implementiert werden, insbesondere die Ansteuerung der Motoren und des Detektors. Dies widerspricht aber den Paradigmen Wartbarkeit und Portabilität des Softwareengeneerings. Erstens würde die Lesbarkeit des Quelltextes sehr unübersichtlich werden, da die Funktionalität, wie zum Beispiel in der Funktion `GetIntensityOnPosition`, komplett über mehrere Event-Handler verteilt, implementiert werden müßte. Zweitens ließe sich die Erweiterung bei einer Umstellung auf ein 32-Bit Betriebssystem nur schwer realisieren, da sich dort die Dialogprogrammierung anderer Mittel bedient.

Die Autoren der „Automatischen Justage" haben sich deshalb dazu entschlossen, auf einen Abbruch des Justagevorganges mit einem Dialogknopf zu verzichten. Dadurch bleibt die Funktionalität getrennt von der Oberflächenprogrammierung. Dies erleichtert die Portabilität als auch die Wartbarkeit des Quelltextes durch andere Programmierer. Im Hinblick darauf, daß das RTK-Steuerprogramm auf 32-Bit umgestellt und unter dem Betriebssystem Windows98 benutzt werden soll, kann dann ein Abbruch des Justagevorganges durch einen Dialogknopf schnell und einfach mittels Thread Programmierung realisiert werden, da Windows98 präemptives Multitasking[4] besitzt.

7.3 Implementation der Funktionalität

Bei der Entwicklung der Programmfunktion „Automatische Justage" wurde nach dem inkrementellen Modell der Softwareentwicklung vorgegangen

[4]Beim präemptiven Multitasking können mehrere Threads in einem Zeitintervall ausgeführt werden, wobei jedem Thread ein Zeitfenster zugeteilt wird. Ist die Zeit für einen Thread abgelaufen, werden seine Daten gesichert und anschließend die Daten vom nächsten Thread geladen, der dann weiter ausgeführt wird.

(nähere Ausführungen sind in [2, S.120ff.] zu finden). Wie beim V-Modell steht beim inkrementellen Modell die vollständige Erfassung und Modellierung der Anforderungen am Anfang des Softwareprozesses. Der Unterschied zum V-Modell besteht darin, daß zu Beginn nur ein Teil der Anforderungen implementiert wird. Somit steht schon frühzeitig eine einsatzfähige, aber noch nicht vollständige Programmfunktion zur Verfügung, die durch das inkrementelle Hinzufügen weiterer Funktionen ergänzt werden kann. Die in der Designphase definierten Schnittstellen der Programmkomponenten bilden die Grundlage für eine sichere Erweiterung der Funktionalität, das heißt, hinzugefügte Programmkomponenten passen in das bisherige System.

So wurden für die konkrete Implementation der automatischen Justagefunktion zuerst die grundlegenden mathematischen Klassen `TMatrix`, `TVektor` und `TMatrizenListe` programmiert. Mit der Teil-Implementation der Klasse `TransformationClass` konnten bereits Transformationen von Koordinatensystemen durchgeführt werden. Die Transformationsklasse umfaßte zu diesem Zeitpunkt hauptsächlich die Methoden

- `Goldener_Schnitt`,
- `translate_to_worldpoints`,
- `translate_from_worldpoints` und
- `KoordinatenTransformation`.

Danach wurde zur Erstellung des Dialogfensters übergegangen. Der Dialog beinhaltete bereits die benötigten Dialogelemente, die aber nicht alle mit vollständiger Funktionalität ausgestattet waren. Zur Beurteilung der sinnvollen Anordnung und der Vollständigkeit reichte das aus. Die Dialogklasse `TAutomaticAngleControl` wurde danach sukzessive um die geforderten Funktionen zur Dialog- und Justagesteuerung erweitert. Die Ansteuerung der Motoren und Detektoren ist in Form der Methode `GetIntensityOnPosition` in die Klasse `TransformationClass` integriert worden. Diese Methode wiederum nutzt die Memberfunktion `MeasureIntensity`, um die Meßwerte für die Röntgenintensität vom Zähler auszulesen. Auch in die Initialisierung im Klassenkonstruktor wurden Motorsteuerungsfunktionen integriert, um die aktuellen Motorpositionen der Antriebe zu ermitteln und zu speichern.

7.3.1 Hinweise zur Detektoransteuerung

Die konkrete Realisierung der Detektoransteuerung stellte ein Problem dar, da die von der C-Schnittstelle `c_layer.h` zur Verfügung gestellten Funktionen nicht die erwartete Funktionalität besitzen. So bietet die C-Schnittstelle

zur Steuerung der Detektorgeräte keine Funktion an, die zum Ermitteln der Intensität der Röntgenstrahlung am 0-dimensionalen Detektor dienen könnte. Der Zugriff auf diese Größe kann nur über Klassenmethoden der `TDevice`-Klasse und der von ihr abgeleiteten Klassen erfolgen. Da in der Membervariablen *fIntensity* nicht die aktuelle Strahlungsintensität gespeichert ist, muß zur Ermittlung der aktuellen Größe explizit eine Messung gestartet, der Abrufvorgang (engl.: Polling) der Detektorkarte ausgelöst und der Wert der Variablen mit einer speziellen Methode ausgelesen werden. Eine vollständige Detektorabfrage ist mit dem anschließenden Programmcode realisierbar.

```
TDevice* Detektor;
float fXRayIntensity;

// Messung starten
Detektor->MeasureStart();

// Detektorkarte auslesen
while (Detektor->PollDevice() != R_MeasOk)

// Roentgen-Intensitaet ermitteln
while (Detektor->GetData(fXRayIntensity) != R_OK)
```

Die Ermittlung der Röntgenstrahlungsintensität mit einem 0-dimensionalen Detektor läuft nach folgendem Schema ab:

1. Die Messung an der Detektorkarte wird gestartet, wobei die Möglichkeit besteht, länger andauernde Messungen auszulösen.

2. Das Auslesen der Detektorkarte wird veranlaßt. Ist das Auslesen abgeschlossen, wird der ermittelte Zählerwert in der `TDevice`-Membervariablen *fIntensity* gespeichert. Bei korrekter Funktionsweise kehrt die Methode `PollDevice()` mit dem Rückgabewert `R_MeasOk` zurück.

3. Ist die gespeicherte Intensität gültig, liefert die Methode `GetData()` den Wert der `TDevice`-Membervariablen *fIntensity* im Übergabeparameter zurück. Die Methode wird dann mit der Rückgabe `R_OK` beendet.

Zur Feststellung der Strahlungsintensität wird ein 0-dimensionaler Detektor (der Szintillationszähler SCSCS-2 der Firma Radicon) verwendet, für den die gesonderte Detektorklasse (`TRadicon`) existiert, die von der Klasse `TDevice` abgeleitet ist. Zur Laufzeit werden somit im obigen Programmbeispiel die Methoden des Detektorobjekts vom Typ `TRadicon` aufgerufen.

Beim praktischen Test der Röntgenintensitätsbestimmung trat eine eine weitere Schwierigkeit auf. Wenn während der Ermittlung der Detektorwerte mit den oben genannten Befehlen das Zählerfenster nicht geöffnet war, konnten keine Strahlungswerte ermittelt werden. Das Zählerfenster dient der visuellen Rückmeldung für die am Detektor anliegende Röntgenstrahlung. Nur wenn es angezeigt wird, ist durch die Timer-Routine sichergestellt, daß die Werte der Röntgenstrahlungsintensität ständig aktualisiert werden. Deshalb wird bei der Initialisierung des Dialoges „Automatische Justage" (Methode `TAutomaticAngleControl::Dlg_OnInit`) für den Fall, daß der Detektor korrekt ins System eingebunden ist (Detektor-Schnittstellenfunktion `dlIsDeviceValid`), abgefragt, ob auch das Zählerfenster angezeigt wird (Test der Detektorklassenvariablen *bDeviceOpen*) und gegebenenfalls ein neues Fenster vom Typ `TCounterWindow` erzeugt. Der folgende Programmabschnitt zeigt die programmtechnische Umsetzung des beschriebenen Vorgehens.

```
BOOL TAutomaticAngleControl::Dlg_OnInit(HWND hwnd, HWND hwndCtl,
                                        LPARAM lParam)
{
  TModalDlg::Dlg_OnInit(hwnd, hwndCtl, lParam);
  // Verwendung des akt. Zaehlergeraets aus der Deviceliste
  Sensor = lpDList->DP();
  :
  // Test, ob Zaehler korrekt im Programm initialisiert wurde
  if( !dlIsDeviceValid(CounterDevice) )
  {
    :
  }
  else
  {
    strcat(status, "Counterdevice...ok!");
    // falls Zaehlerfenster nicht geoeffnet ist,
    // werden die Zaehlerwerte nicht korrekt eingelesen
    if(!Sensor->bDeviceOpen)
    {
      // Zaehlerfenster oeffnen
      NewWindow((TCounterWindow *)
                new TCounterWindow(Sensor->GetId()));
    }
  }
  :
};
```

Kapitel 8

Test

Jede Software die nach den Kriterien des Softwareengeneerings erstellt wird, durchläuft mehrere Stadien im Laufe ihrer Entwicklung. In den meisten Fällen dient das Wasserfallmodell [16] als Vorschrift für den Entwicklungsprozeß der Software . Dabei muß es sich nicht nur um eine Neuentwicklung der Software handeln, auch bei der Erweiterung von Funktionalität eines bestehenden Programmes sollte Software nach ingenieursmäßigen Gesichtspunkten erstellt werden.

Nachdem in den vorangegangenen Kapiteln die ersten drei Phasen der Entwicklung - Analyse & Definition, Design und Implementation - dokumentiert wurden, folgt in diesem Kapitel eine oft vernachlässigte Phase des Entwicklungsprozesses, der Softwaretest.

8.1 Testziele

Zur Überprüfung der Funktionsweise der Programmerweiterung „Automatische Justage" mußte ein umfangreicher Test durchgeführt werden. Dabei wurden drei Testziele ins Auge gefaßt:

1. Zuerst sollte der Algorithmus in unterschiedlichen Anwendungssituationen seine Tauglichkeit unter Beweis stellen. Es ging darum, sicherzustellen, daß mit der „Automatischen Justage" verschiedene Proben optimal für einen Topographie-Vorgang eingestellt werden können. Da abzusehen war, daß nicht alle Proben mit dieser Justage eingestellt werden können, ging es darum, eine Aussage darüber zu finden, welcher Prozentsatz von Proben mit dem Algorithmus automatisch justiert werden kann.

2. Außerdem sollten mit Hilfe des Tests der „Automatischen Justage" Schwächen des Suchalgorithmus offenbart werden. Anhand dieser Er-

gebnisse kann im Verlauf der Weiterentwicklung des Steuerprogrammes die „Automatische Justage" verbessert werden.

3. Wichtig war es auch, zu untersuchen, ob der Dialog der „Automatischen Justage" das von ihm geforderte Verhalten zeigte. Das heißt, es galt zu untersuchen, ob der Dialog gegen unterschiedliche Eingabewerte, die laut Spezifikation nicht vorgesehen sind, robust ist.

8.2 Teststrategie

Für den Test wurden verschiedene Strategien angesetzt. Zum einen sollten Tests mit unterschiedlichen Proben durchgeführt werden und zum anderen Tests, die der Überprüfung der Benutzerschnittstelle (Dialogfenster) dienen sollten. Die Testsszenarien, die mit den Proben absolviert wurden, gliedern sich in zwei Kernpunkte bezüglich der dabei verfolgten Strategie:

1. Test mit probenunabhängigen Einstellungen. Darunter fallen wiederum 2 unterschiedliche Fälle, die es zu testen galt:

   (a) Die Apparatur betreffende Einstellungen:
   Hierbei wurde untersucht, welche Auswirkung die Anzahl der Messungen pro Meßschritt auf die Dauer der Justage hat und wie hoch die Fehleranfälligkeit ist, wenn nur ein Meßwert pro Position genommen wird.

   (b) Algorithmusspezifische Einstellungen:
   Hierunter zählen die Parameter, die durch ihre Variation den Algorithmus selbst noch beeinflussen. Dazu gehören die Parameter Toleranz, Durchläufe.

2. Probenabhängige Einstellungen:
   Bei den probenabhängigen Einstellungen werden die Parameter des Suchbereichs variiert. Damit soll festgestellt werden, welche Einstellungen die günstigsten sind, so daß sie für die meisten Voreinstellungen der Proben verwendet werden können. Als weiteres Ergebnis dieser Tests erhält man eine Aussage darüber, wieviele Proben überhaupt korrekt eingestellt wurden.

Der Abbruchparameter „Maximale Intensitätsdifferenz" stellt eine algorithmusspezifische Einstellung dar, die aber auch von den zu justierenden Proben abhängig ist. Bei schwierig einzustellenden Proben könnte es sinnvoll sein, die Justage nicht abbrechen zu lassen und so möglicherweise doch zu einem guten Ergebnis zu gelangen.

Bei der Betrachtung des Abbruchparameters wurde so vorgegangen, daß während der Tests der Parameter, bis auf zwei Ausnahmen, immer auf **aus** gestellt blieb. Durch Beobachtung der Justageergebnisse wurde eine Aussage darüber getroffen, ob die automatische Justage mit eingeschaltetem Abbruchkriterium eine optimale Einstellung der Probe vorgenommen hätte. Das würde dann bedeuten, daß mit Hilfe des Kriteriums die Dauer der Justage verkürzt worden wäre.

Um das Kriterium ausreichend beurteilen zu können, wurden die möglichen Auswirkungen des Parameters für die während des Tests zur Verfügung gestellten Proben gesammelt.

Für die Testszenarien des Dialoges wurden keine Proben benötigt, denn bei diesen Tests kam es nicht darauf an den Suchalgorithmus zu untersuchen, sondern es galt festzustellen, wie robust der Dialog der „Automatischen Justage" gegen falsche Eingaben ist. Das RTK-Steuerprogramm unterstützt sowohl für die Antriebe als auch für die Detektoren Testgeräte, die die Funktionalität der echten Hardware vortäuschen. Der nächste Abschnitt wird dieses Thema behandeln.

8.3 Test des Dialoges ohne Proben

Der Test des Dialogs „Automatische Justage" dient der Überprüfung der korrekten Funktionsweise der Benutzerschnittstelle. Es mußte sichergestellt werden, daß die Erweiterung robust gegenüber Fehleingaben des Benutzers ist. Der Nutzer interagiert zur Steuerung der automatischen Justage mit dem Dialogfenster und kann dort Parameter aktivieren, Werte eingeben und ändern. Außerdem kann der Anwender eine Hilfe zum Justagedialog abrufen und den Vorgang der Justage starten.

Um die korrekte Arbeitsweise der Eingabeschnittstelle zu testen, ist es nicht zwingend notwendig, die reale Hardware ansteuern zu lassen. Es reicht aus, wenn die von der Software bereitgestellten Testgeräte verwendet werden. Das Steuerprogramm kann für jeden Antrieb einen Motor vom Typ `TMotor` verwenden, der dann einen Testmotor darstellt, mit der gleichen Funktionalität wie reale Motoren. Es handelt sich dabei aber nicht um simulierte Hardware, sondern um eine softwaremäßig umgesetzte Funktionsweise.

Das bedeutet, daß ein Testmotor nicht die Bewegung simuliert, um eine Position anzufahren, sondern sich sofort an dieser Position befindet. Das bringt u.a. einen hohen Zeitgewinn für den Test. Die Einstellungen zur Einbindung der Testantriebe werden in der Initialisierungsdatei vorgenommen. Ein entsprechender Eintrag in der Datei `develop.ini` würde für die Antriebsachse „Beugung fein" (DF) folgendermaßen lauten:

```
[Motor0]
Name=DF
Unit=Sekunden
;Type=C-812ISA
Type=TMotor
  .
  .
```

Als Motortyp wird lediglich `TMotor` eingetragen. Um diese Änderung später leicht rückgängig machen zu können, kann der vorherige Eintrag durch Voranstellen eines Semikolons auskommentiert werden.

Der zur Justage benötigte 0-dimensionale Röntgendetektor steht ebenfalls als per Software simuliertes Zählergerät zur Verfügung. Die zugehörige Klasse `TDevice` ist vom Programmierer der Steuersoftware implementiert worden. Dieser Zähler stellt eine sehr einfache Simulation eines Detektors dar und liefert Intensitätswerte, die nur in Abhängigkeit von den Positionen der Achsen „Beugung fein" und „Tilt" berechnet werden.

Im Softwaresanierungsprojekt hat sich ein Student mit der Implementation eines Testdetektors auseinandergesetzt, der Röntgenintensitätswerte liefert, die an einer realen Probe in Abhängigkeit von der Stellung der Motorachsen „Tilt", „Beugung fein" und „Kollimator" gemessen wurden [15].

Die neu implementierte Klasse `Testdev` ist zur Simulation des Detektorverhaltens besser geeignet, da sie auf den Daten einer existenten Probe aufbaut und alle an der Feinjustage beteiligten Antriebe berücksichtigt. Deshalb wurde während der Entwicklungs- und Erprobungsphase des Automatischen Justage-Algorithmus der in der oben genannten Studienarbeit programmierte Testdetektor benutzt. Der Zähler wird mittlerweile auch aufgrund des erfolgreichen Einsatzes bei der Simulation der Justage als Ersatz zum einfachen Testdetektor angewendet.

Um den Simulationsdetektor zu verwenden, müssen wiederum Änderungen an der Datei `develop.ini` vorgenommen werden.

```
[Device0]
Name=Counter
;Type=Radicon
;Type=TDevice
Type=Test
  .
  .
```

Soll der Hardwaredetektor verwendet werden, muß als Typ `Radicon` eingetragen werden. Für den einfachen Softwaredetektor kann ein beliebiger

Typ angegeben werden, vorausgesetzt, die Bezeichnung unterscheidet sich von den im Programm definierten Detektortypen (z.B. `TDevice`). Wird der erweiterte Testzähler aus der oben genannten Studienarbeit benutzt, hat der Detektor den Typ `Test`.

Der Funktionstest des Dialogs wurde mit der folgenden Strategie durchgeführt:

1. Eingabe von korrekten Parameterwerten, die innerhalb der Bereichsgrenzen der einzelnen Dialogparameter liegen
2. Eingabe von Parameterwerten, die außerhalb der Bereichsgrenzen liegen
3. Eingabe von nicht erlaubten Parameterwerten (Buchstaben, Zeichen, Mix von Buchstaben,Zeichen und Zahlen)
4. Bedienung der Schaltelemente
   a) Drücken von Start, Beenden, Hilfe
   b) Bewegen der Schieberegler des Statusfensters
5. Untersuchung der Option „Logdatei"
   a) wird bei aktiviertem Optionsfeld die Protokolldatei `justage.log` ins Programmverzeichnis geschrieben
   b) werden die Justagedaten korrekt in das Protokoll übernommen
   c) wird eine bestehende Logdatei korrekt erweitert
6. Untersuchung der Option „maximale Intensitätsdifferenz"
   a) läuft die automatische Justage bei deaktiviertem Optionsfeld mit der gesamten Anzahl von Durchläufen ab
   b) wird bei aktiviertem Optionsfeld die Justage abgebrochen, wenn bei aufeinanderfolgenden Algorithmusdurchläufen die Röntgenintensität um mehr als den angegebenen Wert absinkt
   c) wird bei deaktiviertem Optionsfeld die Eingabe von Werten für die Intensitätsdifferenz verhindert

Nach Absolvierung der Tests traten einige Probleme auf, von denen die schwerwiegenden aber in einem darauffolgenden Korrekturschritt in den Programmdateien korrigiert wurden.

So stürzte das Programm im Testfall 4a) nach mehrmaligem Start der Justage (ca. 25 Mal) mit einem Speicherschutzfehler ab. Die Ursache wurde

in dem Programmteil lokalisiert, der die Informationen in das Statusfenster schreibt. Bis zum Zeitpunkt des Dialogtests wurde der Status fortwährend auch nach einem Neustart der Justage an den bestehenden Statusfensterinhalt angehängt. Nachdem der zur Darstellung benutzte Zeichenpuffer mit mehr als 32000 Zeichen gefüllt war, kam es zum Absturz.

Das Problem wurde dadurch umgangen, daß beim wiederholten Start der Justage der Inhalt des Statusfensters gelöscht wird und somit keine Gefahr besteht, daß es zum Überlauf des Puffers kommt.

Bei der Eingabe von Parametern mit gemischten Zeichen und Zahlen werden Zahlen zu Beginn der Eingabe akzeptiert und einer weiteren Bereichsüberprüfung unterzogen.

Die Eingabe von „`67ui`" beispielsweise im Parameterfeld des Kollimatorsuchbereichs wird als Zahl `67` angenommen. Dieser Wert liegt innerhalb des erlaubten Bereichs und wird als Parameter zur Justage verwendet. Dagegen wird die Eingabe von „`u67i`" nicht akzeptiert und der entsprechende Voreinstellungswert als Parameter eingesetzt.

Dieses Dialogverhalten ist zwar nicht hundertprozentig korrekt, kann aber unter Berücksichtigung der geringen Wahrscheinlichkeit solcher Fehleingaben akzeptiert werden.

Ansonsten sind alle oben beschriebenen Tests erfolgreich verlaufen. Parametereingaben, die außerhalb der zugelassenen Wertebereiche liegen, werden auf die Voreinstellungen bzw. auf die oberen und unteren Grenzen zurückgesetzt (Testfälle 1 bis 3). Bei eingeschalteter Option „Logdatei" wird eine Protokolldatei `justage.log` im aktuellen Programmverzeichnis angelegt bzw. eine bereits bestehende Datei ergänzt (Testfall 5). Die protokollierten Daten werden korrekt in die Datei geschrieben. Wenn das Optionsfeld deaktiviert ist, wird keine Protokolldatei erzeugt.

Die Option „maximale Intensitätsdifferenz" führt zu dem erwarteten Justageverhalten (Testfall 6). Bei Aktivierung folgt ein frühzeitiger Abbruch beim Verschlechtern der Intensitätswerte um einen bestimmten Wert. Ist die Option deaktiviert läuft die Justage mit der vorgegebenen Anzahl von Durchläufen ab.

Abschließend kann also festgehalten werden, daß der Dialog „Automatische Justage" robust gegenüber fehlerhaften Eingaben des Benutzers ist und das erwartete Verhalten zeigt.

8.4 Testfälle mit Proben

Der Justagealgorithmus wurde mit unterschiedlichen Probenarten und verschiedenen Parametereinstellungen getestet. Die Proben waren bis auf eine

Schichtsysteme. Die Schichtproben bestanden zum einen aus einer Silizium(**Si**)- Schicht mit aufgebrachter Silizium-Germanium(**SiGe**)- Schicht und zum anderen aus einer Gallium-Arsenid(**GaAs**)- Schicht mit aufgebrachter **CuAsSe**$_2$-Schicht. Die einzige Probe, die kein Schichtsystem war, bestand aus einem planen Siliziumkristall, der als Kollimator eingesetzt wurde und der Untersuchung seiner Topographie unterzogen werden sollte.

8.4.1 Test der probenunabhängigen Einstellungen

Zum Test des Zeitverhaltens und der Güte der Justage werden für eine repräsentative Probe[1] die verschiedenen probenunabhängigen Parameter variiert.

Es wurden folgende Einstellungen getestet:

- Toleranz (steht für die Größe des kleinsten Intervalls bei der Optimierung mit dem Verfahren des Goldenen Schnitts):
  minimal **0.1** und maximal **1**,
- Anzahl der Durchläufe (ein Durchlauf ist ein Iterationsschritt des Justagealgoritmus):
  minimal **1** und maximal **7**,
- Anzahl der Messungen pro Röntgenintensitätsbestimmung:
  minimal **1** und maximal **5**,
- Maximale Intensitätsdifferenz (ist der aktuell gemessene Intensitätswert um mehr als diese Differenz schlechter als der im Durchlauf zuvor erreichte Wert, bricht der Algorithmus ab):
  minimal **0** und maximal **9999**.

Aus den Wertebereichen der Parameter sind folgende, für die Einstellung der Probe sinnvolle Testwerte ausgewählt worden:

- Toleranz: 0.1 / 0.5 / 1.0
- Durchläufe: 3 / 5 / 7 (erst ab 3 sinnvoll, siehe Erläuterungen im Text)
- Anzahl der Intensitätsmessungen: 1 / 2 / 3 / 5
- Maximale Intensitätsdifferenz: aus (es wurde beobachtet, ob ein Abbruch nötig gewesen wäre, für den Fall, daß die Intensität stark abfällt).

[1] Eine Probe die schon sehr gut automatisch justiert werden konnte.

Es ist beispielsweise nicht sinnvoll, den Algorithmus mit weniger als 3 Durchläufen auszuführen. Für einen bzw. zwei Durchläufe müßte sich die Probe in einer sehr nahen Umgebung des Maximums befinden. Davon kann aber im normalen Anwendungsfall nicht ausgegangen werden.

Die unterschiedlichen Anzahlen der Messungen zur Bestimmung der Intensität spiegeln verschiedene Szenarien wider. Wird nur eine Messung durchgeführt, findet keine Korrektur eventueller Meßwertschwankungen statt. Im Fall von zwei Messungen wird der Mittelwert der gemessenen Intensitäten bestimmt. Diese Methode eleminiert kleine „Ausreißer" und kann somit auch nicht als optimale Variante angesehen werden. Wenn mehr als drei Meßwerte an einer Stelle genommen werden, ermittelt das Programm den Median und kann dadurch „Ausreißer" sehr gut ausschließen. Eine Ermittlung der Röntgenintensität mit der Maximalanzahl von fünf Messungen dient der Beurteilung des Einflusses des Parameters auf die Justagedauer.

Um die probenunabhängigen Parameter zu testen, wurde der Suchbereich fest gelassen. Die Intensitätsdifferenz blieb, wie zuvor beschrieben, auch ausgeschaltet. Folgende Tabelle soll einen Überblick über die ausgesuchten Testfälle geben. Es wurde darauf geachtet, daß die erzeugten Testfälle eine sinnvolle Kombination bezüglich der Variation der Werte ergeben.

Testfall Nr.	1	2	3	4	5	6	7	8	9	10
Toleranz	1	1	1	1	0.1	0.1	0.1	0.5	0.5	0.5
Durchläufe	3	5	7	7	7	5	3	5	5	5
Anzahl der Messg.	3	2	1	5	3	3	3	3	2	1

Die Testfälle sind jeweils in einer Tabelle zusammengefaßt. Im oberen Teil sind die Parameter zu finden, die im Dialogfenster verändert werden können. Um besser zu erkennen, welche Werte zum vorherigen Testlauf verändert wurden, sind die entsprechenden Parameter fett hervorgehoben. Dies soll die Lesbarkeit der einzelnen Testdaten vereinfachen.

Im unteren Teil der Tabelle sind die Anfangs- und Endwerte verzeichnet. Das sind zum einen die Positionwerte für die Antriebe „Beugung fein"(DF), „Tilt"(TL) und „Kollimator" (CC) und zum anderen die Intensität der Strahlung und die Halbwertsbreite. Zur Überprüfung des Justageergebnisses wurde die Probe nach dem Justagevorgang mit einem 2-dimensionalen Detektor, der an einen Monitor angeschlossen war, untersucht. Bei voller Ausleuchtung der Probe hat man eine perfekte Einstellung. Dies würde bedeuten, daß über der gesamten Probe die Strahlung gemäß der Bragg-Bedingung reflektiert wird. Diese Testfälle wurden mit einem Pfeil unter der jeweiligen Spalte gekennzeichnet.

Für die folgende Testreihe stand ein aus Silizium mit aufgebrachter SiGe-Schicht bestehender Halbleiter zur Verfügung. Die Probe hatte die interne Bezeichnung PF 916324/08/5.

Testfall 1					
Dialogparameter		Toleranz:	1	Suchbereich	
		Durchläufe:	3	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	3	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	7.77	DF:	5.74		
TL:	-5.23	TL:	-10.74		
CC:	50	CC:	-21.04		
Intensität:	≈22700	Intensität:	≈32000		
HWB:	22.43	HWB:	14.96		
Dauer der Justage:		7:17 min			

Bemerkung: Keine optimale Justage. Vermutung: Die Toleranz ist zu hoch, um innerhalb von drei Durchläufen in die Nähe des Maximums zu gelangen. Im nächsten Tesfall wird deshalb die Anzahl der Durchläufe erhöht.

Testfall 2					
Dialogparameter		Toleranz:	1	Suchbereich	
		Durchläufe:	**5**	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	**2**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	7.77	DF:	k.A.		
TL:	-5.23	TL:	k.A.		
CC:	50	CC:	k.A.		
Intensität:	≈23500	Intensität:	≈29000		
HWB:	22.43	HWB:	k.A.		
Dauer der Justage:		ca. 9 min			

Bemerkung: Die Startwerte waren dieselben wie im Testfall 1, bis auf die Anzahl der Durchläufe und der Intensitätsmessungen. Bei diesem Test stürzte das Programm am Ende der Justage ab. Die Ursache wurde lokalisiert und es stellte sich heraus, daß der Absturz nicht dem Programm anzulasten war. Der Testdurchlauf war jedoch so weit fortgeschritten, daß eine Einschätzung der Justagedauer und Probenausleuchtung möglich war. Bei der Analyse des Logfiles fiel auf, daß der Algorithmus beim Regeln des Kollimators die falsche Richtung wählte. Die letzte Intensität betrug ca. 22000, während das Maxi-

mum des Justagelaufs bei 29000 lag. Der über zwei Messungen gemittelte Intensitätswert hat nach Auswertung der Protokolldatei keine „Ausreißer" zu kompensieren gehabt und ist nicht ursächlich für die schlechte Justage verantwortlich.

Die unterschiedlichen Startwerte der folgenden Testläufe sind dadurch begründet, daß durch den Absturz und nachfolgenden Neustart des Programms die Positionswerte der Antriebe verstellt wurden. Der Ausgangspunkt für die Justage ist jedoch in etwa der gleiche, nur die Positionswerte unterscheiden sich von den vorhergehenden Werten.

Testfall 3					
Dialogparameter		Toleranz:	1	Suchbereich	
		Durchläufe:	**7**	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	**1**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	10.01	DF:	16.37		
TL:	-8.97	TL:	-7.9		
CC:	-105.50	CC:	-174.29		
Intensität:	≈25000	Intensität:	≈44000		
HWB:	17.6	HWB:	11.8		
Dauer der Justage:		13:39 min			

Bemerkung: Da nur eine Messung zur Intensitätsbestimmung durchgeführt wurde, lief die Justage auch bei sieben Durchläufen in vertretbarer Zeit ab. Die Probe ist noch nicht optimal eingestellt. Das zeigen die nicht komplette Ausleuchtung der Probe und die Halbwertsbreite.

Testfall 4					
Dialogparameter		Toleranz:	1	Suchbereich	
		Durchläufe:	7	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	**5**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	17.66	DF:	16.37		
TL:	-8.59	TL:	-7.9		
CC:	-103.78	CC:	-174.29		
Intensität:	≈25300	Intensität:	≈45000		
HWB:	17.6	HWB:	9.61		
Dauer der Justage:		21:59 min			

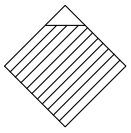
Bemerkung: Die Justage dauerte sehr lang. Das ist auf die Anzahl der Messungen gepaart mit der hohen Anzahl von Durchläufen zurückzuführen. Die Probe ist gut ausgeleuchtet, was sich auch in einer guten Halbwertsbreite widerspiegelt.

Testfall 5					
Dialogparameter		Toleranz:	**0.1**	Suchbereich	
		Durchläufe:	7	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	**3**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	21.58	DF:	24.36		
TL:	-8.6	TL:	-9.46		
CC:	-105.67	CC:	-46.33		
Intensität:	≈28000	Intensität:	≈43000		
HWB:	17.6	HWB:	9.68		
Dauer der Justage:		21:37 min			

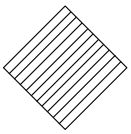
Bemerkung: Die Toleranz und die Anzahl der Durchläufe sind ausschlaggebend dafür, daß die Justage sehr lange dauerte. Ausleuchtung und Halbwertsbreite der Probe sind gut.

