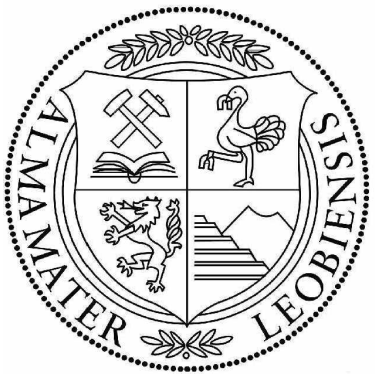
MONTANUNIVERSITÄT LEOBEN
Studienrichtung: Industrieller Umweltschutz



DIPLOMARBEIT

am Institut für Automation

zum Thema

Profilmessung von glühenden Stäben und Knüppeln

durchgeführt von

Norbert Koller

Zusammenfassung

In dieser Diplomarbeit wird eine Methode zur berührungslosen Vermessung des Querschnitts von glühenden Stäben und Knüppeln über einer Walzstraße vorgestellt. Die Temperatur des Walzgutes beträgt ca. 1000^oC, deren Geschwindigkeit bis zu $6m/s$.
Das dabei verwendete Messprinzip basiert auf dem Lichtschnittverfahren. Dieses Verfahren wird durch einen Laser, welcher auf das Walzgut eine Linie projiziert und einer CCD-Kamera, die als Sensor agiert umgesetzt. Die zur Berechnung benötigten Messalgorithmen wurden in *Matlab*® erstellt.
Vor der Messung werden die Laser- und Kameraposition durch Kalibration bestimmt. Dies geschieht mit einem speziell für diesen Zweck angefertigten Kalibrationsobjekt.
Um systematische Messfehler zu verhindern, wird ein zweiter Laser hinzugezogen. Dadurch ist es möglich, die Laufrichtung des Walzgutes zu ermitteln und den unverzerrten Querschnitt zu berechnen. Weiters wurden verschiedene Lasertypen, mit unterschiedlichen Leistungen und Wellenlängen getestet und die geeignetste Variante in einem Prototypen realisiert. Den Störeinfluss der vom Walzgut emittierten Temperaturstrahlung eliminiert ein Wellenlängenfilter vor dem Objektiv der CCD-Kamera. Der Prototyp wurde abschließend im Walzwerk der Firma Edelstahl Witten-Krefeld GmbH erfolgreich getestet.
Aufbauend auf den Erkenntnissen dieser Diplomarbeit wird ein industrietaugliches Messgerät gebaut.

Abstract

The aim of this work is the non-contact profile measurement of glowing steel rods and billets. The temperature of the slabs is approximately 1000^oC, their velocity up to $6m/s$.
The used principle of measurement is the light sectioning method. This method is a well known measurement technique for optical determination of object sections. A laser plane is projected onto the object from one direction. The emerged profile on the scene is viewed from a different direction using a ccd-camera.
All algorithms needed for calculation were generated in *Matlab*® .
Before measuring the profile, the positions of the laser and the camera are acquired by calibration. For this purpose a calibration object with known dimensions was manufactured. To avoid systematic errors the measurement is done with two lasers. By using a second laser it is possible to acquire the running direction of the measured slab and so the undistorted profile can be calculated.
Further various types of lasers with different wavelengths and powers were tested and the most suitable application was realized in a prototype. The emitted temperature radiation from the glowing slab was absorbed by an interference filter in front of the objective of the ccd-camera. Finally the prototype was tested successfully in the production line at Edelstahl Witten-Krefeld GmbH.
Due to the insights of this thesis an industrial measurement device will be constructed.

An dieser Stelle erkläre ich Eides Statt, die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Literatur angefertigt zu haben.

Leoben am 4. November 2002 (Norbert Koller)

Mein Dank gilt

**Herrn O. Univ. Prof. Dipl.-Ing. Dr. techn.
Paul O'LEARY**

für die Betreuung und Unterstützung bei all meinen Anliegen, sowie für die Bereitstellung aller dafür notwendigen Einrichtungen des Institutes.

Edelstahl Witten-Krefeld GmbH.

für die Vergabe dieser Diplomarbeit und die Bereitstellung aller notwendigen Mittel für die Verwirklichung dieser.

Weiters möchte ich mich bei

**Herrn Dipl.-Ing. Dr. mont.
Ronald Ofner**

und

beim gesamten Team des Instituts für Automation

für die Unterstützung und der spontanen Hilfsbereitschaft bei all meinen Anliegen bedanken.

Auch möchte ich mich an dieser Stelle bei meinen Eltern bedanken,
die mir dieses Studium hier in Leoben ermöglicht haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Aufgabenstellung

Das Ziel dieser Arbeit ist die berührungslose Querschnittvermessung von glühenden Stäben und Knüppeln, nach dem Walzvorgang, bei der Firma Edelstahl Witten-Krefeld GmbH.

Die Messung sollte aus Sicherheitsgründen und damit der Bezug zur Walzeneinstellung gegeben bleibt, so nahe wie möglich am Walzgerüst durchgeführt werden. Nur aufgrund einer genauen Bestimmung des Querschnitts des Walzgutes kann die Einstellung des Walzgerüstes vorgenommen werden.

Die zur Zeit verwendete Methode der Querschnittvermessung findet im Labor statt. Dabei wird ein kurzes Stück von dem Walzgut abgeschnitten und auf Umgebungstemperatur abgekühlt. Im Anschluss wird der Querschnitt per Hand vermessen. Dies ist eine sehr aufwändige Methode und es benötigt einige Zeit, bis die Ergebnisse vorliegen.

In dieser Arbeit soll das Prinzip des Lichtschnittverfahrens umgesetzt und daraus eine berührungsfreie, optische Messmethode entwickelt werden, welche sofort die Warmmaße des Walzgutes misst. Dabei sollte der absolute Messfehler weniger als 0.5% betragen. Die Aufnahme der Messdaten erfolgt mit einer schwarzweiß CCD-Kamera (Charged Coupled Device). Zu deren Auswertung am Computer soll ein einfach zu bedienender Kalibrations- und Messalgorithmus in *Matlab*® erstellt werden.

Weiters sollen die, für diese Messmethode erforderlichen Laser, in Kombination mit Interferenzfilter getestet werden, um anschließend die bestmögliche Kombination in einem Prototypen zu verwirklichen. Dieser soll zuerst im Labor und anschließend im Walzwerk vor Ort getestet werden.

Das vorrangige Ziel bei der Messung im Walzwerk ist nicht die Industrietauglichkeit des Prototypen, sondern die Prüfung, ob dieses Verfahren bei den dort herrschenden Umweltbedingungen anwendbar ist.

1.2 Didaktische Ziele

- Erwerb der Fähigkeit, ein Projekt selbstständig durchzuführen
- Erlernen von neuen mathematischen Rechenverfahren
- Umgang mit optischen Geräten wie Laser und Kamera
- Erlernen einer neuer Programmiersprache zur Bildverarbeitung - *Matlab*®
- Kritisches Hinterfragen von Messergebnissen
- Verfassen eines umfangreichen Berichtes über das Projekt

Kapitel 2

Problemstellung

Bei dieser optischen Messmethode wird mit Hilfe einer digitalen Kamera ein digitales Bild vom zu vermessenden Objekt aufgenommen und am Computer ausgewertet. Die Kamera liefert ein zweidimensionales Bild (Länge und Breite). Da das zu vermessende Objekt aber drei Dimensionen besitzt (Länge, Breite und Höhe), geht eine Dimension verloren. Je nach Position und Winkel der Kamera zum Objekt, ist das Bild eine Projektion dieser drei Dimensionen. Diese "verlorengegangene" dritte Dimension ist also noch im Bild enthalten, muss jedoch mit einem Trick herausgelesen werden. Um die Höheninformation aus dem Bild herauslesen zu können, wird das Objekt mit einer Laserebene geschnitten.

Alle handelsüblichen Laser emittieren Licht im roten Wellenlängenbereich. Das in diesem Fall zu vermessende Objekt hat eine Temperatur von ca. 1000^oC und ist somit rot glühend. Um eine genaue Messung durchführen zu können, ist es unbedingt erforderlich, dass sich der Laserstrahl im Bild vom glühenden Objekt eindeutig unterscheidet.

Ein weiteres Problem ist die Vermessung der Objekte. Im Bild können Längeninformationen nur in Pixel angegeben werden. Eine Längenangabe in Meter oder Millimeter ist ohne Vergleich mit einem Objekt bekannter Größe nicht möglich. Die Größe des Objekts hängt vom Abstand der Kamera zum Objekt und vom verwendeten Objektiv ab.

Während der Inbetriebnahme bei der Firma Edelstahl Witten-Krefeld GmbH erwiesen sich die durch das Walzgut erzeugten Vibrationen am Messort als großes Problem.
Die zu vermessenden Stäbe sind ca. 6.5 Tonnen schwer und haben eine Geschwindigkeit von bis zu 6 m/s. Dadurch werden einige der aufgenommenen Messbilder verwackelt und können nicht mehr mit der geforderten Genauigkeit ausgewertet werden.

Weitere Probleme bereiten die extremen Umweltbedingungen bzw. Störeinflüsse am Messort. Durch die hohen Temperaturen müssen die Messinstrumente vor thermischer Zerstörung geschützt werden. Hohe Schmutz-, Staub- und Wasserkonzentrationen stellen eine hohe Anforderung an den Schutz und die Reinigung der optischen Teile.
Auf die letztgenannten Probleme wird nur hingewiesen, da das vorrangige Ziel dieser Arbeit die Untersuchung der Durchführung und Machbarkeit dieser Messmethode ist.

2.1 Funktionsprinzip

Das dieser optischen Messmethode zugrundeliegende Prinzip beruht auf dem Lichtschnittverfahren.

2.1.1 Lichtschnittverfahren

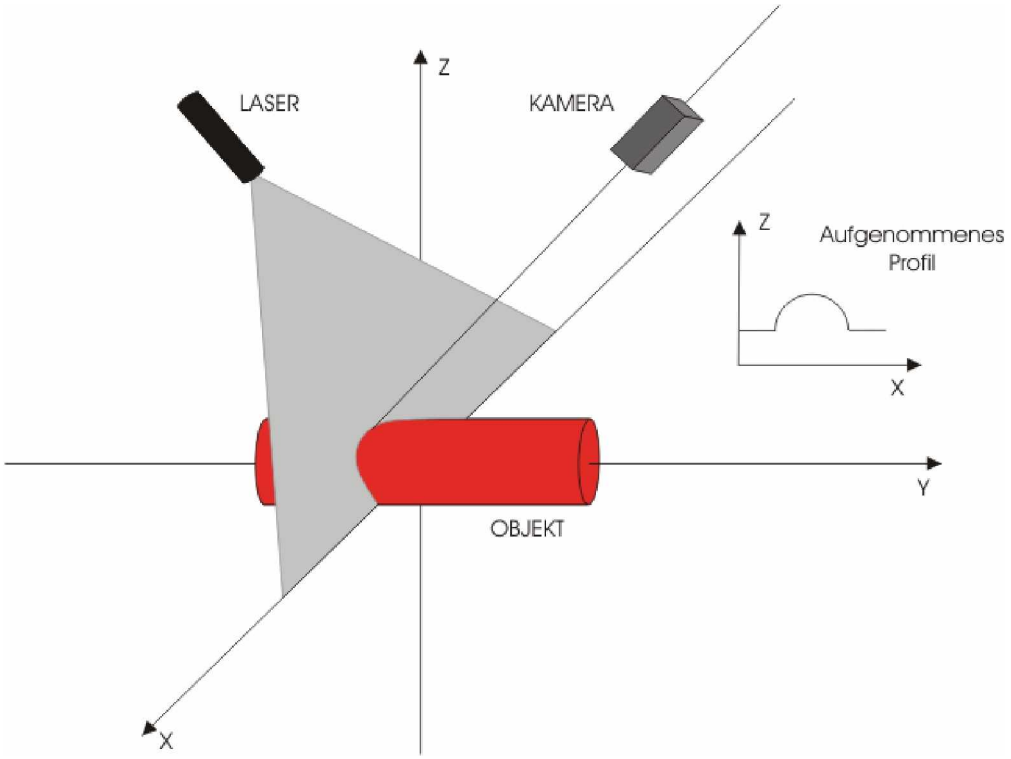


Abbildung 2.1: Prinzip des Lichtschnittverfahrens

Das Lichtschnittverfahren [1] ist eine vielfach angewandte Methode um Objekte zu vermessen. Eine Lichtebene wird auf das zu vermessende Objekt aus einer bestimmten Richtung projiziert. Diese Ebene wird meist von einem Laser erzeugt. Die Schnittlinie der Ebene mit dem Objekt wird von einer Kamera aufgenommen. Sind die Laser- und die Kameraposition bekannt, so kann allen Punkten in der Laserebene ein Positionsvektor, in einem vorher festgelegten Koordinatensystem, zugeordnet werden. Die Laser- und die Kameraposition muss vorher mit einem geeigneten Kalibrationsverfahren bestimmt werden.
Mit der in Abbildung 2.1 gezeigten Anordnung des Lasers und der Kamera ist die Messung der oberen Hälfte des Objekts möglich. Wird angenommen, dass das Objekt symmetrisch ist, so kann mit geeigneten mathematischen Modellen der gesamte Querschnitt errechnet werden. Ist dies nicht möglich, so müssen mehrere Laser und Kameras so positioniert werden, damit der gesamte Querschnitt erfasst werden kann.

Da die Laserebene das Objekt nicht parallel zur x-z-Ebene schneidet, müssen die gemessenen Punkte in die x-z-Ebene projiziert werden. Aus diesem Grund ist die Voraussetzung für die Vermessung des Querschnitts mit einem Laser, dass die Längsachse des Objekts parallel zur y-Achse des Koordinatensystems ist. Nur in diesem Fall entspricht eine Projektion der gemessenen Punkte in die x-z-Ebene des Koordinatensystems dem unverzerrten Querschnitt des Objekts. Diese Anordnung ist jedoch sehr schwierig zu realisieren und kann nicht sichergestellt werden. Wird jedoch ein zweiter Laser dem Messsystem hinzugefügt, so kann eine Projektionsrichtung errechnet werden und die Ausrichtung des Objekts wird unabhängig von der y-Achse. Die Voraussetzung für den Einsatz von zwei Lasern ist lediglich, dass der Querschnitt in dem zu vermessenden Abschnitt des Objekts konstant ist.
Wird das Objekt in Richtung seiner Längsachse bewegt, so ist es mit dieser Methode möglich, ein dreidimensionales Abbild des Objekts zu errechnen.

2.1.2 Kalibrationsverfahren

Vor dem Messvorgang müssen die Laser- und die Kameraposition mit einem geeigneten Kalibrationsverfahren ermittelt werden. Die beschriebene Methode beruht auf dem Projekt "Kalibrationsverfahren für einen Lichtschnittmesskopf [2]".
Zu diesem Zweck wurde am Institut für Automation ein Kalibrationsobjekt entworfen. Dieses Kalibrationsobjekt wurde aus Aluminium gefertigt und eloxiert. Das Objekt besteht aus drei parallelen Ebenen mit je acht Leuchtdioden. Eine weitere Leuchtdiode ist in der Mitte der vorderen Ebene. Die Abmessungen der Ebenen und die Abstände der Leuchtdioden sind genau bekannt und im Anhang B angeführt.

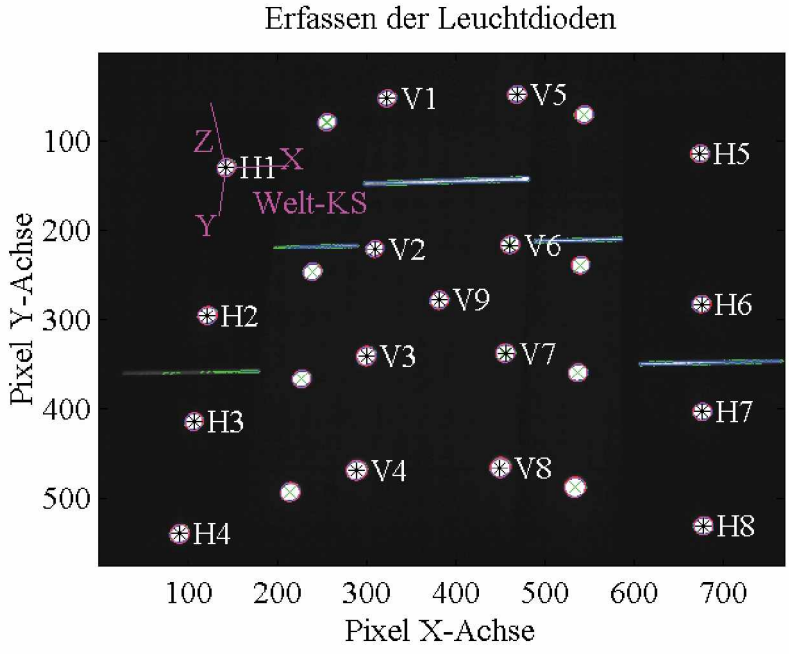


Abbildung 2.2: Prinzip des Kalibrationsverfahrens

Zur Bestimmung der Kamera- bzw. Laserposition werden mehrere Koordinatensysteme eingeführt. Das erste Koordinatensystem, das Weltkoordinatensystem, ist dreidimensional und hat seinen Ursprung im Mittelpunkt der linken oberen Leuchtdiode. Als Maßeinheit werden Millimeter verwendet. Ein weiteres Koordinatensystem, das Pixelkoordinatensystem, ist zweidimensional und hat seinen Ursprung in der linken oberen Ecke des Bildes. Als Maßeinheit werden Pixel (Bildpunkte) verwendet. Das dritte Koordinatensystem, das Laserkoordinatensystem, ist ebenfalls zweidimensional und hat seinen Ursprung im Schnittpunkt der Laserebene mit der y-Achse des Weltkoordinatensystems. Hier werden als Maßeinheit ebenfalls Millimeter verwendet. Das Laserkoordinatensystem ist so gedreht, dass es in der Laserebene liegt.
Für die Berechnung der Transformation zwischen dem Pixelkoordinatensystem und dem Weltkoordinatensystem werden mindestens je vier Punkte auf zwei verschiedenen Ebenen benötigt. Zu diesem Zweck werden die Leuchtdioden auf der hinteren (H1 bis H8 in Abbildung 2.2) und auf der vorderen Ebene (V1 bis V9) identifiziert und von deren Mittelpunkten die Pixelkoordinaten ermittelt. Die Abmessungen der Mittelpunkte im Weltkoordinatensystem sind durch die Geometrie des Kalibrationsobjekts bekannt.
Mit Hilfe dieser beiden berechneten Projektionsmatrizen, für die hintere bzw. vordere Ebene, ist es möglich, jedem beliebigen Punkt (Vektor) der jeweiligen Ebene im Pixelkoordinatensystem einen Vektor im Weltkoordinatensystem zuzuordnen. Da das Pixelkoordinatensystem zweidimensional ist, ist es noch nicht möglich, den jeweiligen Vektoren eine Höheninformation zuzuordnen. Zu diesem Zweck wird die Position der Laserebene bestimmt.

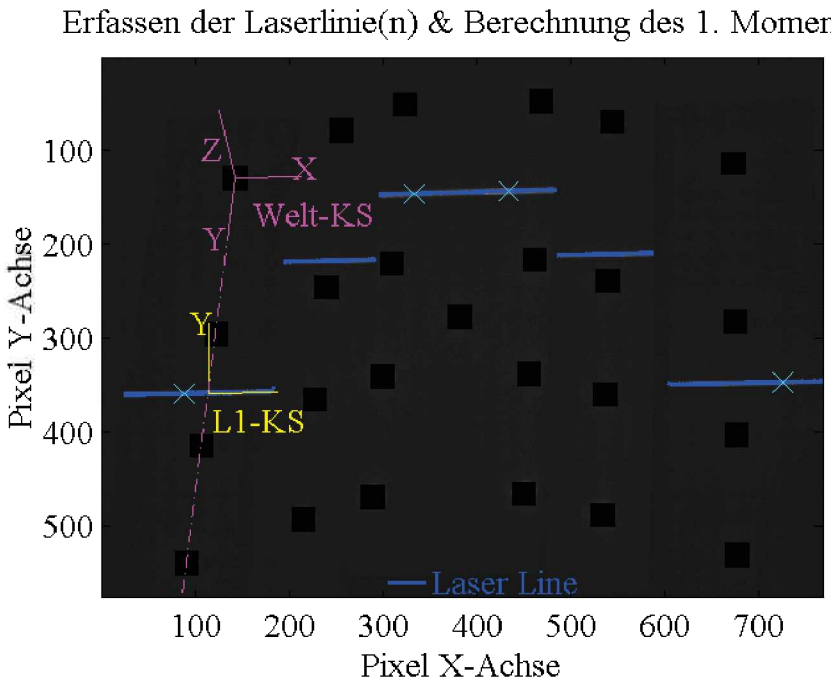


Abbildung 2.3: Bestimmung der Laserebene

Es werden die Laserpunkte auf der hinteren und der vorderen Ebene in Pixelkoordinaten ermittelt. Die Vektoren der Laserpunkte werden mit den zuvor ermittelten Projektionsmatrizen in das Weltkoordinatensystem umgerechnet. Aus der Geometrie des Kalibrationsobjekts ist die Höhe der einzelnen Laserpunkte im Weltkoordinatensystem bekannt. Da die Laserpunkte alle in einer Ebene liegen müssen, ist es möglich, die Ebenenparameter der Laserebene zu errechnen. In diese Ebene wird das Laserkoordinatensystem gedreht, und die Projektionsmatrix zwischen dem Laserkoordinatensystem und dem Weltkoordinatensystem berechnet. Anhand dieser Projektionsmatrix ist es nun möglich, jeden beliebigen Punkt auf der Laserlinie in Pixelkoordinaten als dreidimensionalen Punkt im Weltkoordinatensystem in Millimeter zu berechnen.
Nach erfolgter Kalibration wird das Kalibrationsobjekt entfernt. Die Laser- bzw. Kameraposition darf nach der Kalibration nicht mehr verändert werden.

2.2 Aufnahme der Messdaten

Die Aufnahme der Messdaten erfolgt mit einer digitalen schwarzweiß CCD-Kamera (Charged Coupled Device, Typ: Pulnix TM-6CN). Der Spektralbereich dieser Kamera liegt von $\lambda \sim 400 - 900nm$ (Abbildung 2.4). Dieser Bereich entspricht ungefähr dem für das menschliche Auge sichtbaren Bereich. Die maximale Empfindlichkeit liegt bei $\lambda \sim 550nm$.

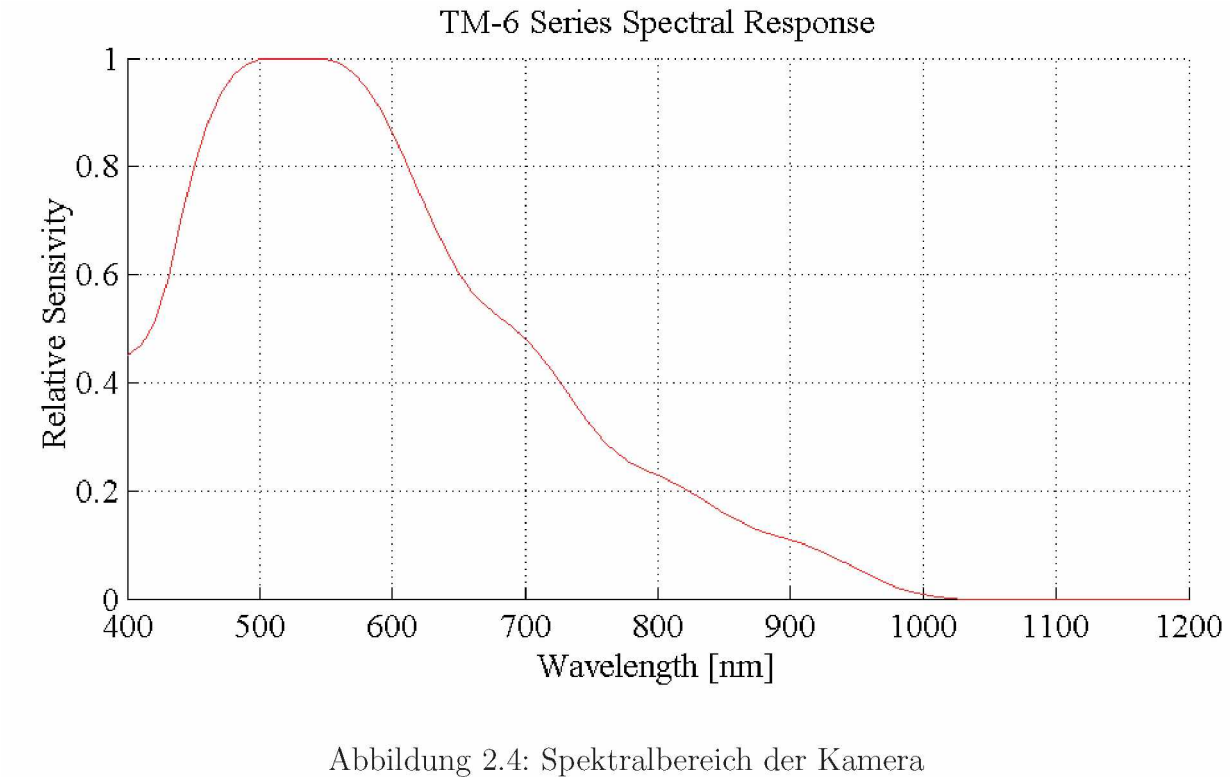


Abbildung 2.4: Spektralbereich der Kamera

Abbildung 2.4 zeigt den Spektralbereich der Digitalkamera und wurde aus deren Datenblatt entnommen.

2.2.1 Temperaturstrahlung eines festen Körpers

Ein erhitzter fester Körper sendet Strahlung von der kleinsten bis zu den größten Wellenlängen aus, wobei die Strahlungsintensität der einzelnen Wellenlängen sehr verschieden ist. Die Strahlungsintensität ist von der Temperatur, Emissionsgrad und von der Wellenlänge abhängig. Diese Abhängigkeit wird für den schwarzen Körper vom Planck'schen Strahlungsgesetz wiedergegeben. Ein schwarzer Körper ist ein solcher, der die gesamte auf ihn fallende Strahlung absorbiert. Er sendet die größte Strahlung aus, die bei dieser Temperatur als Wärmestrahlung möglich ist.

Das Planck'sche Strahlungsgesetz: [5]

$$Js_{\lambda,\theta} = \epsilon \; C_1 \; \frac{\lambda^{-5}}{e^{\frac{C_2}{\lambda\theta}} - 1} \qquad \left[\frac{W}{m^2 \; cm}\right] \tag{2.1}$$

$Js_{\lambda,\theta}$...Strahlungsintensität
ϵ...Emissionsgrad, dieser ist für den schwarzen Körper $\epsilon = 1$ und für reale Körper $0 < \epsilon < 1$
λ...Wellenlänge in cm
θ...Temperatur in K
$C_1 = 3.74 \; 10^{-8} \; \frac{W \, cm^4}{m^2}$
$C_2 = 1.44 \; cm \; K$

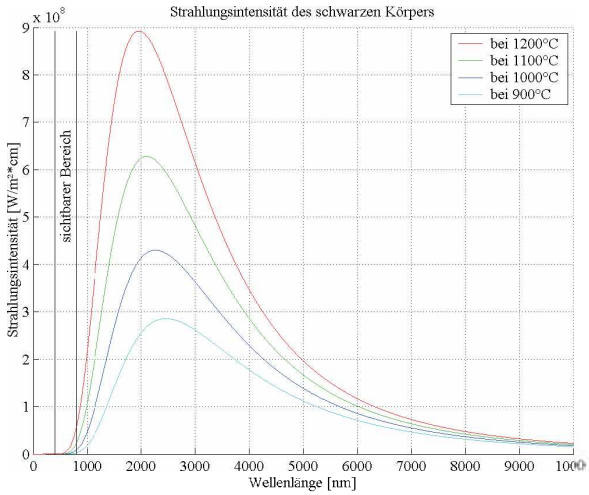


Abbildung 2.5: Strahlungsintensität des schwarzen Körpers

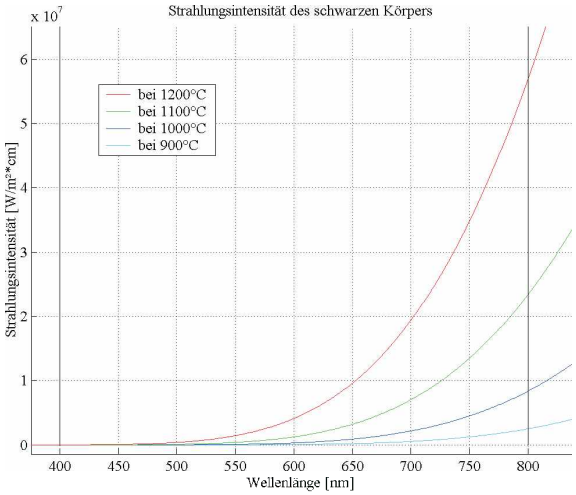


Abbildung 2.6: Ausschnitt "Sichtbarer Bereich"

Wie in Abbildung 2.5 zu sehen ist, liegt das Maximum der Intensität im infraroten Bereich ($\lambda > 1\mu m$). Aus diesem Grund wird die Temperaturstrahlung auch dunkle Wärmestrahlung genannt. Mit zunehmender Temperatur verschiebt sich das Maximum immer weiter in Richtung kleinere Wellenlängen. Die Summe der Energie $Js_{\lambda,\theta} \; d\lambda$ aller Wellenlängen ergibt die Gesamtstrahlungsenergie des strahlenden Körpers. Diese ist daher durch den Flächen-

inhalt unter der Kurve gegeben.
Der für diese Arbeit interessante Bereich liegt im sichtbaren Wellenlängenbereich ($\lambda \sim 400nm - 800nm$). Dieser Bereich wird in Abbildung 2.6 dargestellt. Die Intensität der Strahlung nimmt in diesem Bereich mit steigender Wellenlänge und Temperatur stark zu.

2.2.2 Filterung der Strahlung

Die abgestrahlte Energie des glühenden Walzgutes ist eine erhebliche Störquelle. Für eine hinreichend genaue Messung ist es unbedingt erforderlich, dass der Laserstrahl auf der Oberfläche des Walzgutes noch eindeutig zu erkennen ist. Um den Störeinfluss möglichst gering zu halten, wird ein Interferenzfilter vor dem Objektiv angebracht. Dieser sollte die Wellenlänge des Laserstrahls ungehindert durchlassen und alle anderen absorbieren. Da dies nicht möglich ist, werden immer einige wenige störende Wellenlängen durchgelassen und die gewünschte Wellenlänge des Laserstrahls abgeschwächt. Diese Abschwächung muss mit einer höheren Laserleistung kompensiert werden.
Solche Filter sind für fast alle Wellenlängen erhältlich. Eine typische Filterkennlinie für $\lambda = 685nm$ ist in Abbildung 2.7 dargestellt.

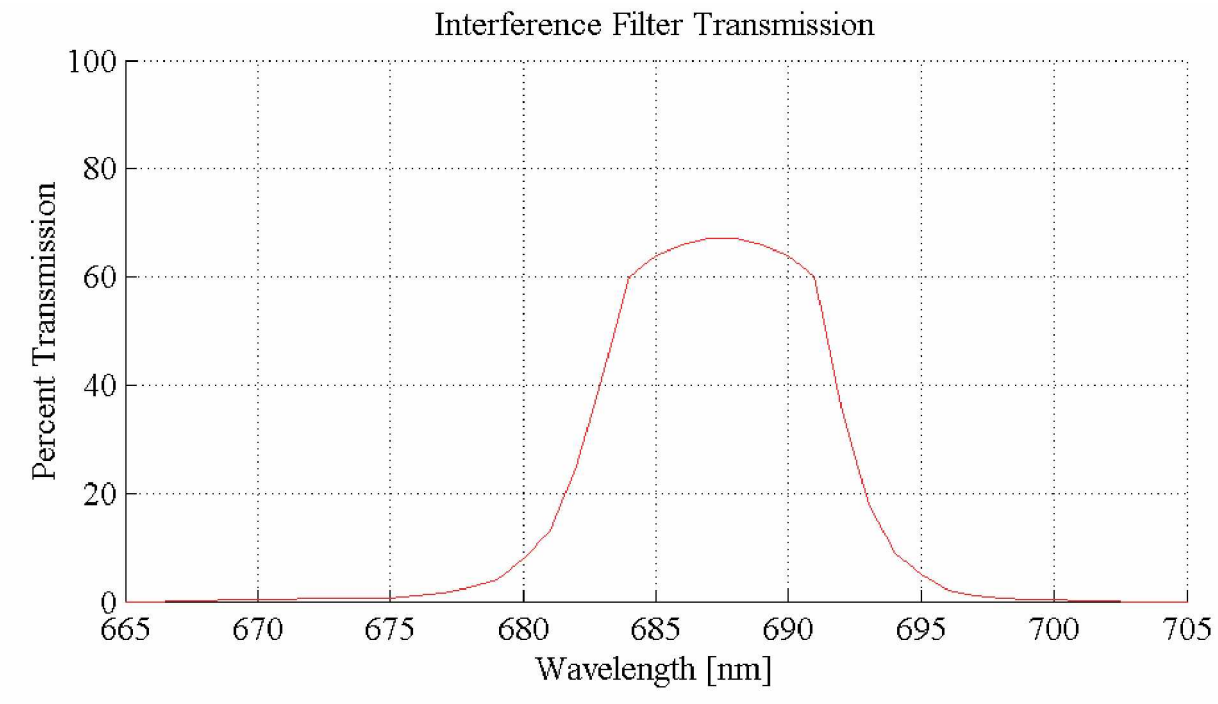


Abbildung 2.7: Kennlinie des Interferenzfilters

2.2.3 Auswahl der Wellenlänge für die Laser

Die Wahl der Wellenlänge hängt von mehreren Faktoren ab:

Damit die Laserstrahlen erfasst werden können, muss deren Wellenlänge im Spektralbereich der Kamera liegen. Die maximale Empfindlichkeit der Kamera liegt im grünen Bereich ($\lambda \sim 550nm$), wie in Abbildung 2.4 zu erkennen ist. Auch der Störeinfluss der Temperaturstrahlung nimmt mit steigender Wellenlänge zu (Abbildung 2.6).

Im Laufe dieser Arbeit wurden Messungen mit Lasern unterschiedlicher Wellenlängen durchgeführt. Es zeigte sich, dass bei Verwendung von grünen Lasern eine viel geringere Strahlleistung erforderlich ist, um die gewünschte Messgenauigkeit zu erreichen.
Grüne Laser besitzen aber nicht nur Vorteile gegenüber handelsüblichen roten (Wellenlängenbereich von $\lambda \sim 600 - 700nm$). Einige Händler bieten grüne Laser an, deren Entwicklung ist jedoch noch nicht so weit fortgeschritten, wie die der roten. Das größte Problem dieser Laser ist die Baugröße, die um ein vielfaches größer ist. Weitere Nachteile sind der viel höhere Preis und die geringere Lebensdauer (Mean time before failure).
Aus diesen Gründen wird die Messung mit zwei roten Lasern ($\lambda = 685nm$) durchgeführt. Die Nachteile des roten Wellenlängenbereichs müssen mit einer höheren Laserleistung kompensiert werden. Reichten bei einem grünen Laser $10 - 20mW$ Strahlleistung aus, so werden im roten Wellenlängenbereich Leistungen von $50 - 100mW$ benötigt. Der Preis dieser leistungsstarken roten Laser liegt noch immer unter jenem, der schwächeren grünen. Auch die Baugröße ist viel kompakter und die Lebensdauer größer.

Um die abgestrahlte Energie des zu vermessenden Walzgutes bestmöglich herauszufiltern, wird vor das Objektiv der Kamera ein Interferenzfilter montiert. Dieser Interferenzfilter für $\lambda = 685nm$ lässt nur einen schmalen Wellenlängenbereich durch und absorbiert den Rest. Die Kennlinie des Interferenzfilters wurde aus dessen Datenblatt entnommen und ist in Abbildung 2.7 dargestellt.
Dadurch ergibt sich ein weiteres Problem. Wie bereits in Kapitel 2.1.2 erwähnt, müssen die Laser- und Kameraposition durch Kalibration bestimmt werden. Die roten Leuchtdioden auf dem Kalibrationsobjekt besitzen ihr Intensitätsmaximum bei $\lambda = 625nm$. Dieser Wellenlängenbereich wird vom Interferenzfilter zur Gänze absorbiert und die Leuchtdioden werden dadurch unsichtbar.
Ein Tausch der Leuchtdioden am Kalibrationsobjekt ist ohne dessen Zerstörung nicht möglich. Eine Kalibration kann somit nur ohne Filter durchgeführt werden. Nach erfolgter Kalibration wird der Filter am Objektiv angeschraubt. Dabei ist zu beachten, dass die Position der Kamera nicht verändert werden darf.

Kapitel 3

Mathematische Grundlagen

Die in diesem Kapitel beschriebenen mathematischen Methoden werden als Grundlage für alle weiteren Betrachtungen angesehen.

3.1 Lineare Algebra

3.1.1 Matrizenoperationen

- Transponierte Matrix: [8]

  Bei einer transponierten Matrix werden die Zeilen mit den gleichstelligen Spalten vertauscht.

$$\mathbf{A} = (a_{ik})_{m,n} \leftrightarrow \mathbf{A^T} = (a_{ki})_{n,m} \tag{3.1}$$

  Zum Beispiel:

$$\mathbf{A} = \begin{pmatrix} 1\,2\,3 \\ 4\,5\,6 \\ 7\,8\,9 \end{pmatrix} \leftrightarrow \mathbf{A^T} = \begin{pmatrix} 1\,4\,7 \\ 2\,5\,8 \\ 3\,6\,9 \end{pmatrix} \tag{3.2}$$

- Inverse Matrix: [8]

  Die Inverse einer Matrix ist die Umkehrabbildung dieser.

$$\mathbf{A}\,\mathbf{A^{-1}} = \mathbf{A^{-1}}\,\mathbf{A} = \mathbf{I} \tag{3.3}$$

  Wobei $\mathbf{I}$ die Einheitsmatrix mit 1 in der Hauptdiagonale und 0 sonst ist.

- Orthogonale Matrix: [8]

  Eine Matrix ist orthogonal, wenn die Transponierte dieser Matrix gleich der Inversen ist.

$$\mathbf{A^T} = \mathbf{A^{-1}}\ bzw.\ \mathbf{A^T}\,\mathbf{A} = \mathbf{I} \tag{3.4}$$

- Diagonale Matrix: [8]

  Eine diagonale Matrix besitzt nur Einträge in der Hauptdiagonale, alle anderen Einträge sind 0.

$$\mathbf{A_d} = \begin{pmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{pmatrix} \tag{3.5}$$

  Die Inverse einer diagonalen Matrix lässt sich leicht berechnen. Deren Einträge sind die reziproken Werte der Ausgangsmatrix.

$$\mathbf{A_d^{-1}} = \begin{pmatrix} \frac{1}{a_1} & & \\ & \ddots & \\ & & \frac{1}{a_n} \end{pmatrix} \tag{3.6}$$

- Singuläre Matrix, bzw. der Rang einer Matrix: [8]

  Stellt man ein lineares Gleichungssystem in Matrizenschreibweise dar, so erhält man eine Gleichung der folgenden Form:

$$\mathbf{A}\,\vec{a} = \vec{b} \tag{3.7}$$

  In dieser Gleichung ist $\vec{a}$ der gesuchte Lösungsvektor, welcher n Einträge besitzt. Die Matrix $\mathbf{A}$ enthält die Linearfaktoren von $\vec{a}$, und $\vec{b}$ enthält die von $\vec{a}$ unabhängigen Skalare. Dieser Lösungsvektor ist eindeutig definiert, wenn mindestens n voneinander linear unabhängige Gleichungen gefunden werden können. Lineare Unabhängigkeit ist nur dann gegeben, wenn eine Gleichung nicht durch Multiplikation mit einem Skalar durch eine andere ausgedrückt werden kann. Dies wird durch den Rang einer Matrix ausgedrückt, welcher die Anzahl der linear unabhängigen Gleichungen (Zeilen) in der Matrix angibt. Somit muss der Rang der Matrix $\mathbf{A} \geq n$ sein, damit $\vec{a}$ eindeutig definiert ist.
  Ist der Rang der Matrix $\mathbf{A} < n$, dann spricht man von einer singulären Matrix. Herkömmliche Lösungsverfahren liefern in diesem Fall kein Ergebnis, da die Determinante der Matrix Null ist und somit die inverse Matrix nicht mehr bestimmt ist.

3.1.2 Homogene Koordinaten

Man gehe von einem dreidimensionalen, kartesischen Koordinatensystem aus, welches in weiterer Folge als raumfestes Koordinatensystem bezeichnet wird. In diesem Koordinatensystem ist ein Punkt P anhand seines Punktvektors wie folgt definiert:

$$\vec{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.8}$$

Wendet man für diesen Punkt die Schreibweise für homogene Koordinaten [8] an, so erhält man:

$$\vec{P_h} = \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} \tag{3.9}$$

Auf die gleiche Weise können ebene, homogene Koordinaten definiert werden. Ein Punkt Q der Ebene hat somit die Koordinaten:

$$\vec{Q_h} = \begin{bmatrix} X \\ Y \\ w \end{bmatrix} \tag{3.10}$$

Die Schreibweise mit homogenen Koordinaten hat den Vorteil, dass Koordinatentransformationen über eine Transformationsmatrix erfolgen können, welche sowohl den translatorischen als auch den rotatorischen Anteil der Transformation enthält.

3.1.3 Projektionsvorschrift

Mithilfe einer Projektionsvorschrift [6] ist es möglich, eine homogene Projektion zwischen zwei Ebenen im Raum zu berechnen. Diese spezielle Art der Transformation ist eindeutig definiert, wenn mindestens je vier Punkte mit ihren Vektoren in beiden Ebenen bekannt sind. Diese Projektion ist durch folgende Beziehung definiert:

$$\vec{p_r} = \mathbf{H}\,\vec{p_0} \tag{3.11}$$

Es wird ein Punkt p_0 von einer Ebene auf eine andere Ebene projiziert, wo er als Punkt p_r erscheint. p_0 muss dabei in homogenen Ebenenkoordinaten vorliegen.

$$\begin{bmatrix} X_r \\ Y_r \\ w_r \end{bmatrix} = \begin{bmatrix} h_{11}\, h_{12}\, h_{13} \\ h_{21}\, h_{22}\, h_{23} \\ h_{31}\, h_{32}\, h_{33} \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ 1 \end{bmatrix} \tag{3.12}$$

Der aus dieser Multiplikation erhaltene Vektor $\vec{p_r}$ liegt in nicht affinen Koordinaten vor.

Umwandlung von $\vec{p_r}$ in affine Koordinaten:

$$\vec{p_r^a} = \begin{bmatrix} \frac{X_r}{w_r} \\ \frac{Y_r}{w_r} \\ 1 \end{bmatrix} \tag{3.13}$$

Sind die Punkte p_0 und p_r in beiden Ebenen bekannt, so kann die $3 * 3$ Matrix $\mathbf{H}$ wie folgt berechnet werden:

Setzt man voraus, dass man diese Matrix immer durch eine ihrer Komponenten dividieren kann, so bleiben acht signifikante unbekannte Größen übrig. Somit müssen mindestens vier Punkte in beiden Ebenen bekannt sein. Durch die Aufspaltung der Punktvektoren in ihre X- bzw. Y-Komponenten lassen sich acht Gleichungen aufstellen, durch die man die acht unbekannten Komponenten berechnen kann. Dabei bedient man sich folgender Vorgehensweise:

Die projizierten Punkte werden in affinen Koordinaten dargestellt.

$$\begin{aligned} X_r^a &= \frac{X_r}{w_r} = \frac{h_{11}\,X_0 + h_{12}\,Y_0 + h_{13}\,1}{h_{31}\,X_0 + h_{32}\,Y_0 + h_{33}\,1} \\ Y_r^a &= \frac{Y_r}{w_r} = \frac{h_{21}\,X_0 + h_{22}\,Y_0 + h_{23}\,1}{h_{31}\,X_0 + h_{32}\,Y_0 + h_{33}\,1} \end{aligned} \tag{3.14}$$

Durch ausmultiplizieren der Gleichungen erhält man:

$$\begin{aligned} -h_{11}\,X_0 - h_{12}\,Y_0 - h_{13} + h_{31}\,X_0\,X_r^a + h_{32}\,Y_0\,X_r^a + h_{33}\,X_r^a &= 0 \\ -h_{21}\,X_0 - h_{22}\,Y_0 - h_{23} + h_{31}\,X_0\,Y_r^a + h_{32}\,Y_0\,Y_r^a + h_{33}\,Y_r^a &= 0 \end{aligned} \tag{3.15}$$

Schreibt man nun für alle Punkte (n beliebig, mindestens vier) die Gleichungen in Matrizenschreibweise an, so erhält man:

$$\begin{bmatrix} -X_0(1) & -Y_0(1) & -1 & 0 & 0 & 0 & X_0(1)X_r^a(1) & Y_0(1)X_r^a(1) & X_r^a(1) \\ 0 & 0 & 0 & -X_0(1) & -Y_0(1) & -1 & X_0(1)Y_r^a(1) & Y_0(1)Y_r^a(1) & Y_r^a(1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -X_0(n) & -Y_0(n) & -1 & 0 & 0 & 0 & X_0(n)X_r^a(n) & Y_0(n)X_r^a(n) & X_r^a(n) \\ 0 & 0 & 0 & -X_0(n) & -Y_0(n) & -1 & X_0(n)Y_r^a(n) & Y_0(n)Y_r^a(n) & Y_r^a(n) \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0 \tag{3.16}$$

oder kurz:

$$\mathbf{G}\,\vec{h} = 0 \tag{3.17}$$

Diese Gleichung kann nun mit der Methode der Singulärwertzerlegung gelöst werden.

In *Matlab*® wurde diese eben beschriebene Berechnungsmethode als Funktion "GenProj.m" erstellt.

3.2 Koordinatentransformation

Geht man davon aus, dass neben einem raumfesten Koordinatensystem ein zweites Koordinatensystem existiert, welches durch Lage des Ursprung und Ausrichtung der Achsen im raumfesten Koordinatensystem bekannt ist, so kann eine Koordinatentransformation zwischen den beiden erfolgen. Man kann die Lage eines Punktes P im anderen, lokalen Koordinatensystem P' bestimmen, indem der Vektor des Punktes P mit der Transformationsmatrix $\mathbf{T}$ multipliziert wird. Man spricht von einer Transformation. [9]

$$\vec{P'} = \mathbf{T}\ \vec{Ph} \tag{3.18}$$

$$\begin{bmatrix} X_{P'} \\ Y_{P'} \\ Z_{P'} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11}\ r_{12}\ r_{13}\ \Delta x \\ r_{21}\ r_{22}\ r_{23}\ \Delta y \\ r_{31}\ r_{32}\ r_{33}\ \Delta z \\ 0\ \ 0\ \ 0\ \ 1 \end{bmatrix} \begin{bmatrix} X_{Ph} \\ Y_{Ph} \\ Z_{Ph} \\ 1 \end{bmatrix} \tag{3.19}$$

Die Transformationsmatrix setzt sich aus den rotatorischen (r) und den translatorischen (Δ) Anteilen zusammen. Sie ist das Produkt dreier Rotationsmatrizen und einer Translationsmatrix.

Die Rücktransformation eines Punktes P' von einem lokalen Koordinatensystem in das raumfeste Koordinatensystem erfolgt durch Multiplikation des Punktes P' mit der inversen Transformationsmatrix.

$$\vec{Ph} = \mathbf{T^{-1}}\ \vec{P'} \tag{3.20}$$

3.2.1 Bestimmung der Translationsmatrix

Die translatorischen Anteile der Transformationsmatrix [9] bestimmen die Lage des Ursprungs des lokalen Koordinatensystems im raumfesten Koordinatensystem durch deren Anteile auf den jeweiligen Achsen.

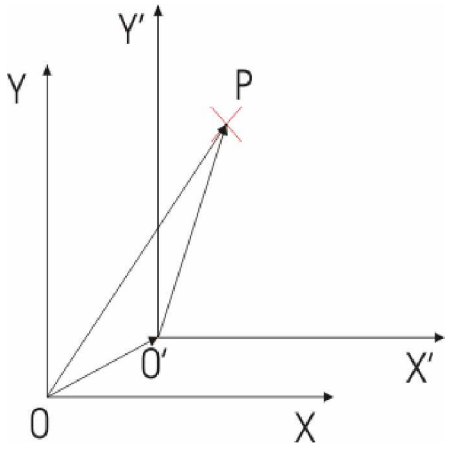


Abbildung 3.1: Prinzip der Translation

Gegeben sei ein Punkt P, mit den Koordinaten im raumfesten Koordinatensystem 0 mit dem Ursprung $[0;0;1]$:

$$P = \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} \tag{3.21}$$

Um die Koordinaten des Punktes P im lokalen Koordinatensystem feststellen zu können, muss die Lage des Ursprungs dieses Koordinatensystems im raumfesten bekannt sein.

$$0' = \begin{bmatrix} x_{0'} \\ y_{0'} \\ 1 \end{bmatrix} \tag{3.22}$$

Die translatorischen Anteile bilden sich nun aus der Differenz beider Ursprungskoordinaten:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} x_{0'} \\ y_{0'} \\ 1 \end{bmatrix} = \begin{bmatrix} -x_{0'} \\ -y_{0'} \\ 0 \end{bmatrix} \tag{3.23}$$

Durch die folgende Definition wird die Translationsmatrix gebildet:

$$T_T = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

3.2.2 Bestimmung der Rotationsmatrizen

Die rotatorischen Anteile der Transformationsmatrix bestimmen die Ausrichtung der drei Achsen des lokalen Koordinatensystem zum raumfesten.

Methode nach Euler: [9]

Die Lage eines, um einen festen Punkt 0 frei drehbaren Körpers ist durch drei unabhängige Parameter bestimmt. Als solche eignen sich drei von Euler eingeführte Winkel (ϕ, θ *und* ψ), welche die Ausrichtung der Achsen eines lokalen Koordinatensystems (X''', Y''' *und* Z''') gegen ein raumfestes (X, Y *und* Z) mit gemeinsamen Ursprung festhalten.

Diese Winkel beschreiben drei aufeinanderfolgende Drehungen. Sind diese Winkeln positiv so erfolgt die Drehung im Uhrzeigersinn, negativ gegen den Uhrzeigersinn.

$$\begin{aligned} 0 \leq \phi, \theta, \psi \leq \pi \quad & \textit{Drehung mit dem Uhrzeigersinn} \\ -\pi \leq \phi, \theta, \psi \leq 0 \quad & \textit{Drehung gegen den Uhrzeigersinn} \end{aligned} \tag{3.25}$$

1. Drehung um die Z-Achse:

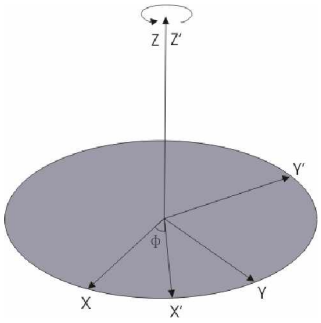


Abbildung 3.2: Drehung um die Z-Achse

Durch die Drehung der Z-Achse, um den Winkel ϕ, erhält man die neue Achsenausrichtung X', Y' *und* Z'.

Die Rotationsmatrize errechnet sich wie folgt:

$$R_\phi = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 & 0 \\ -\sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.26}$$

2. Drehung um die X'-Achse:

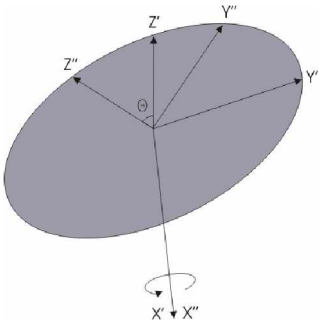


Abbildung 3.3: Drehung um die X'-Achse

Die neue Achsenausrichtung ist somit X'', Y'' *und* Z''.

Die Rotationsmatrize für die Drehung um die X'-Achse:

$$R_\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

3. Drehung um die Z''-Achse:

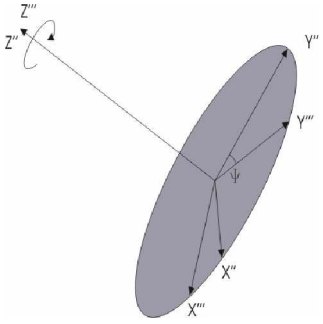


Abbildung 3.4: Drehung um die Z''-Achse

Und eine Drehung um die X''-Achse komplettiert die Rotation in die gewünschte Achsenausrichtung X''', Y''' *und* Z'''.

Die Rotationsmatrize für diese Drehung lautet:

$$R_\psi = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 & 0 \\ -\sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

Die drei Rotationen werden noch einmal in einem Gesamtbild veranschaulicht.

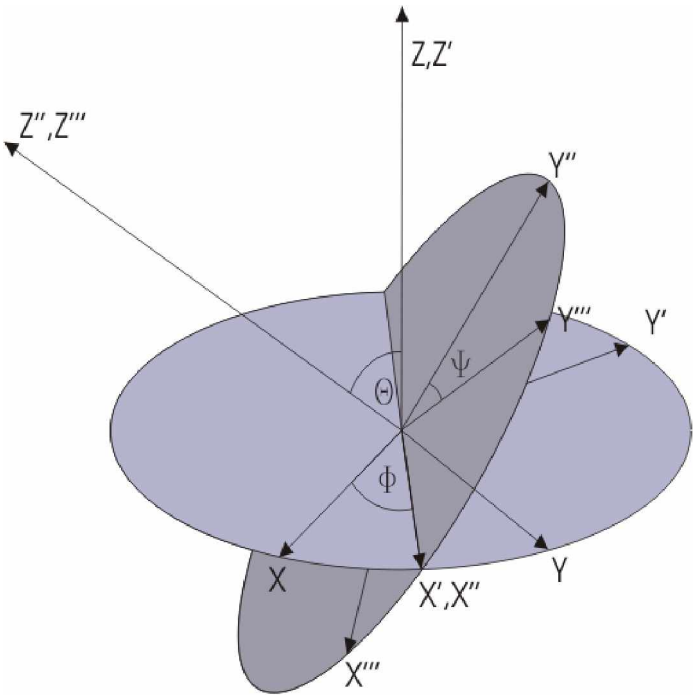


Abbildung 3.5: Prinzip der Rotation

Die Transformationsmatrix, zusammengesetzt aus den drei Rotationsmatrizen und der Translationsmatrix, errechnet sich nun folgendermaßen:

$$T = R_\psi \, R_\theta \, R_\phi \, T_T \tag{3.29}$$

Dabei ist die Reihenfolge der Matrizenmultiplikation von entscheidender Bedeutung, da durch sie gewährleistet ist, dass zuerst die Verschiebung des Ursprungs und anschließend die einzelnen Rotationen erfolgen.

