

---

# **Szenenkonstruktion und Kamerakalibrierung aus stereoskopischen Bildquellen gleicher und verschiedener Auflösungen**

---

**Thesis**  
zur Erlangung des Grades  
**Master of Science**

Medieninformatik 4.Semester Anja Kretschmer 254793

Betreut von: Prof. Dr. Thomas Schneider



Fakultät Digitale Medien der Hochschule Furtwangen  
Wintersemester 17/18 - Sommersemester 18

# Inhaltsverzeichnis

<b>Abstract</b>	<b>4</b>
<b>1 Einleitung</b>	<b>5</b>
<b>2 Model der Bildaufnahme mit einer Kamera</b>	<b>7</b>
2.1 Lochkameramodell zur Abbildung eines Punktes auf die Bildebene . . . . .	7
2.2 Koordinatentransformation . . . . .	8
2.3 Aufnahme mit einer willkürlichen Kameraorientierung . . . . .	11
<b>3 Geometrische Beziehungen zwischen Punktekorrespondenzen</b>	<b>14</b>
3.1 Korrespondenzen planarer Punktmenge mit Homographien . . . . .	14
3.2 Korrespondenzanalyse für beliebige Punkte im Raum (Epipolare Geometrie) . . . . .	16
3.3 Bestimmung von Homographie und Fundamentalmatrix aus Punktekorrespondenzen .	19
<b>4 Synthetische Rekonstruktion</b>	<b>23</b>
4.1 Simulierte Bildaufnahme einer virtuellen Szene . . . . .	24
4.2 Bildanalyse . . . . .	26
4.2.1 Bestimmung der extrinsischen Kameraparameter . . . . .	26
4.2.2 Szenenrekonstruktion durch Triangulation . . . . .	28
4.3 Auswirkungen von unterschiedlichen Kameraauflösungen . . . . .	32
4.4 Geometrie eines Sensors . . . . .	32
4.5 Auswirkungen auf die Szenenrekonstruktion . . . . .	33
<b>5 Reelle Rekonstruktion</b>	<b>37</b>
5.1 Stereoaufbau . . . . .	38
5.2 Korrespondenzanalyse . . . . .	38
5.3 Normierter-Acht-Punkt-Algorithmus . . . . .	39
5.3.1 Singularität der Fundamentalmatrix . . . . .	41
5.3.2 Singulärwerte der essentiellen Matrix . . . . .	43
5.4 Szenenrekonstruktion mit Sampson-Approximation . . . . .	43
5.5 Ergebnisse einer Stereoanalyse mit Kameras unterschiedlicher Auflösung . . . . .	49
<b>6 Szenenrekonstruktion durch Rektifizierung</b>	<b>53</b>
6.1 Szenenrekonstruktion mit Rektifizierung . . . . .	53
6.2 Rektifizierung mit Homographien . . . . .	55
6.2.1 Projektive Transformation . . . . .	57
6.2.2 Ähnlichkeitstransformation . . . . .	61
6.2.3 Scherungstransformation . . . . .	63
6.3 Rektifizierung mit unterschiedlichen Kameraauflösungen . . . . .	65
<b>7 Punktesortierung in Schachbrettmustern</b>	<b>68</b>
7.1 Sortierungsalgorithmus . . . . .	69
7.2 Resultate bei stark verzerrten Schachbrettern . . . . .	76
<b>Zusammenfassung und Ausblick</b>	<b>79</b>
<b>Anhang</b>	<b>80</b>
<b>Eidesstattliche Erklärung</b>	<b>149</b>

<b>Abbildungsverzeichnis</b>	<b>150</b>
<b>Literaturverzeichnis</b>	<b>152</b>

# **Abstract**

Das Erkennen von dreidimensionalen Objekten ist Kern aktueller Forschungsbereiche wie dem autonomen Fahren, der Entwicklung von hochpräzisen medizinische Navigationssystemen oder der Landvermessung mittels Drohnen. Ziel ist es mit Hilfe weniger Kamerabilder einer Szene eine möglichst präzise, dreidimensionale Rekonstruktion dieser zu erhalten. Kamerabilder mit gleicher Auflösung werden in vielen Rekonstruktionsalgorithmen vorausgesetzt, obwohl bestimmte Maschinen wie Drohnen oft mit verschiedenen Kameras, wie zum Beispiel einer RGB-Kamera und einer Infrarotkamera mit unterschiedlichen Auflösungen, ausgestattet sind.

In dieser Arbeit wurde ein Szenenrekonstruktionsalgorithmen für stereoskopische Bildaufnahmen mit unterschiedlichen Kameraauflösungen implementiert und analysiert. Der entwickelte Algorithmus besitzt eine eingebaute Kamerakalibrierung, welche die extrinsischen Kameraparameter bestimmt und gemeinsam mit den zuvor bestimmten intrinsischen Kameraparametern eine genaue Rekonstruktion der Szene durchführt. Mit bekannten intrinsischen Kameraparameter, welche die Auflösung der Kameras berücksichtigt, kann der Algorithmus durch Triangulation eine dreidimensionale Szene bis zu einer Skaleninvarianz genau rekonstruieren. Der Algorithmus wurde an einem synthetischen Beispiel überprüft und erweitert um somit mit reellen Bilddaten eine Szene zu rekonstruieren. Der entstandene Algorithmus kann eine Szene bis auf eine Skaleninvarianz genau wiederherstellen. Jedoch ist die Rekonstruktion durch Triangulation anfällig auf Bildfehler und muss mit Näherungsverfahren korrigiert werden.