Testfall 6					
Dialogparameter		Toleranz:	0.1	Suchbereich	
		Durchläufe:	**5**	DF:	20
		max. Intensitätsdiff.:	**5000**	TL:	20
		Anzahl der Messg.:	3	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	30.24	DF:	26.88		
TL:	-11.3	TL:	-5.4		
CC:	-90	CC:	-17.97		
Intensität:	≈30000	Intensität:	≈46000		
HWB:	14.97	HWB:	9.01		
Dauer der Justage:		15:18 min			

Bemerkung: Durch die Justage wurde eine sehr gute Halbwertsbreite für diese Probenart und hundertprozentige Ausleuchtung erreicht. Der Algorithmus benötigte trotz testweise eingeschaltetem Abbruchkriterium „Maximale Intensitätsdifferenz" die volle Anzahl der Durchläufe zur Probeneinstellung. In diesem Fall hätte das Kriterium keine Zeitersparnis gebracht. Die Verminderung der Anzahl der Durchläufe führte zu einer Verringerung der Justagedauer.

Testfall 7					
Dialogparameter		Toleranz:	0.1	Suchbereich	
		Durchläufe:	**3**	DF:	20
		max. Intensitätsdiff.:	**aus**	TL:	20
		Anzahl der Messg.:	3	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	21.34	DF:	25.26		
TL:	-8.6	TL:	-8.24		
CC:	-105	CC:	-42.13		
Intensität:	≈28000	Intensität:	≈43000		
HWB:	17.6	HWB:	9.68		
Dauer der Justage:		9:43 min			

Bemerkung: Die Anzahl von drei Durchläufen wirkt sich günstig auf die Dauer der automatischen Justage aus, verhindert aber ein besseres Ergebnis. Trotzdem erreichte die Justage eine gute Halbwertsbreite bei fast vollständiger Ausleuchtung der Probe.

Testfall 8					
Dialogparameter		Toleranz:	**0.5**	Suchbereich	
		Durchläufe:	5	DF:	20
		max. Intensitätsdiff.:	aus	TL:	20
		Anzahl der Messg.:	3	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	29.4	DF:	27.41		
TL:	-11	TL:	-3.15		
CC:	-90	CC:	-5.32		
Intensität:	≈31000	Intensität:	≈46000		
HWB:	14.6	HWB:	8.87		
Dauer der Justage:		12:58 min			

Bemerkung: Dieser Test erreichte das beste Justageergebnis mit dieser Probe. Die Probe ist zu 100% ausgeleuchtet, und die Halbwertsbreite ist sehr gut. Der eingestellte Wert der Toleranz auf 0.5 stellt keine Verschlechterung der Optimierung dar. Im Verlauf des Justageprozesses trat ein Intensitätssprung auf (die Intensität sank auf 23000). Danach bewegte sich die Optimierung aber wieder im Bereich des Maximums. Bei Aktivierung des Parameters „Maximale Intensitätsdifferenz" wäre der Justageprozeß zu früh abgebrochen worden.

Testfall 9					
Dialogparameter		Toleranz:	0.5	Suchbereich	
		Durchläufe:	5	DF:	20
		max. Intensitätsdiff.:	**5000**	TL:	20
		Anzahl der Messg.:	**2**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	29.84	DF:	22.86		
TL:	-11	TL:	-7.89		
CC:	-90	CC:	6.91		
Intensität:	≈31000	Intensität:	≈46000		
HWB:	14.6	HWB:	8.93		
Dauer der Justage:		10:27 min			

Bemerkung: Die Verringerung der Anzahl der Meßwerte zur Bestimmung der Röntgenintensität führte zu einer Beschleunigung der Justage bei gleichbleibend guter Optimierung. Im Laufe dieses Testdurchlaufs wurde das Maximum schon im zweiten Durchgang gefunden. Im Durchlauf 3 kam es zu einem Intensitätseinbruch, der bei eingeschaltetem Intensitätsdifferenz-Kriterium zum Abbruch geführt hätte. Danach wurden aber wieder Intensitäten in der Nähe des Maximums erreicht, so daß der Abbruch nicht gerechtfertigt gewesen wäre.

Testfall 10					
Dialogparameter		Toleranz:	0.5	Suchbereich	
		Durchläufe:	5	DF:	20
		max. Intensitätsdiff.:	5000	TL:	20
		Anzahl der Messg.:	**1**	CC:	100
Startwerte		**Endwerte**		**Ausleuchtung**	
DF:	29.62	DF:	31.95		
TL:	-11	TL:	-3.2		
CC:	-90	CC:	-19.21		
Intensität:	≈31000	Intensität:	≈45000		
HWB:	14.6	HWB:	9.29		
Dauer der Justage:		7:31 min			

Bemerkung: Da nur eine Intensitätsmessung durchgeführt wurde, ist diese Justage in kurzer Zeit absolviert worden. Das Ergebnis der Justage ist sehr gut.

8.4.2 Bewertung der Tests mit probenunabhängigen Einstellungen

Die Auswertung der Testfälle mit probenunabhängigen Einstellungen brachte eine Reihe von Erkenntnissen, die in die anschließenden Tests der von der Probe abhängigen Parameter eingeflossen sind. Die Erkenntnisse sind in der folgenden Übersicht zusammengefaßt.

- **Toleranz der Optimierung**:

1 Dieser Toleranzwert ist zu ungenau für das kleinste Intervall beim Goldenen Schnitt zur Optimierung auf einer Achse. Das spiegelt sich in den Testergebnissen (Testfälle 1 bis 3) wider. Nur im Ausnahmefall (Testfall 4; mit Maximalanzahl von Durchläufen und Intensitätsmessungen) sind gute Ergebnisse erzielbar.
Der Justagevorgang wird verkürzt, aber die Wahrscheinlichkeit für schlechte Ergebnisse ist sehr hoch.

0.5 Mit diesem Toleranzwert und 5 Durchläufen sind die Ergebnisse durchweg sehr gut. Die Justage läuft relativ schnell.

0.1 Diese Toleranzgrenze stellt einen sicheren Wert dar, bei dem die Ergebnisse der automatischen Justage auch für unterschiedliche Durchlaufanzahlen gut bis sehr gut sind.
Es muß aber eine erhöhte Justagedauer in Kauf genommen werden. Die Justagezeiten sind bei dieser Einstellung jedoch noch vertretbar (ca. 20% mehr als bei einer Toleranz von 0.5).

- **Anzahl der Durchläufe**:

3 In den vorliegenden Tests wurde das Justagemaximum meist am Ende der Justage erreicht. Das Ergebnis war aber schlechter im Vergleich zu Justagen mit mehr Durchläufen. Das zeigt, daß nach drei Durchläufen das Optimium nicht zuverlässig gefunden wurde.
Durch die geringe Anzahl von Algorithmusdurchläufen benötigt die Justage nur die Hälfte der Zeit einer Optimierung in 7 Durchläufen.

5 Mit dieser Einstellung sind die besten Ergebnisse erzielt worden. Sie stellt einen guten Kompromiß im Hinblick auf Justageergebnis und -dauer dar.

7 Die Justageergebnisse sind bei dieser Einstellung nicht besser als bei der mit 5 Durchläufen. Die Justage dauert aber ca. 40% länger.

- **Anzahl der Intensitätsmessungen**:

1 Bei einer Messung können „Ausreißer" die Justage negativ beeinflussen. Damit ist diese Einstellung nicht sicher.
Die Dauer des gesamten Justageprozesses wird aber drastisch reduziert (um ca. 40% gegenüber 3 Messungen).

2 Der Mittelwert der Messungen genügt beim vorliegenden Test, um geringe Intensitätsschwankungen auszugleichen. Im Fall von großen Intensitätsschwankungen würde die Justage aber auch beeinträchtigt werden.
Die Justagedauer wird um ca. 20% verlängert (gegenüber 1 Messung).

3 Durch die Bestimmung des Medians können „Ausreißer" sehr gut ausgeschlossen und somit die automatische Justage optimal durchgeführt werden.
Dies stellt die sicherste Einstellung dar, die auch vom zeitlichen Mehraufwand vertretbar ist.

> 3 Die Dauer der Justage erhöht sich unverhältnismäßig, ohne einen erkennbaren Zugewinn an Sicherheit im Rahmen der verwendeten Meßapparatur zu erreichen.

Die folgenden Ausschnitte aus unterschiedlichen Protokolldateien zeigen den Einfluß der Medianfunktion bei einer unterschiedlichen Anzahl von Messungen auf den tatsächlich an den Justagealgorithmus übergebenen Intensitätsmeßwert. Es ist gut zu erkennen, daß bei drei und fünf Messungen ein gemessener Intensitätswert den Median bildet, wohingegen bei zwei Messungen ein berechneter Mittelwert zurückgeliefert wird.

Protokolldateiauszüge:
(M - Messung, I - Intensität, letzter Wert - Rückgabeintensität)

Testfall 3 (1 Messung, kein Einfluß):
`M: 1 I:`*`37268.76`* `Median:37268.76` *`I: 37269`*

Testfall 2 (2 Messungen, Mittelwert):
`M: 1 I:27640.00 Median:27640.00`
`M: 2 I:28446.14 Median:28043.07` *`I: 28043`*

Testfall 1 (3 Messungen, Median):
`M: 1 I:22401.73 Median:22401.73`
`M: 2 I:`*`22854.48`*` Median:22628.10`
`M: 3 I:23552.96 Median:22854.48` *`I: 22854`*

Testfall 4 (5 Messungen, Median):
`M: 1 I:`*`34941.40`*` Median:34941.40`
`M: 2 I:35585.13 Median:35263.27`
`M: 3 I:34543.12 Median:34941.40`
`M: 4 I:35151.64 Median:35046.52`
`M: 5 I:34896.49 Median:34941.40` *`I: 34941`*

- **Maximale Intensitätsdifferenz**:

  Bei der Auswertung der Protokolldateien stellte sich heraus, daß ein Abbruch in den meisten Fällen nicht nötig war. Für den Testfall 8 hätte sich beispielsweise mit aktivierter Intensitätsdifferenz nicht so ein ausgezeichnetes Ergebnis erzielen lassen. Es wäre für die Einstellung der betrachteten Probe nicht sinnvoll gewesen, das Kriterium zu aktivieren.

Es kristallisierten sich folgende günstige Werte für die getesteten Einstellungen heraus. Dabei wurden solche Parameterwerte vorgezogen, die auch sicher eine gute bis sehr gute Probenjustage liefern.

- Toleranz der Optimierung: **0.1**,
- Anzahl der Durchläufe: **5**,
- Anzahl der Intensitätsmessungen: **3**,
- Maximale Intensitätsdifferenz: **aus**.

Diese Werte werden aufgrund der Testläufe als Voreinstellungen für die probenunabhängigen Parameter des Dialoges festgelegt. Dadurch ist sichergestellt, daß die automatische Justage ohne Nutzereingriff mit diesen Voreinstellungen gute bis sehr gute Ergebnisse liefert. Das Abbruchkriterium „Max. Intensitätsdifferenz" wird in den folgenden Testläufen weiter untersucht, um

danach eine endgültige Aussage zur Voreinstellung treffen zu können. Ist das Kriterium ausgestellt, dauert die Justage im ungünstigen Fall nur länger, das Ergebnis wird dadurch nicht beeinträchtigt.

8.4.3 Test der probenabhängigen Einstellungen

Beim Test der probenabhängigen Einstellungen wurden mehrere unterschiedliche Arten von Proben getestet. Dabei musste jede Probe eine Testreihe durchlaufen, bei der der Suchbereich für das Intensitätsmaximum variiert wurde. Ziel dieses Testverfahrens war es, einen Suchbereich zu ermitteln, der für den Großteil der verschiedenen Proben am besten geeignet ist. Dementsprechend sollte dieser Suchbereich auch für die hier getesteten Proben gute Ergebnisse liefern.

Die übrigen Einstellungen, die im Dialog der „Automatischen Justage" verlangt werden, wurden auf die beim Test der probenunabhängigen Parameter (Abschnitt 8.4.2) ermittelten Werte eingestellt. Der Parameter „Max. Intentsitätsdifferenz" blieb auch bei diesen Tests ausgeschaltet. Anhand der Protokolldateien der jeweiligen Testreihen wurde aber untersucht, wie sich dieser Parameter im aktivierten Zustand auf die Tests und deren Ergebnisse ausgewirkt hätte.

Folgende feste probenunabhängige Einstellungen wurden benutzt:

- Toleranz der Optimierung: **0.1**,
- Anzahl der Durchläufe: **5**,
- Anzahl der Intensitätsmessungen: **3**,
- Maximale Intensitätsdifferenz: **aus**.

Bei der Variation des Suchbereiches stellten sich folgende Testfälle als sinnvoll heraus:

Testfall Nr.		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Suchbereich	DF[2]:	20	20	50	50	50	50	50	50
	TL[3]:	20	20	20	20	20	20	50	50
	CC[4]:	50	100	50	100	150	200	50	150

[2]Einheit für „Beugung fein": Winkelsekunden
[3]Einheit für Tilt: Winkelminuten
[4]Einheit für Kollimator: Mikrometer

Jede Probe wurde nun mit einer Testreihe von 8 Testfällen untersucht. Dabei blieb der Ausgangspunkt, an dem der Justagevorgang gestartet wurde, immer gleich. Da es während dieser Testreihen auch einige Abstürze gab, die - das soll hier nocheinmal betont werden - nicht von der Funktion „Automatische Justage" hervorgerufen wurden, mußten die Ausgangspunkte in diesen Fällen anhand der Anfangsintensität wieder neu eingestellt werden. Notwendig wurde dies, weil die Positionen nach einem Absturz nicht mehr stimmten und die Antriebe einem Referenzpunktlauf unterzogen werden mußten. In den Testdaten ist das daran zu erkennen, daß die Anfangswerte der Justage unterschiedlich zu denen sind, die in den vorangegangenen Testfällen aufgeführt waren.

Da es sich um eine große Anzahl von Testfällen handelt, wurden die genauen Testdaten in den Anhang B der vorliegenden Arbeit verschoben. In diesem Abschnitt sollen nur die genauen Ergebnisse, die von Interesse sind, dargestellt werden. Die wichtigsten Kriterien sollen hier sein:

- Intensität,
- Halbwertsbreite (HWB),
- Ausleuchtung der Probe (Test mit einem 2-dimensionalen Detektor mit Monitor).

Zusätzlich zu den hier ausgewerteten Kriterien, wurde die Dauer des jeweiligen Justagevorganges angegeben. Anhand der Justagezeiten kann eine generelle Aussage darüber getroffen werden, wie lange ein durchschnittlicher Suchprozeß dauert.

Für jede Probe werden die Ergebnisse, die bei den jeweiligen Testfällen ermittelt wurden, in einer Tabelle dargestellt.

Testreihe mit Probe: $CuAsSe_2$ auf GaAs-Schicht								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	44300	47000	45700	46500	47000	46000	48000	43000
HWB:	k.A.	k.A.	k.A.	k.A.	k.A.	k.A.	k.A.	k.A.
Ausleucht.:	90%	100%	90%	100%	100%	100%	100%	90%
Dauer(min):	14:50	15:33	13:49	15:01	16:32	17:15	15:23	18:35
		↑		↑	↑	↑	↑	

Die Probe $CuAsSe_2$ auf GaAs-Schicht ist auf Grund ihrer Form eine besondere Probe. Für eine vernünftige Voreinstellung mußte der Kollimator in der Grobjustage um ca. 700 μm verstellt werden. Ansonsten wäre der Suchalgorithmus der „Automatischen Justage" an dieser Probe gescheitert, da über so große Entfernungen der Kollimator nicht optimiert werden kann.

Aufgrund der komplizierten Probengeometrie war es auch nicht möglich, in den Tests mit dieser Probe die Qualität der Einstellung mit Hilfe der Halbwertsbreite zu bestimmen. Da aber als zweite Möglichkeit zum Einschätzen der Qualität der Einstellung ein 2-dimensionaler Detektor zu Verfügung stand, konnte eine Bewertung der Testergebnisse anhand der Ausleuchtung der Probe durchgeführt werden.

Bei 5 Testfällen erreichte die Justage eine volle Ausleuchtung der Probe, das heißt, die Probe war in diesen Fällen für einen Topographievorgang fertig eingestellt. Die Justage dauerte bei diesen Tests im Schnitt ca. 15 Minuten.

Bei der Betrachtung der Log-Dateien fiel auf, daß eine aktivierte Intensitätsdifferenz bei dieser Probe nicht von Nutzen gewesen wäre. Denn mehrmals wurde das Intensitätsmaximum erst nach einem Einbruch und anschließendem Anstieg der Intensitätswerte gefunden.

Testreihe mit Probe: SiGe-Schicht auf Silizium PF916324/8/8								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	43000	49000	42000	44000	55000	48000	41000	51000
HWB:	12.29	10.49	11.8	10.9	9.29	10.4	12.84	9.49
Ausleucht.:	90%	100%	90%	100%	100%	100%	90%	100%
Dauer(min):	14:24	15:57	16:06	16:47	17:48	17:52	18:13	19:07
		↑		↑	↑	↑		

Diese Probe lieferte fast durchgängig gute Ergebnisse mit allen Testfällen. Allerdings mußte für eine komplette Ausleuchtung der Kollimator über einen größeren Bereich verstellt werden. Dies machte sich bei den Testfällen bemerkbar, die im Suchbereich für den Kollimator 100 oder mehr Mikrometer zu Verfügung stellten. Eine Halbwertsbreite von ca. 10 Winkelsekunden ist ein guter Wert.

Testreihe mit Probe: PF916324/04/20								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	53000	53000	52000	62000	61000	64000	52000	51000
HWB:	10.6	10.57	11.1	8.07	8.35	7.85	11.21	11.24
Ausleucht.:	90%	90%	80%	100%	100%	100%	80%	80%
Dauer(min):	15:18	14:17	17:09	17:48	15:24	18:52	15:23	18:29
				↑	↑	↑		

Bei dieser Probe führten nur die Testfälle zum Ziel, die im Suchbereich eine große Kollimatorverstellung zuließen. Die Testfälle 4, 5 und 6 erreichten dabei eine sehr gute Halbwertsbreite für diese Probe. In dieser Testreihe wäre eine Aktivierung der „Max. Intensitätsdifferenz" sinnvoll gewesen. Dadurch hätte sich die Dauer der Justagen von ca. 15 Minuten auf ca. 10 Minuten reduziert.

Testreihe mit Probe: PF916324/8/7								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	56000	57000	57000	56000	53000	58000	52000	51000
HWB:	8.9	8.89	9.0	9.1	9.35	8.54	9.45	9.7
Ausleucht.:	100%	100%	100%	100%	90%	100%	90%	90%
Dauer(min):	13:47	15:03	13:27	14:39	15:32	18:27	14:15	18:45
	↑	↑	↑	↑		↑		

Da diese Testreihe genau auf die mit der Probe PF916324/8/8 folgte, war der Kollimator für diese Probe schon gut eingestellt. Deshalb war auch ein großer Suchbereich für den Kollimator nicht notwendig. Die Ergebnisse waren mit allen Testfällen zufriedenstellend.

An dieser Testreihe erkennt man sehr gut, daß ein großer Suchbereich für Tilt nicht günstig ist, denn in beiden Testfällen in denen der Bereich von Tilt auf 50 Sekunden gestellt wurde, waren die Ergebnisse am schlechtesten.

Bezüglich des Parameters „Max. Intensitätsdifferenz" kann gesagt werden, daß es sich bei dieser Testreihe die Waage hielt. Die Hälfte der Testfälle hätte eine kürzere Dauer bei der Justage mit Aktivierung des Parameters erreicht.

Testreihe mit Probe: PF916324/08/5								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	43000	41300	43000	43000	43000	42000	41000	40500
HWB:	9.63	9.6	9.6	9.6	9.7	9.8	10.24	10.35
Ausleucht.:	100%	100%	100%	100%	100%	90%	90%	90%
Dauer(min):	14:20	15:55	14:53	14:57	14:36	14:36	16:16	15:09
	↑	↑	↑	↑	↑			

Die Ergebnisse dieser Testreihe sind alle gut. Die Halbwertsbreiten sind bei fast allen Testfällen identisch. Eine Ausnahme bilden die Testfälle 7 und 8. Wiederum wird die Vermutung bestätigt, daß große Suchbereiche für Tilt sich negativ auf die Justage auswirken.

Eine Aktivierung der „Max. Intensitätsdifferenz" hätte sich nicht als günstig herausgestellt. In den meisten Fällen fielen die Intensitätswerte ab und näherten sich erst später dem Intensitätsmaximum an.

Testreihe mit Kollimatorprobe								
Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intensität:	10500	15300	34000	31500	30000	33000	35000	32500
HWB:	8.52	8.15	6.78	7.24	7.5	6.92	6.51	7.36
Ausleucht.:	20%	30%	40%	40%	40%	40%	40%	40%
Dauer(min):	13:01	16:36	15:12	16:50	15:31	16:05	14:31	15:37

Mit dieser Probe verlief die Justage in der Testreihe durchgehend schlecht.

Die Ausleuchtung erreichte maximal 30 bis 40%. Auch für die Halbwertsbreite wurden bessere Werte erwartet. Diese müßte sich in einem Bereich zwischen 3-4 Winkelsekunden befinden.

Höchstwahrscheinlich hätte der Kollimator vorher besser positioniert werden sollen. Die Vermutung liegt nah, daß für eine optimale Justage der Kollimator eine Verstellung um mehr als $200\mu m$ benötigt hätte.

Leider kam es auch bei dieser Testreihe zu einem Absturz des Programmes. Auffällig war auch eine Fehlfunktion des Detektors, hervorgerufen durch eine nicht korrekte Erdung des Gerätes. Ob nun diese negativen Einflüsse schuld an der schlechten Justage waren, kann nicht genau gesagt werden. Es kann auch sein, daß das Problem in der Größe der Probe lag, da die anderen Proben viel kleiner waren und damit vollständig vom Detektor erfaßt wurden.

8.4.4 Bewertung der Tests mit probenabhängigen Einstellungen

Zur Auswertung der Tests mit den probenabhängigen Einstellungen sollen die Ergebnisse zunächst in einer Tabelle zusammengefaßt werden. Diese Tabelle soll einen Überblick geben, wie gut sich die gewählten Suchbereiche in den einzelnen Testreihen geschlagen haben.

Wichtigstes Kriterium war dabei die Ausleuchtung der Probe. Eine mit dem Programm ermittelte gute Halbwertsbreite bedeutete nicht zwangsläufig eine vollständige Ausleuchtung der Probe. Außerdem konnte in der ersten Testreihe keine Halbwertsbreite aufgrund der komplizierten Probengeometrie ermittelt werden. Deshalb soll die Ausleuchtung als wichtigstes Kriterium gelten.

Die folgende Tabelle zeigt, in wieviel Testreihen der jeweilige Testfall zu einer 100%igen Ausleuchtung führte. Damit soll ermittelt werden, welche Suchbereiche mit hoher Wahrscheinlichkeit zum Erfolg führen.

Testfall	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Anzahl mit 100%iger Ausleuchtung (max. 6 möglich)	2	4	2	5	4	4	1	1

Die Kollimatorprobe funktionierte mit keinem Testfall. Sie soll bei der Bestimmung des besten Suchbereiches nicht betrachtet werden.

Anhand der Tabelle kann man gut erkennen, daß der Suchbereich (DF:50, TL:20, CC:100), der den Testfall 4 bildete, am besten in den Testreihen abgeschnitten hat.

Die Testfälle 2, 5 und 6 erwiesen sich ebenfalls als gute Einstellungen, obwohl sie jeweils einmal scheiterten, wenn man die Testreihe mit der Kollimatorprobe bei dieser Bewertung unberücksichtigt läßt.

Ebenfalls ist gut aus der Tabelle abzulesen, daß die Testfälle 1, 3, 7 und 8 nicht ideal waren. In 1 und 3 lag das Problem in einem zu klein gewählten Suchbereich für den Kollimator. In den Fällen 7 und 8 ist vermutlich der zu hohe Bereich für Tilt für die schlechten Ergebnisse verantwortlich. Außerdem war der Suchbereich für den Kollimator im Testfall 7 zu klein.

Schlußfolgernd kann gesagt werden, daß der Suchbereich für den Kollimator mindestens 100μm betragen sollte. Am besten würde sich ein Bereich zwischen 100 und 200μm eignen. Allerdings sollte die korrekte Einstellung des Kollimators nicht mehr als 200μm von der Ausgangsposition der Justage liegen. Bestes Beispiel dafür ist die Testreihe mit der Kollimatorprobe.

Für den Suchbereich von Tilt bleibt festzuhalten, daß dort der Wert nicht über 20 Winkelsekunden gehen sollte.

Der Suchbereich für eine Justage sollte also in den folgenden Wertebereichen liegen:

- DF: 20 - 50 Winkelsekunden
- TL: ca. 20 Winkelminuten
- CC: 100 - 200μm

Bei der Betrachtung der „Max. Intensitätsdifferenz" liegt der Schluß nahe, daß diese Einstellung mit Vorsicht zu benutzen ist. In einigen Fällen wurde trotz eines Abfalls der Intensitätswerte anschließend noch ein Intensitätsmaximum gefunden.

Ursprünglich war diese Einstellung dafür gedacht, als Abbruchkriterium in den Algorithmus einzugreifen. Die Tests haben aber gezeigt, daß eine Aktivierung dieser Option negative Folgen haben kann.

In den Fällen, in denen ein starker Intensitätsabfall während des Suchvorganges auftrat und die Werte auf einem niedrigen Intensitätsniveau blieben, hätte die Aktivierung der „Max. Intensitätsdifferenz" als Abbruchkriterium Sinn gemacht und damit für eine Verkürzung der Justagedauer gesorgt. Allerdings können solche Fälle nicht vorhergesagt werden.

8.5 Zusammenfassung des Tests

Beim Test der „Automatischen Justage" wurden zwei wichtige Punkte getestet. Zum einen war dies der Test der Benutzerschnittstelle der Softwareerweiterung und zum anderen der Test der Funktionalität der Softwareerweiterung - die automatisierte Justage einer grob vorjustierten Probe.

Der Test der Dialogschnittstelle der neu implementierten Funktion zeigte, daß Eingabefehler bei der Parameterfestlegung sicher abgefangen werden. Bereichsüberschreitungen werden sinnvoll verarbeitet, so daß im Fall von fehlerhafter Bedienung eine Ausführung der automatischen Justage gewährleistet bleibt und das Hauptprogramm nicht beeinträchtigt wird. Durch den Test offengelegte Probleme der Dialogsteuerung sind sofort analysiert und gelöst worden. Der Dialog kann sicher ausgeführt werden und reagiert fehlertolerant und robust auf die Eingaben des Anwenders.

Beim Testen des Algorithmus der „Automatischen Justage" kam es darauf an, den folgenden Fragen nachzugehen: Welche Einstellungen sind für eine Justage günstig? und Welche Erfolgsquote erreicht die „Automatische Justage"?

Aufgrund der Testergebnisse wurden die folgenden Voreinstellungen für die Dialogparameter als Kompromiß zwischen Justagedauer und -sicherheit gesetzt:

- Toleranz: 0.1
- Durchläufe: 5
- Anzahl der Messungen: 3
- Max. Intensitätsdifferenz: deaktiviert
- Suchbereich:
  - DF: 50
  - TL: 20
  - CC: 100

Bei 5 von 6 Proben konnte die „Automatische Justage" mit diesen Einstellungen die Probe so justieren, daß sie für einen Topographievorgang korrekt eingestellt war.

Problematisch ist für den Justagealgorithmus, wenn der Suchbereich zu groß gewählt wird. Dies gilt insbesondere für den Kollimator.

Kapitel 9

Nutzerdokumentation

9.1 Einführung

Die Steuerprogrammfunktion „Automatische Justage" hat die Aufgabe, eine Probe optimal im Hinblick auf einen anschließenden Topographievorgang zu justieren. Das Ziel des Justagevorganges ist es, daß die Probe auf dem Probenteller optimal ausgeleuchtet wird. Physikalisch bedeutet dies, daß die Röntgenstrahlung, mit der die Probe bestrahlt wird, von der gesamten Fläche der Probe reflektiert wird. Anders formuliert - es wird ein Intensitätsmaximum der von der Probe reflektierten Röntgenstrahlung gesucht. Erreicht wird die korrekte Einstellung der Probe durch einen Suchalgorithmus, der die Freiheitsgrade Beugung Fein und Tilt des Probetellers sowie den Kollimator so verändert, daß ein Intensitätsmaximum am angeschlossenen Detektor registriert werden kann.

9.2 Voraussetzungen für eine „Automatische Justage"

Bezüglich der Hardware und Software müssen mehrere Vorbedingungen erfüllt sein, um die Funktion „Automatische Justage" nutzen zu können. Grundvoraussetzung ist ein intakter Topographiearbeitsplatz mit einem angeschlossenen Personalcomputer, der die Steuerung übernimmt.

9.2.1 Softwarevoraussetzungen

Auf dem Arbeitsplatzrechner muß das aktuelle RTK-Steuerprogramm unter dem Betriebssytem Windows 3.1 installiert sein. Die für eine korrekte

Funktionsweise benötigten Dateien müssen enthalten sein. Die folgende Liste führt die relevanten Dateien auf:

- `develop.exe`
- `develop.ini`
- `motors.dll`
- `counters.dll`
- `splib.dll`
- `win488.dll`
- `sphelp.hlp`
- `asa.dll`
- `bc450rtl.dll`
- `scs.prg`
- `standard.mak`

9.2.2 Hardwarevoraussetzungen

Für die „Automatische Justage" müssen die für einen Topographiearbeitsplatz typischen Antriebe und Detektoren eingebunden sein. Die Funktion „Automatische Justage" testet beim Start das Vorhandensein folgender Antriebe:

- DF - „Beugung fein"
- TL - „Tilt"
- CC - „Kollimator",

außerdem muß ein 0-dimensionaler Detektor im System eingebunden sein:

- Typ „Radicon".

Für eine Voreinstellung der Probe ist es empfehlenswert, daß auch die Antriebe für die Freiheitsgrade:

- DC - „Beugung grob"
- AR - „Azimutale Rotation"

im System vorhanden sind. Möchte man als Nutzer sicher gehen, daß alle hier aufgelisteten Antriebe dem RTK-Steuerprogramm bekannt sind, kann dies im Dialog *Ausführen/Manuelle Justage...* (Abb. 9.1) überprüft werden. Unter „Aktueller Antrieb” wird durch Anklicken des Pfeils eine Liste der registrierten Antriebe angezeigt.

Möchte man wissen, ob der Detektor korrekt eingestellt ist, öffnet man den Dialog „Zählerkonfiguration” unter dem Menüpunkt *Einstellungen/Detektoren/Detektoren...* (Abb. 9.2). Dort muß in der linken unteren Ecke des Dialogs „Counter” oder „Zähler” stehen. Sollte dort „Test” oder „SCS2” aufgeführt sein, dann muß das Steuerprogramm beendet und im Konfigurationsfile `develop.ini`[1] unter dem Eintrag `[DeviceX]` *Name=Counter* oder *Name=Zähler* eingetragen werden.

9.3 Voreinstellungen für die „Automatische Justage”

Für eine automatische Einstellung der Probe müssen zuvor einige Vorbedingungen erfüllt sein, um später sinnvolle Ergebnisse beim Suchen nach dem Intensitätsmaximum zu erhalten.

9.3.1 Manuelle Justage des Freiheitsgrades „Azimutale Rotation”

Wurde die Probe auf dem Probenteller ordnungsgemäß plaziert, so muß als erstes, bevor die „Automatische Justage” gestartet werden kann, der Freiheitsgrad „Azimutale Rotation” eingestellt werden. Der folgende Abschnitt soll bei der Justage dieses Freiheitsgrades behilflich sein.

Im Dialog *Ausführen/Manuelle Justage...* (Abb. 9.1) wählt man den Antrieb „Tilt” aus. Dieser Antrieb muß an der Position 0 stehen. Wenn das nicht der Fall ist, so kann dieser durch direkte Eingabe eines neuen Winkels an die Stelle 0 bewegt werden.

