Technische Universität Hamburg-Harburg
Vision Systems

Prof. Dr.-Ing. R.-R. Grigat

Synchronisation der Bildakquisition von Endoskopen mit optischen Trackern

Bachelor Arbeit

von Lukas Petersen

10. Dezember 2012

Erklärung

Ich, Lukas Petersen, geboren am 30.11.1989 in Hamburg, versichere hiermit an Eides statt, dass ich diese von mir bei der Technischen Universität Hamburg-Harburg (TUHH) vorgelegte Bachelorarbeit selbstständig verfasst habe. Ich habe ausschließlich die angegebenen Quellen und Hilfsmittel benutzt.

Ort, Datum	Unterschrift

Danksagung

Ich möchte mich ganz herzlich bei Herrn Professor Grigat, sowie den Mitarbeitern des Instituts TUHH Vision Systems bedanken, dass mir die Möglichkeit zu dieser Bachelor Arbeit gegeben wurde. Ich habe viel gelernt und die Arbeit hat Spaß gemacht. Besonders gut hat mir die hervorragende Betreuung bei der Arbeit gefallen. Jederzeit hatte ich einen Ansprechpartner bei Problemen und Fragen und konnte auch einiges über andere interessante Projekte erfahren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das Problem der Synchronisation mehrerer Datenströme existiert nicht erst seit dem digitalem Zeitalter. In den 1920er- bis 1930er-Jahren gab es beispielsweise den Umschwung von Stummfilmen zu vertonten Filmen. Dazu wurde die Tonspur mit dem Lichttonverfahren [1] auf das gleiche Medium, hier eine Filmrolle, aufgebracht und so synchron zum Video wiedergegeben. Mit dem Beginn der verteilten digitalen Systeme wurde die Synchronisation immer wichtiger.

Diese Arbeit ist wie folgt aufgebaut: Kapitel 1 gibt eine Übersicht über die Problematik der Synchronisation. Kapitel 2 gibt eine Einführung in die wichtigsten mathematischen Grundlagen. In Kapitel 3 geht es darum, wie die Problemstellung gelöst wurde. Kapitel 4 beinhaltet die Quantifizierung der Ergebnisse und in Kapitel 5 wird abschließend ein Fazit gezogen und ein Ausblick auf mögliches weiteres Vorgehen gegeben.

1.1 Synchronisation im Allgemeinen

In verteilten Systemen muss auf verschiedene Techniken für die Synchronisation zurückgegriffen werden. Eine Technik ist beispielsweise die der Angleichung der Ende-zu-Ende-Verzögerung [2], bei der der Zeitpunkt des Abschickens des Paketes relevant für die Zeit des Abspielens ist. Alle Datenpakete, die zum gleichen Zeitpunkt abgeschickt werden, werden auch gleichzeitig auf der Empfängerseite abgespielt. Um entgegen der Problematik der nicht vorhersehbaren Verzögerungszeiten bei der Paketübertragung zu wirken, werden Puffer bei den Senken eingesetzt.

Eine weitere relevante Technik ist das Interleaving. Beim Interleaving werden zusammengehörige Pakete (beispielsweise Ton- und Bildspur eines Videos) in ein Paket zusammengefasst und so übertragen [2].

In dieser Arbeit befinden sich die Datenströme nicht in einem verteilten System, sondern in einem lokalen System. Allerdings wird in dieser Arbeit auf Mechanismen der Synchronisation in verteilten Systemen zurückgegriffen werden. Ein Mechanismus,

auf den zurückgegriffen wird, wird das Puffern von Daten sein. Das ist notwendig, da eine Synchronisation, wie sie in dieser Arbeit realisiert wird, durch das Puffern von Daten erst möglich ist. Auch sind für die lokale Synchronisation die Zeitpunkte des Abschickens der Datenpakete relevant, ähnlich der Methode der Angleichung der Ende-zu-Ende-Verzögerung.

Für die zeitliche Basis der Synchronisation wird eine globale Systemuhr benutzt. Dies ist ein Vorteil gegenüber verteilten Systemen. In verteilten Systemen ist die Uhrensynchronisation ein nicht triviales Problem. Für dieses Problem gibt es zwar verschiedene Lösungsansätze, die allerdings keine Synchronisation gewährleisten, welche so genau ist, wie die Verwendung der gleichen Uhr.

1.2 Trackingsystem von NDI

Das Trackingsystem, das in dieser Arbeit benutzt wurde, kommt von NDI und heißt Polaris Vicra. Das Polaris Vicra System ist ein optisches Trackingsystem, das häufig im medizinischen Bereich verwendet wird. Ein Positionssensor, der aus zwei optischen Sensoren und Infrarotdioden besteht, emittiert Infrarotlicht, welches auf die retroreflektierende Marker trifft; das reflektierte Infrarotlicht wird von den optischen Sensoren wieder aufgefangen und anhand dieser Daten wird die Position im Raum und die Orientierung berechnet. Die kompletten Spezifikationen sind in [3] zu finden.

NDI liefert zu den Trackingsystem eine Programmierschnittstelle [4], eine Bedienungsanleitung [3], verschiedene Instrumente und ein Beispielprogramm, auf welchem diese Arbeit aufbaut.

1.3 3D-Rekonstruktion

Diese Arbeit ist aus der Motivation heraus entstanden eine Auswertung für die Genauigkeit einer 3D-Rekonstruktion quantifizieren zu können. Die 3D-Rekonstruktion wird in [5] genauer beschrieben und geschieht durch einen Ansatz mit „Structure from Motion“.

Die Genauigkeit der Rekonstruktion hängt bei „Structure from Motion“ mit der Genauigkeit der geschätzten Bewegung zusammen. Diese geschätzten Bewegungen können nach Beendigung dieser Arbeit mit den Bewegungsdaten des NDI Trackingsystems verglichen werden, um so eine Aussage über die Genauigkeit treffen zu können. Dies ist möglich, da der Fehler bei „Structure from Motion“ in der Regel größer ist, als der des Trackingsystems.

Später soll eine Leber rekonstruiert werden, weswegen zum einen ein Endoskop verwendet wird und zum anderen die Testobjekte die Form einer Leber aufweisen.

1.4 Ziel der Arbeit

Ziel der Arbeit ist die Implementierung eines Programms, das auf dem von NDI gelieferten Beispielprogramm basiert. Das Programm soll die 3D-Tracker-Daten mit den Videodaten eines Endoskops synchronisieren. Die Ausgabe soll eine Textdatei mit den 3D-Daten sein, sowie ein Video, welches von dem Endoskop kommt. Es wird eine eindeutige Zuordnung von einem 3D-Datum zu einem Einzelbild des Endoskops geben.

Kapitel 2

Mathematische Grundlagen

Dieses Kapitel beschreibt die mathematischen Verfahren, welche für die Problemlösung relevant sind. In erster Linie sind Quaternionen und homogene Koordinaten für die Implementierung in Kapitel 3.2 interessant. Die Hauptkomponentenanalyse, die Projektion einer Geraden auf eine Ebene, sowie das Gram-Schmidtsche Orthogonalisierungsverfahren ist für die Bestimmung der Transformation der Polariskoordinaten in Weltkoordinaten, welche in 3.3 behandelt wird, relevant.

2.1 Synchronisation

Das für die Umsetzung in dieser Arbeit ausgewählte Verfahren basiert auf dem Prinzip der Auswahl des 2-Tupels $(\text{3D-Datum}, \text{Bild})$ mit dem betragsmäßig kleinsten zeitlichen Abstand zueinander.

Das Verfahren funktioniert wie folgt: Gegeben seien zwei Mengen $\mathcal{M}_1$ und $\mathcal{M}_2$, wobei $\mathcal{M}_1$ mit 2-Tupeln $(t_{3\text{D}_i}, \text{3D-Datum}_i)$ und $\mathcal{M}_2$ mit den 2-Tupeln $(t_{\text{Bild}_j}, \text{Bild}_j)$ gefüllt sind und $t_{3\text{D}}$ und t_{Bild} die entsprechenden Aufnahmezeiten des zugehörigen Bildes bzw. 3D-Datums sind.

Sei

$$\mathcal{M}_1 = \{(t_{3\text{D}_0}, \text{3D-Datum}_0), (t_{3\text{D}_1}, \text{3D-Datum}_1), \ldots, (t_{3\text{D}_m}, \text{3D-Datum}_m)\} \tag{2.1}$$

$$\mathcal{M}_2 = \{(t_{\text{Bild}_0}, \text{Bild}_0), (t_{\text{Bild}_1}, \text{Bild}_1), \ldots, (t_{\text{Bild}_n}, \text{Bild}_n)\} \tag{2.2}$$

und t_I das Intervall, in dem die Zuordnung $(\text{3D-Datum}, \text{Bild})$ zulässig ist.

Die Mengen werden bis zu einem Zeitpunkt t_0 untersucht, wobei nur eine Zuordnung bis $t_0 - \frac{t_I}{2}$ stattfinden kann.

Ein Bild, welches zum Zeitpunkt t_0 aufgenommen wurde, könnte gegebenenfalls ideal zu einem 3D-Datum eines Zeitpunktes $t > t_0$ passen. Diese Zuordnung würde aber nicht stattfinden, wenn die Menge bis t_0 untersucht werden würde. Deswegen wird

nur eine Zuordnung bis $t_0 - \frac{t_I}{2}$ gemacht. So ist sichergestellt, dass zu jedem Bild das komplette Intervall überprüft werden kann und eine Zuordnung stattfinden kann.

Für die Ergebnismenge $\mathcal{E}$ gilt:

$$\begin{aligned}\mathcal{E} = \{(\text{3D-Datum}_i, \text{Bild}_j) | ((\forall i | t_{\text{3D}_i} \leq t_0) \wedge (\forall j | t_{\text{Bild}_j} \leq t_0 - \frac{t_I}{2})) \\ \in ((t_{\text{3D}_i} - \frac{t_I}{2} \leq t_{\text{Bild}_j} \leq t_{\text{3D}_i} + \frac{t_I}{2}) \wedge min(|t_{\text{Bild}_j} - t_{\text{3D}_i}|))\}\end{aligned} \tag{2.3}$$

$\mathcal{E}$ beschreibt sämtliche 2-Tupel $(\text{3D-Datum}, \text{Bild})$, für die die Zeitpunkte der Aufnahmen möglichst nah beieinander und im zulässigen Intervall liegen. Für ein Bild, welches zum Zeitpunkt $t = t_0 - \frac{t_I}{2}$ aufgenommen wurde, kann auch eine Zuweisung stattfinden, da die 3D-Daten bis zum Zeitpunkt t_0 vorliegen.

2.1.1 Alternativen

Alternativ zu der in 2.1 beschriebene Methode, gäbe es die Möglichkeit der Interpolation der 3D-Daten. Unter der Annahme, dass die Bewegung näherungsweise linear zwischen zwei 3D-Daten, also in einem Zeitraum von ca. 50 ms beim NDI Polaris Vicra System, ist, könnte zwischen diesen 3D-Daten linear interpoliert werden.

Auch andere Arten der Interpolation sind denkbar (bspw. Polynominterpolation oder Splineinterpolation), die gegebenenfalls auch über mehrere zurückliegende 3D-Daten verlaufen.

Interpolationsverfahren haben den Nachteil, dass Annahmen zu der Form der tatsächlichen Bewegungen getroffen werden müssen, um das optimale Interpolationsverfahren auswählen zu können. Des Weiteren können Interpolationsverfahren eine hohe Komplexität haben, welche bei der Echtzeitverarbeitung großer Datenmengen zu zeitlichen Problemen führen könnte.

2.2 Homogene Koordinaten

Homogene Koordinaten stellen unter anderem eine Erweiterung des 3-dimensionalen Raumes dar. In [6] beschreiben Richard Hartley und Andrew Zisserman das Prinzip der homogenen Koordinaten.

Eine Rotation eines Objektes im $\mathbb{R}^3$ lässt sich mit einer $3x3$-Matrix ausdrücken. Dafür werden die Koordinaten des Objektes von rechts mit der Matrix multipliziert. Eine Translation hingegen ist eine Addition mit einem $3x1$-Vektor. Um die Translation auch mit einer Matrixmultiplikation ausdrücken zu können sind die homogenen Koordinaten notwendig.

Sei $\vec{v} = \begin{pmatrix} x & y & z \end{pmatrix}^T$ ein Vektor im $\mathbb{R}^3$. Die dazugehörigen Vektoren $\vec{\hat{v}}$ in homogenen Koordinaten sind dann dann $\vec{\hat{v}} = \begin{pmatrix} sx & sy & sz & s \end{pmatrix}^T$. Üblicherweise wird $s = 1$ gewählt, damit später die Vektoren nicht noch skalieren werden müssen. Nun lassen sich Rotationen, Translationen und weitere Transformationen als Matrixmultiplikation darstellen. Eine Rotation um die y-Achse mit dem Winkel ϕ sieht dabei wie folgt aus:

$$R_y = \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

Eine Translation hätte folgende Form:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

Nun lässt sich mit einer Matrixmultiplikation

$$\vec{\hat{v}} = T\vec{\hat{v}} \tag{2.6}$$

eine Translation um $\vec{t} = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T$ ausführen.

Analog zur Translation kann durch eine Matrixmultiplikation auch eine Rotation ausgedrückt werden.

2.3 Quaternionen

Quaternionen werden von NDI zur Darstellung der Orientierung und für Rotationen genutzt; sie besitzen verschiedene Vorteile gegenüber der Euler'schen Geometrie.

Die Definition und Rechenregeln für Quaternionen, sowie die Interpretation von Quaternionen als Rotationen werden in [7] beschrieben.

Quaternionen sind im Allgemeinen eine Erweiterung der komplexen Zahlen in das 3-dimensionale, wobei ein Quaternion q ein 4-Tupel (w,x,y,z) ist, für das gilt:

$$q := w + x \cdot i + y \cdot j + z \cdot k \tag{2.7}$$

Ein Vorteil der Darstellung einer Rotation in Quaternionen gegenüber der Darstellung als Rotationsmatrix ist, dass die Forderung keine Skalierung mit der Rotation einhergehend zu haben, durch Normierung der Rotationsquaternion zu erreichen ist. Bei einer Rotationsmatrix hingegen muss darauf geachtet werden, dass es sich um eine orthonormale Matrix mit der Determinante 1 handelt. Um aus einer nicht orthonormalen Matrix eine orthonormale Matrix zu gewinnen ist erheblich mehr Rechenaufwand nötig als bei der Normierung eines Quaternions.

Aus Quaternionen kann auch eine äquivalente Rotationsmatrix gebildet werden. Dies sieht wie folgt aus:

Sei q das Quaternion aus Formel 2.7, welches eine Rotation um eine Rotationsachse im $\mathbb{R}^3$ beschreibt, dann ist

$$R = \begin{pmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) \end{pmatrix} \tag{2.8}$$

die dazugehörige Rotationsmatrix.

2.4 Hauptkomponentenanalyse

Die Hauptkomponentenanalyse kann als orthogonale Projektion von einem Datensatz der Dimension D in einen niedriger dimensionierten Unterraum M, bei dem die Varianz der projizierten Daten maximiert wird, definiert werden. Die Daten liegen als Menge D-dimensionaler Punkte vor.

Das prinzipielle Vorgehen ist wie folgt: Erst wird die erste Achse in die Richtung der größten Varianz gelegt. Die nächste Achse ist orthogonal zu der ersten Achse und wird in die Richtung der zweitgrößten Varianz gelegt. Dies kann bist zur Dimension D weitergeführt werden.

Nun kann ein M-dimensionaler Unterraum ausgewählt werden, indem die M Achsen für den Unterraum genommen werden, die in absteigender Reihenfolge die größte Varianz haben.

In [8] lassen sich sowohl die Definition, als auch der Algorithmus an sich finden.