3.3 Die Methode der Singulärwertzerlegung (Singular Value Decomposition)

Die Singulärwertzerlegung (SVD) [6] kann dazu verwendet werden, die bestmögliche nichttriviale Lösung von $\vec{h}$ für die folgende Gleichung zu berechnen:

$$\mathbf{G} \, \vec{h} = 0 \tag{3.30}$$

Die triviale Lösung dieser Gleichung wäre die, dass alle Komponenten von $\vec{h}$ Null sind. $[h_1, \ldots, h_n = 0]$

Angewendet wird die Singulärwertzerlegung auf $\mathbf{G}$ folgendermaßen:

$$\mathbf{G} \overset{\text{SVD}}{\mapsto} [\mathbf{U}, \mathbf{S}, \mathbf{V}] \tag{3.31}$$

Ist $\mathbf{G}$ eine $m * n$ Matrix, dann existieren die orthogonalen Matrizen

$$\begin{aligned} \mathbf{U} &= [u_1, \ldots, u_m] \, \epsilon \, \mathbb{R}^{m \times n} : \| \, \mathbf{U} \, \|_2 = \mathbf{I} \\ \mathbf{V} &= [v_1, \ldots, v_n] \, \epsilon \, \mathbb{R}^{m \times n} : \| \, \mathbf{V} \, \|_2 = \mathbf{I} \end{aligned} \tag{3.32}$$

so, dass

$$\begin{bmatrix} \\ \mathbf{G} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{U} \\ \\ \end{bmatrix} \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_p \end{bmatrix} \begin{bmatrix} \\ \mathbf{V^T} \\ \\ \end{bmatrix} \tag{3.33}$$

ist, wobei $s_1 \geq s_2 \geq \ldots \geq s_p \geq 0$. s_i sind die Singulärwerte (Singular Values) von $\mathbf{G}$. u_i und v_i sind die linken und rechten, i-ten singulären Spaltenvektoren von $\mathbf{G}$. In diesem Fall sind v_i die orthogonalen Basisvektoren von $\mathbf{G}$ und u_i die orthogonalen Normalprojektionen von $\mathbf{G}$ auf $\mathbf{V}$. Die Singulärwerte sind die Varianzen der Basisvektoren von $\mathbf{V}$ (Varianz = 2. Moment um den Mittelwert).

Für s_p gilt:

$$s_p = min_{v_p \neq 0} \frac{\| \, \mathbf{G} \, v_p \, \|_2}{\| \, v_p \, \|_2} \tag{3.34}$$

Die gesuchte Lösung für $\vec{h}$ in Gleichung 3.30 ist der zu s_p korrespondierende Basisvektor v_p.

Wird die Singulärwertzerlegung auf einen verrauschten Datensatz, der Form

$$\mathbf{G}\,\vec{h} \;=\; \vec{e} \quad wobei \quad \vec{e} \;=\; [e_1, \ldots, e_n \;\neq\; 0] \tag{3.35}$$

angewendet, so wird $\vec{h}$ so angepasst, dass er die Gleichung am besten beschreibt. $\vec{e}$ ist in dieser Gleichung der Fehlervektor.
Diese Form von Gleichung erhält man immer, wenn es sich bei $\mathbf{G}$ um Messdaten bzw. Messpunkte handelt, welche mit einem mathematischen Gesetz beschrieben werden können. Im Lösungsvektor $\vec{h}$ sind dann die Koeffizienten des gesuchten Gesetzes bzw. der gesuchten Polynomfunktion enthalten. Die Funktion weist den geringsten quadratischen Mittelwert zu den Messpunkten auf (least mean square fit).

Rückgabeparameter der Singulärwertzerlegung:

Die Anwendung der Singulärwertzerlegung erfolgt in *Matlab*® mit dem Befehl "svd". Als Rückgabeparameter erhält man die drei Matrizen $\mathbf{U}, \mathbf{S}$ *und* $\mathbf{V}$

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] \;=\; svd(\mathbf{G}) \tag{3.36}$$

Die Bedeutung der Rückgabeparameter wird am besten mit einem Beispiel veranschaulicht.

Gegeben sei ein Datensatz $\mathbf{P}$, welcher in der Ebene eine Linie beschreibt. Dieser Datensatz ist verrauscht, die einzelnen Punkte liegen nicht exakt auf der Linie.

$$\mathbf{P} \;=\; \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \tag{3.37}$$

Gesucht ist eine Gerade welche den Datensatz am besten beschreibt.

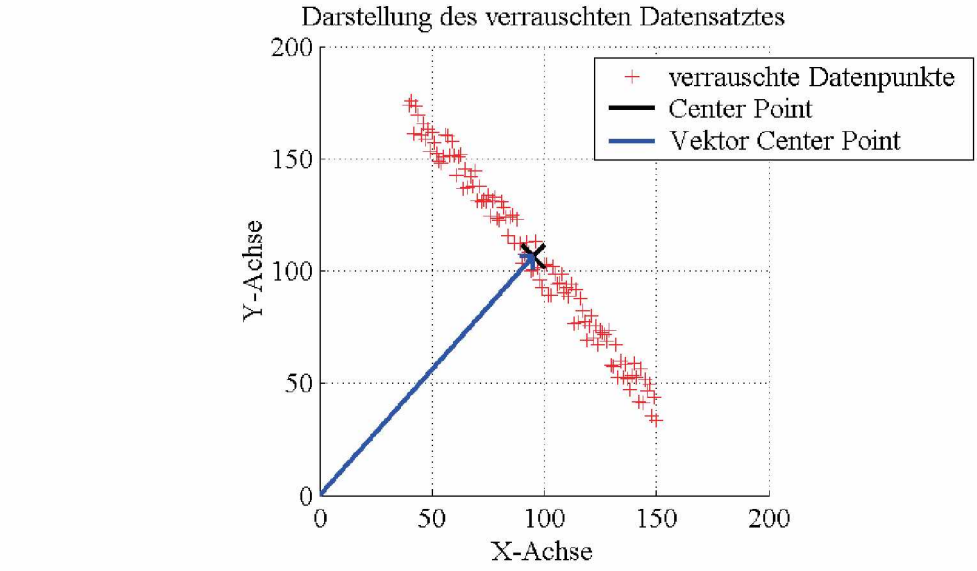


Abbildung 3.6: Darstellung der Datenpunkte

Um zu große Vektoren zu vermeiden wird ein "Center Point" berechnet und von den x_i und y_i Werten abgezogen. Dieser "Center Point", mit den Koordinaten $[x_0, y_0, 1]$, liegt immer auf der angepassten Hauptachse.

$$x_0 = \frac{1}{n}\sum_{i=1}^{n} x_i \;,\; y_0 = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.38}$$

$$xm_i = x_i - x_0 \;,\; ym_i = y_i - y_0 \;,\; 1-1 = 0 \tag{3.39}$$

Die Geradengleichung kann mit "planner line" Koordinaten folgendermaßen angeschrieben werden (Siehe Kapitel 3.4.1):

$$\begin{bmatrix} xm_1 & ym_1 \\ xm_2 & ym_2 \\ \vdots & \vdots \\ xm_i & ym_i \end{bmatrix} \begin{bmatrix} Y \\ -X \end{bmatrix} = 0 \tag{3.40}$$

Diese Gleichung anders angeschrieben:

$$\mathbf{G}\,\vec{h} = 0 \tag{3.41}$$

Auf $\mathbf{G}$ die Singulärwertzerlegung angewendet erhält man $\mathbf{U}, \mathbf{S}$ *und* $\mathbf{V}$:

$$\mathbf{G} \overset{\text{SVD}}{\mapsto} [\mathbf{U}, \mathbf{S}, \mathbf{V}] \tag{3.42}$$

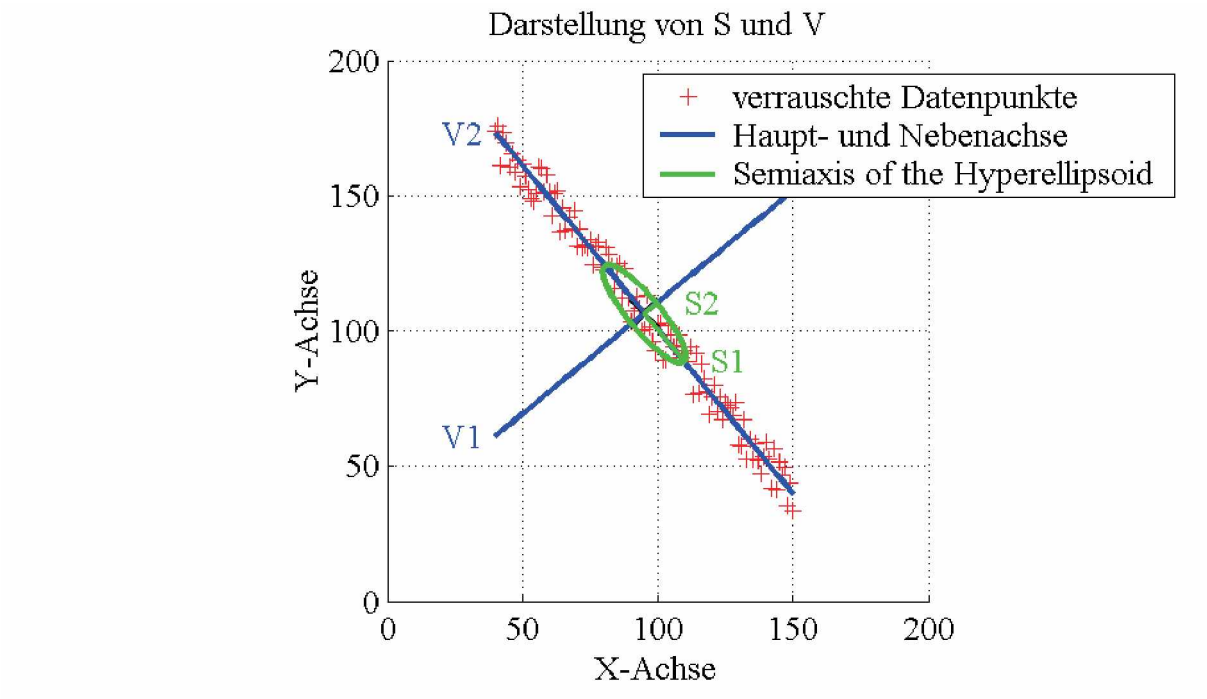


Abbildung 3.7: Darstellung der Rückgabewerte

Der Rückgabewert **V** ist in diesem Fall eine $2 * 2$ Matrix (Ebene - zwei Dimensionen). In dieser Matrix steht pro Dimension eine Lösung, also in diesem Beispiel zwei Geraden in "planner line" Koordinaten. Diese zwei Geraden stehen senkrecht aufeinander. Sie geben die Richtung der Halbachsen der Fehlerellipse an. Die gesuchte Lösung mit dem kleinsten Gesamtfehler steht in der 2.Spalte von **V**.

In der diagonalen Matrix **S** sind die Singulärwerte enthalten. Die Standardabweichung lässt sich daraus wie folgt ermitteln:

$$\sigma = \sqrt{s_i} \tag{3.43}$$

Da die Varianz nur für positive Werte definiert ist, sind auch die Singulärwerte immer positiv. Sie werden mit ansteigender Ordnung immer kleiner.

$$s_1 > s_2 > \ldots > s_p \geq 0 \tag{3.44}$$

Bei einer exakten Lösung (kein Fehler) ist s_p Null. Somit ist ersichtlich, dass die beste Lösung mit dem kleinsten Fehler immer in der letzten Spalte von **V** steht.
Die Standardabweichungen sind auch die Halbachsen der Fehlerellipse (Semi axis of the hyper ellipsoid). In der Ebene handelt es sich um eine Ellipse. In der dritten Dimension spricht man von einem Ellipsoid und in höheren Dimensionen von einem Hyperellipsoid.
Im Rückgabeparameter **U** sind die Fehlervektoren der jeweiligen Punkte enthalten. Der Punkt mit dem kleinsten Fehler hat den kleinsten absoluten Betrag in der zugehörigen Spalte.

In diesem Beispiel ist dieser Punkt:

$$P_{min.\,Fehler} \rightarrow min(abs(U(:,2))) \tag{3.45}$$

3.4 Anpassen von Funktionen

Alle gemessenen Datenpunkte des zu bestimmenden Objekts sind verrauscht bzw. liegen aufgrund von Schmutzpartikeln nicht auf dem Objekt. Somit muss aus dem Datensatz die bestmögliche Funktion mit dem geringsten quadratischen Fehler gefunden werden.

3.4.1 Anpassen von Linien in der Ebene

Bei der hier verwendeten Methode nach Klein [6] wird eine Linie durch Verwendung von "planner line" Koordinaten dargestellt. Eine Gerade ist durch zwei auf ihr liegende Punkte eindeutig definiert.
So folgt für eine Gerade in der Ebene:

$$\vec{P_1} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \; und \; \vec{P_2} = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \tag{3.46}$$

Ein Punkt

$$\vec{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.47}$$

liegt nur dann auf der Gerade, wenn er sich als eine Linearkombination (Vielfaches) der beiden ersten Punkte darstellen lässt.

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0 \tag{3.48}$$

Durch Lösen dieser Determinante erhält man:

$$x\,(y_1 - y_2) \;-\; y\,(x_1 - x_2) \;+\; (x_1\,y_2 - x_2\,y_1) \;=\; 0 \tag{3.49}$$

Diese Gleichung angeschrieben mit den "planner line" Koordinaten X,Y und N:

$$x\,Y \;-\; y\,X \;+N \;=\; 0 \tag{3.50}$$

bzw. in Matrizenschreibweise:

$$[x\;y\;1] \begin{bmatrix} Y \\ -X \\ N \end{bmatrix} = 0 \tag{3.51}$$

Beschreibt man einen verrauschten Satz von n Punkten mit einer Linie, so ergibt sich für jeden Punkt P_i eine Abweichung e_i.

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} Y \\ -X \\ N \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{3.52}$$

oder kurz:

$$\mathbf{G}\;\;\vec{l} = \vec{e} \tag{3.53}$$

Gelöst wird diese Gleichung mit der Singulärwertzerlegung.

Die "planner line" Koordinaten lassen sich folgendermaßen in die gewöhnliche Form der Geradengleichung ($y = k\;x + d$) umrechnen.

Aus Gleichung 3.50 folgt:

$$y = \frac{Y}{X}\;\;x + \frac{N}{X} \tag{3.54}$$

Somit errechnen sich die Steigung k und der Abstand d:

$$k = \frac{Y}{X}\;\;,\;\; d = \frac{N}{X} \tag{3.55}$$

In *Matlab*® wurde diese Berechnungsmethode als Funktion "FitLine.m" erstellt.

3.4.2 Anpassen von parallelen Linien in der Ebene

Diese Methode kann auch zum Anpassen von parallelen Linien verwendet werden.
Für zwei verrauschte Datensätze von Punkten, welche zwei parallele Linien beschreiben, ergeben sich die "planner line" Koordinaten wie folgt:

$$Linie\,1 \;:\; X_1 \;,\; Y_1 \;und\; N_1$$

$$Linie\,2 \;:\; X_2 \;,\; Y_2 \;und\; N_2$$

Diese beiden Linien sind zueinander parallel, wenn gilt: $X_1 = X_2$ und $Y_1 = Y_2$.
Somit lässt sich anschreiben:

$$\begin{bmatrix} {}_1x_1 & {}_1y_1 & 1 & 0 \\ {}_1x_2 & {}_1y_2 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ {}_1x_m & {}_1y_m & 1 & 0 \\ {}_2x_1 & {}_2y_1 & 0 & 1 \\ {}_2x_2 & {}_2y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ {}_2x_n & {}_2y_n & 0 & 1 \end{bmatrix} \begin{bmatrix} Y \\ -X \\ N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} {}_1e_1 \\ {}_1e_2 \\ \vdots \\ {}_1e_m \\ {}_2e_1 \\ {}_2e_2 \\ \vdots \\ {}_2e_n \end{bmatrix} \tag{3.56}$$

Die beste nicht triviale Lösung lässt sich wiederum mit der Singulärwertzerlegung errechnen.

In *Matlab*® wurde diese Berechnungsmethode als Funktion "FitParLine.m" erstellt.

3.4.3 Anpassen einer Ebene

Auf die gleiche Art und Weise kann eine Ebene zu einem bekannten Datensatz von n Punkten im Raum angepasst werden.

Die Gleichung einer Ebene in Koordinatendarstellung:

$$E_1\,x + E_2\,y + E_3\,z + E_4 = 0 \quad ,wobei \quad \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = Normalvektor\ der\ Ebene. \tag{3.57}$$

Diese Gleichung angeschrieben in Matrizenschreibweise:

$$[x\ y\ z\ 1] \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \end{bmatrix} = 0 \tag{3.58}$$

Diese Gleichung ist eindeutig definiert, wenn mindestens drei Punkte der Ebene, welche nicht auf einer Geraden liegen, bekannt sind.
Für einen verrauschten Satz von n Punkten schreibt man:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{3.59}$$

Die beste nicht triviale Lösung wird wiederum mit der Singulärwertzerlegung errechnet.

In *Matlab*® wurde diese Berechnungsmethode als Funktion "FitLaserPlane.m" erstellt.

3.4.4 Anpassen von Kreisen

Ein Kreis wird in der Ebene durch die ideale Kreisgleichung wie folgt beschrieben:

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0 \tag{3.60}$$

Diese Gleichung ist erfüllt, wenn ein Punkt mit den Koordinaten $[x_i; y_i]$ auf dem Kreis mit dem Radius r und dem Zentrum $[x_0; y_0]$ liegt.

Die ideale Kreisgleichung ausmultipliziert ergibt:

$$x_i^2 + y_i^2 - 2\ x_i\ x_0 - 2\ y_i\ y_0 + x_0^2 + y_0^2 - r^2 = 0 \tag{3.61}$$

Diese Gleichung umgeschrieben auf Grassmann Koordinaten:

$$C_1\ (x_i^2 + y_i^2) - C_2\ x_i + C_3\ y_i - C_4 = 0 \tag{3.62}$$

Die Koeffizienten C_1, C_2, C_3 *und* C_4 werden "tetra-circular" Koordinaten genannt, sind homogen, und beschreiben einen dreidimensionalen Raum.
Für die eindeutige Lösung dieser Gleichung sind mindestens 3 Punkte nötig. Wenn drei Punkte P_1, P_2 *und* P_3 einen Kreis definieren, so liegt ein Punkt P ebenfalls auf dem Kreis wenn:

$$\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0 \tag{3.63}$$

Wenn diese Matrix von Rang 3 ist, d.h. es ist eine lineare Kombination in der Matrix enthalten, so beschreibt der Datensatz einen Kreis. Rang 2 bedeutet, der Datensatz beschreibt eine Gerade, und besitzt die Matrix den Rang 1 bedeutet dies, dass alle Daten in einem Punkt liegen.

Die Berechnung der Determinante der Matrix mittels der Unterdeterminanten ergibt:

$$(x^2+y^2)\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} - x\begin{vmatrix} x_1^2+y_1^2 & y_1 & 1 \\ x_2^2+y_2^2 & y_2 & 1 \\ x_3^2+y_3^2 & y_3 & 1 \end{vmatrix} + y\begin{vmatrix} x_1^2+y_1^2 & x_1 & 1 \\ x_2^2+y_2^2 & x_2 & 1 \\ x_3^2+y_3^2 & x_3 & 1 \end{vmatrix} - \begin{vmatrix} x_1^2+y_1^2 & x_1 & y_q \\ x_2^2+y_2^2 & x_2 & y_2 \\ x_3^2+y_3^2 & x_3 & y_3 \end{vmatrix} = 0 \tag{3.64}$$

Nun werden die Unterdeterminanten durch Konstanten ersetzt und man erhält die Gleichung 3.62. Die Beschreibung des Kreises durch die Unterdeterminanten ist gleich der Beschreibung mit "tetra circular" Koordinaten.

Diese Definition kann wiederum verwendet werden, um exakte Lösungen zu berechnen. In unserem Fall sind jedoch die Datensätze verrauscht und somit muss das Gleichungssystem überbestimmt vorliegen, d.h. es sind mehr als 3 Punkte erforderlich. Die Gleichung muss dafür wie folgt angeschrieben werden:

$$C_1\,(x_i^2+y_i^2) \;-\; C_2\,x_i \;+\; C_3\,y_i \;-\; C_4 \;=\; e_i \;\neq\; 0 \tag{3.65}$$

angeschrieben in Matrizenschreibweise:

$$\begin{bmatrix} x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^2+y_n^2 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{3.66}$$

Um den Lösungsvektor $\vec{C}$ zu ermitteln wird wieder die Singulärwertzerlegung angewendet.

Aus den Gleichungen 3.61 und 3.62 folgt für den Kreismittelpunkt $[x_0; y_0]$ und den Radius r:

$$x_0 \;=\; -\frac{C_2}{2\,C_1} \quad , \quad y_0 \;=\; -\frac{C_3}{2\,C_1} \quad \textit{und} \quad r \;=\; \sqrt{x_0^2+y_0^2-\frac{C_4}{C_1}} \tag{3.67}$$

Diese Funktion wurde in *Matlab*® mit dem Algorithmus "FitCircle.m" realisiert.

3.4.5 Anpassen von Kreissegmenten

Die soeben beschriebene Methode zum Anpassen von Kreisen mit Grassmann Koordinaten [6] eignet sich nur, wenn die verrauschten Datenpunkte über den gesamten Umfang des Kreises verteilt vorliegen. Die Fehlerverteilung der zufälligen Fehler entspricht immer einer Gauß'schen Fehlerfunktion und die Summe aller Fehler hebt sich in diesem Fall auf. Ist vom Kreis jedoch nur ein Kreissegment bekannt, hebt sich die Summe aller Fehler nicht mehr auf und diese Methode kann nur noch als gute Näherung verwendet werden.
Um wiederum die best mögliche Lösung zu errechnen, werden der Mittelpunkt $[x_0; y_0]$ und der Radius r iterativ angepasst. [7]

Der Fehler einer Kreisfunktion ist gegeben durch die Gleichung:

$$F \;=\; \sum_{i=1}^{n} \left(f(x_i,y_i)\right)^2 \quad , \quad f(x_i,y_i) \;=\; (x_i-x_0)^2+(y_i-y_0)^2-r^2 \tag{3.68}$$

Man könnte nun eine Fehlerrechnung für diese Gleichung durchführen, dazu die partiellen Ableitungen dieser Gleichung für x_0, y_0 *und* r bilden und eine Kreisfunktion für den geringsten Fehler berechnen. Das Problem dabei ist, dass die partiellen Ableitungen nicht mehr linear sind und eine exakte Lösung der Gleichungen sehr komplex wird.

Die im Folgenden beschriebene Methode ist erheblich einfacher zu rechnen.

Der Fehler eines Punktes P_i nach Gleichung 3.68 entspricht ungefähr dem Normalabstand des Punktes vom angepassten Kreis.

$$h_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} - r \tag{3.69}$$

Der gesamte Fehler errechnet sich somit:

$$H = \sum_{i=1}^{n} h_i^2 \tag{3.70}$$

Bei dieser Methode wird nach einer vorausgegangenen Anpassung x_0, y_0 *und* r iterativ korrigiert, damit H ein Minimum wird.

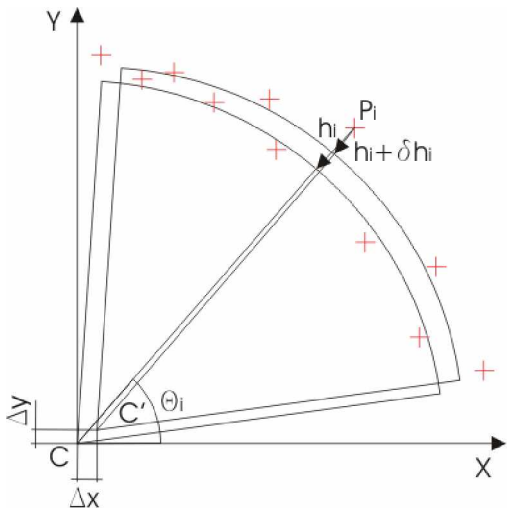


Abbildung 3.8: Prinzip der Kreissegmentanpassung

Wird ein Kreismittelpunkt $C[x_0, y_0]$ (Dargestellt in Abbildung 3.8) mit dem zugehörigen Radius r um Δx und Δy nach C' verschoben, so ändert sich der Fehler für einen Punkt Pi um δh_i.

Die Änderung des Fehlers, bei konstantem Radius, beträgt:

$$\delta h_i|_{r=konst.} \simeq -\Delta x \cos(\theta_i) - \Delta y \sin(\theta_i) \tag{3.71}$$

Bleibt hingegen der Mittelpunkt des Kreises $C[x_0, y_0]$ fest, und wird nur der Radius des Kreises verändert, so ergibt sich für die Fehleränderung:

$$\Delta h_i|_{x_0, y_0 = konst.} \simeq -\Delta r \tag{3.72}$$

Somit beträgt der gesamte Fehler H, durch die Korrektur des Mittelpunktes und des Radius, nach Gleichung 3.70:

$$H = \sum_{i=1}^{n}(h_i + \delta h_i)^2 = \sum_{i=1}^{n}(h_i - \Delta x \cos(\theta_i) - \Delta y \sin(\theta_i) - \Delta r)^2 \tag{3.73}$$

Mit der Vereinfachung, dass Θ_i durch die Verschiebung konstant bleibt, sind die partiellen Ableitungen für $\Delta x, \Delta y$ *und* Δr linear. Um den Fehler zu minimieren, werden die partiellen Ableitungen Null gesetzt und ergeben sich zu:

$$\begin{aligned}
\tfrac{\partial H}{\partial \Delta x} &= 2 \textstyle\sum_{i=1}^{n}(h_i - \Delta x \cos(\theta_i) - \Delta y \sin(\theta_i) - \Delta r)\,(-\cos(\theta_i)) = 0 \\
\tfrac{\partial H}{\partial \Delta y} &= 2 \textstyle\sum_{i=1}^{n}(h_i - \Delta x \cos(\theta_i) - \Delta y \sin(\theta_i) - \Delta r)\,(-\sin(\theta_i)) = 0 \\
\tfrac{\partial H}{\partial \Delta r} &= 2 \textstyle\sum_{i=1}^{n}(h_i - \Delta x \cos(\theta_i) - \Delta y \sin(\theta_i) - \Delta r)\,(-1) = 0
\end{aligned} \tag{3.74}$$

Diese drei Gleichungen, ausgedrückt in Matrizenschreibweise:

$$\begin{bmatrix} \sum\cos^2(\theta_i) & \sum\sin(\theta_i)\cos(\theta_i) & \sum\cos(\theta_i) \\ \sum\sin(\theta_i)\cos(\theta_i) & \sum\sin^2(\theta_i) & \sum\sin(\theta_i) \\ \sum\cos(\theta_i) & \sum\sin(\theta_i) & N \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta r \end{bmatrix} = \begin{bmatrix} \sum h_i\cos(\theta_i) \\ \sum h_i\sin(\theta_i) \\ \sum h_i \end{bmatrix} \tag{3.75}$$

N ist die Anzahl der Messpunkte.

Die Gleichung 3.75 wird mit Hilfe der Cramer'schen Regel, einer einfache Methode die 3 Unbekannten aus 3 Gleichungen zu berechnen, nach Δx, Δy *und* Δr aufgelöst.

Der Mittelpunkt und der Radius werden korrigiert:

$$\begin{aligned}
x_{0,new} &= x_{0,old} + \Delta x \\
y_{0,new} &= y_{0,old} + \Delta y \\
r_{new} &= r_{old} + \Delta r
\end{aligned} \tag{3.76}$$

Mit den neuen Mittelpunktkoordinaten und dem neuen Radius kann die Kreissegmentanpassung solange iterativ wiederholt werden, bis der Gesamtfehler eine Toleranzgrenze unterschritten hat oder ein Iterationsabbruch erfolgt.

Diese Methode wurde in *Matlab*® mit den Routinen "FitArc.m" und "OptimizeCircle.m" umgesetzt.

3.4.6 Anpassen von Datenpunkten mit Splines

Liegen Datenpunkte so vor, dass sie keiner definierten mathematischen Funktion folgen, ist es mit Hilfe von Splines möglich, eine stetige Funktion den Datenpunkten anzunähern. Dies hat vor allem den Vorteil, dass die differenziellen Ableitungen ermittelt werden können. Ein weiterer Vorteil ist die Glättung von Ausreißern.

Kubische Splines: [10]

Die Splinefunktion besteht aus einer Zusammensetzung von Polynomen höherer Ordnung. Die Anzahl der Polynome wird durch die Anzahl der Knotenpunkte vorgegeben. Die Polynome sind an den Stützstellen stetig miteinander verbunden.
Die kubische Splinefunktion $S(x)$, zu den Stützstellen $x_0 < x_1 < \ldots < x_n$ und den dazugehörigen Stützwerten y_i, ist durch folgende Eigenschaften festgelegt:

$$\begin{aligned} &S(x_i) = y_i \quad (i = 0, 1, 2, \ldots, n) \\ &S(x) \textit{ ist ein Polynom der Ordnung } \leq 3 \\ &S''(x_0) = S''(x_n) = 0 \end{aligned} \tag{3.77}$$

Durch diese Bedingungen ist die Funktion S eindeutig bestimmt. Die Funktion ist zweimal stetig ableitbar, und an den Enden wird diese zweite Ableitung Null.

Zur numerischen Berechnung der Spline Interpolierenden $S(x)$ seien

$$h_i = x_{i+1} - x_i > 0 \quad (i = 0, 1, 2, \ldots n-1) \tag{3.78}$$

die Länge der Teilintervalle $[x_i, x_{i+1}]$, in welchen für $S(x)$ der Ansatz gelte:

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \ , \ x \in [x_i, x_{i+1}]. \tag{3.79}$$

Durch diesen Ansatz ist die Interpolationsbedingung und die Stetigkeit der zweiten Ableitung an den inneren Stützstellen berücksichtigt. Die Bedingung der Stetigkeit der ersten Ableitung an den $(n-1)$ inneren Stützstellen x_i, liefert die $(n-1)$ linearen Gleichungen:

$$\begin{aligned} a_i &= \tfrac{1}{6\,h_i}(y''_{i+1} - y''_i) \\ b_i &= \tfrac{1}{2}y''_i \\ c_i &= \tfrac{1}{h_i}(y_{i+1} - y_i) - \tfrac{h_i}{6}(y''_{i+1} + 2y''_i) \\ d_i &= y_i \end{aligned} \tag{3.80}$$

Unter Berücksichtigung von $y''_0 = y''_n = 0$ stellt dies ein lineares Gleichungssystem für die $(n-1)$ Unbekannten $y''_1, y''_2, \ldots, y''_{n-1}$ dar.

Diese Methode wird in *Matlab*® mit dem Befehl "spap2", mit der Bedingung für einen kubischen Spline, umgesetzt. Zusätzlich errechnet *Matlab*® jene Funktion mit dem geringsten quadratischen Abstand zu den Datenpunkten (least mean square fit).

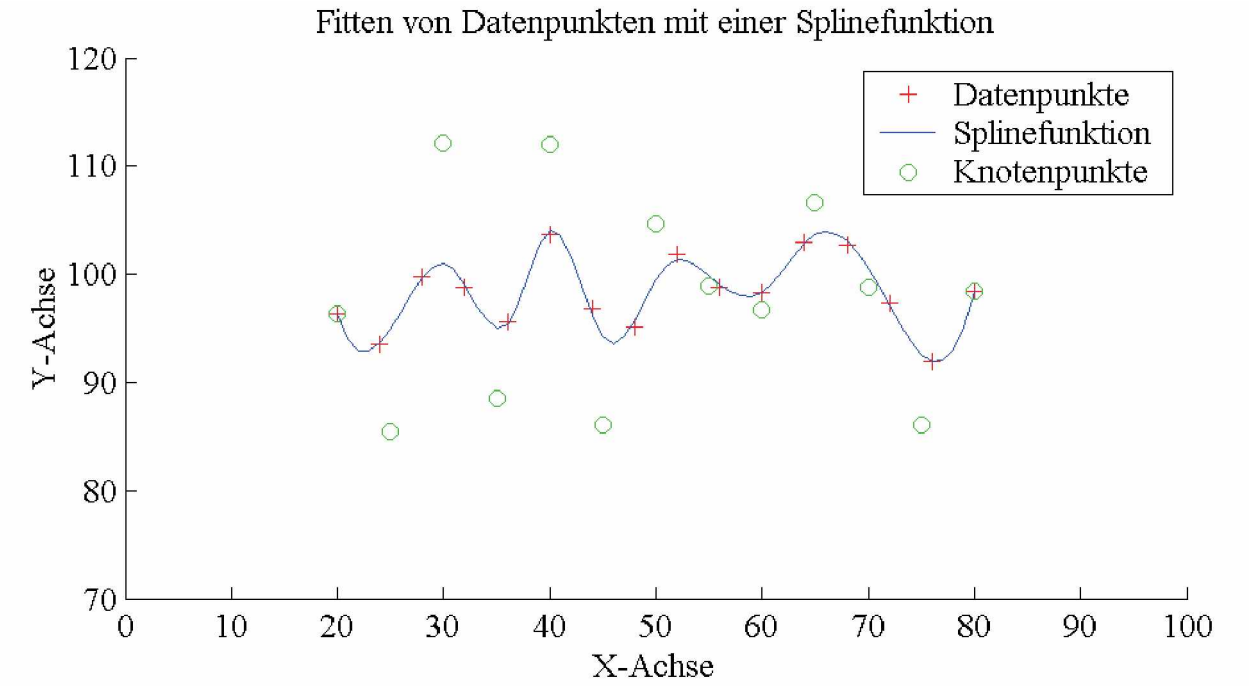


Abbildung 3.9: Beispiel einer Splinefunktion

In diesem Beispiel wird eine Splinefunktion den Datenpunkten bestmöglich angenähert. Dies geschieht in *Matlab*® folgendermaßen:

$$S = spap2(Knots, Order, x, y) \tag{3.81}$$

S ... ist der Rückgabeparameter in welchem die Splinefunktion enthalten ist.
$Knots$ ... ist die Anzahl der gewünschten Knotenpunkte.
$Order$ ... ist die Ordnung der Polynome zwischen den Knotenpunkten, in unserem Fall 3.
$[x, y]$ ... sind die Koordinaten der Datenpunkte.

Kapitel 4

Messaufbau

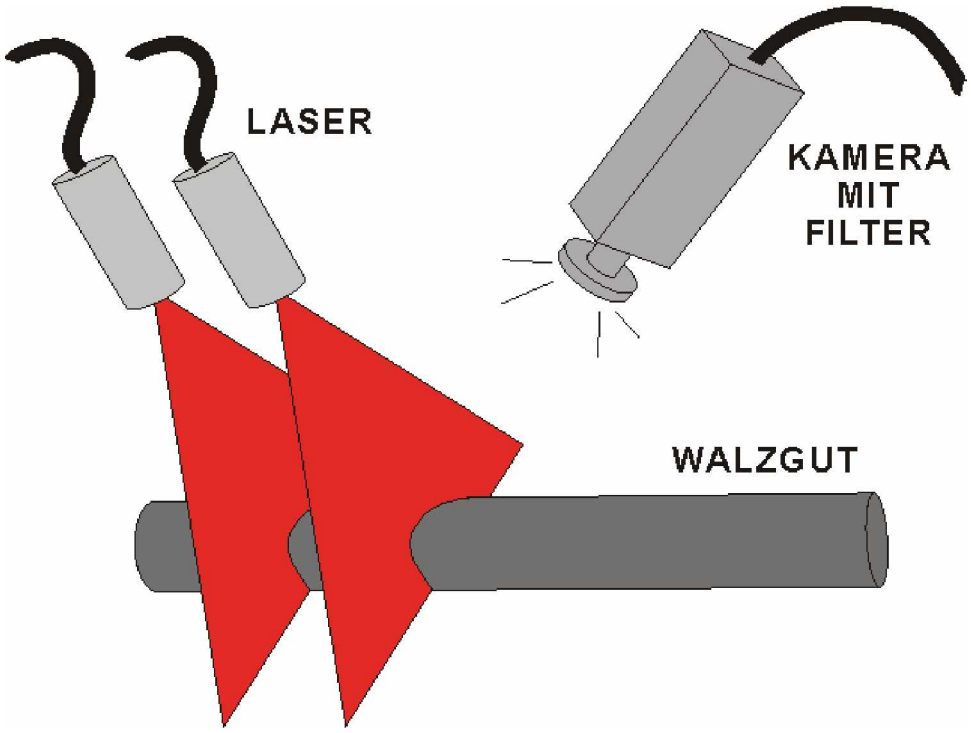


Abbildung 4.1: Messaufbau

Mit zwei Lasern werden zwei ungefähr parallele Linien auf das zu vermessende Walzgut projiziert. Angeordnet werden die Laser so, dass die projizierten Linien das Walzgut ungefähr in einem rechten Winkel zu seiner Längsachse schneiden. Der Abstand der Laser zum Walzgut wird so gewählt, dass sich die Linien auf dem Walzgut noch gut fokussieren lassen.

Die Kamera wird ebenfalls über dem Walzgut positioniert. Dabei müssen die beiden Laserschnittlinien auf dem Kamerabild zur Gänze sichtbar sein. Das Bild muss natürlich auch einwandfrei fokussierbar sein.

Die Wahl der Zylinderlinse vor dem Laser, welche den Laserstrahl in eine Linie aufspaltet, richtet sich nach der gewünschten Linienlänge in der gewünschten Entfernung. Die Laserlinie muss mindestens so lang sein, wie das zu vermessende Walzgut breit ist.
Zu lang sollte die Laserlinie jedoch nicht sein, da sich die Laserleistung auf die ganze Linie aufteilt.
Die Zylinderlinsen werden von den Laserherstellern in vielen Variationen angeboten und fix am Laser montiert ausgeliefert.

Für die Wahl des Kameraobjektivs gilt, dass die Laserschnittlinien vom Kamerabild erfasst und fokussiert werden müssen. Der zu vermessende Querschnitt sollte jedoch auch nicht zu klein auf dem Bild erscheinen.

Der Interferenzfilter vor dem Objektiv der Kamera soll so gewählt werden, dass die Wellenlänge der Laser ungehindert durchgeht und alle anderen absorbiert werden. Der Durchlässigkeitsbereich sollte klein gehalten werden.
Zur Montage des Interferenzfilters vor dem Objektiv wurde eine entsprechende Fassung angefertigt.

Damit die Messbilder auf der Festplatte des Computers gesichert werden können, wurde dieser mit einer Schnittstellenkarte ausgerüstet. Die dafür notwendige Software war der Schnittstellenkarte beigelegt.
Die Kamera ist mit einem Koaxialkabel mit der Karte verbunden.

Als Schutz vor der Temperatur am Messort wurden Laser und Kamera in je ein Gehäuse eingebaut. Die Kühlung dieser Gehäuse erfolgt mit Druckluft. Damit Laser und Kamera vor eventuell mit der Druckluft mitgeführten Öltröpfchen geschützt sind, besitzen die Gehäuse Doppelwände. Die Druckluft strömt dabei zwischen diesen Wänden und führt die Wärme ab, die optischen Komponenten befinden sich im Inneren. Vor Laser und Kamera sind Glasscheiben angebracht. Der Austritt der Druckluft erfolgt vor diesen Glasscheiben, wodurch Schmutzpartikel weggeblasen werden.
Druckluft ist die teuerste Energieform. Aus diesem Grund ist diese Art der Kühlung nur für Probemessungen und nicht für den Dauerbetrieb geeignet.
Die Gehäuse sind an Gelenke montiert, wodurch sie in alle Richtungen geschwenkt werden können.

Alle Kabel, wie Stromversorgung für Kamera und Laser, wurden in Isolierschläuche verpackt und deren Enden mit Glaswolle umwickelt, um sie vor thermischer Zerstörung zu schützen. Dazu könnten auch andere Materialien, die einen ausreichenden Schutz gewährleisten, verwendet werden.

4.1 Messaufbau über der Walzstraße

Abbildung 4.2: Messaufbau bei EWK

In Abbildung 4.2 wird der Messaufbau vor Ort gezeigt. Der Messbalken, an dessen Enden die Gehäuse für Kamera und Laser montiert sind, ist an einem Träger über der Walzstraße befestigt. Die Laufrichtung des zu vermessenden Walzgutes ist längs des Messbalkens. Im linken bzw. vorderen Gehäuse befinden sich die Laser, im rechten bzw. hinteren ist die CCD-Kamera untergebracht. Damit sich der Messbalken durch die hohe Temperatur nicht verformt bzw. verdreht, wurde er mit einer isolierenden Folie umhüllt. Weiters sind noch die Druckluftschläuche und die isolierten Kabel zu erkennen.

Vor der Messung wurde der Messbalken senkrecht über der Stelle am Träger positioniert, wo das Walzgut erwartet wurde. Da der Träger nur ca. 5 Meter vom Walzgerüst entfernt war, konnte diese Position ziemlich genau ausgemessen werden. Der Austritt des Walzgutes am Walzgerüst war bekannt. Während der Messung konnte die Position des Walzgutes mit dem Greifkanter geringfügig verändert werden. Dadurch konnte das Walzgut immer in der Mitte des aufgenommenen Bildes erfasst werden.

Der Computer wurde, geschützt vor Schmutz und Staub, in einem nahegelegenen Leitstand aufgebaut. Ein Koaxialkabel wurde vom Computer zur Kamera gelegt. Die Länge des Kabels betrug ca. 30 Meter und stellte bezüglich den Datenverlusten kein Problem dar.

4.2 Verwendete Hard- und Softwarekomponenten

Im Folgenden werden die wichtigsten, für die Durchführung dieser Diplomarbeit verwendeten Hard- und Softwarekomponenten angeführt.

Hardware:

Komponente	Zubehör	Hersteller	Technische Daten
Kamera	CCD-Kamera	Pulnix	Auflösung 768*576 Pixel
	Shutter	Pulnix	Belichtungszeitregler
	Objektiv	Cosmicar	Brennweite=25mm
	Interferenzfilter	CVI Laser	Wellenlänge=685nm
	Netzteil	Egston	In:220V∼, Out:12V=/1A
Laser	Laserdiode	Lasiris	Wellenlänge=685nm
			Leistung=43mW
	Zylinderlinse	Lasiris	13LR25-S500
	Netzteil	Schäfter&Kirchhoff	In:220V∼,Out:5V=/350mA
Computer	PC	NoName	min. Pentium 4
	IMAQ-Karte	National Instruments	4 Eingänge
Kalibrationsobjekt		Institut für Automation	

Software:

Programm	Version	Hersteller	Verwendung
Measurement & Automation	V2.0	National Instruments	Bildaufnahme
Matlab [3]	V6.1	The Math Works	Softwareerstellung & Ergebnisauswertung
WinEdt [4]	V5.1	Aleksander Simonic	Dokumentation
Corel Draw	V10.4	Corel Corporation	Dokumentation

Kapitel 5

Aufnahme der Kalibrations- bzw. Messbilder

Die Aufnahme der Kalibrations- bzw. Messbilder erfolgt mit dem Programm Measurement & Automation, Version V2.0 der Firma National Instruments.

5.1 Aufnahme eines Kalibrationsbildes

Nach erfolgter Montage von Laser und Kamera (Kapitel 4) werden deren Positionen durch Kalibration ermittelt.

Das Kalibrationsobjekt wird auf einer ebenen Fläche in der Walzstraße dort positioniert, wo später die Messung stattfindet. Dabei ist zu beachten, dass die drei Ebenen des Kalibrationsobjekts parallel zur Laufrichtung des Walzgutes verlaufen müssen.

Bei der Ausrichtung der Kamera muss darauf geachtet werden, dass die Breite des Kalibrationsobjekts zur Gänze auf dem Kamerabild sichtbar ist. Die gesamte Länge des Objekts muss nicht im Bild sein, jedoch mindestens drei Leuchtdiodenreihen. Das Bild wird scharf gestellt und die Kameraposition fixiert.

Beide Laser werden parallel ausgerichtet. Die Linien müssen die Ebenen des Kalibrationsobjekts ungefähr in einem rechten Winkel schneiden. Wichtig dabei ist, dass keine Laserlinie eine Leuchtdiode der hinteren oder vorderen Ebene schneidet. Ist dies der Fall, kann diese Leuchtdiode vom Algorithmus nicht mehr erfasst werden. Die Laserlinien werden ebenfalls fokussiert und deren Position fixiert.
Bevor die Laser eingeschaltet werden, wird auf die Gefährlichkeit dieser hingewiesen. Beide sind Produkte der Laserklasse 3B - Ein direkter Blick in die Strahlenquelle ist unbedingt zu vermeiden!

Sind alle Leuchtdioden und Laserlinien deutlich auf dem Kamerabild erkennbar, kann durch Drücken des Icons "Snap" ein digitales Bild aufgenommen werden. Das Bild muss als "Tag-

ged Image File Format", kurz "*.tif", auf dem Datenträger gesichert werden.

Die Kamera- und Laserposition dürfen nach erfolgter Aufnahme nicht mehr verändert werden. Das Kalibrationsobjekt wird für die Messung nicht benötigt und kann nach einer erfolgreichen Kalibration entfernt werden.

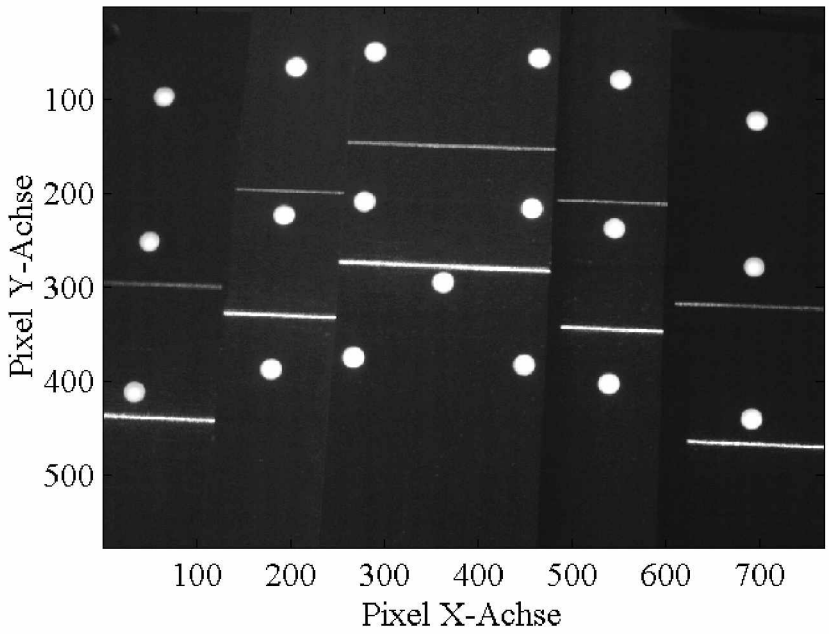


Abbildung 5.1: Kalibrationsbild

In Abbildung 5.1 ist ein typisches Kalibrationsbild zu erkennen. Das von der CCD-Kamera erzeugte Bild ist schwarzweiß und besteht aus $768 * 576$ Bildpunkten (Pixel).
Auf dem Bild sind die Leuchtdioden am Kalibrationsobjekt und die zwei Laserlinien zu erkennen. Dies sind auch die einzigen Bildinformationen, welche für die Kalibration erforderlich sind, alle weiteren erkennbaren Informationen sind Störeinflüsse und sollten, wenn möglich, herausgefiltert werden.
Die letzte Leuchtdiodenreihe ist im Bild nicht vorhanden und wird für die Kalibration auch nicht benötigt.
Die physikalischen Abmessungen des Kalibrationsobjekts werden im Anhang B veranschaulicht.

5.2 Aufnahme eines Messbildes

Damit die vom glühenden Walzgut emittierte Strahlung weitgehend absorbiert wird, muss nach erfolgter Kalibration der Interferenzfilter vor dem Kameraobjektiv montiert werden. Dabei ist zu beachten, dass die Kameraposition nicht verändert wird.

Da trotz Filter der Störeinfluss der emittierten Strahlung noch beträchtlich sein kann, muss die Belichtungszeit möglichst klein eingestellt werden. Dies geschieht mit dem elektronischen Shutter der Kamera. Die Einstellung soll so erfolgen, dass die Laserlinien noch eindeutig zu erkennen sind und die Störeinflüsse weitgehend verschwinden.

Die Messbilder werden, gleich wie die Kalibrationsbilder, durch Drücken des Icons "Snap" aufgenommen. Sie müssen ebenfalls als "Tagged Image File Format" gespeichert werden.

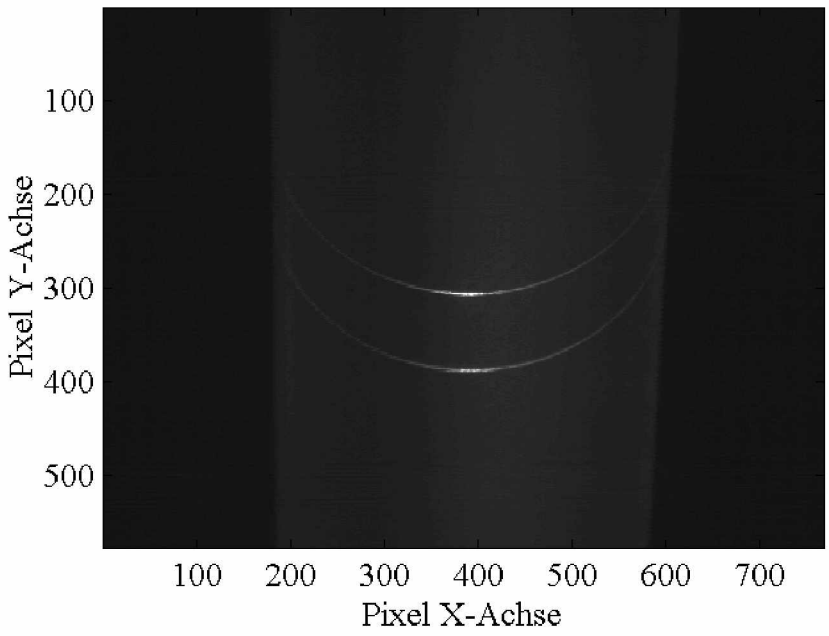


Abbildung 5.2: Messbild

Auf dem Messbild in Abbildung 5.2 sind die Schnittlinien der Laserebenen mit dem zu vermessenden Walzgut zu sehen. Die Temperatur des Walzgutes beträgt ca. 1000^oC, die Geschwindigkeit ca. $3 - 4m/s$.

Kapitel 6

Implementierung in Matlab

In diesem Kapitel wird die Umsetzung des Kalibrations- und Messverfahrens in *Matlab*® V6.1 behandelt.
Das erstellte Programm ist in ein Haupt- und mehrere Unterprogramme unterteilt. Im Hauptprogramm werden die Menüsteuerung und die wichtigsten Rechenoperationen abgewickelt. Die Unterprogramme sind Algorithmen, welche mehrmals vom Hauptprogramm aufgerufen werden.

Nach dem Start des Hauptprogramms "Main.m" gelangt man in das Steuermenü.

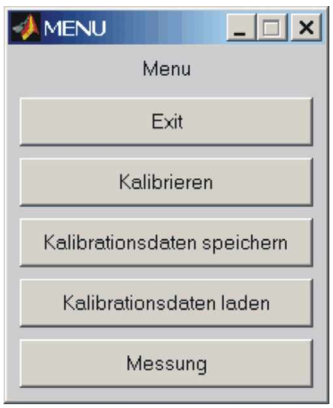


Abbildung 6.1: Steuermenü

- Exit - Verlassen des Programms
- Kalibration - Ermittlung der Kamera- und Laserposition
- Kalibrationsdaten speichern - Es werden die bei der Kalibration ermittelten Daten auf dem Datenträger gesichert
- Kalibrationsdaten laden - Die zuletzt gespeicherten Kalibrationsdaten werden geladen
- Messung - Es wird der Querschnitt eines zu vermessenden Objekts berechnet

6.1 Kalibration

6.1.1 Einlesen eines Kalibrationsbildes

Bevor die Kalibration durchgeführt werden kann, muss ein Kalibrationsbild vom Datenträger geladen werden. Dies geschieht, indem man mit der Maus das Verzeichnis auswählt, in welchem das Bild gespeichert wurde, und das Bild lädt. Das Bild, abgelegt in der Variable "**Pic0**", wird in ein schwarzweiß Bild verwandelt. Diese Variable ist eine 768*576 Matrix. Dies bedeutet, dass in x-Richtung 768 und in y-Richtung 576 Pixel im Pixelkoordinatensystem für die Helligkeitsinformation zu Verfügung stehen. Diese Helligkeitsinformation wird durch einen Wert vom Typ Doubleinteger (16bit) beschrieben. Es stehen somit 65536 verschiedene Graustufen zwischen 0 und 1 zur Verfügung. Dabei bedeutet 0 schwarz und 1 weiß.

6.1.2 Erkennung und Zuweisung der Leuchtdioden

Das Ziel dieses Programmteils ist die Erkennung und in weiterer Folge die Zuweisung von Pixelkoordinaten zu jedem Mittelpunkt einer Leuchtdiode. Dazu wird ein Pixelkoordinatensystem mit seinem Ursprung in der linken oberen Ecke im Bild eingeführt.

Zuerst wird im Bild nach Konturen mit einem bestimmten Schwellenwert gesucht. Dieser Schwellenwert, welcher zwischen 0 und 1 liegen muss, gibt jene minimale Helligkeit vor, ab welcher nach Bildpunkten gesucht wird. Alle Bildpunkte, welche heller als der Schwellenwert sind, werden erfasst und umrandet. Es werden zwei Schwellenwerte (0.20 und 0.30) definiert und die gefundenen Konturen in Variablen gesichert. Die Schwellenwerte werden am Programmanfang mit Konstanten definiert und können bei Kalibrationsbildern mit einem anderen Kontrast verändert werden.