Als nächstes wählt man den Antrieb „Azimutale Rotation” (AR) aus. Im Dialog aktiviert man für diesen Motor den Fahrbetrieb[2]. Ist dies geschehen, bewegt man den Antrieb mit höchster Geschwindigkeit in eine Richtung,

[1] Hinweis: Das Konfigurationsfile befindet sich in der aktuellen Version des Steuerprogramms im selben Verzeichnis wie das RTK-Steuerprogramm, nicht wie in der früheren Version im Windows-Verzeichnis.

[2] Der Fahrbetrieb bezeichnet den Betriebsmodus, bei dem durch Drücken der Pfeiltasten der entsprechende Motor bewegt wird, wobei im Dialog „Manuelle Justage” die Checkbox „Fahren” ausgewählt sein muß.

wobei nach kurzer Zeit das Zählen des Detektors schneller wird, bis hin zum Rauschen. Ist dies nicht der Fall, muß die andere Richtung abgesucht werden.

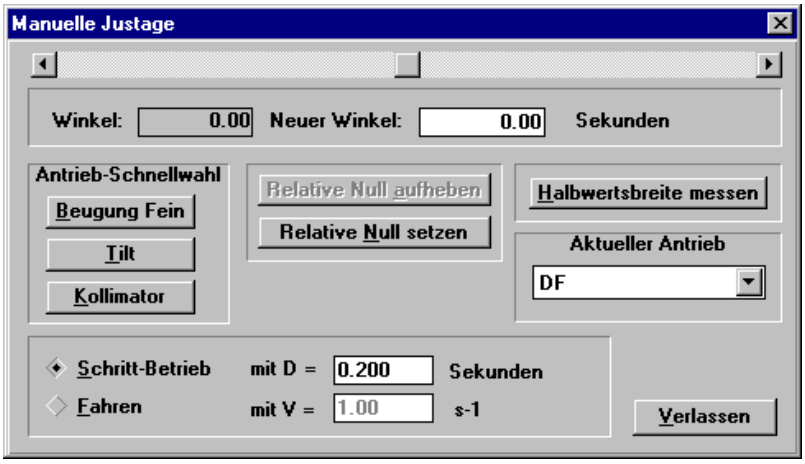


Abbildung 9.1: Dialogfenster „Manuelle Justage"

Wurde ein Peak gefunden, wird der zweite Peak bestimmt. Das heißt, daß der Antrieb „Azimutale Rotation" weiter bewegt werden muß. Findet man in der einen Richtung den zweiten Peak nicht, wird die Richtung geändert. Ist er gefunden, ermittelt man den Mittelpunkt zwischen beiden Peaks, indem die relative Null im Dialog „Manuelle Justage" auf einen Peak gesetzt wird und dann zum zweiten Peak gefahren wird. Dann fährt man mit dem Antrieb „Azimutale Rotation" den halben Abstand zwischen den beiden Peaks an und schließt damit die Justage für den Antrieb „Azimutale Rotation" ab.

Mit dem Motor „Beugung grob" (DC) versucht man anschließend die Intensität zu verbessern, indem man auf dieser Achse in beide Richtungen fährt und dabei auf den höchstmöglichen Intensitätsausschlag stellt. Dabei sollte auf das Geräusch vom Lautsprecher der Zählerkarte geachtet werden, denn die Einstellungen nach Gehör gestalten sich in diesem Stadium der Justage einfacher als die Einstellung nach den Zählerwerten im Zählerfenster.

Damit ist die grobe Voreinstellung abgeschlossen, die für die „Automatische Justage" benötigt wird. Hier sind noch einmal die Schritte in Kurzform zusammengefaßt:

1. Antrieb „Tilt" an Position 0 fahren
2. Mit Antrieb „Azimutale Rotation"(AR) zwei Intensitätspeaks auf dieser Achse suchen
3. Mit Hilfe des Setzens der relativen Null im Dialog „Manuelle Justage" die mittlere Position zwischen den ermittelten Peaks anfahren

4. Mit Antrieb „Beugung grob”(DC) versuchen den Intensitätswert zu maximieren.

9.3.2 Einstellungen für den Detektor

Für den Detektor sollten vor dem Start der „Automatischen Justage” zwei Einstellungen verändert werden. Diese Einstellungen müssen aber nicht zwingend durchgeführt werden, denn auf das Ergebnis des Justagevorganges hat das keine Auswirkung.

Beide Einstellungen werden im Dialog „Zählerkonfiguration” unter dem Menüpunkt *Einstellungen/Detektoren/Detektoren...* (Abb. 9.2) vorgenommen.

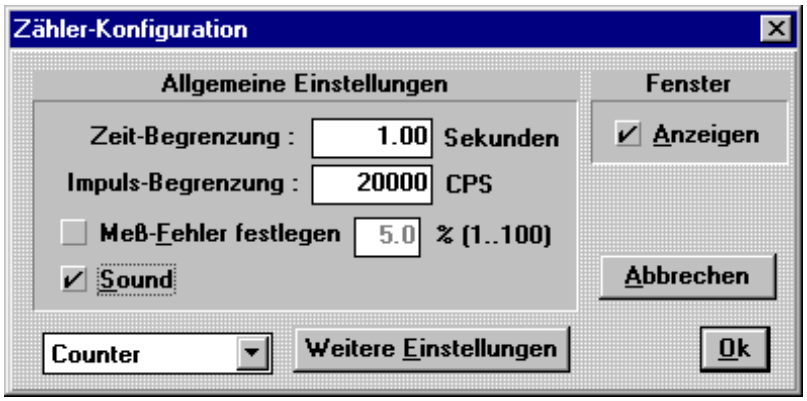


Abbildung 9.2: Dialogfenster „Zählerkonfiguration”

Da der Lautsprecher auf der Zählerkarte, an der der Detektor angeschlossen ist, bei hohen Intensitäten sehr laute Geräusche von sich gibt, wird empfohlen, daß der *Sound* beim Durchführen der „Automatischen Justage” ausgestellt bleibt. Für die Justierung der azimutalen Rotation sollte der Lautsprecher aber eingstellt sein, um besser Intensitätsänderungen wahrnehmen zu können.

Die zweite Einstellung am Detektor betrifft die „Zeit-Begrenzung”. Dort kann minimal ein Wert von 0.6 angegeben werden. Dieser Wert macht Sinn, denn dadurch werden die Werte vom Detektor schneller ausgelesen. Das beschleunigt den Vorgang der „Automatischen Justage”.

9.4 Durchführung der automatischen Justage

Sind die Voraussetzungen und Vorbedingungen gegeben, so kann die „Automatische Justage" gestartet werden. Das Dialogfenster „Automatische Justage" wird über den Menüpunkt *Ausführen/Automatische Justage...* aufgerufen.

9.4.1 Das Dialogfenster und seine Einstellungen

Nach dem Aufruf der „Automatischen Justage" erscheint folgendes Dialogfenster:

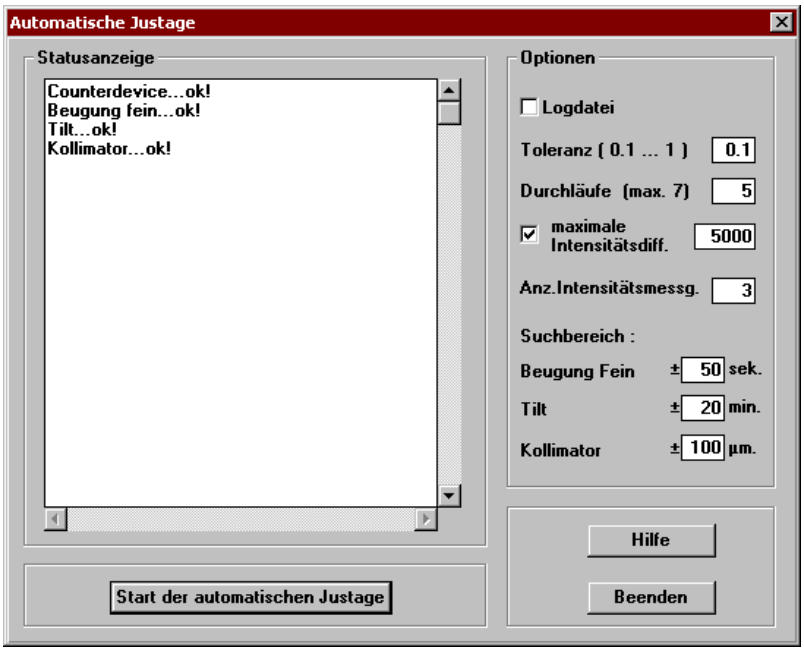


Abbildung 9.3: Dialogfenster „Automatische Justage"

Den größten Teil des Dialoges nimmt das Statusfenster ein. Es dient dazu, dem Nutzer während der Justage Informationen zum Vorgang zu liefern. Beim Start des Dialoges wird überprüft, ob alle benötigten Geräte im System integriert sind. Wenn alles korrekt installiert ist, dann erhält man eine Ausschrift im Statusfenster wie in der Abb. 9.3.

Auf der rechten Seite bekommt der Nutzer eine Anzahl von Optionen zur Auswahl, mit denen er den Justagevorgang beeinflussen bzw. sich nach der Justierung der Probe mehr Informationen übergeben lassen kann. Nachfolgend sollen die möglichen Optionen erläutert werden.

Logdatei

Bei Auswahl (Kästchen wird mit einem Haken markiert) werden alle wichtigen Informationen der „Automatischen Justage" in die Logdatei `justage.log` geschrieben. Das heißt, mit Hilfe der Logdatei kann der Nutzer jeden Meßschritt des Justagevorganges nachvollziehen. Die Datei befindet sich im selben Verzeichnis wie das RTK-Steuerprogramm. Sollte die Datei noch nicht existieren, dann wird die Datei neu erstellt. Ansonsten wird die Logdatei mit jeder Neuausführung der „Automatischen Justage" sukzessive verlängert, wobei in der Logdatei jeder Justagevorgang mit Datum und Uhrzeit protokolliert wird.

Toleranz

Um das Maximum auf den einzelnen Koordinatenachsen zu bestimmen, verwendet der Algorithmus das Verfahren des Goldenen Schnitts. Die Toleranz gibt dabei an, wie klein das letzte Intervall sein soll, indem sich das Maximum befindet. Je kleiner die Toleranz angegeben wird, desto mehr Suchschritte sind nötig. Für einen sinnvollen Wert wurde ein Wertebereich zwischen 0.1 und 1 vorgegeben. Die Entwickler der „Automatischen Justage" empfehlen dem Nutzer den voreingestellten Wert von 0.1 zu benutzen.

Durchläufe

Mit „Durchläufen" ist gemeint, wie oft der Algorithmus wiederholt werden soll. Da es keinen Nachweis gibt, der beweisen kann, daß es sich bei einem Punkt um den Peak handelt, müssen andere Abbruchverfahren die Suche beenden. Eine Möglichkeit ist die, durch das Programm eine Anzahl von Algorithmusdurchläufen vorzugegeben und anzunehmen, daß der Algorithmus nach einer bestimmten Anzahl von Durchläufen den Peak erreicht hat, bzw. sich unmittelbar in seiner Umgebung befindet. Ist der letzte gemessene Intensitätswert nicht der höchste Wert, der während des Justagevorganges erfaßt wurde, so fährt das Programm automatisch den Punkt an, an dem der höchste Intensitätswert festgestellt wurde. Als empfohlener Wert wird die voreingestellte Anzahl von 5 Durchläufen angegeben.

Maximale Intensitätsdifferenz

Die maximale Intensitätsdifferenz gibt an, um wieviel kleiner die Intensität im Vergleich zum vorherigen Durchlauf des Algorithmus sein darf. Wird diese Intensitätsschranke unterschritten, dann wird zum letzten maximalen Intensitätspunkt aus dem vorherigen Durchlauf zurückgekehrt und die automatische Justage beendet. Die Intensitätsdifferenz soll als weiteres Abbruch-

kriterium dienen. Darüber hinaus kann es passieren, dass der Algorithmus kleine „Ausreißer" hat, die durch Meßfehler verursacht werden. Diese „Ausreißer" sollen durch die Prüfung der Intensitätsdifferenz verhindert werden.

Anzahl Intensitätsmessungen

Die Anzahl der Intensitätsmessungen gibt an, wieviel Messungen an einer Position vorgenommen werden. Da die Meßwerte oftmals sehr schwanken können, kann es passieren, daß bestimmte Werte nicht die realen Intensitäten widerspiegeln. Dadurch können falsche Entscheidungen vom Suchalgorithmus getroffen werden. Um diese „Ausreißer" beim Messen auszugleichen, gibt es die Möglichkeit, an einer Position mehrere Messungen durchzuführen. Aus diesen Werten wird mit dem Median-Verfahren ein Wert ermittelt, der möglichst genau den realen Intensitätswert angibt.

Suchbereich

Der Suchbereich gibt an, in welchen Intervallgrenzen auf den einzelnen Achsen das Intensitätsmaximum gesucht werden soll. Ausgangspunkt ist die aktuelle Position der Antriebe. Das Suchintervall auf den einzelnen Achsen ergibt sich aus *akt. Position - gegebener Wert* und *akt. Position + gegebener Wert*. Das Programm testet, ob die Suchintervalle auf den einzelnen Achsen der Antriebe die vorgegebenen Softwareschranken nicht überschreiten. Angenommen die Schranken werden durch die Intervallangabe überschritten, so stellt das RTK-Steuerprogramm die Grenzen kleiner ein. Dadurch können die Motoren die Softwareschranken nicht überfahren. Die vorgegebenen Werte für den Suchbereich sind durch mehrere Tests an unterschiedlichen Proben getestet worden und haben sich als günstige Einstellung für den Justagevorgang herauskristallisiert. Vorsicht sollte bei der Wahl des Intervalls für den Kollimators geboten sein, denn wenn das Suchintervall für diese Achse zu groß oder zu klein gewählt wird, dann kann es passieren, daß die „Automatische Justage" nicht das erhoffte Ergebnis liefert. Ein Wert von 150 scheint gut geeignet zu sein.

Im Dialogfenster „Automatische Justage" besteht außerdem die Möglichkeit, durch Anklicken des „Hilfe"-Knopfes die Online-Hilfe aufzurufen. Die Online-Hilfe beinhaltet die hier beschriebenen Anleitungen für den Nutzer.

Durch Betätigen des „Beenden"-Knopfes kann der Nutzer den Dialog verlassen und gelangt wieder ins Hauptfenster des RTK-Steuerprogramms.

9.4.2 Start des Justagevorganges

Wenn alle optionalen Werte vom Nutzer nach Wunsch eingestellt sind, kann der Suchvorgang nach dem Intensitätsmaximum durch Betätigen des Knopfes „Start der automatischen Justage" gestartet werden.

Während der Justage werden im Status-Fenster Informationen zum Suchvorgang ausgegeben. Nach jeder Koordinatensystemdrehung erfolgt eine Ausgabe der Positionen der Antriebe. Außerdem wird das in diesem Suchschritt gefundene Intensitätsmaximum angezeigt. Zwei Koordinatensystemdrehungen gelten als ein Durchlauf des Algorithmus.

Während des Justagevorganges ist der Dialogknopf „Start der automatischen Justage" ausgegraut. Die Justage kann jetzt nicht unterbrochen werden. Deshalb sollte darauf geachtet werden, daß der Vorgang nicht zu voreilig gestartet wird und die Parameter korrekt eingestellt sind. In der späteren 32-Bit Version des Programmes wird es dann aber möglich sein, die Justage zu unterbrechen.

Die automatische Justage der Probe benötigt ca. 15 Minuten Arbeitszeit.

9.4.3 Ende des Justagevorganges

Wenn der Justageprozeß beendet ist, wird die Position ausgegeben, an der das höchste vom Suchalgorithmus ermittelte Intensitätsmaximum gefunden wurde. Außerdem wird die gemessene Intensität und die Suchzeit ausgegeben. Die Probe sollte nun vollständig ausgeleuchtet und für einen Topograhpie-Meßvorgang eingestellt sein. Um zu kontrollieren, wie gut die Probe justiert ist, kann die Halbwertsbreite im Dialog *Ausführen/Manuelle Justage...* gemessen werden. Dazu muß man den Dialog „Automatische Justage" verlassen.

Wenn eine Logdatei angefertigt wurde, kann diese mit einem Texteditor betrachtet werden.

Kapitel 10

Zusammenfassung und weiterführende Probleme

Die Aufgabe im Rahmen dieser Diplomarbeit bestand darin, ein vorhandenes Steuerprogramm um Funktionalität zu erweitern. Die bestehende Software wird zur Steuerung einer physikalischen Meßapparatur eingesetzt, die der Untersuchung von Halbleiterschichtsystemen mittels Röntgenstrahlung dient.

Eine Teilaufgabe der Software besteht darin, Funktionen für die Ermittlung der Topographie eines Halbleiters zur Verfügung zu stellen. Für diese Aufgabe waren bisher lediglich Funktionen zur manuellen Probeneinstellung und zur Steuerung des Topographievorganges vorhanden.

Der Vorgang der Probeneinstellung für eine Röntgentopographie mußte um eine automatische Justagefunktion ergänzt werden. Von der neuen Funktion „Automatische Justage" wurde gefordert, daß eine per manueller Justage grob eingestellte Probe so justiert wird, daß die Probe nach einer Dauer von maximal 30 Minuten voll ausgeleuchtet ist und eine möglichst geringe Halbwertsbreite bei möglichst hoher Intensität der reflektierten Röntgenstrahlung aufweist.

Im Anschluß an eine automatische Justage soll die in der Versuchsapparatur liegende Probe für eine Topographieaufnahme korrekt eingestellt sein.

Die Erweiterung der bestehenden Software um die neue Funktion „Automatische Justage" erfolgte gemäß den Richtlinien des Software Engineering. Dieser Entwicklungsprozeß unterteilte sich in zwei Teilabschnitte.

Zunächst mußte in der ersten Phase, dem Reverse Engineering, die Software analysiert und dokumentiert werden, da zu der bestehenden Software keine verwertbare Dokumentation vorhanden war. Die gewonnenen Erkenntnisse zur verwendeten Hardware (Kapitel 2) und über die vorhandenen Schnittstellen zur Hardwareansteuerung (Anhang A) sowie die Kommentierung der untersuchten Quelltexte bildeten das Fundament für die zweite Pha-

se der Entwicklung, das Forward Engineering.

Der Softwareerweiterungsprozeß durchlief die Phasen Analyse und Definition, Design, Implementation und Test. Im Laufe dieser Phasen entstanden, als Teile dieser Diplomarbeit, mehrere Dokumente, die im Rahmen des Projektes „Software-Sanierung" weitere Verwendung finden. Zu diesen Dokumenten gehören das Pflichtenheft und die Nutzerdokumentation der neu implementierten Funktion „Automatische Justage".

Die Programmfunktion zur automatischen Einstellung von Proben ist in das bestehende Steuerprogramm eingebettet worden und über den gesonderten Punkt „*Automatische Justage...*" im Untermenü des Hauptmenüpunktes *Ausführen* erreichbar. Beim Aufruf der Funktion erscheint ein Dialogfenster, von dem aus der Justagevorgang gestartet werden kann. Der Anwender hat die Möglichkeit, die automatische Justage durch die Einstellung von Abbruchkriterien und Suchbereichen zu beeinflussen. Werden diese Parameter vom Nutzer nicht geändert, benutzt der Suchalgorithmus die voreingestellten Werte, die während der Testphase ermittelt wurden. Zusätzlich kann der Nutzer eine Online-Hilfe für die Funktion „Automatische Justage" aufrufen.

10.1 Möglichkeiten und Grenzen der Funktion „Automatische Justage"

Während der Testphase wurden wichtige Erkenntnisse zum Einsatz der Programmerweiterung gewonnen. Es konnte ermittelt werden, unter welchen Bedingungen der selbstentwickelte Justagealgorithmus gute bis sehr gute Ergebnisse liefert. Außerdem war es auch möglich, Aussagen darüber zu treffen, in welchen Fällen der verwendete Justagealgorithmus nicht sicher zum Ziel führt. Hier bestehen bereits Ansatzpunkte für Verbesserungen, die aber den Rahmen dieser Diplomarbeit gesprengt hätten und deshalb nicht berücksichtigt werden konnten.

Die neu implementierte Funktion hat die folgenden Eigenschaften:

- Eine grobjustierte Probe kann in 5 von 6 Fällen mit den in der Testphase ermittelten Parametervoreinstellungen für einen Topographievorgang, der unmittelbar darauf folgen kann, gut eingestellt werden.
- Der Vorgang der automatischen Justage dauert im Schnitt 15 bis 20 Minuten.

Es kann also davon ausgegangen werden, daß die Funktion „Automatische Justage" die Anforderungen, die an sie gestellt wurden, erfüllt.

Trotzdem bestehen auch Einschränkungen für den praktischen Einsatz. In der anschließenden Übersicht sind die Grenzen des verwendeten Algorithmus aufgeführt.

- Der Algorithmus ist, nach Auswertung der Tests, nicht in der Lage, Proben korrekt zu justieren, die eine größere als die vom 0-dimensionalen Detektor erfaßbare Fläche einnehmen. So konnte eine Kollimatorprobe mit den Ausmaßen 50mm x 50mm nicht sicher justiert werden.

- Für die Güte der Justage spielt die Stellung der Achse „Kollimator" (CC) eine große Rolle. Der Justagealgorithmus ist nicht dafür ausgelegt, eine Probe zu optimieren, bei der es notwendig ist, die Achse CC um mehr als 200μm in eine Richtung zu verstellen, um einen hohen Ausleuchtungsgrad zu erreichen. Die Konzeption des zur Optimierung eingesetzten Verfahrens des Goldenen Schnitts erlaubt es nicht, auf den Achsen größere Positionsbereiche zu untersuchen, ohne in Kauf zu nehmen, kein Optimum zu finden. Zur Umgehung dieses Problems wird im Ausblick ein Lösungsvorschlag (Kollimatorschätzfunktion) unterbreitet.

10.2 Ausblick

In diesem Abschnitt sollen Ideen zur Verbesserung der Justage mit dem Steuerprogramm vorgestellt werden.

Die im Rahmen der Diplomarbeit entwickelte Softwarelösung der automatischen Justagefunktion ist zur Feinjustage einer Meßprobe vorgesehen. Um den Prozeß der Probeneinstellung komplett zu automatisieren, - das heißt, die Probe muß nur eingelegt werden und auf Knopfdruck werden alle nötigen Schritte zur Justage automatisch vorgenommen - sind noch zwei weitere Schritte zu realisieren. Diese Arbeitsschritte würden sich vor den Prozeß der jetzigen Programmfunktion „Automatische Justage" einordnen.

Der erste Schritt ist die automatische Vorjustierung der Probe bezüglich des Freiheitsgrades „Azimutale Rotation" (AR). Dieser Arbeitsvorgang war vom ursprünglichen Entwickler des Steuerprogramms bereits vorgesehen und sollte mit Hilfe eines Makros realisiert werden. Das dazugehörige Makro-Skript „`AzimutalJustify`" befindet sich in der Makrodatei `standard.mak`, aber die Abarbeitung dieses Skriptes funktioniert noch nicht.

Es bieten sich also zwei Möglichkeiten an, die Justage des Freiheitsgrades „Azimutale Rotation" zu realisieren.

1. Die Kommandos des existierenden Makroprototypen müssen genau analysiert und ausgewertet werden. Möglicherweise lassen sich korrigierbare Fehler im Skript finden, so daß die Möglichkeit zur Ausführung dieses Makros besteht. Im ungünstigsten Fall kann das Makro in der vorliegenden Form nicht genutzt werden und müßte anhand der Arbeitsschritte der manuellen Vorgehensweise neu entworfen werden.

2. Die Justage des Freiheitsgrades „Azimutale Rotation" wird in einem eigenen Programmteil implementiert, wobei der Algorithmus für den Einstellungsvorgang den Schritten ähnelt, die im Kapitel 3 für die manuelle Justage vorgestellt wurden.

Die zweite Lösung ist nach Meinung der Autoren der vorliegenden Arbeit die einfacher und besser zu realisierende, da das Ermitteln der Funktionsweise der Makrobefehle eine größere Einarbeitung erfordert. Außerdem verkompliziert die Makrosteuerung den Ablauf der Justage.

Der zweite Schritt, der ebenfalls für eine vollautomatische Einstellung der Probe realisiert werden müßte, ist die Entwicklung einer Schätzfunktion für den Kollimator. Diese soll Verstellungen der Achse „Kollimator" (CC) über einen größeren Bereich schnell ausführen und dabei die Richtung, in die der Kollimator verstellt werden muß, ermitteln.

Eine Idee für die Realisierung einer solchen Schätzfunktion wäre folgender Ablauf:

1. Den Kollimator in eine Richtung fahren, bis die Intensität steigt.

2. Den Kollimator weiter in die gleiche Richtung fahren, bis die Intensität wieder absinkt.

3. Mit DF nachregeln, bis die Intensität wieder steigt. Die Richtung, in die DF gefahren wurde, merken.

4. Die Schritte 2. bis 3. wiederholen, bis die Richtung beim Nachregeln von DF umschlägt.
   Die Änderung der Bewegungsrichtung deutet darauf hin, daß der Kollimator über das Maximum gefahren.

Die Teilaufgaben „Justage der azimutalen Rotation", „Voreinstellung des Kollimators mit einer Schätzfunktion" und die in dieser Arbeit realisierte Funktion „Automatische Justage" ergeben bei sequentieller Ausführung eine vollautomatische Einstellung der Probe, bei der nur noch das Einlegen der Probe selbst vom Nutzer erledigt werden muß.

Eine weitere Verbesserung der Funktion „Automatische Justage" würde die Möglichkeit darstellen, den Justagevorgang über einen Dialogknopf abbrechen zu können. Da dies aus den genannten Gründen (Kap. 7) nicht in dieser Arbeit realisiert wurde, müßte bei einer Portierung des gesamten Softwareprojekts in die 32-Bit Umgebung die Fähigkeit des Betriebssystems für präemptives Multitasking ausgenutzt werden. Der Suchalgorithmus könnte dann einen Thread darstellen, dessen Abarbeitung durch den Nutzer mittels eines Dialogknopfs beendet wird. Dazu kann der Startknopf nach Betätigung in einen Abbruchknopf umgewandelt werden, so daß bei nochmaliger Betätigung nach einer Sicherheitsabfrage der Abbruch des Justagevorganges erfolgt.

Anhang A

Schnittstellenbeschreibung der Motorenansteuerung

Vom ursprünglichen Entwickler des Programms ist die Hardwareansteuerung, im Speziellen der Zugriff auf die Motoren, von den übrigen Programmteilen gut abgekapselt worden. Der Programmierer hat versucht, alle für die Motoren relevanten Funktionen in einer Dynamic Link Library (DLL)[1], der Bibliotheksdatei `motors.dll`, zusammenzufassen.

Für jede Art der Motorenansteuerung, die grundsätzlich von der verwendeten Hardware-Motorsteuerkarte abhängt, ist im Programm eine spezielle Klasse in C++ implementiert. Um die Motorenhardware auch softwaretechnisch ohne Kenntnisse der Objektorientierung nutzen zu können, hat der Entwickler ein Interface in der Programmiersprache C entworfen. Damit ist es möglich, die Motoren mittels dieser Bibliothek außerhalb des Röntgentopographie-Steuerprogramms anzusteuern.

Dieses C-Interface stellt eine Abstraktionsschicht (High-Level-Layer) der Motoransteuerung dar. Der Benutzer benötigt hierbei keine Informationen über die Art des Hardwarezugriffs, sondern kann die Motoren über die ihnen zugeordnete Achse oder ihre Identifikationsnummer ansprechen und die gewünschte Funktion ausführen lassen.

Im folgenden werden die vom Entwickler bereitgestellten C-Funktionen und ihre Anwendung beschrieben.

[1] Bibliothek von Funktionen, die erst zur Laufzeit dynamisch vom Hauptprogramm geladen wird

Zuerst muß zwischen zwei Arten von Funktionen unterschieden werden:

a) ml*FunctionName* - Diese Funktionen benutzen den Motor mit der übergebenen Motornummer. „ml" steht hierbei für Motorliste, das heißt, es wird auf Funktionen und Membervariablen der Klasse TMList zugegriffen.

b) m*FunctionName* - Diese Funktionen benutzen den zur Zeit aktivierten Motor. „m" steht hierbei für Motor, das bedeutet, daß Funktionen und Membervariablen der von TMotor abgeleiteten Klassen benutzt werden.

A.1 Interface der Motorlisten-Funktionen

Der Großteil der Motorlisten-Funktionen erwartet als Übergabeparameter eine Motor-Identifikationsnummer *mid*, die festlegt, für welchen Antrieb die jeweilige Funktion ausgeführt werden soll.

mlInitializeMotorsDLL

Syntax:	`BOOL WINAPI mlInitializeMotorsDLL( void )`
Beschreibung:	Diese Funktion führt die Initialisierung der Bibliothek `motors.dll` aus. Die Konfigurationsdatei wird nach Einträgen der verschiedenen Motorobjekte (z.B. Objekte der Klassen TMotor, TC812ISA, TC832) durchsucht, die dann erzeugt und der Reihenfolge nach in die Motorenliste (Klasse TMList) aufgenommen werden. Die Membervariablen der Motoren werden mit Werten aus der INI-Datei belegt. Es wird versucht, die Motoren über die Motorsteuerungshardware anzusprechen und zu initialisieren.
Parameter:	-
Rückgabewert:	`TRUE` = Initialisierung erfolgreich `FALSE` = Initialisierung mindestens eines Motors schlug fehl
Siehe auch:	`mlIsServerOK`

mlSetAxis

Syntax:	`BOOL WINAPI mlSetAxis( int mid )`
Beschreibung:	Festlegung der aktiven Achse. Damit wird der Antrieb mit der angegebenen Motor-ID als aktiver Motor eingestellt, indem die TMList-Membervariable *nActiveDrive* mit der übergebenen Motor-Identifikationsnummer *mid* belegt wird. Der Antrieb der so aktivierten Achse kann danach direkt über die m*FunctionName*-Funktionen angesprochen werden.
Parameter:	`int` *mid* - Identifikationsnummer des Motors der speziellen Achse
Rückgabewert:	`TRUE` = Auswahl war erfolgreich `FALSE` = Auswahl schlug fehl, weil die Achse nicht existiert
Siehe auch:	`mlGetAxis`, `mlGetIdByName`

mlGetAxis

Syntax:	`int WINAPI mlGetAxis( void )`
Beschreibung:	Ermittelt die Identifikationsnummer der derzeit aktivierten Achse.
Parameter:	-
Rückgabewert:	Die Membervariable *nActiveDrive* der Klasse TMList, die die Identifikationsnummer des Motors der aktiven Achse speichert, wird zurückgeliefert.
Siehe auch:	`mlSetAxis`, `mlGetIdByName`

mlGetIdByName

Syntax: `int WINAPI mlGetIdByName( TAxisType axis)`

Beschreibung: Gibt die Motornummer (ID) des entsprechenden Achsenmotors zurück.

Parameter: `TAxisType` *axistype* - zulässige Parameter sind X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Monochromator, Absorber, Collimator, DF, DC, Tilt, Rotation. Die Struktur `TAxisType` ist in der Datei `comhead.h` deklariert.

Rückgabewert: ≥ 0 = Identifikationsnummer (ID) des zur übergebenen Achse gehörigen Motors
-1 = Fehler, d.h., die übergebene Achse wurde noch nicht initialisiert

Siehe auch: `mlSetAxis`, `mlGetAxis`, `mGetAxisName`

mlGetDistance

Syntax: `BOOL WINAPI mlGetDistance(int mid,double &position)`

Beschreibung: Ermittelt für den Antrieb mit der angegebenen Motor-ID die aktuelle Stellung im Winkelmaß (Grad etc.). Die Position wird von der Hardware ausgelesen. Falls sich der Motor in Ruhe befindet und kein Übertragungsfehler aufgetreten ist, wird die Motorposition in der Motorklassenvariablen *dAngle* gespeichert.