Alternativ zu der Hauptkomponentenanalyse gäbe es noch den RANSAC[1]-Algorithmus. Ein Vorteil des RANSAC-Algorithmus ist, dass er robuster gegenüber Ausreißer ist. Da in den hier aufgenommenen Messdaten allerdings keine Ausreißer vorzufinden sind, wird in dieser Arbeit auf die Verwendung des RANSAC-Algorithmus verzichtet.

[1] random sample consensus

2.5 Projektion einer Geraden auf eine Ebene

Dieser Abschnitt befasst sich mit der Projektion einer Geraden auf eine Ebene.

Sei $E := \vec{p} + \lambda \cdot \vec{r}_1 + \mu \cdot \vec{r}_2$ eine Ebene im $\mathbb{R}^3$ und $g := \vec{q} + \omega \cdot \vec{r}_g$ eine Gerade im $\mathbb{R}^3$, dann kann die Projektion der Geraden g auf die Ebene E wie folgt berechnet werden.

Sei $A = \begin{pmatrix} | & | \\ \vec{r}_1 & \vec{r}_2 \\ | & | \end{pmatrix}$ und $\vec{b}_1 = \vec{q}$ und $\vec{b}_2 = \vec{q} + \vec{r}_g$, wobei $\vec{b}_1$ und $\vec{b}_2$ zwei Punkte auf der Geraden sind.

Da $\vec{r}_1$ und $\vec{r}_2$ die Ebene aufspannen, müssen die Spalten von A linear unabhängig sein. Dies garantiert, dass die Inverse von $A^T A$ existiert. Nun kann die Projektionsmatrix P berechnet werden.

$$P = A(A^T A)^{-1} A^T \tag{2.9}$$

Mit dieser berechneten Projektionsmatrix können die Punkte $\vec{\hat{x}}_1$ und $\vec{\hat{x}}_2$ mit

$$\vec{\hat{x}}_1 = P\vec{b}_1 \tag{2.10}$$

und

$$\vec{\hat{x}}_2 = P\vec{b}_2 \tag{2.11}$$

berechnet werden, wobei $\vec{\hat{x}}_1$ und $\vec{\hat{x}}_2$ die projizierte Gerade

$$\hat{g} := \vec{\hat{x}}_1 + \hat{\omega} \cdot (\vec{\hat{x}}_2 - \vec{\hat{x}}_1) \tag{2.12}$$

definieren.

Das allgemeine Verfahren der Projektion auf einen Unterraum wird von Gilbert Strang in [9] genauer beschrieben.

2.6 Gram-Schmidtsches Orthogonalisierungsverfahren

Das Gram-Schmidtsche Orthogonalisierungsverfahren ist ein Algorithmus, um mehreren linear unabhängigen Vektoren zu orthonormalisieren. Auch dieses Verfahren ist in [9] zu finden.

Das Verfahren funktioniert wie folgt: Gegeben seien drei linear unabhängige Vektoren $\vec{v}_1, \vec{v}_2$ und $\vec{v}_3$. Sei

$$\vec{w}_1 = \vec{v}_1 / ||\vec{v}_1||_2 \tag{2.13}$$

der erste Vektor. Der zweite Vektor $\vec{w}_2$ wird berechnet durch:

$$\vec{\tilde{w}}_2 = \vec{v}_2 - \frac{\vec{w}_1^T \vec{v}_2}{\vec{w}_1^T \vec{w}_1} \vec{w}_1 \tag{2.14}$$

Jetzt muss $\vec{\tilde{w}}_2$ noch normiert werden.

$$\vec{w}_2 = \vec{\tilde{w}}_2 / ||\vec{\tilde{w}}_2||_2 \tag{2.15}$$

Der letzte Vektor wird analog zum vorherigen Vektor berechnet.

$$\vec{\tilde{w}}_3 = \vec{v}_3 - \frac{\vec{w}_1^T \vec{v}_3}{\vec{w}_1^T \vec{w}_1} \vec{w}_1 - \frac{\vec{w}_2^T \vec{v}_3}{\vec{w}_2^T \vec{w}_2} \vec{w}_2 \tag{2.16}$$

Auch der letzte Vektor wird normiert

$$\vec{w}_3 = \vec{\tilde{w}}_3 / ||\vec{\tilde{w}}_3||_2 \tag{2.17}$$

wodurch die berechneten Vektoren zu einer Orthonormalbasis B zusammengefasst werden können.

$$B = \begin{pmatrix} | & | & | \\ \vec{w}_1 & \vec{w}_2 & \vec{w}_3 \\ | & | & | \end{pmatrix} \tag{2.18}$$

Dieses Verfahren wird in Kapitel 3.3.2 verwendet, wobei in diesem Anwendungsfall zwei orthonormale Vektoren schon bekannt sein werden und somit nur der letzte Schritt zur Berechnung von $\vec{w}_3$ ausgeführt werden muss. Das ist wichtig, da das Gram-Schmidtsche Orthogonalisierungsverfahren aufgrund der Fehlerfortpflanzung instabil ist [10]. Für die Berechnung von nur einem Vektor spielt diese Instabilität nur eine untergeordnete Rolle.

Im Allgemeinen ist das Gram-Schmidtsche Orthogonalisierungsverfahren nicht nur auf die Dimension 3 beschränkt.

Kapitel 3

Umsetzung

Dieses Kapitel zeigt, wie die Lösung des Problems der Synchronisation umgesetzt wurde. Dabei wird zuerst der Versuchsaufbau vorgestellt, um danach auf die Implementierung einzugehen. Zuletzt wird gezeigt, wie anhand von Messwerten die Transformation der Polariskoordinaten zu Weltkoordinaten berechnet wird. In diesem Kapitel wird immer wieder auf die in Kapitel 2 erwähnten Verfahren zurückgegriffen.

3.1 Versuchsaufbau

Auf Seite 40 findet sich eine Darstellung des Versuchsaufbaus, in dem zum einen die verwendeten Instrumente samt Endoskop zu sehen sind und zum anderen die Ausrichtungen der jeweiligen Koordinatensysteme. Die x-Achsen der Koordinatensysteme sind in rot dargestellt, die y-Achsen in grün und die z-Achsen in blau. Zur Übersicht wurden die Kabel zur Ansteuerung der Positionierungsmotoren und die des Endoskops weggelassen.

In der Nähe der unteren linken Ecke der Arbeitsebene ist ein Instrument an einem Metallgestell fixiert. Dieses Instrument definiert das Polariskoordinatensystem. Dahinter befindet sich ein Messinstrument mit einer Metallspitze zum Antasten von Punkten. Das Endoskop ist mit einem motorisierten Positionierungssystem der Firma OWIS GmbH, mit Hilfe eines Metallgestells, verbunden. Damit lassen sich exakte Positionen anfahren. Dies kann für eine Auswertung der Trackergenauigkeit benutzt werden.

Das Weltkoordinatensystem befindet sich in der Nähe des Polariskoordinatensystems am Rand der Arbeitsebene. Beim Weltkoordinatensystem ist die z-Achse in die Ebene gerichtet, weswegen sie nicht zu sehen ist. Die Instrumentenkoordinatensysteme haben den jeweiligen Ursprung in der Mitte der dargestellten Marker und das motorisierten Positionierungssystem der Firma OWIS GmbH hat auch ein eigenes Koordinatensystem, welches am Sockel des unteren Motors dargestellt wird.

Die große Box ist mit Stoff an der Halterung vom Endoskop angebracht. In der Box können Objekte für die spätere 3D-Rekonstruktion positioniert werden. Die Box ist für die Abschirmung vor äußeren Einflüssen, wie beispielsweise Tageslicht, notwendig.

Der Positionssensor für die Aufnahme der Instrumente ist außerhalb der Arbeitsebene so positioniert, dass das Endoskop für sämtliche Bewegungen möglichst im Messvolumen bleibt. Angaben zum Messvolumen und der idealen Positionierung des Positionssensors sind in [3] zu finden.

3.2 Programmierung

Dieses Unterkapitel beschäftigt sich mit der Implementierung des Programms zur Synchronisation der Bildakquisition des verwendeten Endoskops mit dem optischen Trackingsystem von NDI. Dafür wurde das von NDI zur Verfügung gestellte Beispielprogramm erweitert.

Zuerst wird das Klassendiagramm (3.2.1) vorgestellt, um dann auf die einzelnen Bereiche der Programmierung, wie beispielsweise die grafische Benutzeroberfläche (3.2.2), die Endoskopansteuerung (3.2.4) und das Threading (3.2.7) eingehen zu können.

3.2.1 Klassendiagramm

Das Klassendiagramm wird zweigeteilt dargestellt. Zum einen ist auf Seite 42 das Klassendiagramm zur Ansteuerung des Endoskops, sowie zur Verwaltung und Synchronisation der Daten zu finden und zum anderen ist auf Seite 43 das Klassendiagramm zur Ansteuerung des Tackers, sowie für die grafische Oberfläche, zu sehen.

Die Klassendiagramme haben eine UML-ähnliche Struktur, die hier allerdings stark an die Programmiersprache C++ angelehnt ist, um die Funktionsdeklarationen besser darstellen zu können.

Auf der rechten oberen Seite des ersten Klassendiagramms befindet sich der `CDataStreamHandler`. Hierbei handelt es sich um eine abstrakte Klasse, die zur Datenverarbeitung verwendet wird. Von ihr werden die zwei Klassen `CVideoHandler` und `C3dDataHandler` abgeleitet. Diese beiden Klassen sind hauptsächlich für das temporäre und persistente Speichern der Endoskopdaten bzw. 3D-Daten zuständig. Um die Zeitstempel für die jeweiligen Daten zu setzen, benötigen beide Klassen einen Zeitgeber, der beim `CDataStreamHandler` und somit auch in den abgeleiteten Klassen vorliegt.

Sowohl der `CVideoHandler`, als auch der `CDataStreamHandler` haben private Kopier- und Zuweisungsfunktionen. Dies verhindert, dass Kopien der Objekte dieser Klassen erzeugt werden können. Die Notwendigkeit dafür ist gegeben, da beide Klassen auf Hardwareressourcen zugreifen und ein Kopieren der Objekte somit nicht erwünscht ist.

Auffällig ist, dass C3dDataHandler als Singleton realisiert wurde. Dies hat den Grund, dass die Daten extern dem C3dDataHandler hinzugefügt werden. Dafür müssen die zwei schwach mit einander verknüpfte Klassen CSynchronizer und CCommandHandling auf die selbe C3dDataHandler-Instanz zugreifen. Somit ist die Verwendung eines Singleton-Designmusters an dieser Stelle eine elegante Variante.

Zum Zwischenspeichern der Daten existiert eine Templateklasse namens CBuffer. Diese beinhaltet zwei Vektoren, deren Typen erst zur Kompilierzeit festgelegt werden. Beim CVideoHandler ist der erste Typ IplImage* zum Speichern der vom Endoskop kommenden Bilddaten und der zweite Typ LONGLONG für die Zeitstempel. Beim C3dDataHandler ist der zweite Typ auch für den Zeitstempel vorgesehen und somit vom Typ LONGLONG. Im ersten Vektor werden die 3D-Daten in einer Struktur mit dem Namen PointInfo hinterlegt.

Die Pufferklasse gewährleistet, dass die Daten konsistent gehalten werden. Es kann beispielsweise kein Element vom ersten Typ gelöscht werden, ohne dass das dazugehörige Element vom zweiten Typ gelöscht wird.

Die Klasse zur Synchronisation heißt CSynchronizer. Sie beinhaltet jeweils ein Objekt der C3dDataHandler- und der CVideoHandler-Klasse. Die CSynchronizer-Klasse ist zum einen zur Steuerung und Synchronisation der Datenverarbeiter-Klassen zuständig und zum anderen stellt sie die Schnittstelle zur grafischen Benutzeroberfläche dar.

Die CTimer-Klasse ist, wie die CDataStreamHandler-Klasse, abstrakt. Für die eigentlichen Aufgaben des Zeitgebers wurde der CQPCTimer implementiert. Dieser wird in 3.2.5 näher erklärt.

Im Klassendiagramm nicht vertreten sind einige Klassen, die für die Fehlerbehandlung zuständig sind, wie beispielsweise XVideoHandler oder X3dDataHandler. Diese wurden für die Übersicht weggelassen.

Das zweite Klassendiagramm zeigt, wie die Ansteuerung des Polaris Vicra Systems von NDI umgesetzt wurde. Außerdem zeigt sie noch den Hauptteil der grafischen Benutzeroberfläche.

In der oberen Mitte des Klassendiagramms befindet sich die Hauptklasse CCombinedAPISampleApp, die beim Starten des Programms ausgeführt wird. Diese erstellt einen CCombinedAPISampleDlg-Objekt. Dieser Dialog stellt die Hauptschnittstelle zum Benutzer dar. Weitere Dialoge, wie beispielsweise der für die Programmoptionen, werden in diesem Klassendiagramm nicht mit dargestellt, um das Klassendiagramm möglichst übersichtlich zu halten.

Der CCombinedAPISampleDlg-Dialog erstellt ein CCommandHandling-Objekt, welches für das Senden, Empfangen und Verarbeiten der über den COM-Port[1] kommenden Daten zuständig ist. Für die Ansteuerung des COM-Ports existiert ein Comm32Port-Objekt.

[1] Serielle Schnittstelle zur Kommunikation mit dem NDI Polaris Vicra System

Die CCommandHandling-Klasse hat auch noch ein CTimer-Objekt und ein C3d-DataHandler-Objekt. Diese sind für das Hinzufügen von 3D-Daten für die Synchronisation und Ausgabe von Nöten.

Die CCombinedAPISampleDlg-Klasse hat ein CSynchronizer-Objekt, welches die komplette Synchronisation und das Anzeigen der Endoskopdaten übernimmt und ebenfalls eine Aufnahme starten und stoppen kann.

Auf die Darstellung weiterer Dialoge, Objekte und Strukturen wurde zu Gunsten der Übersichtlichkeit verzichtet. Auch die Klassen C3dDataHandler, CSynchronizer und CTimer wurden im zweiten Klassendiagramm nur angedeutet.

3.2.2 Grafische Benutzeroberfläche

In Abbildung C.1 wird die grafische Benutzeroberfläche dargestellt. Die Benutzeroberfläche ist in 9 Bereiche eingeteilt.

Bereich 1 zeigt Informationen zu dem aktuell ausgewählten Instrument an. Außerdem kann hier auch ein Referenzsystem auswählen und Weltkoordinaten aktivieren. Das Referenzinstrument ist in der Regel das in 3.1 erklärte Instrument mit dem Polariskoordinatensystem. Wird kein Referenzinstrument ausgewählt, so werden alle Angaben relativ zum Positionssensor gemacht.

In Bereich 2 kann die Trackerinitialisierung durchgeführt werden und im dritten Bereich der grafischen Benutzeroberfläche kann die Endoskopansteuerung gestartet werden. Es wird ein zusätzliches Fenster mit dem aktuellem Endoskopbild erstellt.

In Bereich 5 wird die Aufnahme der Endoskopdaten und Trackerdaten gestartet. Es werden Ausgabedateien erstellt, für die in Bereich 4 der Name und das Verzeichnis festgelegt werden kann. Auch das aufzunehmende Instrument ist auswählbar.

Bereich 6 ist für die Aufnahme von einzelnen Punkten ohne Synchronisation. Auf die Funktionalität wird in 3.4 eingegangen.

In Bereich 7 werden die aktuellen Daten des Trackers angezeigt und im 8. Bereich sind zusätzliche Informationen zu den aktuellen Messinstrumenten zu sehen.

In Teil 9 sind sämtliche Einstellungen zum NDI System und zur Synchronisation zu finden.

Die grafische Benutzeroberfläche wurde mit MFC[2] realisiert. Da das Beispielprogramm von NDI schon mit MFC realisiert war, wurde nicht auf eine andere Bibliothek für die Visualisierung zurückgegriffen.