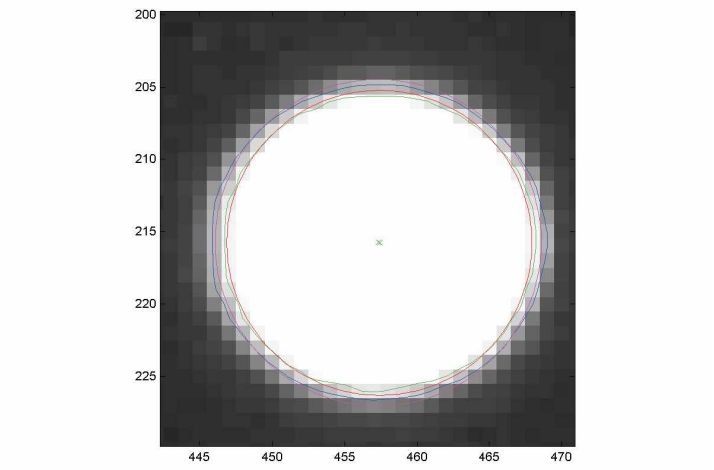


Abbildung 6.2: Erfassung einer Leuchtdiode

In der Abbildung 6.2 wird eine Leuchtdiode mit den zwei Schwellenwerten erfasst. Die innere Kontur (grün) und der angepasste Kreis (rot) wurden für den helleren Schwellenwert gefunden, die blaue Kontur, mit dem errechneten rosa Kreis, für den dunkleren.

Die Wahl der Schwellenwerte ist für die Richtigkeit der Kalibration von entscheidender Bedeutung. Die Schwellenwerte sollten so gewählt werden, dass die dadurch gefundenen Konturen exakt den Umrissen der Leuchtdioden entsprechen. Werden die Schwellenwerte zu hoch angesetzt, das heißt es werden nur sehr helle Konturen erfasst, so kann es passieren, dass nur das helle Zentrum einer Leuchtdiode erfasst wird. Dieses helle Zentrum liegt nicht bei jeder Leuchtdiode genau im Mittelpunkt und die Position der Leuchtdiode am Kalibrationsobjekt wird falsch ermittelt.

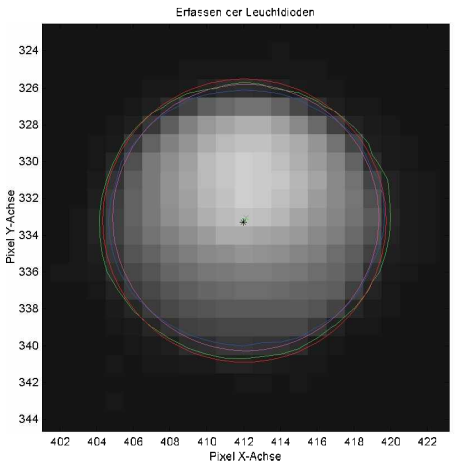


Abbildung 6.3: Richtige Schwellenwerte

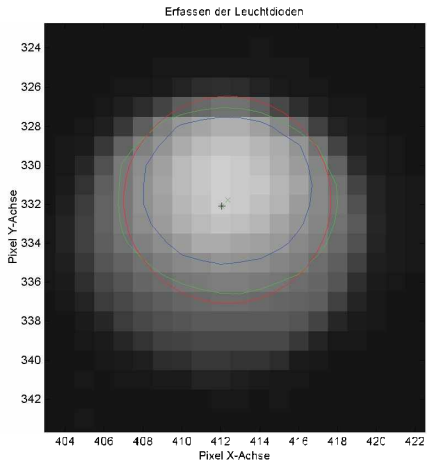


Abbildung 6.4: Falsche Schwellenwerte

In den Abbildungen 6.3 und 6.4 wird die gleiche Leuchtdiode mit verschiedenen Schwellenwerten erfasst. Im linken Bild (Schwellenwerte = 0.20 bzw 0.30) wird der Mittelpunkt der Leuchtdiode richtig gefunden, im rechten Bild (Schwellenwerte = 0.60 bzw 0.70) wird nur das helle Zentrum gefunden und der daraus resultierende Mittelpunkt ist falsch. Wird die Position von einer oder mehreren Leuchtdioden falsch ermittelt, so kann der, bei der anschließenden Messung gemachte Fehler nicht mehr korrigiert werden. Dieser Fehler kann mehr als ein Prozent des Ergebnisses ausmachen.
Damit die im Kalibrationsbild gefundenen Mittelpunkte nicht einzeln auf ihre Richtigkeit hin überprüft werden müssen, werden die falsch ermittelten Mittelpunkte rot eingekreist und eine Warnung ausgegeben.
Die Schwellenwerte dürfen auch nicht zu klein gewählt werden, da sonst sehr viele Konturen gefunden werden.

Alle Konturen, welche nicht eine Leuchtdiode beschreiben, sind Störeinflüsse und müssen aussortiert werden. Die Konturen werden untersucht, ob es sich dabei um einen Kreis handeln kann.

1. Es wird eine grobe Abschätzung durchgeführt, wobei das Proportionsverhältnis überprüft wird. Die Ausdehnung der Kontur in x-Richtung wird durch die Ausdehnung in y-Richtung dividiert und das Ergebnis mit Toleranzfaktoren verglichen. Bei einem idealen Kreis ist das Verhältnis der Breite zur Höhe immer 1.
   Die Toleranzgrenzen wurden wie folgt festgelegt:
$$0.7 < \frac{x - Abmessung}{y - Abmessung} < 1.4 \tag{6.1}$$
2. Den so ermittelten, annähernd runden Objekten, werden mit der in Kapitel 3.4.4 beschriebenen Methode "FitCircle.m" Kreise angepasst. Die Radien, in Pixel, für jeden Kreis werden wieder mit festgelegten Toleranzgrenzen verglichen:
$$5\,Pixel < Radius < 15\,Pixel \tag{6.2}$$
3. Um die Fehlerquelle, einen Störeinfluss fälschlicherweise als Leuchtdiode zu erkennen zu minimieren, werden die gefundenen Kreise nochmals überprüft. Es wird der Mittelwert des Fehlers aus der Kreisanpassung, und der Mittelwert aller Radien der Objekte berechnet. Die Fehler und die Radien der angepassten Kreise werden wieder mit Toleranzgrenzen verglichen.

Alle Objekte, welche diese Toleranzkriterien nicht erfüllen, werden verworfen. Dadurch kann es vorkommen, dass bei schlechten Kalibrationsbildern ein paar Leuchtdioden nicht gefunden werden. Es werden aber nicht alle 25 Leuchtdioden zur Kalibration benötigt, ein falscher Punkt jedoch macht die Kalibration unmöglich. Bei vielen durchgeführten Versuchen ist nie ein Störeinfluss fälschlicherweise als Leuchtdiode identifiziert worden.

Die Mittelpunktkoordinaten für beide Schwellenwerte aller als Leuchtdioden identifizierten Objekte, werden in der Matrix "KreisMittelpunkte_xy" abgelegt. Existieren für eine Leuchtdiode zwei Mittelpunkte, so werden ihre x- und y- Koordinaten gemittelt, andernfalls bleibt der Mittelpunkt gleich.

Die x- und y-Koordinaten aller Mittelpunkte müssen nun sortiert werden. Zu diesem Zweck werden die Leuchtdioden auf dem Kalibrationsobjekt in eine Matrix eingeteilt. Nur anhand dieser Matrix ist die Position einer Leuchtdiode eindeutig zuweisbar.

Abbildung 6.5: Kalibrationsobjekt

$$\begin{bmatrix} x & x & x & 0 & x & x & x \\ x & x & x & 0 & x & x & x \\ 0 & 0 & 0 & x & 0 & 0 & 0 \\ x & x & x & 0 & x & x & x \\ x & x & x & 0 & x & x & x \end{bmatrix} \tag{6.3}$$

Jedes x in der Matrix 6.3 entspricht einer Leuchtdiode in Abbildung 6.5.

Zuweisung der Leuchtdioden in die richtigen Spalten der Matrix:

Es werden die x-Koordinaten der Mittelpunkte der Größe nach sortiert und in einer Liste gespeichert. Dazu wird die Routine "Bubblesort.m" aufgerufen.
Die Routine "Bubblesort.m" funktioniert folgendermaßen:

$$\begin{array}{cccccc} i=1 & & i=2 & & i=3 & \qquad\qquad i=n \\ \\ 5 & & 3 & & 3 & \qquad\qquad 2 \\ 3 & & 5 & & 2 & \qquad\qquad 3 \\ 7 & \Rightarrow & 2 & \Rightarrow & 5 & \Rightarrow \ldots \Rightarrow \quad 5 \\ 2 & & 7 & & 7 & \qquad\qquad 7 \\ 9 & & 9 & & 9 & \qquad\qquad 9 \\ \vdots & & \vdots & & \vdots & \qquad\qquad \vdots \end{array} \tag{6.4}$$

Eine Liste mit n Einträgen wird $i = 1, 2, \ldots, n$-mal untersucht. In jedem Durchlauf wird die Liste vom ersten bis zum $(n-i)$-ten Eintrag durchlaufen. Dabei wird der aktuelle Eintrag immer mit dem nächsten verglichen. Ist der aktuelle größer als der nächste, so werden beide vertauscht. Nach n Durchläufen ist die Liste der Größe nach sortiert.

Anhand der nach der Größe sortierten x-Koordinaten werden die Leuchtdioden in die zugehörigen Spalten in die gesuchte Matrix eingeteilt (Gleichung 6.3). Der Mittelpunkt mit dem kleinsten x-Wert muss in der ersten Spalte stehen, und wird in die erste Zeile der ersten Spalte geschrieben. Die verbleibenden Mittelpunkte in der sortierten Liste werden nun der Reihe nach untersucht.
Durch den ersten Punkt in jeder Spalte und den nächst größeren in der sortierten Liste wird eine Gerade mit der Routine "FitLine.m" berechnet (Kapitel 3.4.1). Aus den so ermittelten "planner line" Koordinaten der Geraden, wird deren Neigung zur Vertikalen berechnet. Ist diese Neigung kleiner als 10^o, wird der Punkt in diese Spalte geschrieben. Für den nächst größeren Punkt in der Liste wird der Normalabstand zu dieser Geraden berechnet. Ist dieser Abstand kleiner als ein festgelegter Toleranzwert, so gehört dieser Punkt ebenfalls in die gleiche Spalte. Sind jedoch die Neigung oder der Normalabstand größer als die Toleranzwerte, so ist der untersuchte Mittelpunkt der erste Punkt der nächsten Spalte. In dieser neuen Spalte wird wiederum eine Gerade berechnet, und die folgenden Punkte mit den Toleranzwerten verglichen. Dies wird solange wiederholt, bis dem Punkt mit der größten x-Koordinate eine Spalte zugewiesen wurde. Diese Toleranzwerte sind in den Konstanten "MaxNeigung" und "Abweichung" festgeschrieben.

Die so erzeugte Matrix muss noch auf ihre Richtigkeit überprüft werden.
Für die Kalibration ist es unbedingt erforderlich, dass die mittlere Leuchtdiode gefunden wird. Eine weitere Bedingung ist, dass auf der hinteren und der vorderen Ebene mindestens vier Leuchtdioden vorhanden sind. Wie in Abbildung 6.5 ersichtlich, muss die Matrix demnach sieben Spalten besitzen. (Die Leuchtdioden auf der mittleren Ebene müssen aus Symmetriegründen gefunden werden.) Ist dies nicht der Fall, wird eine Fehlermeldung am Bildschirm ausgegeben und die Kalibration abgebrochen.

Zuweisung der Leuchtdioden in die richtigen Zeilen der Matrix:

Den Leuchtdioden muss nun eindeutig die richtige Position in den einzelnen Spalten zugeordnet werden. Eine einfache Lösung wäre die Sortierung der Mittelpunkte der Größe nach in y-Richtung. Für den Fall, dass immer alle Leuchtdioden gefunden werden, würde diese Methode auch das gewünschte Ergebnis liefern. Ist jedoch eine Leuchtdiode defekt oder verschmutzt, so ist die Position aller weiteren in einer Spalte falsch, und die Kalibration kann nicht mehr durchgeführt werden. Um dies zu vermeiden, werden die y-Positionen der Mittelpunkte zeilenweise verglichen.
Durch die schiefe Betrachtung des Kalibrationsobjekts mit der Kamera liegen die Leuchtdioden in den jeweiligen Zeilen nicht mehr auf einer Linie. Die Leuchtdioden auf den höheren, mittleren und vorderen Ebenen haben kleinere y-Koordinaten als jene auf der hinteren Ebene (Abbildung 5.1). Darum wird eine um diesen Versatz korrigierte Hilfsmatrix errechnet. Es wird die Verschiebung der mittleren zur hinteren Ebene und der vorderen zur mittleren Ebene berechnet. Diese Verschiebung wird zu den y-Koordinaten der Mittelpunkte in den entsprechenden Spalten addiert. In dieser Hilfsmatrix wird nun zeilenweise der Mittelwert berechnet. Ist der Abstand der y-Koordinate eines Punktes zu diesem Mittelwert zu groß, wird der Punkt in eine neue Zeile geschrieben.
Durch diese Methode wird jeder Leuchtdiode immer die richtige Position zugewiesen, auch wenn einige Leuchtdioden nicht gefunden wurden. Ist dies der Fall, so wird den x und y-Koordinaten der Wert Null zugewiesen.

Diese Matrizen für die x- und y-Koordinaten der Mittelpunkte sind für das Kalibrationsbild in Abbildung 5.1 veranschaulicht:

$$LEDMatrixX = \begin{bmatrix} 98.0 & 206.3 & 290.3 & 0 & 465.1 & 551.6 & 697.0 \\ 50.0 & 193.2 & 278.9 & 0 & 457.4 & 545.6 & 694.1 \\ 0 & 0 & 0 & 362.9 & 0 & 0 & 0 \\ 33.4 & 179.4 & 267.5 & 0 & 449.3 & 539.6 & 691.4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$LEDMatrixY = \begin{bmatrix} 90.8 & 65.2 & 49.0 & 0 & 55.8 & 78.9 & 123.2 \\ 251.9 & 222.9 & 208.2 & 0 & 215.7 & 237.2 & 278.7 \\ 0 & 0 & 0 & 294.1 & 0 & 0 & 0 \\ 411.9 & 386.6 & 374.4 & 0 & 382.0 & 402.4 & 440.0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (6.5)$$

6.1.3 Berechnung der Transformationsmatrix zwischen dem Pixel- und dem Weltkoordinatensystem

Es wird ein Weltkoordinatensystem mit seinem Ursprung im Mittelpunkt der linken, oberen Leuchtdiode eingeführt. In diesem Koordinatensystem werden allen Leuchtdioden Koordinaten in Millimeter zugewiesen. Diese Abstände ergeben sich aus den Abmessungen des Kalibrationsobjekts (Anhang B). Sie werden, getrennt nach ihren x- und y-Koordinaten, in

eine Matrix geschrieben.

$$PhysikX = \begin{bmatrix} 0 & 35 & 55 & 0 & 95 & 115 & 150 \\ 0 & 35 & 55 & 0 & 95 & 115 & 150 \\ 0 & 0 & 0 & 75 & 0 & 0 & 0 \\ 0 & 35 & 55 & 0 & 95 & 115 & 150 \\ 0 & 35 & 55 & 0 & 95 & 115 & 150 \end{bmatrix} \quad PhysikY = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 40 & 40 & 40 & 0 & 40 & 40 & 40 \\ 0 & 0 & 0 & 60 & 0 & 0 & 0 \\ 80 & 80 & 80 & 0 & 80 & 80 & 80 \\ 140 & 140 & 140 & 0 & 140 & 140 & 140 \end{bmatrix} \tag{6.6}$$

Wie in Abbildung 6.5 ersichtlich, ist das Kalibrationsobjekt bezüglich der y-Achse nicht symmetrisch. Der Abstand zwischen der ersten und der zweiten Leuchtdiodenreihe ist nicht gleich dem Abstand zwischen den beiden letzten Reihen. Damit die richtigen Abstände im Weltkoordinatensystem zugewiesen werden können, muss die Ausrichtung des Kalibrationsobjekts ermittelt werden.
Zu diesem Zweck wird der Mittelwert der y-Koordinaten jeder Leuchtdiodenreihe berechnet. Ist der Abstand zwischen zwei benachbarten Reihen deutlich größer als die Abstände der übrigen Reihen, kann die Position bestimmt werden.
Aus der Matrix 6.5 für die y-Koordinaten erhält man für diese Abstände:

$$\begin{aligned} mean(Reihe\,2) - mean(Reihe\,1) &= 235.8 - 77.2 \;\; = 158.6 \\ mean(Reihe\,4) - mean(Reihe\,2) &= 399.6 - 235.8 = 163.8 \end{aligned} \tag{6.7}$$

Der Abstand zwischen der zweiten und der vierten Reihe ist, unabhängig von der Ausrichtung, immer gleich. In diesem Beispiel ist der Abstand zwischen der ersten und der zweiten Reihe ungefähr gleich dem Abstand zwischen der zweiten und der vierten Reihe. Aus diesem Grund muss, auch wenn die letzte Leuchtdiodenreihe im Kalibrationsbild nicht enthalten ist, der Abstand zu dieser größer sein. Somit ist die Ausrichtung immer eindeutig feststellbar und die y-Koordinaten können im Weltkoordinatensystem zugewiesen werden.

Bevor die Transformationsmatrix errechnet werden kann, muss noch geprüft werden, ob genügend Leuchtdioden für diese Berechnung vorhanden sind. Es müssen mindestens vier Leuchtdioden auf zwei verschiedenen Ebenen bekannt sein.
Es wird auf der vorderen und der hinteren Ebene geprüft, ob mindestens je vier Leuchtdioden vorhanden sind. Ist dies nicht der Fall, so wird eine Fehlermeldung ausgegeben und die Kalibration abgebrochen. Eine mögliche Kalibration mit der vorderen und mittleren Ebene wird in diesem Fall nicht durchgeführt, da der Höhenunterschied beider Ebenen nicht so groß ist, wodurch sich Messfehler deutlicher auswirken.

Sind genügend Leuchtdioden vorhanden, können die Projektionsmatrizen für die jeweiligen Ebenen nach der Projektionsvorschrift (Kapitel 3.1.3) berechnet werden. Die Rückgabeparameter, nach zweimaligem Aufruf der Routine "GenProj.m", sind die beiden Projektionsmatrizen "$\mathbf{H_{Vorne}}$" und "$\mathbf{H_{Hinten}}$".

Anhand dieser Projektionsmatrizen ist es möglich, jeden beliebigen Punkt auf den jeweiligen Ebenen von einem Koordinatensystem in das andere zu transformieren.

Nach dieser Berechnung werden die zuvor im Kalibrationsbild gefundenen Leuchtdioden nicht mehr benötigt und gelöscht. Dazu werden die Intensitäten an den entsprechenden Stellen in der Kalibrationsbildmatrix "**Pic0**" Null (=Schwarz) gesetzt. Dadurch wird verhindert, dass die Leuchtdioden die nachfolgende Auffindung der Laserlinien beeinflussen.

In Abbildung 6.6 wird das Ergebnis veranschaulicht.

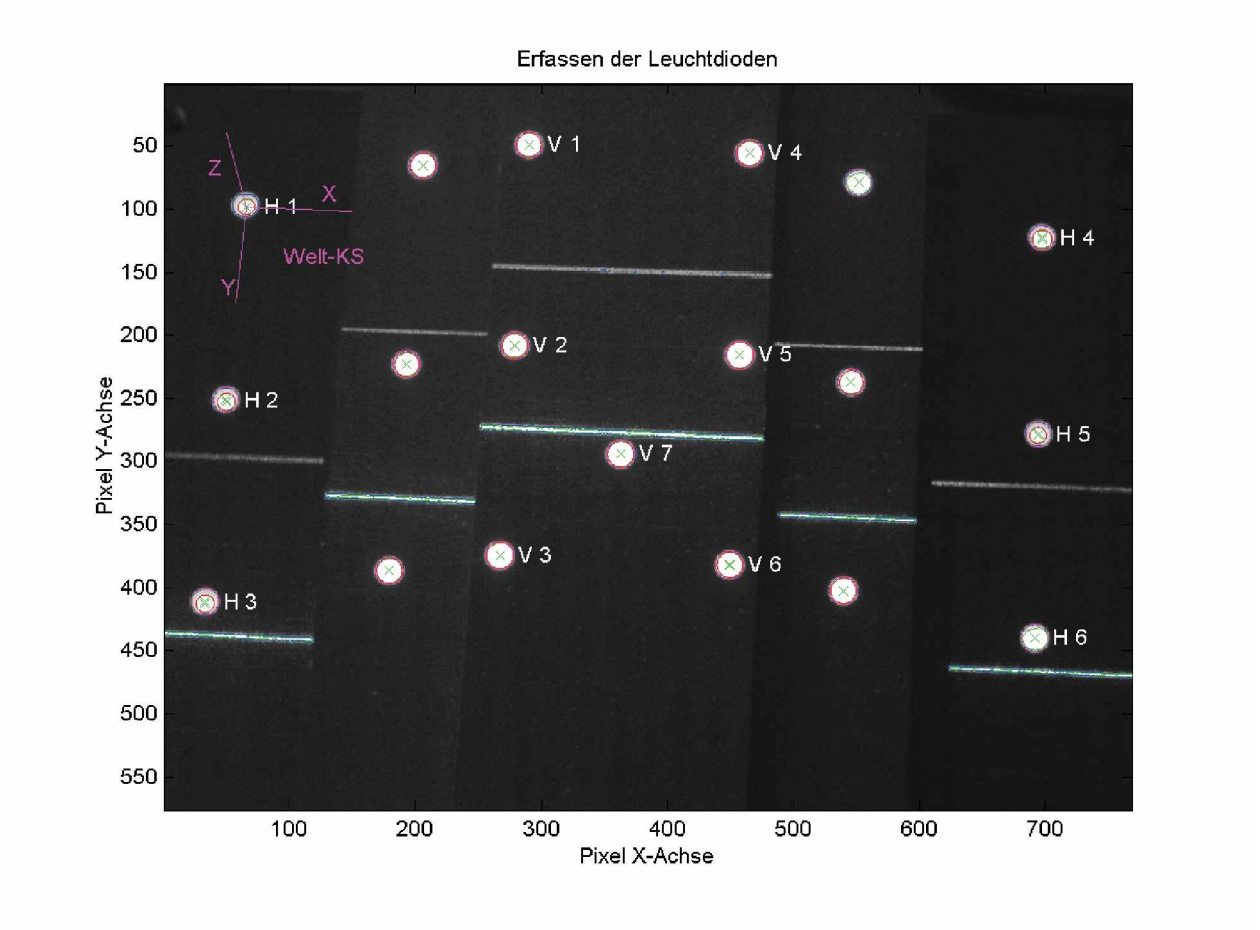


Abbildung 6.6: Darstellung der gefundenen Leuchtdioden

- Das Pixelkoordinatensystem, mit dem Ursprung in der linken, oberen Ecke des Bildes
- Das Weltkoordinatensystem, mit dem Ursprung im Mittelpunkt der linken, oberen Leuchtdiode.
- Die Leuchtdioden auf der hinteren Ebene ($H1$ *bis* $H6$).
- Die Leuchtdioden auf der vorderen Ebene ($V1$ *bis* $V7$).

6.1.4 Finden der Laserschnittlinien im Bild

Das Ziel dieser Routine ist die Zuordnung von Pixelkoordinaten für jeden Punkt der Laserlinien und die Trennung dieser.

Bei der Suche nach den Laserlinien wird nach den maximalen Intensitäten in jeder Spalte im Bild gesucht. Dabei wird davon ausgegangen, dass es sich bei den maximalen Intensitäten um Laserpunkte handelt. Diese Positionen können in *Matlab*® leicht ermittelt werden:

$$[Intensitaet, Pointer] = max(Pic1(:, Spalte)) \tag{6.8}$$

Die Variable "Intensitaet" gibt den Wert der maximalen Intensität an und die Variable "Pointer" die Zeile, in welcher die maximale Intensität auftritt.
Nachdem ein Laserpunkt gefunden und seine Position (Zeile, Spalte) gespeichert wurde, wird die Intensität an dieser Stelle Null (=schwarz) gesetzt. Dadurch wird verhindert, dass ein Laserpunkt bei nochmaliger Suche ein zweites mal gefunden wird.
Eine erneute Suche in der gleichen Spalte liefert die Position der zweiten Laserlinie.
Da beide Laserlinien die gleiche Intensität besitzen, müssen die gefundenen Laserpunkte getrennt werden. Die y-Koordinaten der Punkte werden spaltenweise verglichen - die Punkte mit dem kleineren Wert werden der ersten Laserlinie zugeordnet, der Rest der zweiten.

Da aber nicht in jeder Spalte beide Laserstrahlen existieren müssen, sind auch noch "Ausreißer" enthalten. Diese werden folgendermaßen aussortiert.
Damit eine Laserlinie eindeutig als solche identifiziert wird, werden zwei Kriterien festgelegt. Werden diese Kriterien nicht erfüllt, so werden die Laserpunkte verworfen.

- Jede Laserlinie wird in Segmente unterteilt. Ist ein Lasersegment kürzer als 30 Pixel, so handelt es sich wahrscheinlich um einen "Ausreißer" und wird verworfen. Diese Länge wird mit der Konstanten "MinSegmentlaenge" am Anfang dieser Routine festgelegt.
- Ein Lasersegment endet, wenn die vertikale Abweichung zweier benachbarter Laserpunkte mehr als 5 Pixel beträgt. Dieser Wert wird mit der Konstante "MaxAbweichung" festgelegt. Wird dieser Konstanten ein größerer Wert zugewiesen, dann ist es möglich, dass zwei unabhängige Linien miteinander verknüpft werden.

Die Koordinaten der Liniensegmente und deren Länge werden wieder in Variablen abgespeichert. Um die genaue Position der Lasersegmente zu ermitteln, wird der Intensitätsschwerpunkt in jeder Spalte berechnet (Kapitel 6.1.5). Mit dieser Routine wird das 1. Moment der Intensität des Liniensegments berechnet.

Diese Routine wird auch zum Auffinden der Laserlinien während der Messung verwendet und wurde in *Matlab*® unter dem Nahmen "FindLaserLines.m" erstellt. Wird die Kalibration bzw. Messung mit nur einem Laser durchgeführt, muss die Routine "FindLaserLines_old.m" verwendet werden.

6.1.5 Berechnung des 1. Moments der Intensität [11]

In der vorherigen Routine wurde die Position der Laserlinie im Bild festgestellt. Es wurde nach den maximalen Intensitäten gesucht und so deren Positionen ermittelt.
Die Breite einer Laserlinie umfasst jedoch immer mehrere Pixel. Damit das Zentrum dieser Linie exakt erfasst werden kann, wird in diesem Bereich das erste Intensitätsmoment berechnet.
Die Position des ersten Intensitätsmoments kann folgendermaßen berechnet werden:

$$x_m = \frac{\sum_{x=x_{max}-m}^{x=x_{max}+m} \sum_{y=y_{max}-n}^{y=y_{max}+n} x\ I^p(x,y)}{\sum_{x=x_{max}-m}^{x=x_{max}+m} \sum_{y=y_{max}-n}^{y=y_{max}+n} I^p(x,y)} \tag{6.9}$$

$$y_m = \frac{\sum_{x=x_{max}-m}^{x=x_{max}+m} \sum_{y=y_{max}-n}^{y=y_{max}+n} y\ I^p(x,y)}{\sum_{x=x_{max}-m}^{x=x_{max}+m} \sum_{y=y_{max}-n}^{y=y_{max}+n} I^p(x,y)} \tag{6.10}$$

In diesen Gleichungen bedeuten:

x_m, y_m	... Position des ersten Intensitätsmoments
x_{max}, y_{max}	... Position des Intensitätsmaximums
$I(x,y)$	... Intensität an der Stelle (x,y)
p	... Gewichtungsfaktor der Intensität
m	... horizontaler Abstand in Pixel, welcher in die Berechnung miteinbezogen wird
n	... vertikaler Abstand in Pixel, welcher in die Berechnung miteinbezogen wird

Die Fläche, welche für die Berechnung verwendet wird, ist durch die Konstanten "m" und "n" bestimmt. Das bedeutet, ausgehend von der Position der maximalen Intensität in jeder Spalte, wird eine Fläche in x-Richtung $\pm\ m$ und in y-Richtung $\pm\ n$ aufgespannt. Im Programm sind diese Konstanten mit $m = 3$ und $n = 5$ festgelegt. Der vertikale Abstand wurde größer festgelegt, da die Laserlinie mehrere Pixel breit sein kann und dadurch zur Gänze erfasst wird.
Dem Gewichtungsfaktor wird der Wert $p = 3$ zugewiesen. Je höher dieser Faktor ist, umso stärker werden hellere Punkte gewichtet.

Diese Methode berechnet das Intensitätsmoment sowohl in horizontaler als auch in vertikaler Richtung. Eine früher erstellte Routine "FindLaserLines_old.m" berechnet das Intensitätsmoment nur in vertikaler Richtung. Die Ergebnisse dieser neuen Berechnungsmethode sind deutlich besser. Der Algorithmus wurde für die Messung mit zwei Lasern erstellt und funktioniert bei Verwendung von nur einem Laser nicht. In diesem Fall muss die früher erstellte Routine verwendet werden.

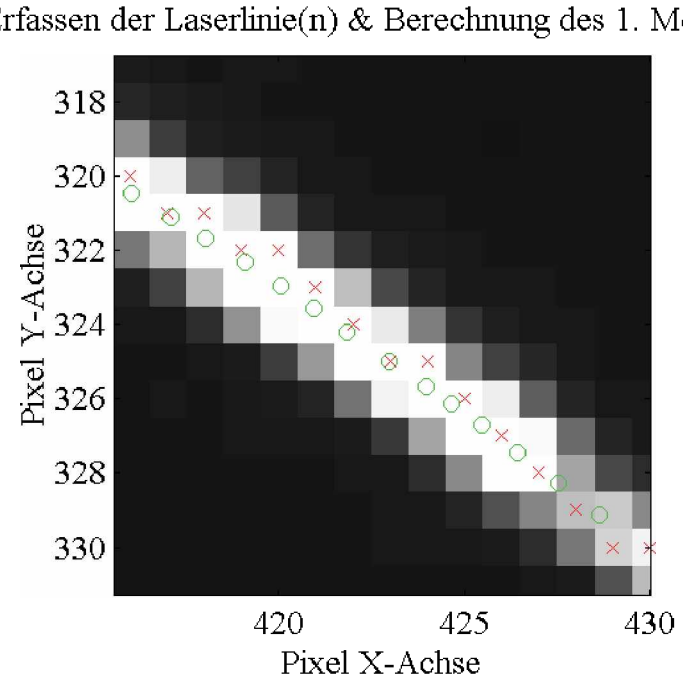


Abbildung 6.7: Berechnung des ersten Intensitätsmoments

In Abbildung 6.7 ist ein Ausschnitt einer Laserlinie veranschaulicht. Die roten "x" kennzeichnen die Positionen der maximalen Intensitäten der Laserlinie, die grünen "o" sind die daraus berechneten Intensitätsmomente.

6.1.6 Bestimmung der Position des Laserkoordinatensystems

Alle im Bild gefundenen Laserlinien bzw. Laserpunkte liegen in Pixelkoordinaten vor. Zur Bestimmung des Laserkoordinatensystems müssen diese Punkte in das Weltkoordinatensystem transformiert werden. Dies geschieht, auf den jeweiligen Ebenen, mit den in Kapitel 6.1.3 berechneten Transformationsmatrizen.
Die Höhen der einzelnen Ebenen sind durch die Abmessungen des Kalibrationsobjekts bekannt und werden bei den Vektoren als z-Koordinate angegeben.

$$P_{Pixelkoordinaten} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad , \quad P_{Weltkoordinaten,\, nichtaffin} = \mathbf{H}_{Ebene}\, P_{Pixelkoordinaten}$$
$$P_{Weltkoordinaten,\, affin} = \begin{bmatrix} \frac{P_{x,\, nichtaffin}}{P_{z,\, nichtaffin}} \\ \frac{P_{y,\, nichtaffin}}{P_{z,\, nichtaffin}} \\ Z_{Ebene} \end{bmatrix} \tag{6.11}$$

Da nun die Positionen der Laserpunkte im Weltkoordinatensystem bekannt sind, können für beide Laser die Laserebenen berechnet werden. Dies geschieht mit der Routine "FitLaserPlane.m" welche im Kapitel 3.4.3 beschrieben wurde. Als Rückgabewerte erhält man

die Ebenenkoordinaten ($E_1, E_2, E_3 \, und \, E_4$) der Ebenengleichung:

$$E_1 \, x \, + \, E_2 \, y \, + \, E_3 \, z \, + \, E_4 \, = \, 0 \tag{6.12}$$

Bestimmung der Transformationsmatrix zwischen dem Welt- und dem Laserkoordinatensystem:

Es wird ein zweidimensionales Laserkoordinatensystem definiert. Der Ursprung dieses Koordinatensystems soll im Schnittpunkt der Laserebene mit der y-Achse des Weltkoordinatensystems liegen. Seine x- und y-Achse sollen so gedreht werden, dass sie in der Laserebene liegen. Als Maßeinheit werden, wie im Weltkoordinatensystem, Millimeter verwendet.

Aus der Ebenengleichung 6.12 und der Lage des Ursprungs des Laserkoordinatensystem, können die translatorischen Anteile folgendermaßen berechnet werden:

$$E_1 \, 0 \, + \, E_2 \, y \, + \, E_3 \, 0 \, + \, E_4 \, = \, 0 \;\; , \;\; x \, = \, z \, = \, 0 \tag{6.13}$$

Durch Umformung dieser Gleichung erhält man:

$$y \, = \, -\frac{E_4}{E_2} \tag{6.14}$$

Daraus folgt für die translatorischen Anteile:

$$\Delta x \, = \, x \, = \, 0 \;\; , \;\; \Delta y \, = \, y \, = \, -\frac{E_4}{E_2} \;\; , \;\; \Delta z \, = \, z \, = \, 0 \tag{6.15}$$

Die rotatorischen Anteile errechnen sich aus den Winkeln ($\phi, \theta \, und \, \psi$), welche die drei Achsendrehungen beschreiben.

Das Prinzip der Rotation ist in der Abbildung 6.8 dargestellt. Die Ausrichtung des Weltkoordinatensystems entspricht der Richtung der $x, y \, und \, z$-Achse, das Laserkoordinatensystem wird durch die Achsen $x''', y''' \, und \, z'''$ wiedergegeben.

Die Winkel für diese Drehungen errechnen sich wie folgt:

$$\begin{aligned} Normalvektor \; der \; x, y - Ebene, \; N_{Welt} \; &= \; \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \;\; (z - Achse) \\ Normalvektor \; der \; x''', y''' - Ebene, \; N_{Laser} \; &= \; \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \;\; (z''' - Achse) \end{aligned} \tag{6.16}$$

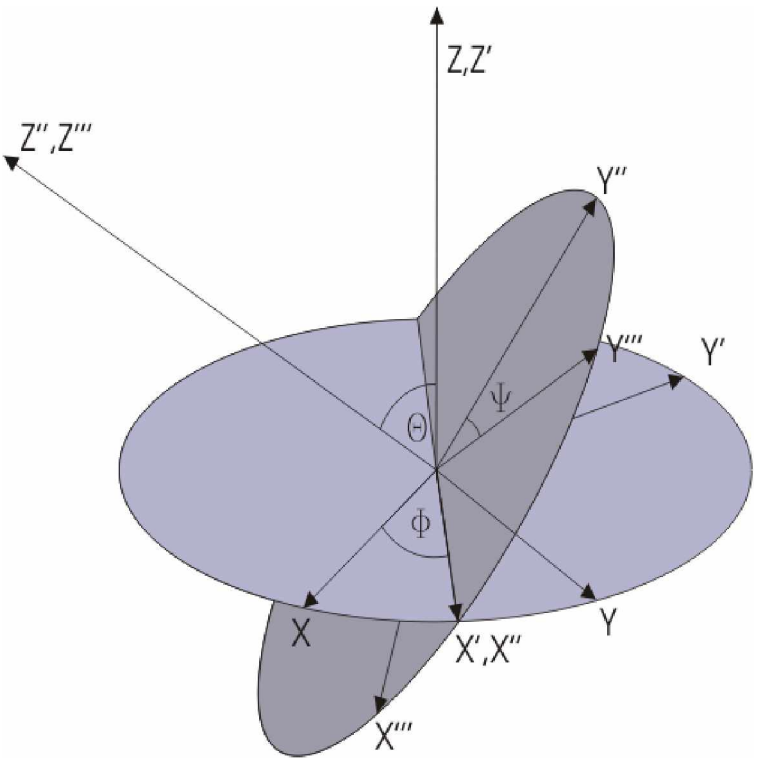


Abbildung 6.8: Prinzip der Rotation

Die erste Drehung erfolgt um die z-Achse und der Winkel ϕ ist jener, zwischen der x- und der x'-Achse. Die x'-Achse steht normal auf die Ebene, welche durch diese beiden Normalvektoren aufgespannt wird:

$$x' = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} E_2 \\ -E_1 \\ 0 \end{bmatrix} \tag{6.17}$$

Daraus ergibt sich für den Winkel ϕ:

$$\phi = \arccos \frac{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} E_2 \\ -E_1 \\ 0 \end{bmatrix}}{\left\| \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\| \left\| \begin{bmatrix} E_2 \\ -E_1 \\ 0 \end{bmatrix} \right\|} = \arccos \frac{E_2}{\sqrt{E_1^2 + E_2^2}} \tag{6.18}$$

Die neuen Achsenrichtungen ergeben sich nach dieser Drehung zu:

$$x' - Achse = \begin{bmatrix} E_2 \\ -E_1 \\ 0 \end{bmatrix} \quad , \quad y' - Achse = \begin{bmatrix} E_1 \\ E_2 \\ 0 \end{bmatrix} \quad , \quad z' - Achse = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.19}$$

Die zweite Drehung erfolgt um die x'-Achse. Dieser Winkel θ ist der Winkel zwischen der z'- und der z''-Achse und ergibt sich zu:

$$\theta = \arccos \frac{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}}{\left\| \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\| \left\| \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \right\|} = \arccos \frac{E_3}{\sqrt{E_1^2 + E_2^2 + E_3^2}} \tag{6.20}$$

Für die neue Ausrichtung der Achsen folgt:

$$x''-Achse = \begin{bmatrix} E_2 \\ -E_1 \\ 0 \end{bmatrix}, \quad y''-Achse = \begin{bmatrix} -E_1E_3 \\ -E_2E_3 \\ E_1^2 + E_2^2 \end{bmatrix}, \quad z''-Achse = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \tag{6.21}$$

Die Drehung nach x''', y''' *und* z''' ist nicht mehr notwendig, da sich die x'', y''-Ebene schon in der Laserebene befindet. Der Winkel ψ ist daher Null.

$$\psi = 0 \tag{6.22}$$

Bei diesen Drehungen ist immer die Drehrichtung zu berücksichtigen. So kann zB. eine Drehung um den Winkel ϕ in Abbildung 6.8 im oder gegen den Uhrzeigersinn erfolgen. Winkel zwischen 0 und 180 Grad beschreiben eine Drehung im Uhrzeigersinn und Winkel zwischen -180 und 0 Grad eine Drehung gegen den Uhrzeigersinn. Die Drehrichtung kann leicht aus den Zielkoordinatenachsen ermittelt werden.
In Abbildung 6.8 erfolgt die Drehung von x nach x' gegen den Uhrzeigersinn, somit muss der Winkel ϕ zwischen -180 und 0 Grad liegen. Würde die y-Komponente von x' negativ sein, müsste auch die Drehrichtung geändert werden.
Vor jeder Drehung wird im Programm die Drehrichtung kontrolliert und falls erforderlich korrigiert.

Die Berechnung der Transformationsmatrix zwischen dem Welt- und dem Laserkoordinatensystem, aus den translatorischen und rotatorischen Anteilen, wurde in Kapitel 3.2 gezeigt. Als Rückgabeparameter erhält man die Transformationsmatrizen "$\mathbf{Laser_{1T}}$" für die erste Laserebene, bzw. "$\mathbf{Laser_{2T}}$" für die zweite.

Im Programm "FitLaserPlane.m" werden die in diesem Kapitel beschriebenen Berechnungen ausgeführt.

6.1.7 Berechnung der Transformationsmatrix zwischen dem Pixel- und dem Laserkoordinatensystem

Damit die im Kalibrationsbild gefundenen Laserlinien in das Laserkoordinatensystem transformiert werden können, wird eine Projektionsvorschrift zwischen dem Pixel- und dem Laserkoordinatensystem berechnet.
Für diese Berechnung müssen wiederum mindestens 4 Laserpunkte mit ihren Koordinaten in beiden Koordinatensystemen bekannt sein (Kapitel 3.1.3).

Es werden je zwei Punkte auf der hinteren und der vorderen Kalibrationsebene für die Berechnung benötigt.
Der Abstand zwischen diesen zwei Punkten auf einer Ebene soll so groß als möglich sein. Dadurch wird der Fehler minimiert. Damit der Abstand zwischen den Punkten möglichst groß ist, werden die Laserlinien in Drittel eingeteilt. Pro Ebene wird so ein Punkt im ersten und im letzten Drittel ausgewählt.
Weiters werden jene Punkte verwendet, welche den geringsten Normalabstand zur Laserebene aufweisen. Diese Punkte können leicht gefunden werden, da aus der Anpassung der Laserebene (Kapitel 3.4.3) der Fehler für jeden Punkt bekannt ist.

$$P_{Welt} \rightarrow min(Fehler) \tag{6.23}$$

Für die Berechnung der Projektionsvorschrift werden die Koordinaten im Laser- und Pixelkoordinatensystem benötigt. Die Transformation in das Laserkoordinatensystem erfolgt mit der in Kapitel 6.1.6 errechneten Transformationsmatrix.

$$P_{Laser} = \mathbf{Laser_{xT}} \quad P_{Welt} \tag{6.24}$$

Die Position dieser Punkte im Pixelkoordinatensystem wurde durch die Berechnung der Intensitätsmomente in Kapitel 6.1.5 bestimmt.
Die Pixel- und Laserkoordinaten von diesen vier Punkten werden jeweils in eine Matrix eingetragen und der Routine "GenProj.m" übertragen. Der Rückgabewert ist die gesuchte Projektionsvorschrift zwischen dem Pixel- und dem Laserkoordinatensystem. Diese Berechnung wird wieder für beide Laserebenen durchgeführt und man erhält die Transformationsmatrizen "$\mathbf{Projektion_{KameraLaser1}}$" und "$\mathbf{Projektion_{KameraLaser2}}$".
Mithilfe dieser Transformationsmatrizen ist es möglich, jedem Punkt auf einer Laserlinie eine eindeutige Position im jeweiligen Laserkoordinatensystem zuzuweisen.

Die einzelnen Koordinatensysteme werden in Abbildung 6.9 veranschaulicht.

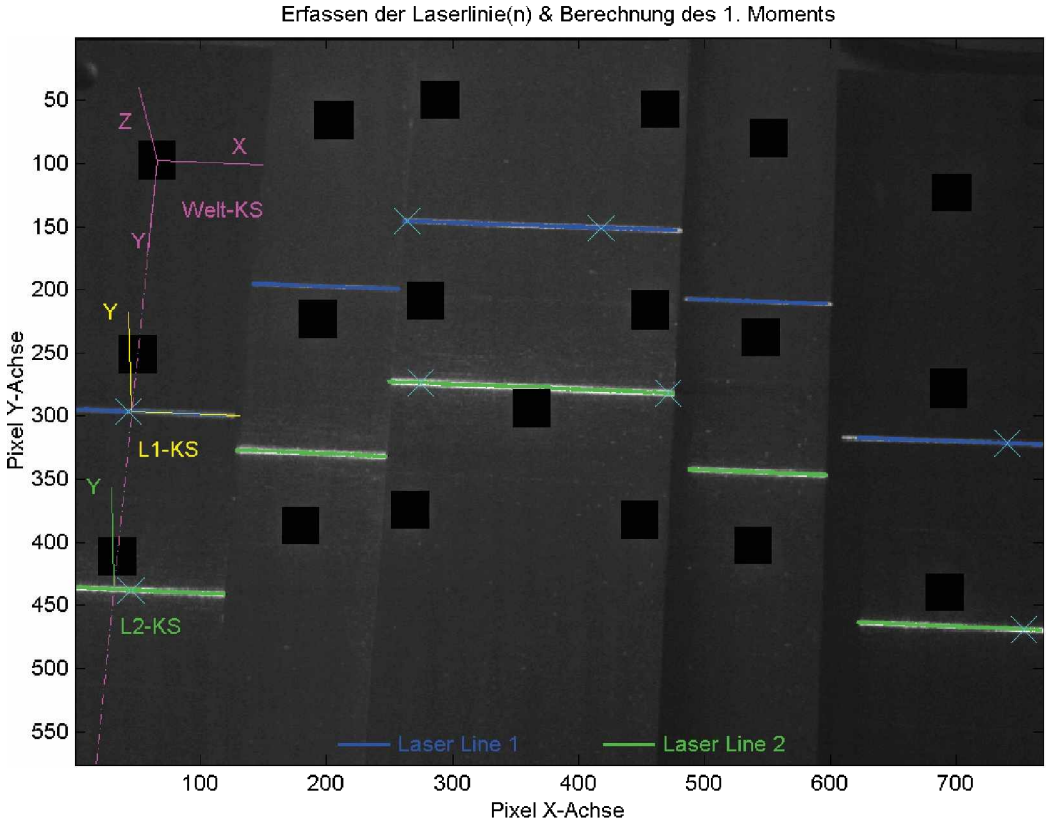


Abbildung 6.9: Prinzip der Kalibration

- Das Pixelkoordinatensystem, mit dem Ursprung in der linken, oberen Ecke.
- Das Weltkoordinatensystem, mit dem Ursprung im Mittelpunkt der linken, oberen Leuchtdiode.
- Die Laserkoordinatensysteme, mit den Ursprüngen in den Schnittpunkten der y-Achse des Weltkoordinatensystems mit den Laserebenen.
- Die Position der errechneten Intensitätsmomente der Laserlinien.
- Je 4 Punkte pro Laserebene, welche zur Berechnung der Projektionsvorschrift verwendet wurden.
- Die gelöschten Leuchtdioden.

6.2 Messung

Nach dem Start der Messung, durch Drücken des entsprechenden Icons im Menü, muss ein Messbild geladen werden. Wie schon bei der Kalibration wird das Bild in der Bildmatrix "**Pic0**" gespeichert.
Auch wird wieder ein Pixelkoordinatensystem mit dem Ursprung in der linken, oberen Ecke eingeführt.

6.2.1 Erfassen der Laserschnittlinien

Die Erfassung der Laserschnittlinien und die Berechnung des Intensitätsmoments erfolgt mit den in den Kapiteln 6.1.4 und 6.1.5 beschriebenen Methoden.
Aus den Rückgabeparametern dieser Routinen erhält man die Koordinaten der Lasersegmente im Pixelkoordinatensystem.

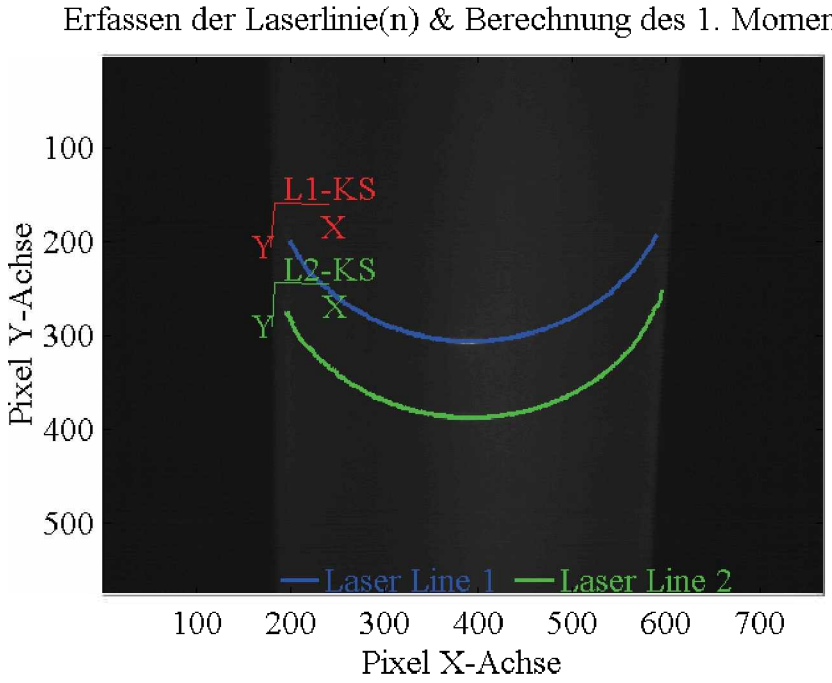


Abbildung 6.10: Erfassen der Laserschnittlinien

In Abbildung 6.10 sind die gefundenen Laserlinien blau bzw. grün dargestellt.

Mit den bei der Kalibration ermittelten Transformationsmatrizen ist es möglich, die Koordinaten der Laserlinien in das Weltkoordinatensystem zu transformieren. Dies geschieht für beide Laserebenen folgendermaßen:

Transformation der Punkte in das Laserkoordinatensystem.

$$P_{Laserx,nichtaffin} = \mathbf{Projektion_{KameraLaserx}}\ P_{Pixelx}$$

$$P_{Laserx,affin} = \begin{bmatrix} \frac{Px_{Laserx,nichtaffin}}{Pz_{Laserx,nichtaffin}} \\ \frac{Py_{Laserx,nichtaffin}}{Pz_{Laserx,nichtaffin}} \\ 0 \\ 1 \end{bmatrix} \tag{6.25}$$

Transformation der Punkte in das Weltkoordinatensystem.

$$P_{Weltx} = \mathbf{Laser_{xT}^{-1}}\ P_{Laserx,affin} \tag{6.26}$$

Wobei x die 1. oder die 2. Laserebene beschreibt.

Im Weltkoordinatensystem liegen nun alle Lasersegmente in dreidimensionalen Koordinaten vor. Ausgehend von diesen Segmenten muss unterschieden werden, welche Segmente auf der Oberfläche des zu vermessenden Objekts liegen und welche zum Hintergrund gehören.

6.2.2 Einteilung der Lasersegmente

Der gesuchte Querschnitt ist im dreidimensionalen Weltkoordinatensystem in der x-z-Ebene ersichtlich, wobei die x-Achse die Objektbreite und die z-Achse die Objekthöhe wiedergibt. Dies stimmt nicht ganz, da das Objekt nicht parallel zur y-Achse liegen muss, kann jedoch zur Unterscheidung der Segmente verwendet werden.
Das zu vermessende Walzgut hat einen runden, quadratischen oder rechteckigen Querschnitt (Anhang A). Aufgrund dieser einfachen geometrischen Formen können die Segmente eingeteilt werden.

Jedem Lasersegment wird mit der in Kapitel 3.4.4 beschriebenen Routine ein Kreis angepasst. Anhand des so errechneten Radius kann die Geometrie des Querschnitts unterschieden werden. Der minimale und maximale Radius des zu vermessenden runden Walzgutes sind bekannt (Anhang A).
Liegt der Radius unter der minimalen Grenze, so ist das Segment ein Ausreißer und wird verworfen.
Ist der Radius zwischen dem minimalen und dem maximalen Radius, so handelt es sich um einen kreisförmigen Querschnitt. Weisen zwei oder mehrere runde Segmente den gleichen Radius und die gleichen Mittelpunktkoordinaten auf, so gehören sie zum gleichen Objekt und werden zusammengefügt.
Ist der Radius größer als der maximale Radius des Walzgutes, so handelt es sich um eine Linie. In diesem Fall wird dem Lasersegment mit der in Kapitel 3.4.1 beschriebenen Routine eine Linie angepasst. Sind die "planner line" Koordinaten von mehreren Liniensegmenten ident, werden sie wieder zusammengefügt.

Die so in Linien- und Kreissegmente unterteilten Lasersegmente werden in den Variablen

"$Laserx_{Circle}$", "$Laserx_{Line1}$" und "$Laserx_{Line2}$" gesichert und an das Hauptprogramm übertragen. Diese Routine wurde unter dem Namen "MergeLaserLine.m" erstellt.

6.2.3 Berechnung des Durchmessers von Stäben

Der Großteil des, bei der Firma Edelstahl Witten-Krefeld GmbH zu vermessenden Walzgutes sind runde Stäbe. Aus diesem Grund wurde das Hauptaugenmerk dieser Arbeit auch auf die Vermessung dieser gelegt.

Ausrichtung der Messpunkte:

Wie schon in Kapitel 6.2.2 erwähnt ist die Laufrichtung des zu vermessenden zylindrischen Walzgutes meist nicht parallel zur y-Achse des Weltkoordinatensystems. Dadurch beschreibt die Projektion der gemessenen Punkte in die x-z-Ebene nicht exakt einen Kreis, sondern eine Ellipse. Würde man mit diesen Messpunkten einen Kreis berechnen, so ergibt sich immer ein systematischer Fehler. Die Größe dieses Fehlers ist abhängig von der Ausrichtung des Walzgutes zur y-Achse des Weltkoordinatensystems. Liegt das Walzgut parallel zur y-Achse so ist der Fehler Null und mit steigender Abweichung nimmt der Fehler zu.
Aus diesem Grund wird die Messung mit zwei Linienlaser durchgeführt, welche das Walzgut an zwei unterschiedlichen Stellen schneiden (Abbildung 6.11). Dadurch kann die Laufrichtung (Mittelpunktvektor) berechnet, und der Fehler korrigiert werden. Diese Korrektur erfolgt durch Drehung der Messpunkte, damit der Mittelpunktvektor parallel zur y-Achse des Weltkoordinatensystems wird.