Viele kommerzielle Szenenrekonstruktions-Applikationen verwenden einen effizienteren Ansatz zur Szenenrekonstruktion. Dieser bestimmt die Bildtiefe näherungsweise durch Rektifizierung. Jedoch werden meist gleiche Kameraauflösungen vorausgesetzt. Aus diesem Grund wurden die ersten Schritte für einen zweiten Algorithmus basierend auf einem Rektifizierungsverfahren implementiert. In den ersten Analysen wurde die Funktionsweise der Rektifizierung überprüft und es konnte festgestellt werden, dass der implementierte Algorithmus für Bildquellen mit unterschiedlicher Auflösungen und gleicher Pixelproportionen geeignet ist.

In einem zusätzlichen Projekt dieser Masterarbeit wurde ein Ansatz zur Sortierung von detektierten Punkten in stereoskopischen Bildern von stark optisch verzerrten Schachbrettern entwickelt. Es wurde ein Algorithmus implementiert, welcher alle zuvor detektierten Eckpunkte eines Schachbrettes sortiert und eindeutig identifiziert. Der Algorithmus kann bei Stereoaufnahmen von zweidimensionalen Schachbrettern genutzt werden um Punktekorrespondenzen zu ermitteln. Der Algorithmus kann zudem auch für Kamerakalibrierungsalgorithmen zur Bestimmung von intrinsischen Kameraparametern verwendet werden um Bildverzerrungen zu korrigieren.

# 1 Einleitung

Die Computer Vision ist ein Fachbereich der Computer Science mit dem Fokus auf die Entwicklung von künstlicher Intelligenz, die ein visuelles Verständnis ihrer Umgebung besitzen. Folglich wird in der Computer Vision der Weg von visuellen Eindrücken oder Bildern aus der Realität in den Rechner beschrieben [1]. Der Mensch ist mit der Fähigkeit ausgestattet, gesehene Bilder zu verarbeiten und kann die ihn umgebene Welt verstehen. Maschinen, die eine ähnliche Fähigkeit besitzen, wären somit ebenfalls in der Lage Entscheidungen auf Grund von visuellen Eindrücken zu fällen. Das entwickeln solcher Maschinen und den damit verbundenen Grundprinzipien und Programmen sind die Forschungsmittelpunkte von aktuellen Anwendungsbereichen wie dem Autonomen Fahren, Motion-Capturing, Bewegungserkennungen oder Service Robotern.

In dieser Masterarbeit wurde ein Algorithmus zur Rekonstruktion einer Szene aus stereoskopischen Bildquellen entwickelt. Das typische Verfahren einer Stereorekonstruktion basiert auf den Grundbausteinen Bildaufnahme und Bildanalyse[1]. In der Bildaufnahme wird eine Szene oder ein Objekt mit Hilfe von Kameras, Sensoren oder Lasern aufgenommen und als digitale zweidimensionale Bilder an den Computer weitergegeben. In der Bildanalyse werden die aufgenommenen Bilder ausgewertet um so die dreidimensionale Szene rekonstruieren zu können. Für die Analyse ist es von Vorteil die Kameraparameter, wie Position und Auflösung, zu kennen. Sind diese jedoch nicht bekannt, können die Bildquellen genutzt werden um die Kameraparameter abzuschätzen. Eine solche Abschätzung wird als Kamerakalibrierung[2, 3, 4, 5] bezeichnet. Die Position und Rotation einer Kamera im Raum werden als die extrinsischen Kameraparameter bezeichnet. Parameter wie die Auflösung oder Brennweiten, werden als die intrinsischen Kameraparameter bezeichnet[2, 3]. Im Zuge dieser Arbeit ist ein Algorithmus entstanden, welcher im Stande ist die extrinsischen Kameraparameter bei Kameras gleicher und unterschiedlicher Auflösung zu bestimmen und eine 3D-Szenenrekonstruktion durchzuführen. Der in dieser Arbeit entwickelte Algorithmus wurde mithilfe eines synthetisch erstellten Beispiels verifiziert und auf eine reelle Szenenaufnahme angewandt. Beim Entwickeln von Algorithmen für Computer Vision Applikationen sieht man sich mit komplizierten Aufgaben und Herausforderungen konfrontiert. Bei der Aufnahme von Bildern, kann es zu unvorhersehbaren Bildfehlern wie beispielsweise Rauschen oder Verzerrungen durch die Kameralinse kommen, was nicht oft zum Verlust von Referenzdaten führt.

