Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Christian Reimann

Entwicklung eines Steuerungsmoduls für einen 3D-Laserscanner

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Christian Reimann

Entwicklung eines Steuerungsmoduls für einen 3D-Laserscanner

Diplomarbeit eingereicht im Rahmen der Diplomprüfung

im Studiengang Softwaretechnik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. rer. nat. Reinhard Baran

Abgegeben am 14. August 2008

Christian Reimann

Thema der Diplomarbeit
Entwicklung eines Steuerungsmoduls für einen 3D-Laserscanner

Stichworte
Laser-Scanner, Lichtschnittverfahren, Laser-Triangulation, 3D Vermessung, Transformation

Kurzzusammenfassung
Die vorliegende Diplomarbeit befasst sich mit dem technischen Aufbau und der Realisierung eines 3D-Laserscanners auf Basis der Lichtschnitttechnik. Es wird die Konfiguration und das Zusammenspiel der Hardwarekomponenten erläutert sowie deren Aufbau. Hierzu wird eine Steuerungssoftware implementiert, die den Ablauf eines Scanvorgangs und die Bildakquisition beschreibt. Nach Abschluss eines Scandurchlauf können die erfassten Bilder anhand perspektivischer Transformation 2-dimensional vermessen und die Rohdaten für die Weiterverarbeitung in anderen Programmen exportiert werden. Zur Verfügung stehen dazu eine CCD-Kamera, ein Laser und ein Drehteller angetrieben über einen Schrittmotor und einer Positioniersteuerung.

Christian Reimann

Title of the paper
Development of a control unit for 3D-Scanning

Keywords
Laser-Scanner, Sheet-of-light, Laser-Triangulation, 3D measurement, transformation

Abstract
This Diplom-thesis is concerned with the technical structure and realization of a 3D-Laserscanners on basis of the light cut technology. The configuration and the interaction of the hardware components are described. A control software is implemented which describes the expiration of a scan procedure and the picture acquisition. After a scan procedure the seized pictures can be measured on the basis of perspective transformation in two dimensions and the raw data for the subsequent treatment in other programs can be exported. A CCD-camera, a laser and a turntable driven by a stepper-motor with positioning control will be used.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Digitalisierung von 3D-Objekten und deren Bildverarbeitung ist heutzutage zur Gewinnung von Bildinformationen nicht mehr wegzudenken. In vielen Forschungs- und Anwendungsbereichen werden heute Verfahren benötigt, um geometrische, dreidimensionale Daten eines Objektes mit Hilfe von geeigneten Messsystemen zu erfassen. In der industriellen Fertigung z.B. in der Qualitätssicherung dienen solche Verfahren zur Ermittlung von Oberflächeninformationen. Dabei werden Abweichungen von den Sollmassen, die in der Produktion von Werkstücken entstehen können, erfasst und mit einer darauf folgenden Datenanalyse fehlerhafte Stücke erkannt und ausgesondert. Verfahren zur optischen Formerfassung findet man auch in der Archäologie zur 3D-Vermessung von Ausgrabungen und deren Archivierung oder zum 3D-Scannen historischer Gegenstände oder Kunstobjekte. Die Visualisierung von Daten spielt außerdem in Bereichen der Medizin, Multimedia-Anwendungen, der Architektur oder der Robotik/Fahrzeugsteuerung zur selbständigen Navigation eine bedeutende Rolle.

1.1. Stand der Technik

Um reale Objekte virtuell darzustellen, stehen eine Reihe von Verfahren zur Verfügung. Generell wird zwischen berührungsfreien und berührungsorientierten Messmethoden unterschieden.
Taktile Sensoren werden meist in der Industrie zur mechanischen Abtastung von Werkstückoberflächen angewendet, dazu kann man unterscheiden zwischen Fühlstiften, Kraft-Momentsensoren oder Drucksensoren. Zum Beispiel ist das CMM Verfahren („coordinate measuring machine“) [1] ein sehr exaktes Scanverfahren, das mit Hilfe einer beweglichen Messprüfspitze Punktkoordinaten von einem Messobjekt aufzeichnet (Abbildung 1.1).
Destruktive Sensoren zerstören sogar das zu messende Objekt, indem die Oberfläche während des Scannens abgetragen wird. Bei Messsystemen ohne mechanische Beeinflussung verwendet man unter anderem induktive, magnetische, pneumatische oder Ultraschallsensoren.

[1] Berührungsorientierte 3D-Scanner: `http://www.dma.ufg.ac.at/`

In der berührungslosen Vermessung von Oberflächen gewinnt die Scannertechnik zunehmend an Bedeutung. Optische Messverfahren bieten eine Vielzahl von Vorteilen im Vergleich zu taktilen Verfahren. So wird beim Scannen kein Kontakt mit dem Objekt benötigt, so dass auch sensible und leicht verformbare Werkstücke zerstörungsfrei gemessen werden können. Weiterhin zeichnen sich optische Verfahren durch eine hohe Messgeschwindigkeit bei hoher Messgenauigkeit aus.

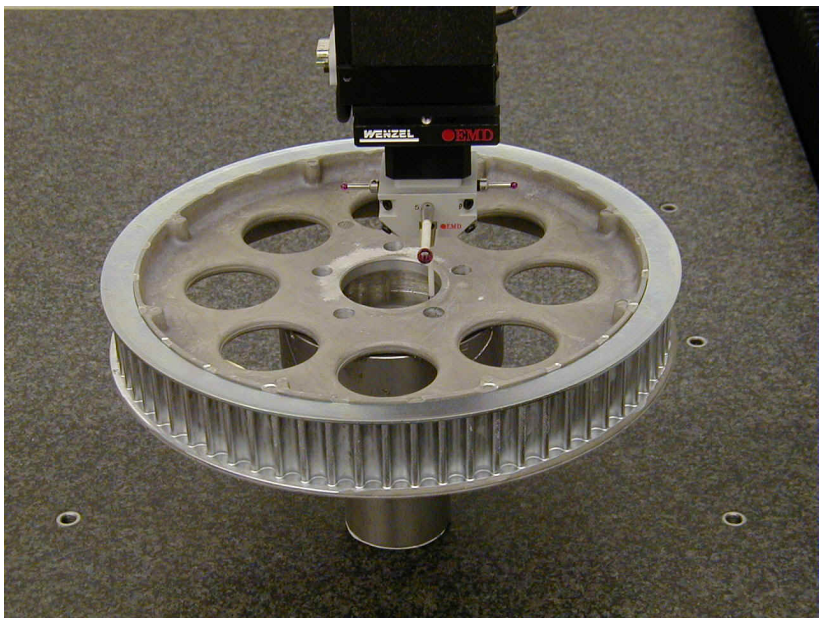


Abbildung 1.1.: Berührungsorientiertes CMM-System der Firma Wenzel

Bei der optischen berührungslosen Erfassung von 3D-Objekten unterscheidet man zwischen aktiven und passiven Methoden. Die 3D-Koordinaten von Gegenständen werden dabei durch Triangulationsverfahren ermittelt.
Bei aktiven Methoden wird eine Oberfläche meist mit gepulstem oder strukturiertem (Laser-) Licht bestrahlt und von lichtempfindlichen Detektoren aufgenommen. Durch die Kenntnis der Sender- und Emfängerposition zueinander können durch entstehende geometrische Verzerrungen oder zeitliche Differenzen von Lichtimpulsen die Entfernungen bestimmt werden.
Passive optische Messverfahren strahlen keine Energie auf die zu vermessenden Objekte. Mit binokularen Messtechniken werden die Tiefeninformation durch Verwendung von zwei Kameras (Stereobildverarbeitung, Stereoskopie) gewonnen. Dabei wird das Objekt aus verschiedenen Blickwinkeln betrachtet. Die Kameras sind dabei in einem bekannten Abstand zueinander positioniert. Eine 3D-Rekonstruktion findet statt, indem die geringfügigen Unterschiede innerhalb der Bilder analysiert werden und der Abstand der betroffenen Bildpunkte ermittelt wird.

In dieser Arbeit wird ein aktives optisches Messverfahren auf der Basis der Laser-Lichtschnitt-Technik angewandt. Zum Scannen werden eine Kamera, ein Laserimpulsgeber und ein Drehteller verwendet. Mit einer Kalibrierung ermittelt man die Sensorgeometrie, in diesem Fall die Position der Kamera zur Lichtquelle, und detektiert die Lichtlinie auf dem beleuchteten Objekt. Das gesamte Objektprofil wird durch das Vorbeibewegen des Objektes am Laser auf einem Drehteller abgescannt.

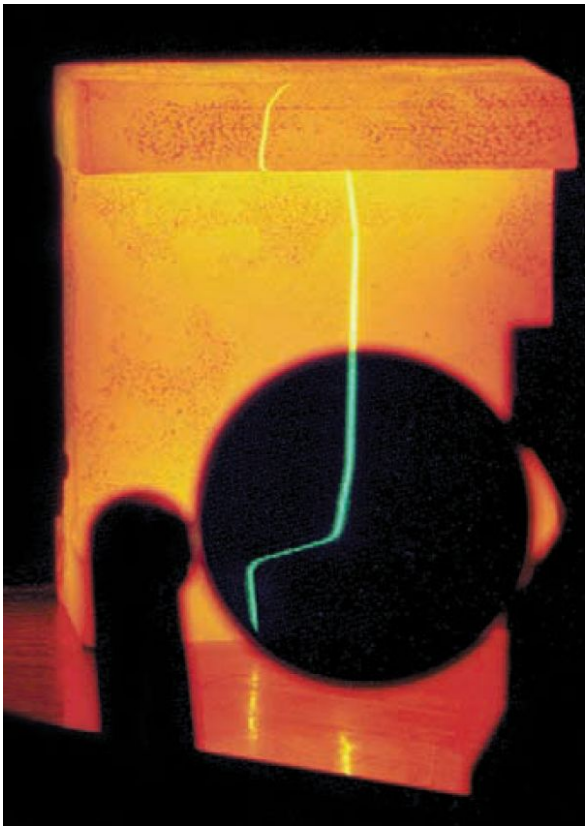


Abbildung 1.2.: Laser-Lichtschnitt
Ein Filter unterdrückt die störende Strahlung des Gusseisens und lässt nur die Laserlinie passieren (Foto: Fh-Institut für Lasertechnik (Fh-ILT), http://www.ilt.fraunhofer.de/).

1.2. Ziel dieser Arbeit

Im Rahmen dieser Diplomarbeit soll eine Anwenderbibliothek erstellt werden, welche die erforderlichen Parameter zum Verbindungsaufbau und Steuerung eines Drehtellers beinhaltet. Die Parameter werden speziell für den verwendeten Schrittmotor und einer Positioniersteuerung konfiguriert. Mit diesen Einstellungen dreht sich das Messobjekt um 360° und dabei werden in Einzelschritten Bilder aufgenommen. Durch ein anschliessendes Einlesen der gespeicherten Bilder findet eine Objektvermessung statt und die Messdaten werden in eine externe Datei exportiert.
Diese Arbeit ist eine Weiterentwicklung, aufbauend auf einer vorherigen Bachelorarbeit Schuhfuss (2007), in der ein Verfahren implementiert wurde zur Kalibrierung des Systems und zur Umrechnung von Bildkoordinaten eines Bildes in Realkoordinaten.

1.3. Gliederung

Kapitel 1 zeigt einen Überblick über 3D-Messtechniken in der heutigen Zeit und deren Anwendungsgebiete.

Kapitel 2 gibt einen genaueren Einblick zu den verschiedenen Ansätzen der optischen Abstandsmessung, der Laser-Triangulation und die mathematischen Zusammenhänge.

Kapitel 3 beschreibt den Technischen Aufbau des 3D-Messsystems.

Kapitel 4 erklärt die Schritte zur Vorbereitung und den Ablauf eines Scanndurchgangs.

Kapitel 5 erklärt die einzelnen Steuerungsbefehle und die Bedeutung der übergebenen Paramter. Zeigt anhand von Programmbeispielen das Ansteuern des Schrittmotors und die Bildakquisition.

Kapitel 6 Stellt die verwendeten Hardware-Komponenten vor und erklärt die wesentlichen Eigenschaften.

Kapitel 7 beinhaltet alle Softwarekomponenten, die zur Realisierung benötigt wurden sowie die Applikationen, die als Hilfestellung zum Einstellen der hardwarespezifischen Parameter dienten.

Kapitel 8 Zusammenfassung und Ausblick.

2. Überblick

In diesem Kapitel werden verschiedene 3D-Messtechniken vorgestellt sowie die Grundlagen von optischen, aktiven Mess-Verfahren. Darin wird ein Überblick der Prinzipien der optischen Abtastung geschaffen. Am Schluss des Kapitels wird auf die in der Arbeit eingesetzte Technik eingegangen.

Inhalt

2.1. Verfahren zur räumlichen Abtastung

Zur räumlichen, kontaktlosen Abtastung gibt es verschiedene Ansätze die Form eines Objektes zu erfassen, um daraus ein 3D-Objekt zu rekonstruieren. Dazu werden meist Abstandssensoren benutzt, die typischerweise Impulse per Laser, Ultraschall oder Infrarot aussenden. Dabei werden unterschiedliche physikalische Eigenschaften zur Tiefengewinnung genutzt.
Ein Infrarotsensor ist ein recht einfaches Prinzip, das oft in der Robotik zur Hinderniserkennung dient.
Ultraschallsensoren entsenden Schallwellen, die an festen Objekten reflektieren. Sie werden in der Regel zur Navigation oder zur Lokalisierung von Objekten benutzt.
Ein Lasersensor strahlt ein Licht aus, das an Objekten reflektiert. Diese Technik wird in verschiedene Möglichkeiten die Entfernung zum Ziel zu messen untergliedert:

- Phasenverschiebung: Messen der Phasendifferenz des ausgesandten Strahls und des reflektierten.
- Triangulation: Die Lichtprojektion auf ein Objekt wird von einer Kamera mit bekannter Geometrie zum Laser beobachtet und die entfernten Punkte anhand deren Dreiecksbeziehung mit Hilfe von Winkeln berechnet.
- Laufzeitmessung: Messen der Zeit (Time-of-flight), die der Lichtstrahl von der Aussendung bis zur Reflexion benötigt.