Parameter: `int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
`double &` *position* - Rückgabeparameter für die ausgelesene Position

Rückgabewert: `TRUE` = Die Position konnte ermittelt werden, wurde in der Motor-Membervariablen *dAngle* abgelegt und wird im `double&` Parameter als Winkel zurückgeliefert.
`FALSE` = Fehlerfall: Motor war in Fahrt oder die Position konnte nicht ausgelesen werden

Siehe auch: `mlGetValue`, `mGetDistance`, `mGetValue`

mlGetValue

Syntax: `double WINAPI mlGetValue(int mid,TValueType vtype)`

Beschreibung: Abhängig vom Wert des Parameters *vtype* wird der dazugehörige Wert der Motorgröße derjenigen Achse zurückgeliefert, die durch *mid* bestimmt wird. Der Typ TValueType (deklariert in `comhead.h`) kann folgende Werte annehmen: *Distance* (Winkelstellung des Motors), *MinDistance*, *MaxDistance*, *Speed*, *Width* (Winkelschrittweite). Die Funktion wertet nur die Membervariablen der Motoren aus. Es findet kein Zugriff auf die Hardware statt.

Parameter: `int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
`TValueType` *vtype* - Parameter gibt an, welcher Motorwert zurückgeliefert werden soll.

Rückgabewert: `double` Wert der durch den `TValueType` Parameter bestimmten Membervariablen des Antriebs, wobei folgende Zuordnung stattfindet: *Distance* = *dAngle*, *MinDistance* = *dAngleMin*, *MaxDistance* = *dAngleMax*, *Speed* = *dSpeed* unter Berücksichtigung von *dwVelocity* bzw. *Width* = *dAngleWidth*

Siehe auch: `mlGetDistance`, `mGetValue`, `mGetDistance`, `mSetValue`

mlMoveToDistance

Syntax: `BOOL WINAPI mlMoveToDistance(int mid,double distance)`

Beschreibung: Startet die Bewegung des Motors, der durch *mid* vorgegeben wird, an die durch *distance* angegebene Motorstellung im Winkelmaß.

Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse `double` *distance* - Winkelposition, an die der Motor bewegt werden soll
Rückgabewert:	`FALSE` = Fehlerfall: `motors.dll` konnte nicht korrekt initialisiert werden, d.h., die Konfigurationsdatei wurde nicht erfolgreich eingelesen. `TRUE` = Da keine Fehlerkontrolle beim Aufruf der Motorlisten-Funktionen vorgenommen wird, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.
Siehe auch:	`mMoveToDistance`, `mMoveByDistance`, `mlGetDistance`

mlIsMoveFinish

Syntax:	`BOOL WINAPI mlIsMoveFinish( int mid )`
Beschreibung:	Stellt fest, ob der Motor mit der angegebenen Motor-ID die anzufahrende Winkelposition erreicht hat.
Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
Rückgabewert:	`TRUE` = Antrieb hat die vorgegebene Stellung angefahren und befindet sich in Ruhe. `FALSE` = Motor ist in Bewegung oder an einer Endlage (Deathband) zum Stehen gekommen.
Siehe auch:	`mIsMoveFinish`, `mlMoveToDistance`

mlGetOffset

Syntax:	`double WINAPI mlGetOffset( int mid )`
Beschreibung:	Gibt das Offset für die Relative Null des angeforderten Antriebs zurück. Der Wert ist in der Motor-Membervariablen *dAngleBias* abgelegt.

Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
Rückgabewert:	`double` Wert der Membervariablen *dAngleBias* des angegebenen Motors. Ist der Wert $\neq 0$, dann wurde die Relative Null gesetzt, und alle Operationen berücksichtigen das Offset.
Siehe auch:	`mSetRelativeZero`, `mIsDistanceRelative`

mlParsingAxis

Syntax:	`TAxisType WINAPI mlParsingAxis( LPSTR axisname )`
Beschreibung:	Umwandlung eines Strings in einen TAxisType (deklariert in `comhead.h`). Dadurch ist es möglich, für eine Achse mehrere Bezeichnungen zu wählen. Z.B. werden die Zeichenketten *AzimutalRotation*, *AZ*, *Rotation* und *Azimute* demselben Achsentyp *Rotation* zugeordnet.
Parameter:	`LPSTR` *axisname* - Zeichenkette des Achsennamens
Rückgabewert:	`TAxisType` = Typ der zugeordneten Motorachse. Folgende Werte sind möglich: X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Absorber, Tilt, Collimator, Rotation, Monochromator, DC, DF.
Siehe auch:	`mGetAxisName`, `mlGetIdByName`, `mlIsAxisValid`

mlIsAxisValid

Syntax:	`BOOL WINAPI mlIsAxisValid( TAxisType axis )`
Beschreibung:	Ermittelt, ob die bestimmte Achse initialisiert wurde, d.h., ein der Motorachse zugeordneter Motor wurde im System gefunden und entsprechend den Einträgen in der Konfigurationsdatei eingerichtet.

Parameter:	`TAxisType` *axis* - Typ der geforderten Achse. Die Enumeration TAxisType ist in `comhead.h` deklariert. Folgende Werte sind möglich: X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Absorber, Tilt, Collimator, Rotation, Monochromator, DC, DF.
Rückgabewert:	`TRUE` = Die angegebene Motorachse ist bereits erfolgreich initialisiert worden. `FALSE` = In der Konfigurationsdatei existiert kein Eintrag für den geforderten Antrieb, oder der Motorachsentyp wurde falsch übergegeben.
Siehe auch:	-

mlIsServerOK

Syntax:	`BOOL WINAPI mlIsServerOK( void )`
Beschreibung:	Stellt fest, ob die `motors.dll` erfolgreich geladen und die Antriebe richtig konfiguriert wurden. Dazu wird der Wert der statischen Variablen *bModulLoaded* zurückgeliefert, der bei erfolgreicher Ausführung der Funktion `mlInitializeMotorsDLL` gesetzt wird.
Parameter:	-
Rückgabewert:	`TRUE` = Die Bibliothek `motors.dll` wurde ohne Probleme eingebunden und alle Antriebe wurden korrekt konfiguriert. `FALSE` = Die Funktion `mlInitializeMotorsDLL` schlug fehl. Der Wert der Variablen *bModulLoaded* wurde nicht auf `TRUE` gesetzt. D.h., die Bibliothek `motors.dll` konnte nicht initialisiert werden.
Siehe auch:	`mlInitializeMotorsDLL`

mlGetAxisNumber

Syntax:	`int WINAPI mlGetAxisNumber( void )`
Beschreibung:	Ermittelt die Anzahl der verfügbaren Antriebsachsen. Dazu wird die Membervariable *nLastDrive* der Klasse TMList ausgewertet.
Parameter:	-
Rückgabewert:	`int` Wert der privaten TMList-Klassenvariablen *nLastDrive* + 1, entspricht Anzahl der Antriebe, die dem Steuerprogramm bekannt sind
Siehe auch:	-

mlSaveModuleSettings

Syntax:	`void WINAPI mlSaveModuleSettings( void )`
Beschreibung:	Speichert für alle im System vorhandenen Antriebe die aktuellen Motorparameter (z.B. Velocity, PositionWidth, AngleMin, AngleWidth, Acceleration, DynamicGain) in der Konfigurationsdatei unter dem zum jeweiligen Motor gehörigen Abschnitt ab. Dazu werden die `SaveSettings`-Methoden der Motorklassen aufgerufen. Außerdem wird für die einzelnen Antriebe der INI-Wert `RestartPossible` auf `TRUE` gesetzt, um ein ordnungsgemäßes Programmabschluß zu signalisieren. Die Motoren werden danach gestoppt.
Parameter:	-
Rückgabewert:	-
Siehe auch:	-

mlSetAngleDefault

Syntax:	`void WINAPI mlSetAngleDefault( void )`
Beschreibung:	Setzt die Voreinstellungen für das Winkeloffset der Relativen Null und die Softwareschranken im Winkelmaß. Dazu werden die Membervariablen *dAngleBias*, *dAngleMin* und *dAngleMax* der einzelnen im System vorhandenen Antriebe auf ihre Defaultwerte zurückgestellt. Für *dAngleMin* und *dAngleMax* werden die Membervariablen *lPositionMin* und *lPositionMax*, die die zulässigen Positionsbereiche in Encoderschritten darstellen, ausgewertet; *dAngleBias* wird auf 0.0 gesetzt.
Parameter:	-
Rückgabewert:	-
Siehe auch:	`mSetAngleDefault`

mlGetVersion

Syntax:	`LPCSTR WINAPI mlGetVersion( void )`
Beschreibung:	Rückgabe der Versionsnummer der `m_layer`-Schicht. Dazu wird der Wert der statischen Variablen *mlVersion* zurückgeliefert.
Parameter:	-
Rückgabewert:	`LPCSTR` Wert der statischen Variablen *mlVersion*, der die Versionsnummer der Motorsteuerungsschicht inklusive Datum der Kompilierung enthält
Siehe auch:	-

mlGetInstance

Syntax:	`HINSTANCE WINAPI mlGetInstance( void )`
Beschreibung:	Gibt das Instanz-Handle des Moduls `motors.dll` zurück.
Parameter:	-
Rückgabewert:	`HINSTANCE` Wert der globalen Variable *hModuleInstance*, die in der `LibMain`-Funktion der `motors.dll` mit dem Instanz-Handle für die DLL belegt wurde, wird zurückgegeben.
Siehe auch:	-

A.1.1 Dialoge zur Motorsteuerung

Vom Entwickler sind auch Dialoge zur Steuerung der Motoren in das Interface aufgenommen worden, um anderen Programmierern die Möglichkeit zu geben, auf komplexe Dialoge ohne großen Aufwand über Aufrufe aus der Dynamischen Bibliothek `motors.dll` zurückgreifen zu können, um beispielsweise innerhalb eines eigenen Programms einen Referenzpunktlauf (→ `mlInquireReferencePointDlg`) durchzuführen oder die Motoren direkt ansteuern zu können (→ `mlPositionControlDlg`).

Diese Herangehensweise ist unvorteilhaft im Hinblick auf die Trennung von Oberfläche, Hauptprogramm und Hardwareansteuerung, ist aber der Intention des Steuerprogrammentwicklers geschuldet.

mlInquireReferencePointDlg

Syntax:	`void WINAPI mlInquireReferencePointDlg( int task)`
Beschreibung:	Startet den „Grundstellung anfahren"-Dialog. Mit dieser Funktion wird ein Dialog aufgerufen, mit dem für alle Motoren des Systems ein Referenzpunktlauf durchgeführt werden kann.

Weiterhin ist es möglich, den Nullpunkt in Relation zum Referenzpunkt neu zu bestimmen und auch den absoluten Nullpunkt zu setzen. Im Steuerprogramm wird diese Funktion über das Hauptmenü (*Einstellungen / Motoren / Grundstellung*) aufgerufen.

Parameter: `int` *task* - Übergabe einer Tasknummer, die festlegt, welche Voreinstellungen der Dialog anzeigen soll. Wird *task* = 99 übergeben, wird veranlaßt, daß alle Antriebe für den Referenzpunktlauf ausgewählt werden.

---

mlOptimizingDlg

Syntax: `void WINAPI mlOptimizingDlg ( void )`

Beschreibung: In den meisten Fällen sind die Steuerparameter nur im Rahmen eines konkreten Versuchsaufbaus richtig bestimmbar. Da den meisten Antrieben zudem auch ein mechanisches Spiel anhaftet, sind die Parameter nur unter Beachtung der realen Anfahrcharakteristik optimal einstellbar. Zum Optimieren wird mit dem Dialog „Manuelle Justage" der zu optimierende Antrieb ausgewählt. Die Funktion `mlOptimizingDlg` startet den „DC-Controller-Parameter"-Dialog für den aktuellen Antrieb. Zuvor wird eine `cm_CallExecuteScan`-Nachricht an die Anwendung gesendet, um das Scan-Fenster zu aktivieren. Der durchgeführte Scan tastet die Motorbewegung 150 Mal ab. Das Ergebnis dieses Scans wird dann im Scanfenster visualisiert. Ziel der Optimierung ist es, daß eine Endposition gleichmäßig und ohne Schwingungen angefahren wird.
Die änderbaren Parameter für Motoren der Steuerkarte C-812 sind die Werte der Membervariablen *dwMaxVelocity* (maximale Geschwindigkeit), *dwAcceleration* (Beschleunigung), *wStaticGain* (statische Verstärkung), *wDynamicGain* (dynamische Verstärkung), *wTorque* (Beschränkung des maximalen Motorstroms), *wPositionWidth* (Schrittweite zum Messen des Anfahrverhaltens).

Bei den Motoren an der Steuerkarte C-832 sind die Bezeichnungen für statische und dynamische Verstärkung anders gewählt: *wKP*, *wKD*. Außerdem läßt sich über *wKI* die Integralverstärkung und durch *wKL* das Integrallimit festlegen.
Der Benutzer hat die Möglichkeit, einen CheckScan durchzuführen, bei dem das Anfahrverhalten des Motors überprüft wird. Dazu ruft das Programm die Interfacefunktion `mStartMoveScan` auf.

Parameter: -

---

mlPositionControlDlg

Syntax: `void WINAPI mlPositionControlDlg ( void )`

Beschreibung: Startet den Dialog zur Positionsansteuerung für die gesamten Antriebe. In diesem Dialog kann der Antrieb in Encoderschritten[2] angesteuert werden. Der Dialog wird im Steuerprogramm über das Hauptmenü (*Einstellungen / Antriebe / Direkte Steuerung*) aufgerufen.

Parameter: -

---

mlSetParametersDlg

Syntax: `void WINAPI mlSetParametersDlg ( void )`

Beschreibung: Startet den „Motor-Parameter"-Dialog.Dieser dient der Einstellung der Softwareschranken und der Schrittweite der einzelnen Antriebe. Die Parameter können jeweils in Encoder- oder Winkeleinheiten verändert werden.
Der Dialog wird im Steuerprogramm über das Haupt-

[2]Ein Encoderschritt ist die kleinste Einheit, mit der die Motorsteuerkarten die Schrittmotoren steuern können. Die Umrechnung von Winkelangaben in diese Einheit wird vom Steuerprogramm in der Motor-Klassenfunktion `TMotor::Translate` realisiert.

menü (*Einstellungen / Antriebe / Parameter*) aufgerufen.

Parameter: -

A.2 Interface der Motor-Funktionen

Im Unterschied zu den Motorlisten-Funktionen beziehen sich die Funktionen des Interface der Motoren auf den aktuell eingestellten Antrieb, der in der Motorlisten-Membervariable *nActiveDrive* gespeichert ist. Der Parameter *mid* entfällt somit als Übergabe der Motor-Identifikationsnummer für die jeweiligen Funktionen.

mMoveToDistance

Syntax: `BOOL WINAPI mMoveToDistance( double distance )`

Beschreibung: Startet die Bewegung des Motors, der als aktueller Antrieb eingestellt ist, an die durch *distance* angegebene Motorstellung im Winkelmaß.

Parameter: `double` *distance* - Winkelposition, an die der Motor bewegt werden soll

Rückgabewert: `FALSE` = Fehlerfall: Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden, d.h., die Konfigurationsdatei wurde nicht erfolgreich eingelesen.
`TRUE` = Da keine Fehlerkontrolle beim Funktionsaufruf stattfindet, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.

Siehe auch: `mMoveByDistance`, `mlMoveToDistance`, `mGetDistance`

mMoveByDistance

Syntax:	`BOOL WINAPI mMoveByDistance( double distance )`
Beschreibung:	Bewegt den aktuellen Antrieb relativ zur aktuellen Motorposition um eine bestimmte Distanz, die im Parameter *distance* im Winkelmaß übergeben wird.
Parameter:	`double` *distance* - Winkel, um den der Motor bewegt werden soll
Rückgabewert:	`FALSE` = Fehlerfall: Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden, d.h., die Konfigurationsdatei wurde nicht erfolgreich eingelesen. `TRUE` = Da keine Fehlerkontrolle bei der Funktionsausführung stattfindet, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.
Siehe auch:	`mMoveToDistance`, `mlMoveToDistance`, `mGetDistance`

mSetLine

Syntax:	`BOOL WINAPI mSetLine( int channel , BOOL state )`
Beschreibung:	Mit dieser Funktion wird zur Zeit der Digital-Port der C-812-Steuerkarte angesprochen.
Parameter:	`int` *channel* - Nummer des Kanals, der als Ausgangsport eingestellt werden soll. Korrekte Werte für *channel* liegen zwischen 1 und 16. `BOOL` *state* - gibt an, ob der Ausgangskanal auf 0 Volt (logisch 0 → *state* = `FALSE`) oder 5 Volt (logisch 1 → *state* = `TRUE`) gesetzt werden soll.
Rückgabewert:	`TRUE` = Erfolgreiche Kommunikation über den Digital-Port der C-812-Steuerkarte

`FALSE` = Fehlerfälle:

a) Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden.

b) Der Parameter *channel* liegt außerhalb des zulässigen Bereichs.

c) Die notwendigen Motorkommandos konnten nicht ohne Probleme an die Motorsteuerkarte übermittelt werden.

Siehe auch: -

mIsMoveFinish

Syntax: `BOOL WINAPI mIsMoveFinish( void )`

Beschreibung: Stellt fest, ob der aktuelle Motor die anzufahrende Winkelposition erreicht hat.

Parameter: -

Rückgabewert: `TRUE` = Antrieb hat die vorgegebene Stellung angefahren und befindet sich in Ruhe.
`FALSE` = Motor ist in Bewegung oder an einer Endlage (Deathband) zum Stehen gekommen.

Siehe auch: `mlIsMoveFinish`, `mMoveToDistance`

mIsRangeHit

Syntax: `BOOL WINAPI mIsRangeHit( void )`

Beschreibung: Diese Funktion stellt fest, ob eine Bereichsbeschränkung überschritten wurde. Dazu wird der Wert der Motor-Membervariablen *bRangeHit* zurückgeliefert.

Diese Variable wird in der Umrechnungsfunktion `BOOL TMotor::Translate(long &pos,double ang)` gesetzt, falls der übergebene Winkelparameter nicht ordnungsgemäß in eine Motorpositionen innerhalb der Softwareschranken umgerechnet werden kann. Außerdem kann *bRangeHit* in der Motorfunktion `BOOL TC_832::IsLimitHit(void)` verändert werden.

Parameter: -

Rückgabewert: `TRUE` = Fehlerfall: Antrieb sollte eine Position anfahren, die außerhalb des zulässigen Bereichs liegt.
`FALSE` = Es kam zu keiner Bereichsüberschreitung.

Siehe auch: -

mIsCalibrated

Syntax: `BOOL WINAPI mIsCalibrated( void )`

Beschreibung: Stellt fest, ob für den aktuellen Motor ein gültiger Referenzpunktlauf durchgeführt wurde. Der Wert der Motor-Membervariable *bCalibrated* wird zu diesem Zweck zurückgegeben.

Parameter: -

Rückgabewert: `TRUE` = Gültiger Referenzpunktlauf für den aktuellen Antrieb ist sichergestellt.
`FALSE` = Für den aktuellen Motor existiert kein gültiger Referenzpunktlauf.

Siehe auch: -

mIsDistanceRelative

Syntax: `BOOL WINAPI mIsDistanceRelative( void )`

Beschreibung: Gibt an, ob sich die Positionsangaben (im Winkelmaß)

	des aktuellen Antriebs auf eine gesetzte Relative Null beziehen oder ob mit absoluten Angaben gearbeitet wird. Es wird dazu geprüft, ob die Motor-Membervariable *dAngleBias* Null ist.
Parameter:	-
Rückgabewert:	`TRUE` = *dAngleBias*(Winkel-Offset) $\neq$ 0, d.h., alle Winkelangaben beziehen sich auf die Relative Null. `FALSE` = Winkel-Offset *dAngleBias* = 0, d.h., alle Winkelangaben sind absolute Werte.
Siehe auch:	`mSetRelativeZero`, `mlGetOffset`, `mSetAngleDefault`

mGetDistance

Syntax:	`BOOL WINAPI mGetDistance( double & position )`
Beschreibung:	Ermittelt für den aktuellen Antrieb die aktuelle Stellung im Winkelmaß (Grad etc.). Die Position wird von der Hardware ausgelesen und wird, falls der Motor sich in Ruhe befindet und kein Übertragungsfehler aufgetreten ist, in der Motorklassenvariablen *dAngle* gespeichert und im `double` Parameter zurückgeliefert.
Parameter:	`double &` *position* - Rückgabeparameter für die ausgelesene Position
Rückgabewert:	`TRUE` = Die Position konnte ermittelt werden, wurde in der Motor-Membervariablen *dAngle* abgelegt und wird im `double&` Parameter als Winkel zurückgeliefert. `FALSE` = Fehlerfall: Motor war in Fahrt, oder die Position konnte nicht ausgelesen werden.
Siehe auch:	`mGetValue`, `mlGetValue`, `mlGetDistance`

mStopDrive

Syntax:	`void WINAPI mStopDrive( BOOL restart )`
Beschreibung:	Mit dieser Funktion wird die Bewegung des aktuellen Antriebs gestoppt. Abhängig vom Parameter *restart* wird der Motor abrupt (*restart* = `FALSE`)oder kontrolliert angehalten.
Parameter:	`BOOL` *restart* - gibt an, ob ein Neustart des Antriebs möglich sein soll (*restart* = `TRUE`) oder ob der Motor durch das Aufheben des Motorstroms sofort gestoppt werden soll (*restart* = `FALSE`).
Rückgabewert:	Obwohl die aufgerufenen Motorfunktionen ihrerseits prüfen, ob die Motorkommandos erfolgreich abgesendet werden konnten, wird der Rückgabeparameter von der `mStopDrive` Funktion nicht mehr weiterverarbeitet.
Siehe auch:	`mMoveToDistance`, `mMoveByDistance`

mGetDistanceProcess

Syntax:	`double WINAPI mGetDistanceProcess( void )`
Beschreibung:	Die Funktion ermittelt die aktuelle Motorposition des aktiven Antriebs von der jeweiligen Steuerkarte und speichert diese in der Motor-Membervariablen *lPosition* ab. Außerdem wird die Position von Encoderschritten ins Winkelmaß umgerechnet und in der Membervariablen *dAngel* abgelegt. Der Wert von *dAngel* wird als Rückgabewert geliefert.
Parameter:	-
Rückgabewert:	`double` Wert der Motor-Membervariablen *dAngle* = aktuelle Winkelposition des aktiven Antriebs
Siehe auch:	`mGetDistance`

mGetValue

Syntax: `double WINAPI mGetValue( TValueType )`

Beschreibung: Abhängig vom Wert des Parameters *vtype* wird der dazugehörige Wert der Motorgröße der aktiven Achse zurückgeliefert. Der Typ TValueType (deklariert in `comhead.h`) kann folgende Werte annehmen: *Distance* (Winkelstellung des Motors), *MinDistance*, *MaxDistance*, *Speed*, *Width* (Winkelschrittweite). Die Funktion wertet nur die Membervariablen der Motoren aus. Es findet kein Zugriff auf die Hardware statt.

Parameter: `TValueType` *vtype* - Parameter gibt an, welcher Motorwert zurückgeliefert werden soll.

Rückgabewert: `double` Wert der durch den `TValueType` Parameter bestimmten Membervariablen des aktiven Antriebs, wobei folgende Zuordnung stattfindet: *Distance* = *dAngle*, *MinDistance* = *dAngleMin*, *MaxDistance* = *dAngleMax*, *Speed* = *dSpeed* unter Berücksichtigung von *dwVelocity* bzw. *Width* = *dAngleWidth*

Siehe auch: `mGetDistance`, `mlGetValue`, `mlGetDistance`, `mSetValue`

mGetUnitType

Syntax: `TUnitType WINAPI mGetUnitType( void )`

Beschreibung: Diese Funktion ermittelt, welche Einheit für den aktuellen Motor zur Angabe von Positionen verwendet wird, indem der Wert der Motor-Membervariablen *eUnit* zurückgeliefert wird. Der Typ von eUnit ist TUnitType und in der Headerdatei `comhead.h` deklariert. Es sind die folgenden Einheiten möglich: Grad, Minuten, Sekunden, Millimeter, Mikrometer, Channel und None (keine Einheit).

Parameter: -

Rückgabewert: `TUnitType` Wert der Motor-Membervariablen *eUnit* - gibt an, in welcher Einheit die Positionsangaben des aktiven Antriebs im Steuerprogramm vorgenommen werden. Die Enumeration TUnitType ist in `comhead.h` deklariert und kann folgende Werte annehmen: Grad, Minuten, Sekunden, Millimeter, Mikrometer, Channel, None.

Siehe auch: `mGetAxisUnit`, `mGetAxisName`

mSetValue

Syntax:
```
BOOL WINAPI mSetValue( TValueType vtype,
                       double     value )
```

Beschreibung: Die Funktion setzt für den aktuellen Antrieb die Werte von Geschwindigkeit (*Speed*) oder Winkelschrittweite (*Width*). Abhängig vom Wert des Parameters *vtype* vom Typ TValueType (deklariert in `comhead.h`) wird der dazugehörige Wert der Motorgröße der aktiven Achse gesetzt. Hierzu werden nur die Membervariablen der Motoren verändert. Es findet kein Zugriff auf die Hardware statt.

Parameter: `TValueType` *vtype* - Parameter gibt an, welcher Motorwert gesetzt werden soll. Zulässig sind nur die Werte *Speed* (Geschwindigkeit) → Membervariable *dSpeed* und *Width* (Winkelschrittweite) → Membervariable *dAngleWidth*.
`double` value - zuzuweisender Wert der durch den `TValueType` Parameter bestimmten Motorgröße des aktiven Antriebs (bei *Speed* ist die Einheit des Werts Units pro Sekunde, der Wert wird automatisch durch `MaxVelocity`) beschränkt).

Rückgabewert: `TRUE` = *Speed* bzw. *Width* konnten korrekt verändert werden.

	`FALSE` = Fehlerfall: Entweder sollte ein anderer Wert als *Speed* oder *Width* gesetzt werden, oder der Wert der Winkelschrittweite lag außerhalb der Softwarebeschränkung.
Siehe auch:	`mGetValue`, `mlGetValue`

mActivateDrive

Syntax:	`void WINAPI mActivateDrive( void )`
Beschreibung:	Diese Funktion aktiviert den aktuellen Antrieb, indem die sie die Regelung der Motorsteuerkarte für diesen Motor aktiviert.
Parameter:	-
Rückgabewert:	-
Siehe auch:	`mStopDrive`

mSetCorrectionState

Syntax:	`void WINAPI mSetCorrectionState( BOOL state )`
Beschreibung:	Mit dieser Funktion wird die Korrektur bei der Umrechnung von Positionen in Winkelstellungen mit Hilfe der Funktion `TMotor::Translate` gesteuert. Ist die Korrektur aktiviert, erfolgt die Umrechnung der Motorposition von Encoderschritten in Winkel über ein Polynom 3. Grades. Andernfalls wird eine lineare Korrektur vorgenommen. Außerdem wird ermittelt, ob der Antrieb Beugung grob (DC) vorhanden ist, um die Abhängigkeiten von DC und DF (Beugung fein) auszugleichen. Da sich die Motoren DC und DF bei Bewegung gegenseitig, bedingt durch die Kopplung mittels eines gekrümmten Hebels, beeinflussen, werden die Positionen der beiden Antriebe korrigiert.

Parameter:	`BOOL` *state* - gibt an, ob die Korrektur bei der Positionsumrechnung in Winkelangaben durch ein Polynom 3. Grades (*state* = `TRUE`) oder linear (*state* = `FALSE`) erfolgen soll. Achtung: Wenn der aktuelle Antrieb nicht kalibriert ist (Membervariable *bCalibrated* = `FALSE`) oder nicht korrigiert werden kann bzw. soll (*bCorrection* = `FALSE`), setzt die Funktion den Korrekturstatus auf lineare Korrektur (*eCorrStatus* = `CorrLinear`).
Rückgabewert:	-
Siehe auch:	-

mGetAxisName

Syntax:	`LPCSTR WINAPI mGetAxisName( void )`
Beschreibung:	Liefert den Namen des aktuellen Antriebs zurück. Der Name wird bei der Initialisierung der Dynamischen Bibliothek `motors.dll` aus dem *Name*-Eintrag des Motors in der Konfigurationsdatei ausgelesen und in der TMotor-Membervariablen *szCharacteristic* gespeichert.
Parameter:	-
Rückgabewert:	`LPCSTR` Wert der Motor-Membervariablen *szCharacteristic*, in der der Name des aktuellen Antriebs aus der INI-Datei abgelegt ist.
Siehe auch:	`mGetAxisUnit`, `mlParsingAxis`, `mlGetIdByName`, `mlIsAxisValid`

mGetAxisUnit

Syntax:	`LPCSTR WINAPI mGetAxisUnit( void )`
Beschreibung:	Die Funktion liefert den Wert der TMotor-Membervariablen *szUnit* als Einheit für die Positionsangaben des aktiven Antriebs, wie sie im Steuerpro-

gramm vorgenommen werden, zurück. Die Einheit wird bei der Initialisierung der Dynamic Link Library `motors.dll` aus dem *Unit*-Eintrag des Motors in der Konfigurationsdatei ausgelesen und in *szUnit* gespeichert.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Membervariablen *szUnit*, die die Einheit für die Positionsangaben des aktiven Antriebs enthält

Siehe auch: `mGetUnitType`, `mGetAxisName`

mGetDF

Syntax: `LPCSTR WINAPI mGetDF( void )`

Beschreibung: Gibt die Formatzeichenkette für die Stellengenauigkeit zur Darstellung der Motorwerte des aktuellen Antriebs im Steuerprogramm zurück.
Das Format wird mit Hilfe des *Digits*-Eintrags in der Konfigurationsdatei beim Initialisieren des Motors festgelegt und in der Motor-Membervariable *DFmt* als Zeichenkette der Form `"%.2lf"` abgespeichert. *DFmt* wird dazu verwendet, die Werte bestimmter Antriebsgrößen wie z.B. *dMinAngle*, *dSpeed*, *dDistance* etc. mit einer einheitlichen und sinnvollen Anzahl von Nachkommastellen in den Dialogen zu präsentieren.
Es wird eine Nachkommastelle weniger als bei Verwendung des Formatstrings *SFmt* berücksichtigt.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Variablen *DFmt*, der bei der Initialisierung des aktuellen Antriebs abhängig vom Konfigurationsdateieintrag *Digits* mit einem Formatstring der Art `"%.2lf"` belegt wurde.

Siehe auch: `mGetSF`

mGetSF

Syntax: `LPCSTR WINAPI mGetSF( void )`

Beschreibung: Die Funktion gibt im Gegensatz zu `mGetSF` eine Formatzeichenkette für die Schrittweitenwerte des aktuellen Antriebs zur Festlegung der Darstellungsgenauigkeit im Steuerprogramm zurück.
Das Format wird mit Hilfe des *Digits*-Eintrags in der Konfigurationsdatei beim Initialisieren des Motors festgelegt und in der Motor-Membervariable *SFmt* als Zeichenkette der Form `"%.2lf"` abgespeichert. *SFmt* wird hauptsächlich dazu verwendet, die Werte der Antriebsschrittweite (Membervariable *dAngleWidth*) mit einer einheitlichen und sinnvollen Anzahl von Nachkommastellen in den Dialogen zu präsentieren.
Es wird eine Nachkommastelle mehr als bei Verwendung des Formatstrings *DFmt* berücksichtigt.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Variablen *SFmt*, der bei der Initialisierung des aktuellen Antriebs abhängig vom Konfigurationsdateieintrag *Digits* mit einem Formatstring der Art `"%.2lf"` belegt wurde.

Siehe auch: `mGetDF`

mSetAngleDefault

Syntax: `void WINAPI mSetAngleDefault( void )`

Beschreibung: Setzt die Voreinstellungen für das Winkeloffset der Relativen Null und die Softwareschranken im Winkelmaß. Dazu werden die Membervariablen *dAngleBias*, *dAngleMin* und *dAngleMax* des aktuellen Antriebs auf ihre Defaultwerte zurückgestellt.