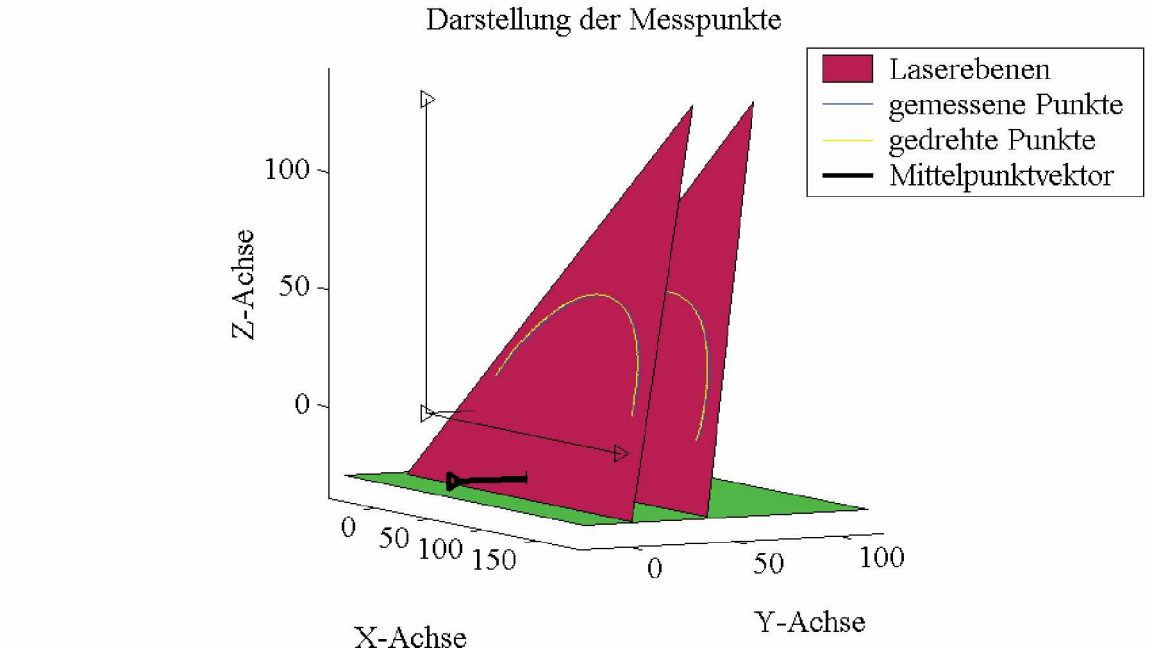


Abbildung 6.11: Darstellung der Messpunkte

Durch die, in Kapitel 6.2.2 durchgeführten, Kreisanpassungen sind die Mittelpunktkoordinaten und die Radien beider auf dem Walzgut liegenden Lasersegmente bekannt. Aus der Kalibration sind die Ebenengleichungen beider Laserebenen bekannt (Kapitel 6.1.6).
Die Kreismittelpunkte müssen auf den jeweiligen Laserebenen liegen und somit ergibt sich für die 1. Laserebene:

$$Vektor_{Laser1} = \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} \tag{6.27}$$

x_m und z_m sind die Mittelpunktkoordinaten. y_m erhält man aus der Ebenengleichung der Laserebene.

$$E_1\ x_m + E_2\ y_m + E_3\ z_m + E_4 = 0 \tag{6.28}$$

Durch Umwandlung dieser Gleichung erhält man für y_m:

$$y_m = \frac{-(E_1\ x_m + E_3\ z_m + E_4)}{E_2} \tag{6.29}$$

Diese Mittelpunkte werden für beide Laserebenen berechnet und in den Variablen "Vektor1" und "Vektor2" gespeichert. Durch Subtraktion beider Vektoren erhält man die gesuchte Laufrichtung des Walzgutes.

$$Normalvektor = Vektor_{Laser2} - Vektor_{Laser1} \tag{6.30}$$

Der Vektor wird in der Variable "Normalvektor" gespeichert, da dieser normal auf der Stirnfläche des Walzgutes steht. Dieser Normalvektor wird noch auf seine Richtung hin überprüft und so ausgerichtet, dass er immer in Richtung positiver y-Achse zeigt.

Damit die Winkel für die Drehung berechnet werden können, muss die exakte Position der Laufrichtung des Walzgutes bestimmt werden. Die Richtung des durch Subtraktion der Mittelpunkte ermittelten Vektors ist nicht ganz korrekt, da die Mittelpunkte aus Ellipsen und nicht aus Kreisen berechnet wurden. Um die exakte Richtung zu erfassen, erfolgt die weitere Berechnung iterativ.

Es wird eine Koordinatentransformation (Kapitel 3.2) so durchgeführt, dass der Normalvektor parallel zur y-Achse gedreht wird. Eine Translationsmatrix wird in diesem Fall nicht benötigt, da nur die Ausrichtung der Achsen und nicht der Koordinatenursprung verändert wird. Die Winkel der Rotationsmatrizen errechnen sich wie folgt:

Der Winkel ϕ, welcher die Drehung um die z-Achse beschreibt, ist der Winkel zwischen der y- und der y'-Achse (Abbildung 3.5).

Die Richtungen dieser Achsen sind:

$$y - Achse = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad y' - Achse = \begin{bmatrix} Normalvektor_x \\ Normalvektor_y \\ 0 \end{bmatrix} \tag{6.31}$$

Der Winkel zwischen diesen beiden Vektoren errechnet sich somit:

$$\phi = \arccos \frac{Normalvektor_y}{\sqrt{Normalvektor_x^2 + Normalvektor_y^2}} \tag{6.32}$$

Eine Drehung um die x'-Achse beschreibt der Winkel θ, welcher zwischen der y'- und y''-Achse liegt. Die Richtung der y''-Achse ist gleich der Richtung des Normalvektors. Somit errechnet sich der Winkel θ:

$$\theta = \arccos \frac{Normalvektor_x^2 + Normalvektor_y^2}{(\sqrt{Normalvektor_x^2 + Normalvektor_y^2}\sqrt{Normalvektor_x^2 + Normalvektor_y^2 + Normalvektor_z^2})} \tag{6.33}$$

Die Drehung der z''-Achse um den Winkel ψ wird nicht mehr benötigt, da die Rotation der y-Achse parallel zum Normalvektor bereits durch die beiden anderen Drehungen erfolgte.
Bei beiden Drehungen muss wieder die Drehrichtung kontrolliert werden und, falls nötig, werden die Vorzeichen der Winkel umgedreht. Die Berechnung der Rotationsmatrizen aus den drei Winkeln und die daraus folgende Transformationsmatrix "$T_{Rotation}$", erfolgt wie in Kapitel 3.2 gezeigt.

Berechnet werden diese korrigierten Lasersegmente wie folgt. Dies wird hier für die 1. Laserebene gezeigt und ist für die 2. Laserebene ident.

Zuerst müssen von allen Laserpunkten die Mittelpunktkoordinaten abgezogen werden. Dadurch wird der neue Ursprung der Koordinaten in diesen Mittelpunkt gelegt. Dies ist unbedingt erforderlich, da die Rotation um diesen Mittelpunkt erfolgt.

$$P_1 = Laser1_{Circle} - Vektor_{Laser1} = \begin{bmatrix} Laser1_{Circlex} - Vektor_{Laser1x} \\ Laser1_{Circley} - Vektor_{Laser1y} \\ Laser1_{Circlez} - Vektor_{Laser1z} \end{bmatrix} \tag{6.34}$$

Diese Punkte werden von der Richtung ihres Normalvektors "Normalvektor" in die Richtung der Welt y-Achse transformiert.

$$P_{1Trans} = \mathbf{T}_{\mathbf{Rotation}}^{-\mathbf{1}} \; P_1 \tag{6.35}$$

Aus diesen transformierten Punkten wird wieder der Kreismittelpunkt für beide Laserebenen berechnet. Der berechnete Mittelpunkt entspricht exakt dem "wahren" Kreiszentrum, da die Punkte einen Kreis beschreiben. Durch Subtraktion der Mittelpunkte wird der neue Mittelpunktvektor berechnet, der parallel zur y-Achse ist. Mit Hilfe der Rotationsmatrix wird dieser Vektor zurück in die Laufrichtung des Walzgutes gedreht und ist der neue Normalvektor des nächsten Iterationsschritts.
Die Iteration wird solange durchgeführt, bis die Abweichung zwischen dem neuen und dem alten Normalvektor minimal ist, oder die Anzahl der vorgegebenen Iterationsschritte erreicht ist.

Die iterative Ermittlung der Laufrichtung und die Berechnung der transformierten Kreispunkte wurde als Routine unter dem Namen "RotateCircle.m" erstellt.

Die vor der Transformation abgezogene y-Komponente des Mittelpunkts wird allen Punkten wieder zuaddiert. Durch die anschließende Projektion der Punkte in die x-z-Ebene ist dieser Schritt nicht erforderlich, die Darstellung in Abbildung 6.12 wäre aber sonst nicht möglich.

$$Laser1_{CircleKorr} = P_{1Trans} + Vektor_{Laser1} \tag{6.36}$$

Der Mittelpunktvektor dieser Punkte "$Laser1_{CircleKorr}$" und "$Laser2_{CircleKorr}$" ist nun parallel zur Welt y-Achse und somit können sie, durch Nullsetzten ihrer y-Koordinaten, in die x-z-Ebene projiziert werden. Abbildung 6.12 veranschaulicht diese Projektion.

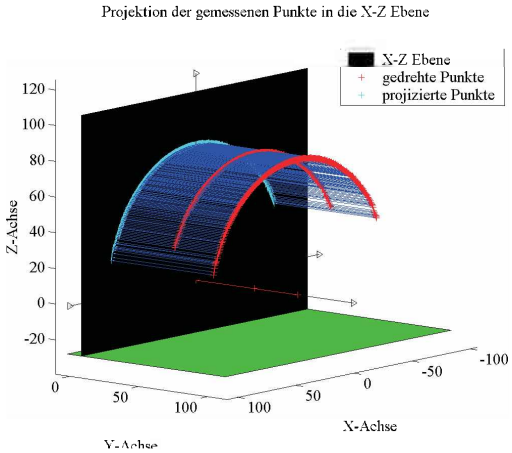


Abbildung 6.12: Projektion der Messpunkte in die x-z-Ebene

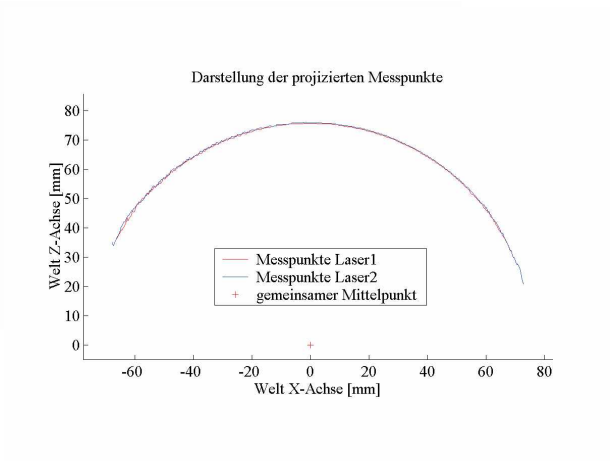


Abbildung 6.13: Darstellung der Messpunkte in der x-z-Ebene

Die beiden projizierten Lasersegmente sind in Abbildung 6.13 in der x-z-Ebene veranschaulicht. Nur in dieser Projektionsebene beschreiben diese Segmente eine Kreisfunktion.
Sie liegen übereinander, da der Querschnitt des Walzgutes über die gesamte Länge konstant bleibt.

Auswahl der Messpunkte welche für die Berechnung verwendet werden:

Für die folgende Berechnung des Durchmessers wird nur noch die x-z-Ebene betrachtet. Berechnet werden beide Lasersegmente unabhängig voneinander.

Die erste Berechnung des Durchmessers wird mit allen Punkten in einem Lasersegment durchgeführt. Dies geschieht mit der in Kapitel 3.4.5 beschriebenen Methode zum Anpassen eines Kreises an ein Kreissegment. Rückgabeparameter dieser Routine sind der Radius und die Mittelpunktkoordinaten.

Dieser Radius ist nur dann korrekt wenn alle Messpunkte tatsächlich auf der Oberfläche des Walzgutes liegen und diese nicht beschädigt ist. In diesem Fall folgen die Messpunkte einer idealen Kreisfunktion.

Fehlerquellen für eine Verfälschung des Messergebnisses:

- Zunder auf der Oberfläche des Walzgutes.
- Abflachung des Walzgutes.
- Reflexionen der Laserlinie auf der Oberfläche des Walzgutes.

Alle diese Fehlerquellen führen zu Abweichungen von der idealen Kreisfunktion. Bei den folgenden Berechnungen wird versucht, diese Abweichungen zu finden und herauszufiltern.

Abbildung 6.14: Beispiel einer stark verzunderten Oberfläche

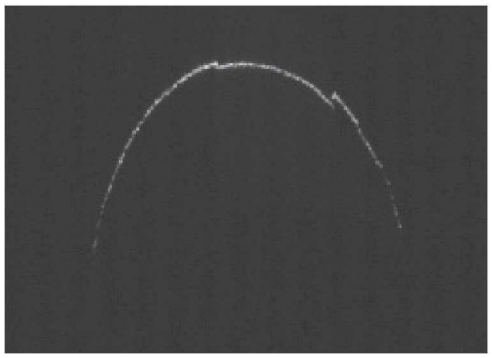


Abbildung 6.15: Dazugehöriges Messbild

In Abbildung 6.14 ist ein stark verzundertes Stück eines Walzgutes zu sehen. Im dazugehörigen Messbild in Abbildung 6.15 ist die dadurch verursachte Unstetigkeit eindeutig erkennbar.

Damit diese Abweichungen erfasst werden können, muss eine mathematische Funktion gefunden werden, welche den Verlauf der Datenpunkte am besten wiedergibt. Dies wird mit einer im Kapitel 3.4.6 beschriebenen Splinefunktion erreicht. Die Splinefunktion wird mit allen Punkten eines Segments berechnet.
Der Vorteil von Splines ist die Stetigkeit der Funktion über den gesamten Messbereich. Dadurch können die erste und die zweite Ableitung dieser Funktion berechnet werden. Ein weiterer Vorteil ist die Glättung von Ausreißern.
Der Rückgabeparameter der *Matlab*® Routine "spap2" ist die Splinefunktion "Sp". Aus dieser Splinefunktion werden nun neue Datenpunkte berechnet, welche exakt auf dieser Funktion liegen.

$$\begin{aligned} X &= [x_1, \ldots, x_n] = min(x_{Circle}) : 0.2 : max(x_{Circle}) \\ Y &= [y_1, \ldots, y_n] = fnval(Sp, X) \end{aligned} \tag{6.37}$$

Dies bedeutet, dass die x-Koordinaten der neuen Punkte auf der Splinefunktion im gesamten Messbereich gleichverteilt werden. Der Abstand zwischen den einzelnen Punkten beträgt 0.2mm. Mit dem Befehl "fnval" werden zu den jeweiligen x-Koordinaten die zugehörigen y-Koordinaten der Splinefunktion berechnet.

Von dieser Splinefunktion wird die erste und zweite Ableitung berechnet.

$$\begin{aligned} dSp &= \frac{dSp}{dx} = fnder(Sp) \\ ddSp &= \frac{d^2Sp}{dx^2} = fnder(dSp) \end{aligned} \tag{6.38}$$

Die angepasste Splinefunktionen und deren Ableitungen werden in Abbildung 6.16 für den fehlerhaften Datensatz aus Abbildung 6.15 veranschaulicht.

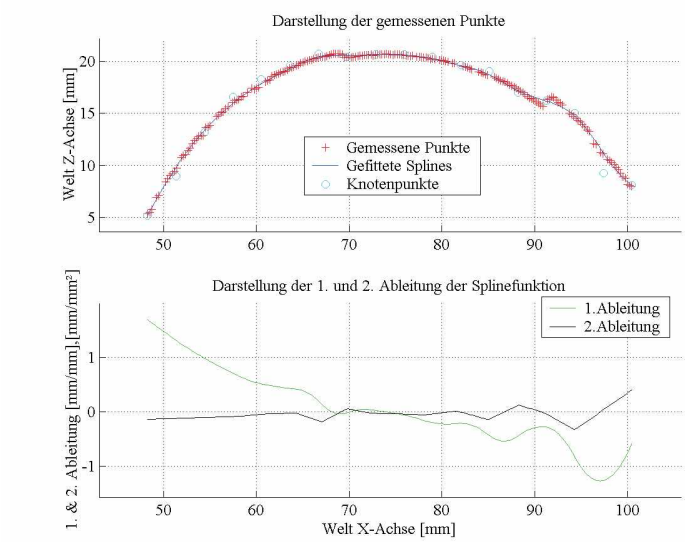


Abbildung 6.16: Fehlerbehaftete Messung

Für das erste Auswahlverfahren, um die verfälschten Messpunkte aussortieren zu können, wird die zweite Ableitung einer idealen Kreisfunktion berechnet.

Die ideale Kreisgleichung lautet:

$$(x_i - x_0)^2 + (y_i - y_0)^2 - r^2 = 0 \tag{6.39}$$

Wobei $x_0.y_0$ die Mittelpunktkoordinaten sind und r der Radius ist.

Werden die Mittelpunktkoordinaten von allen Datenpunkten x_i, y_i abgezogen, so verbleibt:

$$x_i^2 + y_i^2 - r^2 = 0 \; oder \; y = \sqrt{r^2 - x^2} \tag{6.40}$$

Von dieser Funktion können die erste und die zweite Ableitung berechnet werden:

$$\frac{dy}{dx} = \frac{-x}{\sqrt{r^2 - x^2}} \tag{6.41}$$

$$\frac{d^2y}{dx^2} = \frac{-r^2}{\sqrt{(r^2 - x^2)^3}} \tag{6.42}$$

Aus Gleichung 6.42 ist ersichtlich, dass die 2. Ableitung der idealen Kreisfunktion immer negativ sein muss. Ist die 2. Ableitung Null, so ist an dieser Stelle eine Abflachung und die Messpunkte sind offensichtlich falsch. (Die 2. Ableitung ist nur bei Konstanten und Funktionen 1. Ordnung Null. die Kreisfunktion ist jedoch eine Funktion 2. Ordnung!)
Es werden Grenzwerte für einen Toleranzbereich festgelegt, mit welchen die 2. Ableitung der Splinefunktion verglichen wird.

$$\frac{d^2y_{i,min}}{dx^2} = \frac{d^2y_i}{dx^2} - abs(\frac{max(\frac{d^2y_i}{dx^2}) - min(\frac{d^2y_i}{dx^2})}{2}) \tag{6.43}$$

$$\frac{d^2y_{i,max}}{dx^2} = \frac{d^2y_i}{dx^2} + abs(\frac{max(\frac{d^2y_i}{dx^2}) - min(\frac{d^2y_i}{dx^2})}{2}) \mid \{\frac{d^2y_{i,max}}{dx^2} \leq -0.005\} \tag{6.44}$$

Die maximale Toleranzgrenze wird mit -0.005 begrenzt, da während den Versuchen festgestellt wurde, dass es sich bei größeren Werten immer um eine Abflachung handelt. Sonst entsprechen die Toleranzgrenzen immer 50% der Differenz der maximalen und minimalen idealen zweiten Ableitung.
Alle Datenpunkte, deren zweite Ableitung außerhalb dieser Grenzen liegt, werden verworfen. Dies wird in Abbildung 6.17 für einen fehlerbehafteten Datensatz gezeigt.

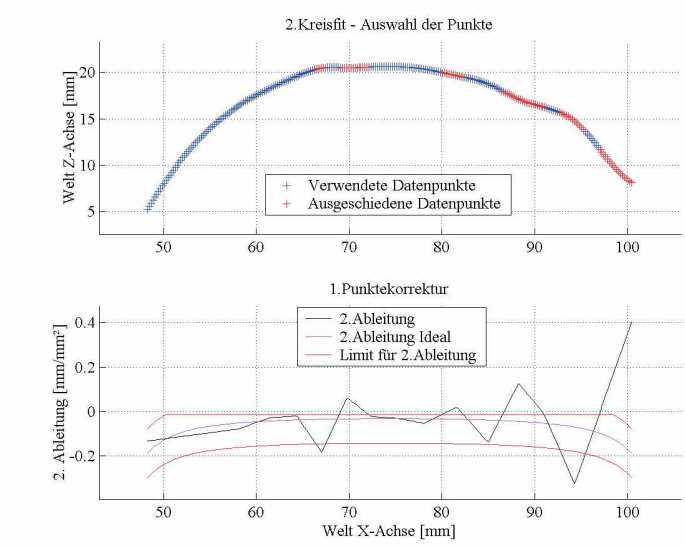


Abbildung 6.17: Fehlerbehaftete Messung

Die roten Messpunkte in Abbildung 6.17 liegen mit ihrer zweiten Ableitung nicht im Toleranzbereich und werden aussortiert. Mit den verbliebenen Datenpunkten wird wieder, mit der in Kapitel 3.4.5 beschriebenen Methode, ein Kreis aus einem Kreissegment berechnet.

In den Versuchen hat sich gezeigt, dass dieses Auswahlverfahren oft noch nicht ausreicht, um die gewünschte Messgenauigkeit zu erreichen. Darum werden die Messpunkte einem weiteren Auswahlverfahren unterzogen. Die Grundlage für dieses Verfahren ist der Krümmungsradius

eines idealen Kreises. Dieser hat den Vorteil, dass er bei einem idealen Kreis über den gesamten Umfang konstant ist.

Definition des Krümmungsradius für einen Kreis: [8]

$$\kappa_{Kreis} = \frac{1}{r} \quad , \quad r = Radius \tag{6.45}$$

Der Krümmungsradius wird mit dem Radius aus der vorherigen Anpassung des Kreises berechnet. Dieser Radius soll zwar optimiert werden, der Krümmungsradius ändert sich aber nur wenig, wenn der Radius leicht variiert.

Definition des Krümmungsradius für eine allgemeine Funktion $f(x)$: [8]

$$\kappa_i = \frac{|f''(x_i)|}{\sqrt{(1 + f'(x_i)^2)^3}} \tag{6.46}$$

Die erste und die zweite Ableitung der Splinefunktion sind bekannt und somit kann der Krümmungsradius für alle x_i berechnet werden.
Die Toleranzgrenzen für dieses Auswahlverfahren werden wieder mit 50% des idealen Krümmungsradius festgelegt:

$$|\kappa_i - \kappa_{Kreis}| < 0.5\ \kappa_{Kreis} \tag{6.47}$$

Das Ergebnis dieses Auswahlverfahrens ist in Abbildung 6.18 dargestellt.

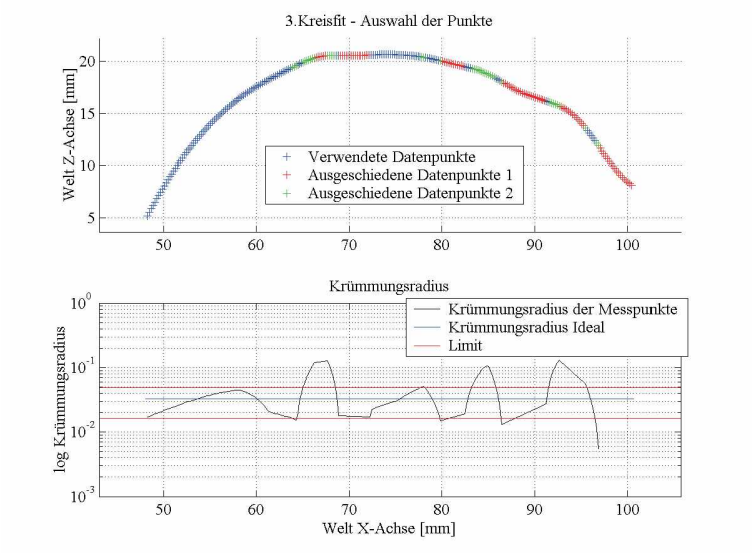


Abbildung 6.18: Fehlerbehaftete Messung

In Abbildung 6.18 sind die aussortierten Messpunkte blau eingezeichnet. Die roten Punkte sind im vorherigen Auswahlverfahren ausgeschieden und die grünen in diesem.

Nach diesem Auswahlverfahren wird das endgültige Ergebnis mit den verbleibenden Messpunkten berechnet. Dazu wird wieder die Routine "FitArc.m", welche in Kapitel 3.4.5 beschrieben wurde, verwendet.

Die Berechnung der Splinefunktion und die beiden Auswahlverfahren wurden in *Matlab*® im Unterprogramm "CalculateCircle.m" erstellt.

6.2.4 Darstellung der Messergebnisse für Stäbe

In der Abbildung 6.19 werden die errechneten Durchmesser für den Datensatz dargestellt.

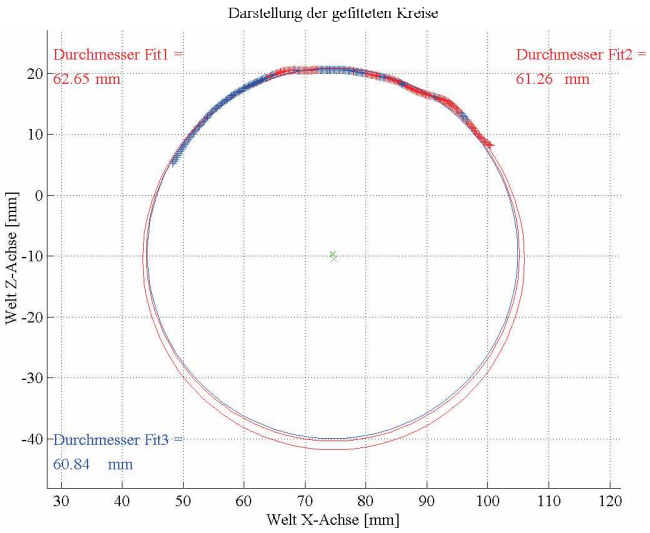


Abbildung 6.19: Fehlerbehaftete Messung

Das Endergebnis ist in der Abbildung 6.19 blau, als "DurchmesserFit3", dargestellt. Zur Berechnung des "DurchmesserFit1" wurden alle Messpunkte verwendet, für "Durchmesser-Fit2" nur jene, welche die Toleranzen des ersten Auswahlverfahrens erfüllten. Ausgeschiedene Punkte sind rot dargestellt.

Beurteilung des Ergebnisses in Abbildung 6.19

Das Walzgut wurde im glühenden Zustand vermessen. Die Probe wurde vor der Vermessung auf ca. 1000^oC aufgeheizt. Vor dem Aufheizvorgang wurde der Durchmesser mit $60.0mm$ bestimmt. Zum Vergleich muss dieses Maß erst auf das Warmmaß umgerechnet werden.
Die dafür notwendige Umrechnungsformel lautet:

$$D = D_0 (1 + \alpha \Delta T) \tag{6.48}$$

Wobei α der lineare Ausdehnungskoeffizient von Eisen ist und $\alpha = 12.3\ 10^{-6}\ \frac{1}{K}$ beträgt. ΔT ist die Temperaturdifferenz, D_0 das Kaltmaß und D das Warmmaß.

Daraus ergibt sich für das Warmmaß:

$$D = 60.0\ (1 + 12.3\ 10^{-6}\ 980)\ =\ 60.7mm \tag{6.49}$$

Wie man in der Abbildung erkennen kann, ist die erste Berechnung des Durchmessers mit allen Messpunkten nur eine Näherung. Erst durch die Selektion der Messpunkte in den Auswahlverfahren nähert sich der berechnete Durchmesser dem wahren Wert.

6.2.5 Berechnung der Höhe von Knüppeln

Mit dem in Kapitel 4 veranschaulichten Messaufbau ist es nicht möglich, den Querschnitt von rechteckigen Objekten zu ermitteln. Da die Messung mit nur einer Kamera durchgeführt wird, kann nur eine Seite vermessen werden. In diesem Fall, da die Kamera über dem Walzgut positioniert wird, ist eine Vermessung der Höhe möglich.
Im Gegensatz zur Vermessung von Stäben wird auch die Laserschnittlinie mit dem Hintergrund benötigt. Dieser Hintergrund muss eine waagrechte Ebene mit konstanter Höhe sein. Das Walzgut muss mit einer Seitenfläche eben auf diesem Hintergrund liegen.

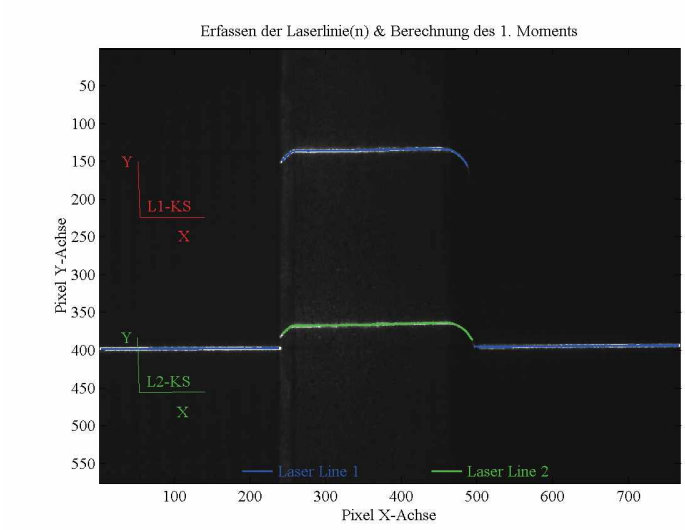


Abbildung 6.20: Erfassung eines rechteckigen Querschnitts

Nach der Erfassung (Kapitel 6.2.1) und der Einteilung (Kapitel 6.2.2) der Laserschnittlinien liegen deren Punkte im Weltkoordinatensystem in den Variablen "$Laserx_{Line1}$" und "$Laserx_{Line2}$" vor. Die Erfassung der Laserschnittlinien ist in Abbildung 6.20 dargestellt.

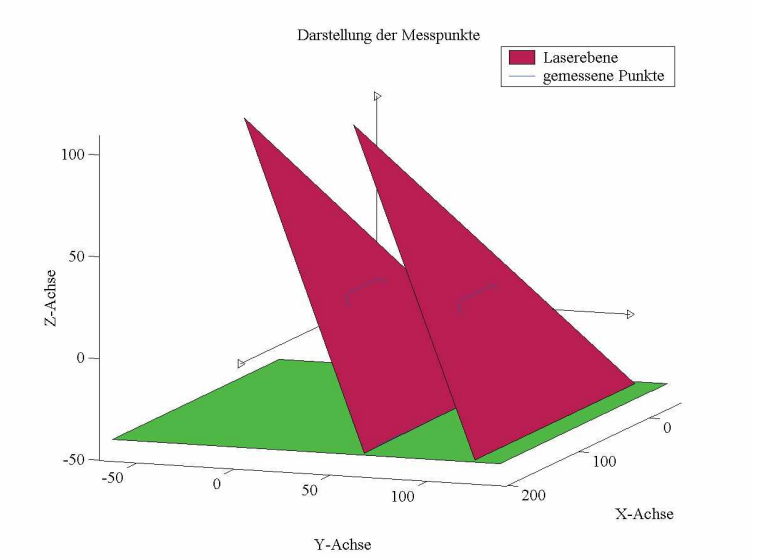


Abbildung 6.21: Darstellung der Messpunkte

Dargestellt werden die Messpunkte im Weltkoordinatensystem in Abbildung 6.21.

Aufgrund der Annahme, dass der Knüppel mit einer Seitenfläche eben auf dem Hintergrund aufliegt, müssen die Laserschnittlinien auf dem Walzgut und dem Hintergrund zueinander parallel sein. Es werden, mit der in Kapitel 3.4.2 beschriebenen Methode, zwei parallele Linien angepasst.
Wie in Abbildung 6.20 zu sehen ist, sind die Ecken des Knüppels abgerundet. Werden für die Berechnung einer Linie alle Datenpunkte der Schnittlinie auf dem Knüppel verwendet, so ergibt sich durch diese Radien ein Fehler. Die angepasste Linie wird nach unten verschoben und die errechnete Höhe wird kleiner als die tatsächliche. Zur Vermeidung dieses Fehlers werden von dieser Schnittlinie nur die mittleren 2/3 für die Berechnung verwendet.
Die Rückgabeparameter dieser Routine sind die "planner line" Koordinaten beider Geraden.

Aus den "planner line" Koordinaten kann leicht der Normalabstand der Geraden zueinander berechnet werden. Dieser Normalabstand entspricht auch der Höhe des Knüppels.
Wie schon in Kapitel 3.4.1 gezeigt, folgt für die Abstände der Geraden auf der y-Achse:

$$d_1 = \frac{N_1}{X} \quad , \quad d_2 = \frac{N_2}{X} \tag{6.50}$$

Die Höhe des Knüppels errechnet sich aus der Differenz dieser Abstände.

$$Höhe = |(d_1 - d_2)| \tag{6.51}$$

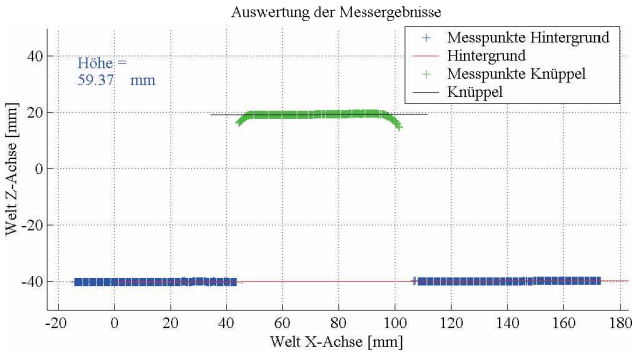


Abbildung 6.22: Errechnete Höhe des Knüppels

Der Knüppel wurde bei Raumtemperatur vermessen. Die Vermessung der Höhe mit einer Schiebelehre ergab 59.5mm.

Es wäre möglich, die Breite durch Abzählen der Pixel bzw. Länge der Schnittlinie auf dem Knüppel zu bestimmen. Dies wird aber durch die Radien erschwert und kann somit nur als Näherung verwendet werden. Eine wirklich genaue Messung ist nur mit einer zweiten Kamera möglich!

Kapitel 7

Messergebnisse

7.1 Labormessungen

Die Labormessungen wurden mit Lasern verschiedener Wellenlängen durchgeführt.

7.1.1 Messung mit einem grünen Laser

Die Messung wurde mit einem grünen Laser am Institut für Metallurgie durchgeführt. Das zu vermessende Rohrstück wurde vorher auf 1200^oC aufgeheizt. Bei Raumtemperatur betrug der Durchmesser des Rohrstücks $42.8mm$.

Technische Daten des Lasers:

Typ	: *LAP 20RYL*
Anbieter	: *Laser Applikation GmbH*
Wellenlänge	: $532nm$, grün
Laserleistung	: $20mW$
Laserklasse	: $3A$
Lebensdauer	: $< 10000h$
Durchmesser	: $55mm$
Länge	: $470mm$
Preis	: 2480 *Euro*

Abbildung 7.1: LAP 20RYL

Da am Institut für diesen Wellenlängenbereich kein Interferenzfilter vorhanden ist, wurde die Messung ohne Filter durchgeführt. Dies war möglich, da der grüne Wellenlängenbereich viel günstiger liegt als der rote (Kapitel 2.2.3).
Umrechnung des Rohrdurchmessers auf das Warmmaß:

$$D = D_0 (1 + \alpha \Delta T) \tag{7.1}$$

$$D = 42.8(1 + 12.3\ 10^{-6}\ 1180) = 43.4mm \tag{7.2}$$

Veranschaulichung der Messergebnisse:

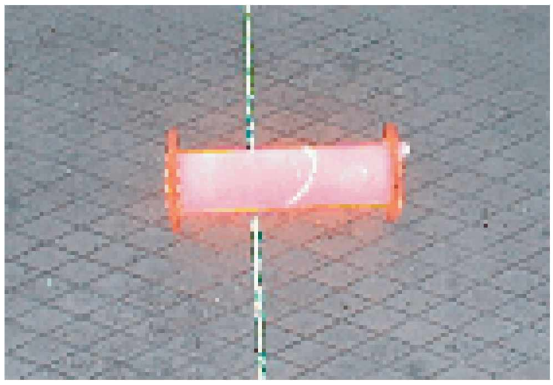


Abbildung 7.2: Messung mit einem grünen Laser

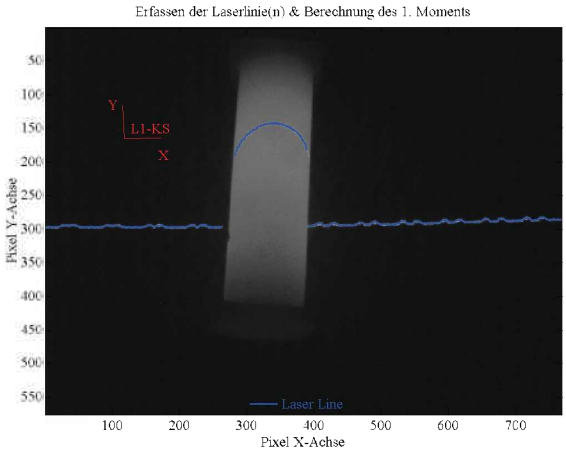


Abbildung 7.3: Dazugehöriges Messbild

Abbildung 7.2 zeigt das glühende Rohrstück, welches von der grünen Laserlinie geschnitten wird. Daneben in der Abbildung 7.3 ist das zugehörige Messbild zu sehen. Da kein Interferenzfilter verwendet wurde, ist das glühende Rohrstück auf dem Messbild noch gut zu erkennen. Dies stellt eine erhebliche Störquelle dar, da die Laserlinie unter Umständen nicht mehr gefunden werden kann. Die Blende vor dem Objektiv muss soweit geschlossen werden, damit die Laserlinie noch erkennbar bleibt und das glühende Rohrstück weitgehend verschwindet.
Nur im grünen Wellenlängenbereich ist es möglich, glühendes Walzgut ohne Interferenzfilter zu vermessen.

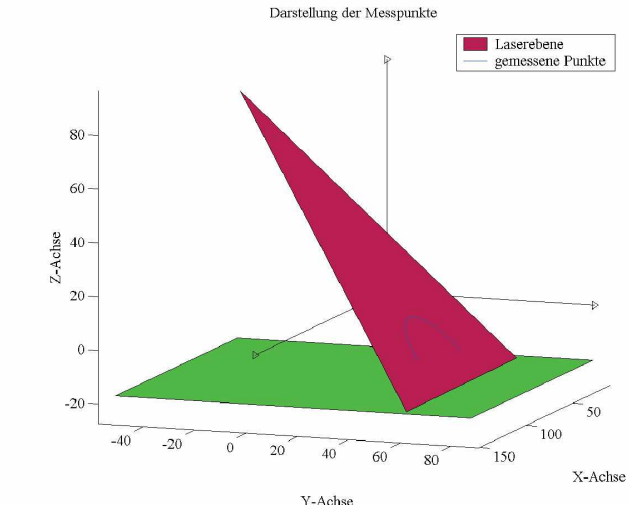


Abbildung 7.4: Darstellung der Messpunkte

In Abbildung 7.4 werden die Messpunkte im Weltkoordinatensystem dargestellt.

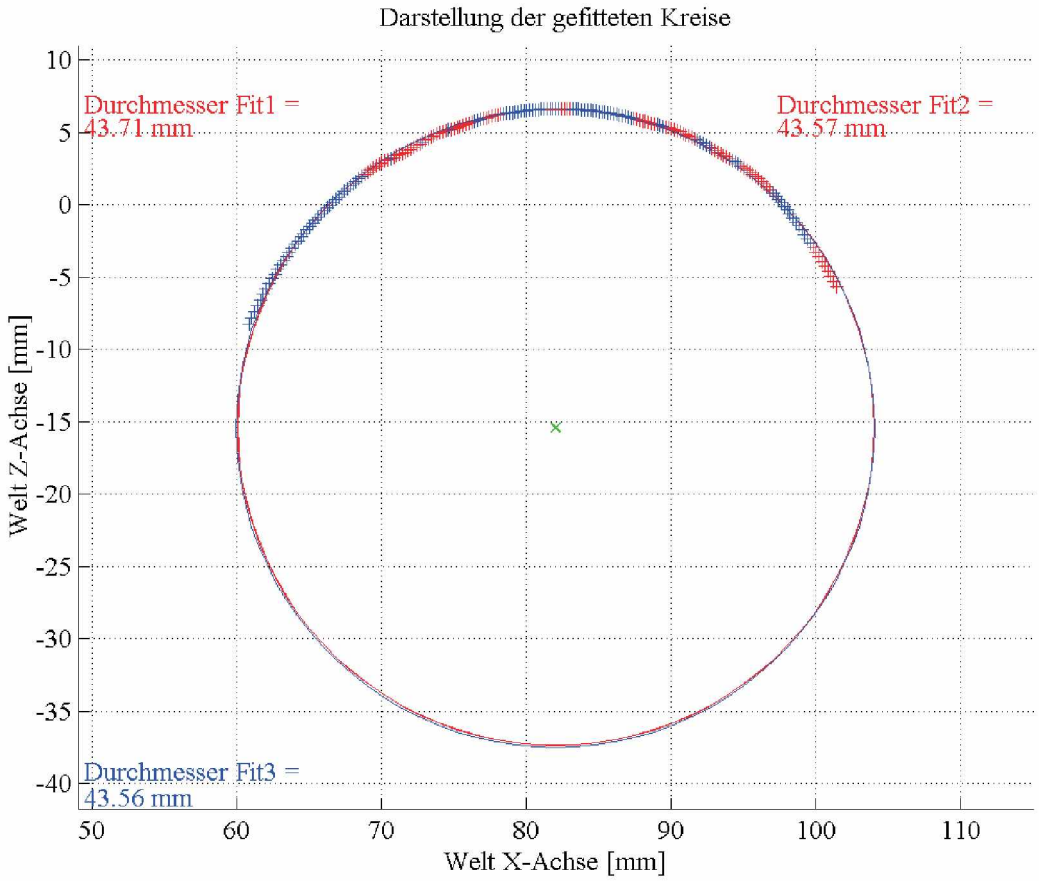


Abbildung 7.5: Errechneter Durchmesser

Das Ergebnis ist in Abbildung 7.5 ersichtlich. Der berechnete Durchmesser, nach Auswahl der Messpunkte, beträgt:

$$\textbf{Durchmesser} = \textbf{43.6mm} \tag{7.3}$$

7.1.2 Messung mit einem roten Laser

Diese Messung wurde am Institut für Wärmetechnik durchgeführt. Das zu vermessende Walzgut ist ein Stück von einem Stab der Firma Edelstahl Witten-Krefeld GmbH. Die Probe wurde vor der Messung auf ca. 1000^oC aufgeheizt. Bei Raumtemperatur betrug der Durchmesser $60.0mm$.

Technische Daten des Lasers:

Typ	: $LAS - 670 - 100$
Anbieter	: *Laser 2000 GmbH*
Wellenlänge	: $670nm$, rot
Laserleistung	: $100mW$
Laserklasse	: $3B$
Lebensdauer	: $50000 - 100000h$
Durchmesser	: $19mm$
Länge	: $90mm$
Preis	: 2300 *Euro*

Abbildung 7.6: LAS-670-100

Für diese Messung wurde ein Interferenzfilter verwendet. Dieser hatte seine maximale Durchlässigkeit bei $\lambda = 670nm$. Zur Kalibration musste der Interferenzfilter abgenommen werden, da die Leuchtdioden ihr Maximum bei $\lambda = 625nm$ besitzen und daher bei Verwendung dieses Filters unsichtbar sind.

Umrechnung des Stabdurchmessers auf das Warmmaß:

$$D = D_0 \, (1 + \alpha \, \Delta T) \tag{7.4}$$

$$D = 60.0(1 + 12.3 \; 10^{-6} \; 980) = 60.7mm \tag{7.5}$$

Der Stab war durch mehrmaliges Aufheizen leider sehr stark verzundert, wodurch der Durchmesser an manchen Stellen verändert wurde.

Veranschaulichung der Messergebnisse:

Abbildung 7.7: Messung mit einem roten Laser

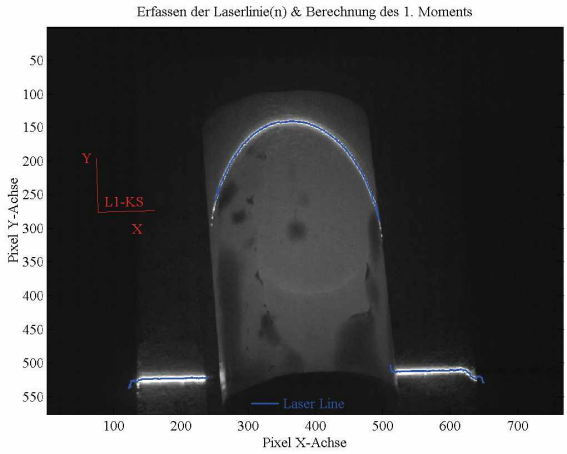


Abbildung 7.8: Dazugehöriges Messbild

In Abbildung 7.8 ist der glühende Stab eindeutig zu erkennen. Trotz Interferenzfilter vor dem Objektiv ist dieser störende Einfluss noch vorhanden, kann jedoch mit Hilfe der Belichtungszeit und der Blendeneinstellung minimiert werden.

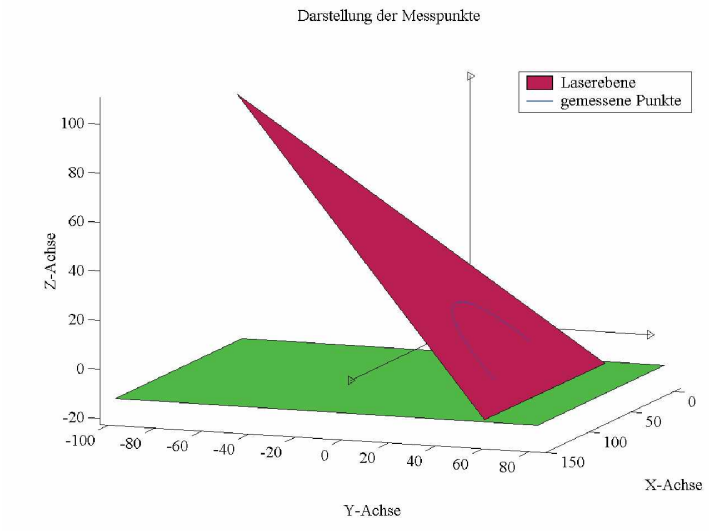


Abbildung 7.9: Darstellung der Messpunkte

In Abbildung 7.9 werden die Messpunkte im Weltkoordinatensystem dargestellt.

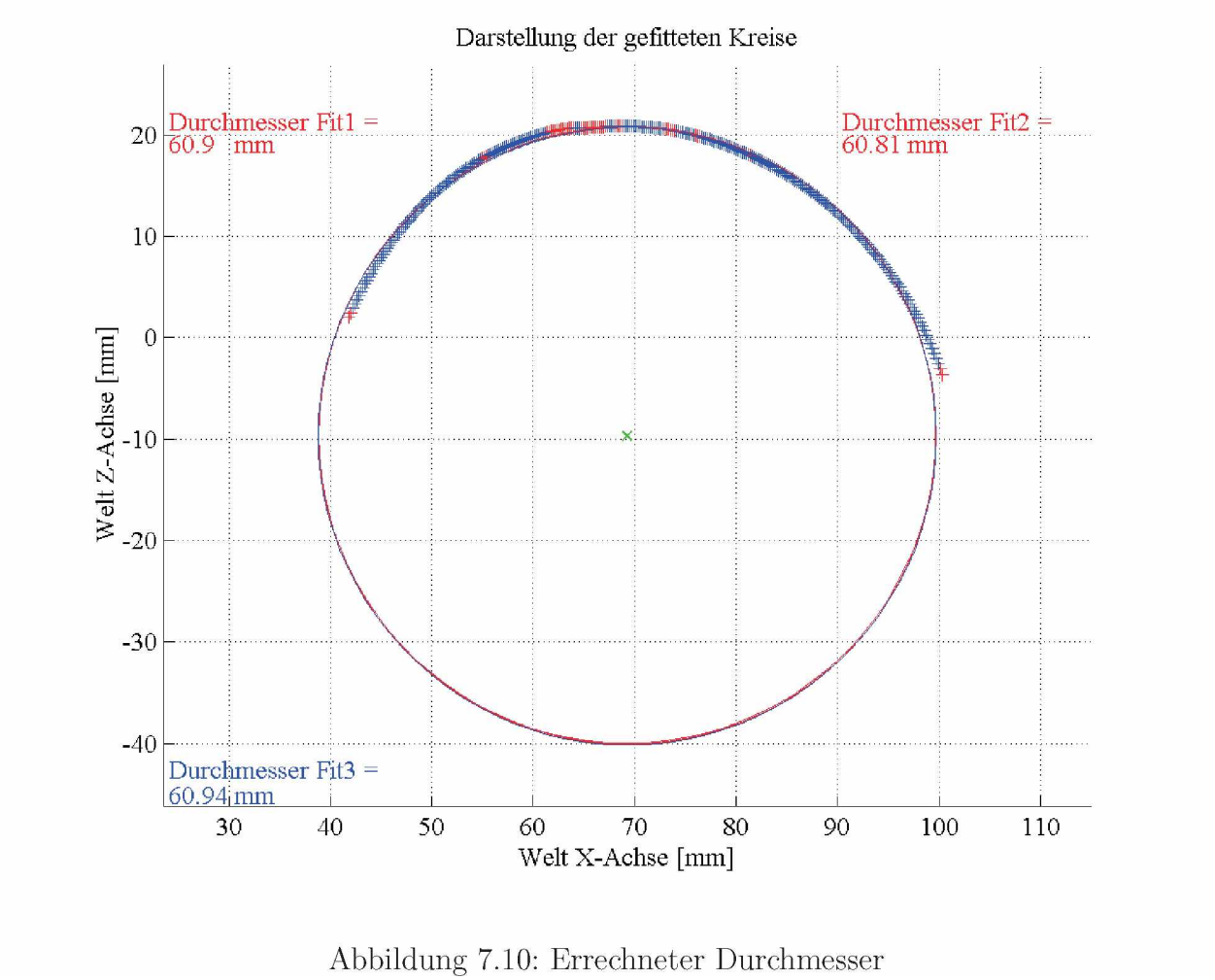


Abbildung 7.10: Errechneter Durchmesser

Das Ergebnis ist in Abbildung 7.10 ersichtlich. Der berechnete Durchmesser, nach Auswahl der Messpunkte, beträgt:

$$\textbf{Durchmesser} = \textbf{60.9mm} \tag{7.6}$$

Mögliche Fehlerquellen, die das Ergebnis verfälschen können:

1. Durch die stark verzunderte Oberfläche des Stabes beträgt das Warmmaß des Durchmessers nicht mehr $60.7mm$.

2. Wie in Abbildung 7.10 ersichtlich, wirkt sich bei dieser Messung der Nachteil bei Verwendung von nur einem Laser aus. Die Zylinderachse des Stabes ist nicht parallel zur Welt y-Achse und dadurch beschreibt die Laserschnittlinie keinen Kreis, sondern eine Ellipse in der x-z-Ebene. Aus diesem Grund liegen die Enden der Lasersegmente nicht auf dem berechneten Kreis.
   Mit einem Laser kann die Ausrichtung der Zylinderachse nicht bestimmt und der Fehler nicht korrigiert werden.

7.1.3 Messung mit zwei roten Lasern

Diese Messung wurde am Institut für Metallurgie durchgeführt. Das zu vermessende Rohrstück wurde vor der Messung wieder auf $1200°C$ aufgeheizt. Der bei Raumtemperatur mit einer Schiebelehre ermittelte Durchmesser betrug $42.6mm$.

Technische Daten der Laser:

Typ	: $55CM - 685 - 43$
Anbieter	: $Schäfter \& Kirchhoff$
Wellenlänge	: $685nm$, rot
Laserleistung	: $43mW$
Laserklasse	: $3B$
Lebensdauer	: $> 50000h$
Durchmesser	: $25mm$
Länge	: $95mm$
Preis	: 995 $Euro$

Abbildung 7.11: 55CM-685-43

Zur Minimierung der störenden Temperaturstrahlung wurde wieder ein Interferenzfilter vor dem Objektiv verwendet. Dieser Filter hat seine maximale Durchlässigkeit bei $\lambda = 685nm$. Seine Kennlinie ist in Abbildung 2.7 dargestellt. Die Kalibration wurde ohne Filter durchgeführt.

Umrechnung des Rohrdurchmessers auf das Warmmaß:

$$D = D_0 \, (1 + \alpha \, \Delta T) \tag{7.7}$$

$$D = 42.6(1 + 12.3 \; 10^{-6} \; 1180) = 43.2mm \tag{7.8}$$

Veranschaulichung der Messergebnisse:

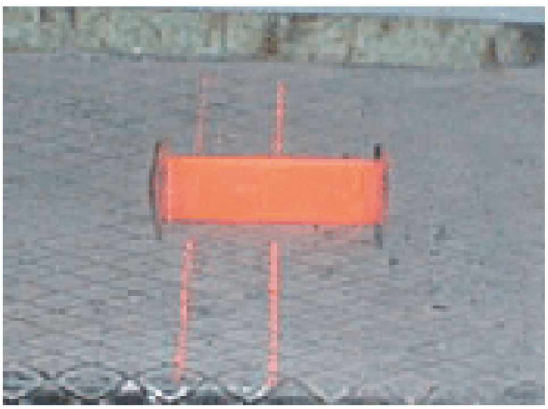


Abbildung 7.12: Messung mit zwei roten Lasern

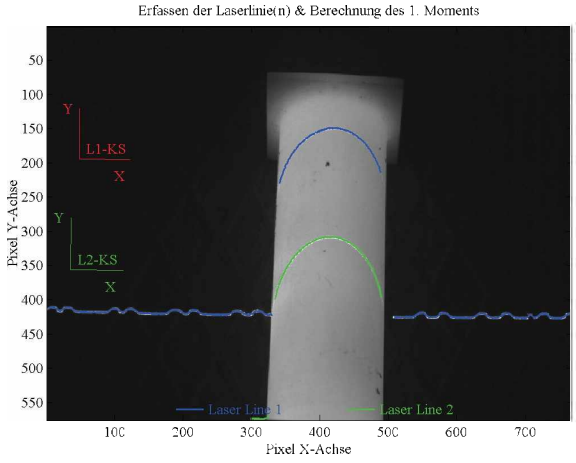


Abbildung 7.13: Dazugehöriges Messbild

Durch Einstellung der Belichtungszeit der Kamera und der Blende des Objektivs wurden die Störeinflüsse großteils eliminiert (Abbildung 7.13).