Um die komplette Oberfläche eines Objektes zu erfassen und somit ein ganzes Objekt mit dieser Technik zu vermessen, muss sich entweder der Sensor oder das Objekt durch einen Vorschub (Rotativ/Linear) bewegen.

2.2. Lasertriangulation

Bei der Lasertriangulation wird ein punktförmiger Laserstrahl auf ein Messobjekt ausgestrahlt und von einer Kamera, die in einem sogenannten Triangulationswinkel zum Lasersensor steht, registriert. Die Position von der Kamera zum Laser im Triangulationswinkel wird vorher durch eine Kalibrierung ermittelt. Ändert sich der Abstand der abgetasteten Oberfläche vom Laser, ändert sich auch der Winkel, unter dem der Lichtpunkt beobachtet wird. Damit ist die Position innerhalb der 2D-Koordinaten im Kamerabild verändert. Aus der Positionsänderung wird mit Hilfe der Winkelfunktion die Tiefe des Messobjektes bestimmt.

2.3. Laser-Lichtschnitt-Verfahren

In diesem Projekt wird die Technik eines 3D-Lasertriangulationssensors mit dem Lichtschnitt-Verfahren angewandt. Durch eine andere Struktur des Lichtes ist das Verfahren eine erweiterte Art der Lasertriangulation, wobei nicht mehr einzelne Punkte auf der Oberfläche betrachtet werden, sondern Linien, die das Objekt schneiden. Dieses Objekt reflektiert dann eine Profillinie.
Der Lasersensor projiziert einen gefächerten Linienstrahl. Aus dem bekannten Blickwinkel der Kamera zum Sensor, der durch die Kalibrierung errechnet ist, wird eine Verformung entlang der Laserlinie auf der Objektoberfläche vom Detektor im Abbild bestimmt. Anhand der trigonometrischen Zusammenhänge werden dann wieder die Abstände vom Sensor zum Objekt ermittelt. Dieses Verfahren beschleunigt durch Abtastung ganzer Profillinien die Abstandsmessung sehr.
Das Prinzip der 3D-Vermessung durch Lichtstreifenprojektion ist in der unten abgebildeten Zeichnung dargestellt [2]:

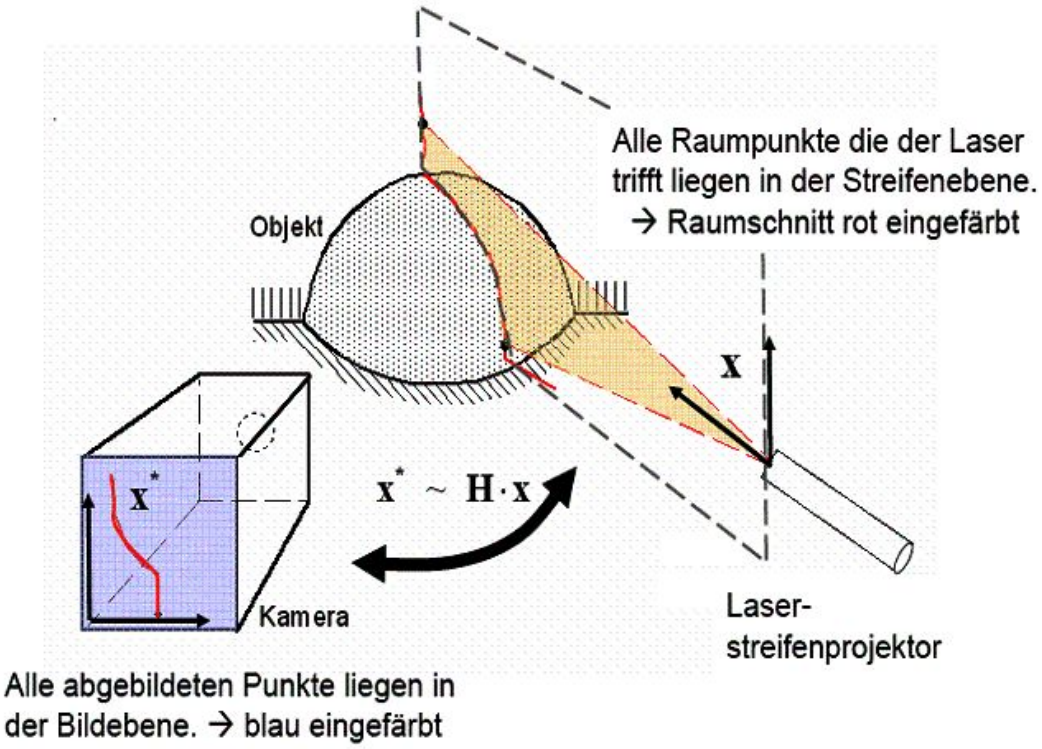


Abbildung 2.1.: Laser-Lichtschnitt-Prinzip

[2] Abbildung übernommen aus Meisel (2007) bzw. dem Präsentationsplakat dieses Projektes.

3. Versuchsaufbau

In diesem Kapitel wird der Aufbau des 3D-Laserscannersystems erläutert. Zusätzlich werden die Befestigungen der Hardwarekomponenten und deren Zusammenspiel beschrieben.

Inhalt

3.1. Der Grundrahmen des Systems

Die Elemente des Grundgestells und der Träger bestehen ausschließlich aus Rexroth-Aluminiumprofilen[3]. Mit den normierten Verbindungselementen zur Profilmontage lassen sich verschiedene Konstruktionen nach dem Baukastenprinzip schnell aufbauen. Alle Profile sind eloxiert und in baureihenspezifischen Rastermaßen aufeinander abgestimmt. An dem Rahmengerüst sind höhenverstellbare Träger montiert, die zur Führung der Querträger dienen. Die Querträger des Systems lassen sich vertikal innerhalb des Grundrahmens verschieben und sind durch Schienen mit den Höhenträgern verbunden. Zur Arretierung werden Nutensteine verwendet. Zur Verstärkung der Profilverbindung der Träger mit dem Grundrahmen werden Winkelstützen verwendet. Am Boden des Gestells ist eine Querstrebe zentral fixiert, an der ein Schrittmotor mit Drehteller montiert ist. Stufenlos höhenverstellbare Stellfüße sind an den Kernbohrungen der Grundprofile zur Ausrichtung des Gesamtrahmens auf dem Untergrund angebracht. Die Maße des gesamten Rahmens betragen: 74(Höhe)x68(Breite)x48(Tiefe) (Maßangaben in cm). Die Anordnung der Komponenten sind in der folgenden Abbildung ersichtlich.

[3] Basic Mechanical Elements, www.boschrexroth.com

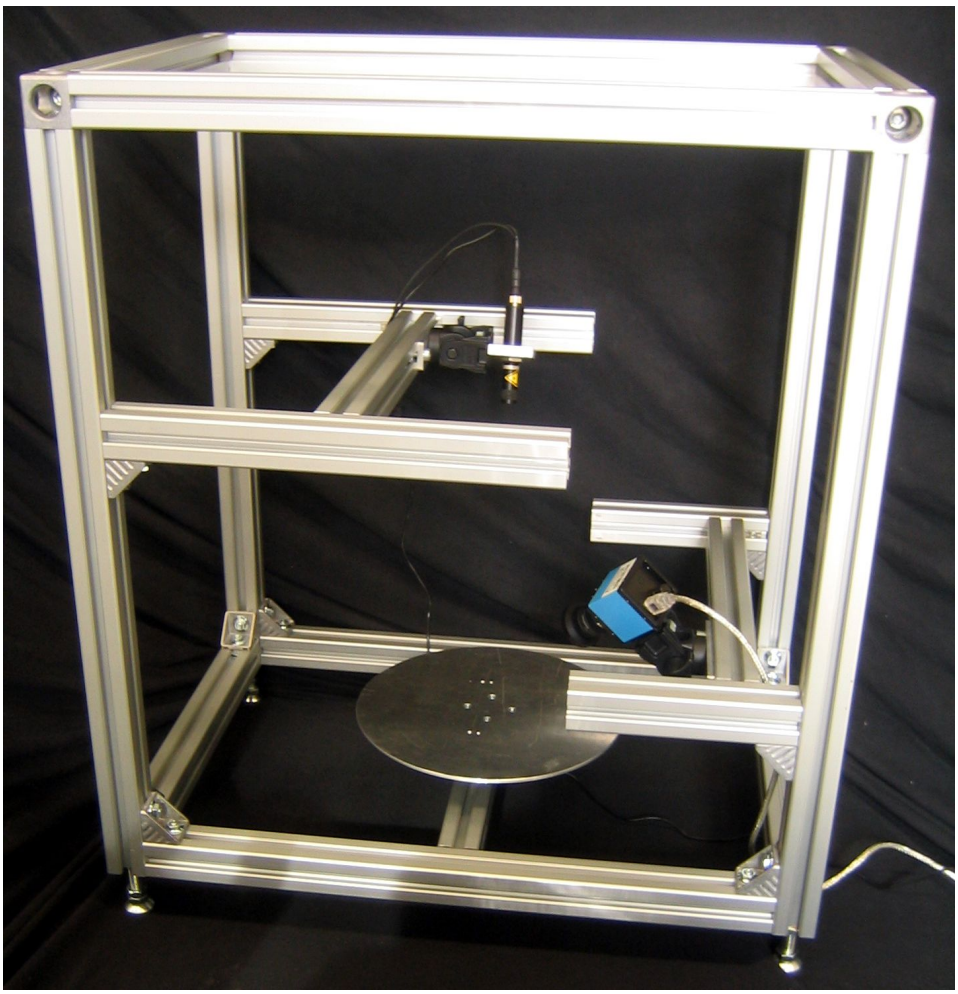


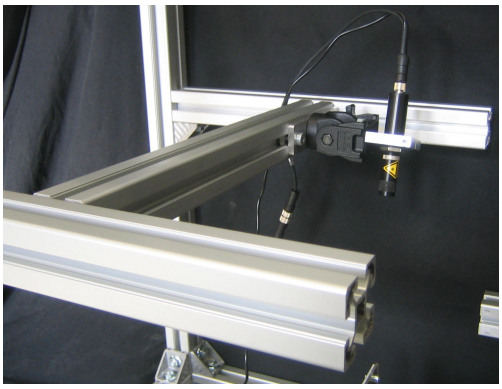
Abbildung 3.1.: Grundrahmen mit Kamera, Laser und Drehtisch

3.2. Drehteller

Als Drehteller wird eine runde Aluminiumplatte mit einem Durchmesser von 285 mm verwendet. Der Teller wird mit dem Schrittmotor über eine Gewindeaufnahme verschraubt. Dazu wurden 4 Bohrungen angebracht, die in einem Radius von 15 mm alle 45° um den Mittelpunkt der Aluminiumplatte positioniert sind.

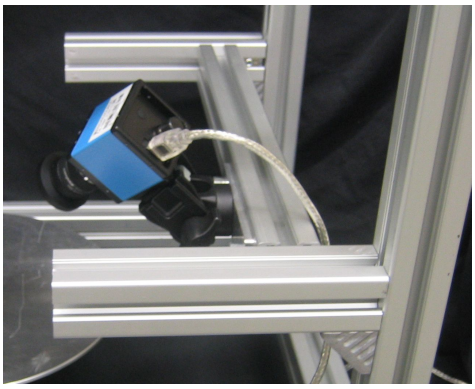
3.3. Montage von Laser- und Kamerakomponente

Als Linearführung zur unterschiedlichen Positonierung von Laser und Kamera werden Schlitten verwendet, die innerhalb der Profilträger verlaufen. Die Fixierung findet durch Nutensteine statt. Zur idealen Ausnutzung aller Freiheitsgrade sind schwenkbare Stative zur Befestigung des Lasers und der Kamera auf den Schlitten angebracht. (Abbildungen übernommen aus Schuhfuss (2007). In Kombination aller Träger- und Halterungskomponenten können somit beliebige Positionen im Verhältniss von Laser und Kamera zum Drehtisch eingestellt werden. Eine ausreichende Fixierung ist durch gute Verschraubung und gewissenhafte Ausrichtung, um Verkantungen zu vermeiden, gewährleistet.

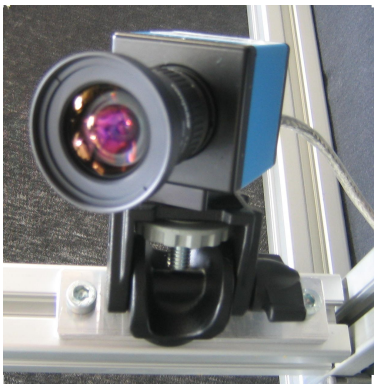


(a) Querträger mit Laserhalterung

(b) Laser mit Halterung



(c) Querträger mit Kamerahalterung



(d) Kamera mit Halterung

Abbildung 3.2.: Montagevorrichtung von Laser und Kamera

3.4. Kalibriervorrichtung

Zum Kalibrieren der Kamera und des Lasers wird auf dem Drehteller eine Kalibriervorrichtung befestigt. Dafür werden eine rechteckige Aluminiumplatte und eine Plexiglasscheibe verwendet. Nachdem ein gedrucktes Testmuster mit bekannter Geometrie zwischen die beiden Platten gelegt ist, werden diese miteinander verschraubt. (Abb.: 3.3). Für einen Kalibriervorgang wird die Aluminiumplatte mit zwei Winkelprofilen mit dem Drehteller verschraubt. Dabei muss das eingespannte Testmuster genau auf die Drehachse des Tellers ausgerichtet werden. Anschließend wird die Laserlinie manuell entlang der oberen Kante der Vorrichtung mittig ausgerichtet.