3.2.3 Trackeransteuerung

Für die Ansteuerung des Trackers wurde von Polaris eine Dokumentation für die Programmierschnittstelle [4] und ein Beispielprogramm mitgeliefert.

[2] Microsoft Foundation Classes

Wie schon in 3.2.1 beschrieben, ist die `CCommandHandling`-Klasse zusammen mit der `Comm32Port`-Klasse zuständig für die Trackeransteuerung. Diese wurden von dem mitgelieferten Beispielprogramm verwendet.

Eine Eigenart der Trackeransteuerung ist, dass bei der Anforderung der Trackerdaten vom Tracker nicht gewartet wird, bis ein neues 3D-Datum vorhanden ist, sondern das jeweils aktuellste Datum zurückgegeben wird. Werden also in einem Zeitraum von unter 50 ms mehrmals Daten vom Tracker angefordert, so werden redundante Daten zurückgeschickt.

Da die Bearbeitung der Rohdaten des Trackers und die Anfrage und Antwort des NDI Systems eine gewisse Zeit dauern, können die Anfragen mit ca. 80 bis 100 Hz stattfinden. Es werden also ca. 3 bis 4 mal hintereinander redundante Daten vom NDI-System empfangen. Dies hat auch zur Folge, dass die 3D-Daten nicht mit tatsächlichen 20 Hz abgefragt werden können, sondern die Abfragerate am schwanken ist. Das Empfangen von redundanten Daten ist interessant für das Setzen des Zeitstempels, da bei der Synchronisation nur das 3D-Datum mit dem exaktesten Zeitstempel benutzt wird, also das erste, welches vom Tracker kommt.

3.2.4 Endoskopansteuerung

Bei dem verwendetem Endoskop handelt es sich um ein USB-Endoskop der Firma Adrolook Endoskope [11]. Da es sich um ein USB-Endoskop handelt, ist eine Ansteuerung mit der Open-Source-Bibliothek OpenCV [12] möglich.

Wie eine USB-Kamera genau ansteuert wird, ist in [13] zu finden. Ein Detail, welches vor Allem bei der Zeitstempel Strategie (3.2.5.2) wichtig ist, ist in den Quelltextausschnitten C.4 und C.5 zu finden.

In C.4 wurden die Befehle `cvGrabFrame(...)` und `cvRetrieveFrame(...)` für die Bildakquisition verwendet, wohin gegen in C.5 nur der Befehl `cvQueryFrame(...)` verwendet wurde. Der Befehl `cvQueryFrame(...)` ist dabei nur eine Kombination aus `cvGrabFrame(...)` und `cvRetrieveFrame(...)`.

Vorteil der ersten Methode ist, dass zu dem Zeitpunkt von `cvGrabFrame(...)` schon das eigentliche Bild vorhanden ist und somit eine präzisere Annahme gemacht werden kann, zu welchem Zeitpunkt das Bild vom Endoskop aufgenommen wurde. Wenn `cvRetrieveFrame(...)` aufgerufen wird, wird das angekommene Bild so verarbeitet (u.a. dekomprimiert), dass es im Programm weiter verwendet werden kann.

Das Empfangen und Verarbeiten der Bilddaten ist nur dem Empfangen gegenüber um einen Faktor von ca. 1600 langsamer[3]. Gerade für eine zeitkritische Anwendung, wie es bei dieser Arbeit der Fall ist, ist es hilfreich, so präzise wie möglich die Annahme der Aufnahmezeit machen zu können.

[3] Jeweils mehr als 1000 Messungen ergaben für `cvGrabFrame(...)` eine mittlere Zeit von ca. 0.019 ms und für `cvQueryFrame(...)` eine mittlere Zeit von ca. 30.635 ms.

Weiteres dazu befindet sich in Kapitel 3.2.5.2.

3.2.5 QueryPerformanceCounter

Seit Windows 2000 Professional bietet Microsoft eine Schnittstelle zum hochauflösenden QueryPerformanceCounter [14]. Die Auflösung des QueryPerformanceCounters hängt vom verwendeten System ab. Sie ist allerdings in der Regel sehr viel höher als die Alternativen (siehe: 3.2.5.1).

Allerdings bringt der QueryPerformanceCounter auch ein großes Problem mit sich. Laut Microsoft könnte es bei Multiprozessorsystemen aufgrund von Fehlern im BIOS[4] dazu kommen, dass verschiedene Prozesse bzw. verschiedene Threads auf unterschiedliche Hardware Timer zugreifen, die nicht synchron sind [14]. Der Zugriff auf unterschiedliche Hardware Timer kommt vor, wenn ein Prozess oder Thread auf einem anderen Prozessor oder Prozessorkern ausgeführt wird, als der andere. Wenn ein Zeitstempel in einem Thread zum Zeitpunkt t_0 aufgenommen wird und ein weiterer in einem anderen Thread zu einem späteren Zeitpunkt t_1 kann es sein, dass aufgrund der unterschiedlichen Hardware Timer laut den Zeitstempeln $t_1 < t_0$ gilt.

Für die Lösung dieses Problems gibt Microsoft einen Ansatz [14]. Es wird empfohlen, den Thread, der auf den QueryPerformanceCounter zugreift, auf einen fest definierten Prozessor zu legen. Um trotzdem die Idee der parallelen Abläufe beizubehalten, wird der Thread nach dem Holen des Zählerstandes wieder zurück auf den ursprünglichen Prozessor gelegt; so werden die Threads einen Großteil der Zeit auf unterschiedlichen Prozessoren ausgeführt, um eine maximale Parallelität zu erhalten. Wie das in der Implementierung gelöst wurde ist in Abbildung C.2 zu finden.

3.2.5.1 Alternative Zeitgeber

Alternativ zum QueryPerformanceCounter bietet Microsoft weitere Schnittstellen zu Zeitgebern [15].

Funktion	**Einheit**	**Genauigkeit**
Now, Time, Timer	Sekunden	1 s
GetTickCount	Millisekunden	ca. 10 ms
TimeGetTime	Millisekunden	ca. 10 ms
QueryPerformanceCounter	QueryPerformanceFrequency	$< 1\ \mu$s[5]

Tabelle 3.1: Zeitgeber Genauigkeitsstufen

[4] Basic Input Output System

[5] Auf dem Testsystem mit einem Intel Xeon X3470-Prozessor. Tests mit anderen Systemen lieferten Werte im ähnlichen Bereich.

Wie zu sehen ist haben die verschiedenen Zeitgeber eine maximale Genauigkeit von 10 ms. Das verwendete Endoskop [11] liefert allerdings 30 Bilder/Sekunde, also ca. alle 33 ms ein neues Bild. Damit wäre ein Zeitstempel mit einer Genauigkeit von 10 ms nicht genau genug, um eine vernünftige Synchronisation durchführen zu können.

Die logische Konsequenz aus diesen Daten ist die Verwendung des QueryPerformanceCounters.

3.2.5.2 Zeitstempel Strategie

Die Zeitstempel für die 3D-Daten werden, genau so wie der für die Bildakquisition, so früh wie möglich im Programm gesetzt, um eine möglichst exakte Annahme über den Aufnahmezeitpunkt zu erhalten. Möglichst früh heißt, dass sie direkt nach dem Empfangen der Daten und noch vor der Weiterverarbeitung gesetzt werden.

Bei den Zeitstempeln für die 3D-Daten und für die Endoskopdaten sieht das wie folgt aus:

```
1 Zeitstempel1 = jetzt
2 fordere Daten an
3 if Daten angekommen
4         Zeitstempel2 = jetzt
5 Zeitstempel = Zeitstempel1 + (Zeitstempel2-Zeitstempel1)/2
```

Abbildung 3.1: Zeitstempel für die 3D- und Video-Daten

In 3.2.4 wurde schon auf den Vorteil der Verwendung von `cvGrabFrame(...)` und `cvRetrieveFrame(...)` gegenüber `cvQueryFrame(...)` eingegangen. Das Anfordern der Daten bei den Endoskopdaten wird folglich mit `cvGrabFrame(...)` realisiert.

In beiden Fällen wird angenommen, dass der Zeitpunkt der Aufnahme des jeweiligen Datums zwischen dem Anfordern und dem Empfang der Daten liegt. Zumindest bei den Endoskopdaten ist diese Annahme aber nicht richtig. In 4.3.3 wird auf den zeitlichen Offset bei der Bildakquisition eingegangen. Für den Zeitstempel der Endoskopdaten hat das zur Folge, dass bei dem finalen Zeitstempel der Offset noch abgezogen wird.

3.2.6 Synchronisation

Das Flussdiagramm zur Synchronisation ist in Abbildung C.3 zu sehen.

Am Anfang wird der Zeitstempel gesetzt, bis zu dem Zeitpunkt, an dem die Daten synchronisiert werden sollen. Danach wird die halbe Intervallzeit abgewartet, damit jedem Bilddatum auch 3D-Daten nach dem Zeitstempel zugewiesen werden können.

Es werden die Zeiten von den Bilddaten bis zum Zeitstempel angefordert und die Zeiten der 3D-Daten bis zum Zeitstempel plus der halben Intervallzeit.

Jetzt werden die Zeiten der 3D-Daten daraufhin überprüft, ob sie im passenden Intervall zum aktuellem Bildzeitpunkt liegen. Ist dies der Fall, landet der entsprechende Index auf einer Kandidatenliste. Sind alle 3D-Daten-Zeiten überprüft, wird der Kandidat mit dem zeitlich kleinsten betragsmäßigen Abstand ausgewählt und zum Speichern vorgemerkt. Dieser Vorgang wird mit den nächsten Zeitstempeln der Endoskopdaten solange wiederholt, bis keine weiteren Daten mehr vorhanden sind.

Sind sämtliche Daten verarbeitet, werden die Ausgabethreads gestartet. Diese löschen auch die Daten aus den entsprechenden Puffern bis zu den entsprechenden Zeitpunkten. Die Bilddaten werden bis zum Zeitpunkt des Setzen des Anfangzeitstempels gelöscht und die 3D-Daten bis zum Zeitpunkt des Setzten des Anfangzeitstempels abzüglich des halben Intervalls.

Ist dies abgeschlossen, kann die Synchronisation von vorne anfangen, da parallel die Puffer weiter gefüllt werden und somit neue Daten vorhanden sein sollten.

Ein Vorteil bei dem verwendeten Algorithmus zur Synchronisation ist, dass er auch für andere beliebige Datenströme verwendbar ist. In 2.1.1 wurde auf eine Strategie zur Synchronisation eingegangen, die aufgrund der speziellen Eigenschaften der 3D-Daten (Interpolierbarkeit) eine Alternative gewesen wäre. Im folgenden Kapitel wird auch noch eine weitere Möglichkeit zur Synchronisation vorgestellt.

3.2.6.1 Alternativen

Eine Alternative zu der oben beschriebenen Synchronisation wäre gewesen, sequenziell erst ein Paket des einen Datenstroms zu empfangen, was dann ein Anfordern eines Paketes des anderen Datenstroms auslöst. Mit Zeitstempeln könnte auch quantifizieren werden, wie gut diese Methode ist. Gegen diese Methode spricht, dass die Synchronisation dabei nicht beliebig genau werden kann, da es stets zu einer Verzögerung zwischen dem Empfang eines Paketes des einen Datenstroms und dem Empfang eines Paketes des anderen Datenstroms kommt. Außerdem würde es zu Problemen kommen, wenn beispielsweise der erste Datenstrom mehr Daten pro Sekunde bereitstellen kann als der zweite Datenstrom. Aus diesem Grund würde auch eine Änderung der Hardware nicht ohne weitere Anpassungen möglich sein.

Auch könnte die in 2.1.1 beschriebenen Methoden der Interpolation implementiert werden. Um diese Möglichkeit der Synchronisation in der Nachbereitung offen zu halten, gibt es die Möglichkeit sich die 3D-Daten inklusive Zeitstempel und zu den Videodaten die passenden Zeitstempel in einer Datei ausgeben zu lassen.

3.2.7 Threading

MFC bietet, wie von Microsoft in [16] beschrieben, zwei Arten von Threads an. Die erste Art ist der sogenannte Benutzeroberflächenthread. Dieser wird hauptsächlich zur

Darstellung und Interaktion mit dem Benutzer verwendet. Die andere Art ist der Arbeitsthread, welcher in 3.2.7.1 näher erläutert wird.

Threads wurden bei der Implementierung benutzt, um möglichst große Nebenläufigkeit zu erhalten und bei Mehrkernsystemen echte Parallelität zu erreichen. Diese ist bei der Synchronisation der Video- mit den 3D-Daten von Vorteil. Die entsprechenden Puffer können parallel gefüllt werden, die Ausgabe verarbeiteter Daten kann parallel ausgeführt werden und auch die Synchronisation an sich kann nebenläufig geschehen.

Auf Vorteile der Nebenläufigkeit gegenüber der sequenziellen Anforderung von Paketen wurde in 3.2.6.1 eingegangen. Der Vorteil bei der Ausgabe der Daten entspricht dem selbigen bei der Verarbeitung der Daten.

Die Probleme, die Threading mit sich bringt, sowie Strategien zur Vermeidung dieser Probleme, werden in 3.2.7.2 erläutert.

3.2.7.1 Arbeitsthreads

Arbeitsthreads sind, im Gegensatz zu den Benutzeroberflächenthreads, für Berechnungen im Hintergrund konzipiert. Bei dem in dieser Arbeit erstellten Programm werden an mehreren Stellen explizit Arbeitsthreads gestartet.

Zum einen wird in der `C3dDataHandler`-Klasse bei der Ausgabe auf einen Arbeitsthread zurückgegriffen, zum anderen wird auch die Funktion `add(...)` der `C3dDataHandler`-Klasse stets aus einem eigenen Thread aufgerufen.

In der `CVideoHandler`-Klasse befinden sich ähnliche Funktionen, die nebenläufig ausgeführt werden. Zum einen existiert auch in der `CVideoHandler`-Klasse die Funktion `startOutputThread(...)` für die Ausgabe der Daten, zum anderen werden auch die vom USB-Endoskop kommenden Daten parallel in einem eigenen Thread angefordert. Dafür ist die Funktion `CVideoHandler::updateFrameThread(...)` implementier. Sie ist außerdem noch für die Darstellung der Bilddaten, die vom Endoskop kommen, zuständig.

Wie in 3.2.7 schon beschrieben, findet die eigentliche Synchronisation nebenläufig statt. Da für die Synchronisation die Klasse `CSynchronizer` zuständig ist, ist auch dort der Arbeitsthread `UINT CSynchronizer::calcThread(...)` zu finden.

Mit der Darstellung in der Benutzeroberfläche, der Aufnahme und Verarbeitung der Daten, sowie der Darstellung der vom Endoskop kommenden Bilddaten, laufen während einer Aufnahme bis zu 11 Threads parallel. Wird zusätzlich noch die Möglichkeit wahrgenommen, statt unkomprimierter Videodaten einen Codec zur Videoausgabe zu verwenden, so steigt die Anzahl an Threads noch einmal.

3.2.7.2 Schutzmechanismen

Microsoft [16] empfiehlt für den Fall, dass mehrere Threads derselben Anwendung gleichzeitig auf eine Ressource zugreifen wollen, die Verwendung einer Semaphore.

Dafür bietet MFC die Klasse `CSemaphore` an. Diese Klasse kann den Zugriff auf gemeinsame Ressourcen blockieren, solange ein anderer Thread den exklusiven Zugriff hält. Auch kann der `CSemaphore`-Klasse die Anzahl der Threads übergeben werden, die maximal parallel auf die geschützten Daten zugreifen können. In dieser Arbeit sind allerdings sämtliche Semaphoren so ausgelegt, dass stets nur ein Thread auf die geteilten Ressourcen zugreifen können.

