Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Niklas Klitscher

Erweiterung eines Röntgensystems mit Laserinfrarotsensoren zur Ermittlung von Objektdimensionen, um die optimale Röntgenanodenspannung zu bestimmen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Niklas Klitscher

Erweiterung eines Röntgensystems mit Laserinfrarotsensoren zur Ermittlung von Objektdimensionen, um die optimale Röntgenanodenspannung zu bestimmen

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

Betreuender Prüfer : Prof. Dr. Robert Heß
Zweitgutachter : Prof. Dr. Marc Hensel

Abgegeben am 16. Januar 2017

Niklas Klitscher

Thema der Bachelorthesis

Erweiterung eines Röntgensystems mit Laserinfrarotsensoren zur Ermittlung von Objektdimensionen, um die optimale Röntgenanodenspannung zu bestimmen

Stichworte

Röntgen, Infrarotlaserscanner, MFC, C++, Multithreading, XML

Kurzzusammenfassung

Die Arbeit befasst sich mit der Erstellung einer Demonstratoreinheit, die veranschaulichen soll, wie eine Röntgenanlage mittels Zusatzsensor erweitert werden kann, um eine passende Röntgenanodeneinstellung vorzuschlagen. Dabei wurde ein Programm mit einer grafischen Oberfläche in C++ mit MFC entwickelt, dass mit dem Sensor kommuniziert und dessen Daten verarbeitet. Es wurde Multithreading angewandt und Kurvenpunkte, der optimalen Röntgenparameter, aus einer XML-Datei ausgelesen. Der Demonstrator wurde aus Aluminium-Profilen gebaut und als Sensor wurde ein 2D Laserinfrarotsensor der Sick AG benutzt.

Niklas Klitscher

Title of the paper

Extension of an x-ray system with laser infrared sensors for the determination of object dimensions in order to determine the optimal x-ray anode voltage

Keywords

roentgen, infrared laser scanner, MFC, C++, Multithreading, XML

Abstract

The work deals with the building of a demonstrator unit. The unit shall show how a x-ray system could be upgraded with an additional sensor to suggest optimal settings of the x-ray anode. A program with a graphical interface was developed in C ++ with MFC that communicates with the sensor and processes its data. Multithreading was applied and curve points, the optimal x-ray parameter, were read from a XML file. The demonstrator was constructed from aluminum profiles. A 2D laser infrared sensor from Sick AG was used as a distance sensor.

Inhaltsverzeichnis

1 Einleitung

Das Ziel dieser Bachelorarbeit ist es eine Demonstratoreinheit zu entwickeln, die ein Röntgensystem mit einem zusätzlichen Distanzsensor darstellt. Mit einer dazu entwickelten Software sollen über die Sensoreinheit Objektdimensionen unter dem Röntgenkopf ermittelt werden und beispielhaft die optimalen Röntgenanodeneinstellungen berechnet werden. Die Software soll eine optische Oberfläche besitzen, auf der die ermittelten Werte und die grafische Darstellung der Messung angezeigt werden.
Die Röntgenanodenwerte sollen aus Datenkurven interpoliert werden, die in einer externen XML-Datei vorhanden sind.

Anhand der Demonstratoreinheit und zugehöriger Software soll die Machbarkeit der Optimierung einer Röntgenanlage demonstriert werden. Somit können durch einen zusätzlichen Sensor, der die Dimensionen des Patienten erkennt und daraus die optimalen Anodeneinstellungen berechnet und vorschlägt, Fehlaufnahmen reduziert werden. Das senkt die Strahlenbelastung des Patienten.

2 Theorie

2.1 Die medizinische Röntgenanlage

Die Röntgenstrahlung wurde 1895 von Wilhelm Conrad Röntgen entdeckt. Röntgenstrahlen sind elektromagnetische Wellen, deren Wellenlänge von circa 5 pm bis 10 nm reicht.

Gleichzeitig wurde auch die Methode der Röntgendiagnostik (kurz Röntgen) von Wilhelm Conrad Röntgen entdeckt. Dabei werden Objekte mit Röntgenstrahlen bestrahlt, wobei die Strahlung das Objekt, abhängig von der Dicke und dem Material, unterschiedlich stark durchdringt. Auf der Rückseite des Objektes belichten die Strahlen dann eine analoge Filmplatte oder heutzutage meist einen digitalen Sensor. Wie in der analogen und digitalen Fotografie entsteht so ein Bild nur, dass dieses Informationen über das nicht sichtbare Innere des durchstrahlten Objektes liefert (Siehe Abb. 2.1 auf S. 8).

Somit ist das Prinzip der Bildaufnahmen beim Röntgen vergleichbar mit der alltäglichen Fotografie. Der einzige Unterschied hier ist, dass dabei nicht das sichtbare Licht (Wellenlänge circa 380 nm bis 700 nm) benutzt wird. Damit kommen wir auch schon zum ersten Problem. Bei der Fotografie kann die Kamera die Lichtintensität jederzeit messen, da das Licht immer vorhanden ist und so die Sensorempfindlichkeit, Blende und Verschlusszeit für ein optimales Bild einstellen.

Bei einer Röntgen-Bildaufnahme funktioniert dies leider nicht so einfach. Die Röntgenstrahlen werden extra von einer Anode erzeugt. Die Einstellmöglichkeiten sind Varianten in der Stromstärke, der Spannung und der Zeit. Aufgrund der Schädlichkeit der Strahlen für den Organismus werden diese nur während der Bildaufnahme erzeugt. Daraus ergibt sich, dass vorher nicht die optimale Strahlendosis für das Bild gemessen werden kann, da diese noch gar nicht für eine Messung vorhanden ist.
In der Medizin befasst sich die Röntgendiagnostik mit der Anwendung dieser Technik auf einen organischen Körper. Ärzte können mithilfe eines Röntgenbildes einen Einblick in den Körper bekommen und so zum Beispiel Knochenbrüche oder Gewebeanomalien feststellen. In der Realität stellt der Arzt die Anodenparameter manuell ein oder wählt verschiedene Programme. Dies macht er abhängig von den Dimensionen des Patienten und welche Art von Materie auf der Aufnahme besonders kontrastreich dargestellt werden soll.

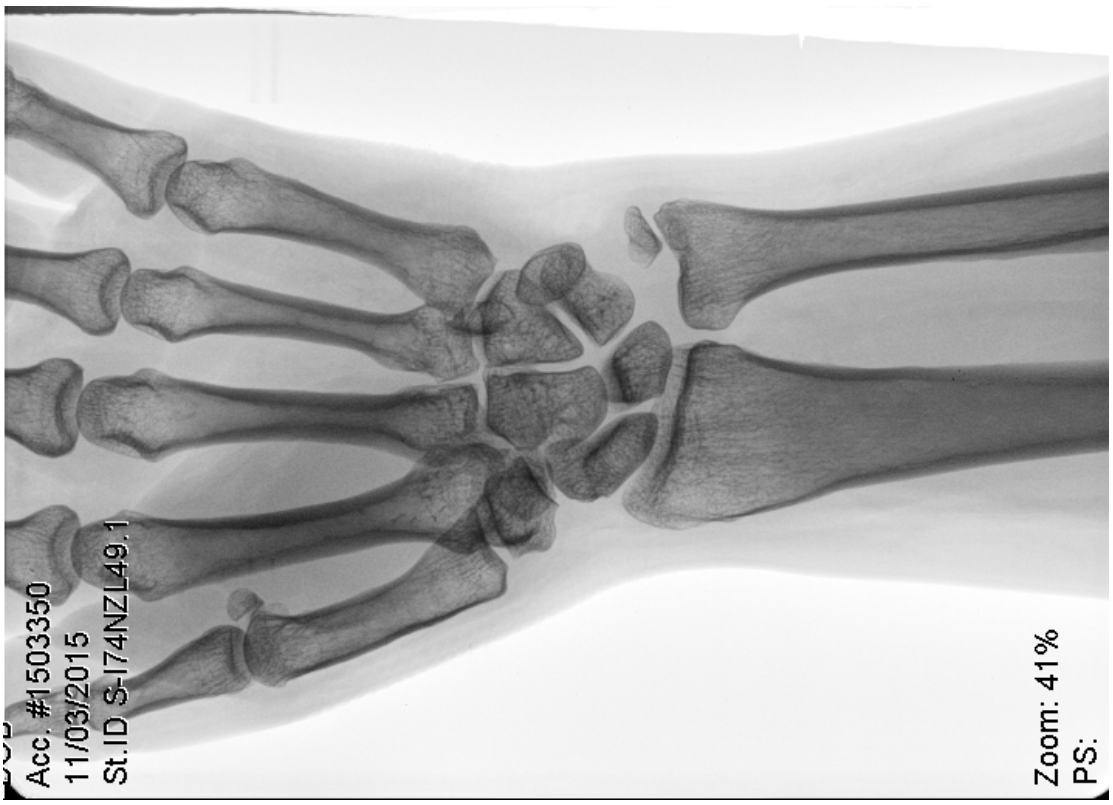

Abbildung 2.1: Röntgenaufnahme eines Handgelenkes (Farben invertiert)
(Quelle: https://commons.wikimedia.org/wiki/File:Falta_de_fusión_del_núcleo_de_la_estiloides.jpg)

2.2 Strahlenbelastung

Beim medizinischen Röntgen kann es zu Fehleinstellungen der Röntgenanodeneinstellung kommen und damit zu schlechten Bildergebnissen. Durch die wiederholte Neuaufnahme eines Röntgenbildes wird der Patient unnötig mit Strahlen belastet. Problematisch ist, dass Röntgenstrahlung in der Lage ist, Schäden am Organismus zu verursachen. Durch diese energiereiche Strahlung können Atome und Moleküle ionisiert werden, wodurch zum Beispiel Veränderungen der Erbstruktur(DNA) hervorgerufen werden können und Krebs zur Folge haben kann.

2.3 Sensorvarianten

Zur räumlichen Messung eignen sich verschiedene Arten von Sensoren, die auch unterschiedliche Funktionsweisen benutzen.
Einige Sensortypen sind zum Beispiel:

- **Laserentfernungsmesser**
  Hierbei sendet ein Laserscanner einen Lichtimpuls aus, der von einem Objekt reflektiert wird und auf einen Detektor trifft. Durch die zeitliche Differenz kann somit eine Entfernung zum Objekt berechnet werden.
- **Time of Flight (ToF Kamera)**
  Hierbei wird das selbe Prinzip, wie beim oben genannten Laserentfernungsmesser benutzt, nur dass gestreutes Infrarotlicht emittiert wird und die reflektierten Strahlen auf einen vielzelligen Sensor fokussiert werden. Durch die Lichtlaufzeitmessung kann somit für jede Zelle eine Entfernung berechnet werden. Das Ergebnis ist ein 3D Bild.
- **Ultraschallsensor**
  Beim Ultraschallsensor wird der Abstand zwischen Sensor und Objekt mittels Echolaufzeitmessung gemessen.
- **Radarsensor**
  Beim Radarsensor wird der Abstand zwischen Sensor und Objekt mittels Radarlaufzeitmessung gemessen.

Die Vorteile der Ultraschall- und Radarsensoren sind, dass die Messung bei nahezu allen Objekten zuverlässig funktioniert, wohingegen die ToF Kamera und der Laserscanner Probleme bei transparenten oder Licht ableitenden Oberflächen haben können. Ein Nachteil ist, dass Radar und Ultraschall nicht gebündelt sind. Somit muss mit mehreren Empfangssensoren ein Muster der reflektierten Wellen ausgewertet werden, um ein 2D oder 3D Bild der Umgebung zu erhalten.

2.4 XML

Extensible Markup Language (kurz: XML) - deutsch: erweiterte Auszeichnungssprache

XML ist ein Standard um Informationen zu strukturieren. Durch diese Struktur werden Abhängigkeiten von Informationen zueinander „ausgezeichnet“. In solch einem Strukturformat können die Informationen standardisiert ausgetauscht, eingelesen oder abgespeichert werden. Durch den strukturellen Aufbau ist es, sogar als Person leicht möglich Informationen und Abhängigkeiten aus den Daten zu lesen.

> „*Mittels XML können somit Auszeichnungssprachen für beliebige Anwendungsbereiche maßgeschneidert werden und anwendungsspezifische Datenstrukturen beschrieben werden.*“ [1][2]
>
> – Prof. Dr. Gustaf Neumann, Wirtschaftsuniversität Wien, Institut für Wirtschaftsinformatik und Neue Medien

Ein weiterer Vorteil ist, dass in der objektorientierten Programmierung XML-Parser es möglich machen aus einem XML-Datensatz ein DOM-Objekt (Document Object Model) zu erstellen. Aus solch einem DOM-Objekt ist es dann möglich mit der richtigen Programmierung die Information auszulesen und zu verarbeiten oder andersherum abzuspeichern.

2.5 Threading, Flags und Semaphoren

Multithreading (eng. *„Thread“* - dt. *„Faden“*) bedeutet, dass es in einem Prozess mehrere „Ablaufstränge“ gibt. Durch moderne Mehrkern-CPUs ist es somit möglich, dass einzelne CPU-Kerne einzelne Threads eines Prozesses ablaufen und so parallel Datenverarbeitung möglich machen.
Die parallele Datenverarbeitung ermöglicht mitunter eine beschleunigte Programmverarbeitung und eine neue Ablaufstruktur. Aber parallel laufende Threads greifen dabei womöglich auf einen gemeinsam genutzten Speicherraum zu. Um dabei Konflikte zu vermeiden, wie zum Beispiel, dass ein Thread einen Bereich liest, während ein anderer zur selben Zeit diesen überschreibt, werden unter anderem Datenstrukturen wie Flags und Semaphoren angewandt. Diese werden zum Steuern des Zugriffes auf gemeinsam zugängliche Ressourcen genutzt, indem in ihnen Information über den Zugriffsstatus abgespeichert werden.

3 Anforderungen

3.1 Betrachtetes Szenario

Es soll das Anwendungsszenario betrachtet werden, in dem es um eine Röntgenanlage mit Liege geht und der Patient oder ein Körperteil direkt unter dem Röntgenkopf auf der Liege liegt.

Abbildung 3.1: Moderne digitale Röntgenanlage von Philips
(Quelle: https://en.wikipedia.org/wiki/Projectional_radiography)

Ziel:
Mittels eines Sensors am Röntgenkopf und einer Software soll dem jeweiligen Arzt die optimale Röntgenanodeneinstellung vorgeschlagen werden.

3.2 Hardwareanforderungen

Die Anforderungen an den Sensor sind eine exakte Messgenauigkeit, einen Messbereich, der dem der Röntgenanlage entspricht und, dass er dem Patienten keinen Schaden zufügt, also zum Beispiel auch augensicher ist. Zudem soll die Röntgenanlage als Demonstrator nachgebaut werden. Dabei soll ein Ausleger den verstellbaren Röntgenkopf simulieren und höhenverstellbar sein.

3.3 Softwareanforderungen

Es soll ein Programm entwickelt werden, dass eine Verbindung zum Sensor herstellt, dessen Messwerte ausliest und daraus die Dimensionen des darunterliegenden Objektes ermittelt. Die zugehörigen, optimalen Röntgenanodeneinstellungen sind als einzelne Kurvenpunkte in einer XML Datei abgespeichert. Aus diesen sollen jeweilige Einstellungen interpoliert werden. Außerdem sollen verschieden Röntgenprogramme zur Verfügung stehen, je nachdem welche inneren Teile des Körpers besonders kontrastreich betrachtet werden sollen.
Anschließend soll die Messung grafisch dargestellt und die berechnete Anodeneinstellung angezeigt werden.

4 Hardwaredesign

4.1 Der Sensor

4.1.1 Sick AG

Da an der HAW kein passender Sensor für die Verwirklichung des Projektes vorhanden war, ging es darum, einen passenden Sensor zu finden. Bei der Recherche nach einem Sensorhersteller kam die Firma Sick ins Spiel. Die Sick AG ist eine deutsche Firma, die weltweit Sensoren für die Automatisierung vertreibt und einen Standort in Hamburg hat. Durch frühere Arbeiten hatte Hr. Prof. Heß Kontakt zu Ingenieuren bei der Sick AG. Bei einem Treffen mit den Ingenieuren konnte die Produktpalette an Sensoren verglichen werden.

4.1.2 Tim

Bei dem Vergleich der Produktpalette schnitten die 2D Infrarotlaserscanner der Firma Sick am besten ab. Sie sind augensicher, haben eine kompakte Bauform und der Messbereich beginnt schon ab 5 cm. Viele, der anderen Sensoren konnten erst ab einer Entfernung von 0,2 m bis 0,5 m messen. Letztendlich fiel die Wahl auf den 2D-Laserscanner TIM571-2050101. Er ist per Ethernet ansprechbar und besitzt einen rotierenden Laserscanner, der ein 2D Abbild der Umgebung macht.
Die Sick AG spendete der HAW Hamburg im Anschluss einen 2D-Laserscanner vom Typ TIM571-2050101.

Abbildung 4.1: Infrarot Laserscanner TIM571
(Quelle: https://www.sick.com)

Weitere nennenswerte Merkmale[3] des TIM57x sind:

Lichtquelle	Infrarot (850 nm)
Laserklasse	1, augensicher
Öffnungswinkel	270 °
Auflösung	0,33 °
Scanfrequenz	15 Hz
Arbeitsbereich	0,05 m ... 25 m
Antwortzeit	Typ. 67 ms
Systematischer Fehler (bis 10 m)	±60 mm
Statistischer Fehler (bis 10 m)	20 mm
Betriebsspannung	9 V DC ... 28 V DC
Leistungsaufnahme	4 W

Tabelle 4.1: Spezifikationen des Tim

Der Sensor besitzt einen, sich im Betrieb, drehen Spiegel unter der Optikhaube. Die Laserstrahlen treffen von dem Laser, der sich im Hauptgehäuse befindet, auf den Spiegel und werden in einem 90 ° Winkel abgelenkt. Die Optikhaube ist dabei lichtdurchlässig für das Licht des Lasers. Werden die ausgesendeten Laserstrahlen von einem Objekt reflektiert, werden diese über den Spiegel zurück ins Gehäuse auf eine Detektier-Einheit gelenkt. Die Strecke ergibt sich aus der Durchlaufzeit des Lichtes.

4.2 Demonstrator

Bei dem Bau der Demonstratorkonstruktion wurde sich für den Werkstoff Aluminium in Form von Profilstangen entschieden. Im Vergleich zu Holz, welches auch zur Überlegung stand, ist das Erscheinungsbild wesentlich professioneller und durch die hohle Struktur sehr leicht. Nach einiger Recherche wurde sich für die Profile von der ©item Industrietechnik GmbH[4] entschieden, da an der Hochschule schon damit gearbeitet wurde und die Preisanfrage sehr schnell vom Kundenservice bearbeitet wurde.

Um die Patientenliege einer Röntgenanlage nachzubilden wurde entschieden, einen handelsüblichen Tisch zu nehmen mit einer ungefähren Höhe und Breite von 70cm. Die Länge ist irrelevant, da der Sensor parallel zur Tiefenseite des Tisches messen soll.

Da auf der Website von ©item sogar CAD-Dateien für jedes Werkteil downloadbar waren, konnte die weitere Planung in einer 3D Umgebung erfolgen. Hierfür wurde 123D© Design von Autodesk© benutzt, um einen virtuellen Aufbau zu erschaffen.

Abbildung 4.2: Virtueller Demonstrator-Aufbau erste Version
(Screenshot aus 123D© Design)

In der ersten Version (Abb. 4.2) wurde sich für eine 1,5m hohe Stange mit verschiebbarem Ausleger und einem Fuß in U-Form entschieden. Da sich so aber nur eine Distanz von 49cm vom Ausleger zur Tischplatte ergab, wurde sich in der finalen Version (Abb. 4.3) für eine Stangenhöhe von 2,5m entschieden, was den Messhöhenumfang um 1m vergrößerte.
Des Weiteren wurde die Form des Fußes von U-förmig zu einem Dreibein geändert. Somit wird ein Teilstück gespart, beziehungsweise ein Zuschnitt. Die Verbindungen am Fuß wurden zudem durch Winkeleinheiten verstärkt.

Abbildung 4.3: Virtueller Demonstrator-Aufbau final
(Screenshot aus 123D© Design)

5 Softwaredesign

5.1 Entwicklungsumgebung

Da auch die meisten Anwendercomputer Windows benutzen, wurde entschieden die Software für einen Windows PC zu entwickeln.

Als Entwicklungsumgebung wurde Visual Studio 2015 von Microsoft gewählt. Das Programm wird von Microsoft kostenlos für Studenten angeboten und ermöglicht es, in einer Vielzahl von Programmiersprachen verschiedene Arten von Anwendungen zu erstellen. Visual Studio bietet zudem einen Debugger an, der es ermöglicht die Vorgänge des erstellten Programmes Schritt für Schritt zu verfolgen, und sich Speicherinhalte anzusehen, um so den richtigen Programmablauf verifizieren zu können.
Seit Version 2015 von Visual Studio gibt es die Ansicht Diagnosetool, die im Debugmode aufrufbar ist. Dabei wird angezeigt, wie viel CPU-Last und Speicherverbrauch das jeweilige Programm, welches debuggt wird, verbraucht. Diese Funktion ist sehr hilfreich, um zum Beispiel Speicherlecks zu erkennen. Eine weitere nützliche Funktion ist es, sich die Durchlaufzeit des Programmes anzeigen zu lassen. Damit ist es möglich sein Programm weiter zu optimieren.

5.2 C++ und MFC

Microsoft stellt in Visual Studio 2015 die Klassenbibliothek MFC (Microsoft Foundation Classes) für die Programmiersprache C++ bereit. MFC macht es möglich, nach Microsoft standardisierte, Programme mit einer grafischen Oberfläche zu erstellen. MFC dient als Verknüpfung zu den jeweiligen Betriebssystemfunktionen, die nicht einfach so ansprechbar sind. Es fungiert als API (eng. application programming interface, dt.: Programmierschnittstelle) und somit lassen sich Betriebssystemfunktionen in C++ objektorientiert nutzen. Eine passende Laufzeitbibliothek ist auf den meisten Windows PCs vorinstalliert und somit sind mit MFC erstellte Programme ohne Zusatz meist direkt lauffähig. Visual Studio beinhaltet für MFC

Projekte auch einen Assistenten, der die Erstellung von Handlern und Methoden vereinfacht.
Für dieses Projekt wurde entschieden, eine Einzelfenster-Anwendung zu gestalten, ein sogenanntes Dialogfenster. Es wurde sich dagegen entschieden mit dem Wizard von Visual Studio eine MFC Anwendung zu kreieren, da diese mit vielen nicht verwendeten Features überlastet sind. Stattdessen wurde ein leeres Interface erstellt [5], worauf dann aufgebaut wurde. Um dann die grafische Oberfläche anzupassen, ist es in Visual Studio möglich per Drag'n'Drop Objekte wie Button, Textfelder und ähnliches auf der Oberfläche zu positionieren und danach per Assistent im Code zu integrieren.