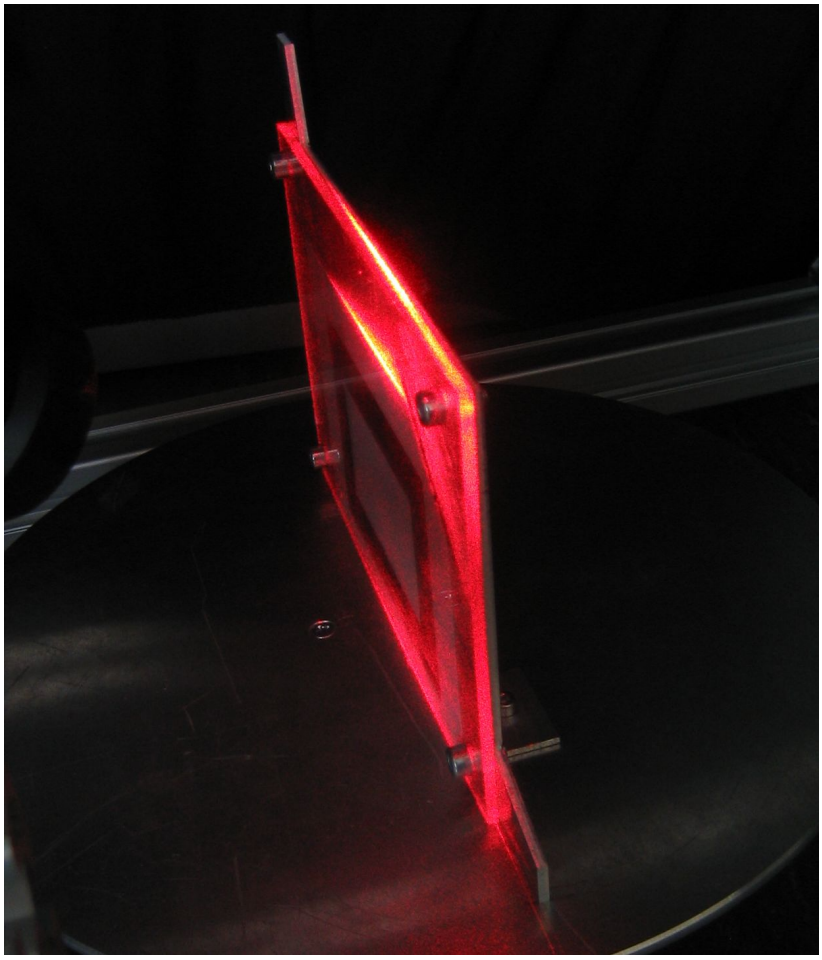


Abbildung 3.3.: Kalibriervorrichtung mit ausgerichteter Laserlinie

4. Realisierung

In diesem Kapitel wird auf das entwickelte Konzept zum Ablauf eines Scannvorgangs eingegangen. Darin sind die einzelnen Schritte von der Vorbereitung der technischen Komponenten zur Kalibrierung sowie das Starten des Scanners zur sequenziellen Bildaufnahme enthalten.

Inhalt

4.1. Prinzipieller Aufbau dieser Arbeit

Der Messaufbau zum Erfassen der Objektkoordinaten besteht im Wesentlichen aus einem Laser, einer CCD-Kamera und einem Drehteller. Laser und Kamera werden nicht bewegt, sie stehen in einem festen sogenannten Triangulationswinkel zueinander. Der Lasersensor projiziert auf das Messobjekt einen gefächerten, linienförmigen Strahl. Die dadurch entstehende Lichtebene schneidet das Objekt entlang einer Profillinie. Entsprechend der Objektform beobachtet man aus dem Blickwinkel einer Kamera eine Verzerrung entlang der Laserlinie. Aus dieser Bildinformation und der bekannten Geometrie von Lichtstrahl und Kamera, wird anhand des Triangulationsprinzips die Kontur des Objektes entlang der Laserlinie ermittelt und ein zweidimensionales Profil gemessen. Zur vollständigen Erfassung der Oberfläche werden durch eine rotative Bewegung des Gegenstandes, auf einem Drehteller, fortlaufend Profillinien aufgenommen. Die 2D-Messdaten werden zur weiteren Bildverarbeitung in einer externen Datei abgelegt.

4.2. Kamerakalibrierung

Die Kalibrierung des Systems ist für die Objektvermessung von zentraler Bedeutung, um zu richtigen Messergebnissen zu gelangen. Durch die Kalibrierung werden die äußeren (extrinsischen) Parameter (Position und Orientierung) sowie die inneren (intrinsischen) Parameter der Kamera bestimmt, die für die 3D-Rekonstruktion erforderlich sind. Die äußeren Parameter werden anhand von bekannten Koordinaten auf einem Kalibriermuster und den dazugehörigen Koordinaten der Punkte auf der Bildebene ermittelt. Die intrinsischen Parameter definieren die Abbildung zwischen dem 3DKamerakoordinatensystem (metrisch) und dem 2D-Bildkoordinatensystem (in Pixel). In diesem Testaufbau wird lediglich die Brennweite als innerer Parameter manuell eingestellt. Mit Hilfe der perspektivischen Transformation können nun die Koordinaten der Laserebene auf die Bildebene abgebildet werden.

4.2.1. Ausrichtung von Kalibriervorrichtung und Laser

Zum Kalibrieren der Kamera und des Lasers wird auf dem Drehteller eine Kalibriervorrichtung auf der Drehachse befestigt. Zwischen einer Metall- und einer Plexiglasplatte befindet sich ein gedrucktes Kalibriermuster (siehe Kapitel: 3 und Abb.: 4.1). Nachdem die Vorrichtung arretiert ist, wird die Laserlinie manuell ausgerichtet. Dabei muss der Strahl innerhalb der Vorrichtung exakt auf die Ebene des Kalibriermusters justiert werden. Als Orientierungshilfe dienen dazu die oberen und unteren hervorstehenden Kanten der Aluminiumplatte. Der Linienstrahl des Lasers sollte scharf eingestellt sein.

Abbildung 4.1.: Testmuster mit bekannter Geometrie zur Kalibrierung

(0, 70)			(100, 70)
	(15, 55)	(85, 55)	
	(15, 15)	(85, 15)	
(0, 0)			(100, 0)

Tabelle 4.1.: Abmessungen des Kalibriermusters

4.2.2. Positionierung der Kamera

Die CCD-Kamera wird in einem Triangulationswinkel zum Laser ausgerichtet. Um eine gute Höhenauflösung zu erreichen wird hier ein Winkel von ca. 45° angewendet. Bei zu großem Beobachtungswinkel besteht das Problem, dass bei nicht ebenen Oberflächen Objektbereiche abgeschattet werden können und so die Laserebene nicht mehr ideal erfasst werden kann. Das Objektiv der Kamera zeigt auf die Kalibriervorrichtung. Das Kalibriermuster sollte bildausfüllend aufgenommen werden.

Zur Einstellung der kameraspezifischen Werte wie u.a. der Brennweite, Belichtungszeit und des Kontrastes, dient die Software-Applikation IC Capture. Dabei ist zu beachten, dass bei der Einstellung der Brennweite das Muster nicht über den Bildrand herausragt. Um eine scharfe Abbildung über die ganze Breite der Kanten des Kalibriermusters zu ermöglichen, ist es empfehlenswert, die Blendenöffnung der Kamera so klein wie möglich zu halten. Eine zu große Blende kann die Bildqualität durch geringere Tiefenschärfe beeinflussen. Deshalb ist es besser die Helligkeit und Schärfe des Bildes anhand der Beleuchtungsstärke einzurichten.

4.2.3. Ablauf eines Kalibriervorgangs

In einem sensorbasierten Lokalisierungsverfahren ist es nötig Referenzpunkte zur Orientierung zur Verfügung zu haben. Da in diesem Projekt auf ein erstelltes Kalibrierbild zurückgegriffen wird, müssen erst mögliche Referenzpunkte extrahiert werden. Hierzu werden im aufgenommenen Bild Merkmale gesucht, die später korreliert werden können. Anhand einer perspektivischen Transformation des Objektbildes mit den korrespondierenden Merkmalen kann dadurch ein Referenzbild ermittelt werden.
Zur Bestimmung der Referenzwerte liegt der Algorithmus aus der Bachelorarbeit Schuhfuss (2007) zugrunde.

1. **Merkmalextraktion**

   Zur Kantendetektion im Originalbild wird der weit verbreitete Canny-Algorithmus angewandt. Mit diesem Algorithmus soll durch mehrere Faltungsoperationen ein Kantenbild entstehen, das nur noch die Linien des Kalibriermusters enthält. Dabei werden Helligkeitsschwankungen zwischen benachbarten Pixel im Bild betrachtet und als Unstetigkeiten der Grauwertfunktion *g(x,y)* im Ausgangsbild aufgefasst. Hierbei findet zuerst eine Glättung durch den Gauss-Filter im Bild statt, um ein Rauschen zu unterdrücken. Mit Hilfe des Sobel-Operators werden dann die Gradienten in X-Richtung g(x) oder in Y-Richtung g(y) bestimmt und somit die horizontalen oder vertikalen Kanten hervorgehoben, indem das Bild mit den Operatoren in die angegebenen Richtungen gefaltet wird. Dadurch entstehen zwei neue Bilder in die jeweilige Richtungen der Kanten mit den Gradienten der einzelnen Pixel.
   Durch das Addieren der Komponenten der Grauwertbilder G_x und G_y können die Gradientenbeträge G

$$G = \sqrt{G_x^2 + G_y^2} \tag{4.1a}$$

   und deren Richtungen

$$\alpha = \arctan \frac{G_y}{G_x} \tag{4.1b}$$

   bestimmt werden[4].

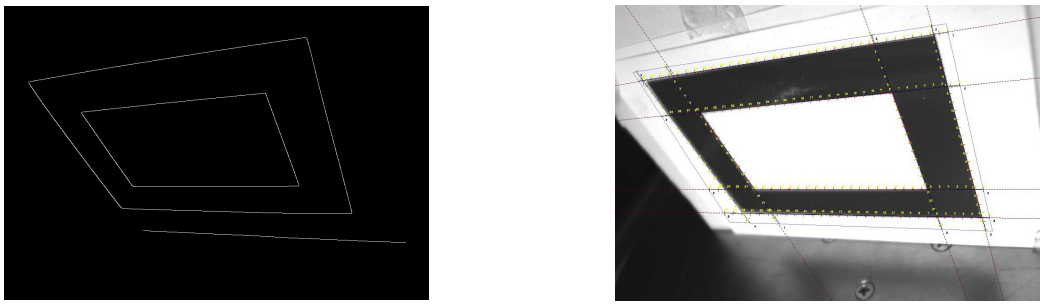


Abbildung 4.2.: Kantendetektion mit Canny und Positionsermittlung

   Daraus entsteht das eigentliche Gradientenbild, das die Kanten in beide Richtungen beinhaltet.

[4] entnommen aus Meisel (2005)

Es ist in dem gefalteten Bild zu erkennen, dass außer den Kanten des Kalibriermusters auch noch Linien in der Umgebung detektiert wurden (Abb.:4.2). Um diese Bereiche zu unterdrücken, wird im ausführenden Kalibrierprogramm zusätzlich ein *"Region-of-Interest"* mit angegeben. Die Grenzen des *"ROI"*-Polygons können mit Hilfe von IC Capture abgeschätzt und beim Ausführen einer Kalibrierung als Parameter über die Kommandozeile angegeben werden in der Form:

$$--roi = "(250,235)(960,125)(470,575)(1065,595)" \tag{4.2a}$$

2. **Ermittlung der Kantenpositionen**

   Mit dem Canny-Algorithmus sind nun die Kanten zur Referenzierung ermittelt worden, aber noch nicht die Geraden und deren Position im Raum. Dafür benutzt man den Hough-Algorithmus zur Erkennung von Geraden in binärisierten Bildern. Dabei ordnet die Hough-Transformation jedem Kantenpunkt im Binärbild die Geraden (Geradenbüschel) zu, die durch diesen Punkt laufen. Jede Gerade ist beschrieben durch die Parameter einer Steigung und Ordinaten-Abschnitt Steinbrecher (1993). Für alle Punkte wird im Parameterraum (Hough-Raum) eine Gerade mit den Parametern gezeichnet. Liegen mehrere Punkte im Raum kollinear, schneiden sich die entsprechenden Hough-Geraden in einem Punkt. Findet man im Hough-Raum Schnittpunkte von Geraden, so hat man Geraden im Bild detektiert. Dadurch sind die Linien ermittelt.

3. **Berechnung von Referenzpunktpaaren**

   Um zuletzt die Eckpunkt-Koordinaten des Kalibriermusters zu bestimmen, werden die Schnittpunkte der horizontalen und vertikalen Geraden im Bild durch die Abgleichung der Koordinaten auf der Laserebene berechnet. Die ermittelten Referenzpunktpaare ermöglichen anhand einer perspektivische Transformation die Transformationsmatrix zu erstellen, die auf alle Bilder angewandt wird, die während eines Scannvorgangs aufgenommen werden.

4.3. Scannen einer Szene von Bildern

Eine Oberflächenvermessung zur anschliessenden 3D-Rekonstruktion setzt voraus, eine ausreichende Anzahl von Bildern eines Objektes zu erstellen. Dabei wird das Messobjekt auf dem Drehteller um 360° am Laser vorbeibewegt und idealerweise pro Grad ein Kamerabild erfasst und abgespeichert. Die Schrittweite ist aber auch optional innerhalb des Programms veränderbar. Der zu vermessende Gegenstand wird zentral auf der Platte des Drehtisches positioniert um den Laserstrahl während des Scannens immer zu erfassen. Die Laserlinie auf dem Objekt besitzt eine hohe Lichtintensität. Um dies optimal auszunutzen, sollte der Messaufbau abgedunkelt werden, bis nur noch die Laserlinie im Bild zu sehen ist. Um die Einstellungen der Kamera im Live-Modus nachzujustieren, kann dafür das IC Capture Programm benutzt werden.
Beim Starten des Scann-Programms findet zuerst ein Verbindungsaufbau mit der Positioniersteuerung und eine Initialisierung der angesteuerten Achse vom verwendeten Motortyp (DMT65+SM240) statt. Anschließend wird die Kamera initialisiert und die Parametereinstellungen übernommen. Zu Beginn eines jeden Scannvorgangs um 360° wird die aktuelle Position des Drehtisches auf 0 gesetzt. Ein Beispielprogramm, in dem alle Parameter gesetzt werden um den Drehvorgang und die Bildaufnahme einzuleiten, ist in Kapitel 5 ausführlich beschrieben.

Abbildung 4.3.: Zwei verwendete Objekte zur Oberflächenabtastung
Diese beiden Testobjekte, eine mexikanische Maja-Maske aus gebranntem Ton und eine Schmuckschatulle aus Holz eignen sich sehr gut zum scannen, da die Materialbeschaffenheit kaum diffuse Reflexion zulässt.