In der synthetischen Rekonstruktion wird zuerst eine 3D-Szene in zwei voneinander unterschiedlich positionierten, simulierten Kameras projiziert um virtuelle Bilddaten zu generieren. Anhand dieser 2D-Bilddaten wird die Kamerakalibrierung getestet. In der synthetischen Rekonstruktion werden die Werte für Auflösung und Brennweite selbst gesetzt. Im Test des Algorithmus mit reellen Bilddaten, wird für dessen Schätzung auf ein bereits existierendes Programm zurückgegriffen. Die intrinsischen Parameter werden mit dem in dieser Thesis entwickelten Algorithmus für die Schätzung der Positionen und Orientierungen der Kameras kombiniert, um die extrinsischen Kameraparameter anhand der synthetischen Daten zu schätzen. Die durch die Schätzung erhaltenen Kameraparameter können im synthetischen Beispiel mit den zuvor definierten Parametern verglichen werden, um den Algorithmus zu verifizieren. Diese Kameraparameter werden im entwickelten Rekonstruktionsalgorithmus dazu verwendet, die ursprüngliche 3D-Szene wieder herzustellen und die Funktionsweise der Rekonstruktion zu analysieren. Anschließend wird der Algorithmus an einem realen Stereobildpaar getestet und die entsprechenden Modifizierungen für die Arbeit mit realen fehleranfälligen Bilddaten genau aufgezeigt.

Ein zweiter Ansatz für einen Szenenrekonstruktionsalgorithmus aufgezeigt. Dieser ist auf eine schnelle und effiziente Rekonstruktion einer Szene ausgelegt und weniger auf die Bestimmung von Kameraparametern. Die Methode basiert auf einer zuvorigen Rektifizierung der Bilder, um Punktekorrespondenzen effizient zu ermitteln. Anhand dieser Punktekorrespondenzen kann eine Tiefenkarte der 3D-Szene er-

stellt werden. Auf diese Weise ist es möglich eine effektive Abschätzung für die Tiefe einer Szene zu bekommen. Die meisten kommerziellen Rekonstruktions-Applikationen setzen gleiche intrinsische Kameraparameter voraus. Um zu testen welche Auswirkungen eine Rektifizierung von Bildern unterschiedlicher Auflösungen hat, wird ein Rektifizierungsalgorithmus nach *Charles Loop & Zhengyou Zhang* implementiert. In Kapitel 6 wird der implementierte Algorithmus auf Abbildungen mit unterschiedlichen Auflösungen angewandt und die Auswirkungen unterschiedlicher Kameraauflösungen analysiert.

In Kapitel 7 wird ein weiterer Algorithmus vorgestellt. Dieser wurde implementiert um die detektierten Eckpunkte eines Schachbretts zu sortieren und eindeutig zu identifizieren und kann für die Korrespondenzanalyse bei Stereoaufnahmen von Schachbrettern mit starker optischen Verzerrung verwendet werden.

## 2 Model der Bildaufnahme mit einer Kamera

Um einen Szenenrekonstruktionalgorithmus zu verstehen werden in diesem Abschnitt grundlegende Bedingungen eingeführt um die Bildaufnahme mathematisch zu beschreiben. Ein abbildendes System besteht aus einem Objekt  $M$ , einer Kamera  $C$  und einer Bildebene  $I$  wie in Abbildung 2.1 dargestellt.