Im Detail sieht die Verwendung von Semaphoren stets wie folgt aus:

```
1 void CVideoHandler::startRecord()
2 {
3         m_semRecord->Lock();
4         m_record = true;
5         m_semRecord->Unlock();
6 }
```

Abbildung 3.2: Beispiel für eine Verwendung von Semaphoren — Setzen einer geschützten Ressource

In diesem Beispiel wird erst die Semaphore, die für die Membervariable `m_record` zuständig ist, gesperrt, um dann `m_record` einen neuen Wert zuzuweisen. Ist dieser Wert zugewiesen wird die Ressource wieder freigegeben und kann auch wieder in anderen Threads verwendet werden.

Würde die Verwendung von Schutzmechanismen beim Threading weglassen werden, so würde es zu undefinierten Verhalten kommen, beispielsweise wenn gleichzeitig ein Thread versucht eine Variable zu lesen, während ein anderer Thread diese Variable verändert.

Auf die möglichen Gefahren, die Threading und insbesondere Semaphoren und andere Schutzmechanismen mit sich bringen muss aufgepasst werden. Eine Gefahr wäre ein Deadlock, der auftreten kann, wenn der eine Thread eine Ressource hält, die der anderer Thread benötigt, und währenddessen auf eine andere Ressource wartet, die der andere Thread hält. Um einen solchen Deadlock der Threads zu verhindern kann bei komplexeren Programmen auf verschiedene Strategien zurückgegriffen werden. Bei dieser Arbeit hat allerdings eine vernünftige Planung und Umsetzung, sowie die vergleichsweise geringe Komplexität des Threadings[6] gereicht, um den Problemen, die das Threading mit sich bringt, ausweichen zu können.

[6] In der Regel greifen nicht mehr als zwei Threads gleichzeitig auf eine Ressource zu.

3.3 Bestimmung der Transformation von Polariskoordinaten zu Weltkoordinaten

Dieses Unterkapitel beschäftigt sich mit der Vermessung der Arbeitsebene, sowie der Berechnung der Transformation T_{PW}[7]. Dabei wurde wie folgt vorgegangen:

Erst wurde die Ebene bestimmt. Danach wurden die Achsen vermessen und am Ende wurden die einzelnen Ergebnisse zusammengefügt und die Transformation $T_{\mathrm{PW}} = [R|\vec{t}]$ aufgestellt.

3.3.1 Ebenenvermessung

Um die Ebene, auf der der Versuch aufgebaut ist, bestimmen zu können, wurde sie zunächst vermessen.

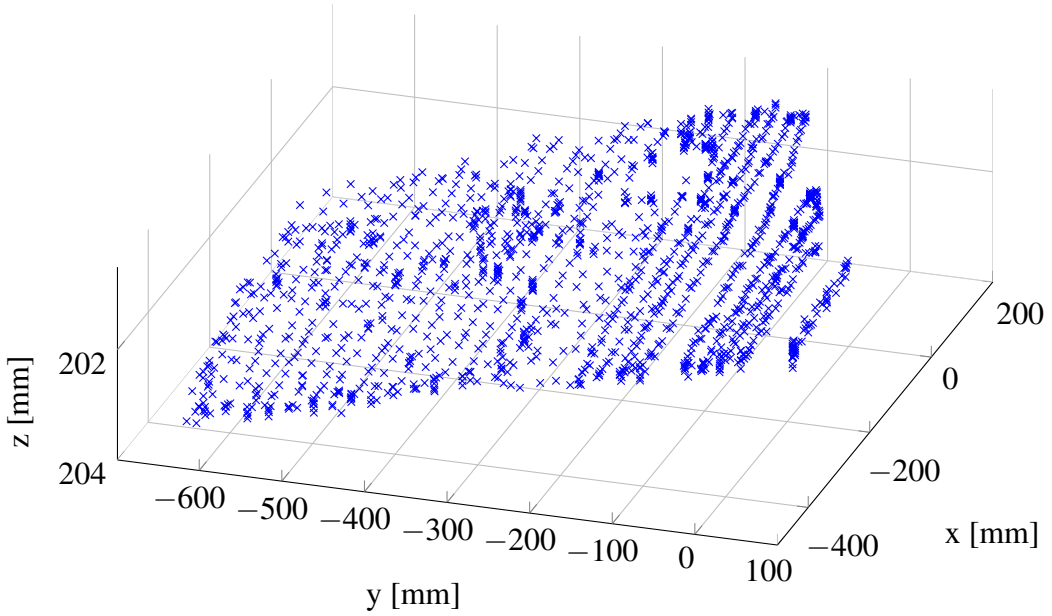

Abbildung 3.3: Vermessung der Ebene mit 8031 Messwerten in Polariskoordinaten

Wie in Abbildung 3.3 zusehen ist, ist die Arbeitsebene zum Polariskoordinatensystem leicht verkippt.

Nach der Vermessung der Ebene kann nun eine Hauptkomponentenanalyse durchgeführt werden. Sie liefert, wie in 2.4 beschrieben, orthogonale Vektoren in die Richtungen der größten Varianzen. Die ersten zwei Vektoren spannen dabei die Ebene auf.

[7] Transfomation Polariskoordinaten zu Weltkoordinaten

Jetzt kann aus den beiden Vektoren das Kreuzprodukt berechnet werden, um den Normalenvektor $\vec{n}$ zu bestimmen. Ist $\vec{n}$ bestimmt, wird noch ein Punkt auf der Ebene benötigt, um die Ebene vollständig zu beschreiben. Hierfür wurde der Mittelpunkt aller Messwerte $\vec{p}$ hergenommen.

Somit ist die Ebene mit

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \vec{n} = \vec{p} \cdot \vec{n} \tag{3.1}$$

definiert. Veranschaulicht wird das ganze in Abbildung 3.4.

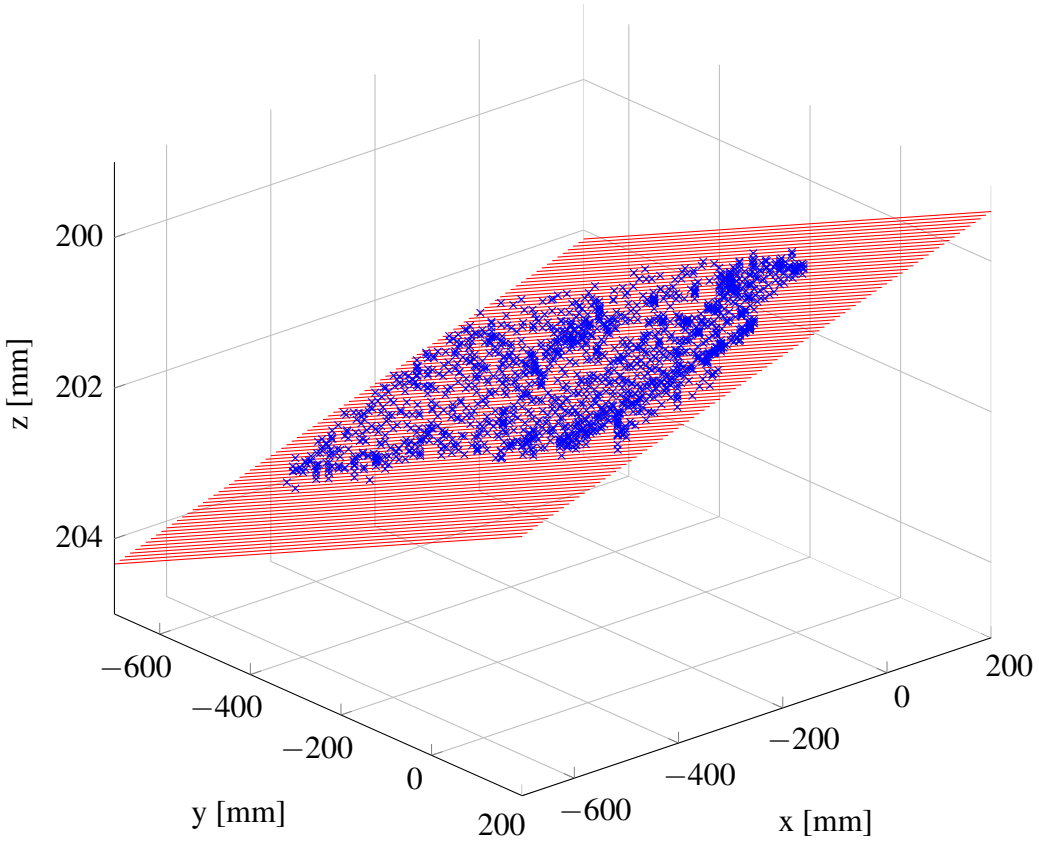

Abbildung 3.4: Messwerte und die daraus resultierende Ebene in Polariskoordinaten

Die euklidischen Abstände der Messpunkte zur Ebene betragen nun im Mittel $\mu_{\text{Abstand}} = 0$ mm. Die Varianz beträgt $\sigma^2_{\text{Abstand}} = 0.009$ mm^2 und der maximale euklidische Abstand eines Messpunktes zur Ebene ist kleiner als 0.46 mm.

3.3.2 Achsenvermessung

Da die Ebene nun bestimmt ist müssen jetzt noch die Achsen bestimmt werden. Das Weltkoordinatensystem wurde, wie in 3.1 beschrieben, in eine Ecke der Arbeitsebene gelegt. Die Achsen verlaufen entlang des 250 mm Lochrasters.

Mit einem Metallgestell, welches sicherstellt, dass die Messung über den Löchern der Achse stattfindet, und einem Messinstrument wurden nun die Achsen vermessen. Die Messung ist in Abbildung 3.5 dargestellt.

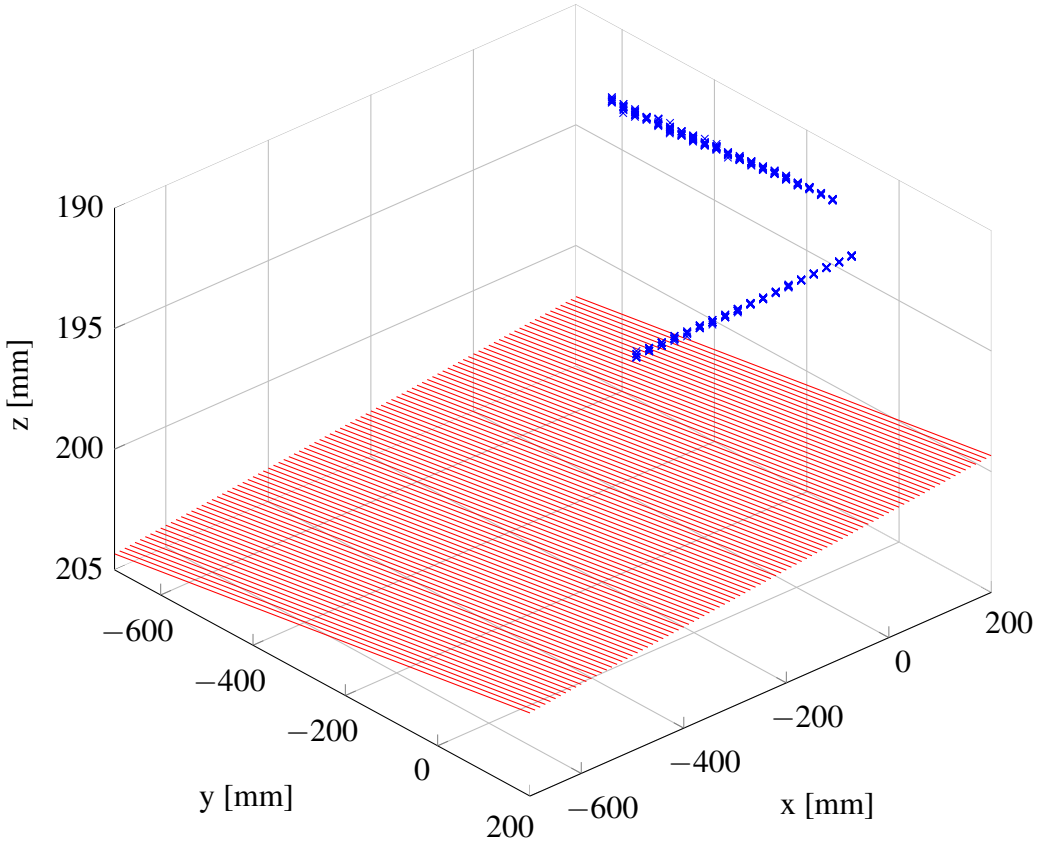

Abbildung 3.5: Vermessung der Achsen mit 5941 Messwerten in Polariskoordinaten

Aufgrund des Metallgestells liegen die Messwerte nicht in der Ebene. Deswegen müssen die berechneten Geraden, wie in 2.5 beschrieben, in die Ebene projiziert werden.

Um aus den Messwerten Geraden zu erstellen wurde erneut die in 2.4 beschriebene Hauptkomponentenanalyse angewandt. Dafür wurden die Messwerte der einzelnen Achsen separat behandelt. Der jeweils erste Vektor mit der größten Varianz beschreibt hierbei die entsprechenden Richtungsvektoren $\vec{r}_{A1}$ und $\vec{r}_{A2}$ der jeweiligen Achse. Danach werden die Geraden in die Ebene projiziert.

Das Projizieren der Geraden des Weiteren noch den Vorteil, dass die Geraden orthogonal zum Normalenvektor $\vec{n}$ sind, was bei der Bestimmung der Basis des Weltkoordinatensystems in 3.3.3 zum Tragen kommt.

Es gibt zwei Möglichkeiten, um den Ursprung des Weltkoordinatensystems zu ermitteln. Die erste Möglichkeit ist es ein Messinstrument auf den Ursprung des Weltkoordinatensystems zu legen und diesen durch eine Messung zu bestimmen. Eine andere Möglichkeit wäre es gewesen den Schnittpunkt der beiden projizierten Geraden zu berechnen.

Aufgrund der Tatsache, dass bei der Achsenvermessung und der darauf folgenden numerischen Verfahren sich ein Fehler fortpflanzen könnte, wurde das erste Verfahren gewählt. Dieser gemessene Punkt wurde dann wieder auf die Ebene projiziert.

$$a_1 := \vec{t} + \lambda \cdot \vec{r}_{\mathrm{A1}} \tag{3.2}$$

und

$$a_2 := \vec{t} + \mu \cdot \vec{r}_{\mathrm{A2}} \tag{3.3}$$

definieren nun die Achsen, die in 3.6 dargestellt werden, wobei $\vec{t}$ der gemessene Punkt ist. Dabei ist a_1 die längere der beiden Achsen.

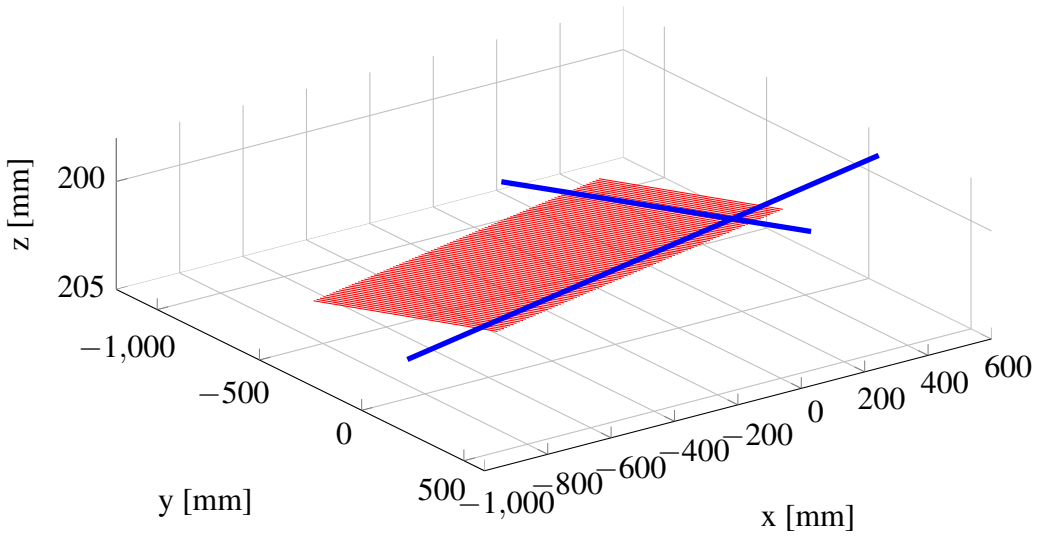

Abbildung 3.6: Projizierte Achsen in Polariskoordinaten

Der euklidische Abstand von dem gemessenen Punkt $\vec{t}$ zum berechneten Schnittpunkt der auf die Ebene projizierten Achsen beträgt ca. 0.5 mm. Unter der Annahme, dass der gemessene Punkt exakter ist, wird dieser für die weitere Berechnung der Transformation T_{PW} verwendet.