Nach jeder Bildaufnahme muss sich der Drehteller um eine Schrittweite in die nächste Position bewegen. Dazu wird eine bestimmte Zeit benötigt, die von der Motorgeschwindigkeit abhängig ist. Während der Zeit der Positionsänderung muss deshalb das Programm solange angehalten werden, bis sich das Objekt in der richtigen Position befindet. Dadurch entstehen keine falschen Bildsequenzen, während sich der Drehteller noch bewegt. Dieses Zeitfenster

ist vor allem bei einer Änderung der Schrittgrösse zu berücksichtigen und muss an die Motorgeschwindigkeit angepasst werden. Die Dauer der Programmunterbrechung sowie die Ermittlung der Winkelposition (in °Grad) werden im Programm anhand der vorher definierten Bilderanzahl/Vollumdrehung kalkuliert[5].

Abbildung 4.4.: Bildsequenzen während eines Scanvorgangs

[5]siehe Anhang B: 'Scannen einer Szene'

4.4. Objektvermessung und Datenexportierung

Die Objektvermessung ist in diesem Projekt vom Scannvorgang abgekapselt. Dieses Konzept ist so vorgesehen um die Entwicklung nachfolgender Programm-Module für die Bildverarbeitung zu vereinfachen.
Die erfassten Bilder werden nach der Beendigung eines Scannvorgangs in einem Verzeichnis gespeichert, das im Programm angegeben wird.

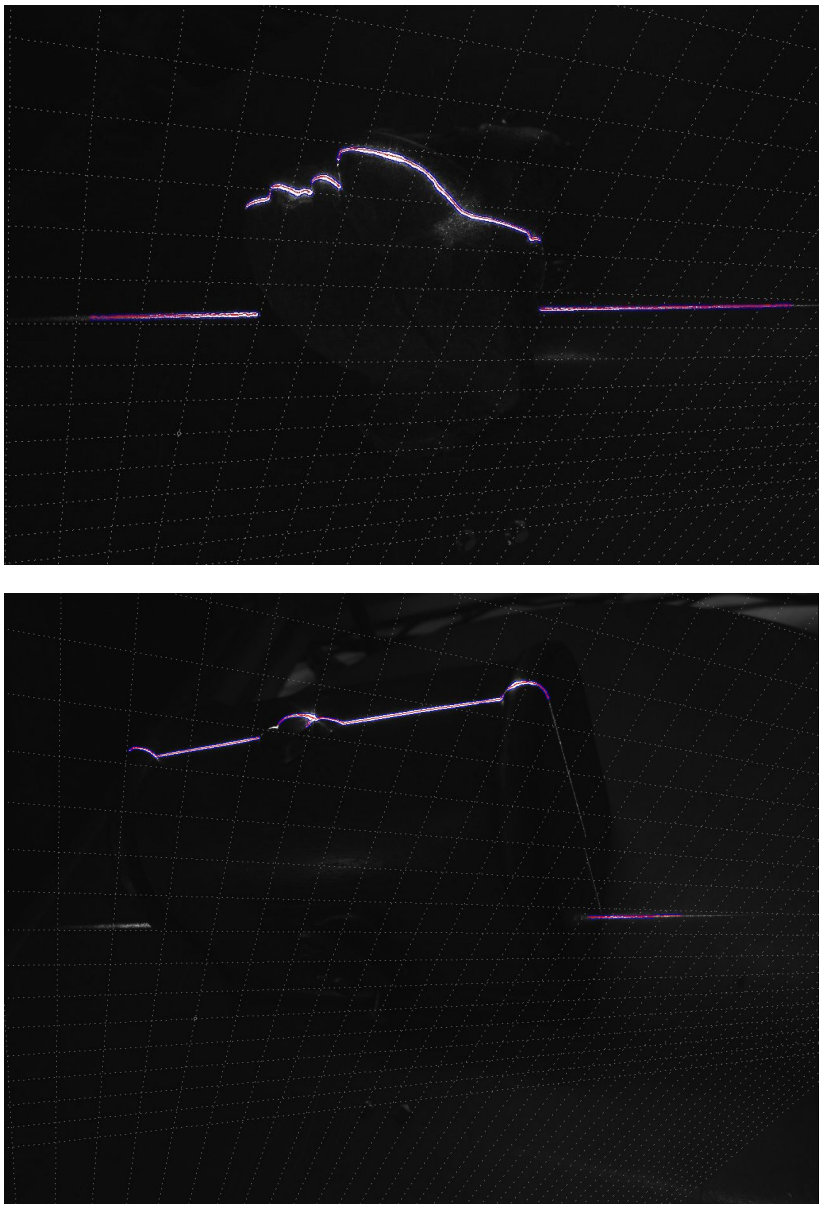


Abbildung 4.5.: Transformierte Darstellung nach einer Objektvermessung

Zum Vermessen werden die Bilder automatisch in aufsteigender Reihenfolge ihrer Erstellung geladen. Im Programm wird das jeweilige Bild und die Transformationsmatrix, welche bei der Kalibrierung erstellt wurde, der Funktion zur Objektvermessung als Parameter übergeben. In dieser Funktion werden die Mittelpunkte der Laserlinie im Kamerabild gesucht und anschließend die Bildkoordinaten in die Koordinaten der Laserebene umgerechnet.
Die somit ermittelten Messdaten werden in eine einzige externe Text-Datei geschrieben. Die Daten der einzelnen Bilder sind dabei durch Textseparatoren getrennt wie in der folgenden Abbildung:

Ohne Winkelübergabe beim Daten schreiben:

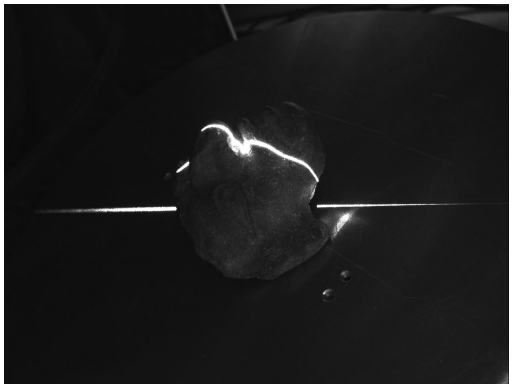
```
# result file. contained dataformat 'x y' image: 1
7.360659 36.320946
9.281705 35.592949
9.589590 35.051175
...............

# result file. contained dataformat 'x y' image: 2
-5.377329 35.350712
-5.344682 34.831707
-5.315247 34.646770
...............
```

Mit angegebenem Positionswinkel:

```
# result file. contained dataformat 'x y' image: 1
-10.589590 32.026375 45
-10.449151 32.002937 45
-10.327989 31.955940 45
.................
# result file. contained dataformat 'x y' image: 2
-18.025991 31.413942 90
-18.053537 31.629829 90
-17.950272 31.426756 90
.................
# result file. contained dataformat 'x y' image: 3
-18.767780 31.627565 135
-18.620653 31.644268 135
-18.526649 31.653923 135
.................
.................
.................
# result file. contained dataformat 'x y' image: 6
-17.925529 31.505798 270
-17.770805 31.489124 270
-17.836164 31.727116 270
.................
# result file. contained dataformat 'x y' image: 7
-22.155220 31.516434 315
-22.013052 31.509314 315
-21.993765 31.505135 315
.................
# result file. contained dataformat 'x y' image: 8
-18.113440 31.705385 360
-18.024090 31.721621 360
-17.907595 31.792019 360
```

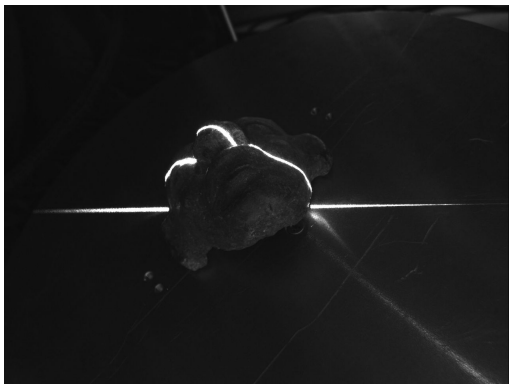
(a) 45°

(b) 90°



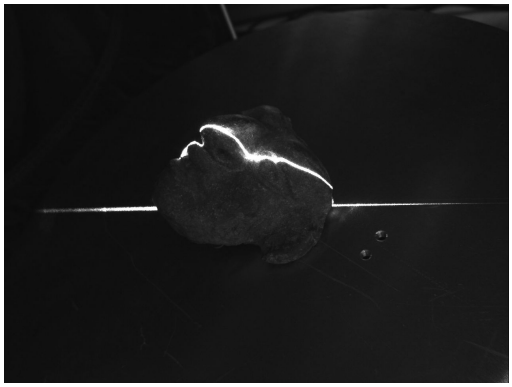
(c) 135°

(d) 180°



(e) 225°

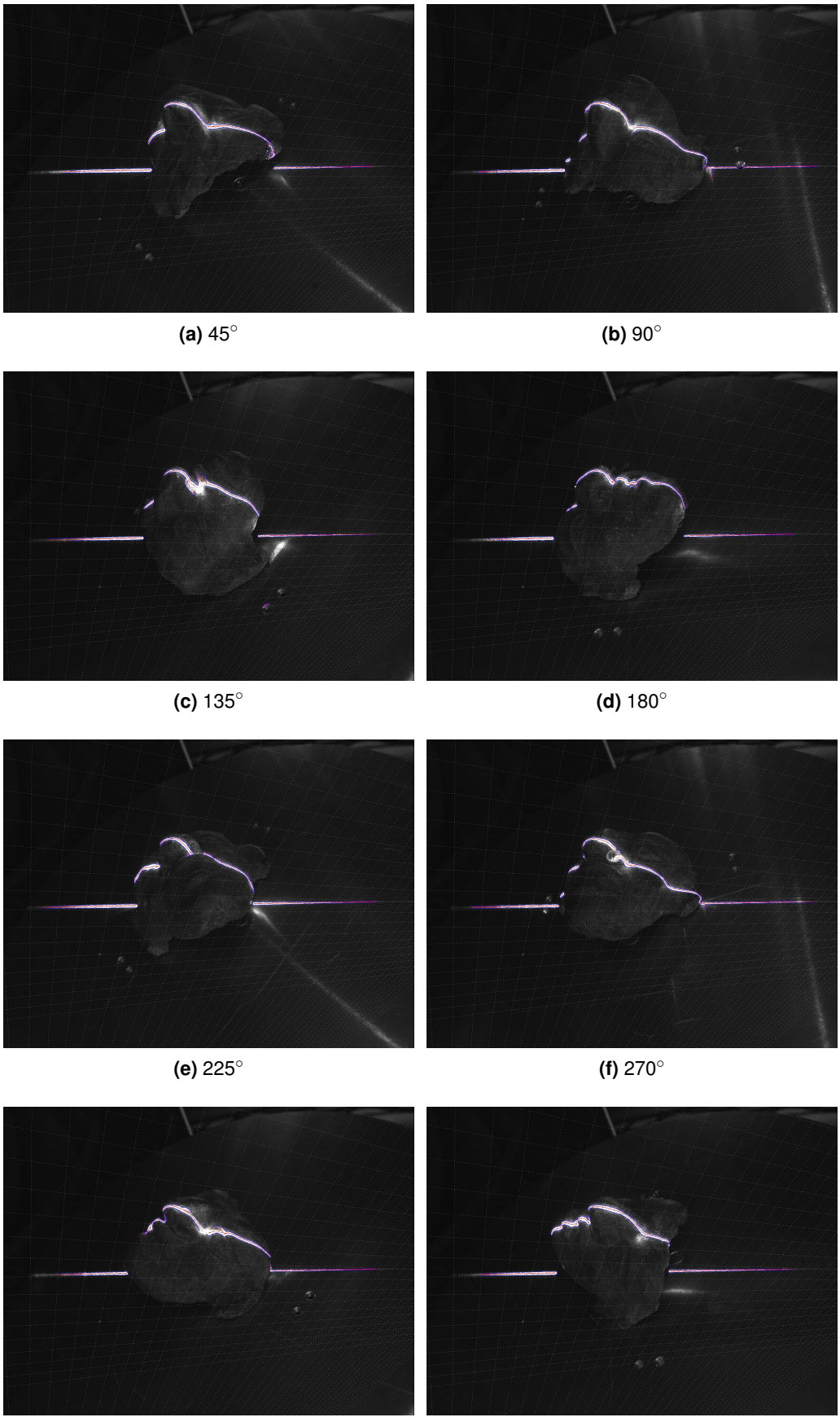
(f) 270°



(g) 315°

(h) 360° = 0°

Abbildung 4.6.: Aufnahmesequenz in 8 Einzelschritten á 45°



(a) 45° **(b)** 90°

(c) 135° **(d)** 180°

(e) 225° **(f)** 270°

(g) 315° **(h)** 360° = 0°

Abbildung 4.7.: Die ausgewerteten Bilder der Aufnahmesequenz

5. Implementierung

Die Aufgabe dieser Arbeit war die Erstellung einer Anwendungsbibliothek die die Steuerungsbefehle mit den spezifischen Parameter für die verwendeten Hardwarekomponenten, deklariert. In dieser Dokumentation werden die angewandten Funktionen und vor allem deren Bedeutung ausführlich erklärt. Anhand von Programmbeispielen zur einfachen Ansteuerung des Drehtellers und der Bildaufnahme wird der schematische Ablauf ersichtlich. Eine Beispielmessung soll zeigen, wie die Objektdaten pro Bildsequenz angelegt und exportiert werden.

Inhalt

5.1. Einbindung der Bibliotheken

Im Programm wird die Bibliothek der Steuerung $ps90.dll$ verwendet. Sie dient zur dynamischen Einbindung der Steuerungs-Funktionen. Dazu stehen zwei Dateien zur Verfügung:

$ps90dyn.h$	deklariert die Klasse **CPS90Win** um die Funktionen einbinden zu können.
$ps90dyn.cpp$	definiert die Klasse **CPS90Win**.

Die Datei $ps90dyn.cpp$ muss in das erstellte Projekt direkt mit eingebunden werden, um die Funktionen zu verwenden. Die Header-Datei $ps90dyn.h$ wird hier über den Installationspfad durch Anlegen eines Property-Sheets[6] als Makro im Projekt angesprochen.