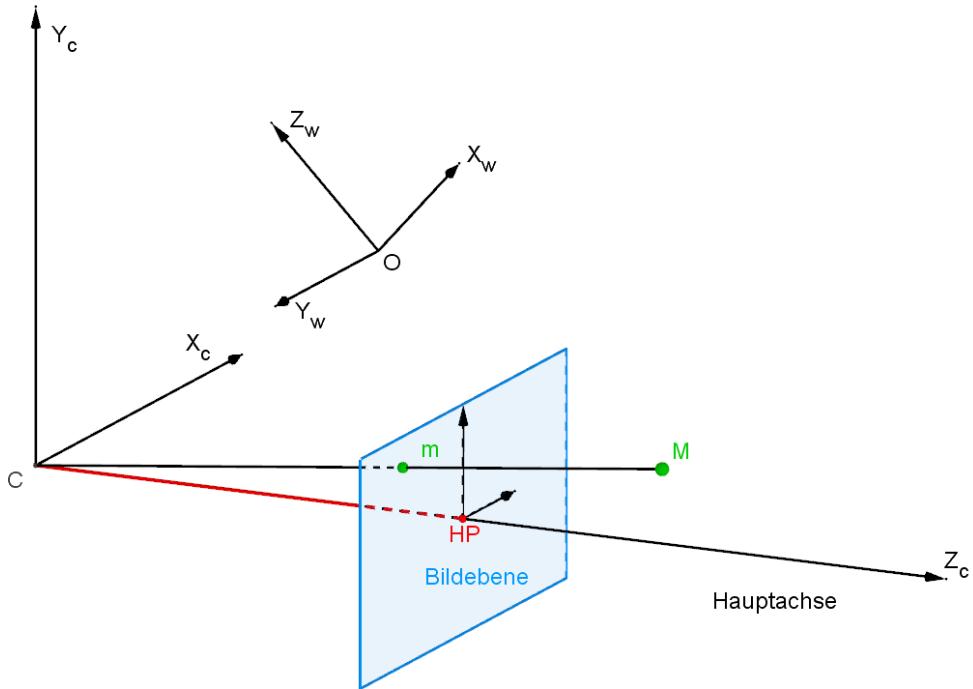


Abbildung 2.1: Schematik eines abbildenden Systems. Ein Punkt  $M$  im Weltkoordinatensystem  $O$  wird durch eine Kamera  $C$  aufgenommen. Diese Aufnahme wird durch eine Projektion, die als Verbindungslinie von  $M$  zu  $C$  zu sehen ist und  $M$  auf  $m$  abbildet, beschrieben.

Ein Punkt  $M$  in einem dreidimensionalen Weltkoordinatensystem wird mit Hilfe einer Kamera, die in einem eigenen dreidimensionalen Kamerakoordinatensystem beschrieben wird, auf die Bildebene  $I$  projiziert. Die Bildebene  $I$  ist durch ein zweidimensionales Bildkoordinatensystem beschrieben. Der projizierte Punkt  $m$  kann mit einem Sensor aufgenommen und abgespeichert werden.

Im Folgenden wird zuerst ein Kameramodell eingeführt um die Projektion auf die Bildebene zu beschreiben. Daraufhin werden Koordinatentransformationen beschrieben um abschließend die Aufnahme eines Punktes mit einer willkürlichen Kameraorientierung zu berechnen.

### 2.1 Lochkameramodell zur Abbildung eines Punktes auf die Bildebene

Mit Hilfe des Lochkameramodells wird die Abbildung eines Objektes auf eine Bildebene beschrieben. Das Modell beruht ausschließlich auf der geometrischen Optik und vernachlässigt physikalische Effekte, wie Beugung oder die Auswirkung der Linse[6]. Das Lochkameramodell besteht aus einem Projektionszentrum  $C$ .  $C$  beschreibt gleichzeitig die Lage des Kamerazentrums und bildet den Ursprung

des Kamerakoordinatensystems[7, 2]. Die Blickrichtung der Kamera wird als Hauptachse bezeichnet. Die Bildebene steht senkrecht zur Hauptachse. Der Schnittpunkt der Hauptachse mit der Bildebene bildet den Hauptpunkt  $HP$ . Der Hauptpunkt ist der Ursprung des Bildebenenkoordinatensystems. Der Abstand vom Projektionszentrum zum Hauptpunkt wird als Brennweite  $\zeta$  beschrieben[2, 7]. Der Bildpunkt  $m$  entsteht am Schnittpunkt der Verbindungsgerade von  $C$  und  $M$  mit der Bildebene  $I$ .

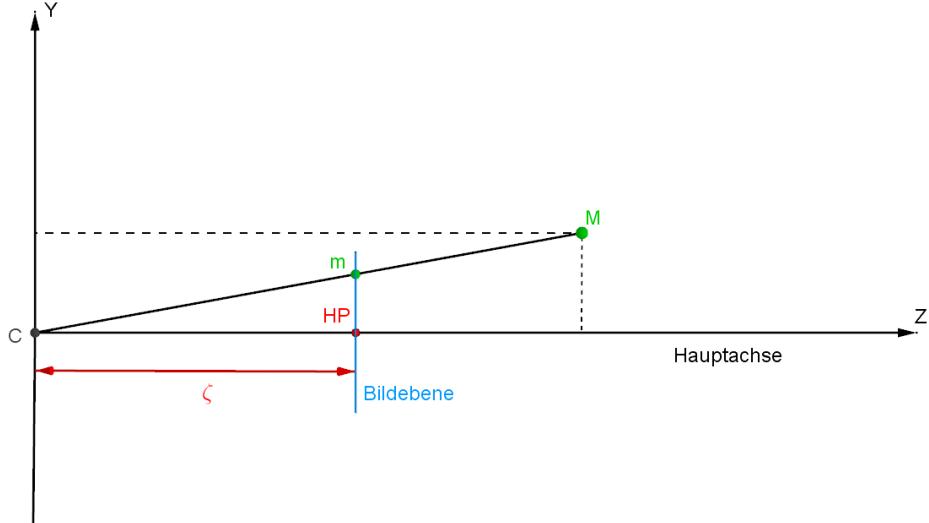


Abbildung 2.2: Die Abbildung zeigt einen Querschnitt des beschriebenen Lochkameramodells. Zu sehen ist das Projektionszentrum  $C$  der Kamera.  $C$  ist gleichzeitig das Kamerazentrum und bildet den Ursprung für das Kamerakoordinatensystem.  $\zeta$  beschreibt den Abstand des Projektionszentrums zur Bildebene. Die Hauptachse beschreibt die Blickrichtung der Kamera. Der Punkt an dem die Hauptachse die Bildebene schneidet wird Hauptpunkt genannt und ist gleichzeitig der Ursprung für das Bildebenenkoordinatensystem. Der Bildpunkt  $m$  entsteht am Schnittpunkt der Verbindungsgerade von  $C$  und  $M$  mit der Bildebene  $I$