	Für *dAngleMin* und *dAngleMax* werden die Membervariablen *lPositionMin* und *lPositionMax*, die die zulässigen Positionsbereiche in Encoderschritten darstellen, ausgewertet; *dAngleBias* wird auf 0.0 gesetzt.
Parameter:	-
Rückgabewert:	-
Siehe auch:	`mlSetAngleDefault`

mSetRelativeZero

Syntax:	`void WINAPI mSetRelativeZero( BOOL status, double offset )`
Beschreibung:	Setzt das Offset für die Relative Null des aktuellen Antriebs. Der `double` Wert des Offsets wird in der Motor-Membervariablen *dAngleBias* abgelegt. Damit wird festgelegt, ob bei Positionsangaben (im Winkelmaß) für den aktuellen Antrieb mit relativen oder absoluten Angaben gearbeitet wird.
Parameter:	`BOOL` *status* - gibt an, ob die Relative Null gesetzt werden soll ($\rightarrow$ *status* = `TRUE`) oder ob sie wieder aufgehoben werden soll. ($\rightarrow$ *status* = `FALSE`) `double` *offset* - der Wert des Offsets (Winkelstellung des Antriebs, an dem die Relative Null gesetzt wird), der von den absoluten Motorpositionen abgezogen wird
Rückgabewert:	-
Siehe auch:	`mIsDistanceRelative`, `mlGetOffset`

mExecuteCmd

Syntax: `int WINAPI mExecuteCmd( LPSTR command )`

Beschreibung: Sendet die Zeichenkette mit dem Motorkommando direkt an die zum aktuellen Antrieb gehörige Motorkarte. Es ist es aber nur vorgesehen, Motoren der Steuerkarten vom Typ C-812 anzusteuern. Wenn die Motorsteuerkarte betriebsbereit ist und es sich um ein gültiges Kommando handelt, wird dieses ausgeführt, ansonsten wird ein Fehlercode zurückgeliefert. Bei C-832-Karten wird kein Kommando ausgeführt, und die Funktion kehrt sofort mit Erfolg zurück, was sicherlich nicht die beste Lösung ist.

Parameter: `LPSTR` *command* - ein gültiges Motorkommando für die C-812-DC-Controllerkarte
Z.B.: „Move Absolute"-Kommando für C-812-Karten: `[`*boardID*`] MA [`*position*`] [RETURN]`
Dabei gibt *boardID* die vom aktuellen Antrieb belegte Achsennummer auf der Steuerkarte an (z.B. 1. Achse am C-812-Controller → *boardID*=1). Der Parameter *position* stellt die anzufahrende Position in Encoderschritten dar. Außerdem muß jedes Kommando mit dem ASCII-Code 13_{dec} (RETURN) abgeschlossen werden.

Rückgabewert: `int` Wert des Fehlercodes, der bei dem Versuch der Kommandoausführung zurückgeliefert wird. Als Fehlercodes sind folgende Werte in `comhead.h` definiert:
`R_OK` = 999 → die Übertragung und Ausführung des Kommandos konnte ohne Fehler vorgenommen werden.
`R_Failure` = 202 → der Kommandomodus der Steuerkarte ist bereits aktiv, d.h., das Kommando kann z.Z. nicht angenommen werden.
`R_TimeOut` = 214 → das Kommando konnte nicht im Zeitrahmen an die Hardware gesendet werden.
`0` → die DC-Controllerkarte ist nicht korrekt vom Steuerprogramm erkannt worden.

Siehe auch: -

mPushSettings

Syntax: `void WINAPI mPushSettings( void )`

Beschreibung: Diese Funktion dient dazu, die Werte der Motormembervariablen *dAngle*, *dAngleMin*, *dAngleMax*, *dAngleWidth* und *dSpeed* in der zum aktuellen Antrieb gehörigen Struktur *Settings* vom Typ TMSettings (deklariert in `m_motcom.h`) zwischenzuspeichern. Dabei werden eventuell früher gespeicherte Werte ohne Warnung überschrieben. Zwischen zwei `mPushSettings`-Aufrufen sollte also ein Aufruf von `mPopSettings` erfolgen, sonst gehen die zuerst gespeicherten Werte verloren.

Parameter: -

Rückgabewert: -

Siehe auch: `mPopSettings`

mPopSettings

Syntax: `void WINAPI mPopSettings( TMParameter mParam )`

Beschreibung: Schreibt die mit der Funktion `PushSettings` in der Struktur *Settings* gespeicherten Werte in die entsprechenden Member-Variablen des aktuellen Antriebs zurück. Die Struktur *Settings* ist eine Motormembervariable vom Typ TMSettings, deklariert in `m_motcom.h`. Beim Zurückschreiben werden die Werte für die Softwareschranken *dAngleMin* und *dAngleMax*, die Winkelschrittweite *dAngleWidth* und die Motorgeschwindigkeit *dSpeed* geändert. Abhängig vom übergebenen Parameter *mParam* vom Typ `TMParameter` (deklariert in `comhead.h`) besteht die Möglichkeit, die alte Winkelstellung, d.h. den alten Wert von *dAngle*, wiederherzustellen.

Parameter: `TMParameter` *mParam* - diese Enumeration ist in

`comhead.h` deklariert und kann folgende Werte annehmen: *ThisPosition*, *OldPosition*, *WaitExecution*, *NoWait*. Die Funktion `mPopSettings` testet aber nur, falls sich der Motor nicht mehr an der alten gespeicherten Winkelstellung befindet, ob *mParam* den Wert *OldPosition* hat. Dann wird die alte Motorposition angefahren.

Rückgabewert: -

Siehe auch: `mPushSettings`

mStartMoveScan

Syntax: `void WINAPI mStartMoveScan( int nTimeTicks, int )`

Beschreibung: Startet einen Scan zum Optimieren des Anfahrverhaltens der Antriebe C-812ISA, C-812GPIB und C-832. Aufgerufen wird diese Funktion vom Dialog *Einstellungen/Antriebe/Optimieren...* aus dem Hauptfenster des RTK-Steuerprogramms. Die Funktion initialisiert zunächst den Speicher für die Scan-Daten. Die Anzahl der Meßwerte ist durch das Steuerprogrammm vorgegeben. Anschließend wird je nach aktuellem Antrieb der Scan gestartet. Dies geschieht mit der Funktion `StartCheckScan` die den Antrieb um *wPositionWidth* relativ zur aktuellen Position fährt. Das heißt, es wird die Position „aktuelle Position"+ *wPositionWidth* angefahren. Die Positionen werden in Encoderschritten angegeben. Danach wird ein Timer aktiviert, der durch sein Auslösen die Funktion `mSavePosition` aufruft. Je nach Antrieb wird nach entsprechenden Zeitintervallen die Funktion `mSavePosition` gestartet.

Parameter: Der 1. Parameter gibt an, in welchen Zeitabständen der Timer aufgerufen wird, der in `StartCheckScan` gestartet wurde. Die Einheit wird in Millisekunden angegeben.

Der 2. Parameter ist unwichtig. Wahrscheinlich war die Entwicklung noch nicht abgeschlossen und deshalb hat der 2. Parameter keine Funktion. Wichtig ist nur, daß dieser Parameter beim Aufruf von `mStartMoveScan` nicht vergessen wird. Sinnvoll ist es ihn mit 0 zu initialisieren.

Rückgabewert: -

Siehe auch: `mlOptimizingDlg`, `mGetMoveScan`, `mGetScanSize`, `mSavePosition`

mSavePosition

Syntax: `void CALLBACK mSavePosition(UINT,UINT,DWORD,DWORD, DWORD)`

Beschreibung: Funktion die aufgerufen wird, wenn ein Timer während eines durch `mStartMoveScan` gestarteten Scans ausgelöst wurde. Die Funktion `StartCheckScan` aktiviert den Timer und gibt als Bearbeitungfunktion `mSavePosition` an. Je nach ausgewähltem Motor wird in einem bestimmten Zeitintervall diese Funktion durch den Timer aufgerufen. Sie speichert dann sukzessive die aktuelle Position in einem Datenfeld ab. Wenn der Motor stoppt, bevor das Scanfile vollständig mit Meßwerten gefüllt ist, wird der *MoveFinishIdx* gesetzt. Dieser kennzeichnet die letzte Position, an der der Motor in Bewegung war. Wurden alle Meßwerte erfaßt und der Motor befindet sich noch nicht im Stillstand, so wird der *MoveFinishIdx* auf 0 gesetzt. Am Ende des Scans wird der Timer wieder deaktiviert und eine Nachricht, daß der Scan abgeschlossen wurde, in die Nachrichtenschleife verschickt.

Parameter: Der 1. Parameter ist die ID des Timers der diese Callback-Funktion aufruft. Über diese ID kann der Timer identifiziert und wenn er nicht mehr benötigt wird, gelöscht werden. Die restlichen Parameter haben in diesem Fall keine Funktion.

Rückgabewert: -

Siehe auch: `mStartMoveScan`, `mGetMoveScan`, `mGetMoveFinishIdx`

mGetMoveScan

Syntax: `LPLONG WINAPI mGetMoveScan( void )`

Beschreibung: Diese Funktion übergibt die Daten, die beim Scan von `mStartMoveScan` zum Optimieren der Dialoge erfaßt wurden. Dabei handelt es sich um ein Feld von Positionsdaten, die in einem Fenster visualisiert werden.

Parameter: -

Rückgabewert: Pointer auf ein Datenfeld mit LONG-Elementen, die die ermittelten Positionsdaten des Checkscans enthalten.

Siehe auch: `mStartMoveScan`, `mGetScanSize`

mGetScanSize

Syntax: `int WINAPI mGetScanSize( void )`

Beschreibung: Gibt die Größe des Datenfeldes an, in dem die Meßwerte des Scans stehen, der in `mStartMoveScan` gestartet wurde.

Parameter: -

Rückgabewert: Die Größe des Scans ist vom Steuerprogramm festgelegt auf *nScanSize*=150 Werte.

Siehe auch: `mStartMoveScan`, `mGetMoveScan`

mGetMoveFinishIdx

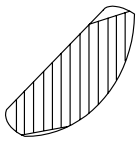
Syntax:	`int WINAPI mGetMoveFinishIdx( void )`
Beschreibung:	Der Wert *MoveFinishIdx* wird in `mSavePosition` ermittelt. Er gibt an, bei welchem Meßwert im Datenfeld des Scans der betrachtete Motor seine Bewegung beendet hatte. War der Motor nach Beenden des Scans noch in Bewegung, so erhält *MoveFinishIdx* den Wert 0.
Parameter:	-
Rückgabewert:	Der Rückgabewert enthält den Index an dem der Motor den Stillstand erreicht hatte.
Siehe auch:	`mStartMoveScan`, `mSavePosition`

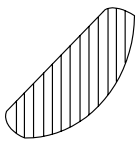
Anhang B

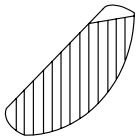
Übersicht über Testfälle mit probenabhängigen Parametern

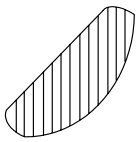
Die hier aufgelisteten Testfälle geben einen genauen Einblick in die Tests mit probenabhängigen Parametern aus dem Abschnitt 8.4.3. Diese wurden aus Platzgründen hier in den Anhang gelegt.

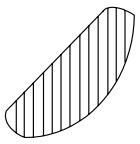
B.1 Testreihe mit Probe $CuAsSe_2$ auf GaAs

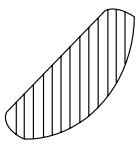
Testfall 1 - $CuAsSe_2$ auf GaAs-Schicht			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 20	145.35	143.4	
TL: 20	-5.6	-1.92	
CC: 50	-453	-490	
Kriterien			
Intensität:	≈21000	≈44300	
HWB:	k.A.	k.A.	
Dauer der Justage:		14:50 min	

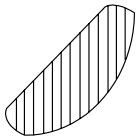
Testfall 2 - $CuAsSe_2$ auf GaAs-Schicht			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 20	152.55	143.2	
TL: 20	-5.6	0.3	
CC: 100	-453	-494.6	
Kriterien			
Intensität:	≈21000	≈47000	
HWB:	k.A.	k.A.	
Dauer der Justage:		15:33 min	

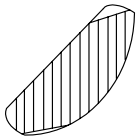
Testfall 3 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	152.1	146.2	
TL:	20	-5.6	-0.12	
CC:	50	-453	-486.8	
Kriterien				
Intensität:		≈21000	≈45700	
HWB:		k.A.	k.A.	
Dauer der Justage:			13:49 min	

Testfall 4 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	151	147.57	
TL:	20	-5.6	4.2	
CC:	100	-453	-488.99	
Kriterien				
Intensität:		≈21000	≈46500	
HWB:		k.A.	k.A.	
Dauer der Justage:			15:01 min	

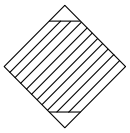
Testfall 5 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	151	146.4	
TL:	20	-5.6	1.2	
CC:	150	-453	-490	
Kriterien				
Intensität:		≈21000	≈47000	
HWB:		k.A.	k.A.	
Dauer der Justage:			16:32 min	

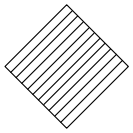
Testfall 6 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	151	148	
TL:	20	-5.6	3.2	
CC:	200	-453	-493.1	
Kriterien				
Intensität:		≈21000	≈46000	
HWB:		k.A.	k.A.	
Dauer der Justage:			17:15 min	

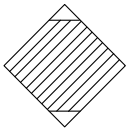
Testfall 7 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	151	145.02	
TL:	50	-5.6	0.97	
CC:	50	-453	-484.8	
Kriterien				
Intensität:		≈21000	≈48000	
HWB:		k.A.	k.A.	
Dauer der Justage:			15:23 min	

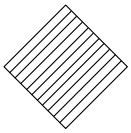
Testfall 8 - $CuAsSe_2$ auf GaAs-Schicht				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	151	138	
TL:	50	-5.6	-5.06	
CC:	150	-453	-522.98	
Kriterien				
Intensität:		≈21000	≈43000	
HWB:		k.A.	k.A.	
Dauer der Justage:			18:35 min	

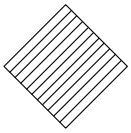
B.2 Testreihe mit Probe SiGe auf Si-Schicht (PF916324/8/8)

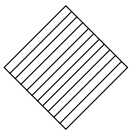
Testfall 1 - PF916324/8/8				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	0	3.87	
TL:	20	0	1.10	
CC:	50	705.7	752.8	
Kriterien				
Intensität:		≈25000	≈43000	
HWB:		15.75	12.29	
Dauer der Justage:			14:24 min	

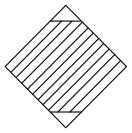
Testfall 2 - PF916324/8/8				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	0	0.1	
TL:	20	0	-1.10	
CC:	100	705.7	802.1	
Kriterien				
Intensität:		≈18000	≈49000	
HWB:		15.75	10.49	
Dauer der Justage:			15:57 min	

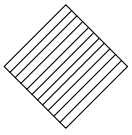
Testfall 3 - PF916324/8/8				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	4.13	
TL:	20	0	0.3	
CC:	50	705.7	754	
Kriterien				
Intensität:		≈18000	≈42000	
HWB:		15.75	11.8	
Dauer der Justage:			16:06 min	

Testfall 4 - PF916324/8/8				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	1.48	
TL:	20	0	0.3	
CC:	100	705.7	787.7	
Kriterien				
Intensität:		≈20000	≈44000	
HWB:		15.75	10.9	
Dauer der Justage:			16:47 min	

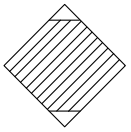
Testfall 5 - PF916324/8/8				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	-6.03	
TL:	20	0	-3.4	
CC:	150	705.7	849.2	
Kriterien				
Intensität:		≈25000	≈55000	
HWB:		15.75	9.29	
Dauer der Justage:			17:48 min	

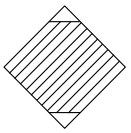
Testfall 6 - PF916324/8/8			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	0	2.08	
TL: 20	0	0.88	
CC: 200	705.7	811.5	
Kriterien			
Intensität:	≈22000	≈48000	
HWB:	15.75	10.4	
Dauer der Justage:		17:52 min	

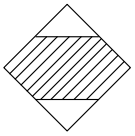
Testfall 7 - PF916324/8/8			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	0	-0.29	
TL: 50	0	-1.09	
CC: 50	705.7	754.94	
Kriterien			
Intensität:	≈22000	≈41000	
HWB:	15.75	12.84	
Dauer der Justage:		18:13 min	

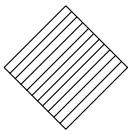
Testfall 8 - PF916324/8/8			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	0	1.47	
TL: 50	0	-5.52	
CC: 150	705.7	854.94	
Kriterien			
Intensität:	≈22000	≈51000	
HWB:	15.75	9.49	
Dauer der Justage:		19:07 min	

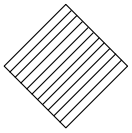
B.3 Testreihe mit Probe PF916324/04/20

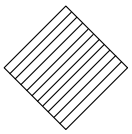
Testfall 1 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	0	11.46	
TL:	20	0	7.96	
CC:	50	0	-3.7	
Kriterien				
Intensität:		≈33000	≈53000	
HWB:		12.88	10.6	
Dauer der Justage:			15:18 min	

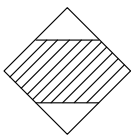
Testfall 2 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	0	7.83	
TL:	20	0	5.96	
CC:	100	0	-3.55	
Kriterien				
Intensität:		≈30000	≈53000	
HWB:		12.88	10.57	
Dauer der Justage:			14:17 min	

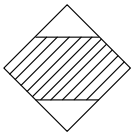
Testfall 3 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	9.25	
TL:	20	0	8.05	
CC:	50	0	2.78	
Kriterien				
Intensität:		≈35000	≈52000	
HWB:		12.88	11.1	
Dauer der Justage:			17:09 min	

Testfall 4 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	15.09	
TL:	20	0	7.63	
CC:	100	0	-69.56	
Kriterien				
Intensität:		≈35000	≈62000	
HWB:		12.88	8.07	
Dauer der Justage:			17:48 min	

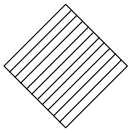
Testfall 5 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	10.02	
TL:	20	0	6.17	
CC:	150	0	-66.82	
Kriterien				
Intensität:		≈35000	≈61000	
HWB:		12.88	8.35	
Dauer der Justage:			15:24 min	

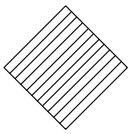
Testfall 6 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	21.28	
TL:	20	0	9.98	
CC:	200	0	-100.77	
Kriterien				
Intensität:		≈35000	≈64000	
HWB:		12.88	7.85	
Dauer der Justage:			18:52 min	

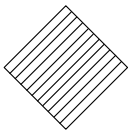
Testfall 7 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	8.75	
TL:	50	0	7.01	
CC:	50	0	-4.05	
Kriterien				
Intensität:		≈35000	≈52000	
HWB:		12.88	11.21	
Dauer der Justage:			15:23 min	

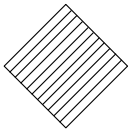
Testfall 8 - PF916324/04/20				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	0	9.15	
TL:	50	0	4.46	
CC:	150	0	14.04	
Kriterien				
Intensität:		≈31000	≈51000	
HWB:		12.88	11.24	
Dauer der Justage:			18:29 min	

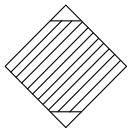
B.4 Testreihe mit Probe PF916324/8/7

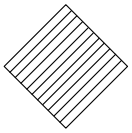
Testfall 1 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	-33.68	-35.32	
TL:	20	8.29	7.79	
CC:	50	-125.71	-71.3	
Kriterien				
Intensität:		≈33000	≈56000	
HWB:		13.34	8.9	
Dauer der Justage:			13:47 min	

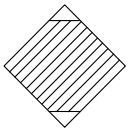
Testfall 2 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	-33.68	-37.83	
TL:	20	8.29	7.23	
CC:	100	-125.71	-76.4	
Kriterien				
Intensität:		≈33000	≈57000	
HWB:		13.34	8.89	
Dauer der Justage:			15:03 min	

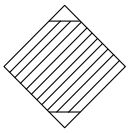
Testfall 3 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-36.34	
TL:	20	8.29	7.98	
CC:	50	-125.71	-89.09	
Kriterien				
Intensität:		≈35000	≈57000	
HWB:		13.34	9.0	
Dauer der Justage:			13:27 min	

Testfall 4 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-37.34	
TL:	20	8.29	7.45	
CC:	100	-125.71	-84.09	
Kriterien				
Intensität:		≈35000	≈56000	
HWB:		13.34	9.1	
Dauer der Justage:			14:39 min	

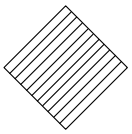
Testfall 5 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-35.52	
TL:	20	8.29	8.29	
CC:	150	-125.71	-64.83	
Kriterien				
Intensität:		≈35000	≈53000	
HWB:		13.34	9.35	
Dauer der Justage:			15:32 min	

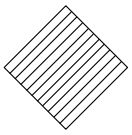
Testfall 6 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-38.72	
TL:	20	8.29	6.3	
CC:	200	-125.71	-80.22	
Kriterien				
Intensität:		≈35000	≈58000	
HWB:		13.34	8.54	
Dauer der Justage:			18:27 min	

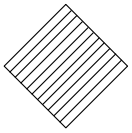
Testfall 7 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-35.72	
TL:	50	8.29	8.3	
CC:	50	-125.71	-61.32	
Kriterien				
Intensität:		≈35000	≈52000	
HWB:		13.34	9.45	
Dauer der Justage:			14:15 min	

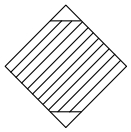
Testfall 8 - PF916324/8/7				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	-33.68	-32.01	
TL:	50	8.29	8.6	
CC:	150	-125.71	-56.32	
Kriterien				
Intensität:		≈35000	≈51000	
HWB:		13.34	9.7	
Dauer der Justage:			18:45 min	

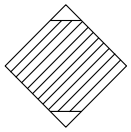
B.5 Testreihe mit Probe PF916324/08/5

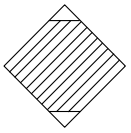
Testfall 1 - PF916324/08/5				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	28.28	30.02	
TL:	20	-11	-5.32	
CC:	50	-90	-52.61	
Kriterien				
Intensität:		≈33000	≈43000	
HWB:		12.5	9.63	
Dauer der Justage:			14:20 min	

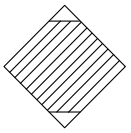
Testfall 2 - PF916324/08/5				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	33.26	30.02	
TL:	20	-7.1	-1.19	
CC:	100	-161.6	-63.89	
Kriterien				
Intensität:		≈27000	≈41300	
HWB:		15.37	9.6	
Dauer der Justage:			15:55 min	

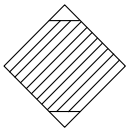
Testfall 3 - PF916324/08/5				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	27.4	28.63	
TL:	20	-11	-8.4	
CC:	50	-90	-55.4	
Kriterien				
Intensität:		≈33000	≈43000	
HWB:		12.5	9.6	
Dauer der Justage:			14:53 min	

Testfall 4 - PF916324/08/5				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	27.8	33.67	
TL:	20	-11	-3.94	
CC:	100	-90	-72.63	
Kriterien				
Intensität:		≈33000	≈42000	
HWB:		12.5	9.8	
Dauer der Justage:			17:57 min	

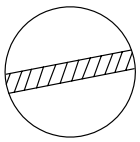
Testfall 5 - PF916324/08/5				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	28.17	32.08	
TL:	20	-11	-3.5	
CC:	150	-90	-81.5	
Kriterien				
Intensität:		≈33000	≈41000	
HWB:		12.5	9.9	
Dauer der Justage:			14:36 min	

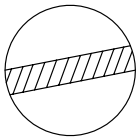
Testfall 6 - PF916324/08/5			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	28.17	33.1	
TL: 20	-11	-4.2	
CC: 200	-90	-75.3	
Kriterien			
Intensität:	≈33000	≈42000	
HWB:	12.5	9.8	
Dauer der Justage:		14:36 min	

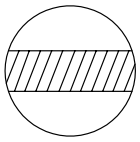
Testfall 7 - PF916324/08/5			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	28.17	34.02	
TL: 50	-11	-6.2	
CC: 50	-90	-83.3	
Kriterien			
Intensität:	≈33000	≈41000	
HWB:	12.5	10.24	
Dauer der Justage:		16:16 min	

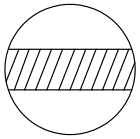
Testfall 8 - PF916324/08/5			
Suchbereich	**Startwerte**	**Endwerte**	**Ausleuchtung**
DF: 50	27.47	32.75	
TL: 50	-11	-2.93	
CC: 150	-90	-86.3	
Kriterien			
Intensität:	≈33000	≈40500	
HWB:	12.5	10.35	
Dauer der Justage:		15:09 min	

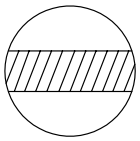
B.6 Testreihe mit Kollimatorprobe

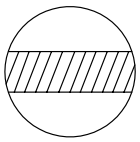
Testfall 1 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	50.6	51.39	
TL:	20	6.2	-13.5	
CC:	50	120.3	83.5	
Kriterien				
Intensität:		≈5100	≈10500	
HWB:		10.97	8.52	
Dauer der Justage:			13:01 min	

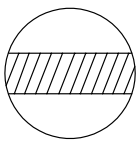
Testfall 2 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	20	50.6	53.17	
TL:	20	6.2	-13.98	
CC:	100	120.3	46.39	
Kriterien				
Intensität:		≈5200	≈15300	
HWB:		10.85	8.15	
Dauer der Justage:			16:36 min	

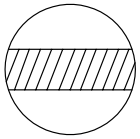
Testfall 3 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	50.88	52,3	
TL:	20	-17.5	-17.5	
CC:	50	29.3	20.8	
Kriterien				
Intensität:		≈28000	≈34000	
HWB:		7.8	6.78	
Dauer der Justage:			15:12 min	

Testfall 4 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	54.3	53.4	
TL:	20	-12	-19.1	
CC:	100	29.3	35.47	
Kriterien				
Intensität:		≈20000	≈31500	
HWB:		8.0	7.24	
Dauer der Justage:			16:50 min	

Testfall 5 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	54.3	52.13	
TL:	20	-12	-15.3	
CC:	150	29.3	20.3	
Kriterien				
Intensität:		≈22000	≈30000	
HWB:		8.1	7.5	
Dauer der Justage:			15:31 min	

Testfall 6 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	57.2	54.5	
TL:	20	-12	-18.7	
CC:	200	29.3	35.51	
Kriterien				
Intensität:		≈21500	≈33000	
HWB:		8.05	6.92	
Dauer der Justage:			16:05 min	

Testfall 7 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	54.9	51.9	
TL:	50	-12	-17.8	
CC:	50	30	21.5	
Kriterien				
Intensität:		≈21500	≈35000	
HWB:		8.1	6.51	
Dauer der Justage:			14:31 min	

Testfall 8 - Kollimatorprobe				
Suchbereich		**Startwerte**	**Endwerte**	**Ausleuchtung**
DF:	50	54.1	59.8	
TL:	50	-12	-19.3	
CC:	150	29.3	20.4	
Kriterien				
Intensität:		≈20000	≈32500	
HWB:		8.1	7.36	
Dauer der Justage:			15:37 min	

Anhang C

Kommentierte Quelltexte

C.1 Dialogklasse TAutomaticAngleControl

C.1.1 m_justag.h

```
#ifndef __M_JUSTAG_H
#define __M_JUSTAG_H

#include "matrix.h"
#include "transfrm.h"


// globale Variable: legt fest ob das JustageLog geschrieben wird
// definiert in m_justag.cpp
extern bool bWriteLogfile;

// globale Funktion um Statusmeldungen in das Logfile
// für die Automatische Justage schreiben
void WriteToJustageLog(char* buf);

// Dialogklasse der Automatischen Justage
class TAutomaticAngleControl : public TModalDlg
{
 public:
           TAutomaticAngleControl();
  BOOL     Dlg_OnInit ( HWND, HWND, LPARAM );
  void     Dlg_OnCommand( HWND, int, HWND, UINT );
  BOOL     LeaveDialog(void);


 private:
  // Tranformationsobjekt als Member definiert
  TransformationClass *Transform;

  TDevice* Sensor;

  // Inhalt des Statusfeldes
  char * status;

  //Aktivierung der Abbruchkriteriums Intensitätsdifferenz
  bool bIntensityDiffActive;
```

```
  // maximale Größe fuer das Status-Textfeld
  unsigned nMaxString;

  // Temporäre Puffer zur Stringzusammensetzung
  char *buf, *buf1;

  // Laufvariablen
  int durchlauf,step,step1,count;

  // Anzahl der Zeilen im Status-Textfeld
  DWORD dwStatusZeilen;

  // Zahl der durchzuführenden Messungen pro Position
  unsigned nMeasureCount;

  // Positionsvariablen, Intensitätsvariablen
  double fPosition, intervall_li, intervall_re, x,y,z,x2,y2,z2;
  double MaxTL,MaxDF,MaxCC;
  float fIntenseDifferenz

  // Variablen zur Ausgabe der Justagedauer
  struct date datum;
  struct time zeit;
  int justageanfang, justageende, justagezeit;

  // Variablen zur Speicherung der Wertebereiche & Intensitätspositionen
  TMotorPositionsWerte Wertebereich, aktWertebereich;
  TIntensityPosition MaxIntensity, LastIntensity, ActIntensity;
};

#endif
```