5.3 Verwendete Fremdbibliothek TinyXML-2

TinyXML-2[6] ist ein frei verfügbarer XML Parser (eng. to parse „analysieren"), der aus der XML Datei ein DOM (Dokument Object Model)-Objekt erstellt. Mit den mitgelieferten Methoden kann dann der Inhalt geladen, verändert und neu abgespeichert werden. Es wurde sich für diesen XML-Parser entschieden, da er sehr leicht in das Projekt zu integrieren und unkompliziert in der Anwendung ist.

6 Realisierung

6.1 Programmcode-Aufteilung

Der Programmcode wurde in verschiedene Dateien aufgeteilt:

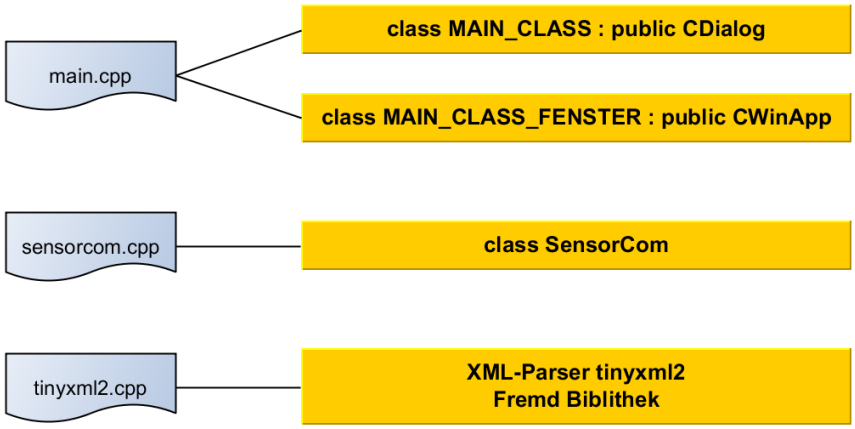

Abbildung 6.1: Dateistruktur mit Klassenaufteilung

Die Klasse **MAIN_CLASS_FENSTER** (Code: 6.1) ist von der CWinApp Klasse abgeleitet. In der MFC Bibliothek gehört diese Klasse zu der Familie „Application Architecture“. Mit dieser Klasse wird das Programm unter Windows geladen.

Das Programm ist eine MFCDialog Anwendung, die in der Klasse MAIN_CLASS beschrieben wird. In der virtuellen Funktion *InitInstance()* von MAIN_CLASS_FENSTER wird mittels *MAIN_CLASS dlg* eine Instanz, der CDialog Klasse, auf dem Stack erstellt.
Mit Hilfe der Memberfunktion *DoModal()* wird die Anwendung sichtbar gemacht.

Nach den Klassen-Deklarationen und der darauffolgenden Message Map wird im Code mit *MAIN_CLASS_FENSTER theApp* (Code: 6.2) gesagt, dass die Anwendung beim Start erzeugt werden soll.

```
1  class MAIN_CLASS_FENSTER : public CWinApp
2  {
3  public:
4          MAIN_CLASS_FENSTER() {  }
5
6  public:
7          virtual BOOL InitInstance()
8          {
9                  CWinApp::InitInstance();
10                 MAIN_CLASS dlg;
11                 m_pMainWnd = &dlg;
12                 INT_PTR nResponse = dlg.DoModal();
13                 return FALSE;
14         }
15 };
```

Code 6.1: MAIN_CLASS_FENSTER Klasse

```
1  MAIN_CLASS_FENSTER theApp;
2  //Es wird eine Instanz mit Namen theApp erzeugt.
```

Code 6.2: zum Starten der Anwendung

Die Klasse **MAIN_CLASS** ist von der CDialog Klasse abgeleitet, die wiederum von der CWnd Klasse abgeleitet ist und beinhaltet den Kern des Programmablaufes und die Zeichnung der Oberfläche.
In dieser Klasse wird die deklarierte Message Map aktiviert um Signale (gennant Messages) vom Windowssystem (zum Beispiel von Buttonklicks) abzufangen und gegebenenfalls Methoden auslösen zu lassen.
Zudem besitzt sie auch die virtuelle Methode OnInitDialog(). Diese wird standardmäßig ausgeführt wenn die Message WM_INITDIALOG, die beim Programmstart, beziehungsweise nach dem Ausführen von *DoModal()* in MAIN_CLASS_FENSTER, an das Programm gesendet wird. Dieses Verhalten muss nicht in der Message Map deklariert werden. In der Basismethode *OnInitDialog()* wird nur der Eingabefokus auf das erste Objekt in der Dialog Box gesetzt. In der überschriebenen Methode wurde die Basismethode aufgerufen und weitere Initialisierungsvorgänge vorgenommen.Nach diesem Verlauf beginnt der Programmverlauf, abhängig von Benutzeraktionen.

Die Klasse **SensorCom** (Name spielt auf Sensor-Kommunikation an) beinhaltet Methoden, die nur etwas mit der Sensorkommunikation und der darauffolgenden Datenaufarbeitung zu tun haben. Ein Objekt dieser Klasse stellt einen Sensor dar. In diesem Projekt wurde nur ein Sensor benutzt, es wäre aber möglich mehrere zu benutzten, und für jeden Sensor ein

Objekt zu erstellen. Durch die Abspaltung von dem restlichen Code in eine separate Datei, ist es leichter diesen Teil in anderen Projekten wieder zu verwenden.

6.2 Klassendiagramm

Folgendes Bild zeigt die selbst erstellten Klassen und deren Zusammenhänge in diesem Projekt.

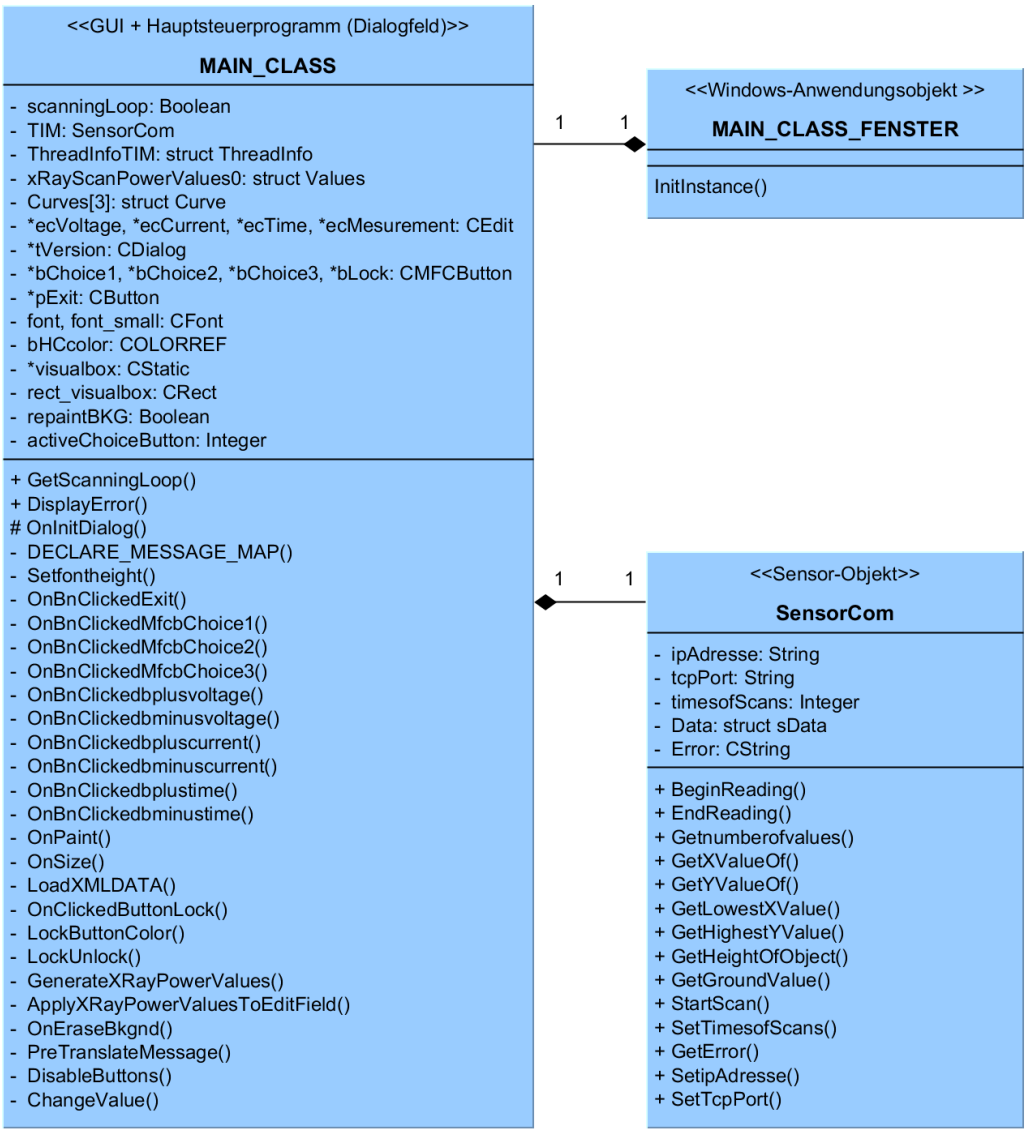

Abbildung 6.2: Klassendiagramm

6.3 User Interface

Die Programmoberfläche wird in C++ mit MFC in einer sogenannten Ressourcenskriptdatei(.rc) und einer Resource Header Datei(resource.h) beschrieben. In der Headerdatei finden sich Definitionen für die Ressourcen und in der Ressourcenskriptdatei wird die gesamte Benutzeroberfläche und die darauf befindlichen Elemente beschrieben.
Die Programmoberfläche wurde hier mit dem visuellen Assistenten von Visual Studio 2015 angelegt. Es wurde folgende Dialog Box Oberfläche erstellt:

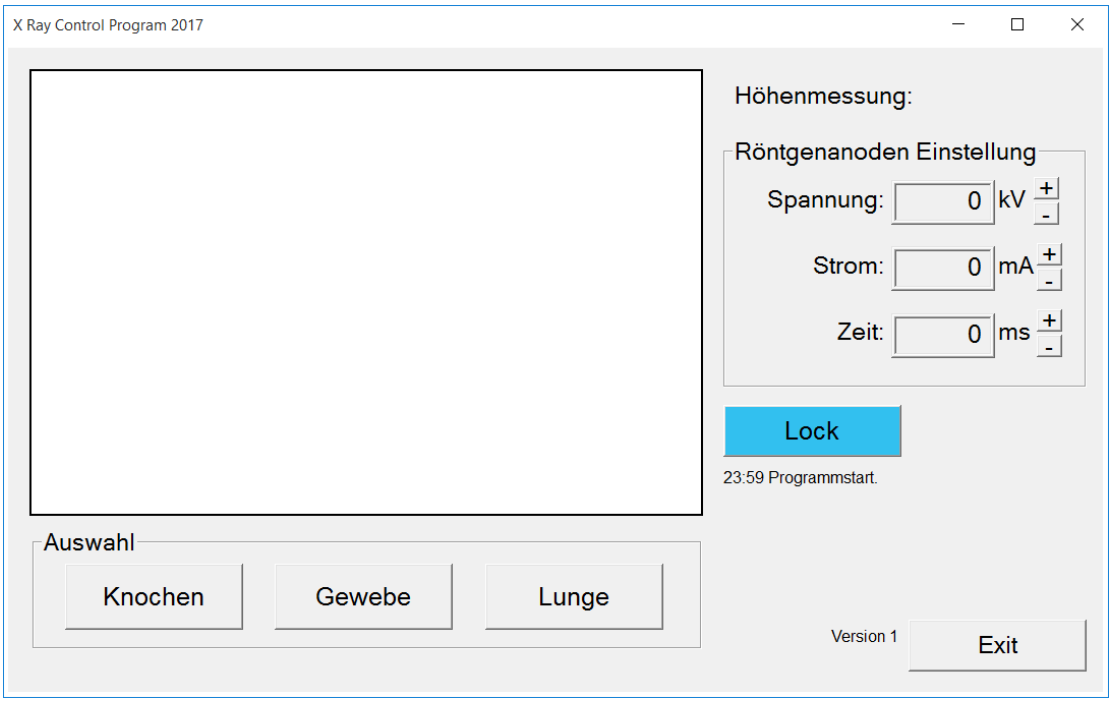

Abbildung 6.3: Programmoberfläche

Um den Zeichenbereich, der grafischen Darstellung von Messwerten, zu definieren, wurde das Dialog Element **Picture Control** verwendet (Weiße Box in Abb. 6.3). Die Größe dieses Elementes passt sich automatisch 1:1 zu der Fenstergröße an, um so eine optimale Ausnutzung der Fläche zu erreichen. Aus designtechnischen Aspekten wurden **Group-box Control** Elemente benutzt, um die Button der Auswahlprogramm und der Röntgenanodenparameteranzeige zusammenzufassen.

6.4 Programmstruktur

Im weiteren Verlauf wird konkret auf Programm-Inhalte eingegangen. Stark vereinfacht kann die Struktur aber wie folgt dargestellt werden:
Nach dem Programmstart beginnt der Initialisierungsprozess, in dem Variablen definiert und bestimmte Methoden aufgerufen werden. Im Anschluss kann der Benutzer per Button das regelmäßige Messen starten und stoppen, wenn erzeugte Anodeneinstellungen vorhanden sind diese manuell ändern, per Button die Kurve, aus dem die Anodenwerte erzeugt werden, ändern oder das Programm beenden.

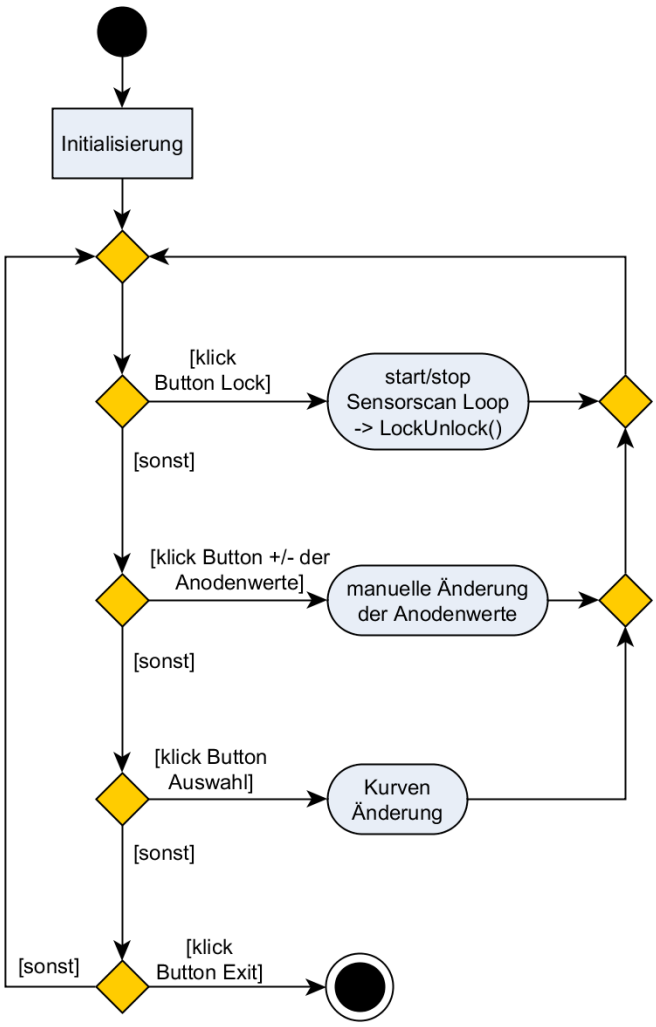

Abbildung 6.4: Allgemeine Programmstruktur aus Usersicht

6.4.1 Kontinuierliche Messungen mittels Thread Auslagerung

Betätigt der Anwender den Button „Lock“ wird über die verknüpfte Methode *OnClickedButtonLock()* die Methode *LockUnlock()* aufgerufen, über die der Start und Stopp der kontinuierlichen Messungen geregelt wird.

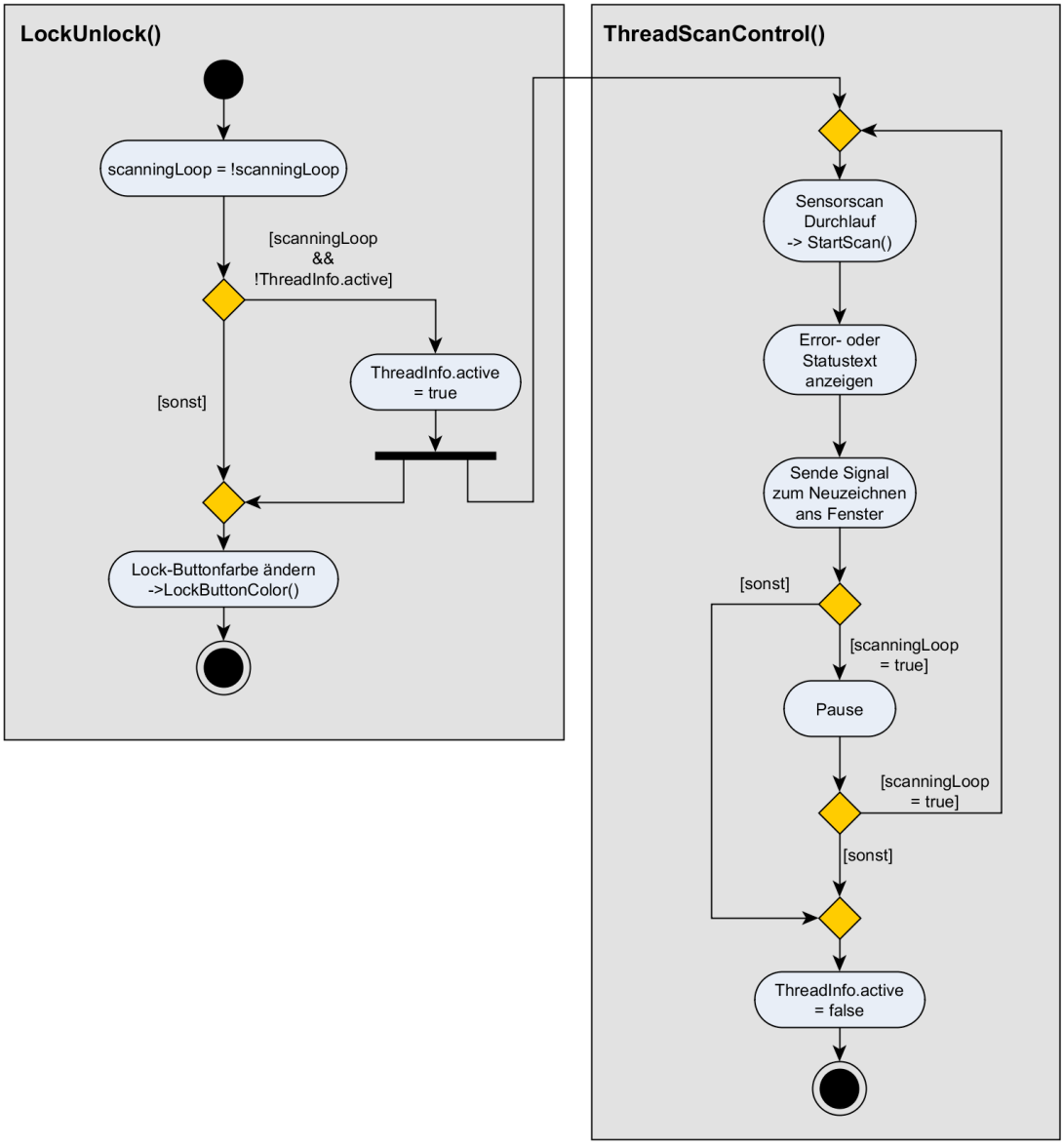

Abbildung 6.5: Aktivitätsdiagramm Threaderstellung

Es wurde entschieden, dass der Prozess des Datenabrufens aus dem Sensor mit anschließender Rohdatenaufarbeitung in einem zusätzlichen Thread laufen soll. Somit kann die Erzeugung neuer Daten parallel zum restlichen Programm ablaufen.
Der Zustand, ob die kontinuierliche Messung gerade aktiv ist, wird in dem Attribut *scanningLoop* (Tab. 6.1) der Klasse MAIN_CLASS als Boolean abgespeichert.

scanningLoop	**Zustand**
false	kontinuierliche Messung deaktiviert
true	kontinuierliche Messung aktiv

Tabelle 6.1: Flag scanningLoop mit zugehörigem Zustand

Die Information, ob der zugehörige Thread zum Sensor aktiv ist, wird in der Struktur *Threadinfo* (Tab. 6.2) als boolesche Variable abgespeichert. Würde nur *scanningLoop* abgefragt werden, könnte der Benutzer durch doppeltes Klicken auf „Lock“ einen zweiten Thread, zusätzlich zu dem Ersten, starten, wenn der erste das kurzzeitige *scanningLoop = false* nicht mitbekommt.
Befindet sich der Thread ThreadScanControl innerhalb der Schleife und der Benutzer klickt das erste Mal auf „Lock“, so wird *scanningLoop* auf false gesetzt. Beim zweiten Klick auf „Lock“ ist *scanningLoop* zwar false, aber der Thread hat sein *activ* Flag noch nicht auf false gesetzt. Deshalb bleibt *scanningLoop* auf false und es wird gewartet, bis der Thread geschlossen wurde. Erst danach kann der Benutzer einen neuen Thread starten. Dies hat den Hintergrund, damit nicht doppelt auf einen Sensor zugegriffen und von zwei Threads in denselben Datenraum geschrieben wird.

Threadinfo.activ	**Zustand**
false	Thread läuft
true	Thread beendet

Tabelle 6.2: Flag Threadinfo.activ mit zugehörigem Zustand

Zuletzt wird in *LockUnlock()* die Methode *LockButtonColor()* aufgerufen, die die Buttonfarbe der „Lock“ Button setzt, abhängig vom Zustand, der durch *scanningLoop* beschrieben wird. Das Setzen der Buttonfarbe dient als Indikator/Feedback für den Anwender, damit dieser erkennt, ob die kontinuierliche Messung aktiv ist.

Thread starten

Code 6.3 zeigt die Methode *LockUnlock()*. Mittels *_beginthread()* wird der Prozess *ThreadScanControl()*, der die kontinuierliche Messung regelt, in einem neuen Thread gestartet. Da neuen Threads nur ein Pointer als Parameter übergeben werden darf, wurde hier ein Zeiger auf die Struktur ThreadInfo übergeben.