Die Projektion eines dreidimensionalen Punktes  $M_\delta$  auf eine zweidimensionale Bildebene, wird durch eine  $3 \times 3$  Kameramatrix  $K_0$  beschrieben.

$$K_0 \cdot M_\delta = \begin{bmatrix} \zeta & 0 & 0 \\ 0 & \zeta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{pmatrix} \zeta X \\ \zeta Y \\ Z \end{pmatrix} \rightarrow \begin{pmatrix} \zeta \frac{X}{Z} \\ \zeta \frac{Y}{Z} \\ 1 \end{pmatrix} \quad (2.1)$$

Die Koordinaten auf der zweidimensionalen Bildebene werden häufig als homogene Koordinaten angegeben. Dazu werden die Koordinaten mit  $Z$  normiert und somit auf die Ebene  $(x, y, 1)^T$  projiziert wird. Zur Vereinfachung wird zuletzt nur die  $x, y$  Koordinaten des entstandenen Bildes angegeben. Gleichung 2.1 beschreibt somit die Abbildung eines Punktes auf die Bildebene.

## 2.2 Koordinatentransformation

Um einen Punkt von einem übergeordneten Weltkoordinatensystem in ein bestimmtes, zum Weltkoordinatensystem rotiertes, Kamerakoordinatensystem zu überführen ist eine Transformation notwendig. Im Folgenden wird der mathematische Weg einer Transformation eines Weltkoordinatensystems  $(O, \delta)$  mit  $\delta = (\hat{d}_1, \hat{d}_2, \hat{d}_3, O)$  in ein Kamerakoordinatensystem  $(C, \beta)$  mit  $\beta = (\hat{b}_1, \hat{b}_2, \hat{b}_3, C)$  beschrieben. Zunächst wird eine Koordinatisierung von Punkten im Weltkoordinatensystem vorgenommen. Ein Punkt  $P_\delta$  bezüglich des Weltkoordinatensystems wird wie folgt beschrieben

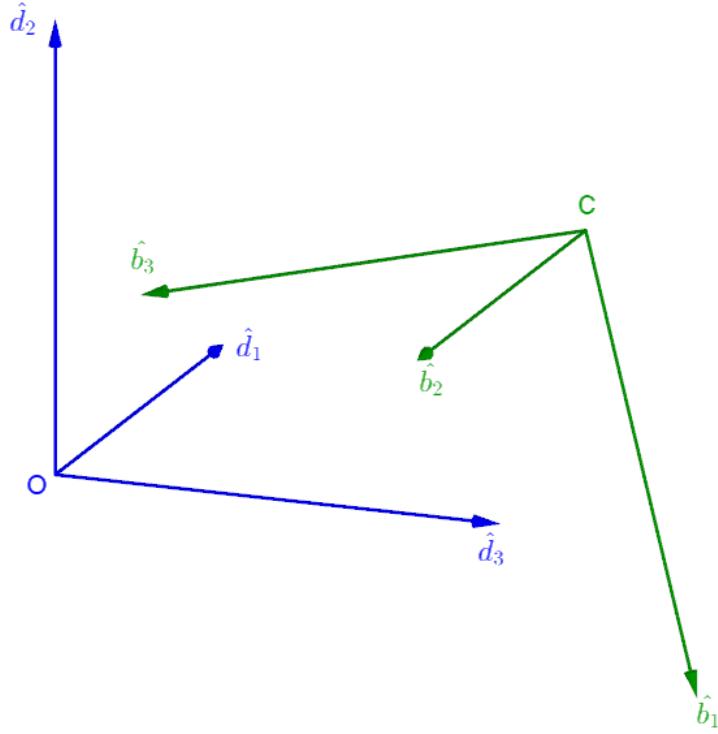


Abbildung 2.3: Ein Weltkoordinatensystem  $(O, \delta)$  mit  $\delta = (\hat{d}_1, \hat{d}_2, \hat{d}_3, O)$  wird zu einem dazu verschobenen und rotiertem Kamerakoordinatensystem  $(C, \beta)$  mit  $\beta = (\hat{b}_1, \hat{b}_2, \hat{b}_3, C)$  transformiert.

$$P_\delta = O + p_{\delta x} \hat{d}_1 + p_{\delta y} \hat{d}_2 + p_{\delta z} \hat{d}_3 \quad (2.2)$$

$$\rightsquigarrow P_\delta = (p_{\delta x}, p_{\delta y}, p_{\delta z})^T = \begin{pmatrix} p_{\delta x} \\ p_{\delta y} \\ p_{\delta z} \end{pmatrix}. \quad (2.3)$$