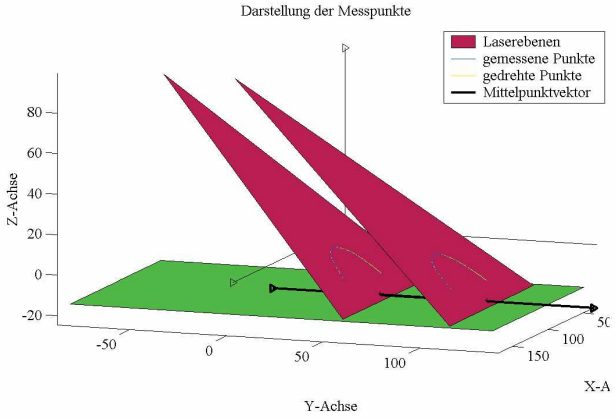


Abbildung 7.14: Darstellung und Korrektur der Messpunkte

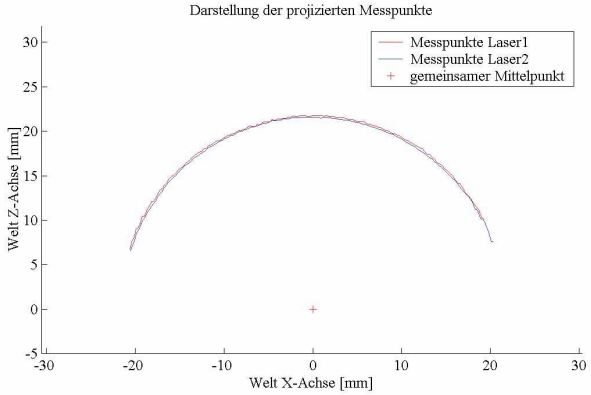


Abbildung 7.15: Projizierte Messpunkte

Mithilfe beider Lasersegmente kann die Richtung der Zylinderachse errechnet und parallel zur Welt y-Achse gedreht werden (Abbildung 7.14). In diesem Fall beschreiben die, in die x-z-Ebene projizierten, Laserschnittlinien eine Kreisfunktion (Abbildung 7.15).

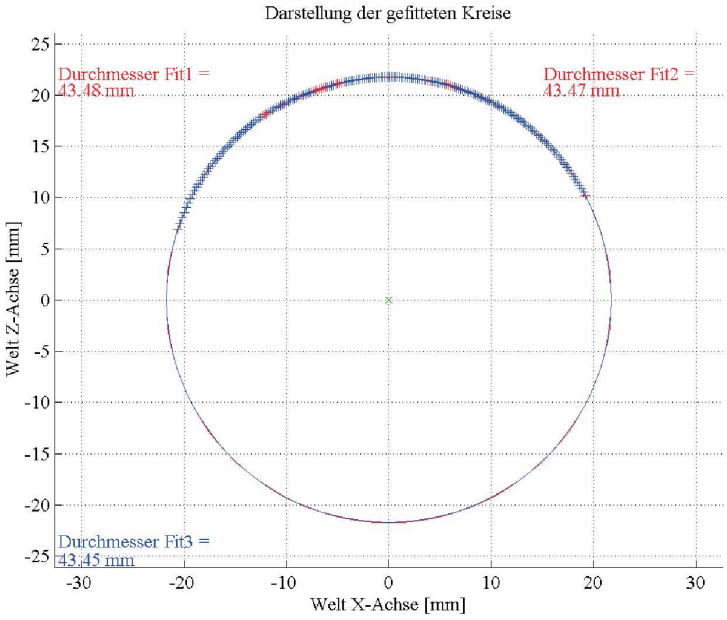


Abbildung 7.16: Errechneter Durchmesser für Laser 1

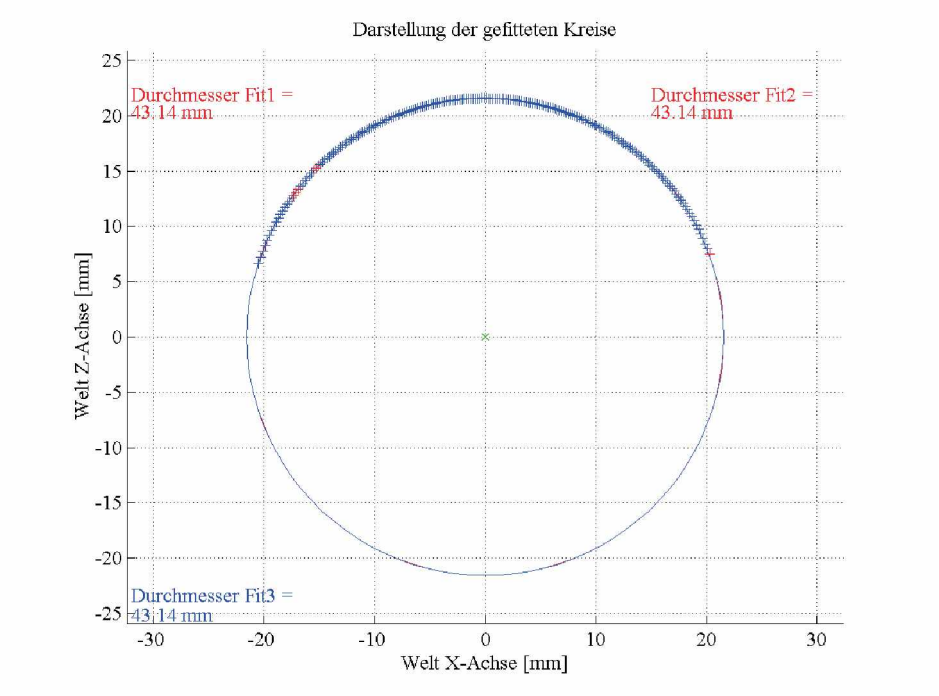


Abbildung 7.17: Errechneter Durchmesser für Laser 2

Die Ergebnisse sind in den Abbildungen 7.16 bzw. 7.17 für beide Laserlinien veranschaulicht und betragen:

$$\begin{aligned} \mathbf{Durchmesser_{Laser1}} &= \mathbf{43.5mm} \\ \mathbf{Durchmesser_{Laser2}} &= \mathbf{43.1mm} \end{aligned} \tag{7.9}$$

Die Abweichung von ca. 0.4mm der Ergebnisse ist auf zufällige Messfehler zurückzuführen. Der Mittelwert der Ergebnisse entspricht dem wahren Wert.

7.2 Messungen über der Walzstraße

Die Messungen wurden über der Walzstraße bei der Firma Edelstahl Witten-Krefeld durchgeführt.

Stäbe wurden vom kleinsten bis zum größten Durchmesser vermessen. Die Temperatur der Stäbe betrug ca. 1000^oC, wurde aber nicht bei jedem Stab gemessen. Bei größeren Durchmessern war die Temperatur etwas höher als bei kleineren. Da die genaue Zusammensetzung der Legierung jedes Stabes und somit sein Ausdehnungskoeffizient unbekannt waren, wird für die Umrechnung des Kaltmaßes auf das Warmmaß der Ausdehnungskoeffizient von reinem Eisen verwendet.

Repräsentativ für die verschiedenen Durchmesser werden folgend einige Messergebnisse veranschaulicht:

7.2.1 Vermessung von kleinen Durchmessern

Die Geschwindigkeit von dünnen Stäben ist höher als die von dickeren und beträgt bis zu $4\ m/s$.

Durchmesser 60mm:

Umrechnung vom Kaltmaß auf das Warmmaß:

$$D = 60.0(1 + 12.3\ 10^{-6}\ 1000) = 60.7mm \tag{7.10}$$

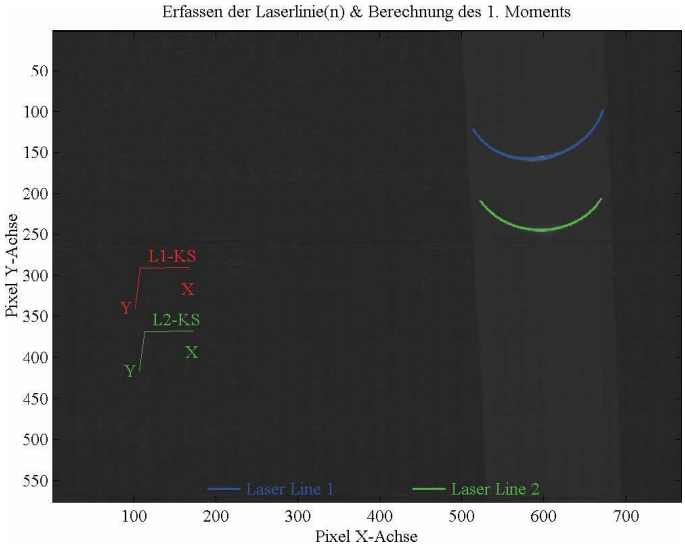


Abbildung 7.18: Erfassen der Laserlinien

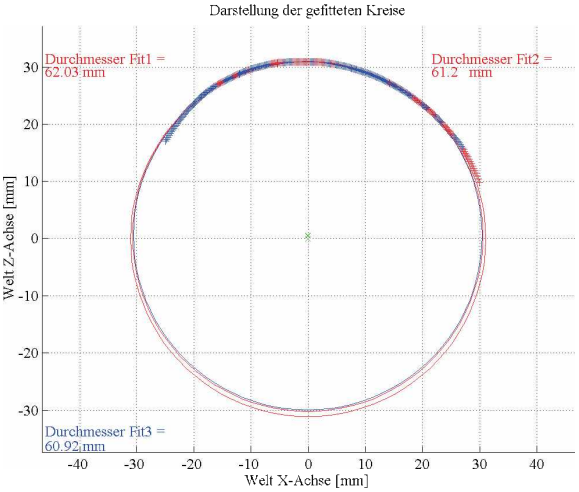


Abbildung 7.19: Ergebnis für Laser 1

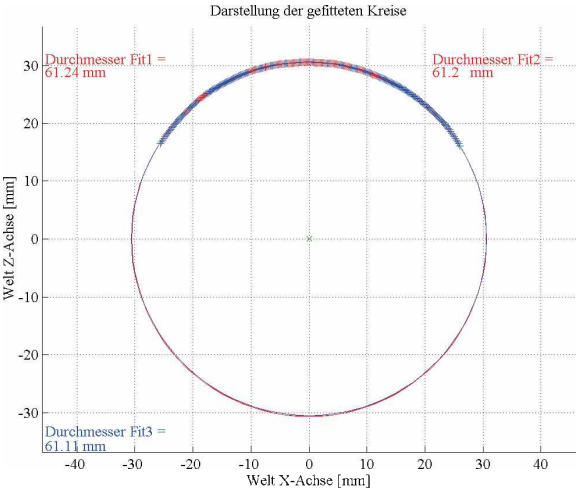


Abbildung 7.20: Ergebnis für Laser 2

Die gemessenen Durchmesser betragen $60.9mm$ und $61.1mm$.

Durchmesser 82mm:

Umrechnung vom Kaltmaß auf das Warmmaß:

$$D = 82.0(1 + 12.3\ 10^{-6}\ 1000) = 83.0mm \tag{7.11}$$

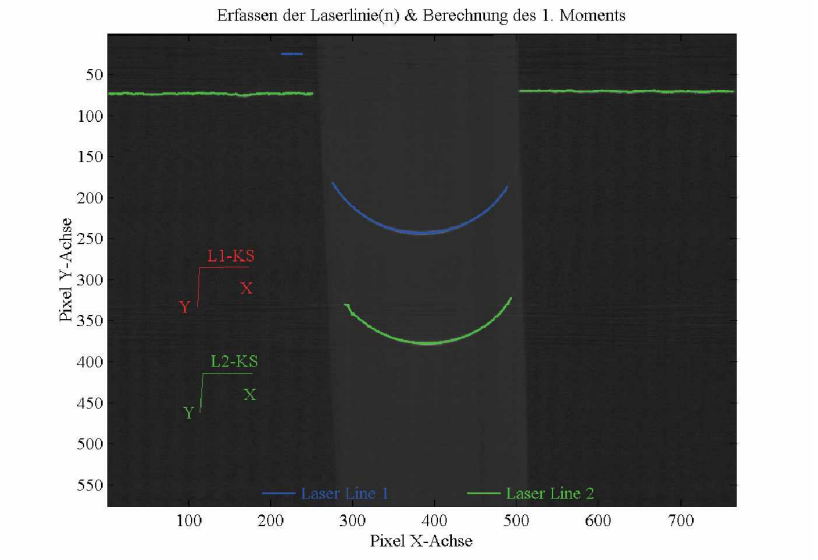


Abbildung 7.21: Erfassen der Laserlinien

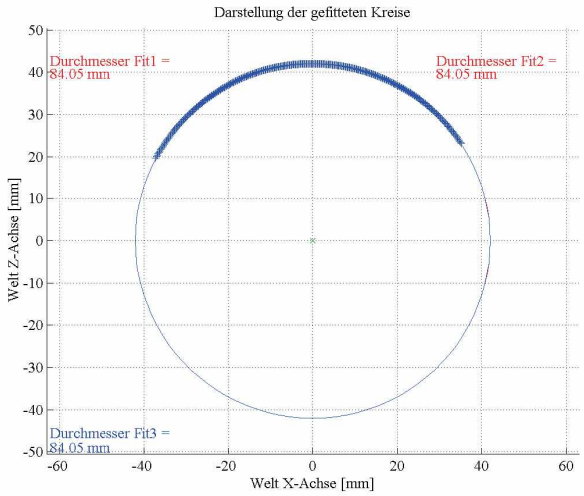


Abbildung 7.22: Ergebnis für Laser 1

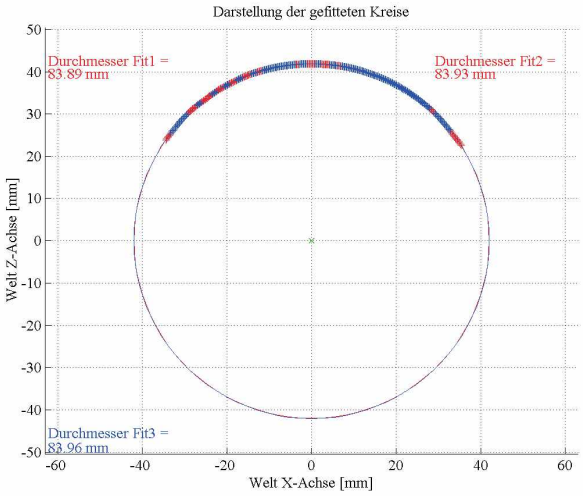


Abbildung 7.23: Ergebnis für Laser 2

Die gemessenen Durchmesser betragen $84.1mm$ und $84.0mm$. Der Fehler zum errechneten Warmmaß beträgt ca. $1mm$. Dabei handelt es sich um einen systematischen Fehler. Bei mehreren durchgeführten Messungen variierte das Ergebnis nur leicht um $84.0mm$.

Mögliche Gründe für den systematischen Fehler:

- Aufgrund der Legierungszusammensetzung ist der Ausdehnungskoeffizient des Stabes höher.
- Nach der Kalibration wurde die Position von Kamera oder Laser verändert.
- Das Kaltmaß war durch die momentane Walzeneinstellung größer als $82.0mm$.

Eine höhere Temperatur als angenommen ist als Fehlerquelle auszuschließen, da die Änderung für einen Fehler in dieser Größenordnung größer sein müsste.

7.2.2 Vermessung von mittleren Durchmessern

Durchmesser 150mm:

Umrechnung vom Kaltmaß auf das Warmmaß:

$$D = 150.0(1 + 12.3\ 10^{-6}\ 1000) = 151.8mm \tag{7.12}$$

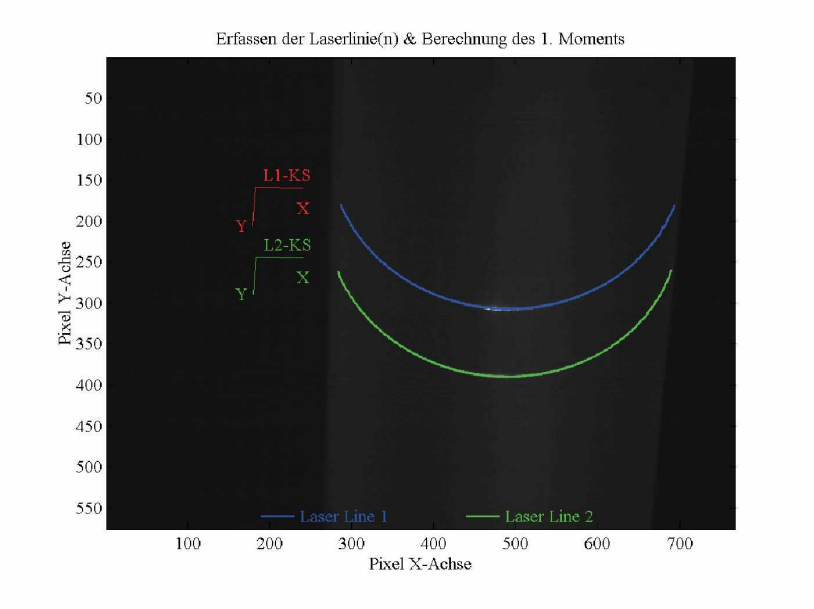


Abbildung 7.24: Erfassen der Laserlinien

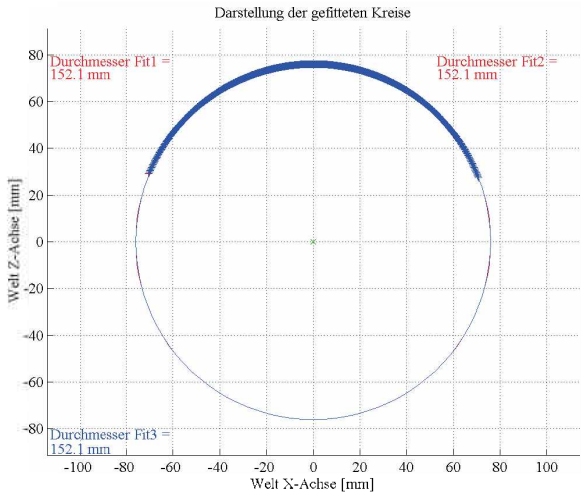


Abbildung 7.25: Ergebnis für Laser 1

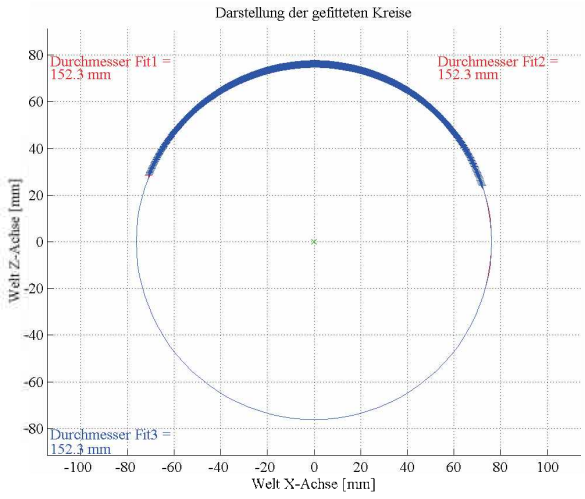


Abbildung 7.26: Ergebnis für Laser 2

Die gemessenen Durchmesser betragen $152.1mm$ und $152.3mm$.

7.2.3 Vermessung von großen Durchmessern

Durchmesser 250mm:

Umrechnung vom Kaltmaß auf das Warmmaß:

$$D = 250.0(1 + 12.3\ 10^{-6}\ 1000) = 153.1mm \tag{7.13}$$

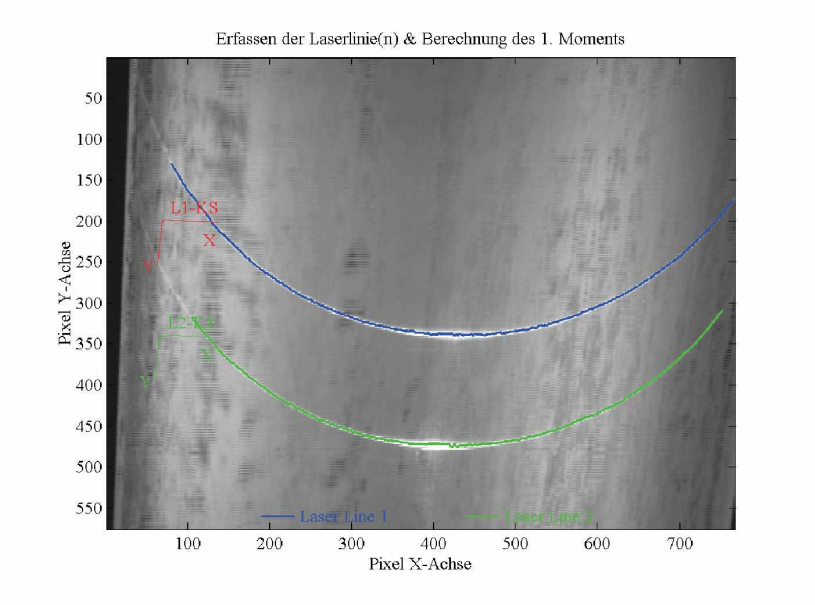


Abbildung 7.27: Erfassen der Laserlinien

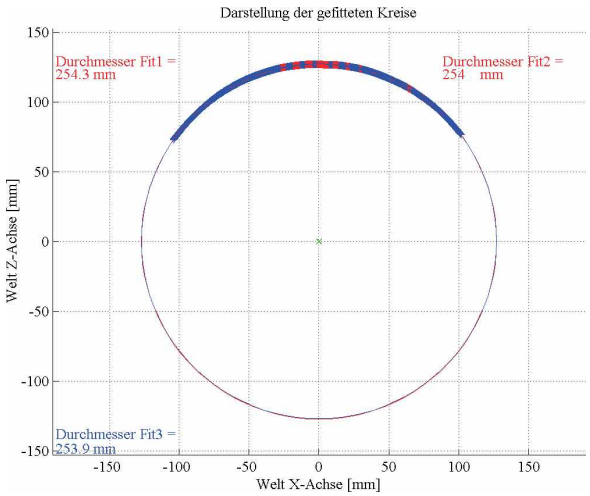


Abbildung 7.28: Ergebnis für Laser 1

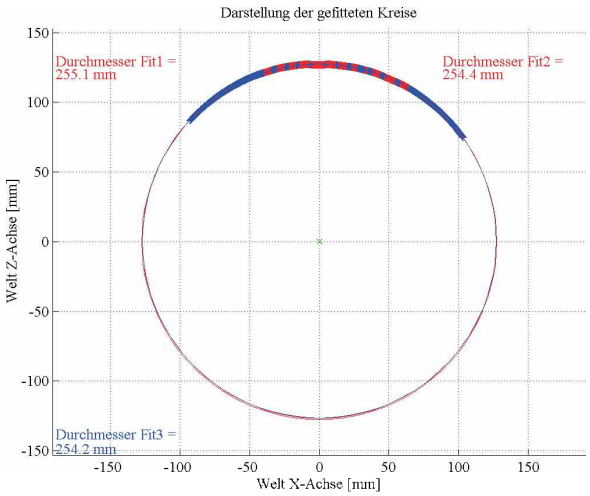


Abbildung 7.29: Ergebnis für Laser 2

Die gemessenen Durchmesser betragen $253.9mm$ und $254.2mm$.

7.2.4 Fehlerbehaftete Messungen

Verwackelte Messbilder:

Das Walzgut bewegt sich mit einer Geschwindigkeit von bis zu 6 m/s über die Rollen auf der Walzstraße und hat ein Gewicht von ca. 6.5 t. Die dadurch entstehenden Vibrationen, spürbar in der gesamten Walzhalle, können das Messergebnis erheblich verfälschen.

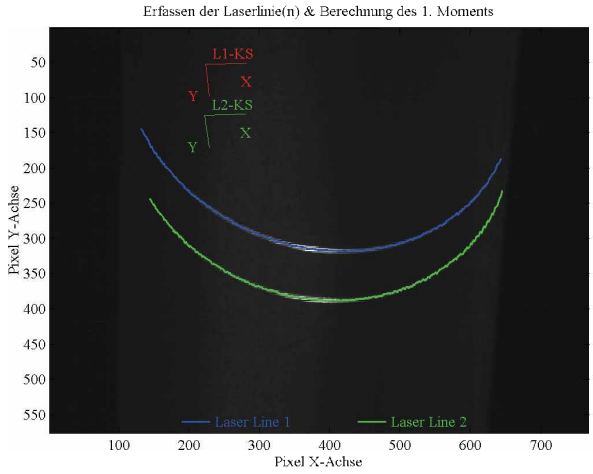


Abbildung 7.30: Verwackeltes Messbild

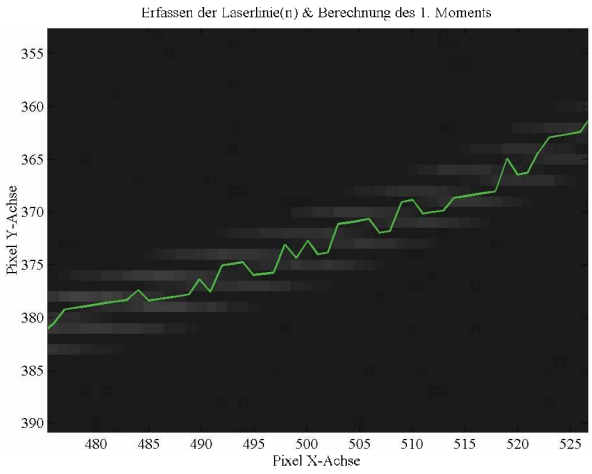


Abbildung 7.31: Ausschnitt aus Abbildung 7.30

Die Aufnahme eines Bildes mit der CCD-Kamera erfolgt in zwei Schritten. Es werden zwei Halbbilder zeitlich verzögert aufgenommen. Zuerst ein Bild mit allen ungeraden Zeilen und 20 Millisekunden später, eines mit den geraden Zeilen. Im Anschluss werden beide Halbbilder zusammengefügt und ergeben das gesamte Bild. Diese Aufnahmetechnik ist als internationale Standardnorm CCIR bekannt. (Comité Consultatif International de Radiocommunication) [12]
Mit diesem Verfahren ist es möglich, bis zu 25 ganze Bilder pro Sekunde aufzunehmen.

Die Zeitverzögerung zwischen den Halbbildern kann bei starken Vibrationen ausreichen, dass sich die Ausrichtung des Walzgutes quer zu seiner Laufrichtung verändert und dadurch verwackelte Gesamtbilder entstehen. Ein solches Bild ist in Abbildung 7.30 dargestellt. Deutlich ist dies am unregelmäßigen Verlauf der Laserlinien ersichtlich.
Es waren bis zu 50% der aufgenommenen Messbilder verwackelt.

Abbildung 7.31 zeigt einen vergrößerten Ausschnitt aus Abbildung 7.30. Hier ist die Verschiebung zweier benachbarter Zeilen eindeutig zu erkennen. Durch diese Verschiebung kann die Position eines Laserpunktes im Pixelkoordinatensystem nicht mehr eindeutig zugewiesen werden. Der daraus resultierende Messfehler kann mehr als ein Prozent des Ergebnisses ausmachen. In den meisten Fällen ist der, aus verwackelten Bildern, errechnete Durchmesser

größer als der wahre Wert.

Eine Verringerung der Belichtungszeit der Kamera brachte keine beobachtbaren Verbesserungen, da die zeitliche Verzögerung zwischen zwei Halbbildern dadurch nicht verändert wird. Versuche, bei denen nur die Halbbilder ausgewertet wurden, lieferten auch nicht die gewünschte Genauigkeit. Die Messbilder waren zwar nicht mehr verwackelt, zur Auswertung stand aber nur noch die Hälfte der Information zur Verfügung.

Grobe Abweichungen von der Kreisform:

Wird auf einen Stab während dem Walzvorgang ein zu hoher Walzdruck ausgeübt, so entstehen an dessen Ober- und Unterseite Abflachungen. Das dadurch überschüssige Material weicht in den Walzspalt aus und verursacht dort Ausbauchungen am Stab.

Ein solcher Querschnitt wird schematisch in Abbildung 7.32 dargestellt.

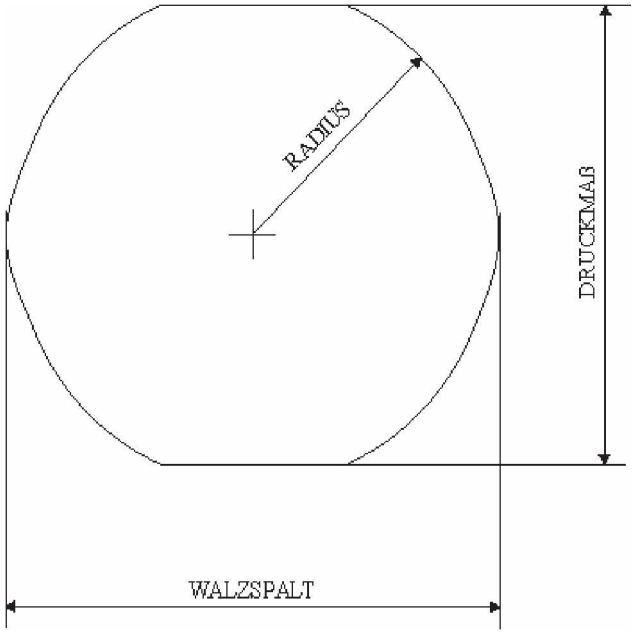


Abbildung 7.32: Abmessungen bei zu hohem Walzdruck

Der Durchmesser an den abgeflachten Stellen wird als "Druckmaß", der an den Ausbauchungen als "Walzspalt" bezeichnet. Zwischen diesen beiden Durchmessern liegt der Stabdurchmesser (= 2 Radien).

Wird ein abgeflachter Stab vermessen, so ist es mit nur einer Kamera sehr schwer, den korrekten Stabdurchmesser zu ermitteln. Die Abflachung bewirkt, dass bis zu 50% der Messpunkte nicht auf der idealen Kreisfunktion liegen. Werden diese Punkte in die Berechnung miteinbezogen, wird der errechnete Durchmesser größer als der wahre. Mit den verwendeten Algorithmen ist es nicht möglich, alle fehlerhaften Messpunkte zu eliminieren.

Die Ausbauchungen an den Seiten des Stabes beeinflussen das Messergebnis nicht, da diese Bereiche mit dem in Kapitel 4 beschriebenen Messaufbau nicht eingesehen werden können. Bei einer Erweiterung der Messung auf zwei Kameras sind diese zu berücksichtigen.

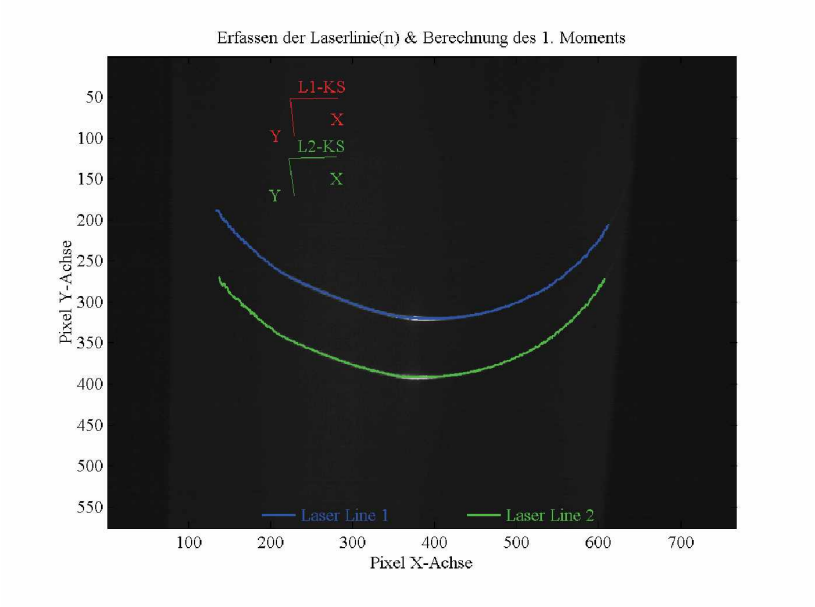


Abbildung 7.33: Abgeflachter Stab

In Abbildung 7.33 wird ein abgeflachter Stab vermessen. Die Abflachung befindet sich nicht exakt auf der Oberseite, da die Kamera nicht vertikal über dem Walzgut positioniert war.

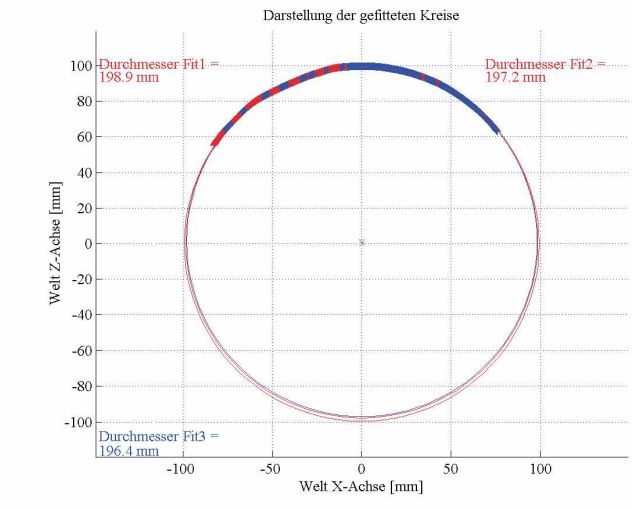


Abbildung 7.34: Messergebnis

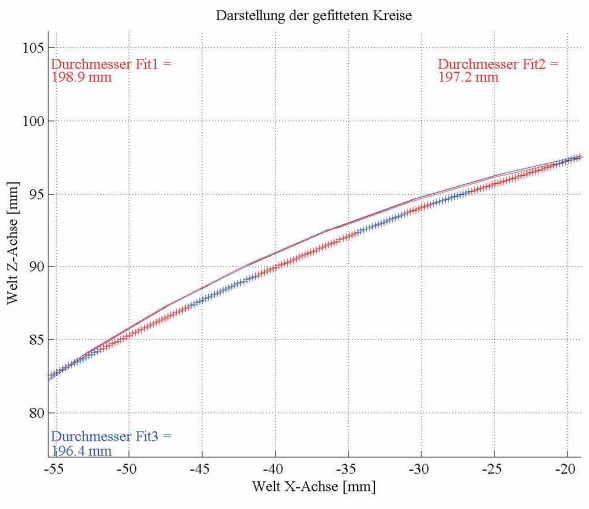


Abbildung 7.35: Abgeflachte Stelle

Abbildung 7.34 zeigt das Ergebnis der Berechnung. Es werden die meisten Messpunkte auf der Abflachung eliminiert, jedoch nicht alle. Der errechnete Durchmesser nach allen Eliminationsverfahren beträgt 196.4mm. Das aus dem Kaltmaß errechnete Warmmaß ist 194.3mm. Durch die Auswahl der Messpunkte, die für die Berechnung verwendet werden, nähert sich das Ergebnis dem wahren Wert an, erreicht diesen jedoch nicht.

$$D_{Fit1} > D_{Fit2} > D_{Fit3} > D \tag{7.14}$$

In Abbildung 7.35 ist die Abflachung aus Abbildung 7.34 vergrößert dargestellt. Diese Darstellung kann dazu verwendet werden, das Druckmaß zu bestimmen. Es wird der Abstand der Abflachung zum errechneten Durchmesser berechnet und unter der Annahme, dass der Stab symmetrisch ist, ergibt sich:

$$Druckmaß = D - 2\ Abstand \tag{7.15}$$

Durch eine geeignete Messanordnung zweier CCD-Kameras wäre auch der Bereich des Walzspaltes einer Messung zugänglich. Dann könnte die Höhe der Ausbauchung ermittelt, und der Durchmesser im Walzspalt berechnet werden.

Somit wären alle drei Maße aus Abbildung 7.32 bekannt.
Diese Abmessungen sind für die richtige Einstellung des Walzgerüstes notwendig. Das Ziel einer künftigen automatisierten Messanlage ist die Ermittlung dieser!

Kapitel 8

Fazit

Es wurde eine Methode zur berührungslosen Querschnittvermessung von glühendem Walzgut vorgestellt. Das Prinzip dieser Messmethode basiert auf dem Lichtschnittverfahren.
Dazu werden in diesem Kapitel die Vor- und Nachteile erörtert.

8.1 Vorteile des Verfahrens

- Das Warmmaß des Walzgutes wird gemessen. Messergebnisse können sofort für die Einstellung des Walzgerüstes verwendet werden.
- Die Auswertung eines Messbildes dauert nur wenige Sekunden.
- Das Walzgut wird durch die berührungslose Messung nicht beschädigt.
- Der Messaufbau ist relativ einfach und die Positionierung der optischen Komponenten ist flexibel.
- Es gibt keine beweglichen Teile und somit keine Verschleißteile.
- Die Anschaffungskosten der optischen Komponenten sind im Vergleich zu alternativen Systemen billig (Kapitel 8.5).
- Das System erfüllt die geforderte Messgenauigkeit von 0.5%.
- Mit ein paar Erweiterungen können mit dieser Methode dreidimensionale Abbildungen der Messobjekte erstellt werden.

8.2 Nachteile des Verfahrens

- Vor der Messung muss kalibriert werden. Fehler, die während der Kalibration verursacht werden, lassen sich bei der Messung nicht mehr beheben. Aus diesem Grund müssen die Abmessungen des Kalibrationsobjekts genau bekannt sein.

- Die Kalibration kann nur ohne Interferenzfilter durchgeführt werden. Bei dessen Montage nach der Kalibration kann die Position der Kamera verändert werden. Auch ist die Brechung des einfallenden Lichtes in die Kamera bei der Kalibration eine andere, als bei der Messung. Dies sind zusätzliche Fehlerquellen.
- Die optischen Komponenten müssen bei den vorherrschenden Bedingungen vor thermischer Zerstörung und vor Verschmutzung geschützt werden.
- Mit der verwendeten CCD-Kamera ist es nicht möglich, verwackelte Messbilder auszuschließen.
- Der in Kapitel 4 vorgestellte Messaufbau ist nicht geeignet, die Breite von Knüppeln zu vermessen.

8.3 Verbesserungsvorschläge

- Kalibration mit Interferenzfilter:

  Damit der Interferenzfilter auch bei der Kalibration verwendet werden kann, sollte ein neues Kalibrationsobjekt gebaut werden. Bei der Auswahl der dafür benötigten Leuchtdioden muss darauf geachtet werden, dass ihr Intensitätsmaximum im Wellenlängenbereich des Filters bzw. der Laser liegt.
  Somit ist die Brechung des einfallenden Lichts in die Kamera bei der Kalibration und bei der Messung ident. Auch ein unbeabsichtigtes Verrücken der Kameraposition, während der Filtermontage nach der Kalibration, wird damit ausgeschlossen.

- Verbesserung der Qualität der Messbilder:

  Um die Reproduzierbarkeit der Messergebnisse zu gewährleisten, müssen verwackelte Messbilder und die daraus resultierenden Messfehler beseitigt werden.
  Mit der verwendeten Aufnahmetechnik ist es nicht möglich, verwackelte Messbilder zu vermeiden, diese Bilder können jedoch mit statistischen Methoden so aufbereitet werden, dass der dadurch verursachte Fehler minimiert wird. Das Ziel ist nicht die Vermeidung der Vibrationen, vielmehr deren mathematische Korrektur.

- Korrektur zufälliger Messfehler:

  Bei der Auswertung der Messbilder entstehen immer zufällige Fehler. Diese Fehler entsprechen der Gauß'schen Normalverteilung und können durch Bildung des Mittelwertes aus vielen Messergebnissen eliminiert werden. Pro Stab werden derzeit zwei bis drei Messbilder per Hand aufgenommen und ausgewertet. Zur Reproduzierbarkeit der Messergebnisse sind viel mehr Messbilder pro Stab nötig.
  Es muss eine Software erstellt werden, die ca. 50-100 Messbilder pro Stab aufnimmt auswertet und aus den Ergebnissen den Mittelwert berechnet.

8.4 Zukünftige Entwicklungen

In dieser Arbeit wurde gezeigt, dass die Methode des Lichtschnittverfahrens geeignet ist, den Querschnitt von glühendem Walzgut hinreichend genau zu vermessen. Damit die Anwendung dieses Verfahrens industrietauglich wird, sind jedoch noch einige Erweiterungen vorzunehmen.
Bei einem zukünftiger Einsatz muss folgendes berücksichtigt werden:

Für die Automatisierung der Walzeneinstellung sind das Druckmaß, der Durchmesser im Walzspalt und der Durchmesser des Stabes nötig (Kapitel 7.2.4). Damit diese drei Maße ermittelt werden können, muss die Messung mit zwei CCD-Kameras durchgeführt werden. Die Anordnung der Kameras und Laser muss dabei so erfolgen, dass mehr als die Hälfte des Umfangs eines Stabes erfasst wird.
Weitere Vorteile zweier Kameras sind die größere Genauigkeit, bedingt durch die größere Anzahl von Messpunkten auf der Oberfläche, und die Möglichkeit einer Vermessung des Querschnitts von Knüppeln.

Während der Messungen vor Ort erwies sich die Kalibration als äußerst umständlicher und zeitaufwändiger Vorgang. Je nach Durchmesser der Stäbe erfolgt deren Austritt an unterschiedlichen Positionen an der Walze. Damit die Stäbe wieder in das Blickfeld der Kamera gebracht wurden, musste der Messbalken neu justiert und kalibriert werden. (Das Justieren des Messbalkens ohne die Position der Kamera und Laser zueinander zu verändern war nicht möglich.) Dies dauerte ca. 10 Minuten und während dieser Zeit stand die Walzanlage still!
Eine Lösungsvariante dieses Problems ist die bewegliche Montage des Messbalkens. Eine Schiene bzw. ein Balken wird quer über der Walzstraße montiert. Der Messbalken, aufgebaut auf einem Schlitten, kann auf dieser Schiene frei positioniert werden.
Die Kalibration kann dann neben der Walzstraße erfolgen und über eine Fernbedienung kann der Messbalken zu jeder Austrittsöffnung der Walze bewegt werden.

Besonders bei dünnen Stäben bzw. Knüppeln kann es passieren, dass diese nicht gerade aus der Walze herauskommen. Im schlechtesten Fall biegt sich das Walzgut nach oben und zerstört alles, was sich in dessen Laufrichtung befindet.
Daher muss aus Sicherheitsgründen der Messbalken ca. 3 Meter über der Walzstraße montiert werden. Dies erfordert eine hohe Anforderung an die Qualität von Laser und Kamera.

Ein weiteres Problem bereitet der Messbalken, an welchem die Laser und Kamera befestigt sind. Dieser darf sich nach erfolgter Kalibration auf keinen Fall durch die Temperatur verformen oder verdrehen. Eine kleine Änderung der Laser- und Kameraposition zueinander würde einen großen Fehler verursachen.
Neben dem Einsatz von Materialien mit kleinen Wärmedehnungskoeffizienten und temperaturbeständigen Profilen, wäre die Kalibration mit markanten Punkten im Hintergrund des Messbildes eine Lösung.

8.5 Alternativen zum Lichtschnittverfahren

Eine Alternative zu der in dieser Diplomarbeit beschriebenen Methode ist die Anwendung der "Conoscopic Holography". [13] Mithilfe dieser holographischen Methode ist es möglich, 3D-Konturen berührungslos zu vermessen. Im Folgenden wird das Prinzip kurz erklärt:

Licht ist eine elektromagnetische Welle, welche durch ihre Amplitude (Helligkeit), Frequenz (Farbe), Polarisierung (Polarisationsebene) und der Phase (Abstand) charakterisiert wird. Wird ein dreidimensionales Objekt beleuchtet, so werden von dessen Oberfläche charakteristische Wellen reflektiert. Mit einer CCD-Kamera ist es nicht möglich die Phase einer Welle zu messen, weshalb nur zweidimensional gemessen werden kann. (Ausnahme: Abhilfen wie beim Lichtschnittverfahren)

In der herkömmlichen holographischen Messtechnik wird mit einem, von der zu vermessenden Oberfläche, reflektierten Strahl und einem Referenzstrahl ein Interferenzbild erzeugt. Dieses Interferenzbild wird von einer CCD-Kamera aufgenommen und ausgewertet.
Der Referenzstrahl wird von einer kohärenten Lichtquelle (Laser) erzeugt. Beide Strahlen haben die gleiche Geschwindigkeit, legen jedoch unterschiedliche Wege zurück. Dieser Wegunterschied ist genau bekannt. Aus diesen Informationen kann die Phase des reflektierten Strahls berechnet, und der Abstand des Punktes zum Messobjekt bestimmt werden. So wird Punkt für Punkt die gesamte Oberfläche rekonstruiert.

Bei der "conoscopic holography" wird kein Laser als Referenzstrahl benötigt. Der vom Objekt reflektierte Strahl wird zuerst polarisiert und anschließend durch einen doppelbrechenden Kristall geführt. Durch diesen doppelbrechenden Kristall wird der Strahl geteilt und dadurch der benötigte Referenzstrahl erzeugt. Diese beiden Strahlen sind zueinander kohärent und somit kann ein Interferenzbild erstellt und ausgewertet werden. Punkt für Punkt wird wiederum die zu vermessende Oberfläche rekonstruiert.

Dieses Verfahren wurde von der Firma CONOLINE in einem Sensor umgesetzt. Dieser Sensor kann bis zu 18000 Punkte pro Sekunde abtasten und daraus die Oberfläche erzeugen. Dadurch können auch bewegte Teile vermessen werden. Ein weiterer Vorteil ist, dass der Sensor vor der Messung nicht kalibriert werden muss.
Nachteile des Sensors sind jedoch der hohe Preis, der ca. 13000 Euro pro Stück beträgt, und die Ungewissheit, ob der Sensor auch bei glühenden Oberflächen einsetzbar ist.

Anhang A

Begriffsdefinition

1. Stab

Unter Stab wird eine rundgewalzte Bramme verstanden.
Der Durchmesser beträgt je nach Anforderung von $50mm$ bis $250mm$. Abhängig vom Durchmesser ergibt sich die Länge des Stabes von $20m$ bis $140m$. Je dünner der Stab, desto länger ist er.

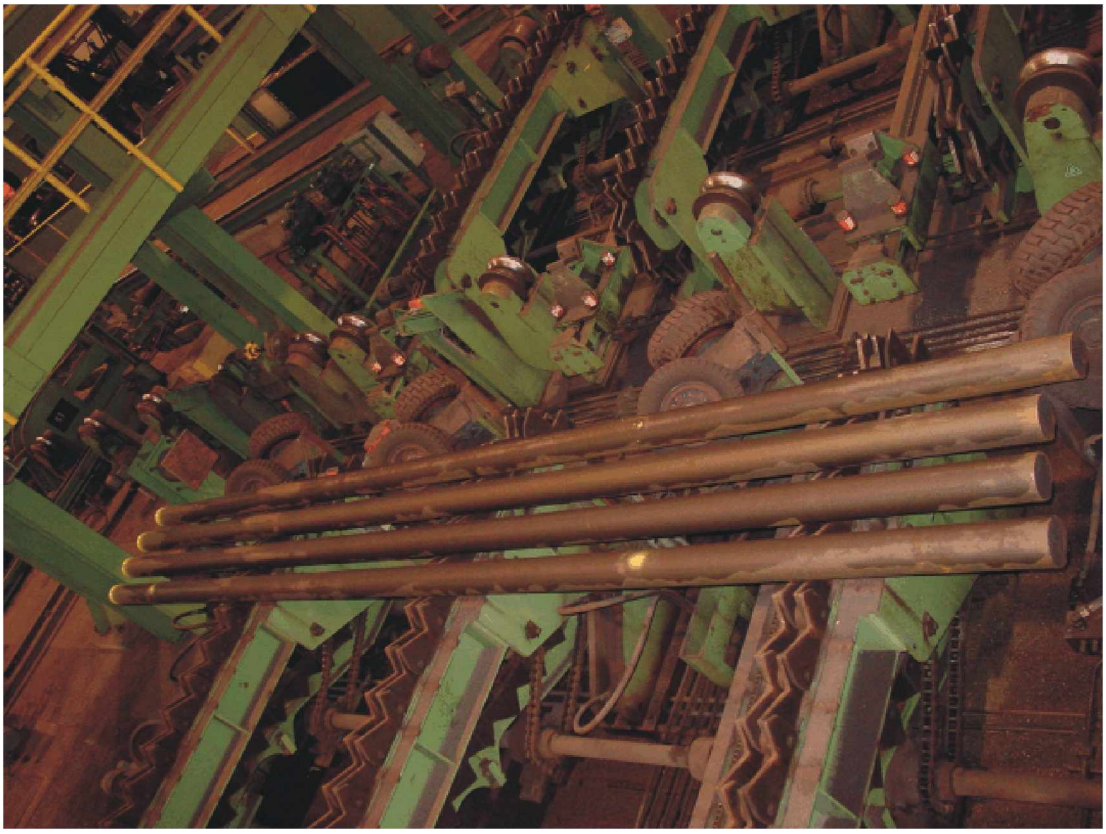


Abbildung A.1: Stäbe

2. Knüppel

Im Unterschied zum Stab werden die Knüppel viereckig gewalzt. Es gibt verschiedene Ausführungsformen für den Querschnitt, dies sind:

- quadratisch mit Radien an den Kanten:
  Seitenlänge von $50mm$ bis $90mm$
- quadratisch, scharfkantig:
  Seitenlänge von $50mm$ bis $160mm$
- rechteckig, abgerundete Kanten oder scharfkantig:
  Abmessungen: Breite von $80mm$ bis $350mm$,
  Höhe von $25mm$ bis $120mm$

Die Länge ist gleich wie beim Stab, abhängig vom Durchmesser, von $20m$ bis $140m$.

Abbildung A.2: Knüppel

3. Walzstraße

Auf der Walzstraße sind in konstanten Abständen Rollen montiert, auf denen das Walzgut transportiert wird. Über der Walzstraße wurden die Messungen durchgeführt.

Abbildung A.3: Walzstraße

4. Walzgerüst

Im Walzgerüst findet der Walzvorgang statt. Pro Gerüst werden die Stäbe bzw. Knüppel 3 bis 4 mal gewalzt. Dies geschieht, indem das Walzgut immer vorwärts und rückwärts durch das Walzgerüst geführt werden. Die Walzgeschwindigkeit ist abhängig vom Durchmesser und beträgt zwischen $2m/s$ und maximal $6m/s$. Ein einzelner Walzvorgang wird auch Stich genannt.
Bei Stäben wird erst der letzte Stich rundgewalzt.

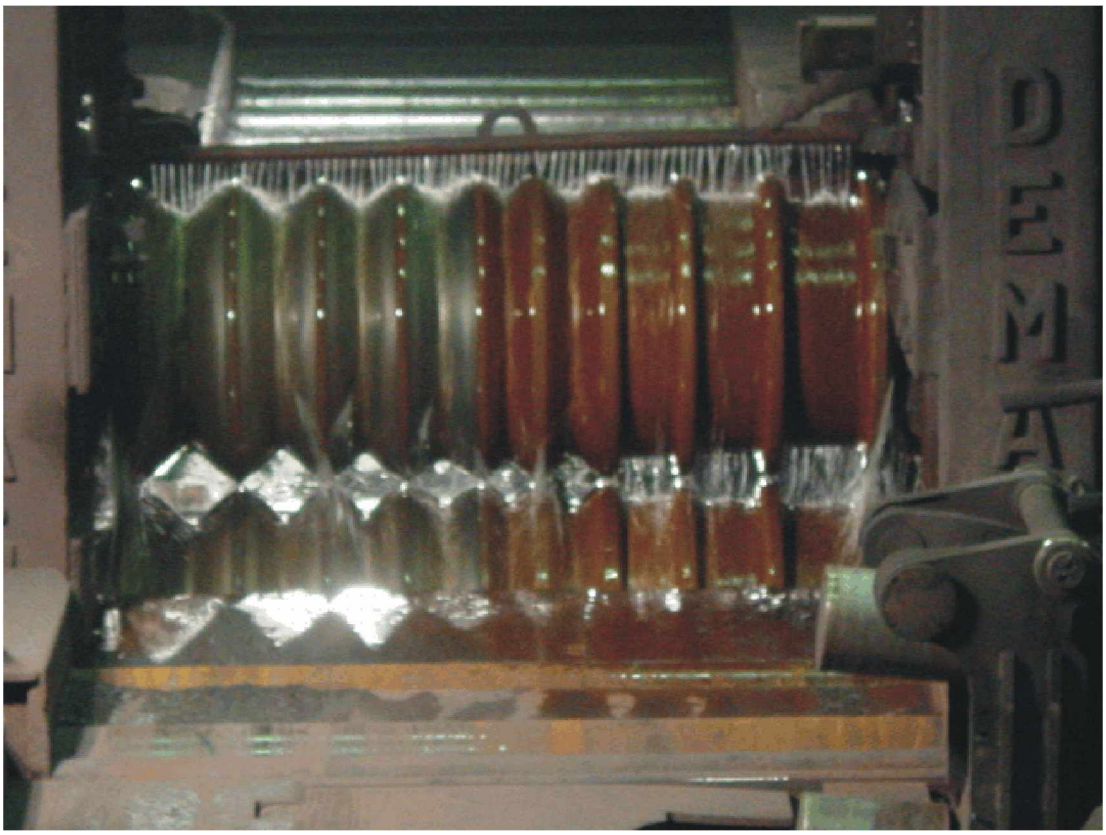


Abbildung A.4: Walzgerüst

5. Führungskasten

Nach dem letzten Stich am Walzgerüst wird das Walzgut im Anschluss durch den Führungskasten geführt. Die Vermessung des Querschnitts soll aus Sicherheitsgründen so nah wie möglich nach dem Führungskasten erfolgen.
Sollte das Walzgut ausbrechen, ist die Gefahr einer Demontage des Messkopfs näher am Führungskasten geringer.