Durch die Erzeugung eines neuen Threads für den Scanprozess ist das Programm in der Lage auf Real-Multithreadingsystem parallel zu arbeiten. Somit kann die grafische Anzeige der Sensordaten und die Beschaffung neuer Daten vom Sensor gleichzeitig erfolgen.

```
1 void MAIN_CLASS::LockUnlock() {
2         scanningLoop = !scanningLoop;
3         if (scanningLoop && !ThreadInfoTIM.active)
4         {
5                 ThreadInfoTIM.active = true;
6                 _beginthread(ThreadScanControl, 0, &ThreadInfoTIM);
7         }
8         LockButtonColor();
9 }
```

Code 6.3: Methode LockUnlock()

An den Thread übergebene Informationen

Die Struktur *ThreadInfo* 6.4 beinhaltet einmal einen leeren Zeiger, einen Zeiger auf ein SensorCom Objekt und das Thread-activ-Flag, welches mit *false* initialisiert wird. Auf dem leeren Zeiger wird die Adresse des Programmfensters (Objekt der Klasse MAIN_CLASS) gespeichert und auf dem SensorCom Zeiger die eines zugehörigem Objektes. Somit bekommt der Thread *ThreadScanControl()* bei der Übergabe dieser Struktur eine Verbindung zum Programmfenster und zu einem Sensorobjekt mitgeteilt.

```
1 struct ThreadInfo {
2         void *mc;
3         SensorCom *sen;
4         bool active = false;
5 };
```

Code 6.4: Struktur ThreadInfo

Thread-Ablauf

```
1  void ThreadScanControl(void *data)
2  {
3          // Casten der Zeiger
4          ThreadInfo* ti = static_cast<ThreadInfo*>(data);
5          MAIN_CLASS* mc = static_cast<MAIN_CLASS*>(ti->mc);
6
7          do {
8                  // Einen Messdurchgang starten
9                  ti->sen->StartScan();
10                 // Error oder Statustext abfangen
11                 // und anzeigen
12                 mc->DisplayError(ti->sen->GetError());
13                 // Message MW_PAINT an MAIN_CLASS senden
14                 mc->Invalidate();
15                 if (mc->GetScanningLoop())
16                 {
17                         // Thread pausieren
18                         Sleep(420);
19                 }
20         } while (mc->GetScanningLoop());
21
22         ti->active = false;
23         _endthread();
24 }
```

Code 6.5: Thread *ThreadScanControl()*

Zunächst muss der Inhalt des leeren Zeigers auf die Struktur *ThreadInfo* und dessen Inhalt *mc* auf die Klasse MAIN_CLASS gecasted werden.
Mit einer do-while-Schleife wurde die kontinuierlich Messung implementiert. Innerhalb der Schleife wird mittels *StartScan()* eine Messung durchgeführt und im Anschluss per *Invalidate()* der CDialog Klasse eine Message gesendet, die das Neuzeichnen auslöst. Zwischen den Messungen wird der Thread für 420ms pausiert, somit erfolgen pro Sekunde ungefähr zwei Messungen. Wird das Flag *scanningLoop* auf false gesetzt wird dies durch die public Methode *GetScanningLoop()* der Klasse MAIN_CLASS abgefragt und die do-while-Schleife wird beendet.
Anschließend wird die Information, dass der Thread nicht mehr aktiv ist abgespeichert und durch *_endthread()* beendet.
Nach jeder Messung wird mit *GetError()* der Fehler- oder Statustext der Messung abgerufen und mit *DisplayError()* auf die Programmoberfläche geschrieben. Weitere Erklärung dazu im Abschnitt 6.6 Errorhandling.

6.4.2 Parallel Sensordaten speichern und lesen

Durch das Multithreading kann zum Beispiel die grafische Darstellung der Sensorwerte, gleichzeitig mit dem Abrufen neuer Werte erfolgen. Damit die Sensorwerte sich nicht in dem Moment ändern während sie dargestellt oder anderweitig gelesen werden, muss der Zugriff geregelt werden.

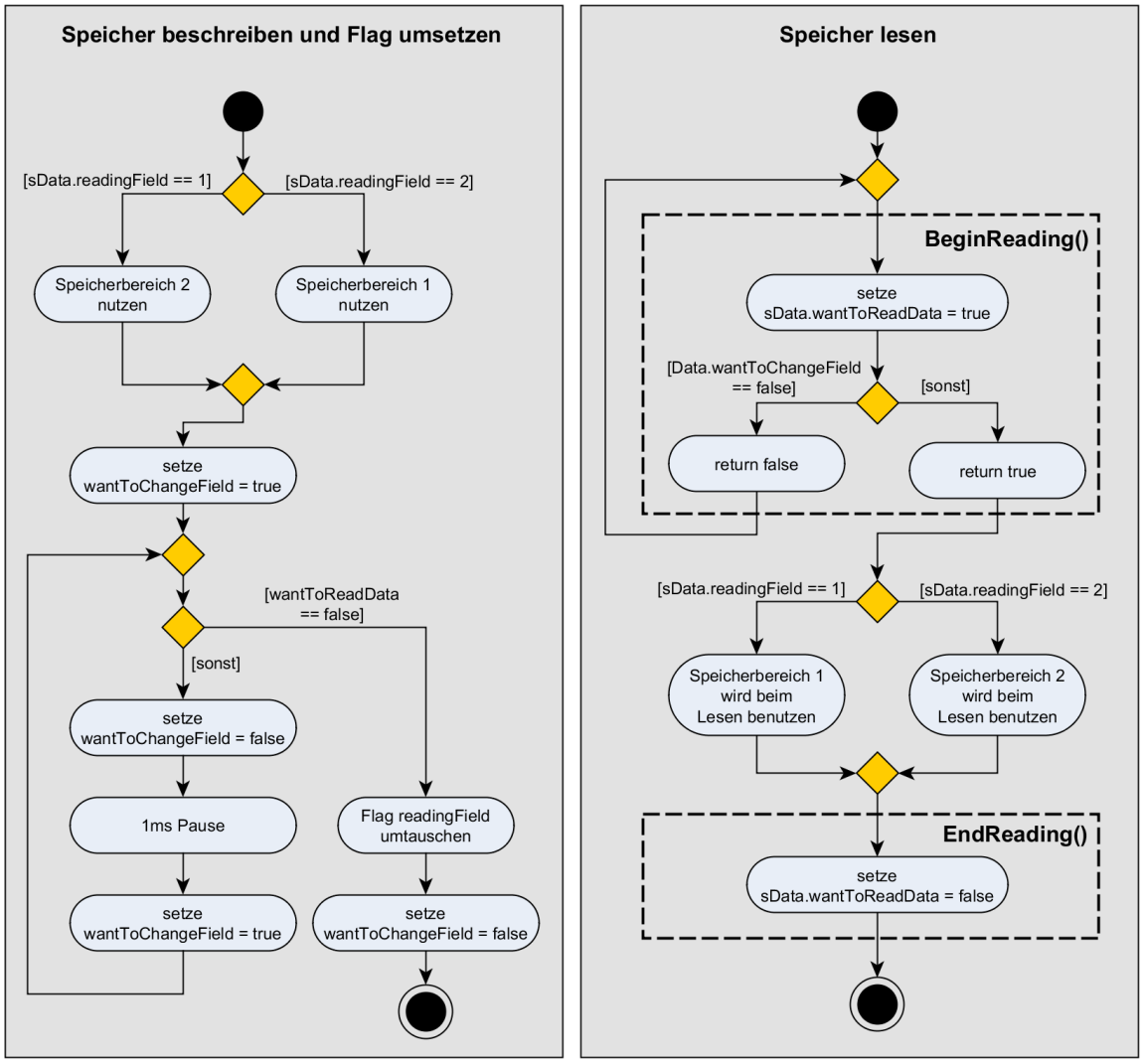

Abbildung 6.6: Aktivitätsdiagramm Speicherzugriffsverwaltung

Es wurde entschieden für die Sensorwerte zwei Speicherfelder zu benutzen, in die abwechselnd neue erzeugte Daten abgespeichert werden. Mittels einer Flag-Variablen (*Da-*

ta.readingField) wird den Threads signalisiert aus welchem Speicherfeld gelesen, und in welches geschrieben werden kann. Diese Methode verringert die Wahrscheinlichkeit, dass ein Vorgang auf den anderen warten muss und dadurch können beide Threads parallel ablaufen. Der einzige Konfliktzeitbereich besteht nun nur noch, während die Flag-Variable umgesetzt wird und gleichzeitig diese abgefragt wird.

Datenzugriff steuern

Der SensorCom Klasse wurden zwei Methoden gegeben, die das Beginnen und Enden des Datenlesens einleiten. Diese beiden Methoden setzen das Flag *wantToReadData* auf jeweils true oder false. Der Prozess, in der Methode *StartScan()*, erkennt an diesem Flag, dass das Datenlesen noch aktiv ist und setzt die Variable, die den Speicherbereich mit den neueren Daten definiert erst dann um, wenn das Flag auf false gesetzt wurde (Code 6.6).

```
1  Data.wantToChangeField = true;
2  // Gegebenfalls warten bis Leseprozess endet:
3  while (Data.wantToReadData) {
4          Data.wantToChangeField = false;
5          Sleep(1);
6          Data.wantToChangeField = true;
7  }
8  // Flag readField umsetzen
9  if (Data.readingField == 1) {
10         Data.readingField = 2;
11 }
12 else {
13         Data.readingField = 1;
14 }
15 Data.wantToChangeField = false;
```

Code 6.6: Flag readingField umsetzen in SensorCom::StartScan() nachdem Sensordaten neu abgespeichert wurden

Durch unglückliche Verteilung der Rechenzeit könnte es passieren, dass der Leseprozess *wantToReadData* umsetzt und schon Daten liest, nachdem dieses vom Schreibprozess ausgelesen, aber das Flag *readingField* noch nicht umgesetzt wurde. Um das zu verhindern setzt der datenschreibende Prozess das Flag *wantToChangeField* vor dem Auslesen von *wantToReadData*. Somit hat jeder Prozess sein Flag, welches seine Absicht wiederspiegelt. Der Rückgabewert von *BeginReading()* ist false, wenn *wantToChangeField* auf true gesetzt wurde. Der Leseprozess ist priorisiert, da in der Programmierung auf den Rückgabewert und somit auf *wantToChangField* gepollt wird (Code 6.7), der schreibende Prozess seine

Absicht *readingField* zu ändern aber kurzzeitig zurück nimmt und sich pausiert bevor er es erneut versucht. In dieser Pause kann der pollende Leseprozess weiterlaufen und das Flag *readingField* wird erst nach Ausführen von *EndReading()* umgesetzt.

```
1 // Pollen auf Rückgabewert von BeginReading()
2 // bis Variable wantToChangeField false ist
3 while (!TIM.BeginReading()) { ; }
```

Code 6.7: Vor dem Abfragen von Sensordaten aus SensorCom BeginReading() auswerten

Struktur *Data*

Ein Objekt der Klasse SensorCom beinhaltet ein Attribut *Data* der folgenden Struktur *sData*. In dieser Struktur sind die Speicherfelder und Steuervariablen, auf die zugegriffen wird, abgelegt.

```
1  struct sData {
2  // Datenfelder für Rohsensordaten im Polarkoordinaten
3          int Data1[2000];
4          int Data2[2000];
5  // Variable für die Anzahl der Sensordaten
6          int lenghtData1 = 2000;
7          int lenghtData2 = 2000;
8  // Datenfelder für umgerechnete Sensordaten im kartesischen Koordinaten
9          double xData1[2000];
10         double xData2[2000];
11         double yData1[2000];
12         double yData2[2000];
13 // Variablen der Tisch zu Sensor Distanz
14         double groundDistance1;
15         double groundDistance2;
16 // Variablen der Objekthöhe
17         double objectHeight1;
18         double objectHeight2;
19 // Flag Variable die das zu lesende Feld definiert
20         int readingField = 0;
21 // Flags für den sicheren Datenzugriff
22         bool wantToChangeField = false;
23         bool wantToReadData = false;
24 };
```

Code 6.8: Struktur mit jeweils zwer Vektoren für die Sensordatem

Im folgenden Text wird die Zahl 1 oder 2 an den Variablenenden durch ein x ersetzt. Es wird immer Bezug auf das, für den Prozess, aktive Feld genommen.

6.4.3 Sensorkommunikation und -datenverarbeitung

In der Thread *ThreadScanControl()* wird die Methode *StartScan()* des Sensorobjekts SensorCom aufgerufen.
StartScan() liest die aktuellen Werte aus dem Sensor aus, bereitet diese auf und speichert sie ab. Der grobe Ablauf ist wie folgt:

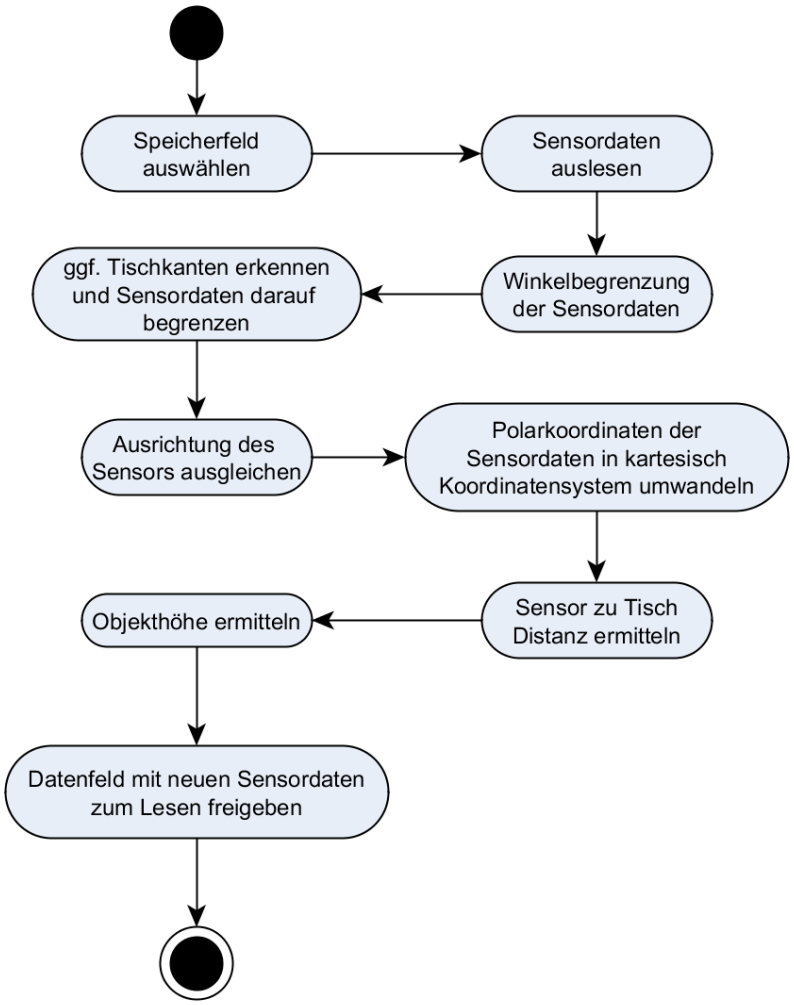

Abbildung 6.7: Aktivitätsdiagramm der *StartScan()* Methode von SensorCom

Speicherfeld auswählen

Es werden Zeiger generiert, die auf die jeweiligen Datenfelder und Variablen zeigen, auf die gerade nicht lesend zugegriffen werden kann. Dafür wird *Data.readingField* ausgewertet.

Sensordaten auslesen

Um die Sensordaten auszulesen wird eine Socketverbindung aufgebaut. Dafür wird die IP-Adresse und der Port aus den Attributen *ipAdresse* und *tcpPort* benutzt. Über diese Schnittstelle wird dem Sensor mitgeteilt, dass er eine Messreihe zurücksenden soll.
Der Sensor antwortet mit einer Zeichenfolge, die die Messwerte in hexadezimaler Schreibweise getrennt durch Leerzeichen beinhaltet. Die einzelnen Messwerte werden als Zahl extrahiert und gespeichert.
Insgesamt fünfmal werden Daten aus dem Sensor ausgelesen. Am Ende wird der Durchschnittswert genommen und in *Data.Datax* abgespeichert. Durch den Durchschnitt von fünf Messungen werden Messabweichungen reduziert.

Winkelbegrenzung der Sensordaten

Der Sensor hat ein Sichtfeld von 270°, welches an den Rändern entbehrliche Informationen enthält. Da eine Röntgenanlage auch einen kleineren Winkelbereich der austretenden Röntgenstrahlung hat, wird es auf 80° mittig beschränkt. Dafür wird der Anfang dieses Bereiches als Adresse des ersten Wertes abgespeichert und die Länge des Bereiches unter *lenghtDatax* (Anzahl der Messwerte) abgespeichert.

Datenbereich auf Tischoberfläche begrenzen

Nun kann es sein, dass in dem 80°-Sichtfeld Messpunkte außerhalb des Tischbereiches liegen. Es wird versucht die Tischkanten zu erkennen und den Bereich durch anpassen des Anfangswert und der Länge daran anzugleichen. Dafür werden von der Mitte ausgehend zu jedem Ende des Wertebereiches immer sechs Werte verglichen. Ist der Durchschnitt der drei außerhalb liegenden Werte 50 cm größer als der inneren, wird dieser Sprung als Tischkante zu Boden Sprung angenommen. Im Anschluss werden die äußeren Messwerte ausgeschlossen, indem der Zeiger auf den Anfangswert neu gesetzt wird und die Anzahl der Messwerte gekürzt wird.
Die Begrenzung auf maximal 80° hat den Effekt, dass bei einer kurzen Messdistanz für kleine Objekte weniger leere Tischoberfläche bis zu den Rändern in der weiteren Verarbeitung mit vorkommt.

Ausrichtung des Sensors ausgleichen

Folgende Probleme können auftreten:

1. Der Sensor ist leicht gedreht und somit befindet sich die senkrechte 0°-Achse nicht in der Mitte des Messwertefeldes.
2. Der Sensor befindet sich nicht direkt über der Mitte des Tisches. Bei erkannten Tischkanten und der Reduzierung des Messwertfeldes auf dessen Oberfläche ist die senkrechte 0°-Achse auch nicht in der Mitte des Messwertefeldes, beziehungsweise nicht zu erkennen.
3. Die Kombination aus Fall 1 und 2.

Um dieses Problem zu lösen, werden an den Seitenbereichen von der jeweils entfernteren Seite Messwerte aus dem betrachteten Bereich herausgenommen, bis der Durchschnitt von jeweils fünf äußeren Werten auf 3 mm genau der gegenüberliegenden Seite entspricht. Damit gehen am Rand Messwerte verloren, die nur Werte der Tischoberfläche sind. Voraussetzung für diesen Vorgang ist, dass an den Rändern genug Tischoberfläche im Wertebereich ist und nicht von einem Messobjekt gänzlich abgedeckt wird.

Kartesisches Koordinatensystem erzeugen

Durch den vorherigen Schritt ist in der Mitte des Datenfeldes die Senkrechte zum Tisch und alle Werte sind um 0.33° weiter ausgerichtet. Jeder Messwert wurde per Winkelsatz ins kartesische Koordinatensystem umgewandelt. Die senkrechte und die horizontale Position werden in jeweils ein Datenfeld (*xDatax* & *yDatax*) abgespeichert. Der Mittelpunkt des Koordinatensystems (0,0) ist der Sensorort.

Abbildung 6.8 auf S.34 zeigt eine 2D Raumsimulation, in dem sich der Sensor in 140 cm Höhe über einem Tisch, mit einem darauf liegenden Objekt, befindet. Die rote Fläche zeigt das Messfeld des Sensors, begrenzt auf 80°.
Abbildung 6.9 auf S.34 zeigt das simulierte Messwertprofil des Sensors bei einem 80°-Sichtfeld.
Gut zu sehen ist, dass sich grade Flächen gekrümmt darstellen, da diese gemessene Entfernung relativ zu dem Sensor ist.

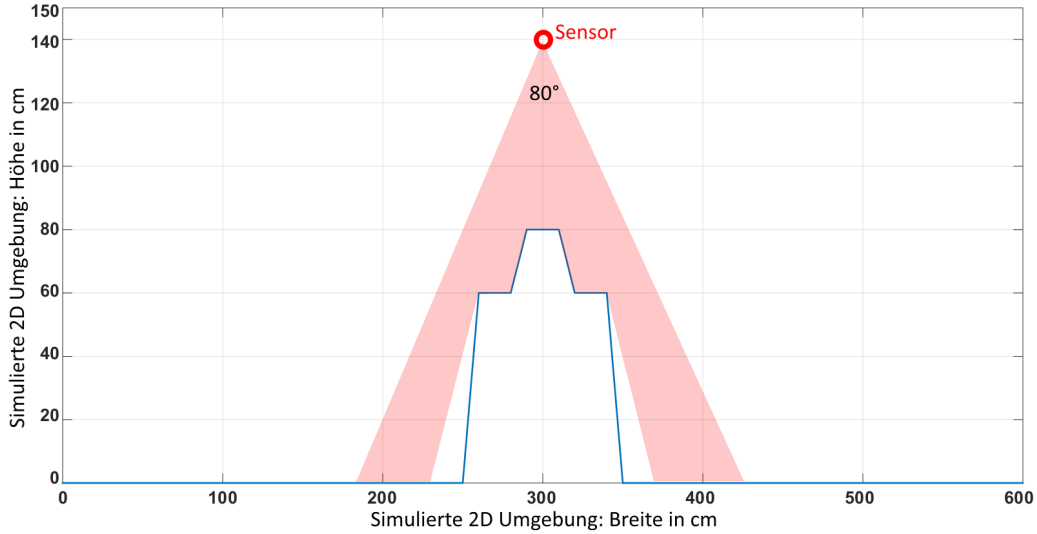

Abbildung 6.8: Simulierte 2D Umgebung mit Sensormessung auf 80° reduziert

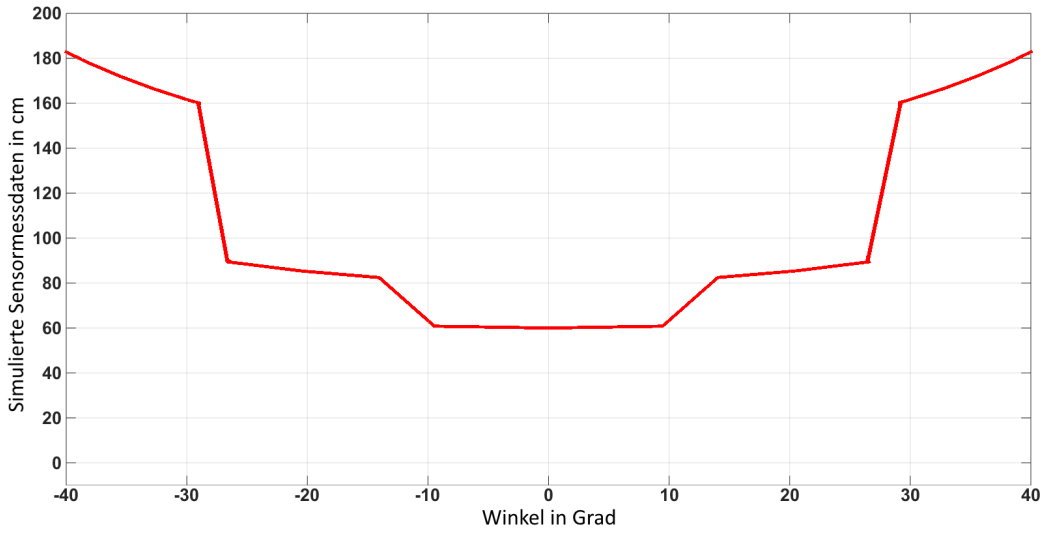

Abbildung 6.9: Simuliertes Sensormessdatenbild der 2D Umgebung auf 80° reduziert

Sensor-zu-Tisch-Distanz ermitteln

Die Sensor-zu-Tisch-Distanz wird aus dem Durchschnitt von jeweils fünf Randwerten der Höhen-Messwerte auf jeder Seite des betrachteten Bereiches im kartesischem Koordinatensystems ermittelt.