Zwischen den beiden Koordinatensystemen  $(O, \delta)$  und  $(C, \beta)$  gelten die folgenden Beziehungen

$$C_\beta = O_\delta + C_{\beta,1} \hat{d}_1 + C_{\beta,2} \hat{d}_2 + C_{\beta,3} \hat{d}_3 \quad (2.4)$$

$$\hat{b}_1 = b_{11} \hat{d}_1 + b_{12} \hat{d}_2 + b_{13} \hat{d}_3 \quad (2.5)$$

$$\hat{b}_2 = b_{21} \hat{d}_1 + b_{22} \hat{d}_2 + b_{23} \hat{d}_3 \quad (2.6)$$

$$\hat{b}_3 = b_{31} \hat{d}_1 + b_{32} \hat{d}_2 + b_{33} \hat{d}_3. \quad (2.7)$$

Diese Beziehungsgleichungen werden in Gleichung 2.2 eingesetzt.

$$\begin{aligned} P_\delta &= O + (C_{\beta,1} + p_{\beta x} b_{11} + p_{\beta y} b_{21} + p_{\beta z} b_{31}) \cdot \hat{d}_1 \\ &\quad + (C_{\beta,2} + p_{\beta x} b_{12} + p_{\beta y} b_{22} + p_{\beta z} b_{32}) \cdot \hat{d}_2 \\ &\quad + (C_{\beta,3} + p_{\beta x} b_{13} + p_{\beta y} b_{23} + p_{\beta z} b_{33}) \cdot \hat{d}_3. \end{aligned} \quad (2.8)$$

Aus Gleichung 2.8 wird ein Gleichungssystem in der Form von Gleichung 2.9 aufgestellt und gelöst.

$$\begin{aligned} p_{\delta x} &= C_{\beta,1} + (C_{\beta,1} + p_{\beta x} b_{11} + p_{\beta y} b_{21} + p_{\beta z} b_{31}) \\ \rightsquigarrow p_{\delta x} - C_{\beta,1} &= (C_{\beta,1} + p_{\beta x} b_{11} + p_{\beta y} b_{21} + p_{\beta z} b_{31}). \end{aligned} \quad (2.9)$$

Das Gleichungssystem lässt sich in Matrixform darstellen als

$$\begin{bmatrix} b_{11} & b_{21} & b_{31} \\ b_{12} & b_{22} & b_{32} \\ b_{13} & b_{23} & b_{33} \end{bmatrix} \begin{pmatrix} p_{\beta x} \\ p_{\beta y} \\ p_{\beta z} \end{pmatrix} = \begin{pmatrix} p_{\delta x} - C_{\beta,1} \\ p_{\delta y} - C_{\beta,2} \\ p_{\delta z} - C_{\beta,3} \end{pmatrix}. \quad (2.10)$$

Wenn  $P_\beta$  gegeben ist, erhält man auf diese Weise direkt  $P_\delta$ . Die inverse Matrix  $D_\beta^{-1}$  kann verwendet werden um  $P_\beta$  aus  $P_\delta$  zu berechnen.

$$D_\beta^{-1} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \quad (2.11)$$

$$\rightsquigarrow \begin{pmatrix} p_{\beta x} \\ p_{\beta y} \\ p_{\beta z} \end{pmatrix} = D_\beta^{-1} \begin{pmatrix} p_{\delta x} - C_{\beta,1} \\ p_{\delta y} - C_{\beta,2} \\ p_{\delta z} - C_{\beta,3} \end{pmatrix}. \quad (2.12)$$

Handelt es sich um ein kartesisches Koordinatensystem, so gilt  $D_\beta^{-1} = D_\beta^T$  und die transponierte Matrix kann für die Koordinatentransformation benutzt werden. Für zwei normierte, kartesische Koordinatensysteme ist  $D$  und  $D^T$  eine Rotationsmatrix  $R$ , weshalb im Folgenden, analog zur Literatur [2, 4, 3],  $D^T = R$  angenommen wird. Um Gleichung 2.12 in einer kompakten Schreibweise zu formulieren, wird  $\vec{p}_\beta = (p_{\beta x}, p_{\beta y}, p_{\beta z})^T$  zu einem vierdimensionalen Vektor mit 1 zu  $\vec{p}_\beta = (p_{\beta x}, p_{\beta y}, p_{\beta z}, 1)^T = (\vec{p}_\beta, 1)^T$  erweitert. Damit lässt sich Gleichung 2.12 als eine Matrixmultiplikation ausdrücken

$$\begin{pmatrix} p_{\beta x} \\ p_{\beta y} \\ p_{\beta z} \\ 1 \end{pmatrix} = D_\beta^T \begin{bmatrix} 1 & 0 & 0 & -C_{\beta,1} \\ 0 & 1 & 0 & -C_{\beta,2} \\ 0 & 0 & 1 & -C_{\beta,3} \end{bmatrix} \begin{pmatrix} p_{\delta x} \\ p_{\delta y} \\ p_{\delta z} \\ 1 \end{pmatrix} = R[I - C] \begin{pmatrix} p_{\delta x} \\ p_{\delta y} \\ p_{\delta z} \\ 1 \end{pmatrix} = T \begin{pmatrix} \vec{p}_\delta \\ 1 \end{pmatrix}. \quad (2.13)$$

Die Transformationsmatrix  $T$  setzt sich aus der Rotationsmatrix  $R$  und der Translationsmatrix  $[I| - C]$  zusammen und wirkt auf den neu definierten vierdimensionalen Vektor. Wichtig dabei ist, dass  $[I| - C]$  eine symbolische Schreibweise für eine  $3 \times 4$  Matrix ist.