[6] Ein Property-Sheet kann unter Visual Studio C++ im Eigenschaften-Manager hinzugefügt werden, die die Pfade der Include-Verzeichnisse und Bibliotheks-Verzeichnisse beinhalten.

5.2. Konfiguration des Schrittmotors DMT65+SM240

Um eine problemlose Ansteuerung des motorisierten Drehtellers zu ermöglichen, wurde eine Header-Datei angelegt, die die Konfigurationseinstellungen global definiert. In dieser Konfigurationsdatei werden alle hardwareseitigen Einstellungen des Motors abgelegt sowie die Schnittstellenbeschreibung. Zusätzlich sind achsenspezifische Parameter mit konstanten Werten vordefiniert, die mit den Steuerungsbefehlen gesetzt werden. Die Bezeichnung der Header-Datei ist ***config.h***. Im Anhang B ist das Listing der gesetzten Parameter zu finden.

5.2.1. Verbindungsaufbau

Die Kommunikation zwischen PC und Steuerung findet über eine USB-Schnittstelle statt. Die PS90-Funktion *Connect(int,int,int,int,int,int,int)* öffnet eine Schnittstelle und baut eine Verbindung auf. Die dazu verwendeten Parameter werden hier erläutert:

Funktionsaufruf mit Voreinstellungen:

```
/** Connection Default : */
long Connect(
          int nController     // Controller Type (1..10)
          int nInt;           // 0 - SERIAL
          int nCom;           // 1 - COM1
          int nBaud;          // 9600
          int nHandshake;     // 0
          int nParity;        // 0
          int nDatabits;      // 8
          int nStopbits;      // 0 - CR
 )
```

In diesem Projekt sind die Parameter folgendermaßen gesetzt:

Parameter	Wert	Bedeutung
nController	3	Steuergerät-Typ PS90
nInt	0	Interface zur Schnittstellen-Definierung 0=Seriell ; 1=NET
nCom	3	COM-Portnummer
nBaud	115200	Datenübertragungsrate (bps)
nHandshake	0	Enderkennung CR
nParity	0	Paritätsbit: 0=Kein ; 1=Ungerade ; 2=Gerade
nDatabits	8	Standartlänge der Datenbits
nStopbits	0	OneStopBit nach Datenblock

Tabelle 5.1.: Verbindungsaufbau mit dem Steuergerät PS90

5.2.2. Motor-Initialisierung

Nachdem eine Verbindung mit dem Steuergerät hergestellt wurde, ist es notwendig, den Motortyp für eine Achse einzustellen. Die angesteuerte Achse muss in allen Funktionen mit übergeben werden. Dabei sind zur Initialisierung des Schrittmotors folgende Funktionen wichtig:

- `long CPS90Win::SetMotorType(int nAxis, int nType)`

  ```
  /** Motor Initialisation
  0=DC - Brush servo motor
  1=Step motor Open Loop Dir/Clock
  2=Step motor Open Loop PMD
  3=Step motor Closed Loop
  4=BLDC servo motor
  */
  ```

  Bedeutung: Einen Motortyp für eine Achse einstellen.
  Mit dieser Funktion wird ein Motortyp für die anzusteuernde Achse gesetzt. Der verwendete Schrittmotor DMT65+SM240 läuft im Open Loop PMD - Modus: 2.

- `long MotorInit(int nAxis)`

  Bedeutung: Eine Achse initialisieren und einschalten.
  Mit dieser Funktion wird die Achse komplett initialisiert und befindet sich anschließend im bestromten Zustand. Sie muss nach dem Einschalten der Steuerung und nach der Einstellung des Motortyps ausgeführt werden.

- `long MotorOn(int nAxis)`

  Bedeutung: Eine Achse einschalten.
  Mit dieser Funktion wird die Achse, nachdem der Motor stromlos geschaltet war, wieder eingeschaltet und befindet sich anschließend im bestromten Zustand.

5.2.3. Achsenspezifische Parameter

Folgende Funktionen wurden im Programm zur Kommunikation mit dem Schrittmotor benutzt:

- `long SetPosMode(int nAxis, int nMode);`

  Bedeutung: Setzt den Positionier-Modus der Achse.
  Der Schrittmotor nutzt den Trapez-Profil-Geschwindigkeitsmodus und fährt das Ziel mit linear steigender Geschwindigkeit an.
  Positionier-Modus: Trapez=0 / S-Kurve=1

- `long SetTargetMode(int nAxis, int nMode);`

  Bedeutung: Positionsangabeformat einer Achse einstellen.
  Bei einer relativen Positionsangabe wird ein Distanzweg zurückgelegt. Bei absoluter Angabe fährt der Schrittmotor eine gewünschte Zielposition an.
  Ziel-Modus: relativ=0 / absolut=1.

- `long SetPositionEx(int nAxis, double dVal);`

  Bedeutung: Dies ist eine erweiterte Funktion.
  Darin wird die aktuelle Positon gesetzt. Die Positionseinheit wird für den spezifischen Motortyp in mm(deg) gerechnet. Die Voreinstellung zählt die Position inkrementell und benutzt deshalb die Standart-Funktion "long SetPosition(int nAxis, double dVal)", welche hier nicht verwendet werden kann, da sonst ein Fehler verursacht wird.

- `long MoveEx(int nAxis, double dVal, bool bNoInc);`

  Bedeutung: Die eigentliche Funktion um die Achse in die neue Zielposition zu fahren.
  Darin werden 3 Parameter übergeben:

  1. Der erste Parameter gibt die anzusteuernde Achse an.
  2. Der zweite Parameter enthält den Wert, um den sich der Motor rotativ bewegen soll. Dieser Wert ist abhängig von den Tischparametern (Spindelsteigung, Untersetzung) und den Schrittmotordaten (Schritte pro Impuls).

     1*° entspricht mit der motorseitigen Einstellung dem Wert: *0.05

  3. Als dritter Parameter wird der boolean-Wert auf *true* gesetzt um den inkrementellen Possitionszähler zu deaktivieren.

5.3. Beispielprogramm zum Ansteuern des Drehtellers

Anhand eines einfachen Programms wird der Programmablauf gezeigt. Die Include-Dateien beziehen sich nur auf die Motoransteuerung.

```
#include "config.h"
#include "stdafx.h"
#include "ps90dyn.h"

int      main(int argc, char* argv[])
{
    // Instanziierung
    CPS90Win stepper;
    // Parameter in Konfigurationsdatei definiert
    stepper.Connect(_CONTROLLER,_INT,_PORT,_BAUD_RATE,
                    _HANDSHAKE,_PARITY,_DATABITS,_STOPBITS);
    stepper.SetMotorType(1,_MOTORTYPE);
    stepper.MotorInit(1);
    stepper.SetPosMode(1,_POSMODE);
    stepper.SetTargetMode(1,_TARGETMODE);

    //Verbindungsabfrage (Optional)
    if(stepmotor.IsConnected()){
            double newPosition = 0;
            // 1° pro Schritt
            double degree    = 0.05;
            for(int i =0 ; i<360 ; i++){
                    //Aktuelle Position setzen
                    stepmotor.SetPositionEx(1,0);
                    stepmotor.MoveEx(1,degree,true);
                    Sleep(1000);
                    newPosition = newPosition + degree;
            }
            cout<<"\nPress any key to homeposition\n"<<endl;
            getchar();
            //In Ausgangsposition zurück fahren
            newPosition = newPosition - degree;
            stepper.MoveEx(1,-newPosition,true);//+/- vorwärts/rückwärts
    }
    else cout<< stepper.GetConnectInfo()<<endl;
    stepper.Disconnect();
    //stepper.~CPS90Win();
    return 0;
}
```

5.4. Bildakquise mit IC Capture

Die Bildaufnahmesequenz während eines Scanvorgangs ist angelehnt an das Beispielprogramm *MemBufferCollection* aus der *IC Imaging Control C++ Class Library*.

Im folgendem werden die wichtigsten Methoden zur Bildaufnahme mit der Imaging Source Kamera erklärt:

- Namespace: DShowLib

- `DShowLib:InitLibrary(SerialNumber)`

  Um die Kamerabibliothek laden zu können, muss eine Lizenznummer mit übergeben werden.

- `Grabber grabber;`

  Dieses ist die Main-Klasse der IC Imaging Control Class Library. Sie liefert die grundlegende Funktionalität, um den Stream der Image Buffer zu steuern.

- `grabber.startLive( true );`

  Starten des Live-Modus (*false* für zeitlich getrennte Einzelbilder)

- ```
  DShowLib::tsPropertyRange TPR;
  TPR = grabber.getPropertyRange( CameraControl_Exposure);
  ```

  Die tsPropertyRange beinhaltet eine Liste von Einstellungsmöglichkeiten der verwendeten Kamera.

- ```
  printf("\n\nExposMin=%d,ExposMax=%d\n", TPR.min,TPR.max);
  grabber.setProperty( CameraControl_Exposure, false);
  grabber.setProperty( CameraControl_Exposure,(long)-9);
  ```

  Hier werden die Einstellungswerte zuerst abgefragt und danach die automatische Belichtung deaktiviert und manuell gesetzt.

- `grabber.setOverlayBitmapPathPosition( ePP_NONE );`

  Legt die Position des OverlayBitmap fest.

- `tFrameHandlerSinkPtr pSink = FrameHandlerSink::create( eY8, 1 );`

  Setzt das Format des Image Buffer. eY800 bedeutet monochrome, 8 bits (1 Byte) pro Pixel.

- `pSink->setSnapMode(true);`

  Aktivieren des Snap Modus. In diesem Modus werden die aufgenommen Bilder in aufsteigender Reihenfolge ihrer Erstellungszeit gepuffert.

- `grabber.setSinkType( pSink );`

  Setzt das Format eines Sink. Ein Sink spezifiziert das Ausgangsziel eines Image Streams.

- `tMemBufferCollectionPtr pCollection = MemBufferCollection::create( info, buffers, pBuf );`

  Erstellen eines neuen Ring-Buffers mit den Parametern:

  FrameTypeInfo: beschreibt den Frametype der MemBufferCollection.

  buffers: Eine Anzahl von Buffern, die in der Collection erstellt werden sollen.

  pBuf: Ein Array mit Image-Data Pointern.

- `pSink->snapImages( 1 );`

  Erstellt ein Bild und speichert es in der MemBufferCollection.

- `grabber.stopLive();`

  Stoppt die Liveaufnahme.

- `grabber.closeDev();`

  Schließt den Device.

- `pCollection->save( "C:\ScanImages\file*.bmp" );`

  Mit dem Platzhalter (*) können die Bilder aus der MemBufferCollection mit aufsteigender Nummer im Bitmap Format im angegebenen Verzeichnis gespeichert werden.

5.5. Auswertung der Bilder

Nach einem abgeschlossenen Scanvorgang können die gespeicherten Bilder vermessen werden. Dazu steht das Programm aus der bereits erwähnten Bachelorarbeit zur Verfügung. Zur Objektvermessung anhand einer Bildtransformation muss die bei der Kalibrierung erstellte Transformationsmatrix eingelesen werden. Zum Laden der einzelnen Bilder wird die sehr nützliche LTIlib-Klasse: "loadImageList"verwendet. Deren Anwendung ist im Anhang B zu finden.

6. Verwendete Hardwarekomponenten

Inhalt

6.1. Universal-Positioniersteuerung PS 90

Abbildung 6.1.: Universal-Positioniersteuerung PS 90 der Firma OWIS

Motorisierte Positioniersysteme werden in vielen Bereichen der Forschung, in Laboratorien sowie im industriellen Bereich zur Automatisierung komplexer Bewegungsabläufe verwendet.

Die PS 90 der Fima OWIS[7] ist eine modular aufgebaute Positioniersteuerung für maximal neun Achsen und wird in diesem Projekt zum Ansteuern des Schrittmotors für lediglich eine Achse benutzt. Für den Schrittmotor in dieser Arbeit wird ein rotatives Wegmesssystem verwendet, das motorspezifisch die Positionsangaben in mm(Grad) einsetzt. Um mit dem Steuergerät PS 90 kommunizieren zu können muss eine Schnittstelle im System vorhanden sein. Hierzu steht eine integrierte USB-Schnittstelle zur Verfügung, die sich virtuell wie eine gewöhnliche serielle Schnittstelle "RS232"verhält.

Die Steuerung erfolgt achsenorientiert. Das heißt, wenn das Steuergerät verbunden und die Achse definiert ist, muss diese Achse überall im Programm mit angegeben werden.

Hinweis zur entwickelten Software:
Die PS 90-Applikation (Abb.: 6.2) besteht aus einem Initialisierungsteil, welcher die erforderlichen Parameter für alle verwendeten Achsen setzt und einschaltet, einer Schleife, die eine Referenzfahrt für alle Achsen durchführt und dem eigentlichen Anwenderprogramm, welches die gewünschte Funktionalität beinhaltet. Da in dieser Anwendung nur eine Achse angefahren wird, ist eine Referenzfahrt zur relativen Positionierung mehrerer Achsen zueinander in diesem Sinne nicht notwendig. Die Initialisierung der angesteuerten Achse geschieht durch das MotorInit(AchsenNr)-Kommando.

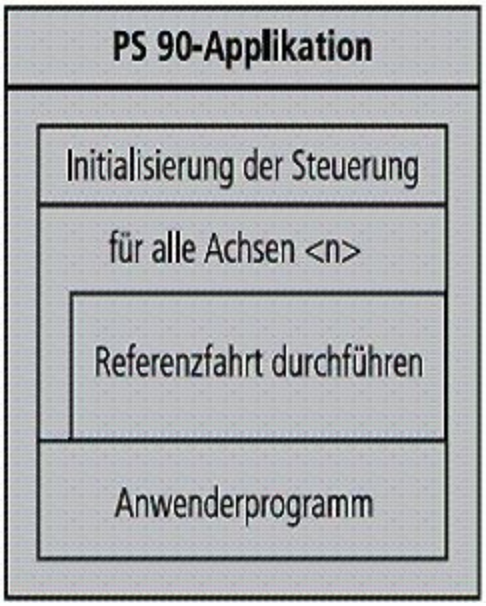


Abbildung 6.2.: PS 90 Applikations-Architektur

[7]OWIS GmbH, Im Gaisgraben 7, 79219 Staufen `http://www.owis-staufen.de/index.html`

6.1.1. Das Softwarepaket für das Steuerungssystem PS 90

Zum Lieferumfang der Steuerung gehört das Softwaretool OWISoft. Dieses Paket wird benötigt um die Kommunikation zwischen PC, PS 90 und dem eingesetzten Drehtisch zu ermöglichen.