Objekthöhe ermitteln

Um die Objekthöhen zu berechnen müssen die Messwerte relativ zur Tischoberfläche betrachtet werden. Im kartesisch Koordinatensystem werden dafür von den Messpunkten die Tisch-zu-Sensor-Distanz abgezogen. Es wurde entschieden, dass die Objekthöhe, das 0,8-Perzentil aller Höhenwerte, die über 5 mm hoch sind, ist. Dadurch, dass nicht der größte Messwert die Objekthöhe bestimmt, können Ausreißer den Wert nicht verfälschen. Die 5 mm Grenze verhindert zudem, dass die Tischoberfläche nicht auch schon als erkanntes Objekt in die Betrachtung fällt.

Neue Daten zum Lesen freigeben

Mit dem Umsetzen des Flag Data.readingField(Siehe S.29 Code 6.6) werden die Datenfelder mit den neuen Sensormessdaten zum Lesen freigegeben.

6.4.4 Grafische Darstellung

Nach jedem Sensorscandurchgang sendet der Thread *ThreadScanControl()* mittels CWnd-Methode *Invalidate()* das WM_PAINT Signal zur CWnd-Klasse MAIN_CLASS, die daraufhin die Funktion *OnPaint()* startet. (Siehe S.27 Code:6.5)

In *OnPaint()* (Siehe S.36 Abb.6.10) wird darauf hin die grafische Darstellung der Sensorwerte in einem Koordinatensystem über den Bereich des Picture Control Feld gezeichnet.
Im Programmentstehungsverlauf stellte sich heraus, dass das Neuzeichnen der Oberfläche bei aufwendigen Berechnungen innerhalb *OnPaint()* ein Flackern verursacht. Das kommt daher, dass zunächst der Fensterinhalt automatisch mit dem Hintergrund überdeckt wird, und darauf alle sichtbaren Objekte gezeichnet werden. Nun kann es passieren, dass das Zeichnen noch nicht fertig ist, bevor das Computersystem die Displayausgabe aktualisiert und eine nicht fertige Programmoberfläche wird kurzzeitig sichtbar.

Um das Problem zu beheben wurde das automatische Neuzeichnen des Hintergrundes deaktiviert. Dafür wurde die CWnd-Methode *OnEraseBkgnd()* überschrieben, die die automatische Hintergrundneuzeichnung steuert. Bei gleichbleibenden Objekten, wie es bei den Buttons die meiste Zeit ist, tritt somit kein Flackern mehr auf, da diese übereinander gezeichnet werden.
Da aber beim Ändern der Größe des Programmfensters die Objekte auf der Oberfläche nicht mehr an der gleichen Stelle sind, muss in dem Fall nach einer Größenänderung der Hintergrund neu gezeichnet werden. Dafür wurde ein Flag *repaintBKG* eingefügt. Die überschriebene CWnD-Methode *OnSize()* wird automatisch bei Fenstergrößenänderungen aufgerufen

und setzt dieses Flag auf true. Daraufhin wird per *Invalidate()* eine Neuzeichnung der Oberfläche erwirkt. In *OnPaint()* wird, falls *repaintBKG* auf true gesetzt wurde, der Hintergrund neu gezeichnet.

Um nun noch zu verhindern, dass die Oberfläche angezeigt wird bevor die Zeichnung des Koordinatensystems mit den Messwerten fertig ist, wurde die Zeichnung gebuffert. Für die Messwertanzeige wird die Größe und Position von dem Control Picture *visualbox* genommen und eine gleichgroße Bitmap erstellt, in die gezeichnet wird. Das Gleiche passiert auch für den Hintergrund. Ist die Bildberechnung fertig, wird am Schluss der Hintergrund und die Messwertanzeige jeweils in einem Stück auf die Oberfläche geladen.

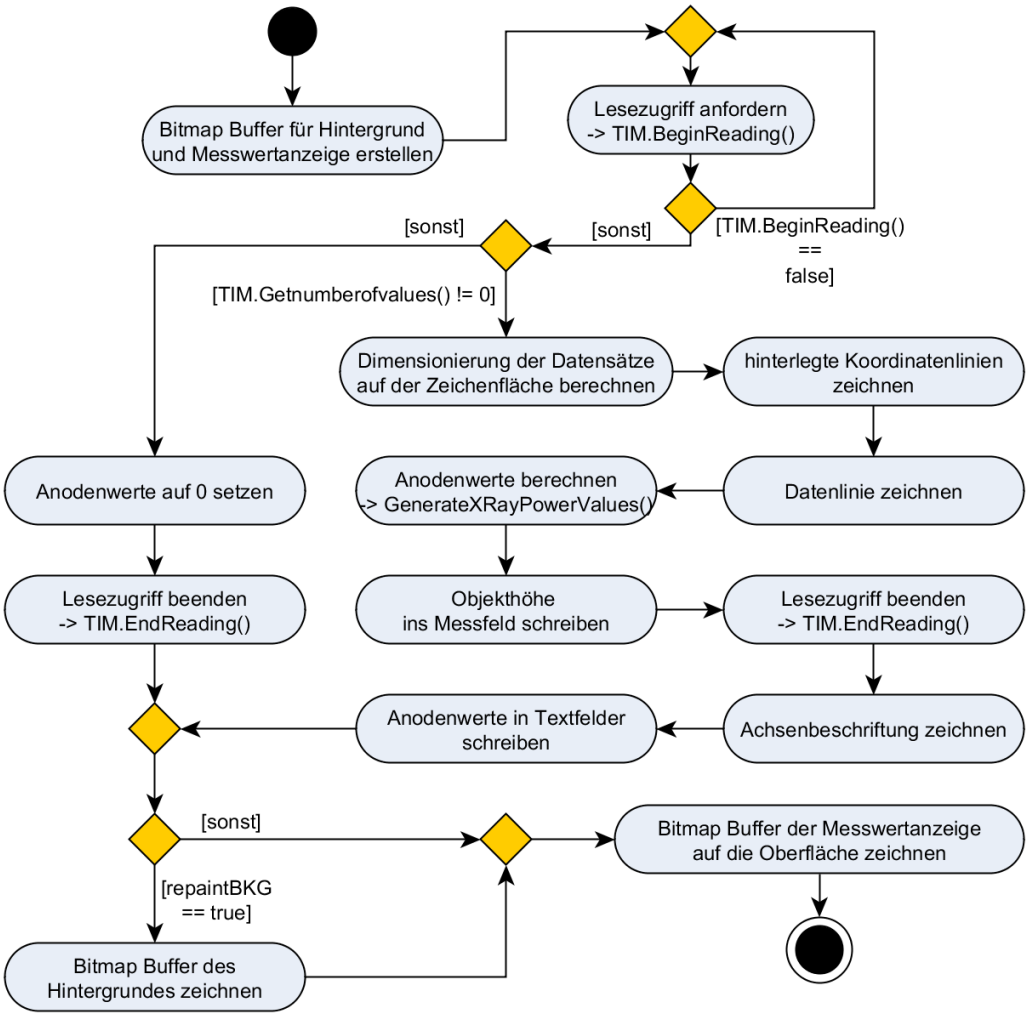

Abbildung 6.10: Aktivitätsdiagramm der *OnPaint()* Methode mit der Visualisierung der Messdaten

6.4.5 XML Auslesen und Erzeugung der Anodenwerte

Um die Anodenwerte aus der Objekthöhe ableiten zu können, sind in einer XML-Datei Punkte aus einer Verlaufskurven der optimalen Werte abgespeichert. Die Datei beinhaltet eine variable Anzahl an Punkten für maximal drei Programme, die über den jeweiligen Auswahl Button bei der Anodenwertberechnung zur Berechnung benutzt werden.

Die Informationen aus der XML-Datei werden im Programm mit der Methode *LoadXMLDATA()* verarbeitet. Dafür wird auf die frei verfügbare TinyXML-2 Bibliothek als XML-Parser zurückgegriffen. Diese erzeugt aus dem XML-Inhalt eine C++ Objekt, aus dem die einzelnen Informationen herausgelesen werden, um diese weiter verarbeiten zu können.