## 2.3 Aufnahme mit einer willkürlichen Kameraorientierung

Ein beliebiger Punkt im Weltkoordinatensystem kann mit der eingeführten Operation auf die Bildebene und schließlich auch auf den Sensor projiziert werden. Es werden insgesamt vier verschiedene Koordinatensysteme definiert. Das Weltkoordinatensystem  $(O, \delta)$  mit  $\delta = (\hat{d}_1, \hat{d}_2, \hat{d}_3)$ , das Kamerakoordinatensystem  $(C, \beta)$  mit  $\beta = (\hat{b}_1, \hat{b}_2, \hat{b}_3)$ , das Bildebenenkoordinatensystem  $(I, \tau)$  mit  $\tau = (\hat{t}_1, \hat{t}_2)$  und als letztes das Sensorkoordinatensystem mit  $(S, \sigma)$  mit  $\sigma = (\hat{u}, \hat{v})$ . Abbildung 2.4 zeigt die Koordinatensysteme schematisch im Überblick.

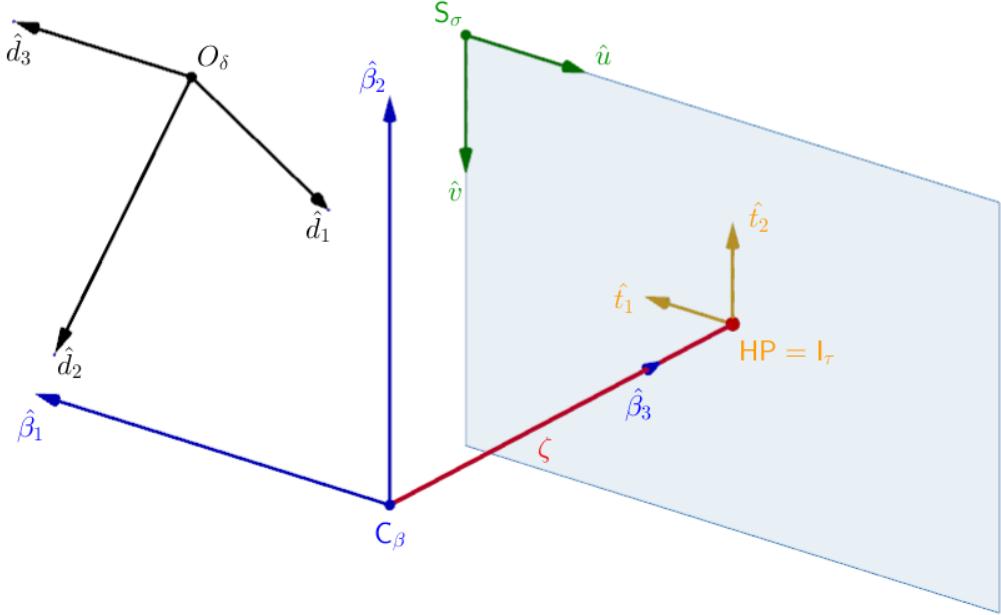


Abbildung 2.4: Das Schaubild zeigt die einzelnen Koordinatensysteme in einem Lochkameramodell. Das Weltkoordinatensystem  $(O, \delta)$  mit  $\delta = (\hat{d}_1, \hat{d}_2, \hat{d}_3)$ , das Kamerakoordinatensystem  $(C, \beta)$  mit  $\beta = (\hat{b}_1, \hat{b}_2, \hat{b}_3)$ , das Bildebenenkoordinatensystem  $(I, \tau)$  mit  $\tau = (\hat{t}_1, \hat{t}_2)$  und das Sensorkoordinatensystem  $(S, \sigma)$  mit  $\sigma = (\hat{u}, \hat{v})$ .

Für die Projektion eines Punktes  $M_\delta = (M_{\delta x} M_{\delta y} M_{\delta z})^T$  bezüglich des Weltkoordinatensystems in einen Punkt  $m_\tau = (m_{\tau x} m_{\tau y} m_{\tau z})^T$  bezüglich des Bildebenenkoordinatensystems kann eine Projektionsmatrix  $P$  definiert werden.