C.1.2 m_justag.cpp

```
#include <stdio.h>

// Dialog-Ressourcen, Help-IDs
#include "rc_def.h"
#include "help_def.h"

// Strukturen, Aufzählungen, IDs des RTK-Programms
#include "comhead.h"
#include "m_devcom.h"

#pragma hdrstop

// Interface zur Motorsteuerung
#include "m_layer.h"

// Interface zur Detektorsteuerung (sehr spartanisch)
#include "c_layer.h"

// Dialog automatische Justage
#include "m_justag.h"

// Globale Definition der Motorachsen
// -> Zugriff durch alle Justage- & Tranformationsobjekte möglich
int nMotorTL=-1, nMotorDF=-1, nMotorCC=-1;

// Globales Flag für Schreiben in Justagelog
bool bWriteLogfile;

// Zeiger auf Counterliste
extern LPDList lpDList;

// Statusmeldungen in Prokolldatei für die Automatische Justage schreiben
void WriteToJustageLog(char* buf)
{
  FILE *hLogFile;

  if ( (hLogFile = fopen("Justage.log","a+")) == NULL )
  {
    MessageBox(GetFocus(), "WriteToJustageLog: Fehler beim Öffnen von \
      Justage.log", "fopen", MB_OK);
  }

  if ( fwrite(buf, 1, strlen(buf), hLogFile) != strlen(buf) )
  {
    MessageBox(GetFocus(), "WriteToJustageLog: Fehler beim Schreiben in \
      Justage.log", "fwrite", MB_OK);
  }

  if ( fclose(hLogFile) == EOF )
  {
    MessageBox(GetFocus(), "WriteToJustageLog: Fehler beim Schließen von \
      Justage.log", "fclose", MB_OK);
  }
}
```

```
//*************** Dialog Automatische Justage*********************************
//*************** Selbständiges Einstellen der Probe**************************

TAutomaticAngleControl::TAutomaticAngleControl( void ):
  TModalDlg("AutomaticAngleControl")
{
  nMaxString=32767;

  try
  {
    // Speicher für Status-Textfeld-String & temporäre Puffer reservieren
    status = new char[nMaxString];
    buf = new char[1024];
    buf1= new char[1024];
  }
  catch (xalloc)
  {
    MessageBox(GetFocus(), "TAutomaticAngleControl:Fehler bei \
                            Speicherreservierung", "xalloc", MB_OK);
    PostQuitMessage(-1);
  }
}

BOOL TAutomaticAngleControl::Dlg_OnInit(HWND hwnd, HWND hwndCtl, LPARAM lParam)
{
  TModalDlg::Dlg_OnInit(hwnd, hwndCtl, lParam);

  // verwendeter Zähler für AutoJustage = akt. Zähler aus der Deviceliste
  Sensor = lpDList->DP();

  // Inhalt des Status-Textfeldes auslesen
  GetDlgItemText(GetHandle(), ID_Status, status, nMaxString);

  // Globale Variable für Schreibpflicht in Justagelog
  bWriteLogfile=false;

  //Deaktivierung des Abbruchkriteriums Intensitätsdifferenz
  bIntensityDiffActive=false;
  CheckDlgButton(hwnd,ID_ActivateIntenseDiff,FALSE);
  EnableWindow(GetDlgItem(hwnd,ID_TextMaxIntDiff),FALSE);
  EnableWindow(GetDlgItem(hwnd,ID_IntenseDifferenz),FALSE);

  // Test, ob Zähler korrekt im Programm initialisiert wurde
  if( !dlIsDeviceValid(CounterDevice) )
  {
    strcat(status, "Counterdevice...fehlt!\r\n");
    // im Fehlerfall wird der START-Button deaktiviert
    EnableWindow(GetDlgItem(hwnd,cm_start_justage),FALSE);
  }
  else
  {
    strcat(status, "Counterdevice...ok!\r\n");
    // Sichergehen, dass Zählerfenster offen ist,
    // ansonsten werden die Zählerwerte nicht korrekt eingelesen
    if(!Sensor->bDeviceOpen)
    {
      // Zählerfenster öffnen
      NewWindow((TCounterWindow *)new TCounterWindow(Sensor->GetId()));
    }
  }
```

```
  // Test, ob der Antrieb "Beugung fein" vorhanden ist
  if( !mlIsAxisValid(Omega) )
  {
    strcat(status, "Beugung fein...fehlt!\r\n");
    // im Fehlerfall wird der START-Button deaktiviert
    EnableWindow(GetDlgItem(hwnd,cm_start_justage),FALSE);
  }
  else
  {
    strcat(status, "Beugung fein...ok!\r\n");
    // Bestimmung der MotorID zum Antrieb Omega (Beugung fein, DF)
    nMotorDF=mlGetIdByName(Omega);
  }

  // Test, ob der Antrieb "Tilt" vorhanden ist
  if( !mlIsAxisValid(Psi) )
  {
    strcat(status, "Tilt...fehlt!\r\n");
    // im Fehlerfall wird der START-Button deaktiviert
    EnableWindow(GetDlgItem(hwnd,cm_start_justage),FALSE);
  }
  else
  {
    strcat(status, "Tilt...ok!\r\n");
    // MotorID zuweisen
    nMotorTL=mlGetIdByName(Psi);
  }

  // Test, ob der Antrieb "Kollimator" vorhanden ist
  if( !mlIsAxisValid(Collimator) )
  {
    strcat(status, "Kollimator...fehlt!\r\n");
    // im Fehlerfall wird der START-Button deaktiviert
    EnableWindow(GetDlgItem(hwnd,cm_start_justage),FALSE);
  }
  else
  {
    strcat(status, "Kollimator...ok!\r\n");
    // MotorID zuweisen
    nMotorCC=mlGetIdByName(Collimator);
  }

  // Abfragen des Fehlerzustands über (de)aktivierten START-Button
  if( !IsWindowEnabled(GetDlgItem(hwnd,cm_start_justage)) )
    strcat(status, "Fehler: Es sind nicht alle Geräte bereit!\r\n");

  // Statusmeldung in Status-Textfenster schreiben
  SetDlgItemText(GetHandle(), ID_Status, status);
  return true;
}



void TAutomaticAngleControl::Dlg_OnCommand(HWND hwnd, int id, HWND hwndCtl,
       UINT codeNotify)
{
  switch( id )
  {
    case ID_CANCEL:
      // Text in Programmstatuszeile schreiben
      SetInfo("Automatische Justage verlassen...");
```

```
  // Motor stoppen und Dialog beenden
  LeaveDialog();
  EndDialog(hwnd,id);
break;

case cm_start_justage:
  // START-Button deaktivieren
  EnableWindow(GetDlgItem(hwnd,cm_start_justage),FALSE);
  // Mauszeiger in Sanduhr umwandeln
  SetCursor(LoadCursor(NULL,IDC_WAIT));
  // Transformationsobjekt erzeugen
  Transform = new TransformationClass;
  FORWARD_WM_COMMAND(GetHandle(),cm_set_parameters,0,0,PostMessage);
break;

case cm_set_parameters:
  // Wert für die Toleranz des "Goldenen Schnitts" aus Dialog holen
  GetDlgItemText(hwnd, ID_Toleranz, buf, 4);
  // Vorzeichen eleminieren
  Transform->toleranz = fabs(atof(buf));
  // Bereichsprüfung, bei Fehler: Defaultwert
  if ( (Transform->toleranz > 1.0) || (Transform->toleranz < 0.1) )
  {
    // Defaultwert
    Transform->toleranz = 1.0;
  }
  // Rückschreiben des korrigierten Wertes in den Dialog
  sprintf(buf, "%.1f", Transform->toleranz);
  SetDlgItemText(hwnd, ID_Toleranz, buf);

  // Wert für die Anzahl der Algorithmus-Durchläufe aus Dialog holen
  GetDlgItemText(hwnd, ID_Durchlauf, buf, 4);
  // gegen Vorzeichen hilft "abs"
  durchlauf = abs(atoi(buf));
  // Bereichsprüfung, bei Fehler: Defaultwert
  if ( (durchlauf < 1) || (durchlauf > 7) )
  {
    // Defaultwert
    durchlauf = 5;
  }
  // Rückschreiben des korrigierten Wertes in den Dialog
  sprintf(buf, "%d", durchlauf);
  SetDlgItemText(hwnd, ID_Durchlauf, buf);

  // Wert für die Intensitätsdifferenz aus Dialog holen
  // Differenz soll auch als Abbruchkriterium dienen
  GetDlgItemText(hwnd, ID_IntenseDifferenz, buf, 8);
  // keine negativen Werte akzeptieren
  fIntenseDifferenz = fabs(atof(buf));
  // Bereichsprüfung (untere Schranke)
  if ( fIntenseDifferenz < 1.0 )
  {
    // Defaultwert
    fIntenseDifferenz=5000.0;
  }
  // Bereichsprüfung (obere Schranke)
  if ( fIntenseDifferenz > 10000.0 )
  {
    // Maximalwert
    fIntenseDifferenz=9999.9;
  }
  // Rückschreiben des korrigierten Wertes in den Dialog
```

```
sprintf(buf, "%.0f", fIntenseDifferenz);
SetDlgItemText(hwnd, ID_IntenseDifferenz, buf);

// Werte für die Intervallgrenze von DF aus dem Dialog auslesen
GetDlgItemText(hwnd, ID_DF_Intervall, buf, 8);
// Vorzeichen abfangen
MaxDF = fabs(atof(buf));
// Bereichsprüfung, bei Fehler: Defaultwert
if ( (MaxDF < 1.0) || (MaxDF > 300.0) )
{
  // Defaultwert
  MaxDF = 50.0;
}
// Rückschreiben des korrigierten Wertes in den Dialog
sprintf(buf, "%.0f", MaxDF);
SetDlgItemText(hwnd, ID_DF_Intervall, buf);

// Werte für die Intervallgrenze von TL aus dem Dialog auslesen
GetDlgItemText(hwnd, ID_TL_Intervall, buf, 8);
// Vorzeichen abfangen
MaxTL = fabs(atof(buf));
// Bereichsprüfung, bei Fehler: Defaultwert
if ( (MaxTL < 1.0) || (MaxTL > 100.0) )
{
  // Defaultwert
  MaxTL = 20.0;
}
// Rückschreiben des korrigierten Wertes in den Dialog
sprintf(buf, "%.0f", MaxTL);
SetDlgItemText(hwnd, ID_TL_Intervall, buf);

// Werte für die Intervallgrenze von CC aus dem Dialog auslesen
GetDlgItemText(hwnd, ID_CC_Intervall, buf, 8);
// Vorzeichen abfangen
MaxCC = fabs(atof(buf));
// Bereichsprüfung, bei Fehler: Defaultwert
if ( (MaxCC < 1.0) || (MaxCC > 500.0) )
{
  // Defaultwert
  MaxCC = 150.0;
}
// Rückschreiben des korrigierten Wertes in den Dialog
sprintf(buf, "%.0f", MaxCC);
SetDlgItemText(hwnd, ID_CC_Intervall, buf);

// Werte für die Anzahl der Messungen pro Intensitätsbestimmung
// aus dem Dialog auslesen
GetDlgItemText(hwnd, ID_MeasureCount, buf ,8);
// Vorzeichen filtern
nMeasureCount = fabs(atoi(buf));
// Bereichsprüfung (untere, obere Grenze)
if ( nMeasureCount < 1 )
{
  nMeasureCount = 1;
}
else if ( nMeasureCount > 6 )
{
  nMeasureCount = 6;
}
// Rückschreiben des korrigierten Wertes in den Dialog
sprintf(buf, "%d", nMeasureCount);
SetDlgItemText(hwnd, ID_MeasureCount, buf);
```

```
  FORWARD_WM_COMMAND(GetHandle(),cm_initialize,0,0,PostMessage);
break;

case cm_initialize:
  // Initialisierung des Transform-Objektes mit den Achsen-Suchintervallen
  Transform->Initialize(MaxDF, MaxTL, MaxCC);

  // Softwareschranken für die Maximumsuche bestimmen und abspeichern
  Wertebereich = Transform->GetOrigPosBorders();

  // Ausgangsintensität auslesen
  // ohne Ausgabe in Logfile [dazu: MeasureIntensity ohne bWriteLogfile]
  if ( bWriteLogfile )
  {
    bWriteLogfile = false;
    MaxIntensity.Intensity = Transform->MeasureIntensity(nMeasureCount);
    bWriteLogfile = true;
  }
  else MaxIntensity.Intensity = Transform->MeasureIntensity(nMeasureCount);

  // Ausgangspositionen der Antriebe, an der die
  // Anfangsintensität gemessen wurde, abspeichern
  MaxIntensity.TL = Wertebereich.OrigTL;
  MaxIntensity.DF = Wertebereich.OrigDF;
  MaxIntensity.CC = Wertebereich.OrigCC;

  // LastIntensity wird mit Intensität=0 initialisiert,
  // da noch keine Justage gestartet wurde (Positionen wie MaxIntensity)
  LastIntensity = MaxIntensity;
  LastIntensity.Intensity = 0;

  // Zeit zur Bestimmung der Justagedauer
  getdate(&datum);
  gettime(&zeit);

  // Infos ins Logfile schreiben
  if (bWriteLogfile)
  {
    sprintf(buf,"##########################################################\
    \nAutomatische Justage gestartet: %02i.%02i.%i %2d:%02d:%02d\
    \n##########################################################\
    \nMotorpositionen:\nTL: %.2f Min: %.2f Max: %.2f\
    \nDF: %.2f Min: %.2f Max: %.2f\nCC: %.2f Min: %.2f Max: %.2f\
    \nIntensität: %.2f\n",
    datum.da_day,datum.da_mon,datum.da_year,
    zeit.ti_hour,zeit.ti_min,zeit.ti_sec,
    Wertebereich.OrigTL,Wertebereich.MinTL,Wertebereich.MaxTL,
    Wertebereich.OrigDF,Wertebereich.MinDF,Wertebereich.MaxDF,
    Wertebereich.OrigCC,Wertebereich.MinCC,Wertebereich.MaxCC,
    MaxIntensity.Intensity);
    WriteToJustageLog(buf);
  }

  // Bestimmung der Justageanfangszeitpunktes (Uhrzeit in Sekunden)
  justageanfang=3600*zeit.ti_hour+60*zeit.ti_min+zeit.ti_sec;

  // Status-Textfeld auslesen - nur 72 Zeichen, da bei jeder neuen Justage
  // nur der jeweilige Justagevorgang im Statusfenster erscheinen soll
  GetDlgItemText(GetHandle(), ID_Status, status, 72);
  strcat(status, "\r\nAutomatische Justage gestartet.\r\n");
  SetDlgItemText(GetHandle(), ID_Status, status);
```

```
  // Durchgangszähler für Optimierung
  // dient dazu, gerade und ungerade Durchgänge des Algo zu erfassen,
  // um abhängig davon abwechselnd Tilt oder Kollimator zu optimieren
  count = 0;

  // Test auf Intensitätsüberschreitung des Detektors
  if (!Transform->bIntensityTooHigh)
  {
    // normales Vorgehen
    FORWARD_WM_COMMAND(GetHandle(),cm_choose_axis,0,0,PostMessage);
  }
  else
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }
break;

case cm_choose_axis:
  // Test auf Intensitätsüberschreitung des Detektors
  if (Transform->bIntensityTooHigh)
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }

  // Ende der Justage ist erreicht bei: count = durchlauf * 2
  // ein Durchlauf des Algorithmus in die Justage von Tilt und Kollimator
  if(count < durchlauf * 2)
  {
    count++;

    // Setzen der aktuellen Achsen-Suchbereiche
    // dazu werden die Bereiche gemäß den KS-Transformationen umgerechnet
    aktWertebereich = Transform->translate_PosBorders();

    //alle ungeraden Durchgänge (count=1,3,5,etc.) wird CC/DF optimiert
    if (count % 2 == 1)
    {
      SendMessage(GetHandle(),WM_COMMAND,cm_optimizing_CC,0);
    }
    //alle geraden Durchgänge (count=2,4,etc.) wird TL/DF optimiert
    else
    {
      SendMessage(GetHandle(),WM_COMMAND,cm_optimizing_TL,0);
    }
  }
  else
  {
    // Ende des Algorithmus ist erreicht
    SendMessage(GetHandle(),WM_COMMAND,cm_post_processing,0);
  }
break;

case cm_optimizing_CC:
  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  // 2. Teil des Algorithmus (Kollimator/DF)
  //(nach Überlegung & Test bei den Physikern ist die Variante
  // 1. CC/DF
  // 2. TL/DF
  // der ürsprünglichen vorzuziehen) deshalb ist CC/DF am Anfang
```

```
  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  // Test auf Intensitätsüberschreitung des Detektors
  if (Transform->bIntensityTooHigh)
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }

  //(z-Achse)=CC: Such-Intervalle für Goldenen Schnitt festlegen
  intervall_li=aktWertebereich.MinCC;
  intervall_re=aktWertebereich.MaxCC;

  // # Logging #
  if (bWriteLogfile) WriteToJustageLog("----------Optimierung CC:\r\n");

  // Goldener Schnitt: Optimierung der Z-Achse(CC)
  // step liefert die Anzahl der benötigten Intervallteilungen,
  // intervall_re/li liefern das letzte Optimierungsintervall
  step = Transform->Goldener_Schnitt(Laufachse_Z, intervall_li,
                                     intervall_re, nMeasureCount);

  FORWARD_WM_COMMAND(GetHandle(),cm_optimizing_DF,0,0,PostMessage);
break;

case cm_optimizing_TL:
  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  // 1. Teil des Algorithmus (Tilt/DF)
  // Erklärung, wieso der 1. Teil des Algorithmus an 2. Stelle steht,
  // nachzulesen am Anfang der Justage (2. Teil des Algorithmus)
  //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  // Test auf Intensitätsüberschreitung des Detektors
  if (Transform->bIntensityTooHigh)
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }

  //(x-Achse)=TL: Such-Intervalle für Goldenen Schnitt festlegen
  intervall_li=aktWertebereich.MinTL;
  intervall_re=aktWertebereich.MaxTL;

  //# Logging #
  if (bWriteLogfile) WriteToJustageLog("----------Optimierung TL:\r\n");

  // Goldener Schnitt: Optimierung der X-Achse(TL)
  // step liefert die Anzahl der benötigten Intervallteilungen,
  // intervall_re/li liefern das letzte Optimierungsintervall
  step = Transform->Goldener_Schnitt(Laufachse_X, intervall_li,
                                     intervall_re, nMeasureCount);

  FORWARD_WM_COMMAND(GetHandle(),cm_optimizing_DF,0,0,PostMessage);
break;

case cm_optimizing_DF:
  // Nachregeln mit DF
  //(y-Achse)=DF: Such-Intervalle für Goldenen Schnitt festlegen
```

```
  // Test auf Intensitätsüberschreitung des Detektors
  if (Transform->bIntensityTooHigh)
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }

  //(y-Achse)=DF: Such-Intervalle für Goldenen Schnitt festlegen
  intervall_li=aktWertebereich.MinDF;
  intervall_re=aktWertebereich.MaxDF;

  // # Logging #
  if (bWriteLogfile) WriteToJustageLog("------------Nachregeln DF:\r\n");

  // Goldener Schnitt: Optimierung der Y-Achse(DF)
  // step1 liefert die Anzahl der benötigten Intervallteilungen,
  // intervall_re/li liefern das letzte Optimierungsintervall
  step1 = Transform->Goldener_Schnitt(Laufachse_Y, intervall_li,
                                      intervall_re, nMeasureCount);

  FORWARD_WM_COMMAND(GetHandle(),cm_calculate,0,0,PostMessage);
break;

case cm_calculate:
  // aktuelle Positionen der Antriebe bestimmen, Protokollierung
  // und abspeichern in Positionen von ActIntensity

  // Test auf Intensitätsüberschreitung des Detektors
  if (Transform->bIntensityTooHigh)
  {
    // Abbruch der Justage
    SendMessage(GetHandle(), WM_COMMAND, cm_post_processing, 0);
  }

  mlSetAxis(nMotorTL);
  mGetDistance(fPosition);
  ActIntensity.TL=fPosition;
  // # Logging #
  sprintf(buf, "Reale Positionen: (Durchlauf %.1f von %d)\r\nTL: %.2f",\
          (float)count/2, durchlauf, fPosition);

  mlSetAxis(nMotorDF);
  mGetDistance(fPosition);
  ActIntensity.DF=fPosition;
  // # Logging #
  sprintf(buf1, "  DF: %.2f", fPosition);
  strcat(buf,buf1);

  mlSetAxis(nMotorCC);
  mGetDistance(fPosition);
  ActIntensity.CC=fPosition;
  // # Logging #
  sprintf(buf1, "  CC: %.2f\r\n", fPosition);
  strcat(buf,buf1);

  // Ausgabe in Statusfenster und LogFile
  GetDlgItemText(GetHandle(), ID_Status, status, nMaxString);
  strcat(status,buf);
  if (bWriteLogfile) WriteToJustageLog(buf);

  // aktuelle Intensität auslesen und in ActIntensity speichern
  ActIntensity.Intensity = Transform->MeasureIntensity(nMeasureCount);
```

```
// # Logging #
sprintf(buf, "Intensität: %.2f\r\nSchritte für Suche: %d\r\n", \
        ActIntensity.Intensity, step+step1);

// Test, ob neues Intensitätsmaximum erreicht wurde
if ( ActIntensity.Intensity > MaxIntensity.Intensity )
{
  // Falls ja: neue MaximalIntensität merken
  MaxIntensity = ActIntensity;
  // Infos fürs Statusfenster
  sprintf(buf1," NEUES MAXIMUM GEFUNDEN\r\n");
  strcat(buf,buf1);
}
// wenn Abbruch-Kriterium Intensitätsdifferenz existiert
else if ( bIntensityDiffActive )
{
  // Test, ob akt. Intensität kleiner als letzte Intensität
  // dazu: Liegt Abweichung der letzten Intensität (LastIntensity)
  // von akt. Intensität (ActIntensity) außerhalb des Limits?
  if((LastIntensity.Intensity-fIntenseDifferenz)>ActIntensity.Intensity)
  {
    // Falls ja: Zurücksetzen der Information der aktuellen Intensität
    ActIntensity = LastIntensity;
    // Test, ob absoluter Maximalwert schon früher erreicht wurde
    if ( LastIntensity.Intensity < MaxIntensity.Intensity )
    {
      // Falls ja: alle Int.Variablen auf max. Intensität setzen
      LastIntensity = MaxIntensity;
      ActIntensity = MaxIntensity;
    }

    // Zurückfahren der Antriebe an Position mit max. Intensität
    mlSetAxis(nMotorTL);
    mMoveToDistance(LastIntensity.TL);
    while( !mIsMoveFinish() );

    mlSetAxis(nMotorDF);
    mMoveToDistance(LastIntensity.DF);
    while ( !mIsMoveFinish() );

    mlSetAxis(nMotorCC);
    mMoveToDistance(LastIntensity.CC);
    while ( !mIsMoveFinish() );

    // Abbruchmeldung im Status-Textfeld & Logfile ausgeben
    sprintf(buf1, "\r\nLetzter Intensitätswert zu niedrig!\
            \r\nAbbruch und Rückkehr zur Maximalpos.\
            \r\nReale Positionen:\r\nTL: %.2f  DF: %.2f  CC: %.2f\
            \r\nIntensität: %.0f\r\n",
            LastIntensity.TL,LastIntensity.DF,
            LastIntensity.CC,LastIntensity.Intensity);
    strcat(buf,buf1);
    strcat(status, buf);
    SetDlgItemText(GetHandle(), ID_Status, status);

    // ins Logfile schreiben
    if (bWriteLogfile) WriteToJustageLog(buf);

    // Ermittlung der Anzahl der Textzeilen im Status-Textfeld
    dwStatusZeilen = SendMessage(GetDlgItem(hwnd,ID_Status), \
                                 EM_GETLINECOUNT,0,0L);
```

```
      // Scrollen des Textes um dwStatusZeilen minus Anzahl neuer Zeilen
      SendMessage(GetDlgItem(hwnd,ID_Status),EM_LINESCROLL,0, \
                  dwStatusZeilen-18L);

      // Abbruch des Justagevorgangs (Verlassen der while-Schleife)
      FORWARD_WM_COMMAND(GetHandle(),cm_post_processing,0,0,PostMessage);
      break;
    }

    // Ansonsten: die alte Intensität mit aktueller überschreiben
    LastIntensity = ActIntensity;

  }//Ende von Flagtest bIntensityDiffActive

  // Ausgabe der erreichten Intensität und ggf. neues Maximum
  strcat(status, buf);
  SetDlgItemText(GetHandle(), ID_Status, status);
  // # Logging #
  if (bWriteLogfile) WriteToJustageLog(buf);

  // Ermittlung der Anzahl der Textzeilen des Status-Textfeldes
  dwStatusZeilen = SendMessage(GetDlgItem(hwnd,ID_Status), \
                               EM_GETLINECOUNT,0,0L);
  // Scrollen des Textes um dwStatusZeilen Zeilen - Anzahl des neuen Textes
  SendMessage(GetDlgItem(hwnd, ID_Status), EM_LINESCROLL, 0, \
              dwStatusZeilen-20L);

  // Durchführung der Koordinatentransformation
  Transform->PosVektor.get_XYZ(x2, y2, z2);

  // # Logging #
  sprintf(buf," PosVektor vor KS-Drehung (%.2f,%.2f,%.2f)\n", x2, y2, z2);

  if (count % 2 == 1)
    //alle ungeraden Durchgänge KS-Drehung X-Achse
    Transform->KoordinatenTransformation(Drehachse_X, \
                                         Transform->PosVektor);
  else
    //alle geraden Durchgänge KS-Drehung um Z-Achse
    Transform->KoordinatenTransformation(Drehachse_Z, \
                                         Transform->PosVektor);
  Transform->PosVektor.get_XYZ(x, y, z);
  sprintf(buf1," neuer PositionsVektor (%.2f,%.2f,%.2f)\n",x,y,z);
  strcat (buf,buf1);
  // ins Logfile schreiben
  if (bWriteLogfile) WriteToJustageLog(buf);

  // Protokollierung der Koordinatensystem-Transformation
  Transform->PosVektor =  \
      Transform->translate_to_worldpoints(Transform->PosVektor);
  Transform->PosVektor.get_XYZ(x2, y2, z2);
  sprintf(buf," PosVektor in Weltkoord. (%.2f,%.2f,%.2f)\n", x2, y2, z2);
  Transform->PosVektor =  \
      Transform->translate_from_worldpoints(Transform->PosVektor);
  Transform->PosVektor.get_XYZ(x2, y2, z2);
  sprintf(buf1," PosVektor im letzten KS (%.2f,%.2f,%.2f)\n", x2, y2, z2);
  strcat (buf,buf1);
  // ins Logfile schreiben
  if (bWriteLogfile) WriteToJustageLog(buf);

  FORWARD_WM_COMMAND(GetHandle(),cm_choose_axis,0,0,PostMessage);
break;
```

```
case cm_post_processing:
  // Statusfenster auslesen
  GetDlgItemText(GetHandle(), ID_Status, status, nMaxString);

  // Ende der Justage und Test ob eine Verbesserung gegenüber dem
  // vor der Justage eingestellten Wert erreicht wurde
  // aktueller Intensitätswert < absolutes Maximum ?
  // Toleranz 1 Prozent (kleine Messungenauigkeiten ausbügeln)
  if (ActIntensity.Intensity+(0.01*MaxIntensity.Intensity) \
      < MaxIntensity.Intensity)
  {
    // falls ja: Zurückfahren der Antriebe an Position mit max. Intensity
    mlSetAxis(nMotorTL);
    mMoveToDistance(MaxIntensity.TL);
    while(!mIsMoveFinish());
    mlSetAxis(nMotorDF);
    mMoveToDistance(MaxIntensity.DF);
    while (!mIsMoveFinish());
    mlSetAxis(nMotorCC);
    mMoveToDistance(MaxIntensity.CC);
    while (!mIsMoveFinish());

    // Infos ins Statusfenster
    sprintf(buf,"\r\nRückkehr zur Maximumposition...\
    \r\nReale Positionen:\r\nTL: %.2f  DF: %.2f  CC: %.2f\
    \r\nIntensität: %.0f\r\n",
    MaxIntensity.TL,MaxIntensity.DF,MaxIntensity.CC,MaxIntensity.Intensity);
    strcat(status, buf);
    // ins Logfile schreiben
    if (bWriteLogfile) WriteToJustageLog(buf);
  }

  // Test auf Intensitätsüberschreitung des Detektors
  if (!Transform->bIntensityTooHigh)
  {
    // falls kein Überlauf stattfand:
    // Nachregelung für Achse DF

    sprintf(buf, "\r\nIntensitätskorrektur für Antrieb DF...\r\n");
    strcat(status, buf);
    // ins Statusfenster & Logfile schreiben
    SetDlgItemText(GetHandle(), ID_Status, status);
    if (bWriteLogfile) WriteToJustageLog(buf);

    Transform->DFCorrection(nMeasureCount, MaxIntensity.DF, \
                            MaxIntensity.Intensity);

    sprintf(buf, "Korrektur: DF:%.2f Intensität:%.0f\r\n\r\n", \
                 MaxIntensity.DF, MaxIntensity.Intensity);
  }
  else
    sprintf(buf, "\r\nJustage wurde wegen Überschreitung \
            \r\nder maximal zulässigen Detektorintensität \
            \r\nabgebrochen. Bitte vermindern Sie \
            \r\ndie Spannung des Röntgenstrahlgenerators.\r\n\r\n");

  // Berechnung der Justagedauer
  getdate(&datum);
  gettime(&zeit);
  justageende=3600*zeit.ti_hour+60*zeit.ti_min+zeit.ti_sec;
```

```
  justagezeit=justageende-justageanfang;
  sprintf(buf1,"Dauer der Justage: %02d:%02d.%02d\r\n",justagezeit/3600, \
          (justagezeit%3600)/60,(justagezeit%3600)%60);
  strcat(buf, buf1);
  strcat(status,buf);
  // Ins Statusfenster schreiben
  SetDlgItemText(GetHandle(), ID_Status, status);
  // geschriebene Zeilen im Statusfenster ermitteln
  dwStatusZeilen = SendMessage(GetDlgItem(hwnd,ID_Status), \
                               EM_GETLINECOUNT,0,0L);
  // entprechend den Scrollbar einstellen, dass das Textende zu sehen ist
  SendMessage(GetDlgItem(hwnd,ID_Status), EM_LINESCROLL, 0, \
              dwStatusZeilen-18L);

  // Logdaten ins Logfile schreiben
  if (bWriteLogfile)
  {
    sprintf(buf1,"############################################################\
    \nAutomatische Justage beendet: %02i.%02i.%i %2d:%02d:%02d\
    \n############################################################\r\n",
    datum.da_day,datum.da_mon,datum.da_year,zeit.ti_hour,
    zeit.ti_min,zeit.ti_sec);
    // erst noch die Dauer rausschreiben
    strcat(buf,buf1);
    WriteToJustageLog(buf);
  }

  // Transformationsobjekt zerstören
  delete Transform;

  SetInfo("Automatische Justage abgeschlossen ...");

  // Start-Button aktivieren
  EnableWindow(GetDlgItem(hwnd,cm_start_justage),TRUE);

  // Maus-Sanduhr in Zeiger umwandeln
  SetCursor(LoadCursor(NULL,IDC_ARROW));
  // Justage beendet
break;

case ID_Help:
  // Aufruf der Hilfe
  WinHelp(hwnd,"SPHELP.HLP",HELP_CONTEXT,Help_AutomaticAngleControlDlg);
break;

case ID_Logfile:
  // Setzen o. Löschen der Auswahlbox für die Protokollierung
  bWriteLogfile=IsDlgButtonChecked(GetHandle(), ID_Logfile);
break;

case ID_ActivateIntenseDiff:
  // (De)Aktivierung des Parameters Max. Intensitätsdifferenz
  bIntensityDiffActive=IsDlgButtonChecked(GetHandle(), \
                                          ID_ActivateIntenseDiff);

  // (De)Aktivierung der entsprechenden Dialogelemente
  if (bIntensityDiffActive)
  {
    EnableWindow(GetDlgItem(hwnd,ID_IntenseDifferenz),TRUE);
    EnableWindow(GetDlgItem(hwnd,ID_TextMaxIntDiff),TRUE);
  }
```

```
      else
      {
        EnableWindow(GetDlgItem(hwnd,ID_IntenseDifferenz),FALSE);
        EnableWindow(GetDlgItem(hwnd,ID_TextMaxIntDiff),FALSE);
      }
    break;

    default:
      // alle weiteren Nachrichten werden von der Default-Methode verarbeitet
      TModalDlg::Dlg_OnCommand(hwnd,id,hwndCtl,codeNotify);

  } // Ende der switch-Anweisung

}


BOOL TAutomaticAngleControl::LeaveDialog(void)
{
  // Detektor initialisieren
  Sensor->MeasureStop();
  Sensor->PopSettings();
  Sensor->MeasureStart();

  // Motoren anhalten
  mStopDrive(TRUE);

  // Speicher für Puffer freigeben
  delete[] status;
  delete[] buf;
  delete[] buf1;

  return TRUE;
}
```