Die XML-Datei muss im Programmordner platziert sein und Settings.xml heißen. Der Aufbau ist folgender:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <XRCP2017>
4    <XRAY-PROGRAMS>
5      <XRAY-CURVE ButtonID="1" Name="Knochen">
6        <!-- Height[cm]  Voltage[kV]  Current[mA] Time[ms] -->
7        <VALUE Height="5" Voltage="45" Current="200" Time="20"/>
8        <VALUE Height="10" Voltage="55" Current="500" Time="50"/>
9        <VALUE Height="20" Voltage="65" Current="1000" Time="100"/>
10       <VALUE Height="30" Voltage="80" Current="1000" Time="200"/>
11       <VALUE Height="40" Voltage="95" Current="1000" Time="500"/>
12     </XRAY-CURVE>
13     <XRAY-CURVE ButtonID="2" Name="Gewebe">
14       <VALUE Height="5" Voltage="40" Current="200" Time="30"/>
15       <VALUE Height="10" Voltage="50" Current="500" Time="70"/>
16       <VALUE Height="20" Voltage="65" Current="1000" Time="150"/>
17       <VALUE Height="30" Voltage="75" Current="1000" Time="300"/>
18       <VALUE Height="40" Voltage="90" Current="1000" Time="700"/>
19     </XRAY-CURVE>
20     <XRAY-CURVE ButtonID="3" Name="Lunge">
21       <VALUE Height="5" Voltage="60" Current="200" Time="10"/>
22       <VALUE Height="10" Voltage="70" Current="500" Time="20"/>
23       <VALUE Height="20" Voltage="90" Current="1000" Time="50"/>
24       <VALUE Height="30" Voltage="120" Current="1000" Time="100"/>
25       <VALUE Height="40" Voltage="150" Current="1000" Time="200"/>
26     </XRAY-CURVE>
27   </XRAY-PROGRAMS>
28   <SENSORSETTINGS IP="169.254.20.20" Port="2112"/>
29 </XRCP2017>
```

Code 6.9: Beispiel Settings.xml

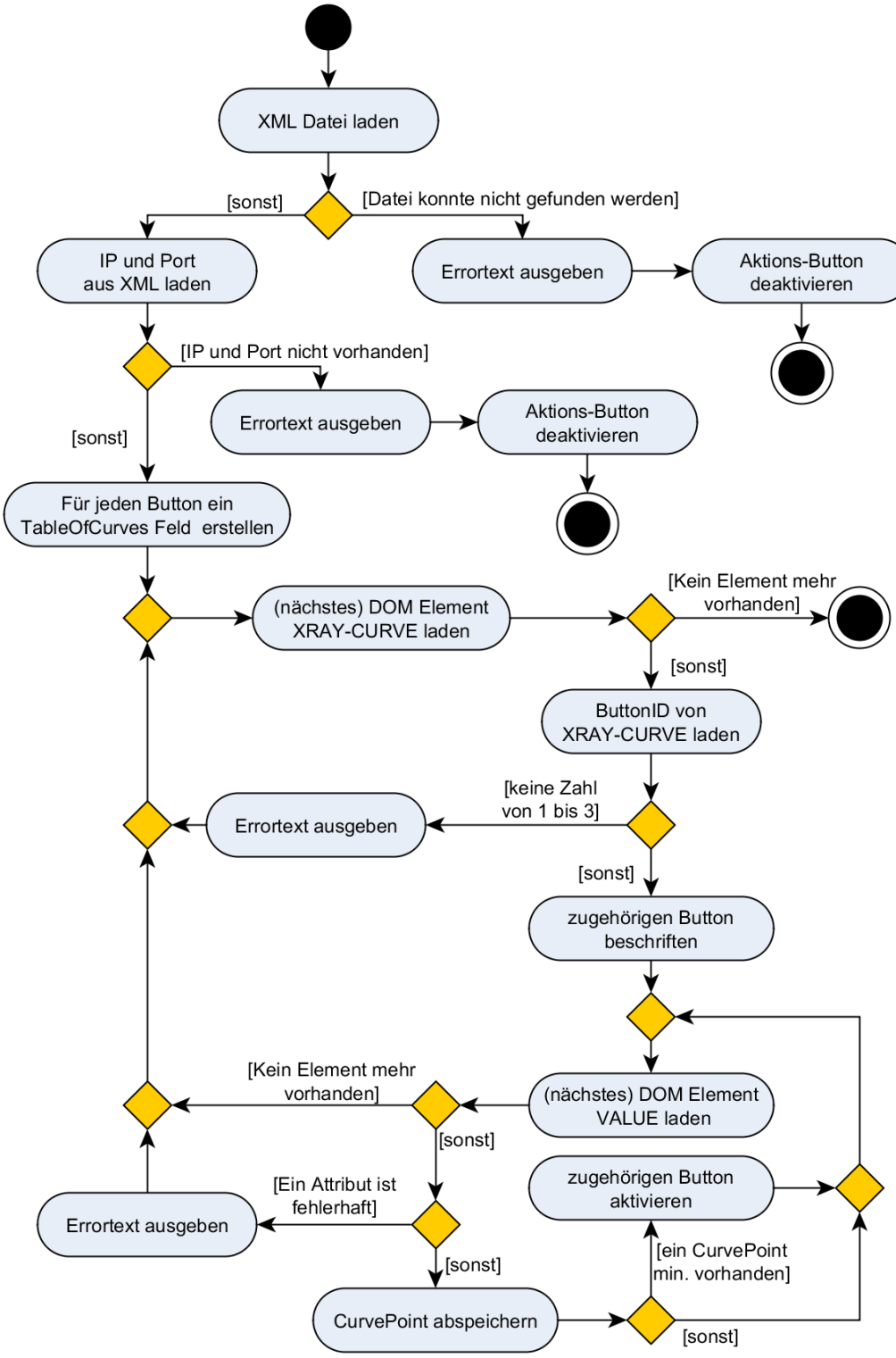

Abbildung 6.11: Aktivitätsdiagramm der Methode *LoadXMLDATA()*

Das XML-Hauptelement ist *<XRCP2017>*. Die einzelnen Kurven sind im Unterelement *<XRAY-PROGRAMS>* als Element *<XRAY-CURVE>* abgespeichert. *<XRAY-CURVE>* hat als Attribut einen *Namen* und die *ButtonID*, die die Platzierung auf dem jeweiligen Auswahl-Button darstellt. Der Inhalt des Attributs *Name* wird in den Buttontext des jeweiligen Buttons geladen. Einzelne Kurvenpunkten sind Unterelemente des Elements *<XRAY-CURVE>* und heißen *<VALUE>*. Die Attribute von *<VALUE>* sind die Information, bei welcher Objekthöhe welche Spannung, Strom und Zeit die perfekte Einstellung für den Anodenbetrieb sind.

In der Methode *LoadXMLDATA()* werden die Kurven in Felder abgespeichert. Dies wurde im Vektor-Element TableOfCurves realisiert. Die Anzahl an Kurvenpunkten ist nicht begrenzt.

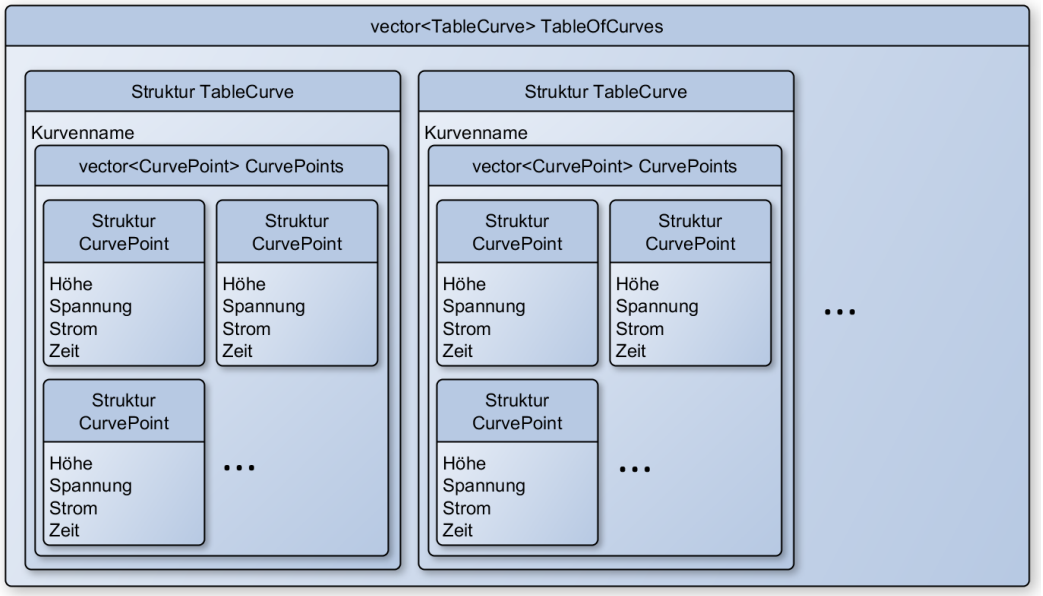

Abbildung 6.12: Aufbau der Speicherstruktur der Ausgangskurven

Röntgenanodenwerte erzeugen

Die Methode *GenerateXRayPowerValues()* wird in der Methode *OnPaint()*, und falls der User die Programmkurven-Auswahl ändert, in der Handlermethode der Auswahl-Button aufgerufen. Diese Methode leitet aus der Objekthöhe und den Kurvenpunkten die Parameter für die Röntgenanode ab. Dabei werden durch lineare Interpolation Kurvenpunkte zwischen den angegebenen Kurvenpunkten berechnet. Damit die Berechnung korrekt abläuft müssen die Kurvenpunkte nach der Höhe gestaffelt in der XML-Datei abgespeichert sein.

6.5 Programmsteuerung

Um das Programm effizienter steuern zu können und die Bedienerfreundlichkeit zu steigern, wurde die Steuerung per Tastatureingabe hinzugefügt. Dafür wurde die Methode *PreTranslateMessage()* der CWinApp-Klasse überschrieben. Diese Methode fängt alle Fenster Messages ab, bevor diese an die verarbeitende Methode *TranslateMessage()* gesendet werden. Liegt der Fokus auf der Anwendung und der Benutzer drückt eine Taste, wird dies in Form einer Message an das Programm geleitet. In der überschrieben Methode *PreTranslateMessage()* werden alle Message, die nicht durch Tastenbetätigung ausgelöst wurden, an die Standardmethode von *PreTranslateMessage()* weitergeleitet. Entsprechen die Messages definierte Tastenbetätigungen, werden entsprechende Aktionen ausgeführt.
In Abbildung 6.13 ist rot markiert, welche Tastendrücke sich auf welche Programmfunktion auswirken.

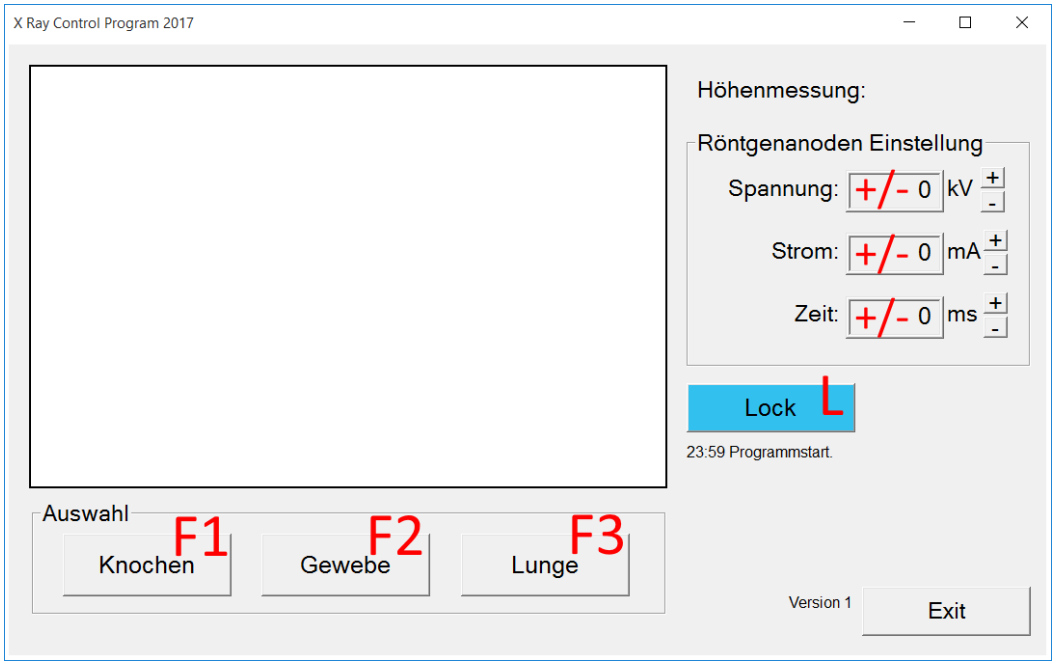

Abbildung 6.13: Programmoberfläche mit rot eingeblendeten Tastensteuerungen

Die Tasten **F1, F2 und F3** haben die gleiche Funktionsweise, wie das Betätigung der jeweiligen Auswahlbuttons. Durch die **L-Taste** kann die kontinuierliche Messung gestartet und gestoppt werden. Ist ein jeweiliges Textfeld der Röntgenanodeneinstellung vom Benutzer fokussiert worden, kann per **+ oder -** der Wert gesteigert oder gesenkt werden. Die Werte werden jeweils in drei Wertebereichen unterschiedlich stark gesteigert oder gesenkt. Durch

die **Tab-Taste** kann der Benutzer leicht die einzelnen Felder der Röntgenanoden Einstellung anwählen und wechseln.
Mit den Button „+“ und „-“, neben den einzelnen Röntgenanodenparametern, kann der Benutzer auch den jeweiligen Wert steigern oder senken.

6.6 Errorhandling

Da es sich bei dem Programm um keine Endanwenderanwendung handelt, wurde eine Textbox unterhalb des „Lock“-Button erstellt, die bestimmte Vorkommnisse protokolliert. Zum Start des Programmes schreibt das Programm die Systemzeit mit dem Text „Programmstart“ in die Textbox. Bei einem Fehler, der in der Programmierung abgefangen wurde, wird ein Fehlertext erstellt und mit der Methode *DisplayError()* angezeigt. Treten während des Verlaufes Fehler auf, dann werden die letzten sieben in dieses Feld geschrieben. Sind die beiden letzten anzuzeigenden Texte identisch, wird nur die angezeigte Zeit aktualisiert. Fehler können zum Beispiel sein, dass die XML-Datei nicht gefunden wurde oder die Verbindung zum Sensor nicht aufgebaut werden kann. Werden vom Sensor erfolgreich Daten empfangen wird dies auch protokolliert.

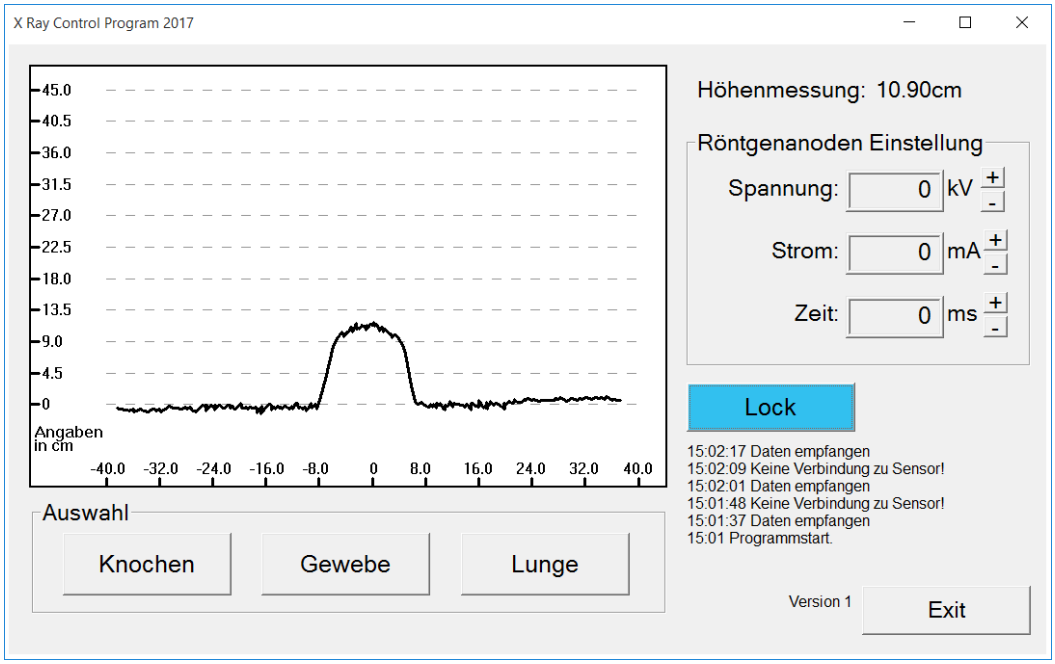

Abbildung 6.14: Programmoberfläche mit Errortext Sensorverbindungsfehler

Kann die XML-Datei nicht gefunden werden, werden außerdem die Auswahl- und der „Lock“-Button gesperrt, da keine Berechnung der Röntgenanodenwert möglich ist.

6.7 Hardwareaufbau

6.7.1 Sensorinbetriebnahme

Der Sensor TIM571 hat jeweils einen Gewindesteckeranschluss für die Stromversorgung und einen für die Datenübertragung. Mitgeliefert werden passende Kabel. Das Stromversorgungskabel ist vierpolig und am Ende sind die Kabelstränge offen aus dem Kabelmantel herausgeführt. Das Datenkabel hat am anderen Ende einen Ethernetstecker.
Für die Stromversorgung wurde ein handelsübliches 12VDC-Netzteil mit Hohlstecker verwendet. Die zugehörigen Kabelstränge von dem Stromversorgungskabel des Sensors, wurden mit einem Adapter auf die passende Buchse gebracht. Die Kabelenden werden mit zusammen schraubbaren Klemmen am Adapter fixiert, und sind so auch leicht wieder zu demontieren.