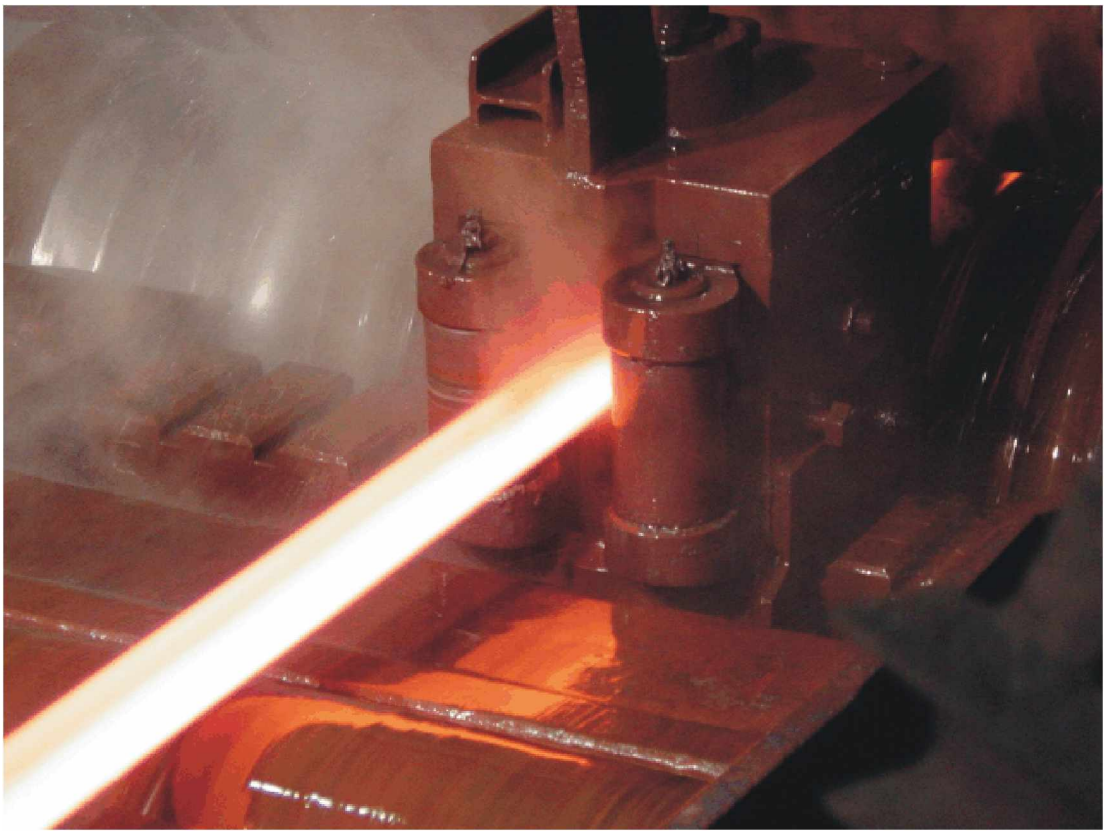


Abbildung A.5: Führungskasten

6. Greifkanter

Die Führung des Walzgutes in die Walze erfolgt mit einem Greifkanter, welcher vom Leitstand über der Walzanlage mit einem Joystick bedient wird. Wenn nötig, kann das Walzgut mit dem Greifkanter auch gedreht werden.

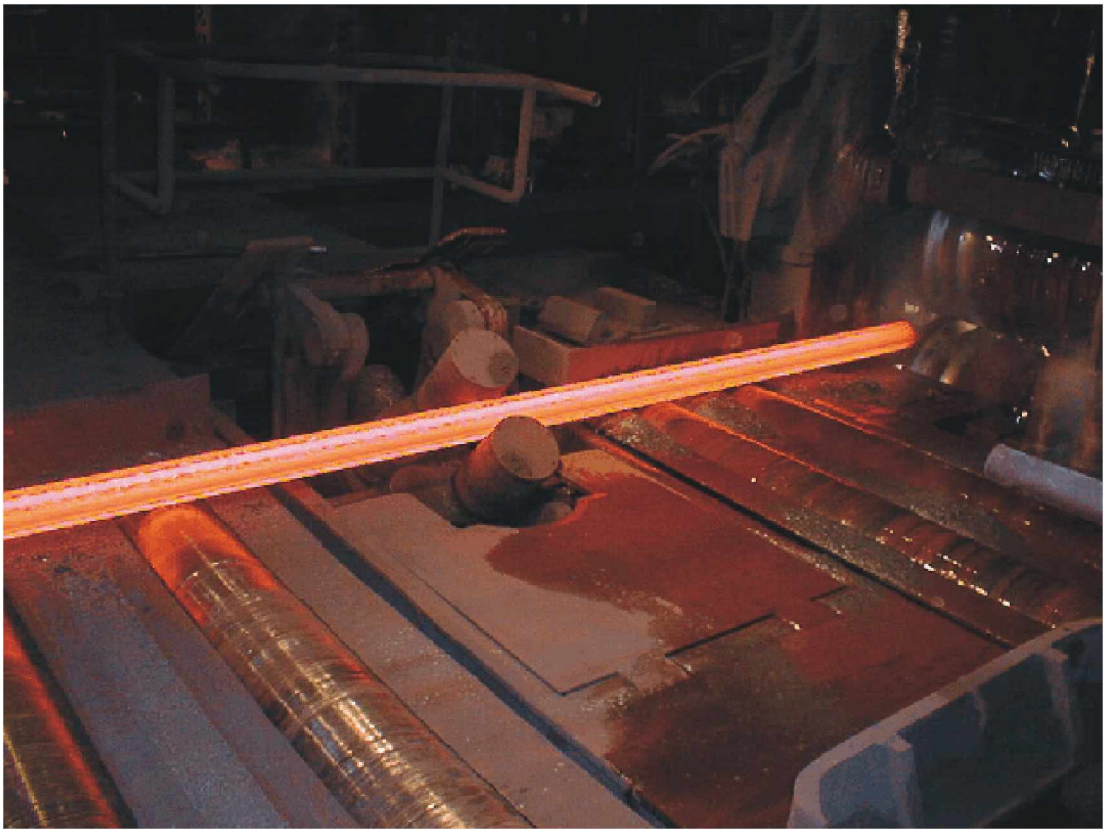


Abbildung A.6: Greifkanter

Anhang B

Abmessungen des Kalibrationsobjekts

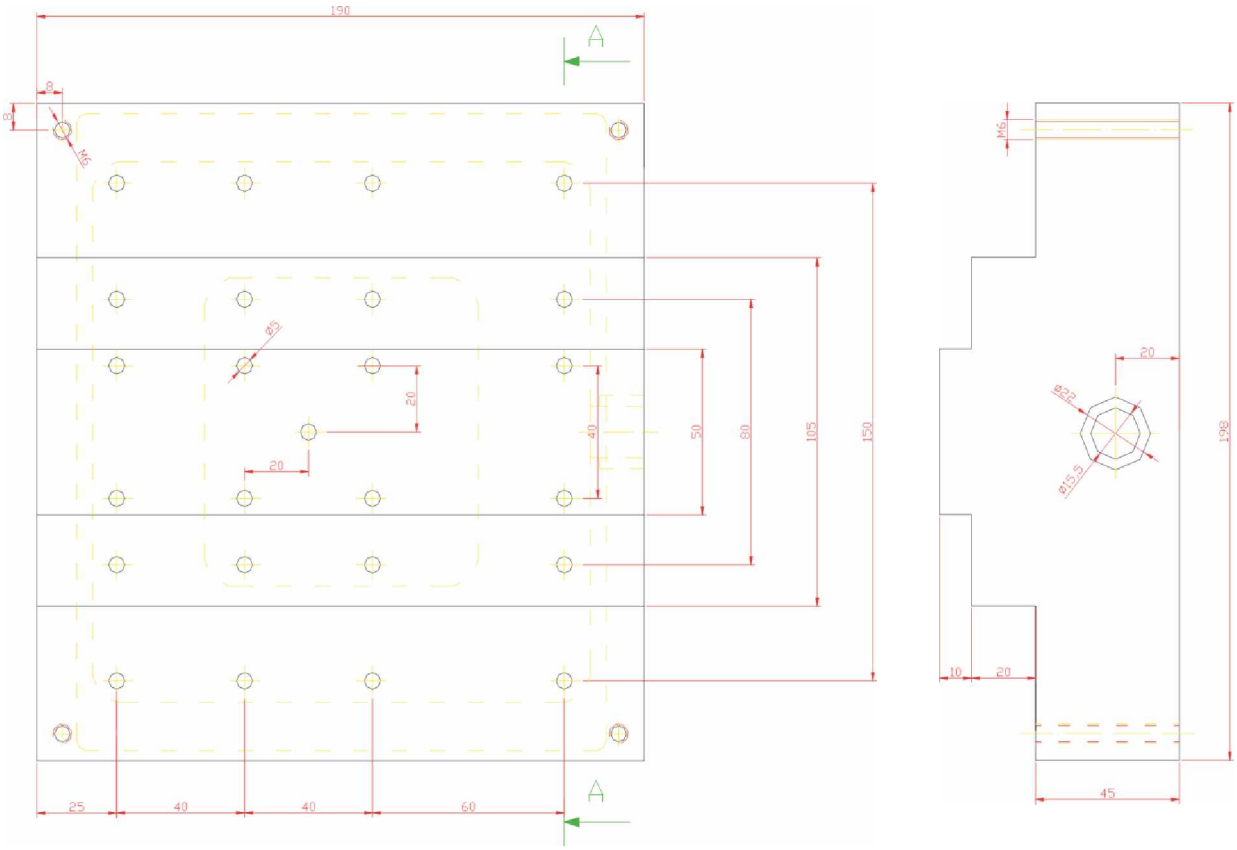


Abbildung B.1: Abmessungen des Kalibrationsobjekts

Anhang C

Matlab - Quellcode

In diesem Teil sind das komplette Haupt- und die Unterprogramme enthalten.

C.1 Hauptprogramm

"Main.m"

```
%
% Description:
%       Profile measurement of hot slab surfaces.
%
% Input Parameters:
%       A calibration image file.
%       A measurement image file.
%
% Return Parameters:
%       Dimensions of the hot slab.
%         in case of a round slab the diameter.
%         in case of a rectangle slab height.
%
% By:     Norbert Koller
% Date:   March - October, 2002
% Version: Original version
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, m9535098@stud.unileoben.ac.at
%
% History:
% Date:   03.2002 - 10.2002
% Comment:
%
% To run this file following subfiles are needed:
%
% Bubblesort.m
% CalculateCircle.m
% Calibration.m
% FindLaserLines.m
% FitArc.m
% FitCircle.m
% FitCirclenew.m
% FitLaserPlane.m
% FitLine.m
% FitParLine.m
% GenProj.m
```

```
% MergeLaserLine.m
% OptimizeCircle.m
% PlotCircle.m
% RotateCircle.m
%
clear;
close all;
%
Exit=0;
KalibrationOK=0;
%
while Exit == 0
  Selection=menu('Menu','Exit','Kalibrieren','Kalibrationsdaten speichern','Kalibrationsdaten laden','Messung');
  switch Selection
  %
  case 1 % Verlassen des Programms
     Exit=1;
     %
  case 2 % Kalibration
     clear;
     close all;
     Exit=0;
     %SetUpGraphics(12); % Unterprogramme, welche nur für die Diagramme
     %A4Figure;              % benötigt werden !!
     %
     % Auswahl eines Kalibrationsbildes
     %
     % File Format "tagged image file" bzw. "*.tif"
     ImageFileTypes='*.tif';
     % Auswahl
     [Imagefilename, Imagepathname]=uigetfile(ImageFileTypes,'Bitte wählen Sie ein Kalibrationsbild:');
     %
     if Imagefilename == 0
       disp('Das gewählte Bild konnte nicht geladen werden');
       disp('Bitte wählen Sie ein korrektes Kalibrationsbild');
       %
     else
       % Einlesen des Files
       Pic=imread([Imagepathname,Imagefilename]);
       DocuName=Imagefilename(1:(findstr(Imagefilename,'.')-1)); % Sicherung des Files
       Pic0=im2double(Pic); %Das Bild wird in ein 16bit Format verwandelt
       %
       OriginalFig=figure;
       imagesc(Pic0); % Darstellung des Bildes
       axis image; % Darstellung der Achsen
       colormap(gray);
       %
       [KalibrationOK,Laser1T,Laser2T,ProjektionKameraLaser1,ProjektionKameraLaser2,Plane1,Plane2,Verkehrt]...
           = Calibration(Pic0);
       %
       if KalibrationOK == 1
         disp('Die Kalibration wurde erfolgreich durchgeführt!');
       else
         disp('Die Kalibration konnte nicht durchgeführt werden!');
       end
       %
     end
     %
  case3 % Kalibrationsdaten speichern
     %
     if KalibrationOK == 0
       disp('Vor dem Speichern muss erst kalibriert werden!');
       %
     else
       savefilename = 'CalibData.mat';
       save(savefilename,'Laser1T','Laser2T','ProjektionKameraLaser1','ProjektionKameraLaser2','Plane1','Plane2','Verkehrt');
     end
  end
```

```
%
case4 % Kalibrationsdaten laden
  %
  clear;
  close all;
  Exit=0;
  %
  loadfilename = 'CalibData.mat';
  load(loadfilename);
  %
  KalibrationOK = 1;
  %
case 5 % Messung
  %
  Exit=0;
  close all;
  %
  if KalibrationOK == 0
    %
    disp('Vor der Messung muss erst kalibriert werden!');
    %
  else
    % Sichern der Kalibrationsdaten
    savefilename = 'Data.mat';
    save(savefilename,'Laser1T','Laser2T','ProjektionKameraLaser1','ProjektionKameraLaser2','Plane1','Plane2','Verkehrt');
    %
    % Löschen aller Variablen
    % ==> Vermeidung, dass bei mehreren Messungen noch Daten in den Variablen stehen!!
    clear;
    %
    % Laden der Kalibrationsdaten
    loadfilename = 'Data.mat';
    load(loadfilename);
    %
    Exit=0;
    KalibrationOK = 1;
    %
    % Auswahl eines Messbildes
    ImageFileTypes='*.tif'; % File Type "tagged image file"
    [Imagefilename,Imagepathname]=uigetfile(ImageFileTypes,'Select image file'); %Auswahl
    if Imagefilename == 0
      disp('Das gewählte Bild konnte nicht geladen werden');
      disp('Bitte wählen Sie ein korrektes Kalibrationsbild');
    else
      Pic=imread([Imagepathname,Imagefilename]); % Einlesen des Bildes
      DocuName=Imagefilename(1:(findstr(Imagefilename,'.')-1)); % Sicherung
      Pic0=im2double(Pic); % Umschalten auf ein 16bit Format
      %
      fig = figure;
      imagesc(Pic0); % Darstellung
      axis image; % Achsen
      colormap(gray);
      hold on;
      title('Messbild');
      xlabel('Pixel X-Achse');
      ylabel('Pixel Y-Achse');
      %
      % Finden der Laserlinien & Errechnung des 1.Intensitätsmoments
      %
      [LaserPoints1,Length1,LaserPoints2,Length2] = FindLaserLines(Pic0,Verkehrt);
      %
      % Umrechnung der gefundenen Laserpunkte in das Laserkoordinatensystem
      %
      PointsLaser1_NA = ProjektionKameraLaser1 * [LaserPoints1(1,:);LaserPoints1(2,:);ones(1,length(LaserPoints1))];
      PointsLaser1_A = [PointsLaser1_NA(1,:)./PointsLaser1_NA(3,:);PointsLaser1_NA(2,:)./PointsLaser1_NA(3,:);...
          zeros(1,length(PointsLaser1_NA));ones(1,length(PointsLaser1_NA))];
      %
      % Umrechnung der Punkte in das WeltKoordinatensystem
```

```
%
PunkteWelt1 = inv(Laser1T) * PointsLaser1_A;
%
% Einzeichnen des Laserkoordinatensystems
%
% Berechnung des Ursprungs und zweier Hilfspunkte
%
U_NA = inv(ProjektionKameraLaser1)*[0 0 1]';
U_A = [U_NA(1,1)/U_NA(3,1);U_NA(2,1)/U_NA(3,1);1];
P1_NA = inv(ProjektionKameraLaser1)*[20 0 1]';
P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
P2_NA = inv(ProjektionKameraLaser1)*[0 20 1]';
P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
%
% Zeichnen des Koordinatenursprungs
%
line([U_A(1,1) P1_A(1,1)],[U_A(2,1) P1_A(2,1)],'color','r');
line([U_A(1,1) P2_A(1,1)],[U_A(2,1) P2_A(2,1)],'color','r');
text(U_A(1,1)+10,U_A(2,1)-15,'L1-KS','color','r');
text(U_A(1,1)-25,P2_A(2,1),'Y','color','r');
text(U_A(1,1)+50,U_A(2,1)+25,'X','color','r');
%
% Zusammenfügen der Liniensegmente
%
[Laser1Circle,Laser1Line1,Laser1Line2] = MergeLaserLine(PunkteWelt1,Length1);
%
if Length2(1) > 0
  % Es ist eine zweite Laserlinie vorhanden
  % Umrechnung der gefundenen Laserpunkte in das Laserkoordinatensystem
  %
  PointsLaser2_NA = ProjektionKameraLaser2 * [LaserPoints2(1,:);LaserPoints2(2,:);ones(1,length(LaserPoints2))];
  PointsLaser2_A = [PointsLaser2_NA(1,:)./PointsLaser2_NA(3,:);PointsLaser2_NA(2,:)./PointsLaser2_NA(3,:);...
     zeros(1,length(PointsLaser2_NA));ones(1,length(PointsLaser2_NA))];
  %
  % Umrechnung der Punkte in das Weltkoordinatensystem
  %
  PunkteWelt2 = inv(Laser2T) * PointsLaser2_A;
  %
  % Einzeichnen des Laserkoordinatensystems
  %
  % Berechnung des Ursprungs und zweier Hilfspunkte
  %
  U_NA = inv(ProjektionKameraLaser2)*[0 0 1]';
  U_A=[U_NA(1,1)/U_NA(3,1);U_NA(2,1)/U_NA(3,1);1];
  P1_NA = inv(ProjektionKameraLaser2)*[20 0 1]';
  P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
  P2_NA = inv(ProjektionKameraLaser2)*[0 20 1]';
  P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
  %
  line([U_A(1,1) P1_A(1,1)],[U_A(2,1) P1_A(2,1)],'color','g');
  line([U_A(1,1) P2_A(1,1)],[U_A(2,1) P2_A(2,1)],'color','g');
  text(U_A(1,1)+10,U_A(2,1)-15,'L2-KS','color','g');
  text(U_A(1,1)-25,P2_A(2,1),'Y','color','g');
  text(U_A(1,1)+50,U_A(2,1)+25,'X','color','g');
  %
  % Zusammenfügen der Liniensegmente
  %
  [Laser2Circle,Laser2Line1,Laser2Line2] = MergeLaserLine(PunkteWelt2,Length2);
  %
  if length(Laser2Circle.XData) > 1
    % Es wurde ein zylindrisches Objekt vermessen !
    %
    % Die Punkte werden parallel zur y-Achse gedreht
    %
    [Laser1CircleKorr,Laser2CircleKorr] = RotateCircle(Laser1Circle,Laser2Circle,Plane1,Plane2);
    %
    % Neue Kreise werden berechnet
    %
```

```
    Antwort = FitCirclenew([(Laser1CircleKorr(1,:)-mean(Laser1CircleKorr(1,:)));...
        (Laser1CircleKorr(3,:)-mean(Laser1CircleKorr(3,:)))]);
    Circle1 = [(Laser1CircleKorr(1,:)-mean(Laser1CircleKorr(1,:))-Antwort(1));...
        zeros(1,length(Laser1CircleKorr));Laser1CircleKorr(3,:)-mean(Laser1CircleKorr(3,:))-Antwort(2)];
    Circle1R = Antwort(3);
    Antwort = FitCirclenew([(Laser2CircleKorr(1,:)-mean(Laser2CircleKorr(1,:)));...
        (Laser2CircleKorr(3,:)-mean(Laser2CircleKorr(3,:)))]);
    Circle2 = [(Laser2CircleKorr(1,:)-mean(Laser2CircleKorr(1,:))-Antwort(1));...
        zeros(1,length(Laser2CircleKorr));Laser2CircleKorr(3,:)-mean(Laser2CircleKorr(3,:))-Antwort(2)];
    Circle2R = Antwort(3);
    %
  end
  %
end
%
% Zeichnen der Laserebenen
%
fig3d = figure;
hold on;
axis equal;
%
title('Darstellung der Messpunkte');
xlabel('X-Achse');
ylabel('Y-Achse');
zlabel('Z-Achse');
%
%camlight left;
lighting flat; % diffuses Licht
view(110,10); % Betrachtungswinkel
%
% Berechnung der Bildabmessungen
%
if length(Laser1Circle.XData) > 1
  % Es wurde ein Zylinder vermessen
  Xmin = min(Laser1Circle.XData) - 40;
  Xmax = max(Laser1Circle.XData) + 40;
  Zmin = max(Laser1Circle.ZData)-1.1*Laser1Circle.R;
  %
  if Length2(1) > 0
    Ymax = max(Laser2Circle.YData) + 30;
  else
    Ymax = max(Laser1Circle.YData) + 30;
  end
else
  % Es wurde ein Quader vermessen
  Xmin = min(PunkteWelt1(1,:)) - 20;
  Xmax = max(PunkteWelt1(1,:)) + 20;
  Zmin = min(PunkteWelt1(3,:));
  if Length2(1) > 0
    Ymax = max(PunkteWelt2(2,:)) + 50;
  else
    Ymax = max(PunkteWelt1(2,:)) + 80;
  end
end;
%
Zmax = max(PunkteWelt1(3,:)) + 80;
%
% Berechnung der Laserebene
      Punkt1x = Xmin;
Punkt1z = Zmin;
Punkt1y = -(Punkt1x*Plane1(1)+Punkt1z*Plane1(3)+Plane1(4))/Plane1(2);
Punkt2x = Xmax;
Punkt2z = Zmin;
Punkt2y = -(Punkt2x*Plane1(1)+Punkt2z*Plane1(3)+Plane1(4))/Plane1(2);
Punkt3x = (Punkt1x+Punkt2x)/2;
Punkt3z = Zmax;
Punkt3y = -(Punkt3x*Plane1(1)+Punkt3z*Plane1(3)+Plane1(4))/Plane1(2);
Ymin = min([Punkt1y Punkt2y Punkt3y]) - 20;
```

```
%
axis([Xmin-10 Xmax+10 Ymin-10 Ymax+10 Zmin-10 Zmax+10]);
%
XLaserPlane = [Punkt1x Punkt2x Punkt3x];
YLaserPlane = [Punkt1y Punkt2y Punkt3y];
ZLaserPlane = [Punkt1z Punkt2z Punkt3z];
s1=patch(XLaserPlane,YLaserPlane,ZLaserPlane,[0.7 0 0.3],'FaceAlpha',0.1);
%
% Einzeichnen des Bodens
%
XGroundZero = [Xmin-5 Xmax+5 Xmax+5 Xmin-5];
YGroundZero = [Ymin-5 Ymin-5 Ymax+5 Ymax+5];
ZGroundZero = [Zmin Zmin Zmin Zmin];
patch(XGroundZero,YGroundZero,ZGroundZero,'green');
%
% Einzeichnen des Koordinatensystems
%
line([0 Xmax],[0 0],[0 0],'color','black','Marker','>');
line([0 0],[0 Ymax],[0 0],'color','black','Marker','>');
line([0 0],[0 0],[0 Zmax],'color','black','Marker','>');
%
% Einzeichnen der Messpunkte
%
if length(Laser1Circle.XData) > 1
  s2=plot3(Laser1Circle.XData,Laser1Circle.YData,Laser1Circle.ZData,'-blue');
elseif length(Laser1Line1.XData) > 1
  s2=plot3(Laser1Line1.XData,Laser1Line1.YData,Laser1Line1.ZData,'-blue');
  if length(Laser1Line2.XData) > 1
    plot3(Laser1Line2.XData,Laser1Line2.YData,Laser1Line2.ZData,'-blue');
  end
end
%
if Length2(1) > 0
  %
  % Einzeichnen der 2. Laserebene und der Punkte
  %
  Punkt1x = Xmin;
  Punkt1z = Zmin;
  Punkt1y = -(Punkt1x*Plane2(1)+Punkt1z*Plane2(3)+Plane2(4))/Plane2(2);
  Punkt2x = Xmax;
  Punkt2z = Zmin;
  Punkt2y = -(Punkt2x*Plane2(1)+Punkt2z*Plane2(3)+Plane2(4))/Plane2(2);
  Punkt3x = (Punkt1x+Punkt2x)/2;
  Punkt3z = Zmax;
  Punkt3y = -(Punkt3x*Plane2(1)+Punkt3z*Plane2(3)+Plane2(4))/Plane2(2);
  %
  XLaserPlane = [Punkt1x Punkt2x Punkt3x];
  YLaserPlane = [Punkt1y Punkt2y Punkt3y];
  ZLaserPlane = [Punkt1z Punkt2z Punkt3z];
  patch(XLaserPlane,YLaserPlane,ZLaserPlane,[0.7 0 0.3],'FaceAlpha',0.1);
  %
  if length(Laser2Circle.XData) > 1
    % Einzeichnen der gemessenen Punkte
    plot3(Laser2Circle.XData,Laser2Circle.YData,Laser2Circle.ZData,'-blue');
    %
    % Einzeichnen des Mittelpunktes
    %
    Vektor1(1) = Laser1Circle.X0;
    Vektor1(3) = Laser1Circle.Z0;
    Vektor1(2) = -(Vektor1(1)*Plane1(1)+Vektor1(3)*Plane1(3)+Plane1(4))/Plane1(2);
    plot3(Vektor1(1),Vektor1(2),Vektor1(3),'+black');
    Vektor2(1) = Laser2Circle.X0;
    Vektor2(3) = Laser2Circle.Z0;
    Vektor2(2) = -(Vektor2(1)*Plane2(1)+Vektor2(3)*Plane2(3)+Plane2(4))/Plane2(2);
    plot3(Vektor2(1),Vektor2(2),Vektor2(3),'+black');
    s4=line([Vektor1(1) Vektor2(1)],[Vektor1(2) Vektor2(2)],[Vektor1(3) Vektor2(3)],'color','black','Linewidth',2);
    %
    % Einzeichnen des Mittelpunktvektors des Zylinders
```

```
%
Punkt1x = 2*Vektor1(1) - Vektor2(1);
Punkt1y = 2*Vektor1(2) - Vektor2(2);
Punkt1z = 2*Vektor1(3) - Vektor2(3);
Punkt2x = 2*Vektor2(1) - Vektor1(1);
Punkt2y = 2*Vektor2(2) - Vektor1(2);
Punkt2z = 2*Vektor2(3) - Vektor1(3);
line([Punkt1x Punkt2x],[Punkt1y Punkt2y],[Punkt1z Punkt2z], 'color','black','LineWidth',2,'Marker','>');
%
% Zeichnen der transformierten Punkte
%
s3=plot3(Laser1CircleKorr(1,:),Laser1CircleKorr(2,:),Laser1CircleKorr(3,:),'-yellow');
plot3(Laser2CircleKorr(1,:),Laser2CircleKorr(2,:),Laser2CircleKorr(3,:),'-yellow');
legend_handles = [s1;s2;s3;s4];
legend(legend_handles,'Laserebenen','gemessene Punkte','gedrehte Punkte','Mittelpunktvektor',1);
%
% Zeichnen der projizierten Zylinderpunkte
% Der neue gemeinsame Mittelpunkt wird auf die y-Achse gelegt
% ===> Gemeinsamer Mittelpunkt [0;y;0]
fig3d = figure;
hold on;
axis equal;
%
title('Projektion der gemessenen Punkte in die X-Z Ebene');
xlabel('X-Achse');
ylabel('Y-Achse');
zlabel('Z-Achse');
%
%camlight left;
lighting flat;
view(130,10);
%
axis([min(Circle1(1,:))-40 max(Circle1(1,:))+40-10 max(mean(Laser1CircleKorr(2,:)),...
    mean(Laser2CircleKorr(2,:)))+40 Zmin-10 max(Circle1(3,:))+50]);
%
% Einzeichnen des Bodens
%
XGroundZero = [min(Circle1(1,:))-30 max(Circle1(1,:))+30 max(Circle1(1,:))+30 min(Circle1(1,:))-30];
YGroundZero = [-10 -10 max(mean(Laser1CircleKorr(2,:)),mean(Laser2CircleKorr(2,:)))+30...
    max(mean(Laser1CircleKorr(2,:)),mean(Laser2CircleKorr(2,:)))+30];
patch(XGroundZero,YGroundZero,ZGroundZero,'green');
%
% Einzeichnen des Koordinatensystems
%
line([min(Circle1(1,:))-40 max(Circle1(1,:))+40],[0 0],[0 0],'color','black','Marker','>');
line([0 0],[0 max(mean(Laser1CircleKorr(2,:)),mean(Laser2CircleKorr(2,:)))+40],[0 0],'color','black','Marker','>');
line([0 0],[0 0],[0 max(Circle1(3,:))+40],'color','black','Marker','>');
%
% Einzeichnen der x-z-Ebene
%
XBackPlane = XGroundZero;
YBackPlane = [0 0 0 0];
ZBackPlane = [Zmin Zmin max(Circle1(3,:))+30 max(Circle1(3,:))+30];
%
s1=patch(XBackPlane,YBackPlane,ZBackPlane,'black');
%
% Einzeichnen der korrigierten und projizierten Messpunkte
%
for i = 1 : length(Laser1CircleKorr(2,:))
  x(i) = mean(Laser1CircleKorr(2,:));
end
s2=plot3(Circle1(1,:),x,Circle1(3,:),'+r');
x = 0;
for i = 1 : length(Laser2CircleKorr(2,:))
  x(i) = mean(Laser2CircleKorr(2,:));
end
plot3(Circle2(1,:),x,Circle2(3,:),'+r');
%
```

```
    plot3(Circle1(1,:),Circle1(2,:),Circle1(3,:),'+cyan');
    s3=plot3(Circle2(1,:),Circle2(2,:),Circle2(3,:),'+cyan');
    %
    for i = 1 : length(Circle2(1,:))
      line([Circle2(1,i) Circle2(1,i)],[mean(Laser2CircleKorr(2,:)) Circle2(2,i)],[Circle2(3,i) Circle2(3,i)],'color','b');
    end
    %
    plot3(0,mean(Laser1CircleKorr(2,:)),0,'+r');
    plot3(0,mean(Laser2CircleKorr(2,:)),0,'+r');
    plot3(0,0,0,'+r');
    line([0 0],[mean(Laser2CircleKorr(2,:)) 0],[0 0],'color','r');
    legend_handles = [s1;s2;s3];
    legend(legend_handles,'X-Z Ebene','gedrehte Punkte','projizierte Punkte',1);
    %
    % Darstellung der gemessenen Punkte in der X-Z Ebene
    %
    fig = figure;
    hold on;
    axis equal;
    Xmin = min(min(Circle1(1,:)),min(Circle2(1,:))) - 10;
    Xmax = max(max(Circle1(1,:)),max(Circle2(1,:))) + 10;
    Ymax = max(Circle1(3,:))+10;
    axis([Xmin Xmax -5 Ymax]);
    title('Darstellung der projizierten Messpunkte');
    xlabel('Welt X-Achse [mm]');
    ylabel('Welt Z-Achse [mm]');
    s1=plot(Circle1(1,:),Circle1(3,:),'r-');
    s2=plot(Circle2(1,:),Circle2(3,:),'b-');
    s3=plot(0,0,'r+');
    legend_handles = [s1;s2;s3];
    legend(legend_handles,'Messpunkte Laser1','Messpunkte Laser2','gemeinsamer Mittelpunkt');
    %
  elseif length(Laser2Line1.XData) > 1
    plot3(Laser2Line1.XData,Laser2Line1.YData,Laser2Line1.ZData,'-blue');
    if length(Laser2Line2.XData) > 1
      plot3(Laser2Line2.XData,Laser2Line2.YData,Laser2Line2.ZData,'-blue');
    end
    legend_handles = [s1;s2];
    legend(legend_handles,'Laserebene','gemessene Punkte',1);
    %
  end
  %
else
  legend_handles = [s1;s2];
  legend(legend_handles,'Laserebene','gemessene Punkte',1);
end
%
% Fitten der korrigierten Punkte (Nur bei runden Objekten)
% Es wird nur noch die X-Z-Ebene betrachtet, da im Falle von zwei Laserlinien
% die gemessenen Punkte bereits in die X-Z-Ebene gedreht wurden.
if length(Laser1Circle.XData) > 1
  %
  if Length2(1) > 0
    %
    % Kreisfit und Darstellung der Ergebnisse
    %
    [Kreis1Fit1,Kreis1Fit2,Kreis1Fit3] = CalculateCircle([Circle1(1,:);Circle1(3,:)]);
    [Kreis2Fit1,Kreis2Fit2,Kreis2Fit3] = CalculateCircle([Circle2(1,:);Circle2(3,:)]);
    %Laser_1=2*Laser1Circle.R
    %Laser_2=2*Laser2Circle.R
  else
    [Kreis1Fit1,Kreis1Fit2,Kreis1Fit3] = CalculateCircle([Laser1Circle.XData;Laser1Circle.ZData]);
  end
  %
else
  if length(Laser1Line1.XData) > 1 & length(Laser1Line2.XData) == 1
  % Fitten der zusammengesetzten Liniensegmente
    AntwortLinie = FitLine(Laser1Linie1.XData,Laser1Line1.ZData);
```

```
          %
          Ymax = max(Laser1Line1.YData) + 30;
          %
        elseif length(Laser1Line2.XData) > 1
          % von der vorderen Linien werden nur die mittleren 2/3 genommen
          n = length(Laser1Line2.XData);
          Linie1.XData = Laser1Line1.XData;
          Linie1.YData = Laser1Line1.ZData;
          Linie2.XData = Laser1Line2.XData(round((1/6)*n) : round((5/6)*n));
          Linie2.YData = Laser1Line2.ZData(round((1/6)*n) : round((5/6)*n));
          % Es werden 2 parallele Linien gefittet
          AntwortLinie = FitParLine(Linie1,Linie2);
          %
          Ymax = max(Laser1Line2.ZData) + 30;
        end
        %
        Xmax = max(Laser1Line1.XData) + 10;
        Xmin = min(Laser1Line1.XData) - 10;
        Ymin = min(Laser1Line1.ZData) - 10;
        %
        % Zeichnen des Welt - Koordinatensystems (Nur X,Z -Ebene)
        %
        fig=figure;
        hold on;
        title('Auswertung der Messergebnisse');
        axis equal;
        xlabel('Welt X-Achse [mm]');
        ylabel('Welt Z-Achse [mm]');
        %
        % Begrenzung der Achsen
        %
        axis([Xmin Xmax Ymin Ymax]);
        grid on;
        %
        % Einzeichnen der Messpunkte
        %
        s1 = plot(Laser1Line1.XData,Laser1Line1.ZData,'+blue');
        %
        % Einzeichnen der gefitteten Linien
        %
        Punkt1X = 0;
        Punkt1Z = AntwortLinie.D1;
        Punkt2X = Xmax;
        Punkt2Z = AntwortLinie.K*Xmax + AntwortLinie.D1;
        s2=plot([Punkt1X Punkt2X],[Punkt1Z Punkt2Z],'color','red');
        %
        if length(Laser1Line2.XData) > 1
          s3=plot(Laser1Line2.XData,Laser1Line2.ZData,'+green');
          Punkt1X = min(Laser1Line2.XData) - 10;
          Punkt1Z = Punkt1X*AntwortLinie.K + AntwortLinie.D2;
          Punkt2X = max(Laser1Line2.XData) + 10;
          Punkt2Z = Punkt2X*AntwortLinie.K + AntwortLinie.D2;
          s4=plot([Punkt1X Punkt2X],[Punkt1Z Punkt2Z],'color','black');
          text(Xmin+11,Ymax - 12,'Höhe=','color','blue','FontSize',12);
          text(Xmin+11,Ymax - 18,num2str(AntwortLinie.D2-AntwortLinie.D1,4),'color','blue','FontSize',12);
          text(Xmin+30,Ymax - 18,'mm','color','blue','FontSize',12);
          legend_handles = [s1;s2;s3;s4];
          legend(legend_handles,'Messpunkte Hintergrund','Hintergrund','Messpunkte Knüppel','Knüppel');
        end
      end
    end
  end
end
```

C.2 Unterprogramme

"Bubblesort.m"[1]

```
%
function Bubble=Bubblesort(List,Row);
%
% Description:
%           This function sorts a given list from the smallest value to the largest.
%           If the input list is a matrix than the list will
%           be sorted by the row, given in 'Row'.
%
% Input Parameters:
%           A given List.
%           Rownumber by which the list should be sorted.
%
% Return Parameters:
%           Sorted list.
%
% By:       Norbert Koller
% Date:     17. April 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
Laenge = length(List);
%
if Row > size(List,1) % ist die Zeilennummer, größer als die Zeilenanzahl,
   Row = size(List,1); % dann wird das Array nach der letzten Zeile sortiert.
end
%
for i = 1 : Laenge
   for j = 1 : (Laenge-i)
     if List(Row,j) > List(Row,j+1)
       Merker=List(:,j);
       List(:,j)=List(:,j+1);
       List(:,j+1)=Merker;
     end;
   end;
end;
%
Bubble=List;
```

"CalculateCircle.m"

```
%
function [KreisFit1,KreisFit2,KreisFit3] = CalculateCircle(Circle);
%
% Description:
%           This function calculates the best circle fit to a given set of points.
%
%
% Input Parameters:
%           Circlepoints.
%
% Return Parameters:
%           Fitted circle parameter.
%
% By:       Norbert Koller
% Date:     Mai 2002
% Version:  1.0
```

[1] In alphabetischer Reihenfolge

```
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% Für den 1. Kreisfit werden alle Punkte verwendet !
Xmean = mean(Circle(1,:));
Ymean = mean(Circle(2,:));
% Nichtlinearer Kreisfit
AntwortKreis = FitArc([(Circle(1,:)-Xmean);(Circle(2,:)-Ymean)]);
%
KreisFit1.X0 = AntwortKreis(1) + Xmean;
KreisFit1.Y0 = AntwortKreis(2) + Ymean;
KreisFit1.R = AntwortKreis(3);
%
% Korrektur der Datenpunkte für den Kreisfit
%
% Errechnen einer stetigen Funktion durch die gefundenen Datenpunkte
% ————————— mit Splines ————————————————
%
Sp=spap2(20,4,Circle(1,:),Circle(2,:));
% 15 Knotenpunkte, 3 Ordnung
% Sp ist nun die Funktion Y=f(X)
%
% Errechnen der Datenpunkte (X,Y) für Y=f(X)
X = min(Circle(1,:)):0.2:max(Circle(1,:));
Y = fnval(Sp,X);
%
% Errechnen der Knotenpunkte der Splinefunktion
for t = 1 : Sp.number
   KnotenX(t) = min(Circle(1,:)) + (t-1)*(max(Circle(1,:))-min(Circle(1,:)))/(Sp.number-1);
   KnotenY(t) = Sp.coefs(t);
end
%
% Errechnen der 1.Ableitung der Splinefunktion
dSp=fnder(Sp);
%
dY = fnval(dSp,X);
% dY - Werte
%
% Fitten einer Linie durch die 1. Ableitung
%AntwortLinie = fitLine(X,dY);
%dYLine = X.*AntwortLinie.K + AntwortLinie.D1;
%
% Errechnen der 2.Ableitung der Splinefunktion ddSp=fnder(dSp);
%
ddY = fnval(ddSp,X);
% d^2Y - Werte
%
% Zeichnen der gemessenen Punkte und der Splinefunktion
%
fig = figure;
subplot(2,1,1);
hold on;
% Festlegen der plot-Abmessungen
Xmin = min(X) - abs((max(X)-min(X))/10);
Xmax = max(X) + abs((max(X)-min(X))/10);
Ymin = min(Y) - abs((max(Y)-min(Y))/10);
Ymax = max(Y) + abs((max(Y)-min(Y))/10);
axis([Xmin Xmax Ymin Ymax]);
%axis equal;
title('Darstellung der gemessenen Punkte');
ylabel('Welt Z-Achse [mm]');
s1=plot(Circle(1,:),Circle(2,:),'+red');
s2=plot(X,Y,'-blue');
s3=plot(KnotenX,KnotenY,'ocyan');
grid on;
legend_handles = [s1;s2;s3];
legend(legend_handles,'Gemessene Punkte','Gefittete Splines','Knotenpunkte');
```

```
%
% Zeichnen der Ableitungen
subplot(2,1,2);
hold on;
Ymin = min(min(dY),min(ddY)) - abs((max(max(dY),max(ddY))-min(min(dY),min(dY)))/10);
Ymax = max(max(dY),max(ddY)) + abs((max(max(dY),max(ddY))-min(min(dY),min(dY)))/10);
axis([Xmin Xmax Ymin Ymax]);
title('Darstellung der 1. und 2. Ableitung der Splinefunktion');
xlabel('Welt X-Achse [mm]');
ylabel('1. & 2. Ableitung [mm/mm],[mm/mm^2]'); s1=plot(X,dY,'-green');
s2=plot(X,ddY,'-black');
grid on;
legend_handles = [s1;s2];
legend(legend_handles,'1.Ableitung','2.Ableitung');
%
% ************************************************
% *** 1.Auswahlkriterium 2.Ableitung muss < 0 sein ***
% ************************************************
%
% Funktion eines idealen Kreises
% y = (r^2 - x^2)^0.5
% 1.Ableitung
% y' = -x/(r^2 - x^2)^0.5
% 2.Ableitung
% y'' = -r^2/(r^2 - x^2)^(3/2)
% ==> 2.Ableitung muss negativ sein !!!
%
ddYideal = -(KreisFit1.R^2)./((KreisFit1.R^2-(X-mean(X)).^2).^(3/2));
% 2. Ableitung der idealen Kreisgleichung (nur wenn X0,Y0 = Ursprung!!)
%
% Minimaler und Maximaler Wert der 2. Ableitung
MinddY = min(ddYideal) - abs(max(ddYideal)-min(ddYideal))/10;
MaxddY = max(ddYideal) + abs(max(ddYideal)-min(ddYideal))/10;
%
% Festlegen der Intervallgrenzen
%
for i = 1 : length(ddY);
   %
   TopBorder(i) = ddYideal(i) + abs(MaxddY+MinddY)/2;
   if TopBorder(i) > -0.005
     % Es handelt sich um eine Abflachung, Limit muss drunter liegen!!
     TopBorder(i) = -0.005;
   end
   %
   BottomBorder(i) = ddYideal(i) - abs(MaxddY+MinddY)/2;
end
%
Ausfall1(:,1) = [-1000;-1000];
%
j = 1;
k = 1;
for i = 1 : length(ddY)
   if ddY(i) >= BottomBorder(i) & ddY(i) <= TopBorder(i)
     % Der Punkt ist OK!
     XKorr1(j) = X(i);
     YKorr1(j) = Y(i);
     dYKorr1(j) = dY(i);
     ddYKorr1(j) = ddY(i);
     j = j + 1;
   else
     % Der Punkt ist nicht OK!
     Ausfall1(:,k)=[X(1,i);Y(1,i)];
     k = k + 1;
   end
end
%
% Zeichnen der Datenpunkteauswahl
%
```

```
fig=figure;
subplot(2,1,1);
hold on;
Ymin = min(Y) - abs((max(Y)-min(Y))/10);
Ymax = max(Y) + abs((max(Y)-min(Y))/10);
axis([Xmin Xmax Ymin Ymax]);
axis equal;
title('2.Kreisfit - Auswahl der Punkte');
ylabel('Welt Z-Achse [mm]');
s1=plot(XKorr1,YKorr1,'+blue');
s2=plot(Ausfall1(1,:),Ausfall1(2,:),'+red');
grid on;
legend_handles = [s1;s2];
legend(legend_handles,'Verwendete Datenpunkte','Ausgeschiedene Datenpunkte');
%
subplot(2,1,2);
hold on;
Ymin = min(min(BottomBorder),min(ddY)) - abs(max(ddY) - min(ddY))/10;
Ymax = max(max(TopBorder),max(ddY)) + abs(max(ddY) - min(ddY))/10;
axis([Xmin Xmax Ymin Ymax]);
title('1.Punktekorrektur');
xlabel('Welt X-Achse [mm]');
ylabel('2. Ableitung [mm/mm^2]');
s1=plot(X,ddY,'-black');
s2=plot(X,ddYideal,'-magenta');
s3=plot(X,TopBorder,'-red');
plot(X,BottomBorder,'-red');
%s2=line([Xmin Xmax],[MaxddY MaxddY],'color','red');
%s3=line([Xmin Xmax],[MinddY MinddY],'color','red');
grid on;
%
legend_handles = |s1;s2;s3];
legend(legend_handles,'2.Ableitung','2.Ableitung Ideal','Limit für 2.Ableitung');
%
% Fitten der korrigierten Datenpunkte
% 2. Kreisfit
%
Xmean = mean(XKorr1);
Ymean = mean(YKorr1);
%
% Nichtlinearer Kreisfit
AntwortKreis = FitArc([(XKorr1-Xmean);YKorr1-Ymean)]);
%
KreisFit2.X0 = AntwortKreis(1) + Xmean;
KreisFit2.Y0 = AntwortKreis(2) + Ymean;
KreisFit2.R = AntwortKreis(3);
%
% ****************************************
% *** 2.Auswahlkriterium Krümmungsradius ***
% ****************************************
%
% Errechnen der Krümmungsradien für alle Datenpunkte (X-Werte)
%
Kruemmung = abs(ddYKorr1)./(((1+(dYKorr1).^2).^.5).^3);
%
% Kruemmungsradius = f''(x) / ∛((1+(f'(x)^0.5)^2))
%
% Krümmungsradius Kreis
% Der Krümmungsradius eines Kreises ist Kr = 1/R und ist über den gesamten Umfang konstant!
KruemmungKreis = 1/KreisFit2.R;
% Der Radius des mit allen Datenpunkten gefitteten Kreis ist zwar verfälscht
% der Wert R ändert sich aber nur wenig!
Toleranz = 0.5;
% Entspricht der maximal zulässigen Abweichung vom Krümmungsradius
Ausfall2(:,1)=[-1000;-1000];
k = 1;
j = 1;
```

```
%
for i = 1 : length(XKorr1)
   %
   if abs(Kruemmung(1,i) - KruemmungKreis) < Toleranz*KruemmungKreis
     % Der Punkt ist OK
     XKorr2(k) = XKorr1(1,i);
     YKorr2(k) = YKorr1(1,i);
     dYKorr2(k) = dYKorr1(1,i);
     ddYKorr2(k) = ddYKorr1(1,i);
     k = k + 1;
   else
     %Der Punkt ist nicht OK
     Ausfall2(:,j)=[XKorr1(1,i);YKorr1(1,i)];
     j = j + 1;
   end;
end;
%
% Zeichnen der Datenpunkteauswahl
fig=figure;
%
subplot(2,1,1);
hold on;
Ymin = min(Y) - abs((max(Y)-min(Y))/10);
Ymax = max(Y) + abs((max(Y)-min(Y))/10);
axis([Xmin Xmax Ymin Ymax]);
title('3.Kreisfit - Auswahl der Punkte');
ylabel('Welt Z-Achse [mm]');
s1=plot(XKorr2,YKorr2,'+blue');
s2=plot(Ausfall1(1,:),Ausfall1(2,:),'+red');
s3=plot(Ausfall2(1,:),Ausfall2(2,:),'+green');
grid on;
legend_handles = [s1;s2;s3];
legend(legend_handles,'Verwendete Datenpunkte','Ausgeschiedene Datenpunkte 1','Ausgeschiedene Datenpunkte 2');
%
subplot(2,1,2);
Ymin = 10^(round(-0.5+log10(min(Kruemmung))));
Ymax = 10^(round(0.5+log10(max(Kruemmung))));
s1=semilogy(XKorr1,Kruemmung,'-black');
hold on;
s2=semilogy([Xmin+5 Xmax-5],[KruemmungKreis KruemmungKreis],'-blue');
s3=semilogy([Xmin Xmax],[(1+Toleranz)*KruemmungKreis (1+Toleranz)*KruemmungKreis],'-red');
s3=semilogy([Xmin Xmax],[(1-Toleranz)*KruemmungKreis (1-Toleranz)*KruemmungKreis],'-red');
grid on;
axis([Xmin Xmax Ymin Ymax]);
legend_handles = [s1;s2;s3];
legend(legend_handles,'Krümmungsradius der Messpunkte','Krümmungsradius Ideal','Limit');
%
%set (AX,'YMinorGrid','on');
%grid (AX);
title('Krümmungsradius');
xlabel('Welt X-Achse |mm|');
ylabel('log Krümmungsradius');
%
% Fitten der korrigierten Datenpunkte
% 2. Kreisfit
%
Xmean = mean(XKorr2);
Ymean = mean(YKorr2);
%
% Nichtlinearer Kreisfit
AntwortKreis = FitArc([(XKorr2-Xmean);(YKorr2-Ymean)]);
%
KreisFit3.X0 = AntwortKreis(1) + Xmean;
KreisFit3.Y0 = AntwortKreis(2) + Ymean;
KreisFit3.R = AntwortKreis(3);
%
% Zeichnen der gefitteten Kreise
%
```

```
fig = figure;
hold on;
axis equal;
grid on;
title('Darstellung der gefitteten Kreise');
xlabel('Welt X-Achse [mm]');
ylabel('Welt Z-Achse [mm]');
Xmin = (KreisFit1.X0 - KreisFit1.R) - (2*KreisFit1.R)/4;
Xmax = (KreisFit1.X0 + KreisFit1.R) + (2*KreisFit1.R)/4;
Ymin = (KreisFit1.Y0 - KreisFit1.R) - (2*KreisFit1.R)/10;
Ymax = (KreisFit1.Y0 + KreisFit1.R) + (2*KreisFit1.R)/10;
axis([Xmin Xmax Ymin Ymax]);
plot(Ausfall1(1,:),Ausfall1(2,:),'+red');
plot(Ausfall2(1,:),Ausfall2(2,:),'+red');
s1=plot(XKorr2,YKorr2,'+blue');
plotCircle( fig , KreisFit1 , 100, 'red' ); % Einzeichnen des Kreises1
plotCircle( fig , KreisFit2 , 100, 'red' ); % Einzeichnen des Kreises2
plotCircle( fig , KreisFit3 , 100, 'blue' );% Einzeichnen des Kreises3
%
%legend_handles = [s1;s2;s3];
%legend(legend_handles,'Messpunkte','KreisFit1','KreisFit2');
%
text(5,525,'Durchmesser Fit1 =','color','red','Units','pixels');
text(5,508,num2str(2*KreisFit1.R,4),'color','red','Units','pixels');
text(55,508,'mm','color','red','Units','pixels');
text(520,525,'Durchmesser Fit2 =','color','red','Units','pixels');
text(520,508,num2str(2*KreisFit2.R,4),'color','red','Units','pixels');
text(570,508,'mm','color','red','Units','pixels');
text(5,30,'Durchmesser Fit3 =','color','blue','FontSize',12,'Units','pixels');
text(5,10,num2str(2*KreisFit3.R,4),'color','blue','FontSize',12,'Units','pixels');
text(55,10,'mm','color','blue','FontSize',12,'Units','pixels');
disp('Der gemessene Durchmesser bertägt:');
D=2*KreisFit3.R
%
```