Installationsanweisung:

Die Software befindet sich auf der mitgelieferten CD. Nach der Installation werden die Dateien auf der Festplatte in ein eigenes Verzeichnis `.../OWISoft/ps90/win32` mit drei Unterverzeichnissen abgelegt. Das Application-Verzeichnis verwendet man um das Steuerungsprogramm eigenständig zu starten, das hier nicht angewendet wird. Von wesentlicher Bedeutung ist der Quellcode, der sich in den folgenden Dateiordnern befindet:

- **VC++** beinhaltet den Quellcode des Steuerprogramms. Der Code beschreibt wie man in einem VC++ Programm die Funktionen deklariert und sie einsetzen kann.
- **SDK** beinhaltet die Dateien zur Einbindung in das für diese Arbeit erstellte VC++ Projekt, die hier genauer beschrieben werden:

  ps90.dll Die PS 90 Funktionsbibliothek enthält alle Funktionen für die Kommunikation und zur Steuerung.

  ps90dyn.h Deklariert die Klasse CPS90Win zur dynamischen Einbindung der Funktionsbibliothek in eine VC++ Anwendung. Darin sind Funtionen zur Systemkonfiguration, Operationen zur Ansteuerung des Steuergeräts und für Anwenderspezifische Einstellungen abgelegt.

  ps90dyn.cpp Quellcodedatei der eingebundenen Funktionsbibliothek. Definiert die Klasse CPS90Win aus der Header-Datei.

Der Programmablauf eines Verbindungsaufbaus zur Initialisierung der Schnittstelle und des Motors, wird im implementierten Projekt, in Kapitel 5 genau beschrieben.

Die technischen Daten der Positioniersteuerung sind im Anhang A angegeben.

6.2. Drehmesstisch DMT65+SM240

Abbildung 6.3.: Schrittmotor für den Drehtmessisch DMT65+SM240 der Firma OWIS

Der verwendete Drehtisch wird angetrieben durch einen Schrittmotor mit einem unbegrenzten Drehwinkel. Die Auflösung beträgt weniger als eine Winkelminute. Der Antrieb erfolgt durch ein vorgespanntes Schneckengetriebe, welches eine nahezu spielfreie Übertragung ermöglicht. Das Untersetzungsverhältnis der Getriebewellen, gerechnet in Richtung des Kraftflusses, beträgt 180.0000:1. Das erklärt den langsameren Lauf der angetriebenen Welle gegenüber der antreibenden.

Im entwickelten Programm werden die empfohlenen Voreinstellungen von OWIS für den Schrittmotor verwendet[8]. Diese Einstellungen betreffen insbesondere die Datenübertragung. Bei der Weiterentwicklung des Projektes ist es von Vorteil die Geschwindigkeit des Drehvorgangs zu erhöhen. Dadurch lässt sich bei einer vollständigen Aufnahmesequenz von z.B. 360 Einzelschritten pro Umdrehung ein Messvorgang stark verkürzen. Die Konfigurationsdaten des Drehtellers DMT65+SM240 werden im folgenden tabellarisch in 6.1 und 6.2 dargetellt. Diese Angaben sind sehr hilfreich bei der Berechnung der Einzelschritte bei unterschiedlichen Winkeleinstellungen. Die Ausgabeformate werden für die Position in [degree] bzw. für die Geschwindigkeit in [degree/sec] angegeben.

[8] Produktangaben unter `http://www.owis-staufen.de/de/handbuecher-index.html`

Spindelsteigung	360.0000 [deg]
Untersetzung	180.0000 : 1

Tabelle 6.1.: Tischparameter

Antriebsart	Schrittmotor Open Loop PMD
Bewegungstyp	Rotativ
Vollschritte pro Umdrehung	200 (entspricht 1.8° pro Schritt)
Mikroschrittfaktor	1/50

Tabelle 6.2.: Schrittmotordaten

Weitere technischen Daten des Drehmesstisch mit dem dazugehörigen Schrittmotor sind im Anhang A angegeben.

6.3. Laser und Kameraobjektiv

(a) Das eingesetzte Lasermodul

(b) Kameraobjektiv H0514-MP

Abbildung 6.4.: Laser und Kameraobjektiv

In diesem Projekt wird ein Laser der Firma Z-Laser verwendet.[9] Die Laserlinie der Optik ist Gaussverteilt. Eine geringer optischer Winkel bis max. 30° ermöglicht eine hohe Intensität.

Das hochauflösende Kameraobjektiv ist speziell für Aufnahmen im Nahbereich mit Megapixel Kameras konzipiert.

Weitere Leistungsmerkmale des Objektivs *Computar H0514 MEGAPIXEL*[10]:

- eine feste Brennweite,
- manuelle Blende,
- geringe Bildverzerrung,
- hohe Lichtintensität,
- eingebaute Arretierung für Fokus und Blende.

Technische Daten sind im Anhang A zu finden.

[9] Homepage:`http://www.z-laser.com/zlaser_de/visionlaser/zv-ttl`

[10] Homepage:`http://www.cbc-de.com/index.php?id=69`

6.4. FireWire Monochrome CCD-Camera DMK 41BF02

Abbildung 6.5.: Firewire CCD-Kamera von IC Imaging Control

Die FireWire-Kamera von The Imaging Source benutzt das Format Y800. Es stellt pro Pixel 256 Grauwerte zur Verfügung. In der verwendeten Kamera ist die Farb-Interpolation abgeschaltet. Y800 ist ein Monochromformat mit 8 Bits. Jedes Pixel wird durch ein Byte dargestellt. Die Ausrichtung der Pixel im Bildspeicher der Kamera verläuft von links nach rechts und von oben nach unten.

Um die gewählten Einstellungen der Parameter besser zu veranschaulichen, benötigt man für die CCD-Kamera[11] eine einfache Anwendungs-Software, um die Wirkung der manuellen oder automatischen Manipulation live zu sehen. Für diesen Zweck bietet The Imaging Source die Software IC Capture an. Mit dieser Anwendung kann man sehr komfortabel die Belichtungszeit (Exposure Time), Kontrastauflösung (Gain) und die Helligkeit (Offset) der Umgebungsbeleuchtung sowie andere Parameter vor dem Scanvorgang anpassen.

Überblick über die allgemeinen Kameradaten:

- FireWire Monochrom-Kamera
- 1/2“ Sony CCD, Progressiver Scan
- 1280 x 960 Pixel
- Bis zu 15 Bilder/s
- Trigger-Eingang und I/O
- IEEE 1394
- Protokoll: DCAM 1.31

[11] CCD = Charge-Coupled Device

Zur Hardware-unabhängigen Programmierung wird in diesem Projekt die mitgelieferte API (Application Programming Interfaces) verwendet. Windows stellt für Bild-Datenströme die API DirectX zur Verfügung. Zur Vereinfachung des Zugriffs auf DirectX entwickelte The Imaging Source das SDK IC Imaging Control. Es bietet die benötigte C++ Class Library für das Scanprogramm. Die Anwendungen der einzelnen Funktionen zur Bildakquisition mit der FireWire-Kamera sind in Kapitel 5 beschrieben.

Um die Bilder, die beim Scannen gespeichert werden, optimal der Beleuchtung anzupassen, wird hier genauer auf die Kamera-Parameter[12] eingegangen. Zur Objektvermessung ist es nötig, die Parameter ideal einzustellen. Dadurch können für die Messdaten beeinflussende Fehlerquellen, wie zum Beispiel durch Reflexionen minimiert werden. In dem implementierten Programm wird nur die Belichtungszeit testweise festgelegt, wobei weitere Einstellungen noch nicht mit einbezogen und manuell justiert wurden.

Gain: Die Kontrastanhebung erfolgt durch die Verstärkung des CCD-Ausgangs-Signals. Bei zu hoher Einstellung kann allerdings ein verrauschtes Bild entstehen.

Exposure: Dieser wichtige Parameter wird im Programm gesetzt. Die Beeinflussung der Belichtungszeit ermöglicht, die Laserlinie optimal im Bild darzustellen. Der Parameter ist kameraspezifisch festgelegt und muss die Grenzbereiche einhalten. Aus dem Datenblatt für *DMK 41BF02 monochrome Kamera* sind die Werte von 1/10000 bis 30 s angegeben.

Offset: Mit diesem Parameter wird das Bild aufgehellt. Dies geschieht, indem der Offset dem CCD-Ausgangs-Signal hinzuaddiert wird. Dadurch werden also alle Grauwerte erhöht.

Sharpness: Dieser Mechanismus, der in der IC Capture-Anwendung vorhanden ist, kann unscharfe Bilder verbessern. Übertriebene Anwendungen führt aber zu Störungen die die Weiterverarbeitung beeinflussen können.

Die Spezifikationen der FireWire-Schnittstelle und den Einstellungen der Kamera sind im Anhang A angegeben.

[12] WhitePaper:`www.theimagingsource.com/de/resources/whitepapers/download/fwcamspecwp`

7. Verwendete Software

In diesem Kapitel sind alle Software-Tools angegeben, die zur Weiterentwicklung des 3D-Laserscanner-Projektes verwendet wurden.

Inhalt

7.1. Entwicklungsumgebung

Das ganze Projekt wurde aufbauend auf einer vorherigen Bachelorarbeit Schuhfuss (2007) mit Microsoft Visual Studio 2005 Professional Edition und dem integrierten Paket: Microsoft Visual C++ 2005 entwickelt.

7.2. Anwendungen

7.2.1. IC Capture

Als Hilfe zur Einstellung der spezifischen Kamera-Parameter wurde IC Capture verwendet. Das Justieren der Parameter an die Umgebungsbeleuchtung kann über die GUI automatisch oder manuell geschehen. Damit ist eine ausreichende Qualität bei der Bildakquisition gewährleistet.

7.2.2. ps90-Application

Die mitgelieferten Applikation zur Positioniersteuerung PS 90 (im Verzeichnis: .../OWI-Soft/ps90/win32/Application/ps90.exe) ist sehr hilfreich bei entstehenden Problemen mit dem Schrittmotor. Beim erneuten Setzen der Tischparameter und der Achseninitialisierung kann man unabhängig vom selbst entwickelten Anwenderprogramm eine Referenzfahrt ausführen. Vor allem eine winkelgenaue Ausrichtung der Kalibriervorrichtung zur Laserlinie vereinfacht mit dieser Anwendung die Montage auf dem Drehteller.

7.3. Softwarebibliotheken und APIs

7.3.1. LTI-Lib Computer Vision Library

Die LTI-Lib ist eine in C++ programmierte Bibliothek, die häufig bei Bildverarbeitungsprozessen verwendet wird. Sie ist im Fachbereich der technischen Informatik an der Universität RWTH Aachen entwicklet worden. Mehrere Anpassungen sind erforderlich um die Bilbliothek in der Entwicklungsumgebung Visual Studio 2005 fehlerfrei einzubinden. Eine neuere Version ist in der Entwicklung, steht aber bis dato nicht als Distribution zu Verfügung. Da die Kalibrierung und Objektvermessung hauptsächlich mit LTI-Lib erstellt wurde, ist es wichtig das komplette Include-Verzeichnis in das Projekt mit einzubinden. Darin befinden sich alle header-Dateien, also die Schnittstellenbeschreibungen, der Library.

Referenz zur LTI-Lib-Dokumentation unter *http://ltilib.sourceforge.net*.

7.3.2. Boost C++ Library Version 1.34.0

Boost ist eine freie C++-Bibliothek bestehend aus einer Vielzahl von portablen Unterbibliotheken. In diesem Projekt wird Boost zum Parsen von Kommandozeilen und zur Textformatierung verwendet. Die benötigten Kommandozeilenparameter zum Einbinden oder Erstellen von externen Dateien werden durch Trennzeichen vom Kommando voneinander getrennt und folgen immer auf das Kommando. Sie gliedern sich in Optionen und Argumente.

- Argumente sind Namen von Dateien, Verzeichnissen oder ähnlichen Objekten, auf die das Kommando angewendet werden soll.
- Optionen werden mit einem oder zwei Bindestrichen eingeleitet. Sie modifizieren die Wirkung eines Kommandos (siehe Kapitel 4).

Referenz zur Boost-Dokumentation unter *http://www.boost.org/doc/libs*.

7.3.3. IC Imaging Control C++ Class Library

Die folgenden Liste zeigt die wesentlichen DLL-Bibliotheken, die notwendig sind, damit das IC Imaging Control API im C++-Projekt richtig eingesetzt werden kann. Wenn beim Erstellen des Programms weitere Bibliotheken benötigt werden z.B. nach einer Weiterentwicklung dieser Arbeit, können die erforderlichen Dateien aus dem Quellverzeichnis der API in das Projektverzeichnis kopiert werden.

- Runtime DLLs for Visual Studio™ 2005 generated applications:

  ```
  TIS_UDSHL07_vc8.dll
  TIS_DShowLib07_vc71.dll
  ICFilterContainer.dll
  ```

  Diese dynamischen Bibliotheken müssen im Verzeichnis der selbst erstellten .exe-Datei angelegt werden.

- System:

  ```
  Msvcp60.dll
  Mfc71.dll
  Msvcp71.dll
  Msvcr71.dll
  ```

  Zielverzeichnis: Windows/System32

Referenz der Klassenbibliotheken unter:

http://www.imagingcontrol.com/support/documentation/class/ref.htm.

8. Zusammenfassung

Inhalt

8.1. Stand der Arbeit

Das erweiterte System ist durch die zusätzlich entwickelten Eigenschaften in der Lage, einen kompletten Scanvorgang zu vollziehen. Dies beinhaltet nach einer Kalibrierung eine variable automatische Ansteuerung eines Schrittmotors zur rotativen Vorschubbewegung, und einer Bildaufnahme. Durch die modulare Einbindung vorhandener Mess-Komponenten können die akquirierten Bilder vermessen und die 2D-Bildinformationen exportiert werden.