Abbildung 6.15: Laserinfrarotsensor TIM571 (Rückansicht)

Bild 6.15 Beschreibung: Links: Abgedeckte Gewindesteckeranschlüsse; Mitte(grau): Hauptgehäuse; Rechts(schwarz): Optikhaube mit sich darunter befindlichem drehbaren Spiegel des Laserinfrarotsensor.

Um den Sensor betriebsbereit zu machen muss nun die Stromversorgung hergestellt und das angeschlossene Ethernetkabel in den entsprechenden PC gesteckt werden. Nun muss am PC die Konfigurationssoftware SOPAS der Firma Sick gestartet werden. Die Software sucht danach den Sensor im Netzwerk. Danach kann in der Software dem Sensor eine IP-Adresse zugewiesen werden und dieser ist nun unter der eingestellten IP-Adresse ansprechbar.

6.7.2 Demonstratoraufbau

Die Aluprofile wurden in zwei 2,5m langen Stücken bestellt, die in der Werkstatt der Hochschule in passende Längen gesägt und die Enden manuell entgratet wurden. Danach wurden die Stücke mit mitbestellten Verbindungswinkeln und -haken zusammengebaut.
Die Verbindung entsteht, indem Nutensteine, die ein Loch mit Gewinde besitzen, in die Profilrillen gelegt werden. Nun kann zum Beispiel ein Winkelstück auf das Profil gelegt werden und eine Schraube greift durch ein Loch in das Gewinde des Nutensteins. Beim Festziehen zieht die Verbindung sich immer weiter zusammen. Der Halt entsteht durch den hohen Anpressdruck beider Stücke aufeinander.

Um den Ausleger höhenverstellbar zu machen, wurden die Verbindungsschrauben zu der senkrechten Hauptstange nicht vollständig festgezogen. In den Nutensteinen ist eine kleine Kugel eingelassen, die gefedert in die Fuge des Profils drückt. Durch diesen geringen Anpressdruck verkantet der Ausleger sich schon und behält die Höhe. Zusätzlich wurde eine Verbindungsschraube durch ein Sterngriff mit passendem Gewindestrang ausgetauscht. Durch Drehen am Sterngriff kann die Verbindung leicht gelockert oder festgesetzt, und somit der Ausleger auf einer Höhe fixiert werden.

Abbildung 6.16: Auslegerbefestigung mit Sterngrifffeststeller

6.7.3 Sensorbefestigung am Demonstrator

Um den Sensor am Demonstrator zu befestigen wurde eine Verbindungplatte gebaut, an die sich der Sensor befestigen lässt und das Befestigungssystem mittels Nutensteine nutzt.

Um die Verbindungsplatte herzustellen, wurde Acrylglas genommen, da dieses leicht zu bearbeiten ist, und am Ende durch die Durchsichtbarkeit farblich sich nicht von den edlen Aluprofilen absetzt. Für die Sensorbefestigung wurden zwei Löcher gebohrt, durch die der Sensor per Schraube und Mutter befestigt wird. Es wurden zwei Löcher gebohrt, durch die eine Schraube passt, die in einen Nutenstein greift.

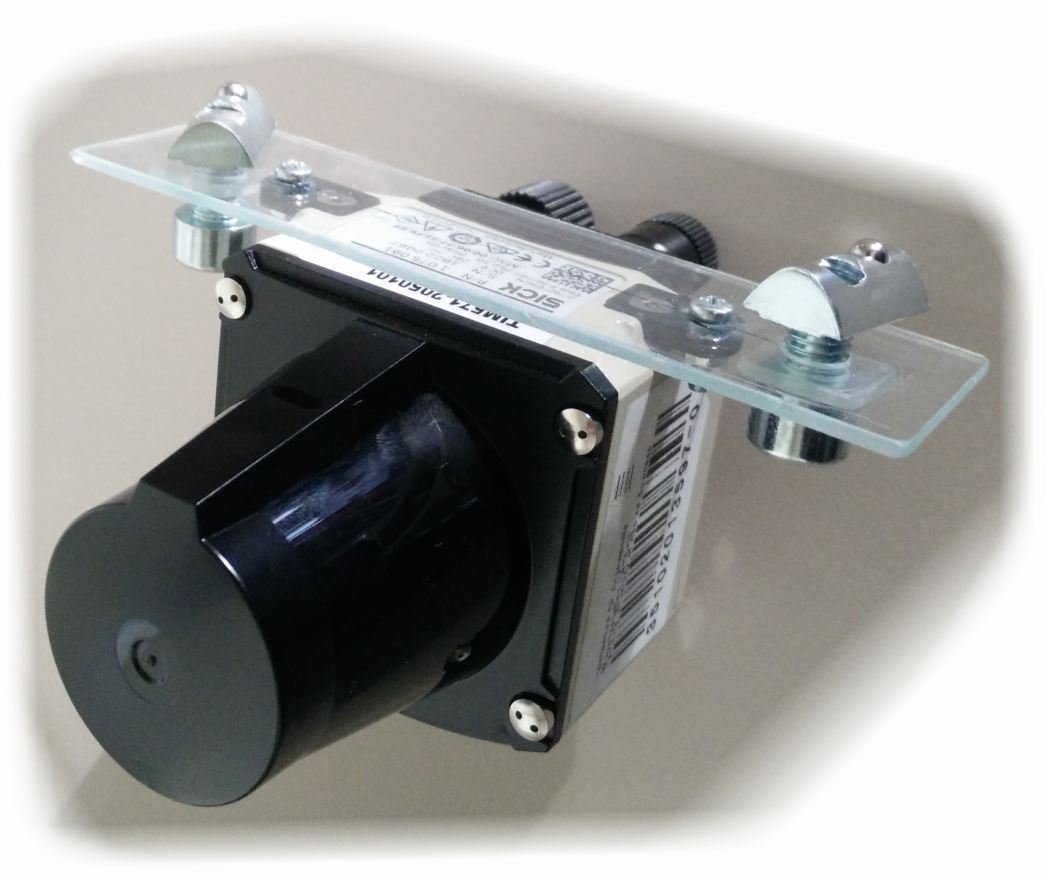

Abbildung 6.17: Laserinfrarotsensor TIM571 mit Verbindungsplatte

7 Tests - Messergebnisse

7.1 Messgenauigkeit

Um die Messgenauigkeit zu ermitteln, wurden verschiedene Messreihen aufgenommen und verglichen. Es wurden jeweils unterschiedlich große Objekte unter den Sensor gelegt und der Sensor zu Tisch Abstand variiert. Es wurden pro Messdurchgang 200 Messungen über die ermittelte Objekthöhe aufgenommen. Zudem wurden zwei Verfahren zur Objekthöhenbestimmung untersucht. Messreihe 1 zeigt die ermittelte Objekthöhe pro Messung. In Messreihe 2 wird die Objekthöhe in jeder Messung aus dem Durchschnitt der aktuellen ermittelten Objekthöhe und den letzten zwei berechnet. Dadurch wird erhofft, die Streuung der Ergebnisse zu reduzieren.

Dabei wurden drei unterschiedliche Objekte untersucht:

1. Eine Kugel mit einem Durchmesser von 14 cm.
   Die Größe könnte in der Praxis ein Bein oder den Kopf eines Patienten widerspiegeln.
2. Ein Zylinder mit einem Durchmesser von 12,6 cm Die Größe könnte in der Praxis ein Bein oder den Kopf eines Patienten widerspiegeln.
   Dieser Gegenstand wurde als Vergleich zu Kugel ausgewählt, um eine mögliche Messungenauigkeit, aufgrund einer parallelen Verschiebung der Messpunktreihe bei der Kugelmessung, zu untersuchen.
3. Eine Box mit einer Höhe von 14,2 cm und einer Breite von 23,4 cm.
   Die Größe könnte in der Praxis den Brustkorb eines Patienten widerspiegeln.
4. Eine Box mit einer Höhe von 3 cm und einer Breite von 10 cm
   Die Größe könnte in der Praxis eine Hand eines Patienten widerspiegeln.

Im folgenden Verlauf wird die Standardabweichung betrachtet. Hierbei handelt es sich um die empirische Standardabweichung, auch genannt Stichprobenstreuung. Sie ist ein Maß, inwieweit die einzelnen Messpunkte vom Durchschnitt aller Messpunkte entfernt sind.

7.1.1 Messobjekt: Kugel

In diesem Versuch wurde eine Kugel mit dem Durchmesser 140 mm als Messobjekt genommen. Die einzelnen Messkurven sind detailliert im Anhang zu finden auf S.61.

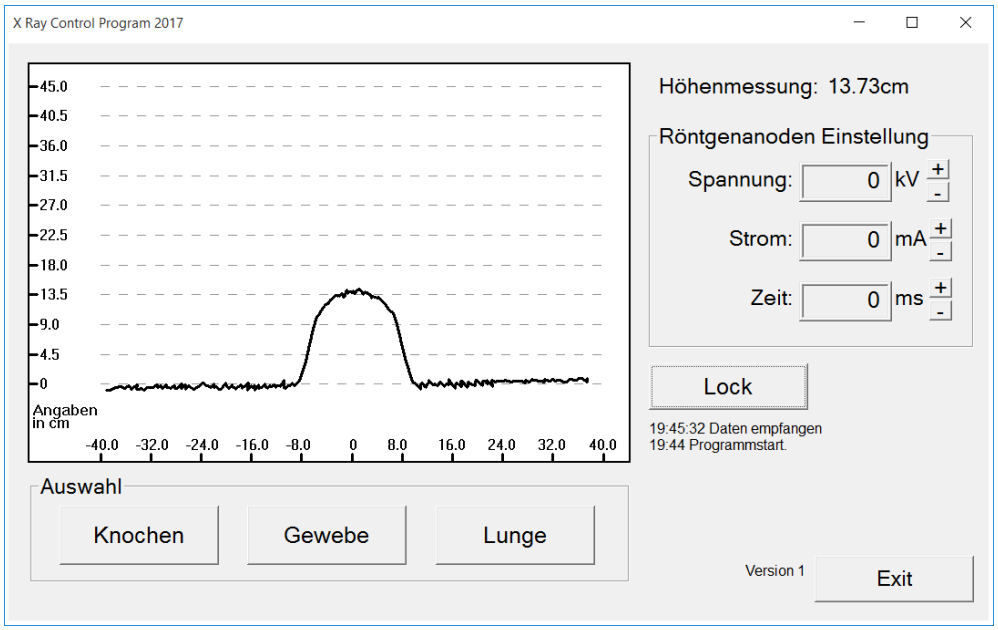

Abbildung 7.1: Screenshot der Messung bei einem Abstand von 40 cm (Kugel 140x140 mm)

Sensor zu Tisch Abstand	Durch-schnitt	Messreihe 1-x max. Wert	Messreihe 1-x min. Wert	Messreihe 1-x Standard-abweichung	Messreihe 2-x max. Wert	Messreihe 2-x min. Wert	Messreihe 2-x Standard-abweichung
mm	mm	mm	mm	mm	mm	mm	mm
300	136,1	138,1	134,0	0,720	137,2	134,8	0,410
400	137,1	139,0	134,8	0,806	138,4	135,9	0,493
500	138,1	140,2	135,3	0,882	139,4	136,2	0,518
600	136,0	138,9	132,9	1,196	137,9	133,7	0,748
700	136,7	140,9	131,8	1,459	138,6	134,6	0,790
1000	135,0	139,9	128,7	1,804	137,0	131,8	1,017
1300	134,0	139,1	126,7	2,388	137,2	130,4	1,379

Tabelle 7.1: Kenndaten der Messungen mit einer Kugel (140x140 mm)

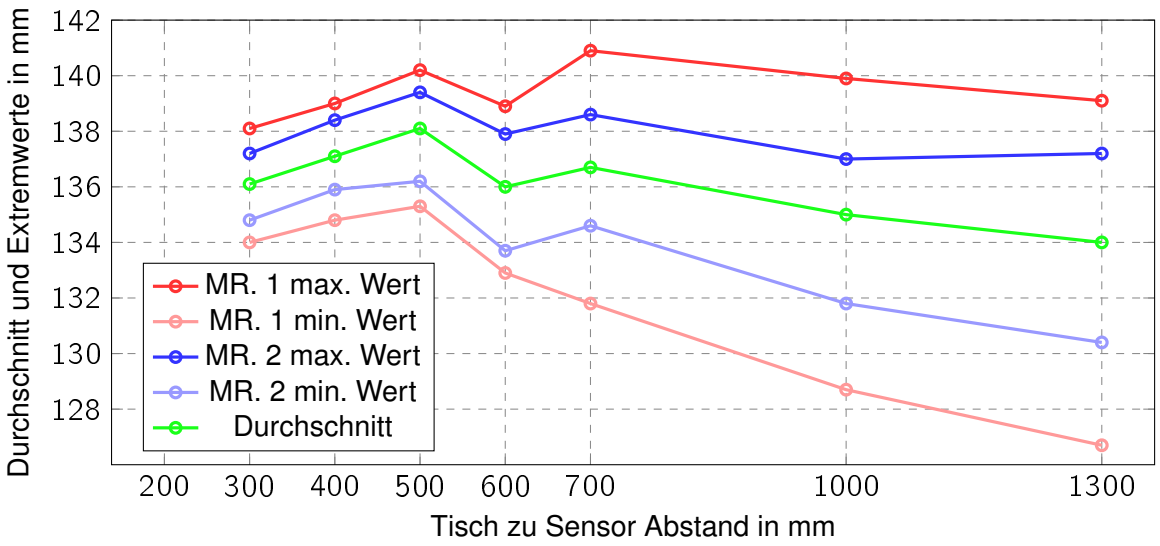

Abbildung 7.2: Vergleich der min. und max. Werte der Messreihen einer Kugel (140x140 mm)

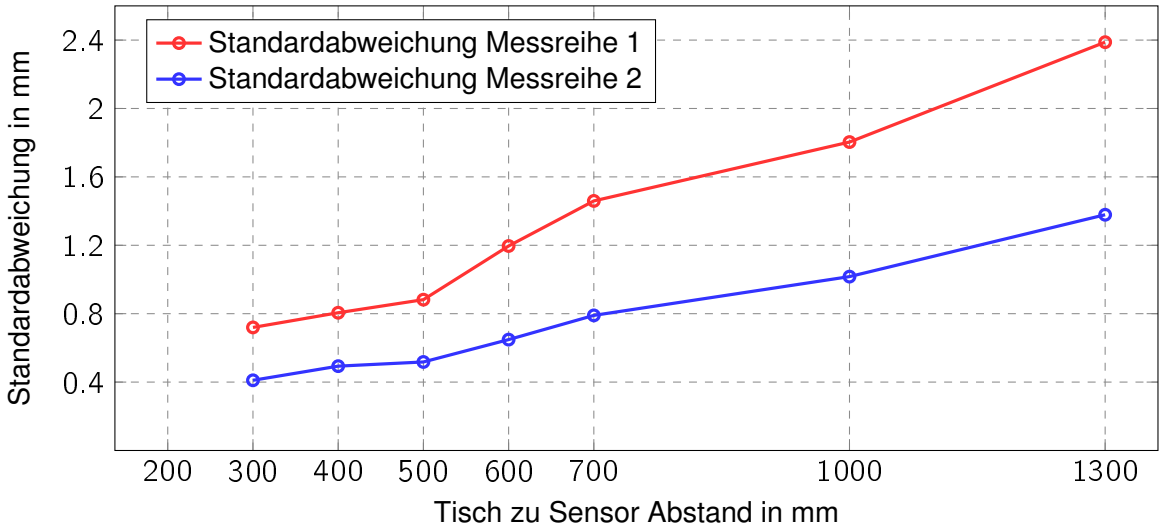

Abbildung 7.3: Vergleich der Standardabweichung der beiden Messreihen einer Kugel (140x140 mm)

7.1.2 Messobjekt: Zylinder 126x126 mm

In diesem Versuch wurde ein Zylinder mit einem Durchmesser von 126 mm als Messobjekt genommen. Das Material des Zylinders war Pappe. Die Messung erfolgte über die Breite des Zylinders. Die einzelnen Messkurven sind detailliert im Anhang zu finden auf S.64.

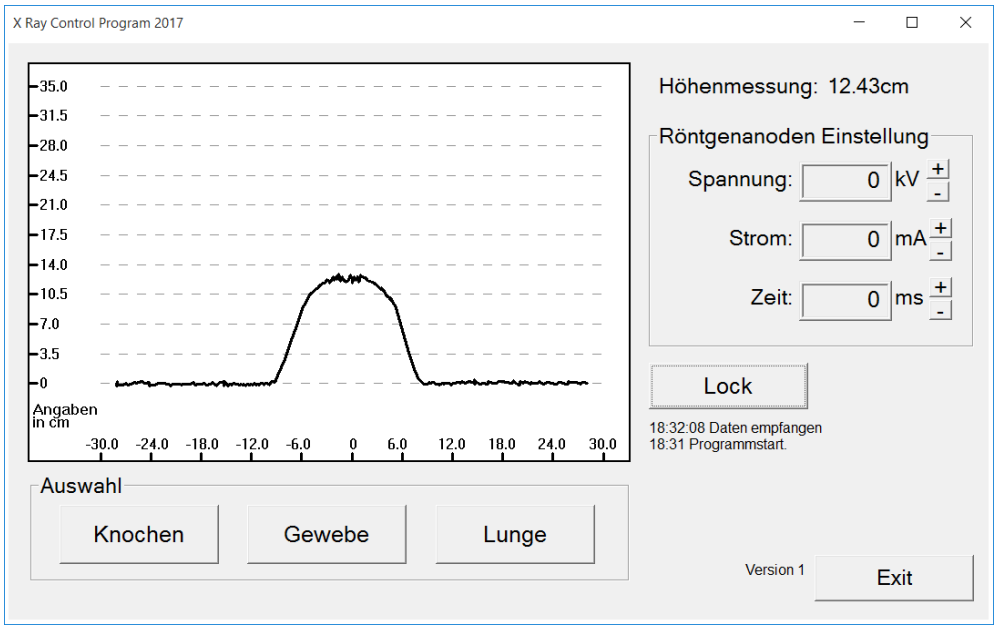

Abbildung 7.4: Screenshot der Messung bei einem Abstand von 30 cm (Zylinder 140x140 mm)

Sensor zu Tisch Abstand	Durch-schnitt	Messreihe 1-x			Messreihe 2-x		
		max. Wert	min. Wert	Standard-abweichung	max. Wert	min. Wert	Standard-abweichung
mm	mm	mm	mm	mm	mm	mm	mm
300	123,2	126,7	121,0	0,913	124,6	121,9	0,532
400	125,9	128,1	123,8	0,889	127,1	124,7	0,501
500	126,6	129,3	123,3	1,118	128,7	125,0	0,648
600	126,7	130,1	122,5	1,252	128,2	124,5	0,714