C.2 Algorithmus und Kernfunktionalität: TransformationClass

C.2.1 transfrm.h

```
#ifndef __TRANSFRM_H
#define __TRANSFRM_H

#include "matrix.h"
#include "m_justag.h"

// Definition von Konstanten zur Festlegung der Transformationsreihenfolge
const unsigned XYZ=REIHENFOLGE_XYZ;
const unsigned ZYX=REIHENFOLGE_ZYX;


// Aufzählungstyp zur komfortablen Angabe der Koordinatensystemdrehachse
// bei Benutzung wird angegeben, um welche Achse eine KS-Drehung
// durchgeführt werden soll
typedef enum
{
  Drehachse_X=1,
  Drehachse_Y,
  Drehachse_Z
} TDrehachse;

// Aufzählungstyp zur komfortablen Angabe der Optimierungsachse
// bei Benutzung wird angegeben, auf welcher Achse der
// Optimierungsalgorithmus das Maximum der Intensität suchen soll
typedef enum
{
  Laufachse_X=1,
  Laufachse_Y,
  Laufachse_Z
} TLaufachse;

// Struktur, die die Positionen der Motoren der
// bestimmten Antriebsachsen aufnimmt (OrigDF/TL/CC)
// zusätzlich werden die Intervallschranken, innerhalb
// derer die Optimierung durchgeführt wird, gespeichert
// (Min/MaxDF/TL/CC)
typedef struct
{
  double MinDF; double MaxDF; double OrigDF;
  double MinTL; double MaxTL; double OrigTL;
  double MinCC; double MaxCC; double OrigCC;
} TMotorPositionsWerte;

// Struktur, in der die Intensität (Intensity)
// an einer bestimmten Position (DF,TL,CC) gespeichert wird
typedef struct
{
  double DF; double TL; double CC;
  float Intensity;
} TIntensityPosition;
```

```
// Transformationsklasse
// zur Durchführung von Koordinatentransformationen
class TransformationClass
{
private:

// Listen zur Abspeicherung der Transformationsmatrizen
TMatrizenListe trafo_hin;
TMatrizenListe trafo_rueck;

// Zähler für durchgeführte KS-Transformationen
unsigned anzahl_koordinatentrafos;

// Ursprungsmotorpositionen und -intervalle
TMotorPositionsWerte Wertebereich;

// Zeichenpuffer für Protokollierung
char *buf, *buf1;

public:
TVektor PosVektor;
float toleranz;

TransformationClass(void);
// Kontruktor

~TransformationClass(void);
// Destruktor

bool Initialize(double MaxDF,double MaxTL,double MaxCC);
// Initialisierung der Transformationsklasse
// 1. Ermittlung der aktuellen Motorpositionen und
// 2. Festlegen der Intervallgrenzen zur Justage

TVektor translate_to_worldpoints(TVektor OrigVektor);
// Berechnung der Weltkoordinaten eines Punktes im akt. KS
// 1. vektor homogen machen
// 2. (HIN-Transformation)
//    Multiplikation des homog. Vektors mit den Trafo-Matrizen von Anfang an
// 3. Rückgabe des Vektors in kartesischer Form

TVektor translate_from_worldpoints(TVektor OrigVektor);
// Umrechnung eines Punktes im Welt-KS in Koordinaten im akt. KS
// 1. vektor homogen machen
// 2. (RÜCK-Transformation)
//    Multiplikation des homog. Vektors mit den Trafo-Matrizen
//    vom Ende der Liste bis zum Anfang
// 3. Rückgabe des Vektors in kartesischer Form

TMotorPositionsWerte translate_PosBorders(void);
// Übersetzung der Intervalle in akt. Koordinaten
// unbedingt mit Vergleich der Softwareschranken !!!
// d.h. notwendige Rückrechnung (translate_to_worldpoints)
// der akt. Schranken in Weltkoordinaten
// aktSchranke(x,y,z) -> weltkoord_aktSchranke(x_w,y_w,z_w)
// x_w-Werte mit Wertebereich.Min/MaxTL bzw. Softwareschranke vergl. usw.
```

```
float GetIntensityOnPosition(unsigned nMeasureCount);
// fährt die zu übersetzenden (TRAFO Klasse) Motorpositionen des PosVektors an
// und gibt Intensität zurück(Bestimmung der durch Medianbestimmung über
// mehreren Messungen mgl. durch Übergabe von nMeasureCount)
// muss alle 3 Motorachsen anfahren
// 1. translate_to_worldpoints(akt.anzufahrenderPosVektor)
// 2. worldkoordinaten-Vektor.get_XYZ(&x,&y,&z)
// 3. nMotorTL mMoveToDistance (x + Wertebereich.OrigTL) weil Ursprung (0,0,0)
//    nMotorDF mMoveToDistance (y + Wertebereich.OrigDF)
//    nMotorCC mMoveToDistance (z + Wertebereich.OrigCC)
// 4. Rückgabe der Intensität

int Goldener_Schnitt(TLaufachse,double &intervall_li,double &intervall_re,
                     unsigned nMeasureCount);
// bekommt aktuelle Koordinaten (Grenzen, akt. PosVektor)
// und rechnet nur im akt. KS für eine bestimmte Achse (x,y,z)
// erst der Aufruf von GetIntensityOnPosition(vektor) führt zu einer
// Bestimmung der realen Positionen

TMotorPositionsWerte GetOrigPosBorders(void) { return Wertebereich; }
// gibt die Wertebereiche der Motoren zurück

bool KoordinatenTransformation(TDrehachse, TVektor vVerschiebung);
// 1. Berechnung des Winkels von vektor zur Dreh-Achse (Einheitsvektor)
// 2. Aufruf der transformiere-Methode der Matrix-Klasse
//    HIN : transformiere(XYZ, ... )
//    RÜCK: transformiere(ZYX, ... )
// 3. Matrizenliste.push(hin)
//    Matrizenliste.push(rück)
// 4. anzahl_koordinatentrafos ++

unsigned get_koordinatentrafos(void) { return anzahl_koordinatentrafos; }
// gibt die Anzahl der KS-Transformationen zurück

float MeasureIntensity(unsigned nMeasureCount);
// sicheres Auslesen der Intensität vom Detektor
// Möglichkeit einer Mehrfachmessung mit Medianbestimmung, um
// Messschwankungen auszugleichen


void DFCorrection(unsigned nMeasureCount, double & fDFPos, float & fIntensity);
// Korrektur für die Motorachse "Beugung Fein"
// Grund: Wenn die Position der maximalen Intensität am Ende der Suche wieder
//        angefahren wird, dann ist die Intensität nicht mehr die gleiche
//        die gemessen wurde. Die Position ist leicht verschoben.
// Lösung: DFCorrection soll in der näheren Umgebung das Maximum wieder finden

};


#endif
```

C.2.2 transfrm.cpp

```
#include <math.h>
#include "rc_def.h"
#include "comhead.h"
#include "m_devcom.h"
#include "m_layer.h"

// Module für die automatische Justage
#include "matrix.h"
#include "m_justag.h"
#include "transfrm.h"

#pragma hdrstop

extern LPDList lpDList;

//! definiert in m_justage.cpp
extern int nMotorDF, nMotorTL, nMotorCC;
extern bool bWriteLogfile;

const double GOLD = 0.61803;

// ***************************************************************************
// ***************** Transformationsklasse *********************************
// zur Durchführung von Koordinatentransformationen


// Kontruktor
TransformationClass::TransformationClass(void)
{
  TVektor tempVektor(0,0,0);
  PosVektor = tempVektor;
  anzahl_koordinatentrafos = 0;
  bIntensityTooHigh = false;

  try
  {
    // Speicher für temporaere Puffer reservieren
    buf = new char[1024];
    buf1= new char[1024];
  }
  catch (xalloc)
  {
    MessageBox(GetFocus(), "TransformationClass::GetIntensityOnPosition: \
      Fehler bei Speicherreservierung", "xalloc", MB_OK);
    PostQuitMessage(-1);
  }

}

// Destruktor
TransformationClass::~TransformationClass(void)
{
  delete[] buf;
  delete[] buf1;
}

// Initialisierung der Intervallgrenzen der Motoren
bool TransformationClass::Initialize(double MaxDF,double MaxTL,double MaxCC)
{
  double fPosition, fTemp;
  // Ermittlung der aktuellen Motorpositionen und
```

```
  // Festlegen der Intervallgrenzen zur Justage
  // Antrieb:Tilt
  // setzen der aktuellen Antriebsachse
  mlSetAxis(nMotorTL);
  // aktuelle Position des Antriebs bestimmen
  mGetDistance(fPosition);
  Wertebereich.OrigTL = fPosition;
  // Tilt: Wertebereich +/- 20 Minuten
  Wertebereich.MinTL = (-1)*MaxTL;
  Wertebereich.MaxTL = MaxTL;
  // Antrieb:Beugung fein
  mlSetAxis(nMotorDF);
  mGetDistance(fPosition);
  Wertebereich.OrigDF = fPosition;
  // DF: Wertebereich +/- 200 Sekunden
  Wertebereich.MinDF = (-1)*MaxDF;
  Wertebereich.MaxDF = MaxDF;
  // Antrieb:Kollimator
  mlSetAxis(nMotorCC);
  mGetDistance(fPosition);
  Wertebereich.OrigCC = fPosition;
  // CC: Wertebereich +/- 100 Mikrometer
  Wertebereich.MinCC = (-1)*MaxCC;
  Wertebereich.MaxCC = MaxCC;
  // Vergleich mit Softwareschranken und ggf. Änderung der Wertebereiche
  if ((fTemp=mlGetValue(nMotorTL,MinDistance)) > Wertebereich.MinTL +
                                                 Wertebereich.OrigTL)
  { Wertebereich.MinTL = fTemp - Wertebereich.OrigTL; }
  if ((fTemp=mlGetValue(nMotorTL,MaxDistance)) < Wertebereich.MaxTL +
                                                 Wertebereich.OrigTL)
  { Wertebereich.MaxTL = fTemp - Wertebereich.OrigTL; }
  if ((fTemp=mlGetValue(nMotorDF,MinDistance)) > Wertebereich.MinDF +
                                                 Wertebereich.OrigDF)
  { Wertebereich.MinDF = fTemp - Wertebereich.OrigDF; }
  if ((fTemp=mlGetValue(nMotorDF,MaxDistance)) < Wertebereich.MaxDF +
                                                 Wertebereich.OrigDF)
  { Wertebereich.MaxDF = fTemp - Wertebereich.OrigDF; }
  if ((fTemp=mlGetValue(nMotorCC,MinDistance)) > Wertebereich.MinCC +
                                                 Wertebereich.OrigCC)
  { Wertebereich.MinCC = fTemp - Wertebereich.OrigCC; }
  if ((fTemp=mlGetValue(nMotorCC,MaxDistance)) < Wertebereich.MaxCC +
                                                 Wertebereich.OrigCC)
  { Wertebereich.MaxCC = fTemp - Wertebereich.OrigCC; }
  return true;
}

// Berechnung der Weltkoordinaten eines Punktes im akt. KS
TVektor TransformationClass::translate_from_worldpoints(TVektor OrigVektor)
{
  // 1. vektor homogen machen
  OrigVektor.mache_homogen();

  // 2. (HIN-Transformation)
  // Multiplikation des homog. Vektors mit den Trafo-Matrizen von Anfang an
  // (1. bis letzte Trafo-Matrix durchlaufen)
  for (unsigned i=1; i <= trafo_hin.elementanzahl(); i++)
  {
    OrigVektor = trafo_hin.zeige(i) * OrigVektor;
  }
  // 3. Rückgabe des Vektors in kartesischer Form
  return OrigVektor.mache_kartesisch();
}
```

```
// Umrechnung eines Punktes im Welt-KS in Koordinaten im akt. KS
TVektor TransformationClass::translate_to_worldpoints(TVektor OrigVektor)
{
  // 1. vektor homogen machen
  OrigVektor.mache_homogen();

  // 2. (RÜCK-Transformation)
  //    Multiplikation des homog. Vektors mit den Trafo-Matrizen
  //    vom Ende der Liste bis zum Anfang
  for (unsigned i=trafo_rueck.elementanzahl(); i > 0; i--)
  {
    OrigVektor = trafo_rueck.zeige(i) * OrigVektor;
  }

  // 3. Rückgabe des Vektors in kartesischer Form
  return OrigVektor.mache_kartesisch();
}


// Übersetzung der Originalintervalle in akt. Koordinaten
TMotorPositionsWerte TransformationClass::translate_PosBorders(void)
{
  TVektor Punkt3D(3);
  TMotorPositionsWerte Intervalle;
  double X,Y,Z;

  // Übersetzung der Intervalle in akt. Koordinaten
  // unbedingt mit Vergleich der Softwareschranken !!!
  // d.h. notwendige Rückrechnung (translate_to_worldpoints)
  // der akt. Schranken in Weltkoordinaten
  // aktSchranke(x,y,z) -> weltkoord_aktSchranke(x_w,y_w,z_w)
  // x_w-Werte mit Wertebereich.Min/MaxTL bzw. Softwareschranke vergl. usw.

  // Intervallgrenzen aus Wertebereich für TL umrechnen
  Punkt3D.set_XYZ(Wertebereich.MinTL, 0, 0);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(Intervalle.MinTL, Y, Z);
  Punkt3D.set_XYZ(Wertebereich.MaxTL, 0, 0);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(Intervalle.MaxTL, Y, Z);

  // Intervallgrenzen für DF umrechnen
  Punkt3D.set_XYZ(0, Wertebereich.MinDF, 0);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(X, Intervalle.MinDF, Z);
  Punkt3D.set_XYZ(0, Wertebereich.MaxDF, 0);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(X, Intervalle.MaxDF, Z);

  // Intervallgrenzen für TL umrechnen
  Punkt3D.set_XYZ(0, 0, Wertebereich.MinCC);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(X, Y, Intervalle.MinCC);
  Punkt3D.set_XYZ(0, 0, Wertebereich.MaxCC);
  Punkt3D = translate_from_worldpoints(Punkt3D);
  Punkt3D.get_XYZ(X, Y, Intervalle.MaxCC);

  return Intervalle;
}
```

```
float TransformationClass::GetIntensityOnPosition(unsigned nMeasureCount)
// fährt die zu übersetzenden (TRAFO Klasse) Motorpositionen des Vektors an
// und gibt Intensität zurück
// muss alle 3 Motorachsen anfahren
// 1. translate_to_worldpoints(akt.anzufahrenderPosVektor)
// 2. worldkoordinaten-Vektor.get_XYZ(&x,&y,&z)
// 3. nMotorTL mMoveToDistance (x + Wertebereich.OrigTL) weil Ursprung (0,0,0)
//    nMotorDF mMoveToDistance (y + Wertebereich.OrigDF)
//    nMotorCC mMoveToDistance (z + Wertebereich.OrigCC)
// 4. Rückgabe der Intensität
{
  float fIntensity;
  double x,y,z,x2,y2,z2;

  // Infos fürs Logging aufbereiten
  PosVektor.get_XYZ(x2, y2, z2);
  sprintf(buf,"Move2Dist:aktVektor (%.2f,%.2f,%.2f) -> ",x2,y2,z2);

  // Umrechnung des PosVektors in Weltkoordinaten
  PosVektor = translate_to_worldpoints(PosVektor);

  // Infos fürs Logging aufbereiten
  PosVektor.get_XYZ(x, y, z);
  sprintf(buf1,"weltVektor (%.2f,%.2f,%.2f)[%d.KS]",x,y,z,
               get_koordinatentrafos());
  strcat (buf,buf1);
  // ins Logfile schreiben
  if (bWriteLogfile) WriteToJustageLog(buf);

  // Auslesen der Weltkoordinaten des PosVektors
  PosVektor.get_XYZ(x, y, z);

  // setzen der Antriebsachse TL und Anfahren der TL-Position
  mlSetAxis(nMotorTL);
  mMoveToDistance (x + GetOrigPosBorders().OrigTL);
  // setzen der Antriebsachse DF und Anfahren der DF-Position
  while (!mIsMoveFinish());
  mlSetAxis(nMotorDF);
  mMoveToDistance (y + GetOrigPosBorders().OrigDF);
  // setzen der Antriebsachse CC und Anfahren der CC-Position
  while (!mIsMoveFinish());
  mlSetAxis(nMotorCC);
  mMoveToDistance (z + GetOrigPosBorders().OrigCC);

  // Warten bis Bewegung beendet ist
  while (!mIsMoveFinish());

  fIntensity = MeasureIntensity(nMeasureCount);

  // Ausgabe in Statusleiste
  sprintf(buf,"I: %.0f", fIntensity);
  SetInfo(buf);

  // ins Logfile schreiben
  strcat(buf, "\n");
  if (bWriteLogfile) WriteToJustageLog(buf);
  // Werte für den PosVektor zurückschreiben
  PosVektor.set_XYZ(x2, y2, z2);
  return fIntensity;
}
```

```
int TransformationClass::Goldener_Schnitt(TLaufachse achse,
                    double & intervall_li, double & intervall_re,
                    unsigned nMeasureCount)
{
  double a,b,u,v,fu,fv, x, y, z;
  int step;

  a=intervall_li;
  b=intervall_re;

  u=GOLD*a+(1-GOLD)*b;
  v=(1-GOLD)*a+GOLD*b;

  // Komponenten des Positionsvektors auslesen
  PosVektor.get_XYZ(x,y,z);

  // # Logging #
  sprintf(buf,">>GoldenerSchnitt: PosVektor (%.2f,%.2f,%.2f)\n",x,y,z);
  if (bWriteLogfile) WriteToJustageLog(buf);

  switch (achse)
  {
    case Laufachse_X: // Setzen der jew. Komponente im PosVektor
                      PosVektor.set_XYZ(u,y,z);
                      // Bestimmung der Intensität
                      fu=GetIntensityOnPosition(nMeasureCount);
                      PosVektor.set_XYZ(v,y,z);
                      fv=GetIntensityOnPosition(nMeasureCount);
                      break;
    case Laufachse_Y: PosVektor.set_XYZ(x,u,z);
                      fu=GetIntensityOnPosition(nMeasureCount);
                      PosVektor.set_XYZ(x,v,z);
                      fv=GetIntensityOnPosition(nMeasureCount);
                      break;
    case Laufachse_Z: PosVektor.set_XYZ(x,y,u);
                      fu=GetIntensityOnPosition(nMeasureCount);
                      PosVektor.set_XYZ(x,y,v);
                      fv=GetIntensityOnPosition(nMeasureCount);
                      break;
    default:
    return -1;
  }

  step=2;

  while ( (fabs(a-b) > toleranz) )
  {
    if (fu > fv)
    {
      b=v; // re. Intervallgrenze nach links verschieben
      v=u;
      u=GOLD*a+(1-GOLD)*b; // neue li. Intervall-Position
      fv=fu;
      // Bestimmung des neuen Funktionswertes  (Intensität)
      switch (achse)
      {
        case Laufachse_X: PosVektor.set_XYZ(u,y,z);
                          fu=GetIntensityOnPosition(nMeasureCount);
                          break;
        case Laufachse_Y: PosVektor.set_XYZ(x,u,z);
                          fu=GetIntensityOnPosition(nMeasureCount);
                          break;
```

```
        case Laufachse_Z: PosVektor.set_XYZ(x,y,u);
                          fu=GetIntensityOnPosition(nMeasureCount);
                          break;
       }
      step++;
    }
    else // wenn (fu <= fv)
    {
      a=u; // li. Intervallgrenze nach rechts verschieben
      u=v;
      v=(1-GOLD)*a+GOLD*b; // neue re. Intervall-Position
      fu=fv;
      // Bestimmung des neuen Funktionswertes (Intensität)
      switch (achse)
      {
        case Laufachse_X: PosVektor.set_XYZ(v,y,z);
                          fv=GetIntensityOnPosition(nMeasureCount);
                          break;
        case Laufachse_Y: PosVektor.set_XYZ(x,v,z);
                          fv=GetIntensityOnPosition(nMeasureCount);
                          break;
        case Laufachse_Z: PosVektor.set_XYZ(x,y,v);
                          fv=GetIntensityOnPosition(nMeasureCount);
                          break;
       }
      step++;
    }
  }
  intervall_li=u;
  intervall_re=v;
  return step;
}

float TransformationClass::MeasureIntensity(unsigned nMeasureCount)
{
  float fIntensity, fIntensity_temp, ftemp, fMedian;
  unsigned counter=0;
  float *Medianliste;

  try
  {
    // Speicher für Medianliste reservieren
    Medianliste = new float[nMeasureCount];
  }
  catch (xalloc)
  {
    MessageBox(GetFocus(), "MeasureIntensity: Fehler bei Speicherreservierung",
               "xalloc", MB_OK);
    PostQuitMessage(-1);
  }
   // aktuelles Zählergerät setzen
  TDevice *Detektor = lpDList->DP();
  do
  {
    // Zähler starten
    Detektor->MeasureStart();

    // Auslesen der Detektorkarte veranlassen
    // im Fehlerfall muss die Messung neugestartet werden
    while (Detektor->PollDevice() != R_MeasOk)
    {
      // Messung neustarten
```

```
  // Detektor->MeasureStop();
 SetInfo("Warte auf Messwerte...");
 MessageBeep(0);
 //Detektor->MeasureStart();
}
// Intensität auslesen (Membervariable fIntensity von Sensor)
while (Detektor->GetData(fIntensity) != R_OK)
{
  SetInfo("Hole Daten vom Counter...");
}

// erfolgreiche Messung
counter++;

// Medianliste erweitern & sortieren *****
fIntensity_temp=fIntensity;

if (fIntensity > 100000.0)
{
  if (!bIntensityTooHigh)
  {
    MessageBox(GetFocus(), "MeasureIntensity: Die Maximalanzahl von 100.000\
           \nDetektorcounts wurde überschritten.\n \
           \nDie 'Automatische Justage' wird abgebrochen. \
           \nBitte regeln Sie die Spannung des Röntgengenerators nach.\n",
           "Überschreitung des Detektormaximums", MB_OK);
    bIntensityTooHigh=true;
  }
}

// noch keine Messwerte in der Liste
if ((counter-1) == 0)
{
  Medianliste[0] = fIntensity_temp;
}
// Messwertliste besteht schon
else
{
  // solange bis alle bisherigen Messwerte durchgelaufen wurden
  for (int i=0; i < counter-1; i++)
  {
    // kleine Elemente zuerst einsortieren
    if(fIntensity_temp < Medianliste[i])
    {
      // Wert an der akt. Stelle merken
      ftemp=Medianliste[i];
      // Wert an der akt. Stelle durch Messwert ersetzen
      Medianliste[i]=fIntensity_temp;
      // Trick, um die weiteren Elemente durchzusortieren
      fIntensity_temp=ftemp;
    }
  }
  // übrig gebliebener Wert wird ans Ende angehängt
  Medianliste[counter-1]=fIntensity_temp;
}

// Medianbestimmung
// ungerade Anzahl von Messungen
if((counter%2)==1)
{
  if(counter==1)
  {
```

```
        // einziges Element
        fMedian=Medianliste[counter-1];
      }
      else
      {
        // mittleres Element der Medianliste
        fMedian=Medianliste[((counter+1)/2)-1];
      }
    }
    // gerade Anzahl von Messungen
    else
    {
      // Mittelwert der beiden mittleren Elemente
      fMedian=(Medianliste[(counter/2)-1]+Medianliste[((counter/2)+1)-1])/2;
    }
    // Infos ins Logfile schreiben
    if (bWriteLogfile)
    {
      sprintf(buf,"\r\nM:%d I:%.2f Median:%.2f ",counter,fIntensity,fMedian);
      WriteToJustageLog(buf);
    }

  } while(counter < nMeasureCount);

  delete[] Medianliste;

  return fMedian;
}

// Durchführung einer kompletten Koordinatentransformation um best. Achse
bool TransformationClass::KoordinatenTransformation(TDrehachse achse, \
                                                    TVektor VerschiebeVektor)
{
  TVektor Einheitsvektor(3);
  TMatrix Matrix;
  double winkel;

  switch (achse)
  {
    case Drehachse_X: // Einheitsvektor der Z-Achse
                      Einheitsvektor.set_XYZ(0,0,1); break;

    case Drehachse_Z: // Einheitsvektor der X-Achse
                      Einheitsvektor.set_XYZ(1,0,0); break;

    // bei Automatischer Justage wird nur um X bzw. Z gedreht
    default:
    return false;
  }

  // Berechnung des Winkels von Vektor zur Koordinatenachse (Einheitsvektor)
  winkel = VerschiebeVektor.winkel(Einheitsvektor);

  // Ausführung der jew. KS-Transformation (Hin - und Rück)
  // Drehung um X-Achse
  if (achse == Drehachse_X)
  {
    // Abspeichern der Hintransformation
    Matrix.transformiere(XYZ, (-1)*VerschiebeVektor, -winkel, 0, 0);
    trafo_hin.push(Matrix);
    // Abspeichern der Rücktransformation
    Matrix.transformiere(ZYX, VerschiebeVektor, winkel, 0, 0);
```

```
    trafo_rueck.push(Matrix);
  }
  // Drehung um Z-Achse
  else if (achse == Drehachse_Z)
  {
    // Abspeichern der Hintransformation
    Matrix.transformiere(XYZ, (-1)*VerschiebeVektor, 0, 0, -winkel);
    trafo_hin.push(Matrix);
    // Abspeichern der Rücktransformation
    Matrix.transformiere(ZYX, VerschiebeVektor, 0, 0,winkel);
    trafo_rueck.push(Matrix);
  }

  anzahl_koordinatentrafos ++;

  // # Logging #
  sprintf(buf,"%d. KS-Drehung: Winkel= %.2f Grad\n",anzahl_koordinatentrafos,
              winkel*180/M_PI);
  if (bWriteLogfile) WriteToJustageLog(buf);

  return true;
}

void TransformationClass::DFCorrection(unsigned nMeasureCount, double & fDFPos,
                                       float & fInt)
{
  double a,b,u,v,fu,fv;
  double fPosition, toleranz=.1;

  // Intervallgrösse selbst angeben
  // Bereich kann klein sein (DF: sek)
  a=-5;
  b=5;

  u=GOLD*a+(1-GOLD)*b;
  v=(1-GOLD)*a+GOLD*b;

  // Position festlegen
  mlSetAxis(nMotorDF);
  mGetDistance(fPosition);

  mMoveToDistance(u+fPosition);
  while (!mIsMoveFinish());
  fu=MeasureIntensity(nMeasureCount);
  mMoveToDistance(v+fPosition);
  while (!mIsMoveFinish());
  fv=MeasureIntensity(nMeasureCount);

  while ((fabs(a-b) > toleranz))
  {
    if (fu > fv)
    {
      b=v; // re. Intervallgrenze nach links verschieben
      v=u;
      u=GOLD*a+(1-GOLD)*b; // neue li. Intervall-Position
      fv=fu;
      // Bestimmung des neuen Funktionswertes (Intensität)
      mMoveToDistance(u+fPosition);
      while (!mIsMoveFinish());
      fu=MeasureIntensity(nMeasureCount);
    }
    else // if (fu <= fv)
```

```
    {
      a=u; // li. Intervallgrenze nach rechts verschieben
      u=v;
      v=(1-GOLD)*a+GOLD*b; // neue re. Intervall-Position
      fu=fv;
      // Bestimmung des neuen Funktionswertes (Intensität)
      mMoveToDistance(v+fPosition);
      while (!mIsMoveFinish());
      fv=MeasureIntensity(nMeasureCount);
    }
  }
  // Intensität & Position zurückliefern
  fInt=fv;
  fDFPos=v+fPosition;
  // Position angefahren
}
```

C.3 Mathematische Hilfsklassen

C.3.1 matrix.h

```
// matrix.h - definiert die Matrix-, Vektor- und MatrizenListe-Klassen.
// Dabei enthält das Array alle Elemente der Matrix. Das wird erreicht, indem
// die Zeilen der Matrix hindereinander gespeichert werden, d.h. Zeile 1,
// Zeile 2, ..., Zeile n.

#ifndef __MATRIX_H
#define __MATRIX_H


// Die Matrix-Bibliothek wurde für Borland C++ 5.0, das den
// C++-Standardtyp bool für Boolean-Variablen unterstützt, entwickelt.
// Um die Bibliothek aber unter Borland C++ 4.5 übersetzen zu können,
// müssen die folgenden include/define Zeilen eingeschoben werden
// (in windows.h ist der Win-API-Typ BOOL definiert):
#include <windows.h>

#define bool  BOOL
#define false FALSE
#define true  TRUE


// Konstanten fuer Reihenfolge der Transformationen
const unsigned REIHENFOLGE_XYZ = 10;
const unsigned REIHENFOLGE_ZYX = 13;

class TVektor;

//****************************************************************************
// Klasse Matrix
// -------------
// mit m Zeilen und n Spalten
// es besteht die Möglichkeit, die Matrizen in homogener Schreibweise
// anzugeben (durch zusätzliche homogene Komponente)
// in diesem Fall wird die Membervariable homogene_koordinaten=true gesetzt
//****************************************************************************
class TMatrix
{
friend TVektor;

protected:
// Membervariablen zur Darstellung der Matrix
double *arr; // eindimensionales Feld zur Speicherung der Matrixelemente
unsigned ze; // Zeilen
unsigned sp; // Spalten
// Flag, das angibt, ob Matrix in homogener Schreibweise dargestellt ist
bool homogene_koordinaten;

public:
// Konstruktoren und Destruktor:
TMatrix();                       // Standardkonstruktor
TMatrix(unsigned m, unsigned n); // initialisiert eine m*n-Nullmatrix
~TMatrix();                      // Klassendestruktor
TMatrix (const TMatrix & mat);   // eigener Copy-Kontruktor
```

```
// Klassenmethoden:

// Test, ob Matrix bzw. Vektor in homogenen Koordinaten angegeben ist
bool ist_homogen(void) const
{
  return homogene_koordinaten;
}

// überladene Operatoren :
TMatrix operator = (const TMatrix & mat); // Matrizenzuweisung
TMatrix operator + (const TMatrix & mat); // Matrizenaddition
TMatrix operator - (const TMatrix & mat); // Matrizensubtraktion
TMatrix operator * (const TMatrix & mat); // Matrizenmultiplikation
TMatrix operator * (double & fakt); // reelle Vervielfachung für TMatrix*double
friend TMatrix operator * (double fakt, const TMatrix & mat); // double*TMatrix

// Berechnung der Inversen einer Matrix,
// sofern es sich um eine reguläre Matrix handelt
TMatrix invers(void);

// Erstellung einer Einheitsmatrix mit Rang m
TMatrix einheitsmatrix(unsigned m);

// **** Transformationsmethoden **************************

// Verschiebe-(Translations)TMatrix erzeugen
// Verschiebung um (Tx,Ty,Tz) Vektor
TMatrix verschiebematrix(const TVektor & vekt);

// Rotationsmatrix erzeugen
// Drehung um x-,y-, bzw. z-Achse mit Winkel
// Anmerkung: wenn Winkel negativ angegeben, dann handelt es sich um eine
//            inverse Drehung
TMatrix rotationsmatrix_x(double winkel);
TMatrix rotationsmatrix_y(double winkel);
TMatrix rotationsmatrix_z(double winkel);

// Zusammenfassung aller Transformationen in einer Funktion
// Reihenfolge: REIHENFOLGE_XYZ = 10
//              REIHENFOLGE_ZYX = 13
TMatrix transformiere(unsigned reihenfolge, const TVektor & verschiebung,
                      double drehung_x, double drehung_y, double drehung_z);
};

//*****************************************************************************
// Klasse Vektor
// -------------
// ist ein Spezialfall einer Matrizenklasse, bei dem gilt: Spaltenanzahl (sp)=1
// es besteht die Möglichkeit, die Vektorkomponenten in homogenen Koordinaten
// anzugeben (durch zusätzliche homogene Komponente)
// in diesem Fall wird die Membervariable homogene_koordinaten=true gesetzt
//*****************************************************************************
class TVektor : public TMatrix
{
public:
// Konstruktoren:

// Standardkonstruktor
TVektor(void) : TMatrix()
{
  sp = 1;
}
```

```
// Konstruktor zur Initialisierung eines Vektors
// mit m Zeilen in kartesischen Koordinaten
TVektor(unsigned m) : TMatrix(m,1)
{
}

// Konstruktor zur Initialisierung eines 3D-Vektors mit kartesischen Koordinaten
TVektor(double x, double y, double z) : TMatrix(3,1)
{
  arr[0]=x; arr[1]=y; arr[2]=z;
}

// Copy-Konstruktor, um aus einer (m,1)-Matrix einen Vektortyp zu machen
TVektor (const TMatrix & mat);

// Klassenmethoden:

// Vektor skalieren
friend TVektor operator * (double fakt, const TVektor & vekt);

// Matrix wird mit einem Vektor multipliziert und das Ergebnis ist ein Vektor
// friend TVektor operator * (const TMatrix & mat, const TVektor & vekt);

// Umwandlung in einen Vektor mit homogenen Koordinaten
TVektor mache_homogen(void);

// Umwandlung in einen Vektor mit kartesischen Koordinaten
TVektor mache_kartesisch(void);

// Berechnung des Betrags (Länge) eines Vektors
double vektor_betrag (void) const;

// Berechnung des Skalarproduktes zweier Vektoren
double skalarprodukt (const TVektor & vekt);

// Berechnung des Winkels zwischen zwei Vektoren
double winkel (const TVektor & vekt);

// Setzen der x,y,z-Koordinaten eines 3dim Vektors
bool set_XYZ(double x, double y, double z);

// Ausgabe der x,y,z-Koordinaten eines 3dim Vektors
bool get_XYZ(double & x, double & y, double & z);

};


//*****************************************************************************
// Matrizenliste-Klasse
// --------------------
// Liste von Transformationsmatrizen
// organisiert als Stack
// mit push(element), pop(), ist_leer()
//*****************************************************************************
class TMatrizenListe
{
private:
// Membervariablen
TMatrix *liste;        // Liste als Zeiger auf die Matrizen
unsigned akt_elemente; // aktuelle Anzahl von Matrixelements in der Liste
unsigned max_elemente; // maximale Anzahl von abzuspeichernden Matrizen
```

```
public:
// Konstruktoren und Destruktoren

// Listenkonstruktor um eine Liste mit einer bestimmten Anzahl von
// Listenelementen zu erzeugen
TMatrizenListe(unsigned anzahl = 10);

// Standarddestruktor
~TMatrizenListe()
{
  if (liste != NULL) delete [] liste;
}

// Klassenmethoden:

// Test, ob Matrizenliste leer ist
bool ist_leer(void)
{
  return (akt_elemente == 0);
}

// Gibt die Anzahl der Listenlemente zurück
unsigned elementanzahl(void)
{
  return akt_elemente;
}

// Matrix zur Liste hinzufügen
bool push(const TMatrix & trafo);

// letzte Matrix aus Liste entfernen
// liefert entfernte Matrix zurück
TMatrix pop(void);

// Listenelement mit Nummer position (1 .. akt_elemente) ausgeben(Direktzugriff)
// liefert die Matrix an der Stelle position zurück
TMatrix zeige(unsigned position);

};

#endif
```