8.2. Genauigkeit und Schnelligkeit

Zu Beachten ist die Zeit, die der Schrittmotor benötigt um die gewählte Position anzufahren. Die Geschwindigkeit ist abhängig von den Motoreigenschaften. Veränderungen von Parametern zur Beschleunigung des Scanvorgangs müssen im Programm mit einbezogen werden (siehe Kapitel 6: Hardwaredokumentation). Darauf muss dann die Bildakquisation genau abgestimmt werden, damit keine verfälschten Messdaten entstehen. Ebenso können Optimierungen der Mess-Algorithmen zur schnelleren Ermittlung der Realkoordinaten von Nutzen sein.

8.3. Erweiterbarkeit

Die Arbeit wurde so konzipiert, dass durch den modularen Aufbau der Komponenten nachfolgenden Projekten eine effektive Weiterentwicklung ermöglicht wird.

8.4. Fazit

Das Projekt zur Erstellung eines 3D-Laserscanners ist eine komplexe Herausforderung, wobei viele Problemstellungen zu beachten sind. Es werden die verschiedenen Zusammenhänge von mathematischen Aspekten in der Bildverarbeitung betrachtet. Dies beginnt mit der Ermittlung der Paramter des mathematischen Kameramodells, die notwendig sind um eine exakte Kalibrierung vorzunehmen. Die eigentliche Objektvermessung verlangt Kenntnisse von Triangulationsverfahren zur Koordinatentransformation für die anschliessende 3D-Rekonstruktion. Aber auch die Konfiguration der einzelnen Hardwarekomponenten zum Ansteuern der Drehvorrichtung und die Verzahnung mit der Bildaufnahme bildet eine komplexe Einheit.

A. Technische Daten

A.1. Universal-Positioniersteuerung PS90

Die PS 90 hat eine USB 2.0-Schnittstelle.Der Anschluss befindet sich auf der Gerätevorderseite. Über einen virtuellen COM-Port-Treiber wird die USB-Schnittstelle als eine virtuelle RS 232-Schnittstelle angesprochen. Alternativ zur USB-Schnittstelle kann die Steuerung über die RS 232 mit einem PC kommunizieren.

Typenbezeichnung : **PS 90**

Technische Daten :

Versorgungsspannung	100....240 V
Stromaufnahme	max.10 A
Ausgangsleistung	240/480 W
Anzahl der Achsen	max. 9
TTL-Eingänge	0-5 V
TTl-Ausgänge	0-5/10 V/mA

Tabelle A.1.: Technische Merkmale des Steuerungsmoduls PS 90

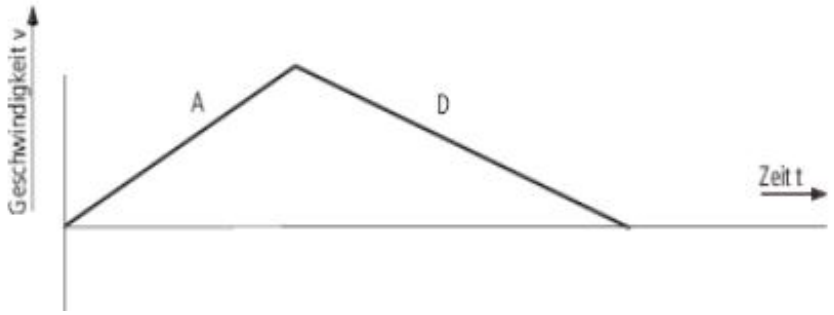


Abbildung A.1.: Trapez-Kurven-Profil-Steuerung

Der Schrittmotor wird über die Steuerung mit einem Trapez-Kurven-Profil angesteuert.
(A= Beschleunigung , D = Verzögerung)

A.2. Drehmesstisch DMT 65 + SM240

Die hardwareseitigen Einstellungen des Motors sind in der selbst erstellten C++ Header-Datei CONFIG.H abgelegt sowie testweise die Ablaufsequenzen (z.B. Schrittweite der Drehachse) konfiguriert.

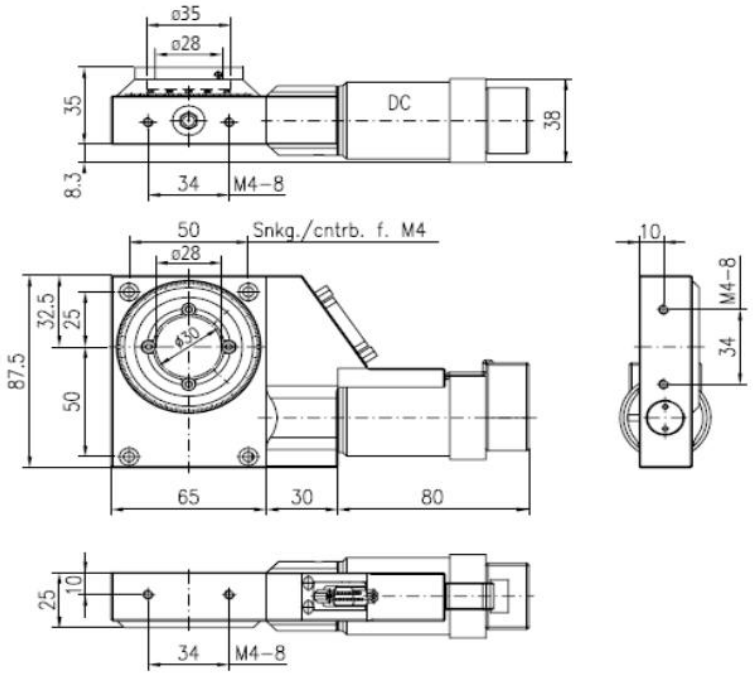


Abbildung A.2.: Drehmesstisch-Abmessungen

Typenbezeichnung :**DMT65 + SM240**

Technische Daten :

Untersetzung	180:1
Rotationswinkel	unbegrenzt
Auflösung	< 1 Winkelminute
Geschwindigkeit	max. 65 °/s
Rundlauf (unbelastet)	< 10 μm
Planlauf	< 15 μm
Schritte/Impulse pro Motorumdr.	200-2000

Tabelle A.2.: Technische Merkmale des Drehmesstellers

A.3. FireWire Monochrome Camera

Die Kamera wird vom PC über den FireWire-Bus eingestellt, der überwiegend für den schnellen Datenaustausch zwischen Computer und Multimedia- oder anderen Peripheriegeräten verwendet wird. Die Kommunikation basiert auf dem Standard-Protokoll DCAM. Es wurde von der Arbeitsgruppe IIDC (http://www.1394ta.org/) der 1394 Trade Association definiert. DCAM definiert die Struktur des Bild-Datenstroms sowie die Parameterübergabe der Kamera (Helligkeit, Belichtungszeit, Weissabgleich, usw.).

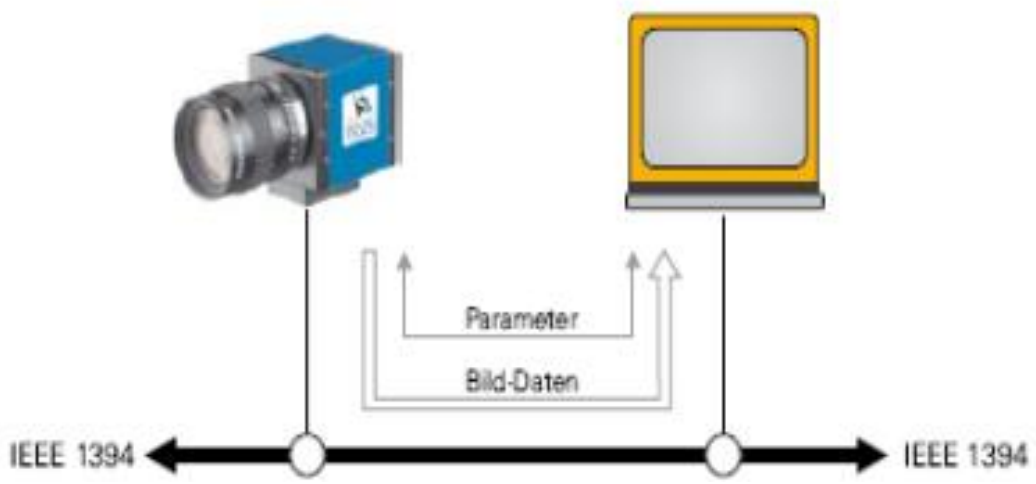


Abbildung A.3.: Datenaustausch zwischen PC - Kamera

Typenbezeichnung :**DMK 41BF02**

Allgemeine Einstellungen:

Video-Formate @ Bildrate	1280 x 960 Y800 @ 15, 7.5, 3.75 fps
Empfindlichkeit	0.5 lx bei 1/7.5s, Verstärkung 20 dB
Dynamikbereich	ADC: 10 bit, Ausgang: 8 bit
Versorgungs-Spannung	8 bis 30 V
Stromaufnahme	ca. 200 mA bei 12 V
Anschlussart	FireWire (IEEE 1394)

Tabelle A.3.: Allgemeine Einstellungen der Kamera

Optische Schnittstelle:

Typ	Progressive Scan
Format	1/2 "
Auflösung	H: 1360, V: 1024
Pixel-Größe	H: 4.65 μm, V: 4.65 μm
Objektivanschluss	C/CS Mount

Tabelle A.4.: Optische Merkmale der Kamera

Einstellungen:

Belichtungszeit	1/10000 bis 30 s
Verstärkung	0 bis 36 dB
Offset	0 bis 511
Sättigung	0 bis 200 % (nur manuell)
Weißabgleich	-2 dB bis +6 dB

Tabelle A.5.: Kamera-Einstellungsmöglichkeiten (manuell/auto)

A.4. Z-Laser TTL-Kompakt-Modul

Beim eingesetzten Laser handelt es sich um ein Produkt der Firma Z-Laser.

Abbildung A.4.: Laser-Abmessungen

Typenbezeichnung :**Z-Laser Z10V-TTL-SLG30**
Technische Daten:

Gaußförmige Linie
Öffnungswinkel 30°
sehr geringe Projektionsabweichung bei der Fokussierung
Wellenlänge 635 nm für Positionierungsanwendung
Ausgangsleistung : 1-40 mW
Betriebsspannung : 5 VDC

Tabelle A.6.: Technische Merkmale des Lasers

B. Quelltexte

B.1. Konfigurationsdatei config.h

```
/**
Diese Konfigurationsdatei legt die Einstellungen des Ansteuerungsprogramms
für den Schrittmotors DMT65+SM240 der Firma OWIS GmbH fest.
Die hardwareseitigen Einstellungen des Motors werden hier abgelegt
sowie die Ablaufsequenzen (z.B. Schrittweite der Drehachse) konfiguriert.
*/

#ifndef __CONFIG_H
#define __CONFIG_H

/** Connection Defaults
    int     nController         // Controller Type
    int     nInt;               //  0 - SERIAL, 1 - NET
    int     nCom;               //  1 - COM1
    int     nBaud;              //  9600
    int     nHandshake;         //  0
    int     nParity;            //  0
    int     nDatabits;          //  8
    int     nStopbits;          //  0 - CR
*/

//Parameter für einen Verbindundungsaufbau mit PS90
#define _CONTROLLER 3
#define _INT        0
#define _PORT       3
#define _BAUD_RATE  115200
#define _HANDSHAKE  0
#define _PARITY     0
#define _DATABITS   8
#define _STOPBITS   0


/** Motor Initialisation
    0=DC-Brush servo motor
    1=Step motor Open Loop Dir/Clock
```

```
36      2=Step motor Open Loop PMD
37      3=Step motor Closed Loop
38      4=BLDC servo motor
39  */
40  #define _MOTORTYPE   2
41
42  /** Position-modus (Position)    int nMode
43          0 = Trapez-Modus (rotativ)
44          1 = S-Kurven-Profil
45      Zielmodus (Target)
46          0 = relativ
47          1 = absolut
48  */
49  #define _POSMODE     0
50  #define _TARGETMODE  1
51
52  /** Untersetzungsverhältnis / Schrittweite
53      SetTarget(int Axis, long lValue)
54      GoTarget(Axis)
55      SetTargetEx(1,1) --> GoTarget = 20 deg
56      SetTargetEx(1,18)--> GoTarget = 360deg
57  */
58  #define _STEP_01     0.05
59  #define _STEP_10     0.5
60  #define _STEP_20     1
61  #define _STEP_45     2.25
62  #define _STEP_90     4.5
63
64  /**
65      Im Programm werden die Schrittweite
66      und die Unterbrechungszeit rechnerisch ermittelt.
67      Der Schrittmotor benötigt bei der
68      angegebenen Konfiguration
69      pro 1°  150 ms, um die Position
70      anzufahren. Hier sind testweise Wartezeiten
71      angelegt.
72  */
73  #define _SLEEP_01    150
74  #define _SLEEP_10    1500
75  #define _SLEEP_20    3000
76  #define _SLEEP_30    4500
77  #define _SLEEP_45    6750
78  #define _SLEEP_90    13500
79
80  #endif // _CONFIG_H_
```

B.2. Ansteuerung des Drehtellers

Das Starten des Scanprogramms erfolgt in der Datei **scanning.cpp**. Es wird eine Verbindung mit der Positioniersteuerung hergestellt und der Schrittmotor initialisiert. Die Funktionen sind in Kapitel 5 erläutert.

```

/**
    In dieser Datei ist die main-Funktion zum Scannen von Objekten abgelegt.
    Zur Ausführung der Kalibrierung und einer Einzel-Objektvermessung muss
    die auskommentierte main-Datei der vorherigen Arbeit verwendet werden.

    Dies ist bewusst so angelegt, da  zum Scannen noch kein Konsolenaufruf
    vorhanden ist. Darin sollen in den kommenden Arbeiten die Schrittweite,
    Winkelangabe und andere wichtige Parameter mit übergeben werden,
    evtl. zur Ermittlung der Rotationsachse,
    die in diesem Projekt noch nicht berücksichtigt wurden.
*/

/**
    Eine statische Variable wird erzeugt, um eine Mehrfacheinbindung zu
    vermeiden.Die Variable MEMBUFS definiert die Anzahl der Bilder, die
    aufgenommen werden sollen. Daraufhin wird eine MemBufferCollection mit
    der MEMBUFS-Größe initialisiert, in welche die Bilder gepuffert werden.
    Mit diesem Variablenwert, der nur einmal im Programm definiert wird,
    werden die Schrittweite und die Sleep-Dauer angepasst.
    Dies bedeutet z.B. mit dem angegebenen Wert von 8 Bildern, dass
    bei einer 360°-Drehung alle 45° ein Bild erstellt wird
    und eine Sleep-Zeit von 6750 ms angegeben werden muss.
*/
#define MEMBUFS 8

/*  Umrechnung der Schrittweite von Winkelgrad in Vollschritte*/
double stepsize = (360/MEMBUFS)*0.05;

/* Warten bis Motor sich in richtiger Position befindet */
double sleep    = (360/MEMBUFS)*150;


/*************** Control-Unit + Motor Initialisierung */
CPS90Win stepmotor;

/*  Verbindungsaufbau mit der Positioniersteuerung PS90 */