"Calibration.m"

```
%
function[Valid,Laser1T,Laser2T,ProjektionKameraLaser1,ProjektionKameraLaser2,Plane1,Plane2,Verkehrt]=Calibration(Pic0);
%
% Description:
%           This function calculates following transformation matrices:
%           Transformation matrix between World- and Pixelcoordinates (Object < - > Photo)
%           Transformation matrix between Pixel- and Lasercoordinates (Photo < - > Laser)
%
% Input Parameters:
%           A calibration image file.
%             In the file we have to set a lot of constants.
%             like: height, length & depth of the calibration object
%                   calibration parameters.
%
% Return Parameters:
%           Valid ... true if the calculation was successful.
%           LaserT ... 4*4 Transformation Matrix between the laser coordinates and world coordinates for both lasers.
%           ProjektionKameraLaser ... 3*3 Transformation Matrix between camera coordinates and laser coordinates for both lasers.
%           Plane ... cartesian plane coordinates for both laser planes.
%           Verkehrt ... true if the picture is set on the bottom.
%
% By:       Norbert Koller
% Date:     April - September 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% ***********************************
```

```
% *** Set all constants for calibration ***
% ***********************************
%
Schwellenwert1 = 0.25;          % 1.Schwellenwert zur Konturerkennung (Helligkeitsstufe)
Schwellenwert2 = 0.30;          % 2.Schwellenwert zur Konturerkennung
MinimalerRadius = 5;            % Minimaler Radius einer LED in Pixel (Radiusgrobabstimmung)
MaximalerRadius = 15;           % Maximaler Radius einer LED in Pixel (Radiusgrobabstimmung)
MinimalesXYVerhaeltnis = 0.7;   % Minimales Höhen/Breitenverhältnis zur Kreiserkennung
MaximalesXYVerhaeltnis = 1.4;   % Maximales Höhen/Breitenverhältnis zur Kreiserkennung
Toleranzfaktor = 1.8;           % Maximal tolerierbarer Fehlerabweichungsfaktor (Kreisfeinabstimmung)
Radiustoleranz = 0.3;           % 30% Radiustoleranz vom Mittelwert ist erlaubt! (Radiusfeinabstimmung)
MaxNeigung = 10;                % Maximal erlaubte Verdrehung des Kalibrationsobjekts (10^o)
Abweichung = 10;                % Maximale Abweichung in Pixel der LED's von der Normale
Zeilenabstand = 40;             % Maximale Abweichung eines Punktes von einer Zeile
RichtigFalsch = 30;             % Mehrabstand der LED's am Kalibrationsobjekt zur Richtungsbestimmung (PhysikY)
%
% Abstände der LED's am Kalibrationsobjekt in mm
PhysikX=[0 35 55 0 95 115 150;0 35 55 0 95 115 150;0 0 0 75 0 0 0;0 35 55 0 95 115 150;0 35 55 0 95 115 150];
% Physik Y wird erst später definiert, da es möglich ist dass das Kalibrationsobjekt um 180^o verdreht ist
PhysikZ=[0 20 30 0 30 20 0;0 20 30 0 30 20 0;0 0 0 30 0 0 0;0 20 30 0 30 20 0;0 20 30 0 30 20 0];
%
PhysikAbstufung = [-24 22.5 50 100 127.5 174];
% d.h.: Abstände vom Ursprung der einzelnen Ebenen in mm
% Vordere Ebene beginnt bei 50mm und endet bei 100mm
% Mittlere Ebene 22.5 - 50 mm und 100 - 127.5 mm
% Hintere Ebene - 24 - 22.5 mm und 127.5 - 174mm
%
Valid = 0; % 1 bei erfolgreicher Kalibration
%
% ********************************
% *** Konturerfassung der LED's ***
% ********************************
fig1=figure(1);
%
hold on; %Bild bearbeiten
title('Erfassen der Leuchtdioden');
xlabel('Pixel X-Achse');
ylabel('Pixel Y-Achse');
%
% Erzeugen einer Kontur um alle Objekte mit den Schwellenwerten
LED1=contour(Pic0,Schwellenwert1,'g'); % green
LED2=contour(Pic0,Schwellenwert2,'b'); % blue
%
HH=findobj('color','g'); % Finden der Konturen
GG=findobj('color','b');
%
n1=length(HH); % Anzahl der gefundenen Objekte
n2=length(GG);
%
% Fitten der Kreise mit dem 1.Schwellenwert
Gueltig =0;
Toleranz=0;
MittlererRadius=0;
%
for i=1:n1
   Nummer=HH(i,1);
   a=get(Nummer);
   XPosition=getfield(a,'XData'); % X und Y Position der Objekte
   YPosition=getfield(a,'YData');
   if (max(YPosition)-min(YPosition)) > 0
     Verhaeltnis=(max(XPosition)-min(XPosition))/(max(YPosition)-min(YPosition));
     %
     if Verhaeltnis > MinimalesXYVerhaeltnis & Verhaeltnis < MaximalesXYVerhaeltnis % Bedingung ob Kreis möglich
       XDataMittelwert=mean(XPosition);
       YDataMittelwert=mean(YPosition);
       Antwort=FitCircle(XPosition-XDataMittelwert,YPosition-YDataMittelwert); % Objekt wird gefittet
       Antwort.X0=Antwort.X0+XDataMittelwert;
       Antwort.Y0=Antwort.Y0+YDataMittelwert;
```

```
      if Antwort.R > MinimalerRadius & Antwort.R < MaximalerRadius % Nur wenn Kreis groß genug erfolgt ein Eintrag
        Gueltig = Gueltig + 1; % 1 Abfrage ob Punkt eine LED sein kann
        Kreisdaten1(Gueltig,:) = Antwort;
        Toleranz=Toleranz+Antwort.Residual;
        MittlererRadius=MittlererRadius+Antwort.R;
      end
    end
  end
end
%
Toleranz=Toleranz/Gueltig; % Mittelwert aller Fehler
MittlererRadius=MittlererRadius/Gueltig; % Mittelwert aller Radien
%
% Prüfen ob der Fehler des gefitteten Objekts akzeptabel ist
Zaehler1=0;
for i = 1 : Gueltig
  %
  if Kreisdaten1(i,1).Residual < Toleranzfaktor * Toleranz &...
      abs(Kreisdaten1(i,1).R-MittlererRadius) < Radiustoleranz*MittlererRadius
    % Nur wenn Fehler akzeptabel ist erfolgt ein Eintrag & Feinabfrage der Radien
    plotCircle( fig1 , Kreisdaten1(i,1) , 100, 'r' ); % Einzeichnen des Kreises
    Zaehler1=Zaehler1+1;
    KreisZentrum_x1(:,Zaehler1)=Kreisdaten1(i,1).X0;
    KreisZentrum_y1(:,Zaehler1)=Kreisdaten1(i,1).Y0;
    Kreisradius1(:,Zaehler1)=Kreisdaten1(i,1).R;
    Error1(:,Zaehler1)=Kreisdaten1(i,1).Residual;
    KreisMittelpunkte_xy1=[KreisZentrum_x1;KreisZentrum_y1];
    % Die X0 und Y0-Werte der Kreiszentren werden in ein "double-array" (Kreismittelpunkte_xy) verpackt.
  end
end
%
% Fitten der Kreise mit dem 2.Schwellenwert
Gueltig =0;
Toleranz=0;
MittlererRadius=0;
%
for i=1:n2
  Nummer=GG(i,1);
  a=get(Nummer);
  XPosition=getfield(a,'XData');
  YPosition=getfield(a,'YData');
  if (max(YPosition)-min(YPosition)) > 0
    Verhaeltnis=(max(XPosition)-min(XPosition))/(max(YPosition)-min(YPosition));
    %
    if Verhaeltnis > MinimalesXYVerhaeltnis & Verhaeltnis < MaximalesXYVerhaeltnis % Bedingung ob Kreis möglich
      XDataMittelwert=mean(XPosition);
      YDataMittelwert=mean(YPosition);
      Antwort=FitCircle(XPosition-XDataMittelwert,YPosition-YDataMittelwert); % Objekt wird gefittet
      Antwort.X0=Antwort.X0+XDataMittelwert;
      Antwort.Y0=Antwort.Y0+YDataMittelwert;
      if Antwort.R > MinimalerRadius & Antwort.R < MaximalerRadius % Nur wenn Kreis groß genug erfolgt ein Eintrag
        Gueltig=Gueltig+1; % Grobe Abfrage ob Punkt LED sein kann!
        Kreisdaten2(Gueltig,:)=Antwort;
        Toleranz=Toleranz+Antwort.Residual;
        MittlererRadius=MittlererRadius+Antwort.R;
      end
    end
  end
end
%
Toleranz=Toleranz/Gueltig; % Mittelwert aller Fehler
MittlererRadius=MittlererRadius/Gueltig; % Mittelwert aller Radien
%
Zaehler2=0;
for i = 1 : Gueltig
  %
  if Kreisdaten2(i,1).Residual < Toleranzfaktor * Toleranz &...
      abs(Kreisdaten2(i,1).R-MittlererRadius) < Radiustoleranz*MittlererRadius
```

```
      % Nur wenn Fehler akzeptabel ist erfolgt ein Eintrag
      plotCircle( fig1 , Kreisdaten2(i,1) , 100, 'm' );
      Zaehler2=Zaehler2+1;
      KreisZentrum_x2(:,Zaehler2)=Kreisdaten2(i,1).X0;
      KreisZentrum_y2(:,Zaehler2)=Kreisdaten2(i,1).Y0;
      Kreisradius2(:,Zaehler2)=Kreisdaten2(i,1).R;
      Error2(:,Zaehler2)=Kreisdaten2(i,1).Residual;
      KreisMittelpunkte_xy2=[KreisZentrum_x2;KreisZentrum_y2];
    end
end
%
% Bilden der Mittelwerte der Mittelpunkte für jeden Kreis
if Zaehler1 > Zaehler2
    Mittelpunkte1 = KreisMittelpunkte_xy1;
    Mittelpunkte2 = KreisMittelpunkte_xy2;
    Radius1=Kreisradius1;
    Radius2=Kreisradius2;
else
    Mittelpunkte1 = KreisMittelpunkte_xy2;
    Mittelpunkte2 = KreisMittelpunkte_xy1;
    Radius1=Kreisradius2;
    Radius2=Kreisradius1;
end
%
Zaehler1 = length(Mittelpunkte1);
Zaehler2 = length(Mittelpunkte2);
%
% Zusammenfügen der LED's bzw. Mittelpunkte
%
j(1:Zaehler2) = 0;
while sum(j) < Zaehler2
    for i = 1 : Zaehler1
      [Fehler,SooG]=min(abs(Mittelpunkte1(1,i)-Mittelpunkte2(1,:))+abs(Mittelpunkte1(2,i)-Mittelpunkte2(2,:)));
      if Fehler < (Radius1(1,i)/2)
        % Für die gefundene LED existieren zwei Kreise (Schwellenwert1 und 2)
        % Berechnung des Mittelwerts für den Mittelpunkt.
        Punkt_x(1,i) = (Mittelpunkte1(1,i)+Mittelpunkte2(1,SooG))/2;
        Punkt_y(1,i) = (Mittelpunkte1(2,i)+Mittelpunkte2(2,SooG))/2;
        j(SooG) = 1;
      else
        % Für die gefundene LED existiert nur ein Kreis (Schwellenwert 1 oder 2)
        % Mittelpunkt bleibt gleich
        Punkt_x(1,i) = Mittelpunkte1(1,i);
        Punkt_y(1,i) = Mittelpunkte1(2,i);
      end
    end
    %
    [Wert,SooG] = min(j);
    if Wert == 0
      i = i + 1;
      Punkt_x(1,i) = Mittelpunkte2(1,SooG);
      Punkt_y(1,i) = Mittelpunkte2(2,SooG);
      j(SooG) = 1;
    end
end
%
Punkteanzahl = length(Punkt_x);
%
Punkt_xy = [Punkt_x(1,:);Punkt_y(1,:)];
%
% Sortierung der LED's in einer Matrix!
%
% Sortieren der Punkte nach x-Richtung
Punkt_Sx = Bubblesort(Punkt_xy,1); % Sortiert die Matrix der Größe nach der 1 Zeile (x-Richtung)
%
% Einteilung der Punkte in die jeweiligen Spalten
Spalte=1;
Zeile=1;
```

```
LEDM1X(1,1)=Punkt_Sx(1,1);
LEDM1Y(1,1)=Punkt_Sx(2,1);
Line.XData = Punkt_Sx(2,1:2);
Line.YData = Punkt_Sx(1,1:2);
i = 2;
while i <= Punkteanzahl
   %
   Antwort = FitLine(Line.XData,Line.YData);
   % Berechnet eine Gerade mit den ersten beiden Punkten in einer Spalte
   Abstand = abs(Punkt_Sx(1,i) - (Antwort.D1+Punkt_Sx(2,i)*Antwort.K));
   %
   if abs(180*(atan(Antwort.K))/pi) < MaxNeigung & Abstand < Abweichung
     % Wenn die Neigung der Geraden geringer als +- 10^o ist so liegt die LED in der gleichen Spalte
     % Gleichzeitig darf der Abstand der LED zur Geraden weniger als 10 Pixel betragen
     Zeile = Zeile + 1;
   else
     Zeile = 1;
     Spalte = Spalte + 1;
     if i < Punkteanzahl
       Line.XData = Punkt_Sx(2,i:i+1);
       Line.YData = Punkt_Sx(1,i:i+1);
     end
   end
   %
   LEDM1X(Zeile,Spalte)=Punkt_Sx(1,i);
   LEDM1Y(Zeile,Spalte)=Punkt_Sx(2,i);
   i = i + 1;
   %
end
%
Mitte = round(Spalte/2);
%
if Mitte < 3 | Mitte > 4 | abs(sum(LEDM1X(:,Mitte)) - LEDM1X(1,Mitte)) > 0 | Spalte == 6
   if Spalte == 6
     disp('Das Kalibrationsbild ist nicht symmetrisch,');
     disp('oder die mittlere LED wurde nicht gefunden!');
   else
     disp('Es wurden falsche Punkte gefunden!');
   end
   disp('Kalibration ist nicht möglich!');
   disp('Wenn im Kalibrationsbild die LED's gefunden wurden,');
   disp('so muss der Variable MaxNeigung ein anderer Wert zugewiesen werden!');
   break
end
%
% Einrichten der Matrix wenn nur die mittlere und die vordere Ebene gefunden wurden
%
if Mitte == 3
   LEDM3X(1:5,1:7) = 0;
   LEDM3Y(1:5,1:7) = 0;
   for i = 2 : 6
     LEDM3X(:,i) = LEDM1X(:,i-1);
     LEDM3Y(:,i) = LEDM1Y(:,i-1);
   end
   LEDM1X = LEDM3X;
   LEDM1Y = LEDM3Y;
end
%
% Sortieren der LED's in Y-Richtung
%
for i = 1 : 7
   SortiertePunkte = Bubblesort([LEDM1X(:,i)';LEDM1Y(:,i)'],2);
   LEDM1X(:,i) = SortiertePunkte(1,:)';
   LEDM1Y(:,i) = SortiertePunkte(2,:)';
end
%
% Berechnen der Y-Verschiebung der einzelnen Ebenen
%
```

```
[Dist1, Pos1] = min(abs(max(LEDM1Y(:,1))-LEDM1Y(:,2)));
[Dist2, Pos2] = min(abs(max(LEDM1Y(:,7))-LEDM1Y(:,6)));
[Dist3, Pos3] = min([Dist1 Dist2]);
%
if Pos3 == 1
   Verschiebung12 = max(LEDM1Y(:,1)) - LEDM1Y(Pos1,2);
else
   Verschiebung12 = max(LEDM1Y(:,7)) - LEDM1Y(Pos2,6);
end
%
% Verschiebung zwischen hinterer und mittlerer Ebene
%
% Ist der Wert in Verschiebung12 negativ so wurde das Kalibrationsbild verkehrt aufgenommen
% => Die Kamera steht Kopf !!!
%
Verschiebung23 = min(min(abs(max(LEDM1Y(:,2))-LEDM1Y(:,3))),min(abs(max(LEDM1Y(:,6))-LEDM1Y(:,5))));
%
if Verschiebung12 < 0
   Verschiebung23 = -Verschiebung23;
   Verkehrt = 1;
else
   Verkehrt = 0;
end
% Verschiebung zwischen mittlerer und vorderer Ebene
%
% Generieren einer um die Verschiebung korrigierten Hilfsmatrix
%
Hilfsmatrix = LEDM1Y;
Hilfsmatrix(:,2) = LEDM1Y(:,2) + Verschiebung12;
Hilfsmatrix(:,3) = LEDM1Y(:,3) + (Verschiebung12 + Verschiebung23);
Hilfsmatrix(:,4) = LEDM1Y(:,4) + (Verschiebung12 + Verschiebung23);
Hilfsmatrix(:,5) = LEDM1Y(:,5) + (Verschiebung12 + Verschiebung23);
Hilfsmatrix(:,6) = LEDM1Y(:,6) + Verschiebung12;
%
% Sortieren der y-Koordinaten der Größe nach!
k = 1;
for i = 1 : size(Hilfsmatrix,1)
   for j = 1 : 7
     if ((Hilfsmatrix(i,j) > abs(Verschiebung12 + Verschiebung23)) & (Verschiebung12 > 0)) |...
        ((Hilfsmatrix(i,j) > 0) & (Verschiebung12 < 0))
       % Nur wenn der Eintrag größer als die Verschiebung ist handelt es sich um einen echten Wert!
       % wenn das Bild verkehrt liegt ist der Eintrag in Verschiebung12 negativ
       % ==> Fehlt eine LED so steht an deren Stelle ein negativer Wert !!!
       % (Änderung am 10.09 bei EWK !!!!!)
       ListeKorr(1,k) = LEDM1X(i,j);
       ListeKorr(2,k) = Hilfsmatrix(i,j);
       k = k + 1;
     end
   end
end
%
Punkt_Sy = Bubblesort(ListeKorr,2);
% Sortiert die Matrix der Größe nach der 2 Zeile (y-Richtung)
%
% Einteilung der Punkte in die jeweiligen Zeilen
%
PunkteproZeile = 1;
Zeileneu = 1;
%
for i = 1 : Punkteanzahl
   Zeile = 1;
   % Suche nach dem Punkt in der Matrix
   % Es wird nach der X-Koordinate gesucht, da durch die Verschiebung
   % einige y-Koordinaten ident sind. Es ist sehr unwahrscheinlich das
   % 2 x-Koordinaten ident sind !!!
   [Wert,Pointer] = min(abs(Punkt_Sy(1,i) - LEDM1X(Zeile,:)));
   while (abs(Wert) > 0) & (abs(Punkt_Sy(1,i) - LEDM1X(Zeile,Pointer)) > 0)
     Zeile = Zeile + 1;
```

```
    [Wert,Pointer] = min(abs(Punkt_Sy(1,i) - LEDM1X(Zeile,:)));
  end
  %
  % Der Punkt wurde in der Spalte == Pointer gefunden
  %
  % Der erste Punkt (kleinster y-Wert) muss in der ersten Zeile liegen
  % -> Mittelwert = erster Punkt !
  if i == 1
    Mittel = Hilfsmatrix(Zeile,Pointer);
  end
  %
  if Pointer == Mitte
    % Der mittlere Punkt wird in eine extra Zeile geschrieben !
    Zeileneu = Zeileneu + 1;
    PunkteproZeile = 1;
    Mittel = -1000;
    % Der Mittelwert wird auf einen unmöglichen Wert gesetzt!
    % damit bei der nächsten LED eine neue Zeile begonnen wird!
  elseif abs(Hilfsmatrix(Zeile,Pointer) - Mittel) < Zeilenabstand
    % Zuweisung der Punkte zu den einzelnen Zeilen
    if PunkteproZeile == 1
      Mittel = Hilfsmatrix(Zeile,Pointer);
      % bei der ersten LED in jeder neuen Zeile wird der Mittelwert auf den korrigierten y-Wert
      % dieser LED gesetzt.
    else
      Mittel = (PunkteproZeile - 1)*Mittel/PunkteproZeile + Hilfsmatrix(Zeile,Pointer)/PunkteproZeile;
      % bei jeder weiteren LED wird der Mittelwert aus allen in der Zeile befindlichen LED's errechnet
      % Mittelwert(n) = (n-1)*Mittelwert(n-1)/n + Wert(n)/n
    end
    PunkteproZeile=PunkteproZeile+1;
  else
    % es wird eine neue Zeile begonnen!
    Zeileneu = Zeileneu + 1;
    Mittel = Hilfsmatrix(Zeile,Pointer);
    PunkteproZeile = 1;
  end
  % Zuweisung der LED's in die Matrix LEDM2
  LEDM2X(Zeileneu,Pointer)=LEDM1X(Zeile,Pointer);
  LEDM2Y(Zeileneu,Pointer)=LEDM1Y(Zeile,Pointer);
end
%
% Überprüfung ob Kalibration möglich ist!
Zeile = Zeileneu;
%
if Zeile < 4 | Zeile > 5
  if Zeile < 4
    disp('Es sind zuwenig Punkte vorhanden!');
  else
    disp('Es wurden falsche Punkte gefunden!');
  end
  disp('Kalibration ist nicht möglich!');
  break
end
%
% Erzeugen einer Prüfmatrix
for i = 1 : Spalte
  for j = 1 : Zeile
    if LEDM2X(j,i) > 0
      LEDMOK(j,i) = 1; % für jeden Eintrag(1) existiert eine LED
    end
  end
end
%
% Einrichten der Matrix
%
LEDMatrixX(1:5,1:7) = 0; % Alle Punkte sollen Null sein!
LEDMatrixY(1:5,1:7) = 0;
LEDMatrixOK(1:5,1:7) = 0;
```

```
%
[Wert,Pointer] = max(LEDM2X(:,Mitte));
% Pointer, in dieser Zeile befindet sich die mittlere LED
Zeilenanfang = 3 - Pointer;
% 3 Position der mittleren LED wenn alle LED's erfasst wurden!
%
for i = 1 : Zeile
   for j = 1 : Spalte
     LEDMatrixX(i+Zeilenanfang,j) = LEDM2X(i,j);
     LEDMatrixY(i+Zeilenanfang,j) = LEDM2Y(i,j);
     LEDMatrixOK(i+Zeilenanfang,j) = LEDMOK(i,j);
   end
end
%
% Überprüfung der Orientierung des Kalibrationsobjekts, ob Richtig oder 180 gedreht
%
Zeile(1:5) = 0;
for j = 1 : 5
   k = 0;
   for i = 1 : 7
     if LEDMatrixY(j,i) > 0
       Zeile(j) = Zeile(j) + LEDMatrixY(j,i);
       k = k + 1;
     end
   end
   if k > 0
     Zeile(j) = Zeile(j)/k;
   else
     Zeile(j) = 0;
   end
end
%
Abstand12 = abs(Zeile(2)-Zeile(1));
Abstand24 = abs(Zeile(4)-Zeile(2));
Abstand45 = abs(Zeile(5)-Zeile(4));
%
if (((mean(LEDMatrixY(1,:))) > 0) & (Abstand12 > Abstand24 + RichtigFalsch)) | (((mean(LEDMatrixY(5,:))) > 0) &...
     (Abstand24 + RichtigFalsch > Abstand45))
   % Das Kalibrationsobjekt liegt verkehrt!
   PhysikY=[0 0 0 0 0 0 0;60 60 60 0 60 60 60;0 0 0 80 0 0 0;100 100 100 0 100 100 100;140 140 140 0 140 140 140];
else
     PhysikY=[0 0 0 0 0 0 0;40 40 40 0 40 40 40;0 0 0 60 0 0 0;80 80 80 0 80 80 80;140 140 140 0 140 140 140];
end
%
% Auswahl der LED's welche für die Kalibration verwendet werden!
% und Überprüfung ob in den Spalten genügend Punkte vorhanden sind! (mind. 2)
%
PunkteVorne = 0;
PunkteHinten = 0;
%
Zaehler=1;
for i = 3 : 2 : 5
   for j = 1 : 5
     if LEDMatrixOK(j,i) == 1 & sum(LEDMatrixOK(:,i)) > 1
       PunkteVorne(1,Zaehler) = LEDMatrixX(j,i);
       PunkteVorne(2,Zaehler) = LEDMatrixY(j,i);
       PunkteVornePhysik(1,Zaehler) = PhysikX(j,i);
       PunkteVornePhysik(2,Zaehler) = PhysikY(j,i);
       Zaehler=Zaehler+1;
     end
   end
end
%
PunkteVorne(1,Zaehler) = LEDMatrixX(3,4); % Der mittlere Punkt wird auch verwendet
PunkteVorne(2,Zaehler) = LEDMatrixY(3,4);
PunkteVornePhysik(1,Zaehler) = PhysikX(3,4);
PunkteVornePhysik(2,Zaehler) = PhysikY(3,4);
%
```

```
Zaehler=1;
for i = 1 : 6 : 7
   for j = 1 : 5
     if LEDMatrixOK(j,i) == 1 & sum(LEDMatrixOK(:,i)) > 1
       PunkteHinten(1,Zaehler) = LEDMatrixX(j,i);
       PunkteHinten(2,Zaehler) = LEDMatrixY(j,i);
       PunkteHintenPhysik(1,Zaehler) = PhysikX(j,i);
       PunkteHintenPhysik(2,Zaehler) = PhysikY(j,i);
       Zaehler=Zaehler+1;
     end
   end
end
% In jeder Spalte müssen mindestens 2 LED's gefunden werden
if sum(LEDMatrixOK(:,1)) < 2 | sum(LEDMatrixOK(:,3)) < 2 | sum(LEDMatrixOK(:,5)) < 2 | sum(LEDMatrixOK(:,7)) < 2
   disp('Kalibration ist nicht möglich!');
   disp('Es sind zuwenig Punkte vorhanden!');
   break
end
%
% Beschriftung der LED's
%
for i = 1 : length(PunkteVorne)
   text(PunkteVorne(1,i)+15,PunkteVorne(2,i),'V','color','w');
   text(PunkteVorne(1,i)+40,PunkteVorne(2,i),num2str(i),'color','w');
end
%
for i = 1 : length(PunkteHinten)
   text(PunkteHinten(1,i)+15,PunkteHinten(2,i),'H','color','w');
   text(PunkteHinten(1,i)+40,PunkteHinten(2,i),num2str(i),'color','w');
end
%
% Errechnung der Transformationsmatrix
PunkteVornePixel = [PunkteVorne(1,:);PunkteVorne(2,:);ones(1,length(PunkteVorne))];
PunkteVorneWelt = [PunkteVornePhysik(1,:);PunkteVornePhysik(2,:);zeros(1,length(PunkteVornePhysik));...
    ones(1,length(PunkteVornePhysik))];
%
PunkteHintenPixel = [PunkteHinten(1,:);PunkteHinten(2,:);ones(1,length(PunkteHinten))];
PunkteHintenWelt = [PunkteHintenPhysik(1,:);PunkteHintenPhysik(2,:);zeros(1,length(PunkteHintenPhysik));...
    ones(1,length(PunkteHintenPhysik))];
%
% Generieren der Projektionsmatrizen für die hintere und die vordere Ebene
%
HHinten = GenProj(PunkteHintenWelt, PunkteHintenPixel);
HVorne = GenProj(PunkteVorneWelt, PunkteVornePixel);
%
% Einzeichnen des Weltkoordinatensystems
% Berechnung des Nullpunktes
Ursprung_nichtaffin = inv(HHinten)*[0 0 1]';
% Rücktransformation des Punktes (0,0) in mm
Ursprung_affin = [Ursprung_nichtaffin(1,1)/Ursprung_nichtaffin(3,1);Ursprung_nichtaffin(2,1)/Ursprung_nichtaffin(3,1);1];
% Dieser Punkt muss im Mittelpunkt der ersten LED in der hinteren Ebene liegen!
P1_NA = inv(HHinten)*[20 0 1]';
P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
P2_NA = inv(HHinten)*[0 20 1]';
P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
P3_NA = inv(HVorne)*[0 0 1]';
P3_A = [P3_NA(1,1)/P3_NA(3,1);P3_NA(2,1)/P3_NA(3,1);1];
line([Ursprung_affin(1,1) P1_A(1,1)],[Ursprung_affin(2,1) P1_A(2,1)],'color','m');
% Zeichnen des Koordinatensystems
line([Ursprung_affin(1,1) P2_A(1,1)],[Ursprung_affin(2,1) P2_A(2,1)],'color','m');
line([Ursprung_affin(1,1) P3_A(1,1)],[Ursprung_affin(2,1) P3_A(2,1)],'color','m');
text(Ursprung_affin(1,1)+30,Ursprung_affin(2,1)+40,'Welt-KS','color','m');
text(Ursprung_affin(1,1)+60,Ursprung_affin(2,1)-10,'X','color','m');
text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)+65,'Y','color','m');
if P3_A(2,1) < Ursprung_affin(2,1)
   text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)-30,'Z','color','m');
else
   text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)+15,'Z','color','m');
```

```
end
%
% Kontrolle ob Kalibration erfolgreich ?
% Für jeden Mittelpunkt werden aus den Weltkoordinaten die Pixelkoordinaten berechnet
for i = 1 : 2
   P_NA(:,i) = inv(HHinten) * [PhysikX(i,1) PhysikY(i,1)1]';
   P_A(:,i) = |P_NA(1,i)/P_NA(3,i);P_NA(2,i)/P_NA(3,i);1];
   P_NA(:,i+2) = inv(HHinten) * [PhysikX(i+3,1) PhysikY(i+3,1)1]';
   P_A(:,i+2) = [P_NA(1,i+2)/P_NA(3,i+2);P_NA(2,i+2)/P_NA(3,i+2);1];
   P_NA(:,i+4) = inv(HHinten) * [PhysikX(i,7) PhysikY(i,7)1]';
   P_A(:,i+4) = [P_NA(1,i+4)/P_NA(3,i+4);P_NA(2,i+4)/P_NA(3,i+4);1];
   P_NA(:,i+6) = inv(HHinten) * [PhysikX(i+3,7) PhysikY(i+3,7)1]';
   P_A(:,i+6) = [P_NA(1,i+6)/P_NA(3,i+6);P_NA(2,i+6)/P_NA(3,i+6);1];
   P_NA(:,i+8) = inv(HVorne) * [PhysikX(i,3) PhysikY(i,3)1]';
   P_A(:,i+8) = [P_NA(1,i+8)/P_NA(3,i+8);P_NA(2,i+8)/P_NA(3,i+8);1];
   P_NA(:,i+10) = inv(HVorne) * [PhysikX(i+3,3) PhysikY(i+3,3)1]';
   P_A(:,i+10) = [P_NA(1,i+10)/P_NA(3,i+10);P_NA(2,i+10)/P_NA(3,i+10);1];
   P_NA(:,i+12) = inv(HVorne) * [PhysikX(i,5) PhysikY(i,5)1]';
   P_A(:,i+12) = [P_NA(1,i+12)/P_NA(3,i+12);P_NA(2,i+12)/P_NA(3,i+12);1];
   P_NA(:,i+14) = inv(HVorne) * [PhysikX(i+3,5) PhysikY(i+3,5)1]';
   P_A(:,i+14) = [P_NA(1,i+14)/P_NA(3,i+14);P_NA(2,i+14)/P_NA(3,i+14);1];
end
P_NA(:,17) = inv(HVorne) * [PhysikX(3,4) PhysikY(3,4) 1]';
P_A(:,17) = [P_NA(1,17)/P_NA(3,17);P_NA(2,17)/P_NA(3,17);1];
%
plot(P_A(1,:),P_A(2,:),'*black');
%
% Die Abweichung zu den gefundenen Mittelpunkten wird berechnet
for i = 1 : 2
   Fehler(i) = abs(LEDMatrixX(i,1)-P_A(1,i)) + abs(LEDMatrixY(i,1)-P_A(2,i));
   Fehler(i+2) = abs(LEDMatrixX(i+3,1)-P_A(1,i+2)) + abs(LEDMatrixY(i+3,1)-P_A(2,i+2));
   Fehler(i+4) = abs(LEDMatrixX(i,7)-P_A(1,i+4)) + abs(LEDMatrixY(i,7)-P_A(2,i+4));
   Fehler(i+6) = abs(LEDMatrixX(i+3,7)-P_A(1,i+6)) + abs(LEDMatrixY(i+3,7)-P_A(2,i+6));
   Fehler(i+8) = abs(LEDMatrixX(i,3)-P_A(1,i+8)) + abs(LEDMatrixY(i,3)-P_A(2,i+8));
   Fehler(i+10) = abs(LEDMatrixX(i+3,3)-P_A(1,i+10)) + abs(LEDMatrixY(i+3,3)-P_A(2,i+10));
   Fehler(i+12) = abs(LEDMatrixX(i,5)-P_A(1,i+12)) + abs(LEDMatrixY(i,5)-P_A(2,i+12));
   Fehler(i+14) = abs(LEDMatrixX(i+3,5)-P_A(1,i+14)) + abs(LEDMatrixY(i+3,5)-P_A(2,i+14));
end
Fehler(17) = abs(LEDMatrixX(3,4)-P_A(1,17)) + abs(LEDMatrixY(3,4)-P_A(2,17));
%
% Wenn die Abweichung zu groß ist, wird eine Warnung ausgegeben
for i = 1 : 17
   if (Fehler(i) > 0.5) & (Fehler(i) < 5)
     text(220,40,'WARNING - bad calibration','color','r','FontSize',12);
     P.X0 = P_A(1,i);
     P.Y0 = P_A(2,i);
     P.R = 20;
     plotCircle( fig1 , P , 100, 'r' ); % Einzeichnen des Kreises
   end
end
%
% ***************************
% *** Finden der Laserlinien ***
% ***************************
%
% Löschen der LED's
%
RightMax = size(Pic0,2);
DownMax = size(Pic0,1);
%
RetuschierWert = 1 + max(max(Radius1),max(Radius2)) - 2;
for i = 1 : length(Punkt_Sx)
   % Ermittlung der Pixel welche gelöscht werden sollen
   Left = round(Punkt_Sx(1,i) - RetuschierWert);
   if Left < 1
     Left = 1;
   end
   Right = round(Punkt_Sx(1,i) + RetuschierWert);
```

```
    if Right > RightMax
      Right = RightMax;
    end
    Up = round(Punkt_Sx(2,i) - RetuschierWert);
    if Up < 1
      Up = 1;
    end
    Down = round(Punkt_Sx(2,i) + RetuschierWert);
    if Down > DownMax
      Down = DownMax;
    end
    % Löschen (schwarzsetzten) der Punkte
    Pic0(Up:Down,Left:Right) = 0;
end
%
% Finden der Laserlinien & Errechnung des 1. Intensitätsmoments
%
[LaserPoints1,Length1,LaserPoints2,Length2] = FindLaserLines(Pic0,Verkehrt);
%
% Zuordnung der Laserpunkte zu den einzelnen Ebenen
%
% Umrechnung der Weltkoordinaten (in mm) in Pixelkoordinaten
% in der Mitte des Kalibrationsobjekts
%
% Errechnen der Bildmitte
MittePixel = [round(size(Pic0,2)/2);round(size(Pic0,1)/2);1];
% Transformation des Punktes in mm - Koordinaten
MitteWelt_NA = HVorne*MittePixel;
MitteWelt_A = [MitteWelt_NA(1,1)/MitteWelt_NA(3,1);MitteWelt_NA(2,1)/MitteWelt_NA(3,1);1];
%
% Transformation der Ebenen-Koordinaten (Anfang und Ende der jeweiligen Ebenen)
%
% Hintere Ebene
%
LeftWelt1 = [PhysikAbstufung(1);MitteWelt_A(2,1);1];
RightWelt1 = [PhysikAbstufung(2);MitteWelt_A(2,1);1];
LeftWelt2 = [PhysikAbstufung(5); MitteWelt_A(2,1);1];
RightWelt2 = [PhysikAbstufung(6);MitteWelt_A(2,1);1];
%
% Rücktransformation der Punkte in das Pixelkoordinatensystem
%
HLeft1_NA = inv(HHinten)*LeftWelt1;
HLeft1_A = [HLeft1_NA(1,1)/HLeft1_NA(3,1);HLeft1_NA(2,1)/HLeft1_NA(3,1);1];
HRight1_NA = inv(HHinten)*RightWelt1;
HRight1_A = [HRight1_NA(1,1)/HRight1_NA(3,1);HRight1_NA(2,1)/HRight1_NA(3,1);1];
HLeft2_NA = inv(HHinten)*LeftWelt2;
HLeft2_A = [HLeft2_NA(1,1)/HLeft2_NA(3,1);HLeft2_NA(2,1)/HLeft2_NA(3,1);1];
HRight2_NA = inv(HHinten)*RightWelt2;
HRight2_A = [HRight2_NA(1,1)/HRight2_NA(3,1);HRight2_NA(2,1)/HRight2_NA(3,1);1];
%
% Vordere Ebene
%
LeftWelt1 = [PhysikAbstufung(3);MitteWelt_A(2,1);1];
RightWelt1 = [PhysikAbstufung(4);MitteWelt_A(2,1);1];
%
VLeft_NA = inv(HVorne)*LeftWelt1;
VLeft_A = [VLeft_NA(1,1)/VLeft_NA(3,1);VLeft_NA(2,1)/VLeft_NA(3,1);1];
VRight_NA = inv(HVorne)*RightWelt1;
VRight_A = [VRight_NA(1,1)/VRight_NA(3,1);VRight_NA(2,1)/VRight_NA(3,1);1];
%
% Zuordnen der Laserpunkte Laserlinie1 zu den einzelnen Ebenen
%
k = 1;
l = 1;
m = 1;
Zaehler = 0;
for i = 1 : length(Length1)
   Zaehler = Zaehler + Length1(i);
```

```
    Points = LaserPoints1(:,(1 + Zaehler - Length1(i)) : Zaehler);
    SooG = mean(Points(1,:));
    %
    if SooG > HLeft1_A(1,1) & SooG < HRight1_A(1,1)
      % Das Liniensegment befindet sich auf der hinteren Ebene links!
      if k == 1
        Laser1HintenLinks = Points;
        k = k + 1;
      else
        Laser1HintenLinks = [Laser1HintenLinks Points];
      end
      %
    elseif SooG > VLeft_A(1,1) & SooG < VRight_A(1,1)
      % Das Liniensegment befindet sich auf der vorderen Ebene!
      if l == 1
        Laser1Vorne = Points;
        l = l + 1;
      else
        Laser1Vorne = [Laser1Vorne Points];
      end
      %
    elseif SooG > HLeft2_A(1,1) & SooG < HRight2_A(1,1)
      % Das Liniensegment befindet sich auf der hinteren Ebene rechts!
      if m == 1
        Laser1HintenRechts = Points;
        m = m + 1;
      else
        Laser1HintenRechts = [Laser1HintenRechts Points];
      end
      %
    end
  end
  %
  % Zuordnen der Laserpunkte Laserlinie2
  %
  if Length2 > 2
    % es wurde eine 2.Laserlinie gefunden
    k = 1;
    l = 1;
    m = 1;
    Zaehler = 0;
    for i = 1 : length(Length2)
      Zaehler = Zaehler + Length2(i);
      Points = LaserPoints2(:,(1 + Zaehler - Length2(i)) : Zaehler);
      SooG = mean(Points(1,:));
      %
        if SooG > HLeft1_A(1,1) & SooG < HRight1_A(1,1)
        % Das Liniensegment befindet sich auf der hinteren Ebene links!
        if k == 1
          Laser2HintenLinks = Points;
          k = k + 1;
        else
          Laser2HintenLinks = [Laser2HintenLinks Points];
        end
        %
      elseif SooG > VLeft_A(1,1) & SooG < VRight_A(1,1)
        % Das Liniensegment befindet sich auf der vorderen Ebene!
        if l == 1
          Laser2Vorne = Points;
          l = l + 1;
        else
          Laser2Vorne = [Laser2Vorne Points];
        end
        %
      elseif SooG > HLeft2_A(1,1) & SooG < HRight2_A(1,1)
        % Das Liniensegment befindet sich auf der hinteren Ebene rechts!
        if m == 1
          Laser2HintenRechts = Points;
```

```
        m = m + 1;
      else
        Laser2HintenRechts = [Laser2HintenRechts Points];
      end
      %
    end
  end
end
%
% Umrechnung der Laserpunkte ins Welt Koordinatensystem
%
% Erzeugung von affinen Koordinaten
Laser1Vorne_A = [Laser1Vorne(1,:);Laser1Vorne(2,:);ones(1,size(Laser1Vorne,2))];
Laser1HintenLinks_A = [Laser1HintenLinks(1,:);Laser1HintenLinks(2,:);ones(1,size(Laser1HintenLinks,2))];
Laser1HintenRechts_A =[Laser1HintenRechts(1,:);Laser1HintenRechts(2,:);ones(1,size(Laser1HintenRechts,2))];
%
% Zuweisen der Höhe zu den Ebenen
HoeheVorne = PhysikZ(1,3);
HoeheHinten = PhysikZ(1,1);
%
% Umrechnung
Laser1WeltVorne_NA = HVorne * Laser1Vorne_A;
Laser1WeltVorne_A = [Laser1WeltVorne_NA(1,:)./Laser1WeltVorne_NA(3,:);Laser1WeltVorne_NA(2,:)./Laser1WeltVorne_NA(3,:);...
    HoeheVorne(ones(1,length(Laser1WeltVorne_NA)))];
Laser1WeltHintenLinks_NA = HHinten * Laser1HintenLinks_A;
Laser1WeltHintenLinks_A = [Laser1WeltHintenLinks_NA(1,:)./Laser1WeltHintenLinks_NA(3,:);...
    Laser1WeltHintenLinks_NA(2,:)./Laser1WeltHintenLinks_NA(3,:);HoeheHinten(ones(1,length(Laser1WeltHintenLinks_NA)))];
Laser1WeltHintenRechts_NA = HHinten * Laser1HintenRechts_A;
Laser1WeltHintenRechts_A =[Laser1WeltHintenRechts_NA(1,:)./Laser1WeltHintenRechts_NA(3,:);...
    Laser1WeltHintenRechts_NA(2,:)./Laser1WeltHintenRechts_NA(3,:);HoeheHinten(ones(1,length(Laser1WeltHintenRechts_NA)))];
%
if Length2 > 2
  % es wurde eine 2.Laserlinie gefunden
  Laser2Vorne_A =[Laser2Vorne(1,:);Laser2Vorne(2,:);ones(1,size(Laser2Vorne,2))];
  Laser2HintenLinks_A =[Laser2HintenLinks(1,:);Laser2HintenLinks(2,:);ones(1,size(Laser2HintenLinks,2))];
  Laser2HintenRechts_A = [Laser2HintenRechts(1,:);Laser2HintenRechts(2,:);ones(1,size(Laser2HintenRechts,2))];
  %
  % Umrechnung
  Laser2WeltVorne_NA = HVorne * Laser2Vorne_A;
  Laser2WeltVorne_A =[Laser2WeltVorne_NA(1,:)./Laser2WeltVorne_NA(3,:);...
      Laser2WeltVorne_NA(2,:)./Laser2WeltVorne_NA(3,:);HoeheVorne(ones(1,length(Laser2WeltVorne_NA)))];
  Laser2WeltHintenLinks_NA = HHinten * Laser2HintenLinks_A;
  Laser2WeltHintenLinks_A = [Laser2WeltHintenLinks_NA(1,:)./Laser2WeltHintenLinks_NA(3,:);...
      Laser2WeltHintenLinks_NA(2,:)./Laser2WeltHintenLinks_NA(3,:);HoeheHinten(ones(1,length(Laser2WeltHintenLinks_NA)))];
  Laser2WeltHintenRechts_NA = HHinten * Laser2HintenRechts_A;
  Laser2WeltHintenRechts_A =[Laser2WeltHintenRechts_NA(1,:)./Laser2WeltHintenRechts_NA(3,:);...
      Laser2WeltHintenRechts_NA(2,:)./Laser2WeltHintenRechts_NA(3,:);HoeheHinten(ones(1,length(Laser2WeltHintenRechts_NA)))];
  %
end
%
% ********************
% *** Laserebenenfit ***
% ********************
%
[Laser1T,ErrorproPunkt1,Plane1] = FitLaserPlane(Laser1WeltHintenLinks_A,Laser1WeltVorne_A,Laser1WeltHintenRechts_A);
%
% Auswahl der Laserpunkte mit dem kleinsten Fehler zur Ermittlung der Projektionsmatrix
% zwischen dem Laser und der Kamera
%
LaengeVorne = length(Laser1WeltVorne_A);
LaengeHintenLinks = length(Laser1WeltHintenLinks_A);
LaengeHintenRechts = length(Laser1WeltHintenRechts_A);
%
% Ermittlung des 1 Punktes (Auf der linken hinteren Laserlinie in den ersten 2/3 !)
ErrorPunkteHintenLinks = ErrorproPunkt1(1 : (round((2/3)*LaengeHintenLinks)));
[Error,Pointer] = min(abs(ErrorPunkteHintenLinks));
KalibPunkt1WeltKS = [Laser1WeltHintenLinks_A(:,Pointer);1];
KalibPunkt1LaserKS = Laser1T * KalibPunkt1WeltKS;
```

```
KalibPunkt1PixelKS = Laser1HintenLinks_A(:,Pointer);
PunktLinks = KalibPunkt1PixelKS(1,1) - 10;
if PunktLinks < 0
   PunktLinks = 0;
end
PunktOben = KalibPunkt1PixelKS(2,1) - 10;
if PunktOben < 0
   PunktOben = 0;
end
line([PunktLinks KalibPunkt1PixelKS(1,1) + 10],[PunktOben KalibPunkt1PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt1PixelKS(1,1) + 10 PunktLinks],[PunktOben KalibPunkt1PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 2 Punktes (Auf der vorderen Laserlinie im ersten 1/3 !)
ErrorPunkteVorne = ErrorproPunkt1(1 + LaengeHintenLinks :(LaengeHintenLinks + round((1/3)*LaengeVorne)));
[Error,Pointer] = min(abs(ErrorPunkteVorne));
KalibPunkt2WeltKS = [Laser1WeltVorne_A(:,Pointer);1];
KalibPunkt2LaserKS = Laser1T * KalibPunkt2WeltKS;
KalibPunkt2PixelKS = Laser1Vorne_A(:,Pointer);
line([KalibPunkt2PixelKS(1,1) - 10 KalibPunkt2PixelKS(1,1) + 10],...
     [KalibPunkt2PixelKS(2,1) - 10 KalibPunkt2PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt2PixelKS(1,1) + 10 KalibPunkt2PixelKS(1,1) - 10],...
     [KalibPunkt2PixelKS(2,1) - 10 KalibPunkt2PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 3 Punktes (Auf der vorderen Laserlinie im letzten 1/3 !)
ErrorPunkteVorne = ErrorproPunkt1((LaengeHintenLinks + round((2/3)*LaengeVorne)):(LaengeHintenLinks + LaengeVorne));
[Error,Pointer] = min(abs(ErrorPunkteVorne));
KalibPunkt3WeltKS= [Laser1WeltVorne_A(:,(Pointer + round((2/3)*LaengeVorne - 1)));1];
KalibPunkt3LaserKS = Laser1T * KalibPunkt3WeltKS;
KalibPunkt3PixelKS = Laser1Vorne_A(:,(Pointer + round((2/3)*LaengeVorne - 1)));
line(|KalibPunkt3PixelKS(1,1) - 10 KalibPunkt3PixelKS(1,1) + 10],...
     [KalibPunkt3PixelKS(2,1) - 10 KalibPunkt3PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt3PixelKS(1,1) + 10 KalibPunkt3PixelKS(1,1) - 10],...
     [KalibPunkt3PixelKS(2,1) - 10 KalibPunkt3PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 4 Punktes (Auf der rechten hinteren Laserlinie in den letzten 2/3 !)
ErrorPunkteHintenRechts = ErrorproPunkt1((LaengeHintenLinks + LaengeVorne + round((1/3)*LaengeHintenRechts))...
     : (LaengeHintenLinks + LaengeVorne + LaengeHintenRechts));
[Error,Pointer] = min(abs(ErrorPunkteHintenRechts));
KalibPunkt4WeltKS = [Laser1WeltHintenRechts_A(:,(Pointer + round((1/3)*LaengeHintenRechts)) - 1);1];
KalibPunkt4LaserKS = Laser1T * KalibPunkt4WeltKS;
KalibPunkt4PixelKS = Laser1HintenRechts_A(:,(Pointer + round((1/3)*LaengeHintenRechts)) - 1);
line([KalibPunkt4PixelKS(1,1) - 10 KalibPunkt4PixelKS(1,1) + 10],...
     [KalibPunkt4PixelKS(2,1) - 10 KalibPunkt4PixelKS(2,1) + 10],'color','cyan');
line(|KalibPunkt4PixelKS(1,1) + 10 KalibPunkt4PixelKS(1,1) - 10],...
     [KalibPunkt4PixelKS(2,1) - 10 KalibPunkt4PixelKS(2,1) + 10],'color','cyan');
%
% Zusammenfassen der Punkte in eine Matrix
%
KalibPunktWeltKS = [KalibPunkt1WeltKS KalibPunkt2WeltKS KalibPunkt3WeltKS KalibPunkt4WeltKS];
KalibPunktLaserKS = [KalibPunkt1LaserKS KalibPunkt2LaserKS KalibPunkt3LaserKS KalibPunkt4LaserKS];
KalibPunktPixelKS = |KalibPunkt1PixelKS KalibPunkt2PixelKS KalibPunkt3PixelKS KalibPunkt4PixelKS|;
%
% Für die vier Punkte im Laserkoordinatensystem soll die Z-Koordinate Null sein!
% Die Transformationsmatrix LaserT wurde ja aus den Laserpunkten berechnet und da
% die X und Y - Richtung in der Laserebene liegen ist Z = 0 !
% eventuelle Z - Anteile ergeben sich aus Rundungsfehlern!
%
KalibPunktLaser = |KalibPunktLaserKS(1:2,:);zeros(1,length(KalibPunktLaserKS));KalibPunktLaserKS(4,:)];
%
% Nun kann mit Hilfe der svd die Transformationsmatrix zwischen dem Laser KS und dem Pixel KS errechnet werden!
%
ProjektionKameraLaser1 = GenProj(KalibPunktLaser,KalibPunktPixelKS);
%
% Einzeichnen des Laserkoordinatensystems
% Ursprung Laserkoordinatensystem (0;0) in mm!
U_NA = inv(ProjektionKameraLaser1)*[0 0 1]';
% Umrechnung in Pixelkoordinaten
U_A = [U_NA(1,1)/U_NA(3,1);U_NA(2,1)/U_NA(3,1);1];
```

```
% Hilfspunkte
P1_NA = inv(ProjektionKameraLaser1)*[20 0 1]';
P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
P2_NA = inv(ProjektionKameraLaser1)*[0 20 1]';
P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
%
line([U_A(1,1) P1_A(1,1)],[U_A(2,1) P1_A(2,1)],'color','y');
line([U_A(1,1) P2_A(1,1)],[U_A(2,1) P2_A(2,1)],'color','y');
if P2_A(2,1) < U_A(2,1)
   text(U_A(1,1)+5,U_A(2,1)+30,'L1-KS','color','y');
else
   text(U_A(1,1)+5,U_A(2,1)-30,'L1-KS','color','y');
end
%text(U_A(1,1)-20,U_A(2,1)-50,'Y','color','y');
text(P2_A(1,1)-20,P2_A(2,1),'Y','color','y');
%
% Proberechnung (Nicht erforderlich!)
Ur_NA = HHinten*[U_A(1,1);U_A(2,1);1];
Ur_A = [Ur_NA(1,1)/Ur_NA(3,1);Ur_NA(2,1)/Ur_NA(3,1);1];
% Weltkoordinaten des Laserursprungs in mm
% !!!!! = inv(Laser1T)*[0 0 0 1]' !!!!!
%
% Einzeichnen des Weltkoordinatensystems
Ursprung_nichtaffin = inv(HHinten)*[0 0 1]';
% Rücktransformation des Punktes (0,0) in mm
Ursprung_affin = [Ursprung_nichtaffin(1,1)/Ursprung_nichtaffin(3,1);Ursprung_nichtaffin(2,1)/Ursprung_nichtaffin(3,1);1];
% Dieser Punkt muss im Mittelpunkt der ersten LED in der hinteren Ebene liegen!
% Hilfspunkte
P1_NA = inv(HHinten)*[20 0 1]';
P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
P2_NA = inv(HHinten)*[0 20 1]';
P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
P3_NA = inv(HVorne)*[0 0 1]';
P3_A = [P3_NA(1,1)/P3_NA(3,1);P3_NA(2,1)/P3_NA(3,1);1];
P4_NA = inv(HHinten)*[0 150 1]';
P4_A = [P4_NA(1,1)/P4_NA(3,1);P4_NA(2,1)/P4_NA(3,1);1];
% Zeichnen des Koordinatensystems
line([Ursprung_affin(1,1) P1_A(1,1)],[Ursprung_affin(2,1) P1_A(2,1)],'color','m');
line([Ursprung_affin(1,1) P2_A(1,1)],[Ursprung_affin(2,1) P2_A(2,1)],'color','m');
line([Ursprung_affin(1,1) P3_A(1,1)],[Ursprung_affin(2,1) P3_A(2,1)],'color','m');
line([Ursprung_affin(1,1) P4_A(1,1)],[Ursprung_affin(2,1) P4_A(2,1)],'color','m','LineStyle','-.');
%
text(Ursprung_affin(1,1)+20,Ursprung_affin(2,1)+40,'Welt-KS','color','m');
text(Ursprung_affin(1,1)+60,Ursprung_affin(2,1)-10,'X','color','m');
text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)+65,'Y','color','m');
if P3_A(2,1) < Ursprung_affin(2,1)
   text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)-30,'Z','color','m');
else
   text(Ursprung_affin(1,1)-35,Ursprung_affin(2,1)+15,'Z','color','m');
end
%
%
— gleiches Spiel für 2.Laser —
if Length2 > 2
   % es wurde eine 2.Laserlinie gefunden
   [Laser2T,ErrorproPunkt2,Plane2] = FitLaserPlane(Laser2WeltHintenLinks_A,Laser2WeltVorne_A,Laser2WeltHintenRechts_A);
   %
   % Auswahl der Laserpunkte mit dem kleinsten Fehler zur Ermittlung der Projektionsmatrix
   % zwischen dem Laser und der Kamera
   %
   LaengeVorne = length(Laser2WeltVorne_A);
   LaengeHintenLinks = length(Laser2WeltHintenLinks_A);
   LaengeHintenRechts = length(Laser2WeltHintenRechts_A);
   %
   % Ermittlung des 1 Punktes (Auf der linken hinteren Laserlinie in den ersten 2/3 !)
   ErrorPunkteHintenLinks = ErrorproPunkt2(1 : (round((2/3)*LaengeHintenLinks)));
   [Error,Pointer] = min(abs(ErrorPunkteHintenLinks));
   KalibPunkt1WeltKS = [Laser2WeltHintenLinks_A(:,Pointer);1];
```

```
KalibPunkt1LaserKS = Laser2T * KalibPunkt1WeltKS;
KalibPunkt1PixelKS = Laser2HintenLinks_A(:,Pointer);
PunktLinks = KalibPunkt1PixelKS(1,1) - 10;
if PunktLinks < 0
  PunktLinks = 0;
end
PunktOben = KalibPunkt1PixelKS(2,1) - 10;
if PunktOben < 0
  PunktOben = 0;
end
line([PunktLinks KalibPunkt1PixelKS(1,1) + 10],[PunktOben KalibPunkt1PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt1PixelKS(1,1) + 10 PunktLinks],[PunktOben KalibPunkt1PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 2 Punktes (Auf der vorderen Laserlinie im ersten 1/3 !)
ErrorPunkteVorne = ErrorproPunkt2(1 + LaengeHintenLinks :(LaengeHintenLinks + round((1/3)*LaengeVorne)));
[Error,Pointer] = min(abs(ErrorPunkteVorne));
KalibPunkt2WeltKS = [Laser2WeltVorne_A(:,Pointer);1];
KalibPunkt2LaserKS = Laser2T * KalibPunkt2WeltKS;
KalibPunkt2PixelKS = Laser2Vorne_A(:,Pointer);
line([KalibPunkt2PixelKS(1,1) - 10 KalibPunkt2PixelKS(1,1) + 10],...
    [KalibPunkt2PixelKS(2,1) - 10 KalibPunkt2PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt2PixelKS(1,1) + 10 KalibPunkt2PixelKS(1,1) - 10],...
    [KalibPunkt2PixelKS(2,1) - 10 KalibPunkt2PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 3 Punktes (Auf der vorderen Laserlinie im letzten 1/3 !)
ErrorPunkteVorne = ErrorproPunkt2((LaengeHintenLinks + round((2/3)*LaengeVorne)):(LaengeHintenLinks + LaengeVorne));
[Error,Pointer] = min(abs(ErrorPunkteVorne));
KalibPunkt3WeltKS = [Laser2WeltVorne_A(:,(Pointer + round((2/3)*LaengeVorne - 1)));1];
KalibPunkt3LaserKS = Laser2T * KalibPunkt3WeltKS;
KalibPunkt3PixelKS = Laser2Vorne_A(:,(Pointer + round((2/3)*LaengeVorne - 1)));
line([KalibPunkt3PixelKS(1,1) - 10 KalibPunkt3PixelKS(1,1) + 10],...
    [KalibPunkt3PixelKS(2,1) - 10 KalibPunkt3PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt3PixelKS(1,1) + 10 KalibPunkt3PixelKS(1,1) - 10],...
    [KalibPunkt3PixelKS(2,1) - 10 KalibPunkt3PixelKS(2,1) + 10],'color','cyan');
%
% Ermittlung des 4 Punktes (Auf der rechten hinteren Laserlinie in den letzten 2/3 !)
ErrorPunkteHintenRechts = ErrorproPunkt2((LaengeHintenLinks + LaengeVorne + round((1/3)*LaengeHintenRechts))...
    : (LaengeHintenLinks + LaengeVorne + LaengeHintenRechts));
[Error,Pointer] = min(abs(ErrorPunkteHintenRechts));
KalibPunkt4WeltKS = [Laser2WeltHintenRechts_A(:,(Pointer + round((1/3)*LaengeHintenRechts)) - 1);1];
KalibPunkt4LaserKS = Laser2T * KalibPunkt4WeltKS;
KalibPunkt4PixelKS = Laser2HintenRechts_A(:,(Pointer + round((1/3)*LaengeHintenRechts)) - 1);
line([KalibPunkt4PixelKS(1,1) - 10 KalibPunkt4PixelKS(1,1) + 10],...
    [KalibPunkt4PixelKS(2,1) - 10 KalibPunkt4PixelKS(2,1) + 10],'color','cyan');
line([KalibPunkt4PixelKS(1,1) + 10 KalibPunkt4PixelKS(1,1) - 10],...
    [KalibPunkt4PixelKS(2,1) - 10 KalibPunkt4PixelKS(2,1) + 10],'color','cyan');
%
% Zusammenfassen der Punkte in eine Matrix
%
KalibPunktWeltKS = [KalibPunkt1WeltKS KalibPunkt2WeltKS KalibPunkt3WeltKS KalibPunkt4WeltKS];
KalibPunktLaserKS = [KalibPunkt1LaserKS KalibPunkt2LaserKS KalibPunkt3LaserKS KalibPunkt4LaserKS];
KalibPunktPixelKS = [KalibPunkt1PixelKS KalibPunkt2PixelKS KalibPunkt3PixelKS KalibPunkt4PixelKS];
%
% Für die vier Punkte im Laserkoordinatensystem soll die Z-Koordinate Null sein!
% Die Transformationsmatrix LaserT wurde ja aus den Laserpunkten berechnet und da
% die X und Y - Richtung in der Laserebene liegen ist Z = 0 !
% eventuelle Z - Anteile ergeben sich aus Rundungsfehlern!
%
KalibPunktLaser = [KalibPunktLaserKS(1:2,:);zeros(1,length(KalibPunktLaserKS));KalibPunktLaserKS(4,:)];
%
% Nun kann mit Hilfe der svd die Transformationsmatrix zwischen dem Laser KS und dem Pixel KS errechnet werden!
%
ProjektionKameraLaser2 = GenProj(KalibPunktLaser,KalibPunktPixelKS);
%
% Ursprung Laserkoordinatensystem (0;0) in mm!
U_NA = inv(ProjektionKameraLaser2)*[0 0 1]';
% Umrechnung in Pixelkoordinaten
U_A = [U_NA(1,1)/U_NA(3,1);U_NA(2,1)/U_NA(3,1);1];
```

```
  % Hilfspunkte
  P1_NA = inv(ProjektionKameraLaser2)*[20 0 1]';
  P1_A = [P1_NA(1,1)/P1_NA(3,1);P1_NA(2,1)/P1_NA(3,1);1];
  P2_NA = inv(ProjektionKameraLaser2)*[0 20 1]';
  P2_A = [P2_NA(1,1)/P2_NA(3,1);P2_NA(2,1)/P2_NA(3,1);1];
  %
  line([U_A(1,1) P1_A(1,1)],[U_A(2,1) P1_A(2,1)],'color','g');
  line([U_A(1,1) P2_A(1,1)],[U_A(2,1) P2_A(2,1)],'color','g');
  %text(U_A(1,1)-20,U_A(2,1)+30,'L2-KS','color','g');
  if P2_A(2,1) < U_A(2,1)
    text(U_A(1,1)+5,U_A(2,1)+30,'L2-KS','color','g');
  else
    text(U_A(1,1)+5,U_A(2,1)-30,'L2-KS','color','g');
  end
  %text(U_A(1,1)-20,U_A(2,1)-50,'Y','color','g');
  text(P2_A(1,1)-20,P2_A(2,1),'Y','color','g');
    % Proberechnung (Nicht erforderlich!)
  Ur_NA = HHinten*[U_A(1,1);U_A(2,1);1];
  Ur_A = [Ur_NA(1,1)/Ur_NA(3,1);Ur_NA(2,1)/Ur_NA(3,1);1];
  % Weltkoordinaten des Laserursprungs in mm
  % !!!!! = inv(Laser2T)*[0 0 0 1]' !!!!!
else
  Laser2T = 0;
  ProjektionKameraLaser2 = 0;
  Plane2 = 0;
end
%
Valid = 1;
%
```