3.3.3 Basis des Weltkoordinatensystems

Die Basis B_W des Weltkoordinatensystems ist orthonormal. Um aus den gemessenen Achsen eine orthonormale Basis bestimmen zu können, wird an dieser Stelle das Gram-Schmidtsche Orthogonalisierungsverfahren (siehe 2.6) verwendet. Unter der Annahme, dass die Vermessung der längeren Achse mit mehr Messungen exakter ist und der Annahme, dass $\vec{n}$, der Normalenvektor der Ebene korrekt ist, kann nun der letzte Basisvektor bestimmt werden.

Dafür wird folgende Rechnung ausgeführt:

$$\vec{\tilde{w}}_3 = \vec{r}_{\mathrm{A2}} - \frac{\vec{n}^T \vec{r}_{\mathrm{A2}}}{\vec{n}^T \vec{n}} \vec{n} - \frac{\vec{r}_{\mathrm{A1}}^T \vec{r}_{\mathrm{A2}}}{\vec{r}_{\mathrm{A1}}^T \vec{r}_{\mathrm{A1}}} \vec{r}_{\mathrm{A1}} \tag{3.4}$$

Auch dieser Vektor wird normiert:

$$\vec{w}_3 = \vec{\tilde{w}}_3 / ||\vec{\tilde{w}}_3||_2 \tag{3.5}$$

Die berechnete Orthonormalbasis mit

$$\vec{w}_2 = \vec{r}_{\mathrm{A2}} / ||\vec{r}_{\mathrm{A2}}||_2 \tag{3.6}$$

$$\vec{w}_1 = \vec{n} / ||\vec{n}||_2 \tag{3.7}$$

lautet nun

$$B_W = \begin{pmatrix} | & | & | \\ \vec{w}_3 & \vec{w}_2 & \vec{w}_1 \\ | & | & | \end{pmatrix} \tag{3.8}$$

Bei der Aufstellung der Orthonormalbasis muss auf die richtige Reihenfolge geachtet werden, um die jeweiligen Achsen richtig abbilden zu können.

Die nun berechnete zweite Achse hat eine Abweichung gegenüber der gemessenen Achse von ca. $\vec{w}_3 \sphericalangle \vec{r}_{\mathrm{A2}} = 0.3891^\circ$.

3.3.4 Zusammenfügen der Ergebnisse

Mit der berechneten Basis B_W und dem gemessenen Ursprung $\vec{t}$ kann nun die Rotation und Translation vom Polariskoordinatensystem zum Weltkoordinatensystem dargestellt werden.

Für die gesamte Transformation T_{PW} gilt:

$$T_{\mathrm{PW}} = \left[B_W | \vec{t} \right] \tag{3.9}$$

Nun können beliebige Punkte $\vec{x}_{\text{Polaris}}$ (dargestellt in homogenen Koordinaten) mit einer einfachen Matrixmultiplikation

$$\vec{x}_{\text{Welt}} = T_{\text{PW}} \cdot \vec{x}_{\text{Polaris}} \tag{3.10}$$

transformiert werden.

Diese Multiplikation ist nur möglich, da homogene Koordinaten benutzt werden, bei denen $s = 1$ (vgl. 2.2) gilt. Dadurch, dass keine Skalierung mit $1/s$ von $\vec{x}_{\text{Polaris}}$ ausgeführt werden muss, kommt bei dieser Matrixmultiplikation ein 3x1-Vektor heraus, der das transformierte 3D-Datum repräsentiert.

3.4 Weitere Funktionalität

Zusätzlich zur Synchronisation gibt es noch die Möglichkeit einzelne 3D-Punkte und fortlaufende 3D-Punkte aufzunehmen. Dafür ist Bereich 6 in Abbildung C.1 zuständig.

Es kann ein Messinstrument ausgewählt werden, welches aufgenommen werden soll. Zusätzlich kann das Messinstrument auch mit einem Offset versehen werden. Dies ist beispielsweise für die Messung mit dem von NDI gelieferten Messinstrument mit Messspitze vorgesehen. Eine kontinuierliche Aufnahme von Messpunkten ist ebenfalls möglich. Der Export der aufgenommenen 3D-Daten in eine Textdatei ist mit einem Klick auf „Save to file“ machbar.

Die Messwerte aus 3.3 wurden mit dieser Funktionalität aufgenommen. Ein weiteres Beispiel ist in den Abbildungen 3.7 und 3.8 zu sehen. Dort wurde ein Lebermodell abgetastet und visualisiert.

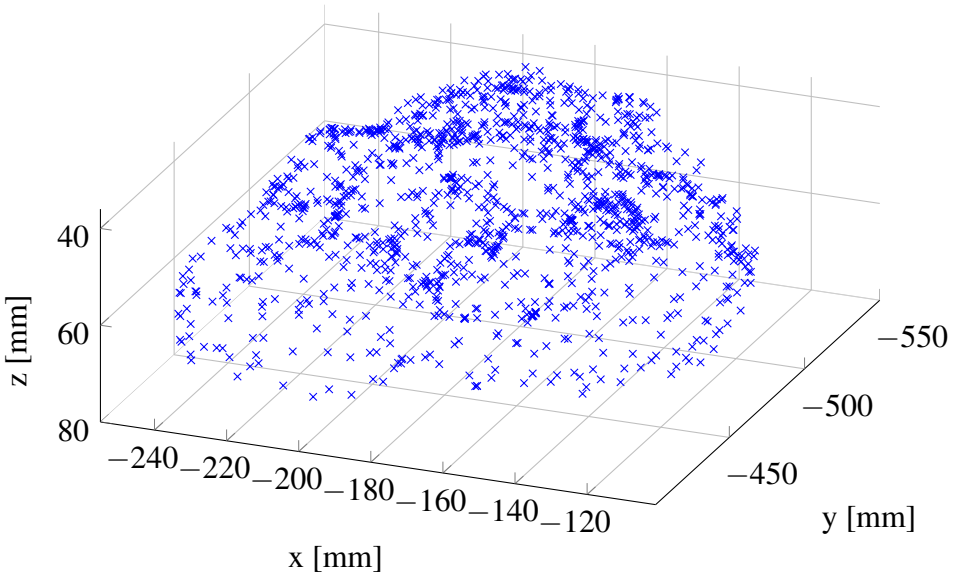

Abbildung 3.7: Vermessung eines Lebermodells

Die abgetasteten Werte wurden zu Darstellungszwecken wieder auf das ursprüngliche Modell projiziert.

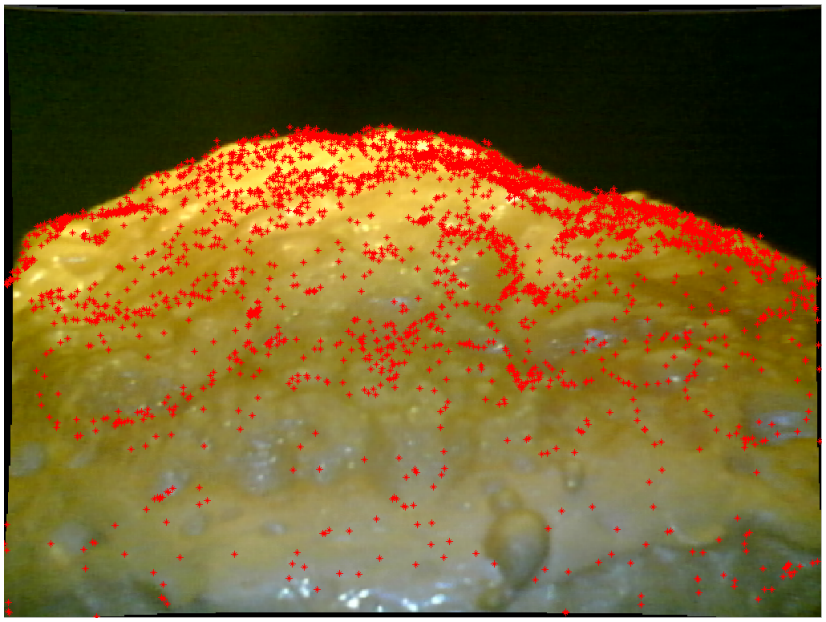

Abbildung 3.8: Rückprojektion der Messwerte auf das Lebermodell

Das zur Abbildung gehörige Video zeigt, dass die Punktwolke, die durch Abtasten

der Schaumstoffleber entstand, über die ganze Zeit an die richtige Stelle zurückprojiziert wird. Bei dem Video ist allerdings auch ein Rauschen zu erkennen, welches zum einen durch den zeitlichen Fehler und zum anderen durch den Messfehler des NDI Systems kommt. Dieses Rauschen äußert sich durch ein Schwanken der Punktwolke.

Zur Kamerakalibrierung gibt es auch die Möglichkeit ein einzelnes Bild mit passenden 3D-Datum aufzunehmen. Diese Funktionalität findet nicht synchron statt. Es werden nur die jeweils aktuellsten Daten ausgegeben. Auch der zeitliche Fehler wird gespeichert, wodurch eine Aussage über die Synchronität der Daten gemacht werden kann.

Kapitel 4

Quantifizierung

Dieses Kapitel befasst sich in 4.1 mit der Quantifizierung des Trackingfehlers, in 4.2 mit dem Fehler der Transformation der Polariskoordinaten zu Weltkoordinaten, in 4.3 mit dem zeitlichen Fehler der Synchronisation und stellt in 4.4 anschaulich die Auswirkung des zeitlichen Fehlers und die des Positionsfehlers dar.

4.1 Trackingfehler

Der optische Tracker hat einen Positionsfehler, welchen die Schnittstelle von NDI auch angibt. Dieses Unterkapitel befasst sich mit der Auswirkung und Verteilung des Trackingfehlers bei der Positionsbestimmung.

4.1.1 Theoretischer Trackingfehler

Laut [3] hat das System von NDI einen mittleren Fehler bei der Positionsbestimmung von 0.25 mm. Diese 0.25 mm sind bezogen auf den euklidischen Abstand zwischen der aufgenommenen Position und der tatsächlichen Position des Messinstruments. Für diese Messungen hat NDI einen Marker an statistisch repräsentativen Stellen im Messvolumen positioniert und jeweils 30 Messungen gemacht.

4.1.2 Messungen des Trackingfehlers

Wie in 4.1 zu sehen ist, scheint der Fehler, den das Polaris Vicra System angibt, normalverteilt zu sein.

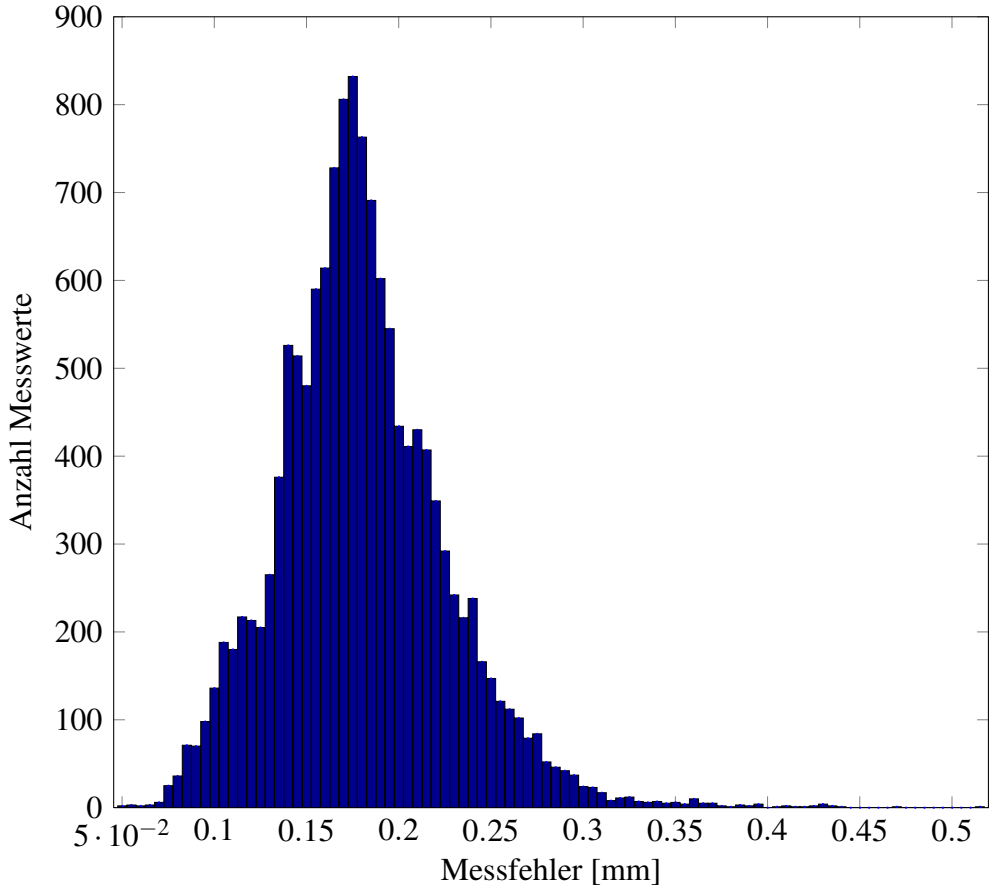

Abbildung 4.1: Verteilung des Trackingfehlers - Messung mitten im Messvolumen mit 13972 Messwerten

Diese Messungen wurden nur in der Mitte des Messvolumens gemacht und nicht am Rand des Messvolumens, wodurch der Mittelwert $\mu = 0.1802$ mm der Messung kleiner ist, als die von NDI angegebenen 0.25 mm. Die Varianz dieses Datensatzes beträgt $\sigma = 0.002043$ mm^2. Wenn das Messinstrument zum Rand des Messvolumens bewegt wird, kann beobachtet werden, dass der gemessene Fehler steigt, sodass davon ausgegangen werden kann, dass die Angaben von NDI zum Messfehler plausibel sind.

4.1.3 Gesamtmessfehler des Trackers

Da in dieser Arbeit die Positionen des zu messenden Instrumentes im Polariskoordinatensystem angegeben werden und das Polariskoordinatensystem der Ursprung eines weiteren Messinstrumentes ist, sind zum einen die Position des zu messenden Instrumentes und zum anderen die Position des Koordinatensystems fehlerbehaftet.

Unter der Annahme, dass die Fehler statistisch unabhängig sind und dass der Fehler des zu messenden Instrumentes

$$X_1 \sim N_1(\mu_1, \sigma_1^2) \tag{4.1}$$

entspricht und der Fehler des Koordinatensysteminstruments

$$X_2 \sim N_2(\mu_2, \sigma_2^2) \tag{4.2}$$

entspricht, so ist der gesamte Messfehler laut [17]:

$$X_1 + X_2 = X_{\text{gesamt}} \sim N_{\text{gesamt}}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \tag{4.3}$$

Da $\mu_1 = \mu_2 = 0.25$ mm gilt, ergibt sich für den Gesamtfehler $\mu_{\text{gesamt}} = \mu_1 + \mu_2 = 2 \cdot 0.25 \text{ mm} = 0.5 \text{ mm}$.

Das bedeutet, dass die Positionsbestimmung einen mittleren Fehler von 0.5 mm hat, vorausgesetzt das Polariskoordinatensystem wird als Ursprung benutzt.