700	126,9	130,5	123,6	1,276	128,7	125,2	0,699
1000	125,5	129,5	118,5	1,936	128,4	122,5	1,054
1300	125,4	129,8	118,5	2,016	127,7	122,0	1,124

Tabelle 7.2: Kenndaten der Messungen von einem Zylinder (126x126 mm)

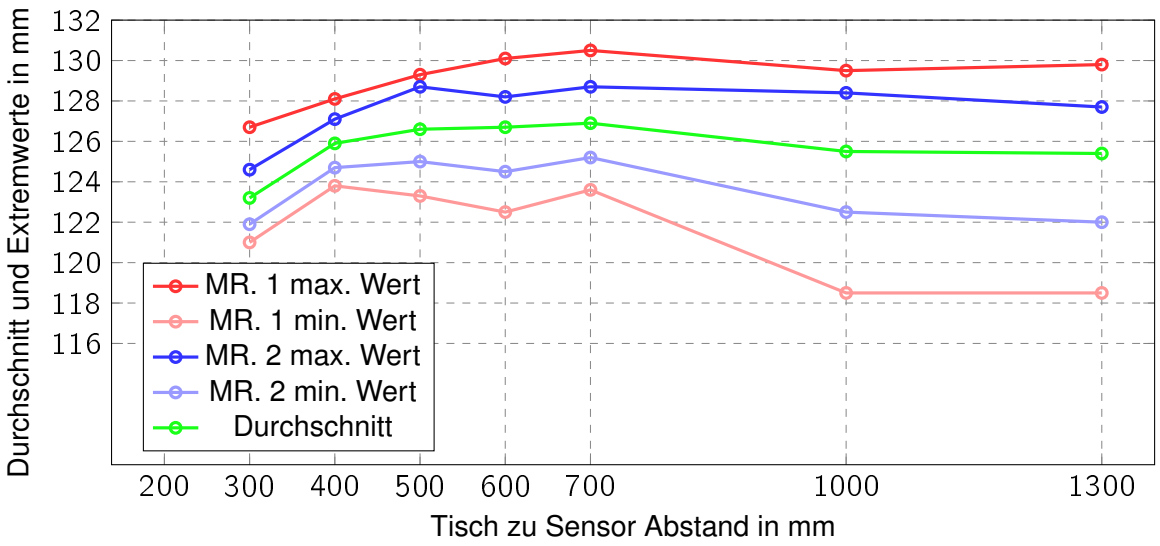

Abbildung 7.5: Vergleich der min. und max. Werte der Messreihen von einem Zylinder (126x126 mm)

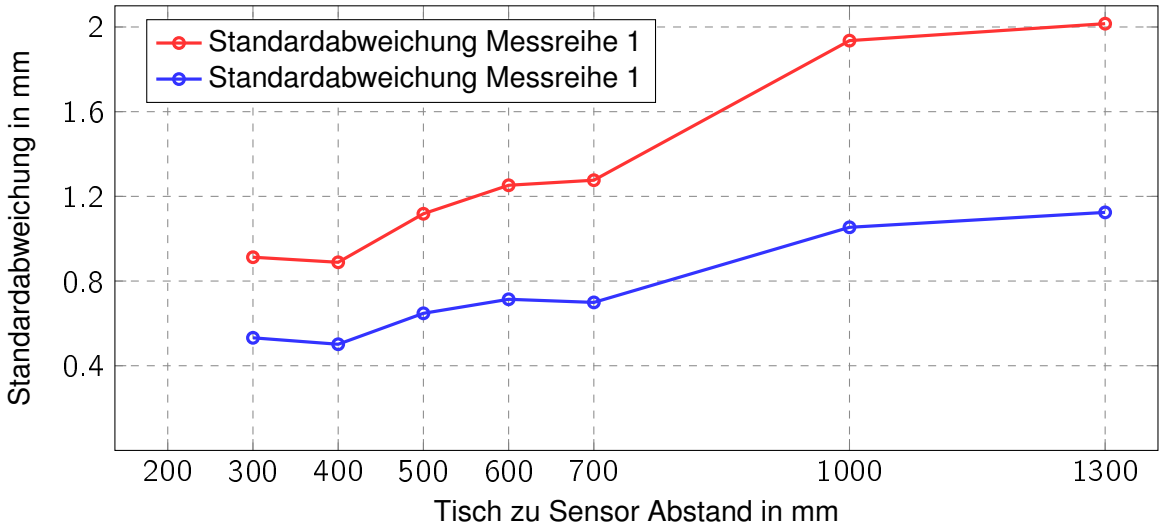

Abbildung 7.6: Vergleich der Standardabweichung der beiden Messreihen von einem Zylinder (126x126 mm)

7.1.3 Messobjekt: Box 142x234 mm

In diesem Versuch wurde eine Box aus Pappe mit einer Höhe von 142 mm und einer Breite von 234 mm als Messobjekt genommen. Die einzelnen Messkurven sind detailliert im Anhang zu finden auf S.67.

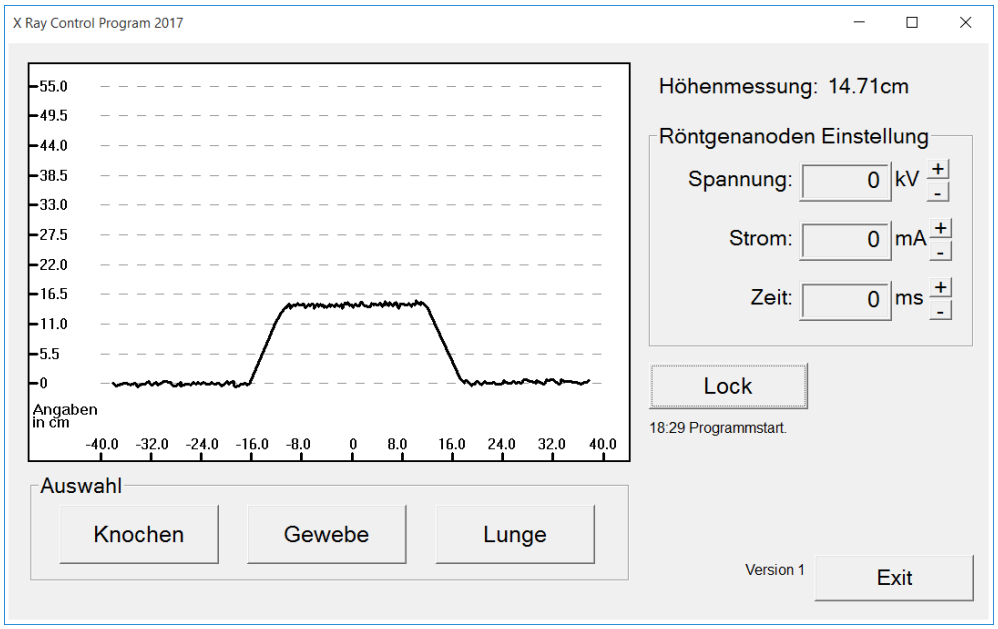

Abbildung 7.7: Screenshot der Messung bei einem Abstand von 50cm (Box 142x234 mm)

Sensor zu Tisch Abstand	Durch-schnitt	Messreihe 1-x			Messreihe 2-x		
		max. Wert	min. Wert	Standard-abweichung	max. Wert	min. Wert	Standard-abweichung
mm	mm	mm	mm	mm	mm	mm	mm
300	140,7	142,3	138,8	0,640	141,9	139,5	0,370
400	143,9	146,5	141,7	0,831	145,1	142,6	0,454
500	147,4	150,2	145,0	0,970	149,3	146,0	0,600
600	147,3	149,7	143,1	1,042	148,6	145,0	0,614
700	146,1	150,0	142,8	1,048	148,0	144,3	0,574
1300	147,0	150,0	143,7	1,132	148,8	144,6	0,680

Tabelle 7.3: Kenndaten der Messungen mit einer Box (142x234 mm), Werte in mm

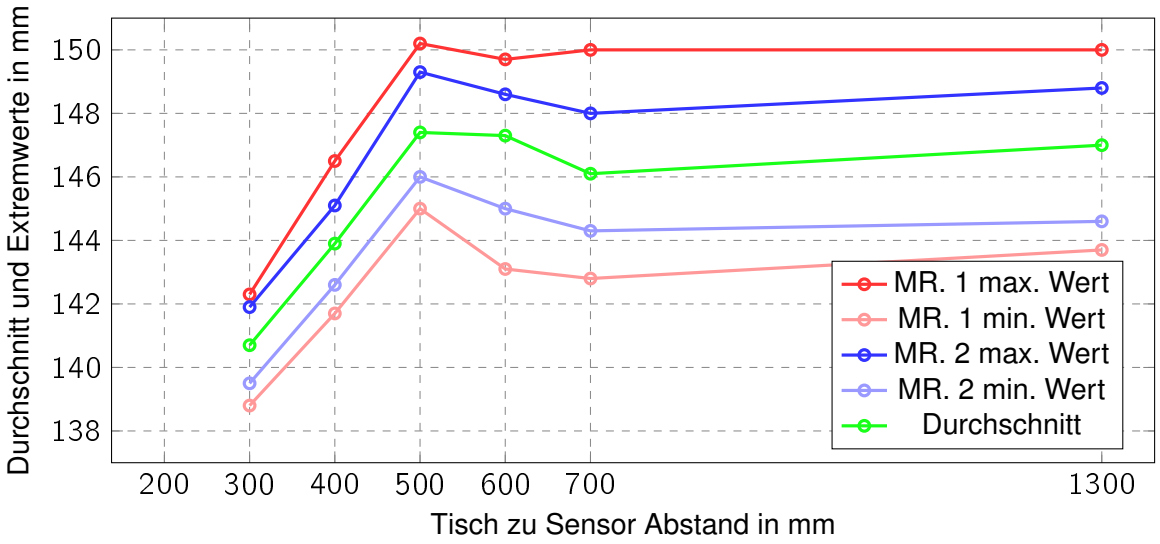

Abbildung 7.8: Vergleich der Messbereiche der Messreihen von einer Box (142x234 mm)

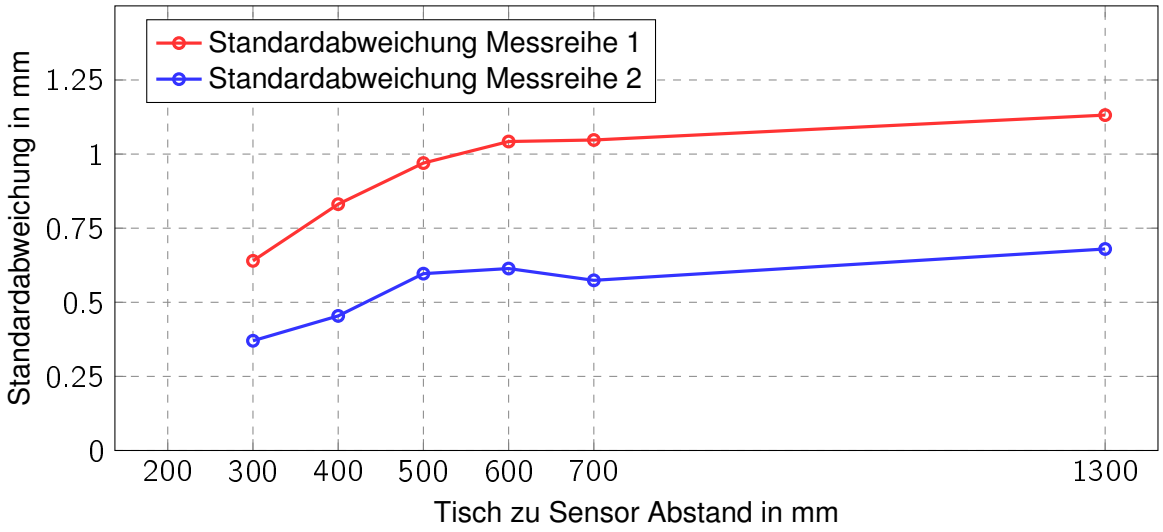

Abbildung 7.9: Vergleich der Standardabweichung der beiden Messreihen einer Box (142x234 mm)

7.1.4 Messobjekt: Box 30x100 mm

In diesem Versuch wurde eine Box aus Pappe mit einer Höhe von 30 mm und einer Breite von 100 mm als Messobjekt genommen. Die einzelnen Messkurven sind detailliert im Anhang zu finden auf S.69.

Sensor zu Tisch Abstand	Durch-schnitt	Messreihe 1-x			Messreihe 2-x		
		max. Wert	min. Wert	Standard-abweichung	max. Wert	min. Wert	Standard-abweichung
mm	mm	mm	mm	mm	mm	mm	mm
200	26,19	28,16	23,93	0,803	27,50	25,05	0,450
300	26,80	28.98	24,45	0,929	28,40	25,67	0,546
400	32,58	34,82	30,76	0,830	33,77	31,41	0,533
500	32,18	35,22	29,64	1,013	33,93	30,60	0,608
600	30,31	32,59	27,72	0,896	31,78	29,02	0,548
700	31,32	33,71	28,57	0,860	32,46	30,24	0,490
1300	34,40	36,74	31,43	1,002	35,63	32,47	0,540

Tabelle 7.4: Kenndaten der Messungen mit einer Box (30x100mm)

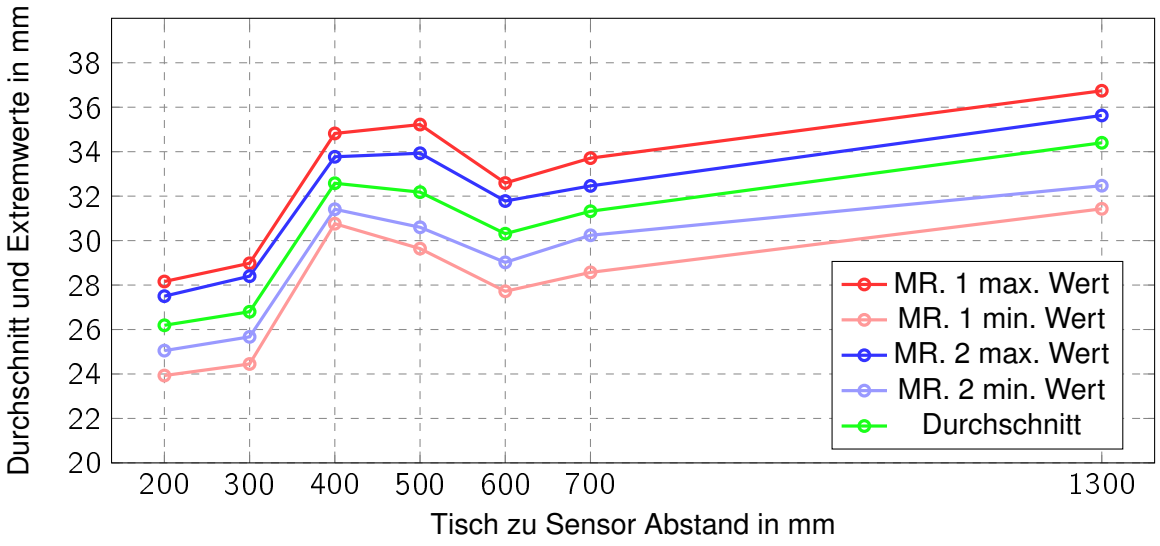

Abbildung 7.10: Vergleich der Messbereiche der Messreihen von einer Box (30x100 mm)

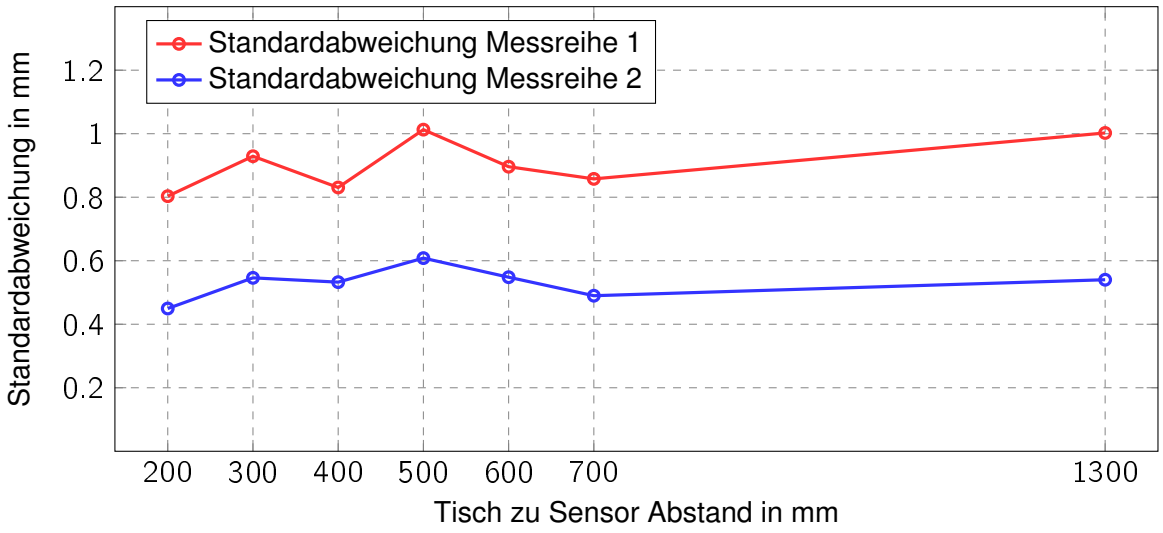

Abbildung 7.11: Vergleich der Standardabweichung der beiden Messreihen einer Box (30x100 mm)

7.1.5 Auswertung Messgenauigkeit

Objekt (Höhe)	totaler Durchschnitt	Messreihe 1		Messreihe 2	
		Spann-weite	max. Ab-weichung	Spann-weite	max. Ab-weichung
Kugel (140 mm)	136,1 mm	14,2 mm	13,3 mm = 9,5%	9,0 mm	9,6 mm = 6,9%
Zylinder (126 mm)	125,7 mm	12,0 mm	7,5 mm = 6,0%	6,8 mm	4,1 mm = 3,3%
Box (142 mm)	145,4 mm	11,4 mm	8,2 mm = 5,8%	9,8 mm	7,2 mm = 5,1%
Box (30 mm)	30,5 mm	12,8 mm	6,7 mm = 22,5%	10,6 mm	5,6 mm = 18,8%

Tabelle 7.5: Kenndaten aller Höhenmessungen pro Messobjekt

Zu bemerken ist, dass bei der Kugel als Messobjekt, meist immer eine niedrigere Höhe, als die reale ermittelt wurde. Zum einen liegt es daran, dass die Oberfläche der Kugel nicht flach ist und somit das 0,8-Perzentil unterhalb des maximalen Wert liegen muss. Im Vergleich mit

den Messwerten des Zylinders stellt sich aber heraus, dass dieser Effekt scheinbar keine große Auswirkung auf das Ergebnis hat. Eine Erklärung dafür kann sein, dass der Sensor womöglich eine Streuung der Distanzermittlung parallel zur eigentlichen mittigen Messpunktreihe hat. Da es zu jeder Seite parallel, zu einer mittig auf der Kugel verlaufenden Messpunktreihe, abfällt, könnte dies die niedrigeren Werte erklären.

Es zeigt sich, dass die Messmethode aus Messreihe 2 nicht nur die Standardabweichung verbessert, sondern auch folgerichtig die Spannweite und die maximale Abweichung der Messwerte verringert. Durch eine Implementierung dieser Messmethode könnt also die Messgenauigkeit verbessert werden.

Werden die Graphen aus den Messreihen „Durchschnitt und Extremwerte“ der Objekte verglichen, ist bei den Objekten „Box“ zu erkennen, dass der Durchschnitt bei einer Messdistanz von unter 40 cm unter der Messobjekthöhe liegen. Ab einer Messdistanz von 40cm liegen die Messwerte meist etwas darüber. Es wäre möglich, dass der Sensor bei verschiedenen Messdistanzen nicht gleich genau misst.
Eine Erklärung, warum dies nicht so ausgeprägt bei der Kugel und dem Zylinder zu sehen ist könnte sein, weil bei einer kleinen Messdistanz die seitlichen abfallenden Ränder aufgrund des Winkels zum Sensor im Verhältnis mit weniger Messpunkten gemessen werden. Somit liegen auf der höher gelegenen Objektoberfläche mehr Messpunkte. Dies gilt natürlich nur, wie in diesem Fall, wenn die Objekte sich exakt unterhalb des Sensors befinden. Die Betrachtung der Objekte mit einer ebenen Oberfläche liefern einen guten Vergleich für die Bestimmung der Messgenauigkeit. Die maximale Abweichung liegt dabei immer unter 1 cm. Im Verhältnis zur tatsächlichen Objekthöhe kann die hier gemessene maximale Abweichung aber über 20 % liegen.

7.2 Softwaretest

Das Programm wurde eine Stunde im Dauerbetrieb getestet und hat keinen Fehler verursacht. Nach der Stunde betrug der Speicherverbrauch zwischen 2,3 MB und 2,5 MB.
Auf einem 1 GHz schnellen Doppelkern PC-System betrug die ungefähre CPU-Last des Programms 2 %.

Um fünf Datenanfragen zu senden und deren Antwort vom Sensor zu bekommen werden im Schnitt 80 ms gebraucht. Dieser Wert liegt deutlich unter dem im Datenblatt angegeben Wert von 67 ms für eine Antwortzeit.

8 Fazit

8.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Demonstratoreinheit gebaut, um zu überprüfen, ob es möglich ist, mittels eines Laserinfrarotsensor Objektdimension verlässlich zu bestimmen, um aus den Messergebnissen optimale Röntgenanodenwerte abzuleiten.
Für die Anforderungen an die Hardware wurde das Szenario eines Röntgenvorganges betrachtet. Es wurde ein Demonstrator entworfen und sich für einen Infrarot-Lasersensor der Sick AG entschieden.
Bei der Programmierung der Software wurde auf Multitasking mittels Multithreading gesetzt, um gleichzeitig Sensordaten abfragen und die grafische Ausgabe der letzten Messdaten ausgeben zu können. Die Grunddaten als Vorlage zur Bestimmung der Röntgenanodenwerte wurden aus einer XML Datei ausgelesen und Zwischenwerte interpoliert. Das Programm hat eine intuitive Programmoberfläche mit Steuerelementen, die auch per Tastatureingaben gesteuert werden können. Zudem wird das Messbild des Sensors grafisch dargestellt. Um den Sensor an dem Ausleger des Demonstrators zu befestigen wurde eine eigene Verbindungsplatte gebaut.