"FindLaserLines.m"

```
%
function [PointsLaser1,Segmentlaenge1,PointsLaser2,Segmentlaenge2] = FindLaserLines(Pic0,Verkehrt);
%
% Description:
%           This function is searching for laser lines in a given picture.
%
% Input Parameters:
%           Pic0 ... a Picture.
%           Verkehrt ... Picture on top/bottom.
%
% Return Parameters:
%           LaserPoints ... X & Y coordinates of the Laser points.
%           Segmentlaenge ... length of the laser segments.
%
% By:       Norbert Koller
% Date:     April - October 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% Constants for finding the laserlines and to calculate the first moment
%
LaserOffset = 20;          % Die 20 Pixel Ober- und Unterhalb der Laserlinie werden gelöscht
MinSegmentlaenge = 20;     % Ist ein Lasersegment kürzer so handelt es sich um einen Ausreißer und wird verworfen
MaxAbweichung = 5;         % Maximal erlaubte Differenz in Pixel zwischen zwei benachbarten Laserpunkten
PixelUp = 5;               % Ein Rechteck um den Punkt mit der Intensitätsmaximum wird
PixelDown = 5;             % in die Moment-Berechnung miteinbezogen!
PixelLeft = 3;             %
PixelRight = 3;            %
Gewichtung = 3;            % Gewichtung der helleren Punkte
%
Picturelength = size(Pic0,2);
Picturedepth = size(Pic0,1);
```

```
%
Pic1 = Pic0;
Laenge1 = 1;
Laenge2 = 1;
Left1 = 1;
Left2 = 1;
Segmentlaenge1 = 0;
Segmentlaenge2 = 0;
Segment1 = 0;
Segment2 = 0;
Laser(1:4,1:Picturelength) = 0;
%
for i = 1 : Picturelength
   %
   % Es wird 2 mal nach den maximalen Intensitäten in jeder Spalte gesucht
   %
   for t = 1 : 2
    [Intensitaet,Pointer] = max(Pic1(:,i));
    %
    MaxInt(t,i) = Pointer;
    %
    % Berechnung der vertikalen Grenzen
    %
    Up = Pointer - PixelUp;
    if Up < 1
      Up = 1;
    end
    Down = Pointer + PixelDown;
    if Down > Picturedepth
      Down = Picturedepth;
    end
    %
    % Berechnung der horizontalen Grenzen
    %
    Left = i - PixelLeft;
    if Left < 1
      Left = 1;
    end
    Right = i + PixelRight;
    if Right > Picturelength
      Right = Picturelength;
    end
    %
    % Errechnen des Moments
    %
    Zaehler1 = 0;
    Zaehler2 = 0;
    Nenner = 0;
    for p = Left : Right
      for q = Up : Down
        Zaehler1 = Zaehler1 + p * (Pic0(q,p))^(Gewichtung);
        Zaehler2 = Zaehler2 + q * (Pic0(q,p))^(Gewichtung);
        Nenner = Nenner + (Pic0(q,p))^(Gewichtung);
      end
    end
    %
    % x-Komponente
    Bright(2*(t-1)+1,i) = Zaehler1/Nenner;
    % y-Komponente
    Bright(2*(t-1)+2,i) = Zaehler2/Nenner;
    %
    % Löschen der Punkte
    %
    Up = Pointer - LaserOffset;
    if Up < 1
      Up = 1;
    end
    Down = Pointer + LaserOffset;
```

```
    if Down > Picturedepth
      Down = Picturedepth;
    end
    % Die Punkte werden gelöscht
    Pic1(Up:Down,i) = 0;
  end
    % Zusammenfügen der Laserlinien
  %
  if i > 1
    if abs(Bright(4,i)-Bright(2,i-1)) <= MaxAbweichung
      % Zusammenfügen Laserlinie 1
      Merker = Bright(1:2,i);
      Bright(1:2,i) = Bright(3:4,i);
      Bright(3:4,i) = Merker;
    elseif abs(Bright(2,i)-Bright(4,i-1)) <= MaxAbweichung
      % Zusammenfügen Laserlinie 2
      Merker = Bright(3:4,i);
      Bright(3:4,i) = Bright(1:2,i);
      Bright(1:2,i) = Merker;
    end
    %
    % Aussortieren der Ausreißer
    % Laserlinie 1
    if (abs(Bright(2,i)-Bright(2,i-1)) > MaxAbweichung | i == Picturelength) & (Laenge1 > MinSegmentlaenge)
      Laser(1:2,Left1:i-1) = Bright(1:2,Left1:i-1);
      Segment1 = Segment1 + 1;
      Segmentlaenge1(1,Segment1) = Left1;
      Segmentlaenge1(2,Segment1) = i-1;
      Laenge1 = 1;
      Left1 = i;
    elseif abs(Bright(2,i)-Bright(2,i-1)) <= MaxAbweichung
      Laenge1 = Laenge1 + 1;
    else
      Laenge1 = 1;
      Left1 = i;
    end
    % Laserlinie 2
    if (abs(Bright(4,i)-Bright(4,i-1)) > MaxAbweichung | i == Picturelength) & (Laenge2 > MinSegmentlaenge)
      Laser(3:4,Left2:i-1) = Bright(3:4,Left2:i-1);
      Segment2 = Segment2 + 1;
      Segmentlaenge2(1,Segment2) = Left2;
      Segmentlaenge2(2,Segment2) = i-1;
      Laenge2 = 1;
      Left2 = i;
    elseif abs(Bright(4,i)-Bright(4,i-1)) <= MaxAbweichung
      Laenge2 = Laenge2 + 1;
    else
      Laenge2 = 1;
      Left2 = i;
    end
  end
end
%
% Trennen der Laserlinien
% Der ersten zwei und letzten zwei Punkte von jedem Segment wird verworfen
%
Segment1 = 1;
Segment2 = 1;
for i = 1 : length(Segmentlaenge1(1,:))
  L1 = max(Laser(2,((Segmentlaenge1(1,i)+round(MinSegmentlaenge/3)):(Segmentlaenge1(2,i)-round(MinSegmentlaenge/3)))));
  L2 = max(Laser(4,((Segmentlaenge1(1,i)+round(MinSegmentlaenge/3)):(Segmentlaenge1(2,i)-round(MinSegmentlaenge/3)))));
  if ((Verkehrt == 0) & (L1 > L2) & (L2 > 0)) | ((Verkehrt == 1) & (L1 > L2))
    if Segment2 == 1
      PointsLaser2 = [Laser(1,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2);Laser(2,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2)];
    else
      PointsLaser2 = [PointsLaser2(1,:),Laser(1,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2);...
          PointsLaser2(2,:),Laser(2,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2)];
    end
```

```
    S2(Segment2) = length(Laser(1,Segmentlaenge1(1,i):Segmentlaenge1(2,i)))-4;
    Segment2 = Segment2 + 1;
  else
    if Segment1 == 1
      PointsLaser1 = [Laser(1,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2);Laser(2,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2)];
    else
      PointsLaser1 = [PointsLaser1(1,:),Laser(1,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2);...
          PointsLaser1(2,:),Laser(2,Segmentlaenge1(1,i)+2:Segmentlaenge1(2,i)-2)];
    end
    S1(Segment1) = length(Laser(1,Segmentlaenge1(1,i):Segmentlaenge1(2,i)))-4;
    Segment1 = Segment1 + 1;
  end
end
%
for i = 1 : length(Segmentlaenge2(1,:))
  L1 = max(Laser(4,((Segmentlaenge2(1,i)+round(MinSegmentlaenge/3)):(Segmentlaenge2(2,i)-round(MinSegmentlaenge/3)))));
  L2 = max(Laser(2,((Segmentlaenge2(1,i)+round(MinSegmentlaenge/3)):(Segmentlaenge2(2,i)-round(MinSegmentlaenge/3)))));
  if ((Verkehrt == 0) & (L1 > L2) & (L2 > 0)) | ((Verkehrt == 1) & (L1 > L2))
    if Segment2 == 1
      PointsLaser2 = [Laser(3,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2);Laser(4,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2)];
    else
      PointsLaser2 = [PointsLaser2(1,:),Laser(3,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2);...
          PointsLaser2(2,:),Laser(4,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2)];
    end
    S2(Segment2) = length(Laser(3,Segmentlaenge2(1,i):Segmentlaenge2(2,i)))-4;
    Segment2 = Segment2 + 1;
  else
    if Segment1 == 1
      PointsLaser1 = [Laser(3,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2);Laser(4,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2)];
    else
      PointsLaser1 = [PointsLaser1(1,:),Laser(3,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2);...
          PointsLaser1(2,:),Laser(4,Segmentlaenge2(1,i)+2:Segmentlaenge2(2,i)-2)];
    end
    S1(Segment1) = length(Laser(3,Segmentlaenge2(1,i):Segmentlaenge2(2,i)))-4;
    Segment1 = Segment1 + 1;
  end
end
%
Segmentlaenge1 = 0;
Segmentlaenge1 = S1;
Segmentlaenge2 = 0;
Segmentlaenge2 = S2;
%
% Einzeichnen der Punkte
%
fig = figure;
imagesc(Pic1); % Ausnützen des ganzen 16-bit Helligkeitsbereich
axis image; % show axis
%colormap(gray);
hold on;
title('Erfassen der hellsten Punkte');
xlabel('Pixel X-Achse');
ylabel('Pixel Y-Achse');
%
plot(MaxInt(1,:),'rx');
plot(MaxInt(2,:),'rx');
%
%
fig = figure;
imagesc(Pic0); % Ausnützen des ganzen 16-bit Helligkeitsbereich
axis image; % show axis
colormap(gray);
title('Erfassen der Laserlinie(n) & Berechnung des 1. Moments');
xlabel('Pixel X-Achse');
ylabel('Pixel Y-Achse');
hold on; % Bild bearbeiten
%
%plot(MaxInt(1,:),'rx');
```

```
%plot(MaxInt(2,:),'rx');
%
%plot(PointsLaser1(1,:),PointsLaser1(2,:),'ob');
%plot(PointsLaser2(1,:),PointsLaser2(2,:),'og');
%
j = 1;
for i = 1 : length(Segmentlaenge1)
   plot(PointsLaser1(1,j:(j+Segmentlaenge1(i)-1)),PointsLaser1(2,j:(j+Segmentlaenge1(i)-1)),'-b','LineWidth',1.5);
   j = j + Segmentlaenge1(i);
end
%
j = 1;
for i = 1 : length(Segmentlaenge2)
   plot(PointsLaser2(1,j:(j+Segmentlaenge2(i)-1)),PointsLaser2(2,j:(j+Segmentlaenge2(i)-1)),'-g','LineWidth',1.5);
   j = j + Segmentlaenge2(i);
end
%
line([190 230],[560 560],'color','b','LineWidth',1.5);
text(237,560,'Laser Line 1','color','b');
line([440 480],[560 560],'color','g','LineWidth',1.5);
text(487,560,'Laser Line 2','color','g');
%
```

"FitArc.m"

```
%
function [newCircle, newDsOut, newErrorOut, newMaxDsOut] = FitArc(points);
%
% Use:  circle=FitArc( points );
%
% Description:
%           Performs a nonlinear fit of a circle to the data points.
%
% Input Parameters:
%           points: the points as a matrix, the column vectors are the coordinates of
%           the individual points.
%
% Return Parameters:
%           circle: a three entry vector containing [xo, yo, r]
%
% By:       Paul O'Leary
% Date:     1 May 2002
% Version:  1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
maxIterationCount=100;
%
xData=points(1,:);
yData=points(2,:);
%
noPoints=length(xData);
%
% Use a linear RMS fit as the initial estimate for the circle
%
[initCircle, Ds, oldError, maxDs]=FitCirclenew( points );
%
[newCircle, newDs, newError, newMaxDs]=OptimizeCircle( points, initCircle, Ds);
%
iterationCount=0;
%
% Iterate a nonlinear optimization
%
errorLimit=1e-9;
while ( (oldError - newError)/newError > errorLimit ) & (iterationCount < maxIterationCount)
```

```
   iterationCount=iterationCount+1;
   oldError=newError;
   [newCircle, newDs, newError,newMaxDs]=OptimizeCircle(points, newCircle, newDs);
end;
%
if nargout > 1
   newDsOut=newDs;
   newErrorOut=newError;
   newMaxDsOut=newMaxDs;
end;
%
```

"FitCircle.m"

```
%
function Circle=FitCircle(X,Y);
%
% Description:
%           This function calculates the best Klein fit of a circle
%           to a given set of data points.
%
% Input Parameters:
%           XData and YData, the x and y coordinates of the data points as vectors.
%
% Return Parameters:
%           Structured data circle with the following fields:
%             X0 - x coordinate of the center.
%             Y0 - y coordinate of the center.
%             R - the radius.
%             Valid - to indicate a valid circle
%
% By:       Norbert Koller
% Date:     15. April 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
XData=X;
YData=Y;
%
if length(XData)==length(YData) & length(XData) >= 3
%
   XY=(XData).^2+(YData).^2;
   G=[XY' XData' YData' ones(1,size(XY',1))'];
   %
   % G ist die Matrix um die Grassmann Koordinaten zu berechnen.
   % Die Gleichung
   %
   %   |x1^2 + y1^2   x1  y1  1|     |C1|    |0|
   %   |x2^2 + y2^2   x2  y2  1|     |C2|    |0|
   %   |     ...       ...  ... 1|  *  |C3| =  |0|
   %   |     ...       ...  ... 1|     |C4|    |0|
   %   |xn^2 + yn^2   xn  yn  1|     |  |    |0|
   %
   % beschreibt einen Kreis mit den Grassmann Koordinaten (C1,C2,C3 und C4) oder auch tetra circular Koordinaten genannt.
   % C1,C2,C3 und C4 sind homogen und definieren einen 3 dimensionalen Raum.
   %
   % Die Lösung dieser Gleichung wird mit der Singulärwertzerlegung berechnet.
   %
   [U S V] = svd(G);
   C=V(:,end);
   C1=C(1,1);
   C2=C(2,1);
   C3=C(3,1);
   C4=C(4,1);
```

```
  %
  Circle.X0=-C2/(2*C1); % X0,Y0...Kreismittelpunkt
  Circle.Y0=-C3/(2*C1);
  Circle.R=sqrt(Circle.X0^2+Circle.Y0^2-C4/C1); % Radius
  %
  if length(XData) >= 4 % Mehr als 3 Punkte
    Circle.Residual=S(4,4); % Standardabweichung
    Circle.Fehler = U(:,4);
  else % Nur 3 Punkte => Idealer Kreis
    Circle.Residual=0;
    Circle.Fehler = U(:,3);
  end;
  %
  if C1 == 0 % Alle Punkte liegen auf einer Geraden!!
    Circle.Valid=0; % Kreisfit nicht möglich!!
  else
    Circle.Valid=1;
  end;
  %
else
  Circle.Valid=0; % Zuwenig oder falsche Punkte!!!
  Circle.R = 10000;
  Circle.Residual = 100000;
end;
%
```

"FitCirclenew.m"

```
%
function [circle, Ds, meanSquareD, maxDs]=FitCircle( points );
%
% Description:
%           This function calculates the best linear error calculation fit of a circle
%           to a given set of data points.
%
% Input Parameters:
%           The x and y coordinates of the data points in an array.
%
% Return Parameters:
%           Structured data circle with the following fields:
%             X0 - x coordinate of the center.
%             Y0 - y coordinate of the center.
%             R - the radius.
%           error parameters for the circular arc fit.
%
% By:       Paul O'Leary
% Date:     1. May 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
xData=points(1,:);
yData=points(2,:);
n=length(xData);
%
% Determine the matrix entries
%
A1=2*sum(xData.^2);
A2=2*sum(xData.*yData);
A3=2*sum(xData);
%
B1=A2;
B2=2*sum(yData.^2);
B3=2*sum(yData);
%
```

```
C1=-A3/2;
C2=-B3/2;
C3=-n;
%
D1=sum(xData.^3 + xData.*(yData.^2));
D2=sum(yData.^3 + yData.*(xData.^2));
D3=(A1+B2)/2;
%
% Lösen von 3 Gleichungen nach Cramer
%
Delta=det([A1 B1 C1; A2 B2 C2; A3 B3 C3]);
Alpha=det([D1 B1 C1; D2 B2 C2; D3 B3 C3]);
Beta=det([A1 D1 C1; A2 D2 C2; A3 D3 C3]);
Gamma=det([A1 B1 D1; A2 B2 D2; A3 B3 D3]);
%
x0=Alpha/Delta;
y0=Beta/Delta;
z0=Gamma/Delta;
%
r=sqrt(x0^2+y0^2-z0);
%
circle=[x0, y0, r];
%
if nargout > 1
   Ds=( (xData-x0).^2 + (yData-y0).^2 ).^(0.5)-r;
   meanSquareD = mean( Ds.^2 );
   maxDs=max( Ds );
end;
%
```

"FitLaserPlane.m"

```
%
function [LaserT,ErrorproPunkt,E14] = FitLaserPlane(Points1,Points2,Points3);
%
% Description:
%           This function fits a plane with given points in the plane
%           and calculates a transformation matrix
%
%
% Input Parameters:
%           Points in a plane.
%
% Return Parameters:
%           Transformation matrix.
%           Error per point.
%           Cartesian plane coordinates.
%
% By:       Norbert Koller
% Date:     May 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
X = [Points1(1,:) Points2(1,:) Points3(1,:)];
Y = [Points1(2,:) Points2(2,:) Points3(2,:)];
Z = [Points1(3,:) Points2(3,:) Points3(3,:)];
%
% Matrix für die singular value decomposition
%
MSVD = [X' Y' Z' ones(length(X),1)];
%
% Anwendung der svd
%
[U,S,V] = svd(MSVD);
```

```
%
% Plane coordinates
%
E1 = V(1,end);
E2 = V(2,end);
E3 = V(3,end);
E4 = V(4,end);
%
ErrorproPunkt = U(:,4);
%
Residual = S(4,4);
%
if E3 < 0
   % Der Normalvektor der Ebene zeigt in die falsche Richtung!
   E14 = [-E1 -E2 -E3 -E4];
else
   E14 = [E1 E2 E3 E4];
end;
%
% Bestimmung der Lage des Laserkoordinatensystems mit EulerWinkeln
% Berechnung der Drehwinkel um welche das Laserkoordinatensystem verdreht ist!
if E14(1) > 0 & E14(2) > 0
   % Die X&Y -Komponenten des Normalvektors sind positiv => Drehung gegen den Uhrzeigersinn
   CS.phi = -acos((E14(2))/(sqrt((E14(1))^2+(E14(2))^2)));
   %CS.phi = subspace([1 0 0]',[-E14(2) E14(1) 0]');
elseif E14(1) < 0 & E14(2) > 0
   % Die Y-Komp. ist positiv, X-Komponente negativ => Drehung mit dem Uhrzeigersinn
   CS.phi = acos((E14(2))/(sqrt((E14(1))^2+(E14(2))^2)));
   %CS.phi = subspace([-E14(2) E14(1) 0]',[1 0 0]');
elseif E14(1) > 0 & E14(2) < 0
   % Die Y-Komp. ist positiv, X-Komponente positiv => Drehung mit dem Uhrzeigersinn
   CS.phi = pi - acos((E14(2))/(sqrt((E14(1))^2+(E14(2))^2)));
else
   % Die Y-Komp. ist negativ, X-Komp negativ => Drehung gegen den Uhrzeigersinn
   CS.phi = pi + acos((E14(2))/(sqrt((E14(1))^2+(E14(2))^2)));
end
%
% Winkel zwischen Welt X-Achse und Laser X-Achse (Rotation um die Z-Achse)
%
if E14(2) > 0
   % Die Y-Komponente des Normalvektors ist positiv => Drehung gegen den Uhrzeigersinn!!
   CS.theta = pi - acos((E14(3))/(sqrt((E14(1))^2+(E14(2))^2+(E14(3))^2)));
   %CS.theta = pi - subspace([0 0 1]',[E14(1) E14(2) E14(3)]');
else
   % Die Y-Komponente ist nur bei auf den Kopf gestellten Bildern negativ !!
   % => Drehung mit dem Uhrzeigersinn !!
   CS.theta = acos((E14(3))/(sqrt((E14(1))^2+(E14(2))^2+(E14(3))^2)));
   %CS.theta = subspace([0 0 1]',[E14(1) E14(2) E14(3)]'); end;
% Winkel zwischen Welt Z-Achse und Laser Z-Achse (Rotation um die X'-Achse)
CS.psi=0;
%CS.psi1 = subspace([-E14(1)*E14(3) -E14(2)*E14(3) (E14(1)^2+E14(2)^2)]',[0 E14(2) E14(3)]');
%
Tx=[cos(CS.phi) sin(CS.phi) 0 0; -sin(CS.phi) cos(CS.phi) 0 0; 0 0 1 0; 0 0 0 1];
Ty=[1 0 0 0; 0 cos(CS.theta) sin(CS.theta) 0; 0 -sin(CS.theta) cos(CS.theta) 0; 0 0 0 1];
Tz=[cos(CS.psi) sin(CS.psi) 0 0; -sin(CS.psi) cos(CS.psi) 0 0; 0 0 1 0; 0 0 0 1];
%
%
%       |  cos(phi)                  sin(phi)                  0           0|
% Tr=   | -sin(phi) * cos(theta)     cos(phi) * cos(theta)     sin(theta)  0|
%       |  sin(phi) * sin(theta)    -cos(phi) * sin(theta)     cos(theta)  0|
%       |  0                         0                         0           1|
%
% Rotationsmatrix um Punkte vom Welt Koordinatensystem ins Laser Koordinatensystem zu transformieren !
%
% Berechnung des translatorischen Abstandes zwischen den Nullpunkten der Koordinatensystame
CS.dx = 0; % x-Abstand des Ursprungs WK-System - LK-System
CS.dy = -E14(4)/E14(2); % y-Abstand des Ursprungs WK-System - LK-System
CS.dz = 0; % z-Abstand des Ursprungs WK-System - LK-System
```

```
%
%
Tt = [1 0 0 -CS.dx; 0 1 0 -CS.dy; 0 0 1 -CS.dz; 0 0 0 1];
%
% Translationsmatrix
% Da der Ursprung des Laserkoordinatensystems auf der Y-Achse des Weltkoordinatensystems liegt
% sind die Abstände CS.dx und CS.dz Null!!
%
% Berechnung der Umrechnungsmatrix
%
LaserT=Tz*Ty*Tx*Tt;
%
% Mithilfe dieser Matrix lässt sich ein Punkt vom Welt KS in das Laser KS transformieren!
% LaserTinv=inv(LaserT); (InverseMatrix)
%
```

"FitLine.m"

```
%
function Line=FitLine(XData,YData);
%
% Description:
%           This function calculates the best Klein fit of a line
%           to a given set of data points.
%
% Input Parameters:
%           XData and YData, the x and y coordinates of the first line as vectors.
%
% Return Parameters:
%    Structured data line with the following fields:
%    X,Y,N  - Planner line coordinates of the line.
%    K      - Slope of the line.
%    D1     - gap of the line on the y-axis.
%    R      - Residual
%    Error  - Error of each point
%    Valid  - to indicate a valid line.
%
% By:       Norbert Koller
% Date:     29. April 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
%XData=DataPoints.XData;
%YData=DataPoints.YData;
%
if length(XData) == length(YData) & length(XData) >= 2
   %
   G=[XData' YData' ones(1,size(XData',1))'];
   %
   % G ist die Matrix um die Grassmann Koordinaten zu berechnen.
   % Die Gleichung
   %
   %     | x1  y1  1 |           |0|
   %     | x2  y2  1 |   | Y|    |0|
   %     | ... ... 1 | * |-X| =  |0|
   %     | ... ... 1 |   | N|    |0|
   %     | xn  yn  1 |           |0|
   %
   % beschreibt eine Linie mit den Grassmann Koordinaten (Y,-X und N)
   % oder auch planner line Koordinaten genannt.
   % Y,-X und N beschreiben die Gleichung der Linie in Koordinatendarstellung.
   %
   % Die Lösung dieser Gleichung wird mit der Singular Value Decomposition berechnet.
   %
```

```
  [U S V] = svd(G);
  %
  C=V(:,end);
  C1=C(1,1);
  C2=C(2,1);
  C3=C(3,1);
  %
  Line.X=-C2; % X,Y,D...Planner Line Coordinates
  Line.Y=C1; % Y * x - X * y + N = 0
  Line.N=C3;
  %
  % Überführen der allgemeinen Geradengleichung in die Normalform
  if Line.X == 0 % Linie parallel zur Y-Achse
    Line.K= inf;
    Line.D1= inf;
  else
    Line.K= Line.Y/Line.X;
    Line.D1= Line.N/Line.X;
  end;
  %
  if length(XData) >= 3 % Mehr als 2 Punkte
    Line.Residual=S(3,3); % Standardabweichung
    Line.Error = U(:,3);
  else
    % Nur 2 Punkte => Ideale Linie
    Line.Residual=0;
    Line.Error = [0;0];
  end;
  %
  Line.Valid = 1;
  %
else
  Line.Valid=0; % Zuwenig oder falsche Punkte!!!
end;
%
```

"FitParLine.m"

```
%
function LinePar = FitParLine(DataPoints1,DataPoints2);
%
% Description:
%           This function calculates the best Klein fit of two parallel lines
%           to a given set of data points.
%
% Input Parameters:
%           XData and YData, the x and y coordinates of the first line as vectors.
%           XData and YData, the x and y coordinates of the second line as vectors.
%
% Return Parameters:
%    Structured data line with the following fields:
%    X,Y,N1,N2 - Planner line coordinates of the lines.
%    K         - Slope of the lines.
%    D1,D2     - gap of the lines on the y-axis.
%    R         - Residual
%    Valid     - to indicate a valid line.
%
% By:       Norbert Koller
% Date:      30. April 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
XData1=DataPoints1.XData;
YData1=DataPoints1.YData;
```

```
XData2=DataPoints2.XData;
YData2=DataPoints2.YData;
%
if length(XData1)==length(YData1) & length(XData2)==length(YData2) & length(XData1) >= 2 & length(XData2) >= 2
   %
   XData = [XData1 XData2];
   YData = [YData1 YData2];
   Line3 = [ones(1,size(XData1',1)) zeros(1,size(XData2',1))];
   Line4 = [zeros(1,size(XData1',1)) ones(1,size(XData2',1))];
   %
   G=[XData' YData' Line3' Line4'];
   %
   % G ist die Matrix um die Grassmann Koordinaten zu berechnen.
   % Die Gleichung
   %
   %     |1x1  1y1  1  0|              |0|
   %     |1x2  1y2  1  0|              |0|
   %     |...  ...  1  0|   |   Y|     |0|
   %     |1xn  1yn  1  0| * |  -X| =   |0|
   %     |2x1  2y1  0  1|   |  N1|     |0|
   %     |2x2  2y2  0  1|   |  N2|     |0|
   %     |...  ...  0  1|              |0|
   %     |2xn  2yn  0  1|              |0|
   %
   % beschreibt zwei parallele Linie mit den Grassmann Koordinaten (Y,-X,N1 und N2) oder auch planner line Koordinaten genannt.
   % Y,-X und N1 beschreiben die Gleichung der ersten Linie in Koordinatendarstellung.
   % Y,-X und N2 beschreiben die Gleichung der zweiten Linie in Koordinatendarstellung
   %
   % Die Lösung dieser Gleichung wird mit der Singular Value Decomposition berechnet.
   %
   [U S V] = svd(G);
   %
   C=V(:,end);
   C1=C(1,1);
   C2=C(2,1);
   C3=C(3,1);
   C4=C(4,1);
   %
   LinePar.X=-C2; % X,Y,D...Planner Line Coordinates
   LinePar.Y=C1; % Y * x - X * y + N = 0
   LinePar.N1=C3;
   LinePar.N2=C4;
   %
   % Überführen der allgemeinen Geradengleichung in die Normalform
   if LinePar.X == 0 % Linie parallel zur Y-Achse
     LinePar.K= inf;
     LinePar.D1= inf;
     LinePar.D2= inf;
   else
     LinePar.K= LinePar.Y/LinePar.X;
     LinePar.D1= LinePar.N1/LinePar.X;
     LinePar.D2= LinePar.N2/LinePar.X;
   end;
   %
   if length(XData) >= 3 % Mehr als 3 Punkte
     LinePar.Residual=S(4,4); % Standardabweichung
   else % Nur 3 Punkte => Ideale Linie
     LinePar.Residual=0;
   end;
   LinePar.Valid = 1;
   %
else
   LinePar.Valid = 0; % Zuwenig oder falsche Punkte!!!
end;
%
```

"GenProj.m"

```
%
function [H] = GenProj (p , pH);
%
% Description:
%           This function calculates the matrix H
%           from the equation p⃗ = H p⃗_H.
%
% Input Parameters:
%           vector p⃗    : world-coordinates for n points.
%           vector p⃗_H   : corresponding pixel-coordinates.
%
% Return Parameters:
%           transformation matrix H.
%
% By:       Paul O'Leary
% Date:
% Version: 1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
if length(p)   = length(pH)
   error('Ungültige Daten');
end
%
if length(p) < 4
   error('Zuwenig Punkte vorhanden');
end
%
n = length(p);
%
% Erzeugen der homogenen Koordinaten
%
pR = [p(1,:); p(2,:); ones(1,n)];
%
% Erzeugen der affinen Koordinaten
%
pRA = [pR(1,:)./pR(3,:); pR(2,:)./pR(3,:)];
%
% Bilden der Einträge der Koeffizientenmatrix G
%
for i = 1 : n
   %
   j = 2*i;
   k = 2*i-1;
   %
   % die Zeilen, die aus den x-Koordinaten der 4 Punkte entstehen
   %
   G(k,1) = -pH(1,i);
   G(k,2) = -pH(2,i);
   G(k,3) = -pH(3,i);
   G(k,4) = 0;
   G(k,5) = 0;
   G(k,6) = 0;
   G(k,7) = pRA(1,i) * pH(1,i);
   G(k,8) = pRA(1,i) * pH(2,i);
   G(k,9) = pRA(1,i);
   %
   % die Zeilen, die aus den y-Koordinaten der 4 Punkte entstehen
   %
   G(j,1) = 0;
   G(j,2) = 0;
   G(j,3) = 0;
   G(j,4) = -pH(1,i);
   G(j,5) = -pH(2,i);
   G(j,6) = -pH(3,i);
```

```
   G(j,7) = pRA(2,i) * pH(1,i);
   G(j,8) = pRA(2,i) * pH(2,i);
   G(j,9) = pRA(2,i);
end
%
% Anwendung der svd
%
[U S V] = svd (G);
%
h = V(:,end);
%
% Sammeln der Einträge in der Abbildungsvorschrift H
%
H = [h(1:3)'; h(4:6)'; h(7:9)'];
%
```

"MergeLaserLine.m"

```
%
function [Circle,Linie1,Linie2] = MergeLaserLine(Punkte,Laenge);
%
% Description:
%           This function is merging lasersegments which belongs together
%           and selects if the segment describes a line or a circle.
%
% Input Parameters:
%           LaserPoints.
%           Length of each segment.
%
% Return Parameters:
%           in case of a circle the circle points.
%           in case of line(s) the line(s) points.
%
% By:       Norbert Koller
% Date:     May 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
Kreissegment = 1;
Liniensegment1 = 1;
Liniensegment2 = 1;
Circle.XData = 0;
Linie1.XData = 0;
Linie2.XData = 0;
Zaehler = 0;
%
for i = 1 : length(Laenge)
   % in der Matrix Punkte stehen alle gefundenen Laserpunkte
   % welche nun in die einzelnen Liniensegmente unterteilt werden
   %
   Zaehler = Zaehler + Laenge(i);
   Points = Punkte(:,(1 + Zaehler - Laenge(i)) : Zaehler);
   %
   % Es wird nur die X,Z-Ebene betrachtet
   %
   % Abzug des Mittelwerts für Kreisfit
   Xmean = mean(Points(1,:));
   Zmean = mean(Points(3,:));
   %
   % 1.Kreisfit ----- mit Grassmannkoordinaten -----
   AntwortKreis=FitCircle(Points(1,:) - Xmean,Points(3,:) - Zmean);
   %
   if AntwortKreis.R > 20 & AntwortKreis.R < 140 & length(Points) > 40 & AntwortKreis.Residual < 2
     % bei den gefundenen Punkten handelt es sich um einen Kreis
```

```
  %
  if Kreissegment == 1
    % 1 Kreissegment
    %
    Circle.XData = Points(1,:);
    Circle.YData = Points(2,:);
    Circle.ZData = Points(3,:);
    Circle.X0 = AntwortKreis.X0 + Xmean;
    Circle.Z0 = AntwortKreis.Y0 + Zmean;
    Circle.R = AntwortKreis.R;
    Kreissegment = Kreissegment + 1;
    %
  elseif abs(AntwortKreis.X0 + Xmean - Circle.X0) < 5 & abs(AntwortKreis.Y0 + Zmean - Circle.Z0) < 5...
      & abs(AntwortKreis.R - Circle.R) < 5
    % Zusammenfügen von zusammengehörigen Kreissegmenten
    % Das Kreissegment gehört zum vorherigen Kreissegment
    Circle.XData = [Circle.XData Points(1,:)];
    Circle.YData = [Circle.YData Points(2,:)];
    Circle.ZData = [Circle.ZData Points(3,:)];
    Xmean = mean(Circle.XData);
    Zmean = mean(Circle.ZData);
    AntwortKreis=FitCircle(Circle.XData - Xmean,Circle.ZData - Zmean);
    Circle.X0 = Circle.X0 + Xmean;
    Circle.Z0 = Circle.Z0 + Zmean;
    Circle.R = Circle.R;
  else
    % Das Kreissegment gehört nicht zum vorherigen Kreissegment
    % und wird aus diesem Grund verworfen!
    disp('WARNUNG: Es wurden 2 Kreise gefunden, es wird nur der erste ausgewertet!');
  end
  %
else
  % bei den gefitteten Punkten handelt es sich um eine Linie
  % Linienfit
  AntwortLinie = FitLine(Points(1,:),Points(3,:));
  if AntwortLinie.Valid == 1
    Slope = (180/pi)*atan(AntwortLinie.K);
    Abstand = AntwortLinie.D1;
    %
    if Liniensegment1 == 1
      % 1.Liniensegment
      Linie1.XData = Points(1,:);
      Linie1.YData = Points(2,:);
      Linie1.ZData = Points(3,:);
      SlopeLinie1 = Slope;
      AbstandLinie1 = Abstand;
      Liniensegment1 = Liniensegment1 + 1;
    elseif ((abs(Slope - SlopeLinie1) < 5) & (abs(Abstand - AbstandLinie1) < 10))
      % Die Punkte werden zum ersten Liniensegment addiert
      % Zusammenfügen von zusammengehörenden Liniensegmenten
      Linie1.XData = [Linie1.XData Points(1,:)];
      Linie1.YData = [Linie1.YData Points(2,:)];
      Linie1.ZData = [Linie1.ZData Points(3,:)];
      AntwortLinie = FitLine(Linie1.XData,Linie1.ZData);
      SlopeLinie1 = (180/pi)*atan(AntwortLinie.K);
      AbstandLinie1 = AntwortLinie.D1;
      %
    elseif (Liniensegment1 > 1) & (Liniensegment2 == 1)
      % Die Punkte gehören zum zweiten Liniensegment
      Linie2.XData = Points(1,:);
      Linie2.YData = Points(2,:);
      Linie2.ZData = Points(3,:);
      SlopeLinie2 = Slope;
      AbstandLinie2 = Abstand;
      Liniensegment2 = Liniensegment2 + 1;
    elseif ((abs(Slope - SlopeLinie2) < 5) & (abs(Abstand - AbstandLinie2) < 10))
      % Die Punkte werden zum zweiten Liniensegment addiert
      Linie2.XData = [Linie2.XData Points(1,:)];
```

```
        Linie2.YData = [Linie2.YData Points(2,:)];
        Linie2.ZData = [Linie2.ZData Points(3,:)];
        AntwortLinie = FitLine(Linie2.XData,Linie2.ZData);
        SlopeLinie2 = (180/pi)*atan(AntwortLinie.K);
        AbstandLinie2 = AntwortLinie.D1;
      end
    end
  end
end
%
```

"OptimizeCircle.m"

```
%
function [nextCircle, newDs, meanSquareDs, maxDs]=OptimizeCircle(points, circle, Ds);
%
% Description:
%           This function is the iterative part of the function "FitArc.m"
%
% Input Parameters:
%           Circlepoints.
%           Radius and the coordinates of the centerpoint of the circle.
%           Error of the circle function.
%
% Return Parameters:
%           next circle parameter.
%
% By:       Paul O'Leary
% Date:     1. May 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
NoPoints=length( points );
%
x0=circle(1);
y0=circle(2);
%
xData=points(1,:) - x0;
yData=points(2,:) - y0;
%
Lengths=(xData.^2+yData.^2).^0.5;
Distances=circle(3) - Lengths;
%
Sines=yData./Lengths;
Cosines=xData./Lengths;
%
A1=mean(Cosines.^2);
A2=mean(Sines.*Cosines);
A3=mean(Cosines);
%
B1=A2;
B2=mean(Sines.^2);
B3=mean(Sines);
%
C1=A3;
C2=B3;
C3=1;
%
h1=mean(Distances.*Cosines);
h2=mean(Distances.*Sines);
h3=mean(Distances);
H=[h1; h2; h3];
%
A=[A1 B1 C1; A2 B2 C2; A3 B3 C3];
```

```
%
F=inv(A)*H;
%
DeltaX0=F(1);
DeltaY0=F(2);
DeltaR=F(3);
%
nextCircle=[ circle(1) - DeltaX0, circle(2) - DeltaY0, circle(3) - DeltaR ];
%
newdX=points(1,:)-nextCircle(1);
newdY=points(2,:)-nextCircle(2);
%
newDs=nextCircle(3) - ( (newdX).^2 + (newdY).^2 ).^(0.5) ;
meanSquareDs=mean( newDs.^2 );
maxDs=max( newDs );
%
```

"PlotCircle.m"

```
%
function plotCircle(OnToFigure, Circle, NoPoints, LineType);
%
% Description:
%           This function plots a circle onto a given figure.
%
% Input Parameters:
%           OnFigure: Handle to the figure on which the plot is to be made.
%           Circle: The circle object which is to be plotted.
%           NoPoints: Number of points to be used during plotting (optional default=256)
%           LindType: The line type specification for the plot (optional default='b')
%
% By:       Paul O'Leary
% Date:     19. April 2000
% Version:  1.0
%
% (c) 2000, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date: 19 April 2000
% Comment: Original version.
%
if nargin < 4
   LineType='b';
end;
%
if nargin == 2
   NoPoints = 2^8;
end;
%
figure(OnToFigure);
hold on;
%
Scale = 2*pi*(0:(NoPoints-1))/NoPoints;
%
Sine = sin(Scale);
Cosine = cos(Scale);
%
XData = Circle.X0 + Circle.R*Sine;
YData = Circle.Y0 + Circle.R*Cosine;
%
plot([XData XData(1)],[YData YData(1)],LineType);
plot(Circle.X0,Circle.Y0,'gx');
%
```

"RotateCircle.m"

```
%
function [Laser1CircleKorr,Laser2CircleKorr] = RotateCircle(Laser1Circle,Laser2Circle,LaserPlane1,LaserPlane2);
%
% Description:
%           This function is rotating the circle points parallel to the y-axis
%
% Input Parameters:
%           Circle points laser1
%           Circle points laser2
%           Plane coordinates laser1
%           Plane coordinates laser2
%
% Return Parameters:
%           Corrected laser points for both circles
%
% By:       Norbert Koller
% Date:     20 September 2002
% Version:  1.0
%
% (c) 2002, Institute for Automation, University of Leoben, Leoben, Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% Berechnung des Mittelpunktvektors der Zylinderpunkte der ersten Laserlinie
% Der Mittelpunkt muss in der Laserebene liegen !
% X & Z Werte aus der Routine MergeLaserLine (Grassmannkoordinaten-Fit)
%
VektorX = Laser1Circle.X0;
VektorZ = Laser1Circle.Z0;
% X*Plane1(1) + Y*Plane1(2) + Z*Plane1(3) + Plane1(4) = 0 ... Glg. der Laserebene
VektorY = -(VektorX*LaserPlane1(1)+VektorZ*LaserPlane1(3)+LaserPlane1(4))/LaserPlane1(2);
VektorLaser1 = [VektorX VektorY VektorZ];
%
% Berechnung des Mittelpunktvektors der Zylinderpunkte der zweiten Laserlinie
%
VektorX = Laser2Circle.X0;
VektorZ = Laser2Circle.Z0;
VektorY = -(VektorX*LaserPlane2(1)+VektorZ*LaserPlane2(3)+LaserPlane2(4))/LaserPlane2(2);
VektorLaser2 = [VektorX VektorY VektorZ];
%
% Berechnung des Normalvektors des Zylinders = Längsachse
%
Normalvektor = (VektorLaser2 - VektorLaser1);
%max(Laser1Circle.ZData)
%max(Laser2Circle.ZData)
%Normalvektor(3) = max(Laser2Circle.ZData)-max(Laser1Circle.ZData)
%
% Abzug der Mittelpunktvektoren von den Kreispunkten
%
P10 = [Laser1Circle.XData-VektorLaser1(1);Laser1Circle.YData-VektorLaser1(2);Laser1Circle.ZData-VektorLaser1(3)];
P20 = [Laser2Circle.XData-VektorLaser2(1);Laser2Circle.YData-VektorLaser2(2);Laser2Circle.ZData-VektorLaser2(3)];
%
% Ab hier wird iterativ versucht die genaue Position des Normalvektors des Zylinders zu ermitteln
% max. 5 Iterationsschritte
%
Normalvektoralt = [0 0 0];
i = 0;
while (i < 5) & (abs(sum(sqrt(Normalvektor.^2))-sum(sqrt(Normalvektoralt.^2))) > 0.001)
   i = i + 1;
   if Normalvektor(2) < 0
     % Der Vektor zeigt in die falsche Richtung
     Normalvektor = -Normalvektor;
   end
   Normalvektoralt = Normalvektor;
   % Die Laserpunkte werden nun parallel zur Welt y-Achse gedreht!
   % Berechnung der Winkel
   %
```

```
if Normalvektor(1) > 0
  % Die X-Komponente des Normalvektors ist positiv
  % Drehung mit dem Uhrzeigersinn
  % (von oben betrachtet!!)
  % phi = Winkel zwischen y & y'
  CS.phi = acos((Normalvektor(2))/(sqrt((Normalvektor(1))^2+(Normalvektor(2))^2)));
  %CS.phi = subspace([Normalvektor(1) Normalvektor(2) 0]',[0 1 0]');
else
  % Drehung gegen den Uhrzeigersinn
  CS.phi = -acos((Normalvektor(2))/(sqrt((Normalvektor(1))^2+(Normalvektor(2))^2)));
  %CS.phi = -subspace([0 1 0]',[Normalvektor(1) Normalvektor(2) 0]');
end
% Winkel zwischen Welt X-Achse und Zylinder X-Achse (Rotation um die Z-Achse)
% Neue Achsenrichtungen
% x' : [Normalvektor(2) -Normalvektor(1) 0]
% y' : [Normalvektor(1) Normalvektor(2) 0]
% z' : [0 0 1]
%
if Normalvektor(3) > 0
  % Die Z-Komponente des Normalvektors ist positiv
  % Drehung gegen den Uhrzeigersinn
  % theta = Winkel zwischen y' und y"
  CS.theta = -acos((Normalvektor(1)^2+Normalvektor(2)^2)/(sqrt(Normalvektor(1)^2+...
      Normalvektor(2)^2)*sqrt((Normalvektor(1))^2+(Normalvektor(2))^2+(Normalvektor(3))^2)));
  %CS.theta = -subspace([Normalvektor(1) Normalvektor(2) 0]',[Normalvektor(1) Normalvektor(2) Normalvektor(3)]');
  % Winkel zwischen z' & z"
  %CS.theta = -acos((Normalvektor(1)^2+Normalvektor(2)^2)/(sqrt((Normalvektor(1)*Normalvektor(3))^2+...
      %(Normalvektor(2)*Normalvektor(3))^2+(Normalvektor(1)^2+Normalvektor(2)^2)^2)))
  % Winkel zwischen der y und y"- Achse
else
  % Drehung mit dem Uhrzeigersinn
  CS.theta = acos((Normalvektor(1)^2+Normalvektor(2)^2)/(sqrt(Normalvektor(1)^2+...
      Normalvektor(2)^2)*sqrt((Normalvektor(1))^2+(Normalvektor(2))^2+(Normalvektor(3))^2)));
  %CS.theta = subspace([Normalvektor(1) Normalvektor(2) 0]',[Normalvektor(1) Normalvektor(2) Normalvektor(3)]');
  %CS.theta = acos((Normalvektor(2))/(sqrt((Normalvektor(2))^2+(Normalvektor(3))^2)));
end
% Winkel zwischen Welt Z-Achse und Zylinder Z-Achse (Rotation um die X'-Achse)
% Neue Achsenrichtungen
% x" : [Normalvektor(2) -Normalvektor(1) 0]
% y" : [Normalvektor(1) Normalvektor(2) Normalvektor(3)]
% z" : [-Normalvektor(1)*Normalvektor(3) -Normalvektor(2)*Normalvektor(3) Normalvektor(1)^2+Normalvektor(2)^2]
%
%CS.theta=0;
%CS.phi =0;
CS.psi=0;
%
Tx=[cos(CS.phi) sin(CS.phi) 0; -sin(CS.phi) cos(CS.phi) 0; 0 0 1];
Ty=[1 0 0; 0 cos(CS.theta) sin(CS.theta); 0 -sin(CS.theta) cos(CS.theta)];
Tz=[cos(CS.psi) sin(CS.psi) 0; -sin(CS.psi) cos(CS.psi) 0; 0 0 1];
%
% Rotationsmatrix!
%
TRotation=Tz*Ty*Tx;
%
% Transformation der Punkte
% Drehung der Punkte um ihre eigenen Mittelpunkte
%
Pgedreht = 0;
Pgedreht = inv(TRotation)*P10;
%
% Die Mittelpunkte werden wieder zuaddiert
%
Laser1CircleKorr = [Pgedreht(1,:) + VektorLaser1(1);Pgedreht(2,:) + VektorLaser1(2);Pgedreht(3,:) + VektorLaser1(3)];
%
% Laser2
%
Pgedreht = 0;
Pgedreht = inv(TRotation)*P20;
```

```
    %
    Laser2CircleKorr = [Pgedreht(1,:) + VektorLaser2(1);Pgedreht(2,:) + VektorLaser2(2);Pgedreht(3,:) + VektorLaser2(3)];
    %
    % Der Normalvektor des Zylinders wird mit den gedrehten Punkten neu berechnet
    %
    % Kreisfit
    %
    CircleL1 = FitCirclenew([(Laser1CircleKorr(1,:) - mean(Laser1CircleKorr(1,:)));(Laser1CircleKorr(3,:))]);
    CircleL2 = FitCirclenew([(Laser2CircleKorr(1,:) - mean(Laser2CircleKorr(1,:)));(Laser2CircleKorr(3,:))]);
    %
    % Die Mittelpunkte sind für Laser1
    %
    %[PlaneY1] = FitPlane(Laser1CircleKorr);
    PlaneY1 = (inv(TRotation)*LaserPlane1(1:3)')';
    PlaneY1(4) = LaserPlane1(4);
    VektorX = mean(Laser1CircleKorr(1,:)) + CircleL1(1);
    VektorZ = CircleL1(2);
    % X*PlaneN1(1) + Y*PlaneN1(2) + Z*PlaneN1(3) + PlaneN1(4) = 0 ... Glg. der Laserebene
    VektorY = -(VektorX*PlaneY1(1)+VektorZ*PlaneY1(3)+PlaneY1(4))/PlaneY1(2);
    %
    VektorY1 = [VektorX VektorY VektorZ];
    %
    % Für Laser 2
    %
    %[PlaneY2] = FitPlane(Laser2CircleKorr);
    PlaneY2 = (inv(TRotation)*LaserPlane2(1:3)')';
    PlaneY2(4) = LaserPlane2(4);
    VektorX = mean(Laser2CircleKorr(1,:)) + CircleL2(1);
    VektorZ = CircleL2(2);
    %
    VektorY = -(VektorX*PlaneY2(1)+VektorZ*PlaneY2(3)+PlaneY2(4))/PlaneY2(2);
    %
    VektorY2 = [VektorX VektorY VektorZ];
    %
    % Der neue Normalvektor errechnet sich zu
    %
    Normalvektor = (VektorY2 - VektorY1);
    %Normalvektor = (TRotation*Normal')';
    %
    % Ende der Iteration
end
%
% ****** Probe ******
%
%Vektor1Korr = inv(TRotation)*VektorLaser1(1:3)';
%Vektor2Korr = inv(TRotation)*VektorLaser2(1:3)';
%
%format long;
%
%Mittelpunktvektor = (Vektor1Korr-Vektor2Korr)
% die x und z- Komponente des Mittelpunktvektors müssen Null sein
%NormalPlane1(1:3)./Normalvektor
% die x und y- Komponenten müssen ident sein
%NormalPlane2(1:3)./Normalvektor
% die x und y- Komponenten müssen ident sein
%
%format short g;
%
```

Literaturverzeichnis

[1] Paul O'Leary, Ronald Ofner - Image Processing and Grassmannian Reduction Applied to the Measurement of Cylindrical Metallic Parts.
© July 2001, Institute for Automation, University of Leoben

[2] Ewald Fauster, Peter Schalk - Kalibrationsverfahren für einen Lichtschnittmesskopf
© Juli-September 2001, Institut für Automation, Montanuniversität Leoben

[3] Image Processing Toolbox - For use with *Matlab*®
Users Guide © 1993-2001 The Math Works, Inc

[4] Helmut Kopka und Patrick W. Daly. A Guide to *LaTeX – Document Preparation for Beginners and Advanced Users - Third edition.*
© Addison–Wesly Longman Limited 1999

[5] VDI - Wärmeatlas - Herausgeber: Verein Deutscher Ingenieure
5. erweiterte Auflage 1988, VDI-Verlag GmbH Düsseldorf

[6] Paul O'Leary - Fitting Geometric Models in Image Processing using Grassmann Manifolds
Institute for Automation, University of Leoben

[7] S.H.Joseph - Unibased Least Squares Fitting of Circular Arcs
Department of Mechanical and Process Engineering, University of Sheffield.

[8] Hans Jochen Bartsch - Taschenbuch mathematischer Formeln
16 verbesserte Auflage, © 1994, Fachbuchverlag Leipzig GmbH

[9] Josef Naas und Hermann L. Schmidt
Mathematisches Wörterbuch, Band 1 (A-K) © 1974, Akademie-Verlag, Berlin

[10] Bronstein und Semendjajew - Taschenbuch der Mathematik
© B.G.Teubner Verlagsgesellschaft Leipzig 1996

[11] Ronald Ofner - Three Dimensional Measurement via the Light-Sectioning Method (Doctoral Thesis) © March 2000, Institute for Automation, University of Leoben

[12] Ronald Ofner - Kamerastandards
© September 1998, Institut für Automation, Montanuniversität Leoben

[13] E. Lombardo, M. Martorelli, V. Nigrelli
Non-Contact Roughness Measurement in Rapid Prototypes by Conoscopic Holography, XII ADM International Conference - Rimini - $5^{th} - July$, 2001