Alternativ zum Benutzen eines Referenzinstrumentes als Koordinatensystem bietet das NDI System die Benutzung des Positionssensors als Ursprung an. Dies hat den Vorteil, dass der Messfehler somit nicht von einem zweiten Messinstrument abhängt und von einem mittleren Fehler von 0.25 mm ausgegangen werden kann. Ein Nachteil dabei ist, dass der Positionssensor fixiert sein sollte, um die Reproduzierbarkeit von Messergebnissen zu gewährleisten. Auch müsste bei jeder Neupositionierung des Positionssensors eine neue Transformation der Polariskoordinaten zu den Weltkoordinaten bestimmt werden, was ein nicht unerheblicher Aufwand wäre (siehe 3.3).

Das fertige Programm bietet sowohl die Option der Verwendung eines Referenzinstrumentes, als auch die Verwendung des Positionssensors als Koordinatenursprung.

4.2 Fehler der Transformation von Polariskoordinaten zu Weltkoordinaten

Wie in 3.3.4 schon erwähnt hat die gemessene x-Achse eine Abweichung zu der berechneten x-Achse von $0.3891°$. Die y-Achse wurde direkt aus der Projektion des Vektors mit der größten Varianz der Hauptkomponentenanalyse der Messwerte der Achse übernommen; die z-Achse ist das Kreuzprodukt der Vektoren mit den größten Varianzen der Hauptkomponentenanalyse zu den entsprechenden Messwerten der Ebene.

Der zu messende Punkt in Weltkoordinaten lautet $\vec{p}_{\text{Welt}_{\text{real}}} = \begin{pmatrix} 212.5 & 825 & -50 \end{pmatrix}^T$. Dieser Punkt ist am Lochraster der Arbeitsebene ausgerichtet und daher sind die Koordinaten bekannt. Gemessen hat der Punkt den Wert $\vec{p}_{\text{Welt}_{\text{gemessen}}}$[1] $= \begin{pmatrix} 223.60 & 824.76 & -51.00 \end{pmatrix}^T$.

[1] Mittelwert aus mehr als 100 Messwerten

Wie zu sehen ist, ist die Abweichung in y-Richtung bei ca. 0.24 mm und z-Richtung bei ca. 1 mm. In x-Richtung ist die Abweichung allerdings über 11 mm. Das liegt an der oben genannte Abweichung zwischen der berechneten Achse und der theoretischen Achse aus den Messwerten.

Dieser Fehler ist zurückzuführen auf die relativ ungenaue Ausrichtung der Messwerte auf den zu messenden Achsen. Das in 3.3.2 erwähnte Metallgestell zum Messen war handgefertigt und damit für exakte Messungen nur bedingt einsetzbar.

4.3 Zeitlicher Fehler

Bei der Synchronisation zeigt der zeitliche Fehler zwischen den Daten der 2-Tupel $(\text{3D-Datum}, \text{Bild})$ die Güte der gefundenen Lösung. In diesem Unterkapitel wird auf die Quantisierung des zeitlichen Fehlers, sowie mögliche Optimierungsverfahren eingegangen.

4.3.1 Theoretischer zeitlicher Fehler

Da das Endoskop eine Aufnahmerate von 30 Bildern/Sekunde und das Trackingsystem eine Aufnahmerate von 20 Daten/Sekunde haben soll, könnte die Verteilung der zeitlichen Fehler wie folgt aussehen:

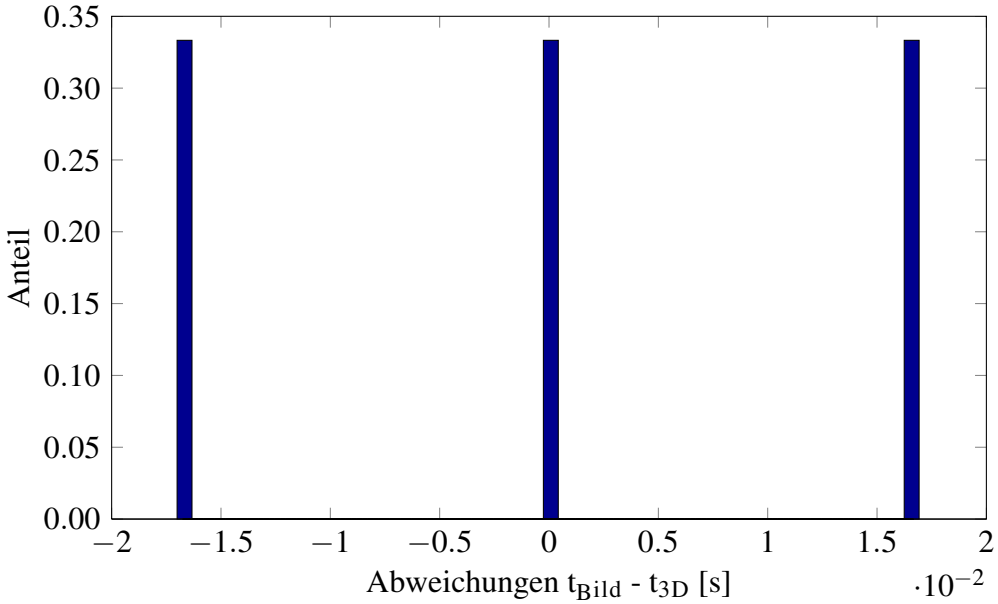

Abbildung 4.2: Verteilung des zeitlichen Fehlers in der Theorie mit gleichen Startzeitpunkten

Mit der Annahme, dass sowohl das Trackingsystem, als auch die Aufnahme der Endoskopbilder zum gleichem Zeitpunkt t_0 startet, so werden alle $\frac{3}{60}$ Sekunden eine Aufnahme vom Trackingsystem und alle $\frac{2}{60}$ Sekunden eine Aufnahme vom Endoskopbild gemacht. Folgende Daten kämen dabei zu Stande: $\mathcal{T}_{\text{3D-Datum}} = \{\frac{0}{60}, \frac{3}{60}, \frac{6}{60}, \frac{9}{60}, \ldots\}$, $\mathcal{T}_{\text{Endoskop}} = \{\frac{0}{60}, \frac{2}{60}, \frac{4}{60}, \frac{6}{60}, \frac{8}{60}, \ldots\}$. Somit beträgt der zeitliche Fehler zwischen einer Aufnahme eines 3D-Datums und der Aufnahme eines Endoskopbildes $-\frac{1}{60}$ Sekunden, 0 Sekunden oder $\frac{1}{60}$ Sekunden, wobei jeder Fall mit einer Wahrscheinlichkeit von $1/3$ Auftritt.

Aufgefallen ist an dieser Stelle auch, dass eine doppelte Zuweisung auftritt. Von drei Bildern werden zwei dem selben 3D-Datum zugewiesen. Das Phänomen der doppelten Zuweisung tritt ebenfalls häufig in der Praxis auf.

Wird nun von unterschiedlichen Startzeitpunkten und von einer schwankenden Frequenz beim NDI-Tracker ausgegangen (siehe 3.2.3), so kann die folgende Fehlerverteilung auftreten (ausgehend von einer Normalverteilung der Trackingfrequenz mit $\mu =$ 20 Hz und Standardabweichung von 0.5 Hz):

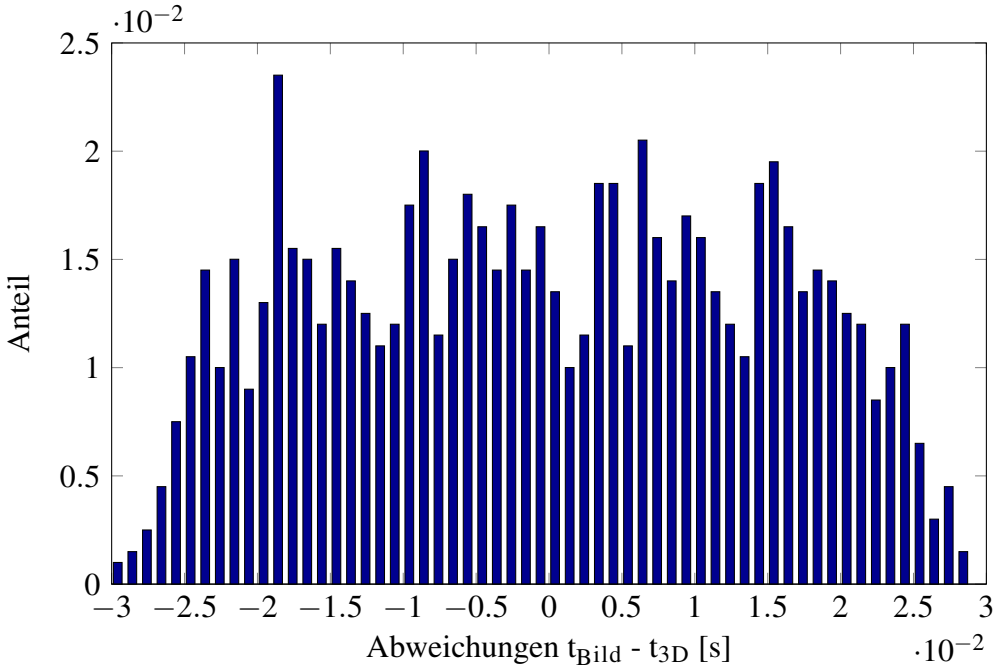

Abbildung 4.3: Verteilung des zeitlichen Fehlers in der Theorie mit verschiedenen Startzeitpunkten und schwankender Aufnahmerate des Trackers

Der größte theoretische zeitliche Abstand zwischen zwei Daten beträgt 25 ms. Dieser Fall tritt auf, wenn die Aufnahme eines Bildes zum Zeitpunkt t_{Bild} genau zwischen den Aufnahmen zweier 3D-Daten $t_{\text{3D-Datum}_1}$ und $t_{\text{3D-Datum}_2}$ geschieht. Da 3D-Datum$_1$

zum Zeitpunkt $t_{\text{3D-Datum}_1} = t_0$ und 3D-Datum$_2$ zum Zeitpunkt $t_{\text{3D-Datum}_1} = t_0 + 50$ ms aufgenommen wurde, ist der maximale zeitliche Abstand von dem Aufnahmezeitpunkt des Bildes zu den Aufnahmezeitpunkten der beiden 3D-Daten 25 ms.

4.3.2 Messungen des zeitlichen Fehlers

Abbildung 4.4 zeigt die Verteilung des zeitlichen Fehlers in der Praxis.

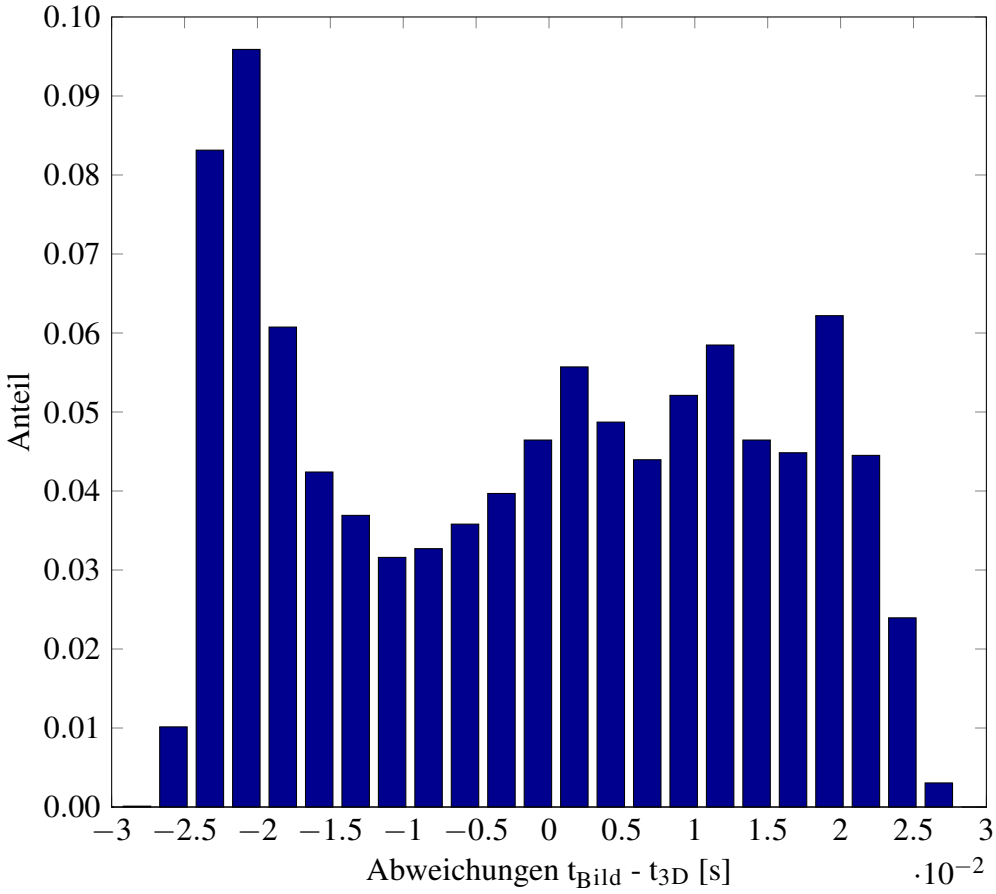

Abbildung 4.4: Verteilung des zeitlichen Fehlers in der Praxis mit 18041 Messwerten

Wird diese Abbildung mit Abbildung 4.3 verglichen, ist eine ähnliche Fehlerverteilung wiederzufinden.

Die Verteilung aus Abbildung 4.4 hat einen mittleren zeitlichen Fehler von $\mu_t = -0.001434$ s und eine Varianz von $\sigma_t^2 = 0.000233$ s^2.

Der betragsmäßige größte Abstand ist kleiner als 0.046 s. In 4.3.1 wurde der betragsmäßig größte zeitliche Abstand zwischen zwei Aufnahmen mit 25 ms angegeben.

Dieser beträgt in der Praxis aber fast das doppelte. Das kommt daher, dass das NDI-Trackingsystem die 3D-Daten zwar mit 20 Hz aufnehmen kann, jedoch durch verzögerte Anfragen die 20 Hz nicht tatsächlich erreicht werden können. In 3.2.3 wurde darauf hingewiesen, dass der Tracker bei mehrmaliger Anfrage in unter 50 ms sein aktuellstes Datum zurück gibt. Auch kommt es durch die Verarbeitung und die Übertragung der Daten zu Verzögerungen, die eine zusätzliche Verminderung der Anzahl der 3D-Informationen zur Folge haben.

4.3.3 Zeitlicher Offset bei der Bildakquisition

Die Verschlusszeit des Endoskops beträgt bei den relevanten Lichtverhältnissen ca. 40 ms. Dieser Wert wurde mit einem Verschlusszeitmessgerät ermittelt. Dieses Messgerät schaltet nacheinander LEDs, die in einem 10x10-Cluster angeordnet sind, mit einer einstellbaren Frequenz ein. Die Einzelbilder können analysiert werden und durch die Anzahl der leuchtenden LEDs kann die Verschlusszeit ermittelt werden. Hier leuchteten in den Einzelbildern stets 40 bis 41 LEDs bei einer Frequenz von 100 Hz.

Dies hat zur Folge, dass die Annahme des Zeitpunktes der Aufnahme falsch ist, wenn davon ausgegangen werden kann, dass die Aufnahme zu dem Zeitpunkt stattfindet, wenn die Endoskopdaten beim Computer ankommen. Also ist hier ein zeitlicher Offset für den Zeitstempel notwendig.

Der zeitliche Offset ist von der Hardware abhängig. Deswegen ist er in den Optionen einstellbar.

4.3.4 Optimierung des zeitlichen Fehlers

Eine Möglichkeit den zeitlichen Fehler zu optimieren ist es, das Intervall, in dem nach dem nächsten Datum gesucht wird, zu verkleinern. Dies schafft zwar, dass nur Daten mit einem gewissen, geringeren Abstand zueinander zugeordnet werden, geht allerdings mit dem Verwerfen von Daten einher.