```

```
39 stepmotor.Connect(
40                 _CONTROLLER,
41                 _INT,
42                 _PORT,
43                 _BAUD_RATE,
44                 _HANDSHAKE,
45                 _PARITY,
46                 _DATABITS,
47                 _STOPBITS
48             );
49 
50     //Kontroll-Abfragen
51     cout<<"IsConnected\t"<<stepmotor.IsConnected()
52         <<"\tIsAxisActive\t"<<stepmotor.IsAxisActive(1)<<endl;
53 
54 stepmotor.SetMotorType(1,_MOTORTYPE);
55 
56     //Kontroll-Abfragen
57     cout<<"GetMotorType\t"<<stepmotor.GetMotorType(1)<<endl;
58     cout<<"GetSerNumber\t"<<stepmotor.GetSerNumber()<<endl;
59 
60 /* Motor-Achse initialisieren und in bestromten Zustand versetzen*/
61 stepmotor.MotorInit(1);
62 stepmotor.MotorOn(1);
63 
64 /*  Geschwindigkeitsmodus   */
65 stepmotor.SetPosMode(1,_POSMODE);
66     cout<<"\nPosMode␣Trapez=0/S-Kurve=1␣␣:\t"
67         <<stepmotor.GetPosMode(1)<<endl;
68 
69 /* Zielmodus: relativ bzw. absolut  */
70 stepmotor.SetTargetMode(1,_TARGETMODE);
71     cout<<"Zielmodus␣rel=0␣/␣abs=1␣␣␣␣␣:\t"
72         <<stepmotor.GetTargetMode(1)<<endl;
```

B.3. Kamera-Initialisierung

In diesem Abschnitt findet ein Verbindungsaufbau mit der Kamera statt. Außerdem wird eine MemBufferCollection mit der angegebenen Anzahl der erstellenden Bilder angelegt.

```
1  if( !DShowLib::InitLibrary( "ISB3200016679" ) )
2  {
3      fprintf( stderr, "The library could not be initialized ");
4      fprintf( stderr, "(invalid license key?).\n");
5      exit( 1 );
6  }
7  cout<<"\nInitLibrary :\n"<<endl;
8
9  Grabber grabber;
10 if( !setupDeviceFromFile( grabber ) )
11 {
12     return -1;
13 }
14
15 DShowLib::tsPropertyRange TPR;
16 TPR = grabber.getPropertyRange( CameraControl_Exposure);
17
18 printf("\n\nExposureMin=%d,    ExposureMax=%d\n",
19                  TPR.min, TPR.max); // -11...-5
20
21 /* in dieser Reihenfolge (Belichtungszeit)*/
22 grabber.setProperty( CameraControl_Exposure, false );  // auto = off
23 grabber.setProperty( CameraControl_Exposure, (long) -9 );
24
25 grabber.setOverlayBitmapPathPosition( ePP_NONE );
26
27 /* Set the image buffer format to eY800.*/
28 /* Let the sink create a matching MemBufferCollection with 1 buffer.*/
29 tFrameHandlerSinkPtr pSink = FrameHandlerSink::create( eY8, 1 );
30 /* We use snap mode.*/
31 pSink->setSnapMode( true );
32 /* Set the sink.*/
33 grabber.setSinkType( pSink );
34 /* Prepare the live mode, to get the output size if the sink.*/
35 if( !grabber.prepareLive( true ) )  // Livekamera an=true
36 {
37     std::cerr << "Could not render the VideoFormat into a eY800 sink.";
38     return -1;
39 }
```

```
40 /* Retrieve the output type and dimension of the handler sink.*/
41 FrameTypeInfo info;
42 pSink->getOutputFrameType( info );
43
44
45 /* Allocate  image buffers of the above calculate buffer size.*/
46 BYTE* pBuf[MEMBUFS];
47 for( int i = 0; i < MEMBUFS; ++i )
48 {
49     pBuf[i] = new BYTE[info.buffersize];
50 }
```

B.4. Laden der Transformationsmatrix

```
1
2 /* Laden der Transformationsmatrix in einer angegebenem Datei */
3
4 std::ifstream matrix_ifs;
5 matrix_ifs.open(matrix_fname);
6
7 if(!matrix_ifs) {
8     cerr << "Error: specified matrix-file could not be opened." << endl;
9     getchar(); exit(1);
10 }
11
12 lti::dmatrix trMatrix;
13 lti::lispStreamHandler lsh;
14
15 lsh.use(matrix_ifs);
16 trMatrix.read(lsh, true);
```

B.5. Scannen einer Szene

```
/*********************************************************/
/*************** Start to Scan  ************************/
/*********************************************************/
/**
Zähler zur Angabe der Bildnummer und zur Berechnung der übergebenen
Winkelpositionen in der das Bild erstellt wurde.
*/
int picNr = 0;
int angle = 0;

/*  Umrechnung der Schrittweite von Grad in Vollschritte/Umdrehung*/
double stepsize = (360/MEMBUFS)*0.05;

/* Warten bis Motor sich in richtiger Position befindet */
double sleep    = (360/MEMBUFS)*150;

if(stepmotor.IsConnected()){
    cout<<"\nGetConnectInfo\t"<<stepmotor.GetConnectInfo()<<endl;
    double pos =0;

    /*Starte den Live-Mode für fast snapping, false für
    zeitl. getrennte Einzelbilder
    */
    grabber.startLive( true );

    for(int i =0 ; i<MEMBUFS ; i++){
        /*   Aktuelle Position der Achse auf Null setzen*/
        stepmotor.SetPositionEx(1,0);
        /**
        Gibt die richtige Gradzahl auf der Konsole aus
        (Abhängig von Untersetzung und Vollschritt pro Umdrehung)
        */
        cout<<"\tSet Position : "<<stepsize*20  <<" degrees"<<endl;
        cout<<"\tSleep time   : "<<  sleep   <<" milisec"<<endl;
        cout<<"\tstepsize     : "<<stepsize<<" calculated"<<endl;

        /* Die Bilder werden in die MemBufferCollection kopiert*/
        pSink->snapImages(1);

        cout<<"\n\t\tsnapImages Nr : "<<i<<"\n"<<endl;

        /*  stepmotor.MoveEx(1,0.5,true);           //0.5=10° ; 1=20°*/
        stepmotor.MoveEx(1,stepsize,true);
        cout<<"\tMoveEx (mm/deg)"<<endl;
```

```
45
46          /*  Warten bis Drehtisch sich in richtiger Position befindet*/
47          Sleep(sleep);
48          /*  Die Möglichkeit von Hand den Scannvorgang fortzufahren*/
49          //cout<<"\nPress next position\n"<<endl;
50          //getchar();
51
52          /*    Position hochzählen     */
53          pos = pos + stepsize;
54          cout<<"position after move: " << pos*20<<" degree"<<endl;
55      }//for
```

B.6. Bilder einlesen zur Vermessung

Zum Laden der einzelnen Bilder wird die LTI-Lib Klasse *lti::loadImageList* verwendet. Diese Klasse erlaubt ein bequemes Laden einer Liste von Bildern aus einem angegebenen Verzeichnis. Um über verschiedene Betriebssystemen gleichbleibend zu sein, werden die Dateinamen alphabetisch sortiert. Diese nützliche Eigenschaft gewährleistet das Laden der Bilder in richtiger Reihenfolge.

```
1  /*LTS Konstruktor mit den Parametern  aus der Klassendefinition measure.h
2  zur Objektvermessung*/
3  lts::Lts::parameters ltsParams;
4  Lts l(ltsParams);
5
6  loadImageList    list;
7  std::ofstream output;
8
9  /*  Das Verzeichnis wählen in der die zu vermessenden Bilder sich befinden*/
10 list.use("c:\\temp\\scanner",loadImageList::parameters::DirName);
11
12 //Abfrage
13 while (list.hasNext()) {
14
15     /*Berechnung der Winkelpositionen, die in die output-
16     Datei mit übergeben werden*/
17     angle = angle +  (360/MEMBUFS);
18     picNr++;
19     /*
20     Anlegen eines Vectors vom Typ: Point-Set
21     in dem die Messdaten abgelegt werden
22     */
23     std::vector<lti::fpoint> results;
24
25     if(!output)
```

```
26                  cerr << "Error␣opening␣file␣"
27                              << output_fname << "␣for␣writing..."
28                              << endl;
29      else {
30          /*hole das nächste Bild */
31          list.apply(flipImg);
32
33          /*Als Parameter zur Bildvermessung wird die
34          Transformationsmatrix mit übergeben*/
35          l.apply(flipImg, trMatrix);
36
37          /*Übergebe die Messpaare in eine Vektorliste*/
38          l.getPointsList(results);
39
40          /*Output-Datei öffnen*/
41          output.open(output_fname.c_str(), ios::out|ios::app);
42
43          /*Textseparator */
44          output << "#␣result␣file.␣contained␣dataformat␣'x␣y'␣image:␣"
45                      <<picNr
46                      <<endl;
47          /*
48          Schreiben in die Output-Datei im Format: 'x y'
49          Mit einer zusätzlichen Winkelangabe oder nur die 2D-Daten
50          */
51          for(size_t i=0; i<results.size(); i++) {
52              output <<
53                  (boost::format("%f␣%f␣%f")
54                                          % results[i].x
55                                          % results[i].y
56                                          % angle)
57                      << endl;
58              //output <<
59              //  (boost::format("%f %f")
60              //                          % results[i].x
61              //                          % results[i].y %)
62              //              << endl;
63          }
64
65          //output << "# end of data\n" << endl;
66          output.close();
67          l.clearPointList();
68      }//else
69  }//while
```

C. Inhalt der CD

Verzeichnis	Inhalt
pdf/	diese Arbeit
quelltext/	enthält das vollständige Projekt mit allen Unterverzeichnissen.
ergebnisse/	Enthält Testbilder von der Kalibrierung, Objektvermessung und Scanvorgang mit Messdaten

Tabelle C.1.: Vezeichnisse innerhalb der CD

Tabellenverzeichnis

Literaturverzeichnis

[Alvarado u. a. 2007] ALVARADO, Pablo ; DOERFLER, Peter ; CANZLER, Ulrich: *LTI Image Processing Library: Developers Guide*. Homepage des LTI-Lib-Projektes der RWTH Aachen. 2007. – URL `http://ltilib.sourceforge.net`

[boost.org 2007] BOOST.ORG: *Boost Libraries and Documentation*. Dokumentationsseite des „Boost C++ Libraries“-Projektes. 2007. – URL `http://www.boost.org/libs/libraries.html`

[Burger 2006] BURGER, M.J.: *Digitale Bildverarbeitung*. Springer-Verlag Berlin Heidelberg, 2006. – ISBN 3-540-30940-3

[Hartley und Zisserman 2004] HARTLEY, R. I. ; ZISSERMAN, A.: *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004

[Jürgens 1995] JÜRGENS, Manuela: *LaTeX-Fortgeschrittene Anwendung*. Fernuniversitaet Hagen. 1995

[Meisel 2005] MEISEL, Andreas: *Robot Vision, Kapitel 3.3.6: Gradientenfilter*. Vorlesungsskript von Prof. Dr.-Ing. Andreas Meisel, Seminar Robot Vision. 2005

[Meisel 2006] MEISEL, Andreas: *Das 3D-Kameramodell der Technischen Informatik der HAW*. Ausarbeitung von Prof. Dr.-Ing. Andreas Meisel an der Hochschule für Angewandte Wissenschaften Hamburg. 2006

[Meisel 2007] MEISEL, Andreas: *3D-Bildverarbeitung, S.36*. Prof.Dr.-Ing. Andreas Meisel, HAW-Hamburg. 2007

[OWIS GmbH] OWIS GMBH, Staufen: *ps90 user info*. – URL `http://www.owis-staufen.de`

[Papula 1994] PAPULA, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler, Band 2*. Wiesbaden : Vieweg Verlag, Fachbücher der Technik, 1994. – ISBN 3-528-64237-8

[Paulus 2001] PAULUS, Dietrich: *Aktives Bildverstehen*. Osnabrück : Osnabrück: Der Andere Verlag, 2001. – ISBN 3-935316-87-9

[Reolon 2006] REOLON, Pascal: *Technische Realisierung eines 3D-Scanners auf Basis der Lichtschnitt-Technik*, University of Zürich, Department of Informatics, Visualizaton and MultiMedia Lab, Diplomarbeit, 2006

[Schuhfuss 2007] SCHUHFUSS, Martin: *Realisierung eines Laser-Triangulationssensors zur 3D-Objektvermessung*, HAW-Hamburg, Diplomarbeit, 2007

[Steinbrecher 1993] STEINBRECHER, Rainer: *Bildverarbeitung in der Praxis*. Oldenbourg Verlag, 1993. – ISBN 3-486-22372-0

[Tobias Erbsland 2008] TOBIAS ERBSLAND, Andreas N.: *LaTeX: Diplomarbeit mit Latex*. Erschienen auf der Webseite. 2008. – URL `http://www.dml.drzoom.ch/diplomarbeit_mit_latex_v1.10.pdf`

[Willemer 2004] WILLEMER, Arnold: *Einstieg in C++*. Galileo Press GmbH, 2004. – ISBN 3-89842-397-2

[Wolf 2006] WOLF, Jürgen: *C++ von A bis Z*. Galileo Computing, 2006. – ISBN 978-3-89842-816-3

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. August 2008 Christian Reimann