C.3.2 matrix.cpp

```
#include <iostream.h>
#include <math.h>
#include "matrix.h"

//*****************************************************************************
// Matrix-Klasse
//*****************************************************************************

// *** Klassenkonstruktoren ********************************************************
// Standardkonstruktor (ohne Argumente)
TMatrix::TMatrix()
{
  arr = NULL;
  ze = sp = 0;
  homogene_koordinaten=false;
}

// Konstruktor, der eine m*n-Nullmatrix initialisiert.
TMatrix::TMatrix(unsigned m, unsigned n)
{
  ze = m; sp = n;
  homogene_koordinaten=false;
  arr = new double[m*n]; // dynamisches Erzeugen des Matrixarrays
  for (unsigned i=0; i < m*n; i++) arr[i] = 0; // Array mit Nullen belegen
}

// Klassendestruktor löscht das dynamische Array
TMatrix::~TMatrix()
{
  if (arr != NULL) delete [] arr;
}

// überladener copy-Kontruktor (wird nur bei Initialisierung aufgerufen)
TMatrix::TMatrix (const TMatrix & mat)
{
  ze = mat.ze;
  sp = mat.sp;
  homogene_koordinaten=mat.homogene_koordinaten;
  arr = new double[mat.ze*mat.sp];
  for (unsigned i=0; i < mat.ze * mat.sp; i++) arr[i] = mat.arr[i];
}

// *** überladene Operatoren ***************************************************
// überladener = Operator : Zuweisungsoperator
TMatrix TMatrix::operator = (const TMatrix & mat)
{
  if (this == &mat) return *this; // *this zurückgeben, wenn Selbstzuweisung

  if (arr != NULL) delete [] arr; // Speicherplatz des Matrix-Arrays freigeben
  ze = mat.ze;
  sp = mat.sp;
  homogene_koordinaten= mat.homogene_koordinaten;
  arr = new double[mat.ze*mat.sp];
  for (unsigned i=0; i < mat.ze * mat.sp; i++) arr[i] = mat.arr[i];
  return *this;
}
```

```
// überladener + Operator: addiert zwei Matrizen gleichen Typs
TMatrix TMatrix::operator + (const TMatrix & mat)
{
  if (ze != mat.ze || sp != mat.sp)
  {
    cerr << "(M.+)Fehler: Matrizen versch.Typs können nicht addiert werden.\n";
    return *this;
  }
  else
  {
  for (unsigned i=0; i < ze*sp; i++)
    arr[i] += mat.arr[i];
  }
  homogene_koordinaten &= mat.homogene_koordinaten;
  return *this;
}

// überladener - Operator: subtrahiert zwei Matrizen gleichen Typs
TMatrix TMatrix::operator - (const TMatrix & mat)
{
  if (ze != mat.ze || sp != mat.sp)
  {
    cerr << "(M.-)Fehler: Kann Matrizen versch. Typs nicht subtrahieren.\n";
    return *this;
  }
  else
  {
    for (unsigned i=0; i < ze*sp; i++)
      arr[i] -= mat.arr[i];
  }
  homogene_koordinaten &= mat.homogene_koordinaten;
  return *this;
}

// überladener * Operator Nr.1: reelle Vervielfachung von Matrizen
// Version TMatrix-Objekt * double-Wert
TMatrix TMatrix::operator * (double & fakt)
{
  for (unsigned i=0; i < ze*sp; i++)
    arr[i] *= fakt;
  return *this;
}

// Version double-Wert * TMatrix-Objekt (friend-Funktion)
TMatrix operator * (double fakt, const TMatrix & mat)
{
  TMatrix temp(mat.ze, mat.sp);
  for (unsigned i=0; i < mat.ze*mat.sp; i++)
    temp.arr[i] = fakt * mat.arr[i];
  temp.homogene_koordinaten=mat.homogene_koordinaten;
  return temp;
}

// überladener * Operator Nr.2: multipliziert zwei Matrizen
TMatrix TMatrix::operator * (const TMatrix & mat)
{
  TMatrix temp(ze,mat.sp);
  if (sp != mat.ze)
  {
    cerr << "(M.*)Fehler: Spalten der 1. Matrix ungleich Zeilen der 2.\n";
    temp.ze = temp.sp = 0;
    temp.arr = NULL;
```

```
  }
  else
  {
    for (unsigned i=0; i < ze*mat.sp; i++)
      for (unsigned j=0; j < sp; j++)
        temp.arr[i] += arr[j + sp*(i/mat.sp)] * mat.arr[j*mat.sp + i % mat.sp];
  }
  temp.homogene_koordinaten=(homogene_koordinaten && mat.homogene_koordinaten);
  return temp;
}

// *** Matrizen - Funktionen ************************************************
// Erstellen einer Einheitsmatrix mit Rang m
TMatrix TMatrix::einheitsmatrix(unsigned m)
{
  if (arr != NULL) delete [] arr;

  ze = m;
  sp = m;

  arr = new double[m*m];

  for (unsigned i=0; i < ze; i++)
  {
    for (unsigned j=0; j < sp; j++)
    {
      if ( i == j) arr[i*sp + j] = 1;
      else arr[i*sp + j] = 0;
    }
  }
  return *this;
}

// Berechnung der Inversen einer Matrix, sofern sie existiert
// Verfahren von Gauss-Jordan (siehe Stoecker S.402)
TMatrix TMatrix::invers(void)
{
  double dummy;

  if ( ze != sp || ze == 0 || sp == 0)
    cerr << "(M.Inv)Fehler: Matrix kann nicht invertiert werden.\n";

  TMatrix temp;

  // Einheitsmatrix temporär erstellen
  temp.einheitsmatrix(ze);

  //Berechnung der Inversen
  for (unsigned k=0; k < ze; k++)
  {
    dummy = arr[k*sp + k];
    for (unsigned j=0; j < sp; j++)
    {
      arr[k*sp + j] = arr[k*sp +j]/dummy;
      temp.arr[k*sp + j] = temp.arr[k*sp +j]/dummy;
    }
    for (unsigned i=0; i < ze; i++)
    {
      if (i != k)
      {
        dummy = arr[i*sp + k];
```

```
        for (unsigned j=0; j < sp; j++)
        {
          arr[i*sp + j] = arr[i*sp + j]-(dummy*arr[k*sp +j]);
          temp.arr[i*sp + j] = temp.arr[i*sp + j]-(dummy*temp.arr[k*sp +j]);
        }
      }
    }
  }
  // Inverse Matrix ist nun in temp
  return temp;
}

// **** Transformationsmethoden ************************************************
// Verschiebe-(Translations)TMatrix erzeugen
// Verschiebung um (Tx,Ty,Tz) Vektor
TMatrix TMatrix::verschiebematrix(const TVektor &vekt)
{
  if ( !(vekt.ze == 4 && vekt.ist_homogen() ||
         vekt.ze == 3 && !vekt.ist_homogen()))
  {
    cerr << "(M.VM)Fehler: Verschiebungsvektor nicht  3-dimensional.\n";
    return *this;
  }
  // Einheitsmatrix 4x4 erzeugen
  this->einheitsmatrix(4);
  // Verschiebungsvektor in die letzte Spalte der Matrix eintragen
  // bei homogenem Vektor wird homogene Komponente mitkopiert (=1)
  for (unsigned i = 0; i < vekt.ze; i++)
  {
    // Zugriff auf 4. Spaltenelement der jeweiligen Zeile
    arr[i*ze+3] = vekt.arr[i];
  }
  homogene_koordinaten=true;
  return *this;
}

// Rotationsmatrix erzeugen
// Drehung um x-,y-, bzw. z-Achse mit Winkel
// Anmerkung: wenn negative Winkel, dann handelt es sich um eine inverse Drehung
TMatrix TMatrix::rotationsmatrix_x(double winkel)
{
  this->einheitsmatrix(4);
  /* Rotation um die X-Achse mit dem Winkel a
       1   0       0     0
       0 cos(a) -sin(a)  0
       0 sin(a)  cos(a)  0
       0   0       0     1
  */
  arr[1*4+1]= cos(winkel); arr[1*4+2]= -sin(winkel);
  arr[2*4+1]= sin(winkel); arr[2*4+2]=  cos(winkel);
  homogene_koordinaten=true;
  return *this;
}

TMatrix TMatrix::rotationsmatrix_y(double winkel)
{
  this->einheitsmatrix(4);
  /* Rotation um die Y-Achse mit dem Winkel a
     cos(a)  0   sin(a)  0
       0     1     0     0
    -sin(a)  0   cos(a)  0
       0     0     0     1 */
```

```
  arr[0*4+0]=  cos(winkel); arr[0*4+2]= sin(winkel);
  arr[2*4+0]= -sin(winkel); arr[2*4+2]= cos(winkel);
  homogene_koordinaten=true;
  return *this;
}

TMatrix TMatrix::rotationsmatrix_z(double winkel)
{
  this->einheitsmatrix(4);
  /* Rotation um die z-Achse mit dem Winkel a
    cos(a) -sin(a)  0   0
    sin(a)  cos(a)  0   0
      0       0     1   0
      0       0     0   1
  */
  arr[0*4+0]= cos(winkel); arr[0*4+1]= -sin(winkel);
  arr[1*4+0]= sin(winkel); arr[1*4+1]=  cos(winkel);
  homogene_koordinaten=true;
  return *this;
}

// Zusammenfassung aller Transformationen X-Y-Z & Verschiebung in einer Funktion
TMatrix TMatrix::transformiere(unsigned reihenfolge, \
                          const TVektor & verschiebung, \
                          double drehung_x, double drehung_y, double drehung_z)
{
  TMatrix tempx, tempy, tempz;

  // Nur 4dim Vektoren inkl. homogener Komponente oder reine 3dim Vektoren
  if ( !(verschiebung.ze == 4 && verschiebung.ist_homogen() || \
         verschiebung.ze == 3 && !verschiebung.ist_homogen()) )
  {
    cerr << "(M.TRAFO)Fehler: Verschiebungsvektor nicht  3-dimensional.\n";
    return *this;
  }

  // Reihenfolge der Transformationen muss angegeben sein: X->Y->Z oder Z->Y->X
  if ( reihenfolge != REIHENFOLGE_XYZ && reihenfolge != REIHENFOLGE_ZYX )
  {
    cerr << "(M.TRAFO)Fehler: Keine gültige Angabe der Trafo-Reihenfolge.\n";
    return *this;
  }
  else
  {
    // Drehungen in der 4x4 Trafo-Matrix
    // Erklärung siehe oben bei Einzelrotationen
    tempx.rotationsmatrix_x(drehung_x);
    tempy.rotationsmatrix_y(drehung_y);
    tempz.rotationsmatrix_z(drehung_z);
    // Verschiebung um Vektor
    this->verschiebematrix(verschiebung);
    if  (reihenfolge == REIHENFOLGE_ZYX)
      // aufgrund der Def. d. Matrixmultiplikation (Beginn der Mult. von rechts)
      // muss die Reihenfolge der Multiplikation umgekehrt notiert werden,
      // mathematisch gilt folgendes:
      // R_rueck = Rx*Ry*Rz*T
      *this = *this * tempz * tempy * tempx;
```

```
    else
      // aufgrund der Def. d. Matrixmultiplikation (Beginn der Mult. von rechts)
      // muss die Reihenfolge der Multiplikation umgekehrt notiert werden,
      // mathematisch gilt folgendes:
      // R_hin = T*Rz*Ry*Rx
      *this = tempx * tempy * tempz * *this;
    homogene_koordinaten=true;
  }
  return *this;
}


//****************************************************************************
// Vektor-Klasse
//****************************************************************************

// Funktion, um aus einer (m,1)-Matrix einen Vektortyp zu machen
TVektor::TVektor (const TMatrix & mat)
{

  if (mat.ze == 0 || mat.sp == 0)
    cerr << "(V)Fehler: Matrixspalten- oder Zeilenanzahl beträgt 0.\n";
  else if (mat.sp != 1)
    cerr << "(V)Fehler: Matrixspaltenanzahl != 1 Keine Umwandlung in Vektor.\n";
  else
  {
    ze=mat.ze;
    sp=1;
    homogene_koordinaten=mat.homogene_koordinaten;
    arr = new double[ze];
    for (unsigned i=0; i < ze; i++) arr[i] = mat.arr[i];
  }
}


// Vektor skalieren
TVektor operator * (double fakt, const TVektor & vekt)
{
  TVektor temp(vekt.ze);
  for (unsigned i=0; i < vekt.ze; i++)
    temp.arr[i] = fakt * vekt.arr[i];
  return temp;
}

// Umwandlung in einen Vektor in homogenen Koordinaten
TVektor TVektor::mache_homogen(void)
{
  unsigned i;

  if (ze == 0)
  {
    cerr << "(V.MH)Fehler: Vektor mit Zeilenanzahl 0.\n";
    return *this;
  }
  else if (this->ist_homogen())
  {
     cerr << "(V.MH)Fehler: Vektor schon in homog. Koord. angegeben.\n";
    return *this;
  }

  TVektor temp(ze+1); // temp-Vektor mit einer zusätzlichen homogenen Komponente
```

```
  // Kopieren des Ursprungsvektors
  for (i=0; i < ze; i++) temp.arr[i] = arr[i];
  // neue Komponente im homogenen temp-Vektor auf 1 setzen
  temp.arr[(ze+1)-1] = 1;

  // aktuellen Vektor um homogene Komponente erweitern
  if (arr != NULL) delete [] arr; // Speicherplatz des Matrix-Arrays freigeben
  ze = ze+1;
  arr = new double[ze];           // neues Vektor-Array erzeugen
  for (i=0; i < ze; i++) arr[i] = temp.arr[i];
  homogene_koordinaten = true;
  return *this;
}

// Umwandlung in einen Vektor in kartesischen Koordinaten
TVektor TVektor::mache_kartesisch(void)
{
  unsigned i;

  if (ze == 0)
  {
    cerr << "(V.MK)Fehler: Vektor mit Zeilenanzahl 0.\n";
    return *this;
  }
  else if (!this->ist_homogen())
  {
     cerr << "(V.MK)Fehler: Vektor bereits in kartes. Koord. angegeben.\n";
    return *this;
  }

  TVektor temp(ze-1); // temp-Vektor ohne homogene Komponente

  // Kopieren des Ursprungsvektors
  // aktueller Vektor ohne homogene Komponente
  for (i=0; i < temp.ze; i++) temp.arr[i] = arr[i];

  if (arr != NULL) delete [] arr; // Speicherplatz des Vektor-Arrays freigeben
  ze = ze-1;
  arr = new double[ze];           // neues Vektor-Array erzeugen
  for (i=0; i < ze; i++) arr[i] = temp.arr[i];
  homogene_koordinaten = false;
  return *this;
}

// Berechnung des Betrags (Länge) eines Vektors
double TVektor::vektor_betrag (void) const
{
  double betrag=0.0;
  unsigned temp_ze;

  if (ze == 0)
  {
    cerr << "(V.VB)Fehler: Kein echter Vektor (Zeilen == 0).\n";
    return -1.0;
  }

  if (this->ist_homogen())
    temp_ze=ze-1; // homogene Komponente des Vektors ignorieren
  else
    temp_ze=ze;   // alle kartesischen Koordinaten betrachten
```

```
  for (unsigned i = 0; i < temp_ze; i++)
  {
    betrag += arr[i]*arr[i];
  }
  return sqrt(betrag);
}

// Berechnung des Skalarproduktes zweier Vektoren
double TVektor::skalarprodukt (const TVektor & vekt)
{
  double skalar=0.0;
  unsigned temp_ze;

  if (ze == 0 || vekt.ze == 0)
  {
    cerr << "(V.SP)Fehler: Mindestens ein Vektor mit Dimension 0.\n";
    return -1.0;
  }
  else if (ze != vekt.ze)
  {
    cerr << "(V.SP)Fehler: Nur Vektoren gleicher Dimension zulässig.\n";
    return -1.0;
  }

  if (this->ist_homogen() && vekt.ist_homogen())
    temp_ze=ze-1; // homogene Komponente des Vektors ignorieren
  else if (!(this->ist_homogen()) && !(vekt.ist_homogen()))
    temp_ze=ze;   // alle kartesischen Koordinaten betrachten
  else
  {
     cerr << "(V.SP)Fehler: 1 Vektor ist in homog. Koord. angegeben.\n";
    return -1.0;
  }

  for (unsigned i = 0; i < temp_ze; i++)
  {
    skalar += arr[i]*vekt.arr[i];
  }
  return skalar;
}

// Berechnung des Winkels zwischen zwei Vektoren in RADIANT
double TVektor::winkel (const TVektor & vekt)
{
  double winkel=-1.0; // falls Rückgabewert -1.0, dann Fehlerfall

  if (ze == 0 || vekt.ze == 0)
  {
    cerr << "(V.Wi)Fehler: Mindestens ein Vektor mit Dimension 0.\n";
    return winkel;
  }
  else if (ze != vekt.ze)
  {
    cerr << "(V.Wi)Fehler: Nur Vektoren gleicher Dimension zulässig.\n";
    return winkel;
  }
  else if ( (this->ist_homogen() && !(vekt.ist_homogen()) ) ||
            (!(this->ist_homogen()) && vekt.ist_homogen() )    )
  {
    cerr << "(V.Wi)Fehler: Nur 1 Vektor in homogenen Koordinaten angegeben.\n";
    return winkel;
  }
```

```
  // Berechnung von |a|*|b|
  if ((winkel = vekt.vektor_betrag() * this->vektor_betrag()) == 0)
  {
    cerr << "(V.Wi)Fehler: Winkel kann nicht berechnet werden (Nullvektor).\n";
    return winkel;
  }
  // Berechnung von a * b / |a|*|b|
  winkel = this->skalarprodukt(vekt) / winkel;
  // Berechnung von arcos (a*b / |a|*|b|) = Winkel zw. 2 Vektoren
  winkel = acos(winkel);
  return winkel;
}

// Setzen der x,y,z-Koordinaten eines 3dim Vektors
bool TVektor::set_XYZ(double x, double y, double z)
{
  if (ze == 3)
  {
    arr[0]=x; arr[1]=y; arr[2]=z;
    return true;
  }
  else return false;
}

// Auslesen der x,y,z-Koordinaten eines 3dim Vektors
bool TVektor::get_XYZ(double & x, double & y, double & z)
{
  if ((ze == 3) || (ze == 4))
  {
    x=arr[0]; y=arr[1]; z=arr[2];
    return true;
  }
  else return false;
}


//*****************************************************************************
// Matrizenliste-Klasse
//*****************************************************************************
// Liste von Transformationsmatrizen
// organisiert als Stack
// mit push(element), pop(), ist_leer()


// Listenkonstruktor um eine Liste mit einer bestimmten Anzahl von
// Listenelementen zu erzeugen
TMatrizenListe::TMatrizenListe(unsigned anzahl)
{
  liste = NULL;
  akt_elemente = 0;
  max_elemente = anzahl;

  if (anzahl != 0 )
  {
    liste = new TMatrix[max_elemente];
  }
}
```

```
// Matrix zur Liste hinzufügen
bool TMatrizenListe::push(const TMatrix & trafo)
{
  if (akt_elemente != max_elemente)
  {
    liste[akt_elemente++] = trafo;
    return true;
  }

  return false;
}

// letzte Matrix aus Liste entfernen
TMatrix TMatrizenListe::pop(void)
{
  TMatrix temp;

  if (akt_elemente)
  {
    temp = liste[--akt_elemente];
  }
  return temp;
}

TMatrix TMatrizenListe::zeige(unsigned position)
{
  TMatrix temp;

  if (position == 0)
  {
    cerr << "(ML.Zeig)Fehler: Nulltes Listenelement existiert nicht.\n";
  }
  else if (akt_elemente && (akt_elemente >= position) )
  {
    temp = liste[position-1];
  }

  return temp;
}
```

Literaturverzeichnis

[1] H. Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung,* Band 1 aus der Reihe *Lehrbücher der Informatik*. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1996.

[2] H. Balzert. *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung,* Band 2 aus der Reihe *Lehrbücher der Informatik*. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford, 1998.

[3] K. Bothe. *Reverse Engineering: the Challenge of Large-Scale Real-World Educational Projects*. 14th Conference on Software Engineering Education and Training, CSEE&T 2001, Charlotte, USA, Febr. 2001. `http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Veroeffentlichungen/CSEET.ps`.

[4] K. Bothe. *Pflichtenheft: RTK-Steuerprogramm / Funktion: Probe und Kollimator manuell justieren*. Humboldt-Universität zu Berlin, Institut für Informatik, Lehrstuhl Softwaretechnik, Version 1.1 vom 7.9.2000. `http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Entwicklerdokumente/Manu_Justage_Ges_Dok/Manuelle_Justage.v1.1.html`.

[5] J. Foley [et al.]. *Einführung in die Computergrafik: Einführung, Methoden, Prinzipien*. Addison-Wesley, Bonn, Paris, Reading, Massachusetts, 1994.

[6] H. Stöcker (Hrsg.). *Taschenbuch mathematischer Formeln und moderner Verfahren*. Verlag Harri Deutsch, Thun, Frankfurt am Main, 2.,überarb. Auflage, 1993.

[7] P. Rennert (Hrsg.). *Kleine Enzyklopädie Physik*. VEB Bibliographisches Institut Leipzig, Leipzig, 1. Auflage, 1986.

[8] U. Sacklowski K. Bothe. *Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen.* 7. Workshop Software Engineering im Unterricht der Hochschulen, SEUH, Zürich, Schweiz, Febr. 2001. `http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Veroeffentlichungen/SEUH_2001.pdf`.

[9] R. Köhler. *Emails mit der Physik - Anforderungen an das RTK-Steuerprogramm.* Humboldt-Universität zu Berlin, Institut für Physik, Arbeitgruppe „Röntgenbeugung an dünnen Schichten", 23.04.1999. `http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Emails/23.04.99.txt`.

[10] R. Köhler. *Emails mit der Physik - Anfrage an den Lehrstuhl Softwaretechnik.* Humboldt-Universität zu Berlin, Institut für Physik, Arbeitgruppe „Röntgenbeugung an dünnen Schichten", 25.06.1998. `http://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Emails/25.06.98.html`.

[11] B. Oestereich. *Objektorientierte Softwareentwicklung: mit der Unified modeling language.* R. Oldenbourg Verlag, München, Wien, 3., aktualisierte Auflage, 1997.

[12] Physik Instrumente PI. *C-832 DC-Motor Controller / Programming Manual.* Physik Instrumente GmbH, D-76333 Waldbronn, Germany, Release: 1.1 Date: 13.April 1993. Low Level Programming and C Program Examples.

[13] Physik Instrumente PI. *C-812 DC-Motor Controller / Operating Manual, Programming Reference, Product Documentation.* Physik Instrumente GmbH, D-76333 Waldbronn, Germany, Release: 5.01 Date: 14.Dez 1993.

[14] A. Schad S. Lühnsdorf. *Radicon SCSCS - Mittlere Implementationsebene.* Humboldt-Universität zu Berlin, Institut für Informatik, Lehrstuhl Softwaretechnik. `http://www.informatik.uni-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJ_SE_WS98/Entwicklerdokumente/Detektoren_Ges_Doku/0-dim-Detktoren/Radicon/Vortrag/ebenen.html`.

[15] K. Schützler. *Studienarbeit:Implementation eines 0-dimensionalen Testdetektors zur Simulation einer realen Probe.* Lehrstuhl Softwaretechnik,

Humboldt-Universität zu Berlin, Math.-Nat.Fakultät II, Institut für Informatik, 2000.

[16] B.-U. Pagel und H.-W. Six. *Software Engineering Band 1: Die Phasen der Softwareentwicklung.* Addison-Wesley, Bonn, Paris, Reading, Massachusetts, 1994.

[17] I.N. Bronstein und K.A. Semendjajew. *Taschenbuch der Mathematik: Ergänzende Kapitel.* B.G.Teubner-Verlag, Leipzig, 1979.

[18] I.M. Bomze und W. Grossmann. *Optimierung - Theorie und Algorithmen.* B.I.Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993.

Index

Abbruchbedingungen, 62
Abbruchkriterium, 46, 112, 118
Anforderungsspezifikation, 36
Anstiegsrichtung, 47
Anstiegsverfahren, 46
Anwendungsszenario, 39
Arbeitspunkt, 30
Architekturentwurf, 64
Ausleuchtung
    der Probe, 114
Automatische Justage
    Eingaben, 41
    Einstellungen, 126
    Suchbereich, 42, 128

Befehlssatz
    der Motorencontrollerkarte, 20
Benutzerfreundlichkeit, 24
Benutzerschnittstelle, 99
Bragg
    -Gleichung, 4
    -Reflex, 4, 27

C-Schnittstelle zur Motorsteuerung
    `mActivateDrive`, 158
    `mExecuteCmd`, 163
    `mGetAxisName`, 159
    `mGetAxisUnit`, 159
    `mGetDF`, 160
    `mGetDistanceProcess`, 155
    `mGetDistance`, 154
    `mGetMoveFinishIdx`, 168
    `mGetMoveScan`, 167
    `mGetSF`, 161
    `mGetScanSize`, 167
    `mGetUnitType`, 156
    `mGetValue`, 156
    `mIsCalibrated`, 153
    `mIsDistanceRelative`, 153
    `mIsMoveFinish`, 152
    `mIsRangeHit`, 152
    `mMoveByDistance`, 151
    `mMoveToDistance`, 150
    `mPopSettings`, 164
    `mPushSettings`, 164
    `mSavePosition`, 166
    `mSetAngleDefault`, 161
    `mSetCorrectionState`, 158
    `mSetLine`, 151
    `mSetRelativeZero`, 162
    `mSetValue`, 157
    `mStartMoveScan`, 165
    `mStopDrive`, 155
    `mlGetAxisNumber`, 145
    `mlGetAxis`, 139
    `mlGetDistance`, 140
    `mlGetIdByName`, 140
    `mlGetInstance`, 147
    `mlGetOffset`, 142
    `mlGetValue`, 141
    `mlGetVersion`, 146
    `mlInitializeMotorsDLL`, 138
    `mlIsAxisValid`, 143
    `mlIsMoveFinish`, 142
    `mlIsServerOK`, 144
    `mlMoveToDistance`, 141
    `mlParsingAxis`, 143
    `mlSaveModuleSettings`, 145
    `mlSetAngleDefault`, 146

mlSetAxis, 139
Dialoge
mlInquireReferencePointDlg, 147
mlOptimizingDlg, 148
mlPositionControlDlg, 149
mlSetParametersDlg, 149
Controllerkarte
C-812, 19
C-832, 20

Datentyp
TMatrix, 73
TMatrizenListe, 73
TVektor, 73
TransformationClass, 71
Design, 59
Detektor, 10, 21
-ansteuerung, 94
0-dimensionaler, 95, 122
2-dimensionaler, 104
Einstellungen, 125
Test-, 100
Dialogelemente, 81
Dialogklasse
TAutomaticAngleControl, 70, 83
TModalDlg, 70
modale, 68
Dialogparameter, 60
Dialogprogrammierung, 79
Dialogsteuerung, 66
Diffraktometrie, 2
Durchläufe, 127

Einstellungen
probenabhängige, 98
probenunabhängige, 98
Entwicklungsumgebung, 15, 79
Event-Handler, 89

Fehleranfälligkeit, 22
Flussdiagramm, 60
Fotoplatte, 30
Freiheitsgrade, 10, 60

Gangunterschied, 3
Gesamttransformationsmatrix, 53
Gradient, 46
Gradientenverfahren, 46

Halbwertsbreite, 5, 13, 30
messen, 129
Handler, 85

Implementation, 94
Initialisierungsdatei, 14
Intensitätsmessungen, 63
Anzahl der, 128
Intensitätsschwankungen, 63
Intensitätsverteilung, 47
Interferenz, 2, 4
Intervallgrenzen, 63

Justage
manuelle, 27
azimutale Rotation, 123
iterativer Prozeß, 30
Relative Null, 142, 146, 154, 161, 162
Referenzpunktlauf, 147, 153

Klassendiagramme, 69
Kollimator, 6
gekrümmter, 8
Schätzfunktion, 134
Kommunikation
Hardware-, 19
Komponenten, 64
Koordinaten
homogene, 53
Koordinatensystemtransformation, 48, 51
Korrektheit, 21

Logdatei, 41, 127

Makro, 133
Makrosteuerung, 134
Max. Intensitätsdifferenz, 127
Median, 72
Modell
    inkrementelles, 93
Monochromatisierung, 6
Multitasking
    präemptives, 93

Nachricht
    WM_COMMAND, 84
    WM_INITDIALOG, 88
    WM_TIMER, 89
Nachrichtenschleife, 85
Netzebenen, 2, 3
Nutzerdokumentation, 121

ObjectWindows, 69
Optimierung
    nichtlineare, 46, 62
Optimierungsverfahren, 46

Peak, 5, 29
Pflichtenheft, 37
    Aufgabe des ∼s, 36
Positionsvektor, 62
Probengeometrie, 115
Probenhalter, 9
Probenteller, 9
Programmfunktion, 66
Projekt
    -datei, 15
    -quelltexte, 15
    -struktur, 15
Projektfenster, 78
Protokolldatei, 41, 111, 113

Qualitätskriterien, 24
Quelltextstruktur, 68

Röntgen
    -beugung, 3
    -diffraktometrie, 2
    -reflektometrie, 2
    -strahlung, 6
    -topographie, 2
      Zwei-Kristall, 6
Reflektometrie, 2
Reflexionskurve, 3
Relaxation, 47
Resource Workshop, 68, 79
Ressourcenbezeichnungen, 82
Ressourcendateien, 79
Ressourcentyp, 82
Robustheit, 22
Rockingkurve, 5, 27
    Flanke der ∼, 31
Rotation, 52
Rotationsmatrix, 56
    inverse, 52

Schichtsystem, 2
Schnittstelle
    IEEE-488, 20
    PC-Bus, 20
    RS232, 19
Skalierung, 53
Softwareschranken, 60
Softwaretest, 97
statische Struktur, 69
Statusfenster, 102, 126
Steuerprogramm, 13
Suche
    eindimensionale, 46
Suchrichtung, 46
Suchverfahren
    dichotome Suche, 50
    Fibonacci-Suche, 50
    Goldener Schnitt, 50
    sequentielle, 49

Test, 97

-strategie, 98
-szenarien, 99
-ziele, 97
der Güte, 103
des Dialogs, 99
des Zeitverhaltens, 103
Funktions-, 101
Parametereinstellungen, 102
Timer, 87
-event, 88
-funktion, 89
-programmierung, 87
Toleranz, 127
Topographie, 2, 27
-vorgang, 27
Transformation, geometrische, 51
Transformationsvektor, 48
Translation, 51
Translationsmatrix, 56

UML-Notation, 69

Versetzungen, 2
Verspannungen, 3
Versuchsaufbau, 11
Voreinstellungen
Automatische Justage, 43, 123
Vorjustierung, 133

Wartbarkeit, 23, 93
Weltkoordinaten, 56
Weltkoordinatensystem, 60
Wertebereich, 60
Windows API, 69
Windowsnachrichtenkonzept, 85

Zähler
-fenster, 70
Radicon-, 21
Zählerkonfiguration, 123
Zielfunktion, 45