Intervall [ms]	**Messwerte**	**Verworfene Tupel** ([%])	$\mu_{\textbf{Fehler}}$ [s]	$\sigma^2_{\textbf{Fehler}}$ [$10^{-3}s^2$]
100	18041	1 (0.0055)	-0.0014	0.233
50	19463	236 (1.2126)	-0.011	0.224
25	19238	10508 (54.6211)	0.0008	0.049
12.5	17526	13442 (76.6975)	0.0003	0.012

Tabelle 4.1: Optimierung des zeitlichen Fehlers durch das Verwerfen von Daten

Wie in Tabelle 4.1 zu sehen ist, kann bei einem maximalen zeitlichen Abstand zwischen der Bildakquisition und der Aufnahme der 3D-Daten von 12.5 ms von mehr als 50% verworfenen Tupeln ausgegangen werden.

Für manche Anwendungszwecke, beispielsweise der Kalibrierung einer Kamera, ist eine so hohe Anzahl von verworfenen Daten akzeptabel, um eine höhere Genauigkeit zu erhalten.

4.3.5 Relevanz des zeitlichen Fehlers

Wird von einem Intervall von 50 ms ausgegangen, so liegt der zeitliche Abstand zwischen der Bildakquisition und der Aufnahme der 3D-Daten bei maximal $t_{\text{Abstand}_{\text{max}}} = 25$ ms. Wird zusätzlich bei der Aufnahme der 3D-Daten von einem Fehler von 0.25 mm im Mittel ausgegangen, dann wird der Fehler durch den zeitlichen Abstand bei einer Bewegung mit einer Geschwindigkeit von

$$v = \frac{0.25 \text{ mm}}{25 \text{ ms}} = 10 \text{ cm/s} \tag{4.4}$$

größer als der mittlere Fehler des optischen Messsystems.

Da bei der Verwendung eines Referenzinstrumentes von einem mittleren Fehler von 0.5 mm ausgegangen werden muss, ist erst bei

$$v = \frac{0.5 \text{ mm}}{25 \text{ ms}} = 20 \text{ cm/s} \tag{4.5}$$

der Fehler des optischen Messsystems kleiner, als der durch den zeitlichen Fehler bei der Zuordnung resultierende örtliche Fehler.

Durch die in 4.3.4 beschriebene Methode der Minimierung des zeitlichen Fehlers ist es möglich, die Geschwindigkeit, ab der der zeitliche Fehler einen größeren Einfluss als der Trackingfehler hat, zu steigern. Dies geht allerdings wieder mit dem Verwerfen von Daten einher.

Um den örtlichen Fehler zusätzlich noch quantifizieren zu können wurden die synchronisierten Daten mit linear interpolierten Daten der gleichen Aufnahme verglichen. Dafür wurden drei Messungen gemacht.

Messung	**Messwerte**	$\mu_{\textbf{Abweichung}}$ [mm]	$\sigma^2_{\textbf{Abweichung}}$ [mm^2]
1	325	0.0504	0.0020
2	310	0.0594	0.0021
3	1157	0.0567	0.0020

Tabelle 4.2: Vergleich der Synchronisation mit linearer Interpolation

Diese drei Messungen wurden mit dem OWIS Positionierungsmotoren gemacht. Dafür wurde das Endoskop beim ersten Test entlang der x-Achse des OWIS-Systems bewegt, beim zweiten Test entlang der y-Achse und beim dritten Test wurde ein Rechteck in der xy-Ebene abgefahren.

Die Motoren des OWIS-Systems haben eine Geschwindigkeit von unter 1 cm/s, was die geringen Abweichungen erklärt. Bei schnelleren Bewegungen wächst die Abweichung zwischen synchronisierten und interpolierten Daten.

4.4 Projektion von Gitterpunkten

Um sich sowohl den in 4.1 quantifizierten Trackingfehler, als auch den in in 4.3 gezeigten zeitlichen Fehler visualisieren zu können, wurden von einem Schachbrettmuster die Eckpunkte der Kacheln aufgenommen und in das Schachbrettmuster zurück projiziert.

Was bei dieser Art der Visualisierung auffällt, ist, dass der zeitliche Messfehler bei schnellen Bewegungen deutlicher sichtbar wird. Auch ist interessant, dass der Unterschied zwischen dem Messrauschen des NDI Systems und dem Synchronisationsfehler sichtbar wird.

Findet keine Bewegung statt, so ist zu sehen, dass die dargestellten Punkte sich um einen Mittelwert herum bewegen. Sobald schnellere Bewegungen stattfinden kann beobachtet werden, dass die Punkte den eigentlichen Ort, an dem sie sein sollten, in der Richtung der Bewegung hinterher laufen. Dieses Fehlerbild entsteht dann durch die Synchronisation.

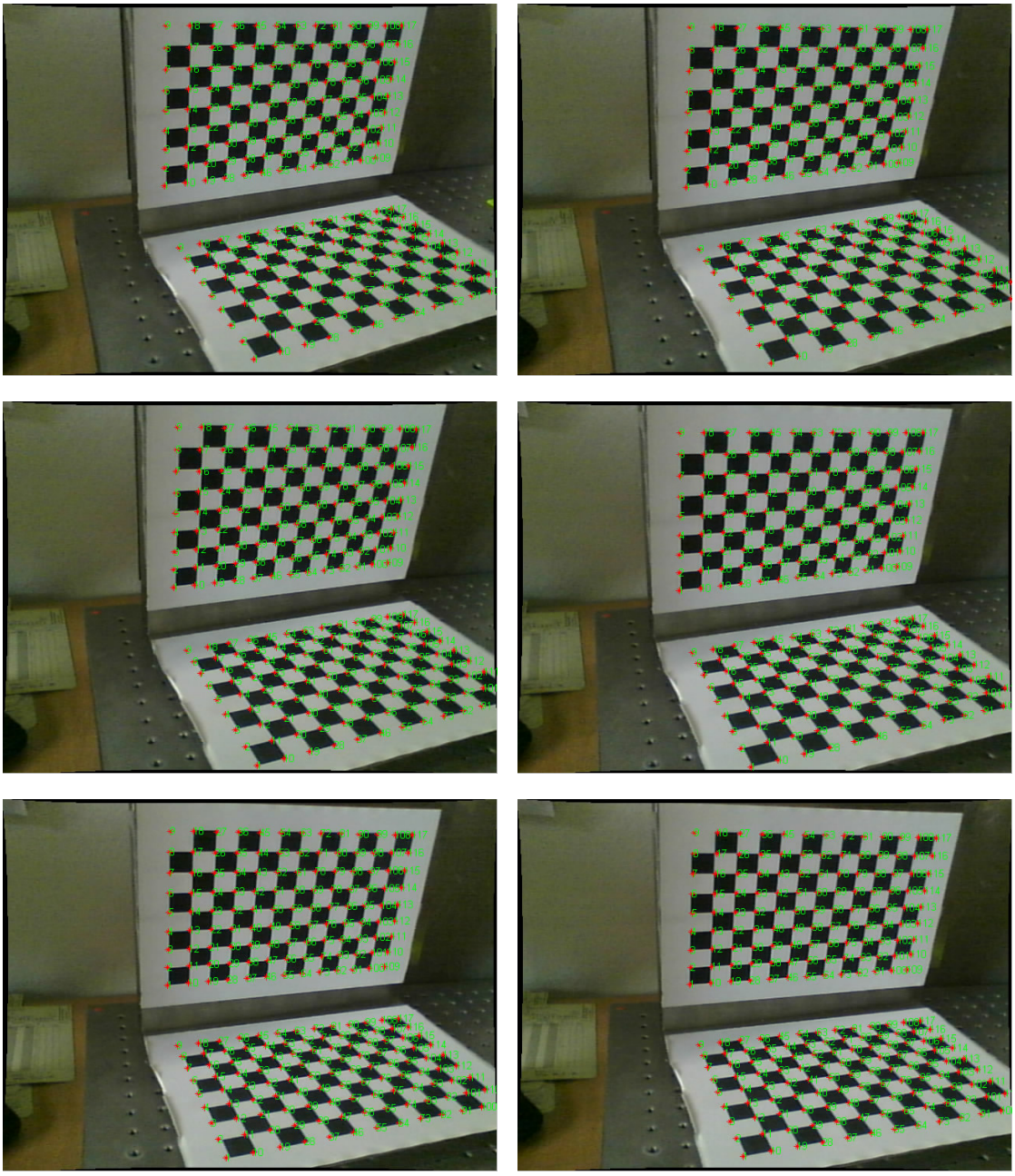

Abbildung 4.5: Videoausschnitt zur Fehlerdarstellung

Kapitel 5

Fazit und Ausblick

Ziel dieser Bachelor Arbeit war es, das von NDI gelieferte Beispielprogramm so zu erweitern, dass die Tracking-Daten möglichst synchron zu den Videodaten eines Endoskops ausgegeben werden können.

In Kapitel 4 wird dargestellt, dass die Aufgabe erfolgreich gelöst werden konnte. Der zeitliche Fehler geht im Mittel gegen 0 s und macht erst bei schnelleren Bewegungen Probleme. Des Weiteren wurde das Programm über die Synchronisation hinaus erweitert. Es bietet jetzt die Möglichkeit, Punkte abzutasten und diese zu Speichern. Auch kann das Programm die Koordinaten direkt in Weltkoordinaten angeben, wenn die entsprechende Transformationsmatrix bekannt ist. Diese Funktionen waren anfangs nicht geplant und sind erst im laufe der Arbeit als wünschenswerte Funktionalität hinzugekommen.

Allerdings ist bei der Berechnung der Transformation für die Weltkoordinaten noch Optimierungspotenzial. Der aktuelle Fehler ist noch zu groß für einen tatsächlichen Einsatz und sollte durch genauere Messwerkzeuge verkleinert werden können. Ein typisches Problem bei optischen Trackingsystemen ist, dass sobald der Sichtkontakt zwischen dem Positionssensor und den Markern unterbrochen ist, keine Daten erfasst werden. Eine Idee, die verfolgt werden könnte ist, dass die 3D-Daten an dieser Stelle bis zu einem gewissen Grad interpoliert werden könnten. Für kurze Unterbrechungen wäre dies eine Möglichkeit weniger Daten verwerfen zu müssen.

Interpolation wäre auch ein Ansatz, den zeitlichen Fehler zu minimieren. Da die Rohdaten ausgegeben werden können, ist auch eine spätere Interpolation über die Messdaten möglich. Somit würde in der Auswertung eine höhere Genauigkeit erreicht werden. Welche Art der Interpolation genutzt werden könnte, müsste dann noch untersucht werden.

Um zu zeigen, wie die Interpolation aussehen könnte, wurde eine Funktion für Matlab geschrieben, welche die Rohdaten einliest und diese dann linear Interpoliert.