In den Tests wurde die Funktion des Systems bestätigt und die Messgenauigkeit in verschiedenen Szenarien festgestellt.

8.2 Ausblick

Das entwickelte System zeigt eine mögliche Verbesserung des heutigen Röntgensystems. Es wäre denkbar, dass in Zukunft Röntgensysteme mit einer hier vorgestellten Technik ausgestattet werden.
Eine weitere Optimierung des Demonstratorsystems kann sein, dass die Ermittlung der Tisch-zu-Sensor-Distanz nicht mehr über die Auswertung des 2D-Laserinfrarotsensors geschieht, sondern über einen dafür extra optimierten Zusatzsensor. Außerdem ließe sich zum Beispiel mit einem 3D Messsensor das Messbild noch weiter optimieren.

Literaturverzeichnis

[1] Prof. Dr. Gustaf Neumann, Wirtschaftsuniversität Wien, Institut für Wirtschaftsinformatik und Neue Medien, “XML.”
http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/Sprache/Auszeichnungssprache/XML.
Abruf 23. Nov. 2016.

[2] Prof. Dr. Gustaf Neumann, Wirtschaftsuniversität Wien, Institut für Wirtschaftsinformatik und Neue Medien, “Auszeichnungssprache.”
http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/Sprache/Auszeichnungssprache/.
Abruf 23. Nov. 2016.

[3] SICK AG, *Online-Datenblatt TiM571-2050101*, 2016.

[4] © item Industrietechnik GmbH. http://www.item24.de/. Abruf Nov. 2016.

[5] Carly Salali, “MFC Interfaces - Creating Simple Hand-crafted (hacked) MFC Interfaces for C++ Programs.”
https://www.youtube.com/watch?v=Su1x9EA5Njk.
Abruf 23. Nov. 2016.

[6] Lee Thomason, “TinyXML-2.”
https://github.com/leethomason/tinyxml2.
Abruf 23. Nov. 2016.

[7] Helmut Erlenkötter, *C Programmieren von Anfang an*. Rowohlt Taschenbuch Verlag, 20. Auflage 2012 . ISBN: 978-3-499-60074-6 .

[8] Helmut Erlenkötter, *C++ Objektorientiertes Programmieren von Anfang an*. Rowohlt Taschenbuch Verlag, 17. Auflage 2016 . ISBN: 978-3-499-60077-7 .

[9] R. Allen Wyke, Sultan Rehmann, Brad Leupen, *XML - Das Entwicklerbuch*. Microsoft Press, 2002. ISBN: 3-86063-636-7.

[10] Urs Gleim, Tobias Schüle, *Multicore-Software*. dpunkt.verlag, 2012. ISBN: 978-3-89864-758-8.

[11] Hans Joachim Eichler, Jürgen Eichler, *Laser: Bauformen, Strahlführung, Anwendungen*. Springer Vieweg, 8. Auflage 2015. ISBN: 978-3-642-41438-1 (eBook).

[12] Olaf Dössel, *Bildgebende Verfahren in der Medizin*. Springer Vieweg, 2. Auflage 2016. ISBN: 978-3-642-54407-1 (eBook).

[13] Thomas Rauber, Gudula Rünger, *Parallele Programmierung*. Springer Vieweg, 3. Auflage 2012. ISBN: 978-3-642-13604-7 (eBook).

[14] Robert Hilbrich, *Platzierung von Softwarekomponenten auf Mehrkernprozessoren*. Springer Vieweg, 2015. ISBN: 978-3-658-11173-1 (eBook).

Tabellenverzeichnis

Abbildungsverzeichnis

9 Anhang

9.1 Messkurven Kugel 140x140 mm

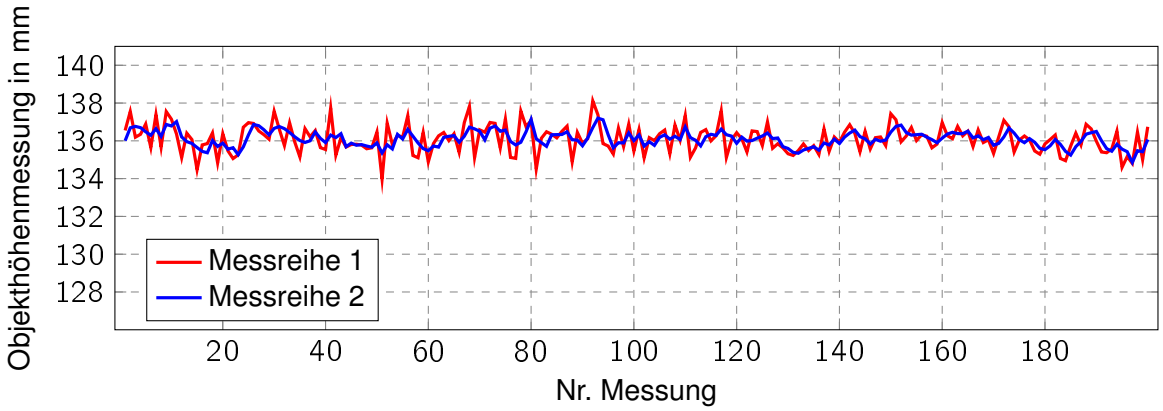

Abbildung 9.1: Objekthöhenmessung einer Kugel (140x140 mm) aus 30cm Höhe

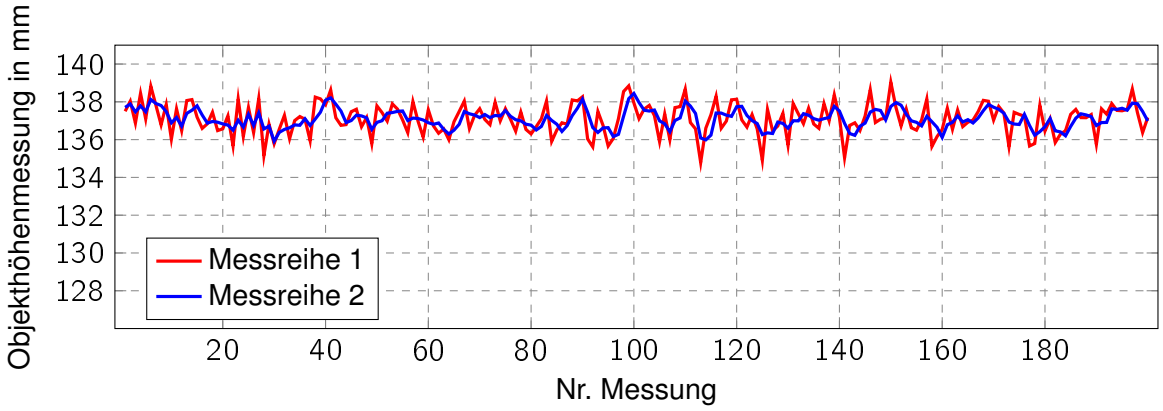

Abbildung 9.2: Objekthöhenmessung einer Kugel (140x140 mm) aus 40cm Höhe

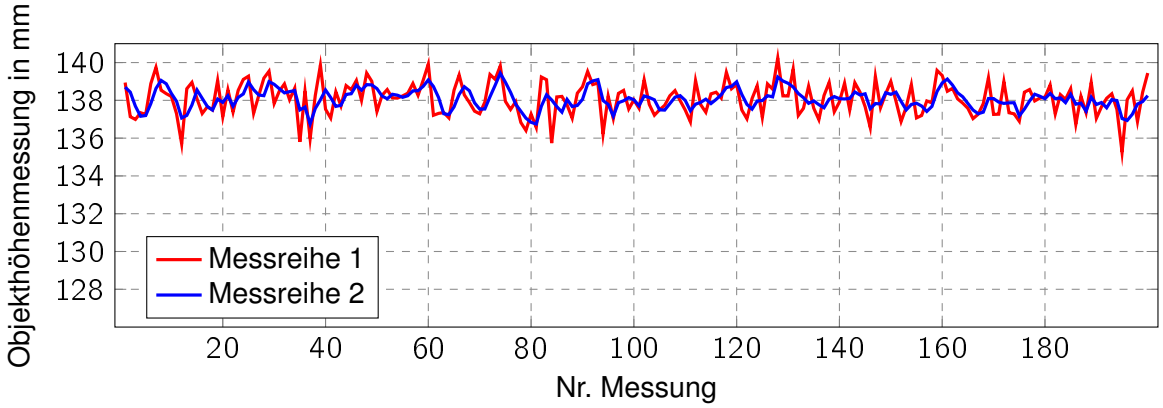

Abbildung 9.3: Objekthöhenmessung einer Kugel (140x140 mm) aus 50cm Höhe

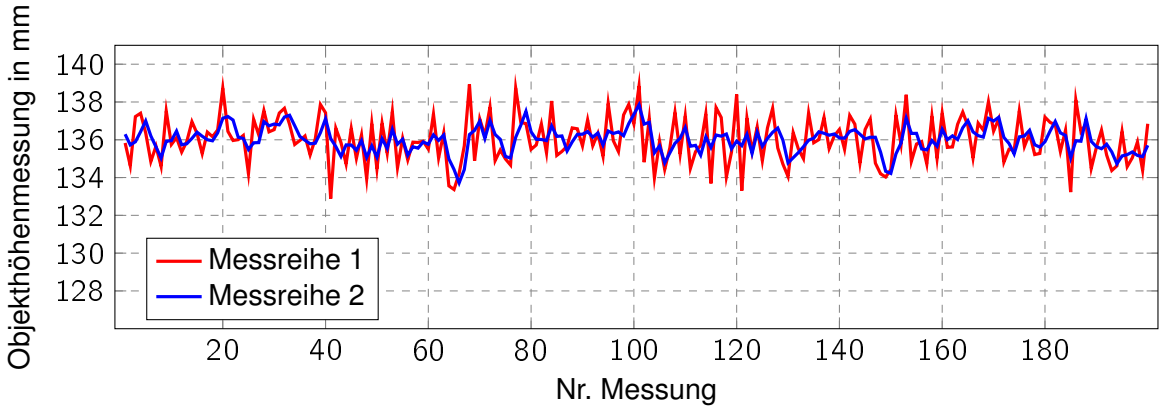

Abbildung 9.4: Objekthöhenmessung einer Kugel (140x140 mm) aus 60cm Höhe

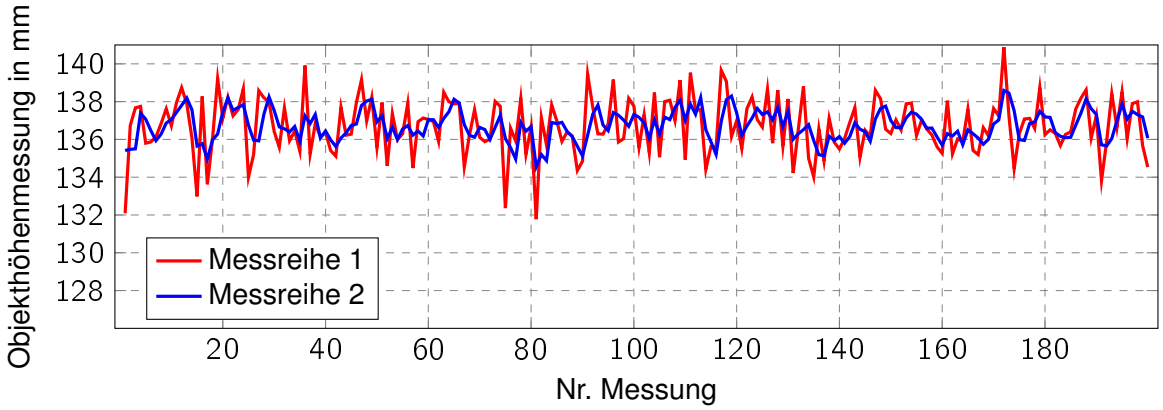

Abbildung 9.5: Objekthöhenmessung einer Kugel (140x140 mm) aus 70cm Höhe

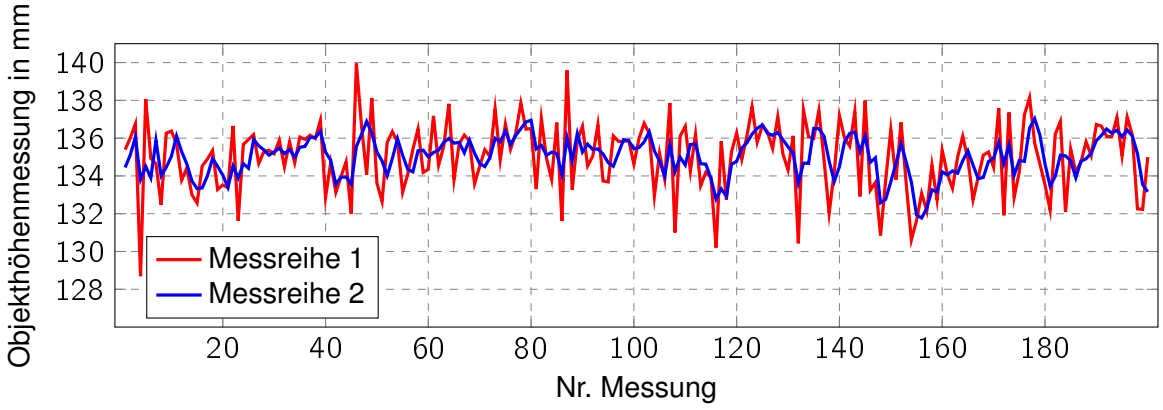

Abbildung 9.6: Objekthöhenmessung einer Kugel (140x140 mm) aus 100cm Höhe

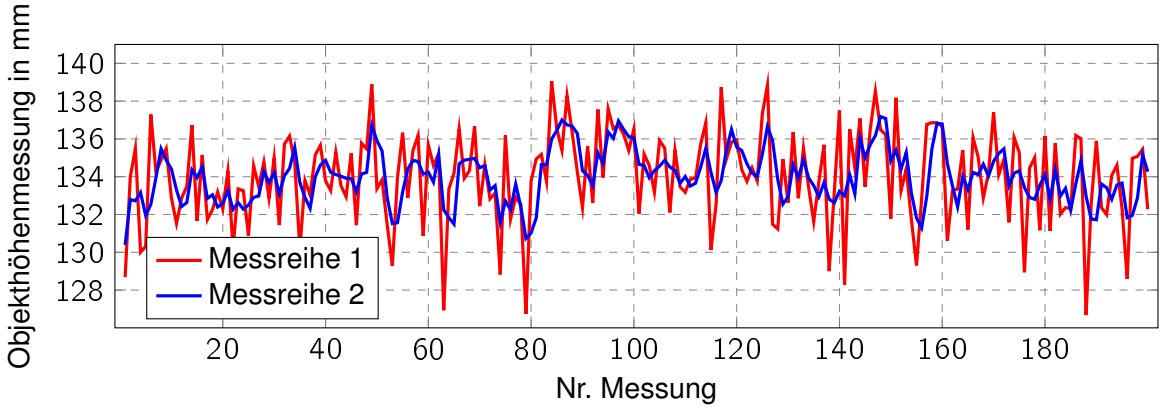

Abbildung 9.7: Objekthöhenmessung einer Kugel (140x140 mm) aus 130cm Höhe

9.2 Messkurven Zylinder 126x126 mm

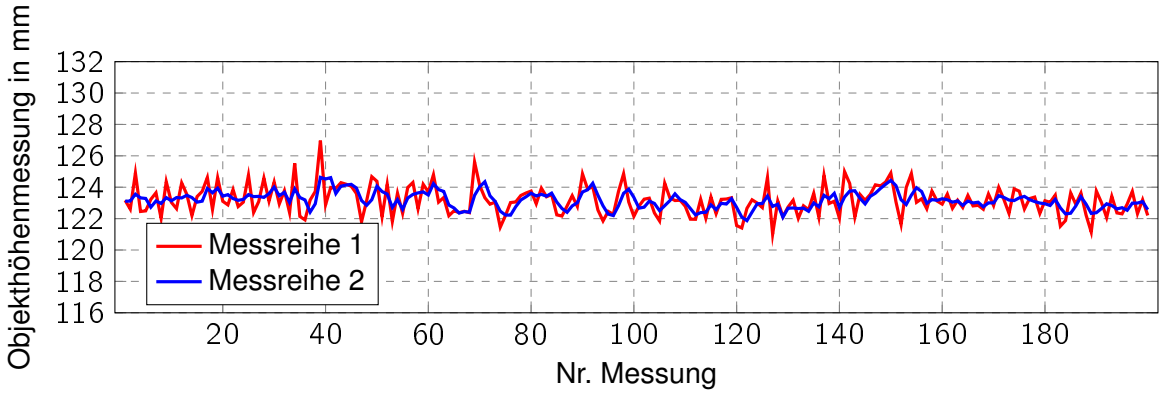

Abbildung 9.8: Objekthöhenmessung eines Zylinders (126x126 mm) aus 30 cm Höhe

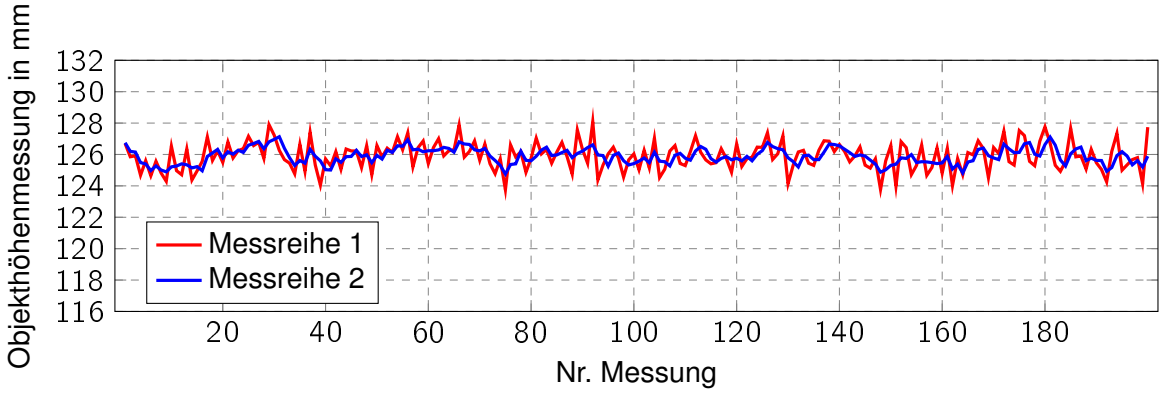

Abbildung 9.9: Objekthöhenmessung eines Zylinders (126x126 mm) aus 40 cm Höhe

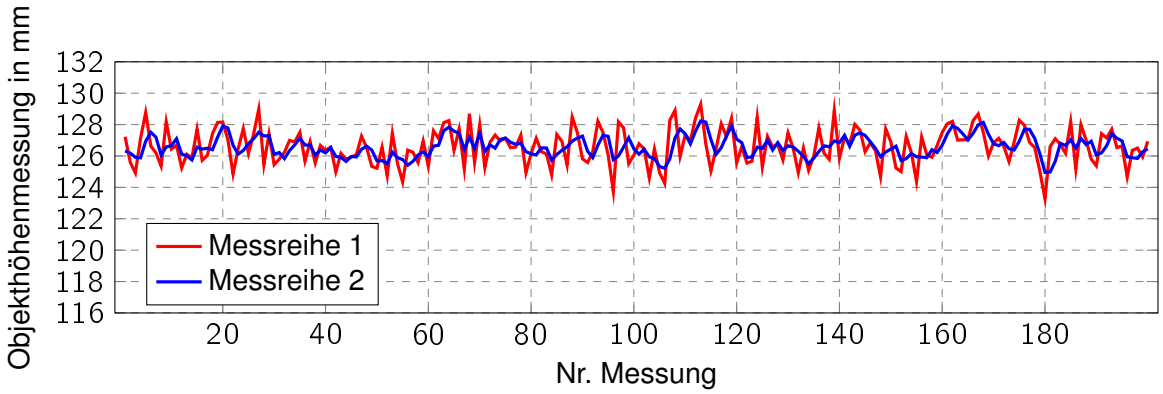

Abbildung 9.10: Objekthöhenmessung eines Zylinders (126x126 mm) aus 50 cm Höhe

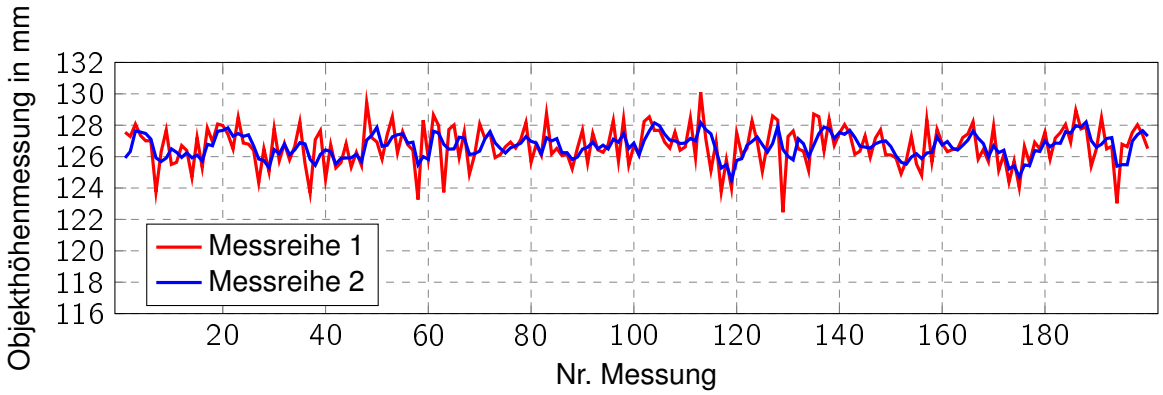

Abbildung 9.11: Objekthöhenmessung eines Zylinders (126x126 mm) aus 60 cm Höhe

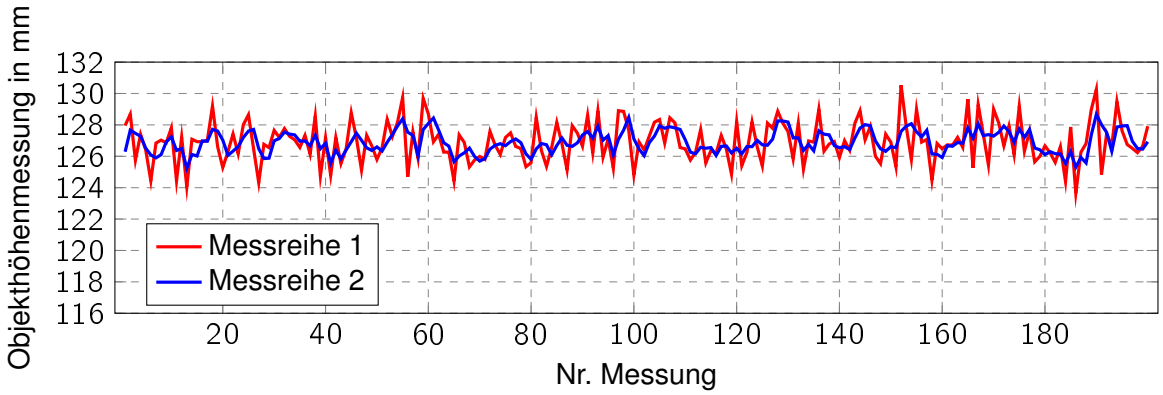

Abbildung 9.12: Objekthöhenmessung eines Zylinders (126x126 mm) aus 70 cm Höhe

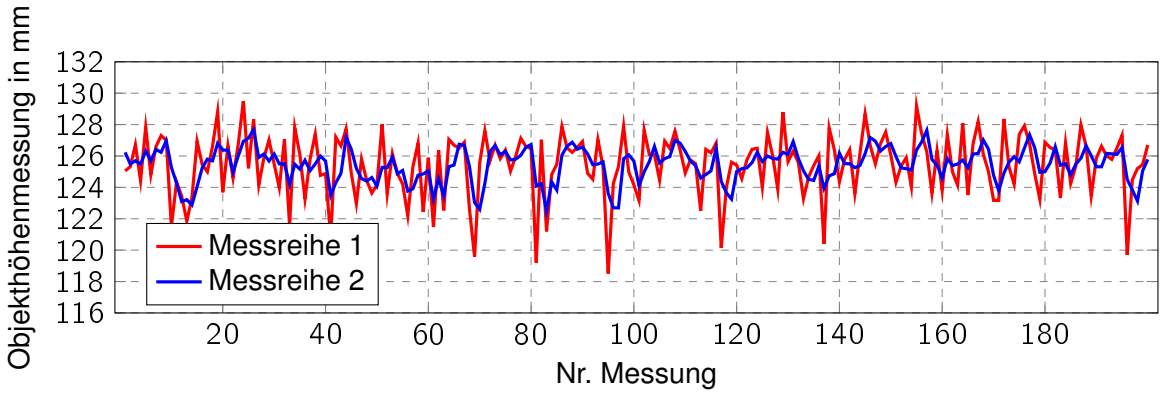

Abbildung 9.13: Objekthöhenmessung eines Zylinders (126x126 mm) aus 100 cm Höhe

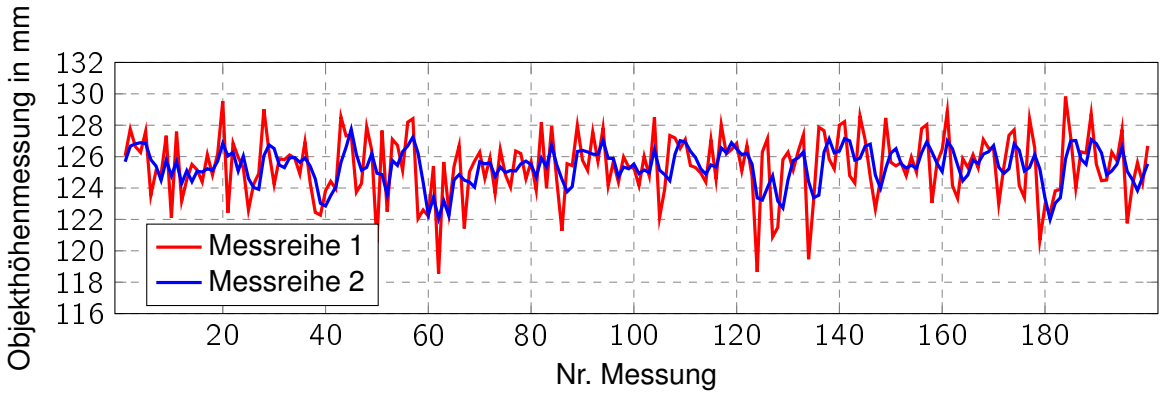

Abbildung 9.14: Objekthöhenmessung eines Zylinders (126x126 mm) aus 130 cm Höhe

9.3 Messkurven Box 142x234 mm

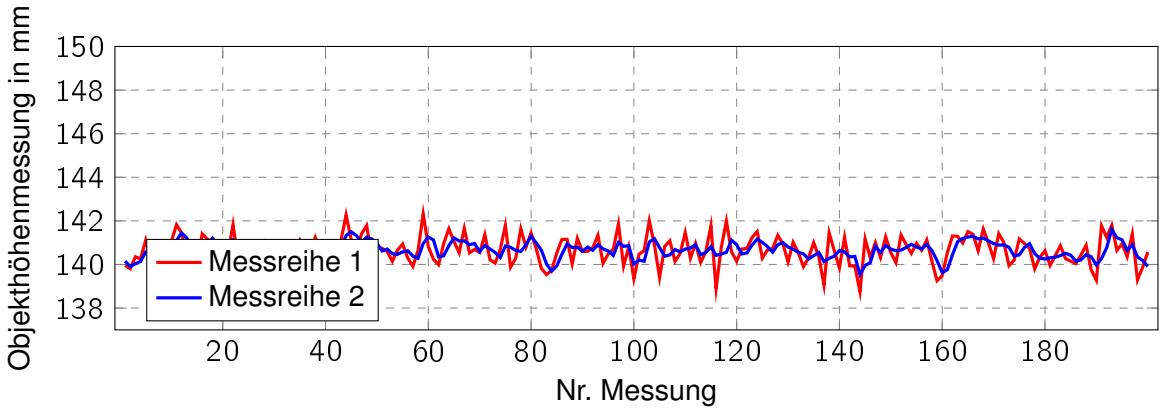

Abbildung 9.15: Objekthöhenmessung einer Box (142x234 mm) aus 30 cm Höhe

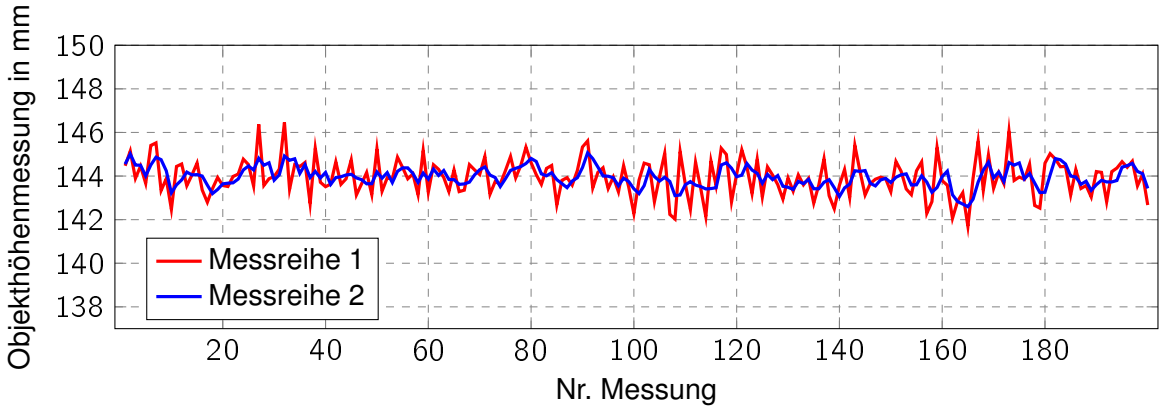

Abbildung 9.16: Objekthöhenmessung einer Box (142x234 mm) aus 40 cm Höhe

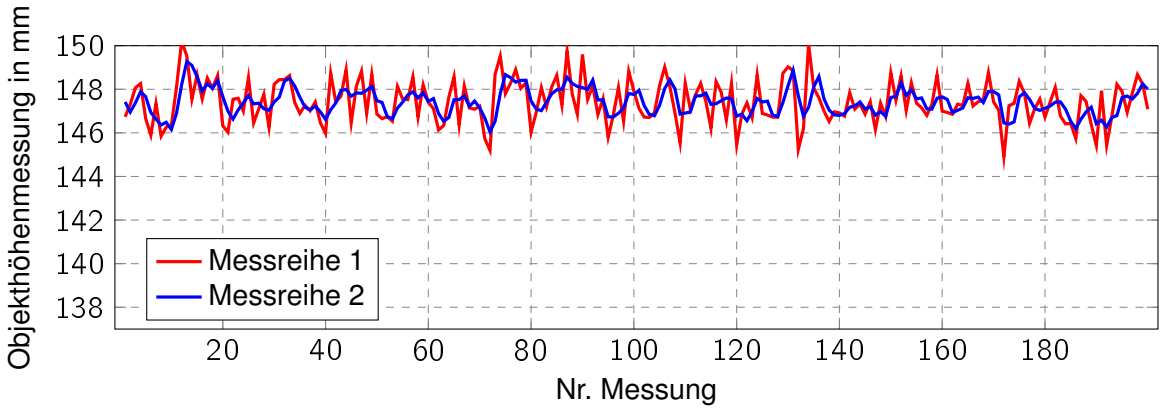

Abbildung 9.17: Objekthöhenmessung einer Box (142x234 mm) aus 50 cm Höhe

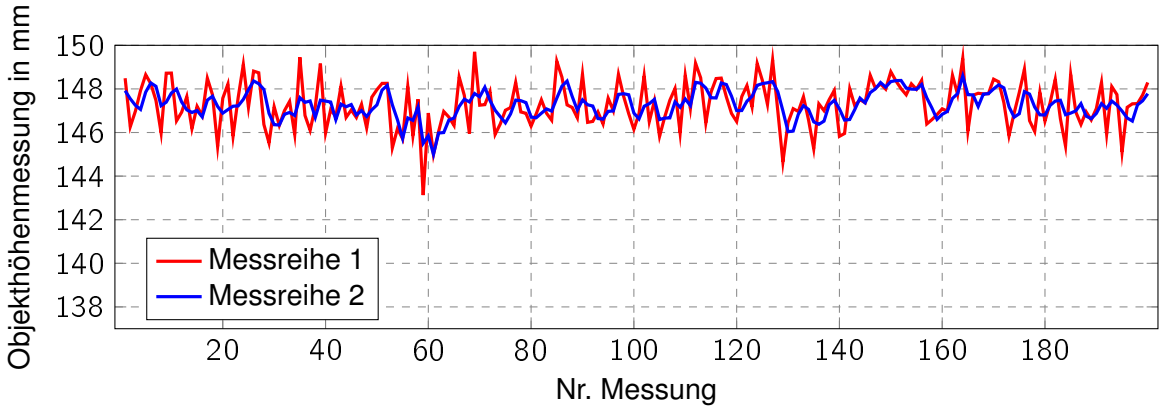

Abbildung 9.18: Objekthöhenmessung einer Box (142x234 mm) aus 60 cm Höhe

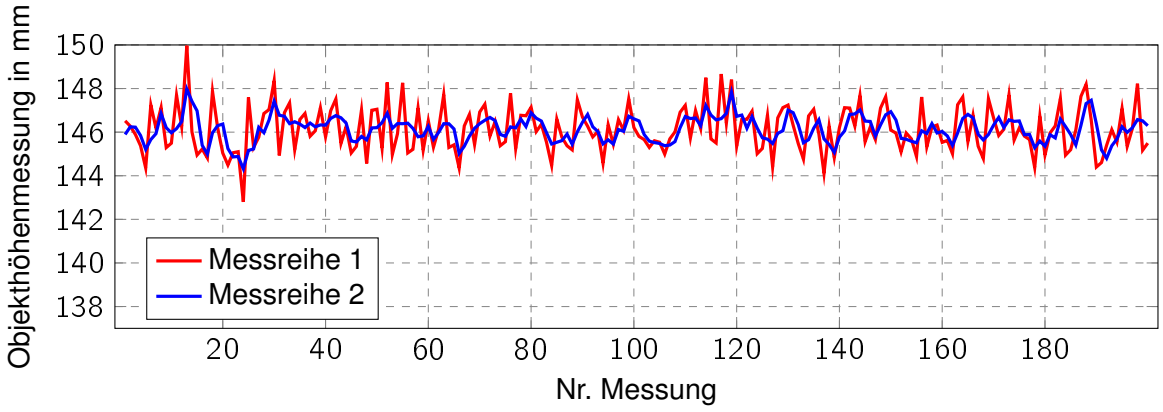

Abbildung 9.19: Objekthöhenmessung einer Box (142x234 mm) aus 70 cm Höhe

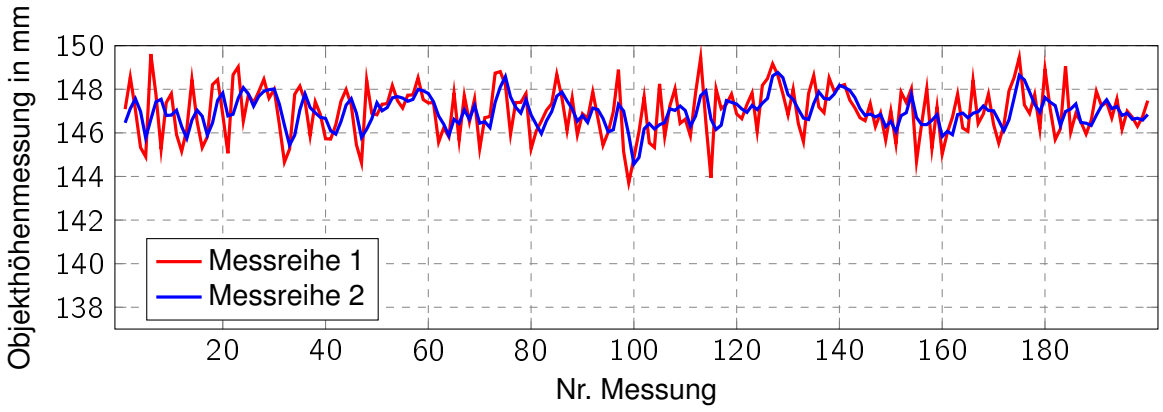

Abbildung 9.20: Objekthöhenmessung einer Box (142x234 mm) aus 1,3 m Höhe

9.4 Messkurven Box 30x100 mm

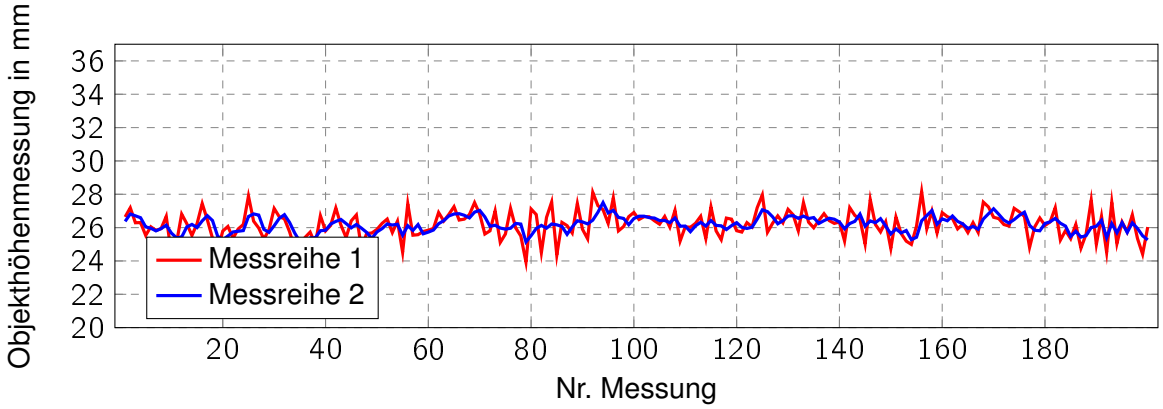

Abbildung 9.21: Objekthöhenmessung einer Box (30x100 mm) aus 20 cm Höhe

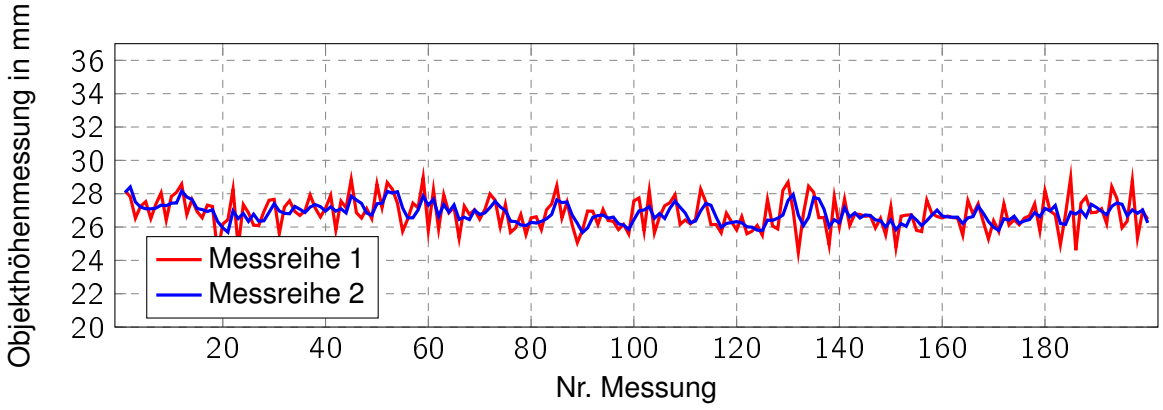

Abbildung 9.22: Objekthöhenmessung einer Box (30x100 mm) aus 30 cm Höhe

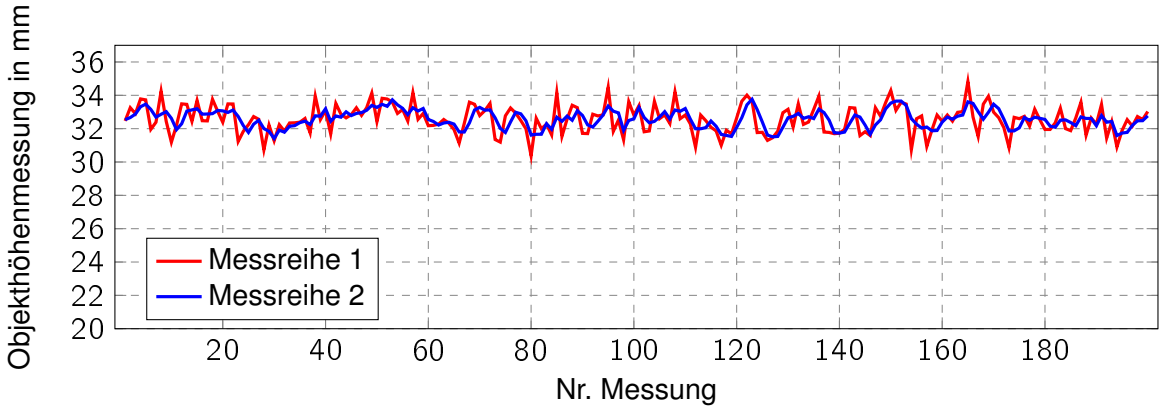

Abbildung 9.23: Objekthöhenmessung einer Box (30x100 mm) aus 40 cm Höhe

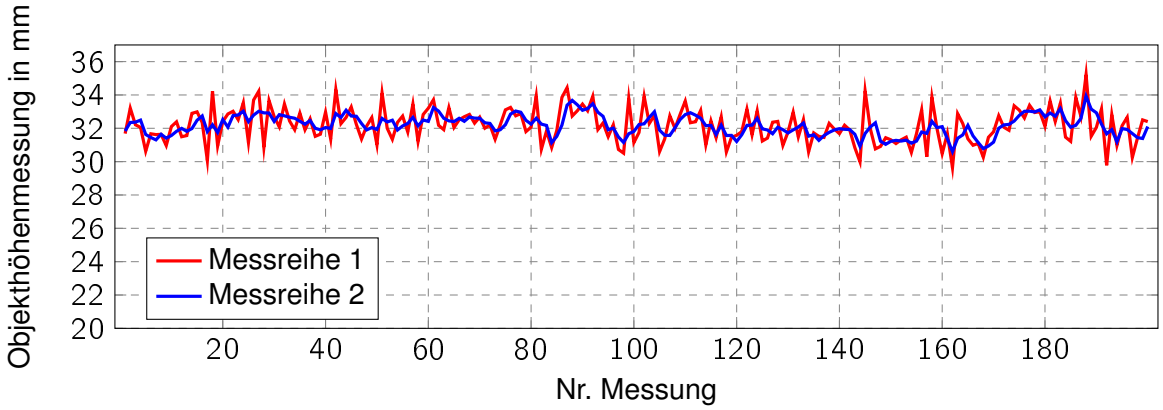

Abbildung 9.24: Objekthöhenmessung einer Box (30x100 mm) aus 50 cm Höhe

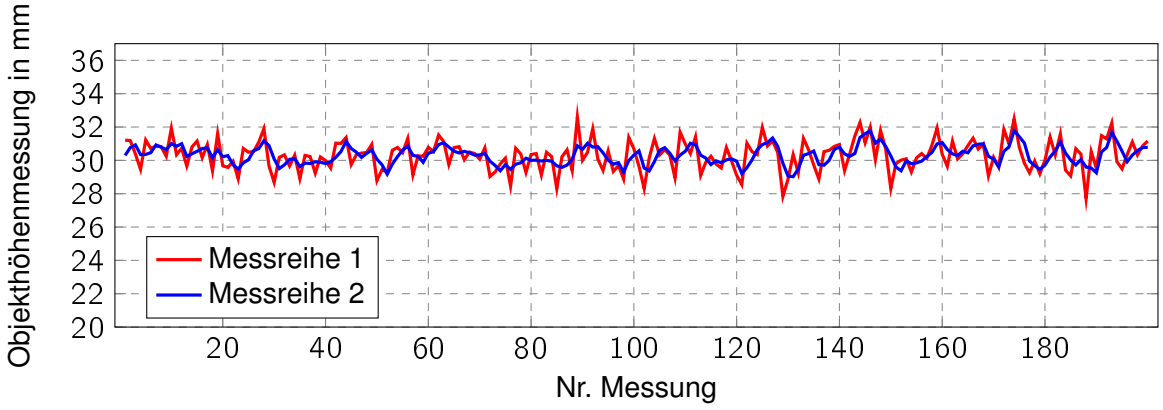

Abbildung 9.25: Objekthöhenmessung einer Box (30x100 mm) aus 60 cm Höhe

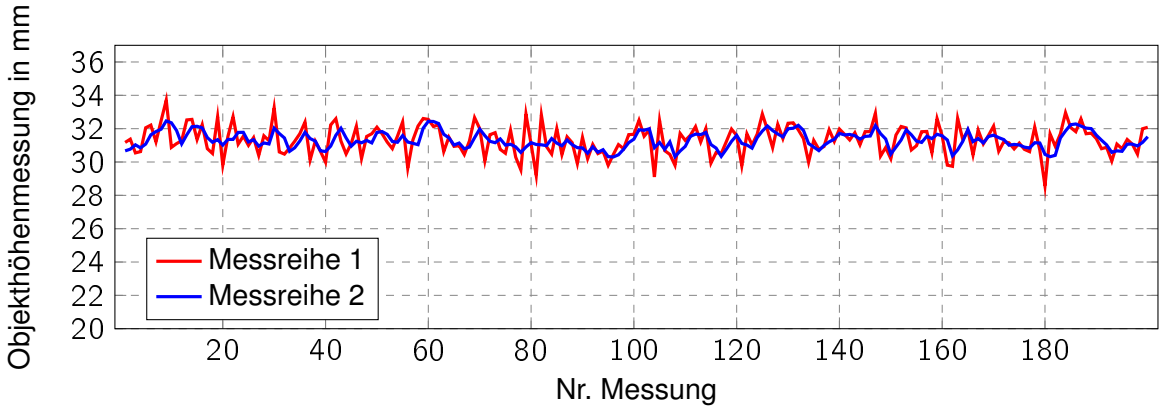

Abbildung 9.26: Objekthöhenmessung einer Box (30x100 mm) aus 70 cm Höhe

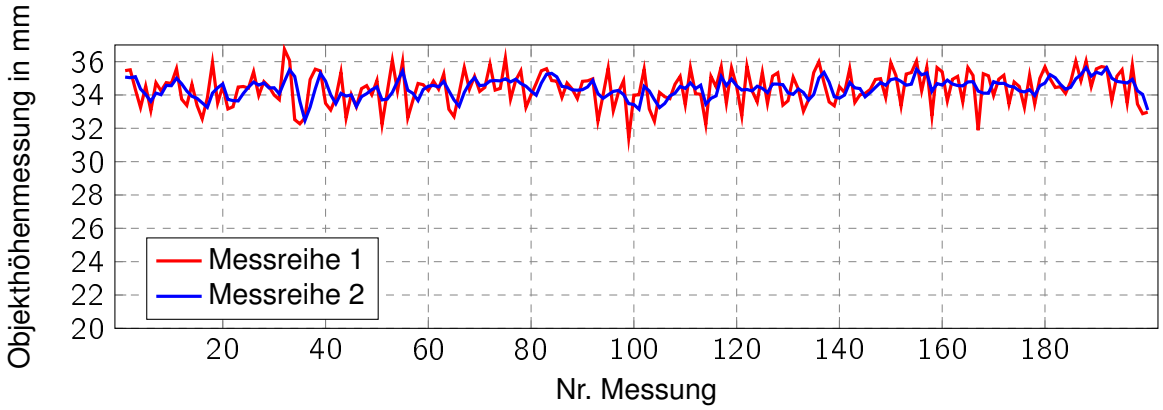

Abbildung 9.27: Objekthöhenmessung einer Box (30x100 mm) aus 1,3 m Höhe

Versicherung über die Selbstständigkeit

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

Hamburg, 16. Januar 2017	
Ort, Datum	Unterschrift

CD

CD Inhalt:
- Diese Bachelorarbeit als PDF
- Quellcode der Software
- Das Programm
- Das Programm (Demomode)

Demomode:
Das Programm mit einer archivierten Antwort vom Sensor. (Lauffähig ohne Sensor)