Anhang A

Versuchsaufbau

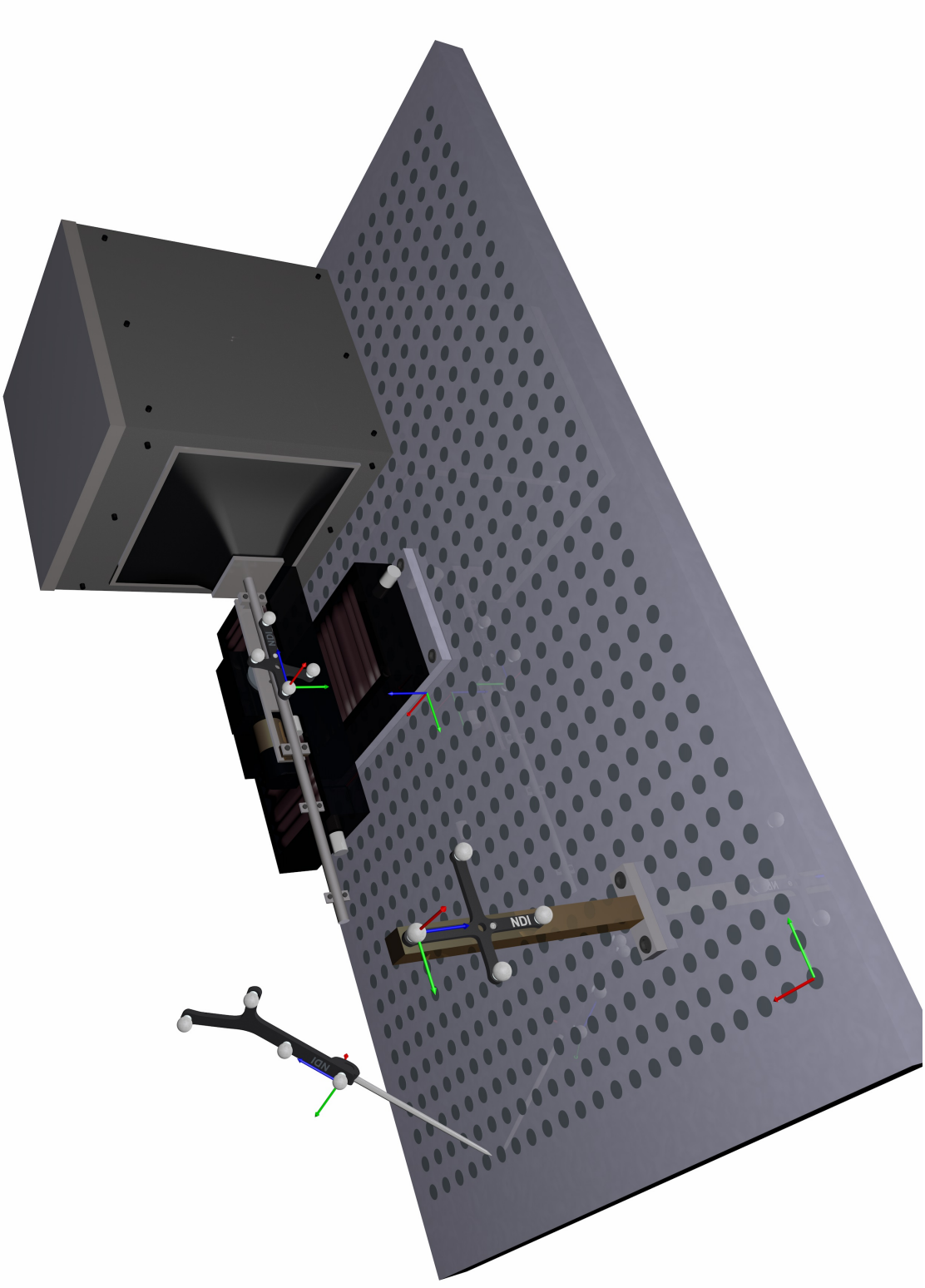

Abbildung A.1: Darstellung des Versuchaufbaus

Anhang B

Klassendiagramme

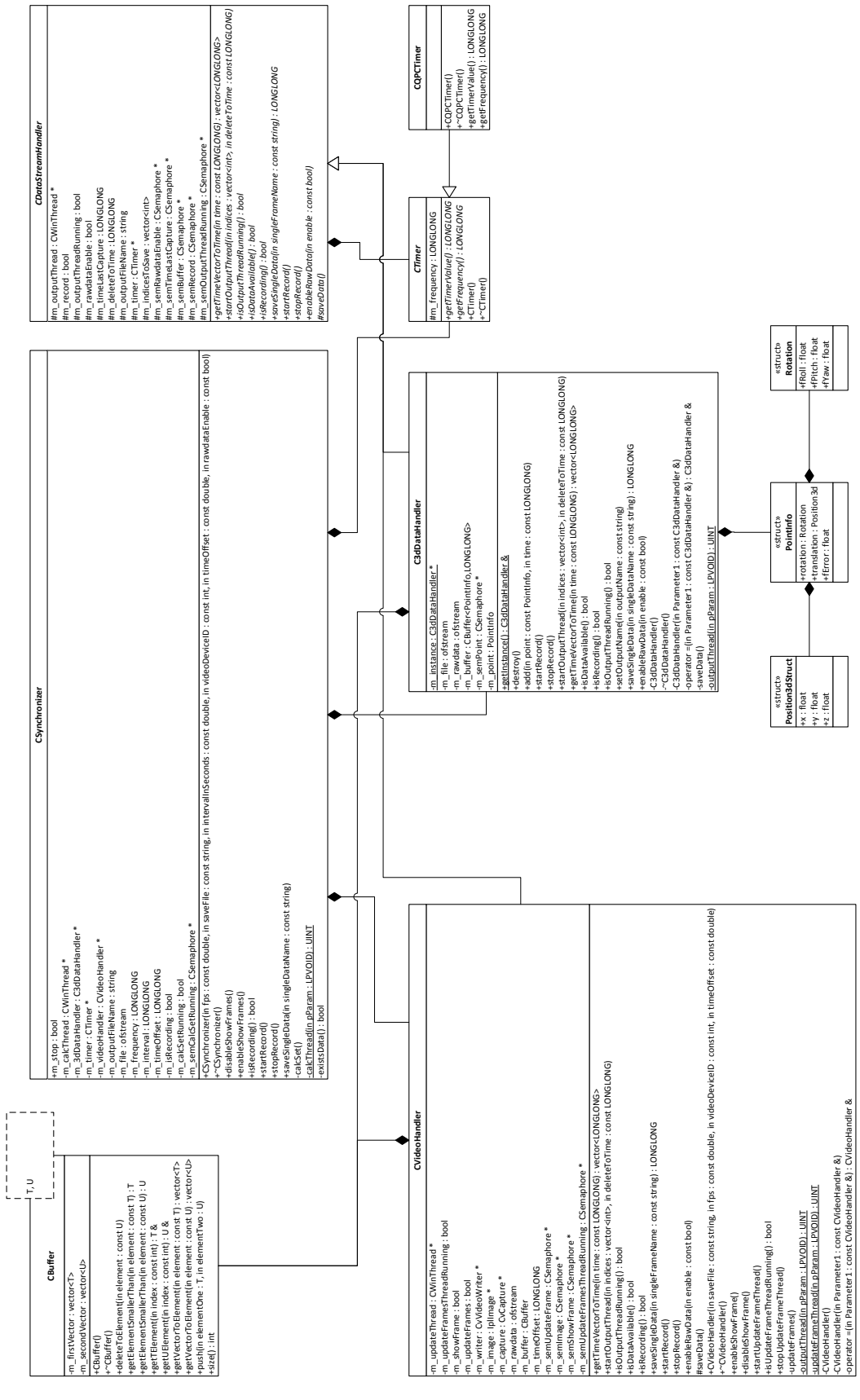

Abbildung B.1: Klassendiagramm zur Synchronisation

Abbildung B.2: Klassendiagramm mit Trackeransteuerung und grafischer Benutzeroberfläche

Anhang C

Programmierung

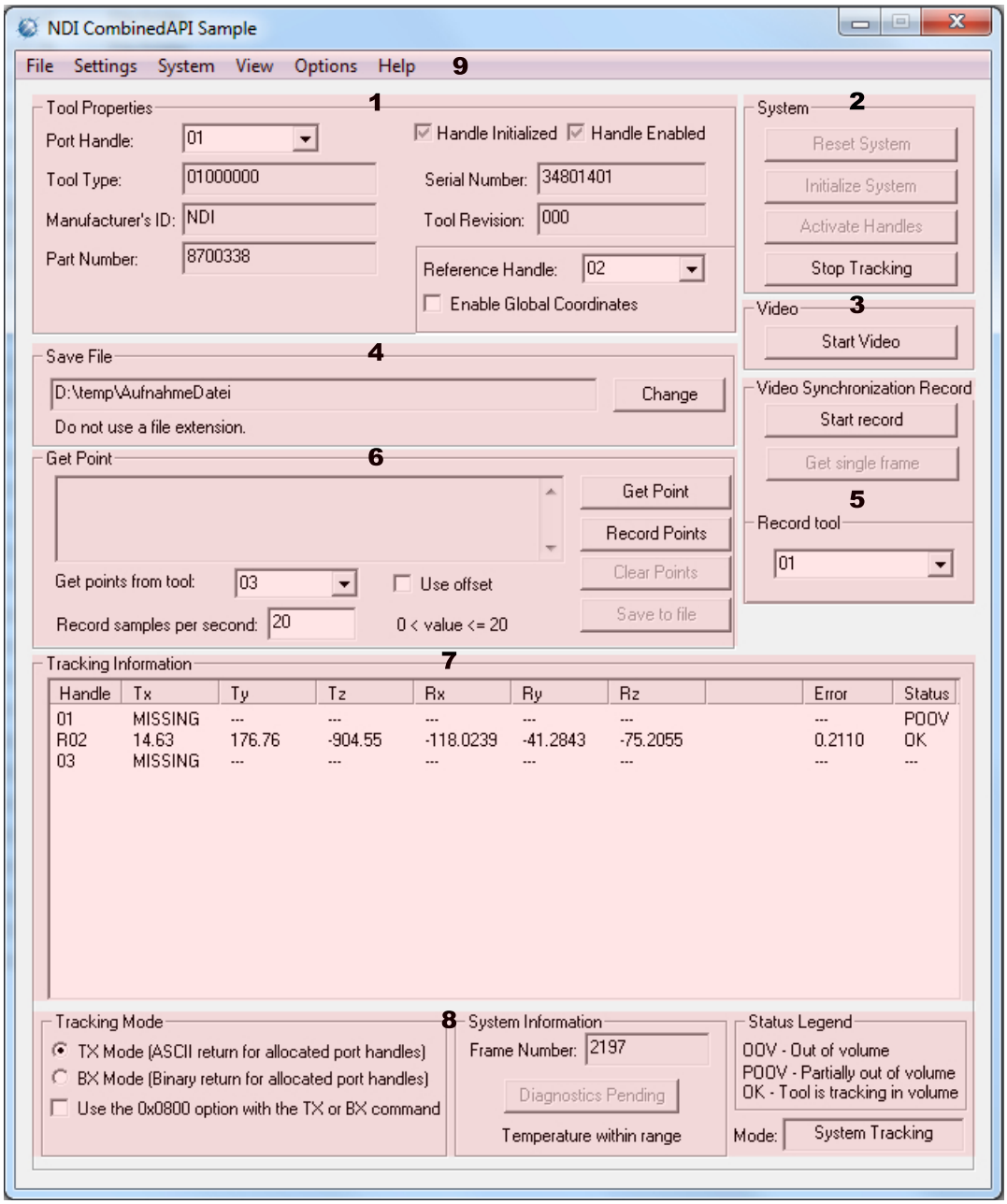

Abbildung C.1: Grafische Benutzeroberfläche

```
1  CQPCTimer::CQPCTimer() throw(XQPCTimer)
2  {
3          m_frequency = 0;
4
5          //Der aktuelle Thread wird auf PROZESSOR_COUNTER_ID gesetzt.
6          //In oldMask befindet sich die Maske für den
7          //ursprünglichen Prozessor.
8          DWORD_PTR oldMask = ::SetThreadAffinityMask(::GetCurrentThread(),
9                                                      PROZESSOR_COUNTER_ID);
10
11         //Fehlerbehandlung, für den Fall eines Threading Fehlers.
12         if(oldMask == 0)
13                 throw(XQPCTimer("Threading problem detected."));
14
15         //Die Frequenz wird in m_frequency gespeichert.
16         if (!QueryPerformanceFrequency((LARGE_INTEGER*)&m_frequency))
17         {
18                 //Fehlerbehandlung falls kein QueryPerformanceCounter existiert.
19                 ::SetThreadAffinityMask(::GetCurrentThread(), oldMask);
20                 throw(XQPCTimer("No QueryPerformanceCounter detected."));
21         }
22
23         //Zurücksetzen auf ursprünglichen Prozessor.
24         ::SetThreadAffinityMask(::GetCurrentThread(), oldMask);
25 }
```

Abbildung C.2: Lösung des Problems des QueryPerformanceCounters mit Multiprozessorsystemen anhand eines Beispiels

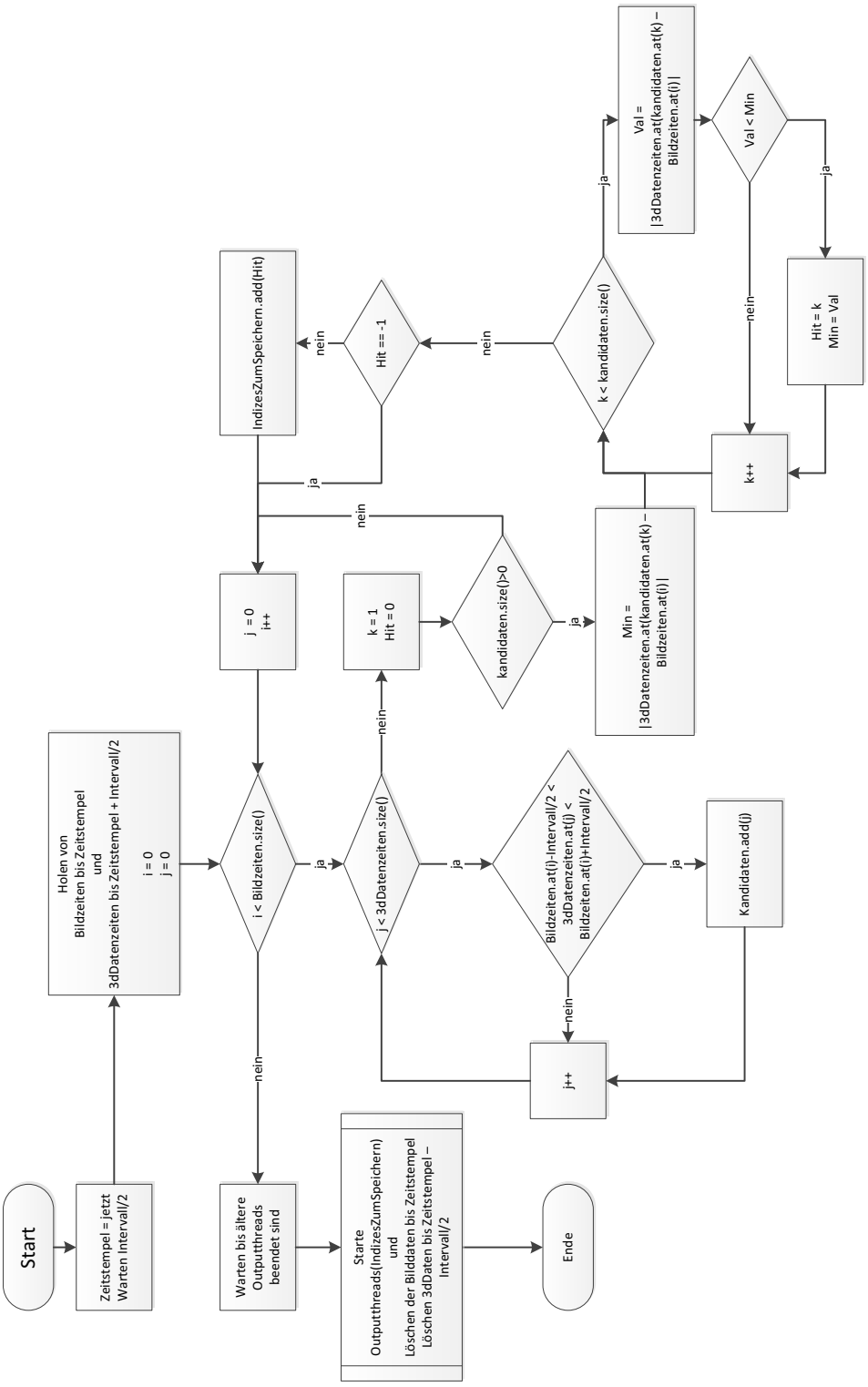

Abbildung C.3: Flussdiagramm zur Synchronisation

```
1  void CVideoHandler::updateFrames() throw(XQPCTimer)
2  {
3          //...
4
5          while(updateFrames)
6          {
7                  //...
8
9                  //timestamp before
10                 cvGrabFrame(m_capture);
11                 //timestamp after
12
13                 //timestamp = timestamp before +
14                 //              (timestamp after - timestamp before)/2
15
16                 //...
17
18                 m_image = cvRetrieveFrame(m_capture);
19
20                 //...
21
22         }
23 }
```

Abbildung C.4: Quelltextausschnitt mit Verwendung von `cvGrabFrame(...)` und `cvRetrieveFrame(...)`

```
1  void CVideoHandler::updateFrames() throw(XQPCTimer)
2  {
3          //...
4
5          while(updateFrames)
6          {
7                  //...
8
9                  //timestamp before
10                 m_image = cvQueryFrame(m_capture)
11                 //timestamp after
12
13                 //timestamp = timestamp before +
14                 //              (timestamp after - timestamp before)/2
15
16                 //...
17
18         }
19 }
```

Abbildung C.5: Quelltextausschnitt mit Verwendung von `cvQueryFrame(...)`

Literaturverzeichnis

[1] Lichttonverfahren -. http://de.wikipedia.org/wiki/Lichttonverfahren. - Abgerufen am 12.11.2012.

[2] Tobias Helbig. *Kommunikation und Synchronisation multimedialer Datenströme in verteilten Systemen* -. Kovač, Hamburg, 1996.

[3] NDI. *Polaris Vicra User Guide* -, 2011.

[4] NDI. *Polaris Application Program Interface Guide* -, 2011.

[5] Andrés F. Mármol Vélez, Jan Marek Marcinczak, and Rolf-Rainer Grigat. *Structure from Motion Based Approaches to 3D Reconstruction in Minimal Invasive Laparoscopy*. Vision Systems, Hamburg University of Technology, 2012.

[6] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision* -. Cambridge University Press, Cambridge, 2004.

[7] Markus Lust. Quaternionen - mathematischer Hintergrund und ihre Interpretation als Rotationen - . Seminar Skript, http://www.uni-koblenz.de/~cg/veranst/ws0001/sem/Lust_quaternion.pdf, 2001. - Abgerufen am 12.11.2012.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning* -. Springer, Berlin, Heidelberg, 2006.

[9] Gilbert Strang. *Lineare Algebra* - . Springer-Verlag GmbH, Heidelberg, 2003.

[10] Alexander Stoffel. Programmierung Numerischer Verfahren - . Vorlesungsskript, http://alex.nt.fh-koeln.de/mapdf/Prognum.pdf, 2011. - Abgerufen am 12.11.2012.

[11] Adronic Components GmbH. *Videoendoskope und Zubehör* -.

[12] Opencv -. http://opencv.org. - Abgerufen am 12.11.2012.

[13] Gary Bradski, Adrian Kaehler, Mike Loukides, and Robert Romano. *Learning OpenCV - Computer Vision with the OpenCV Library* - . O'Reilly Media, Inc., Sebastopol, CA, 2008.

[14] Queryperformancecounter function -. http://msdn.microsoft.com/en-us/library/ms644904, 2012. - Abgerufen am 12.11.2012.

[15] Verwenden von QueryPerformanceCounter zum Messen der Ausführungszeit von Code - . http://support.microsoft.com/kb/172338/de, 2012. - Abgerufen am 12.11.2012.

[16] Multithreading mit C++ und MFC - . http://msdn.microsoft.com/de-de/library/975t8ks0%28v=vs.80%29.aspx. - Abgerufen am 13.11.2012.

[17] Ursa Pantle. Summe, Produkt und Quotient von unabhängigen Zufallsvariablen - . www.mathematik.uni-ulm.de/stochastik/lehre/ws03_04/wr/skript/node38.html, 2004. - Abgerufen am 13.11.2012.