Zuerst muss der Punkt im Weltkoordinatensystem in das Kamerakoordinatensystem transformiert werden. Für die Transformation der Weltkoordinaten in Kamerakoordinaten gilt die Gleichung 2.13

$$\overrightarrow{M_\beta} = T \begin{bmatrix} \overrightarrow{M_\delta} \\ 1 \end{bmatrix} = R \begin{bmatrix} 1 & 0 & 0 & -C_{\beta x} \\ 0 & 1 & 0 & -C_{\beta y} \\ 0 & 0 & 1 & -C_{\beta z} \end{bmatrix} \begin{bmatrix} M_{\delta x} \\ M_{\delta y} \\ M_{\delta z} \\ 1 \end{bmatrix} = R[I - C] \begin{bmatrix} \overrightarrow{M_\delta} \\ 1 \end{bmatrix}. \quad (2.14)$$

Nach der Transformation eines Punkts  $\overrightarrow{M_\delta}$  zu  $\overrightarrow{M_\beta}$  in das Kamerakoordinatensystem erfolgt die Kameraprojektion von  $M_\beta$  auf  $m_\beta$  wie in Gleichung 2.1 beschrieben.

$$\begin{bmatrix} m_{\beta x} \\ m_{\beta y} \\ m_{\beta z} \end{bmatrix} = \begin{bmatrix} \zeta & 0 & 0 \\ 0 & \zeta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M_{\beta x} \\ M_{\beta y} \\ M_{\beta z} \end{bmatrix} = K_0 \overrightarrow{M_\beta}. \quad (2.15)$$

Die Projektion eines Punktes  $\vec{M}_\delta$  auf den Bildpunkt  $\vec{m}_\beta$  kann durch Gleichung 2.14 und 2.15 zusammengefasst werden

$$\vec{m}_\beta = K_0 R [I - C] \begin{bmatrix} \vec{M}_\delta \\ 1 \end{bmatrix}. \quad (2.16)$$

Um den Bildpunkt  $\vec{m}_\beta$  bezüglich eines zweidimensionalen Bildebenenkoordinatensystems anzugeben, wird die Bildebene mit der Tiefkomponente  $m_{\beta z}$  normiert, sodass  $m_{\beta z}$  auf den zweidimensionalen Raum der Bildebene abgebildet wird. Diese Projektion wird durch folgende Gleichung beschrieben

$$\vec{m}_\tau = \begin{bmatrix} m_{\beta x} / m_{\beta z} \\ m_{\beta y} / m_{\beta z} \\ 1 \end{bmatrix}. \quad (2.17)$$

Zuletzt folgt die Transformation der Bildebenenkoordinaten auf den Sensorchip. Der Sensorchip besteht aus einer Ansammlung von Sensorelementen. Diese Sensorelemente können verschiedene Formen annehmen. Die meisten Sensorchips bestehen aus rechtwinkligen, rechteckigen Sensorelementen. Aus diesem Grund wird ein rechtwinkliges Sensorelement mit einer Größe  $lx ly$  angenommen. Diese Sensorelementgröße  $lx ly$  definiert auch die Pixelgröße und bildet die Längenskalierung des Sensorkoordinatensystems. Neben der unterschiedlichen Skalierung wird der Ursprung des Sensorkoordinatensystems in der Regel an einer Ecke des Sensorchips definiert, sodass die Transformation von Bildebenenkoordinaten in Sensorkoordinaten auch eine Translation  $(V_{\sigma x}, V_{\sigma y})$  aufweist[2, 8]. Für einen Punkt  $m_\sigma = (u, v, 1)$  auf dem Sensorkoordinatensystem lassen sich die folgenden Bedingungen herleiten

$$u = m_{\tau x} k_x - V_{\sigma x} \quad (2.18)$$

$$v = m_{\tau y} k_y - V_{\sigma y} \quad (2.19)$$

$$1 = 1. \quad (2.20)$$

$k_x = 1/lx$  und  $k_y = 1/ly$  ist die Pixeldichte in  $\frac{\text{pixel}}{\text{mm}}$ . Es wird angemerkt, dass in der Bildebene und der Sensorebene die Punkte ausschließlich im zweidimensionalen Raum definiert sind. Aus den normierten Bildkoordinaten lässt sich somit folgende Sensormatrix bilden

$$\vec{m}_\sigma = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & V_{\sigma x} \\ 0 & k_y & V_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_{\tau x} \\ m_{\tau y} \\ 1 \end{bmatrix} = R_\sigma \vec{m}_\tau. \quad (2.21)$$

Mit Hilfe der Transformationsmatrix  $R_\sigma$  kann ein Punkt von der Bildebene auf das Sensorelement projiziert werden.

Der hier skizzierte Lösungsweg beschreibt die Bildaufnahme eines Punktes im Lochkameramodel. Die hier eingeführte Projektionsmatrix  $P = K_0 R [I - C]$  gilt für die Abbildung eines Punktes auf den Bildpunkt. Der Bildpunkt wird normiert und in das Sensorkoordinatensystem umgerechnet. In der Literatur wird häufig die Transformation in das Sensorkoordinatensystem bereits in der Kameramatrix zusammengefasst. Damit bildet sich die erweiterte Kameramatrix  $K$

$$K = R_\sigma K_0 = \begin{bmatrix} k_x \zeta & 0 & V_{\sigma x} \\ 0 & k_y \zeta & V_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.22)$$

Mit dieser Kameramatrix wird eine neue Projektionsmatrix mit  $P = KR[I - C]$  gebildet, die einen Objektpunkt auf einen Bildpunkt im Sensorkoordinatensystem abbildet. Durch die Normierung dieses Punktes kann auf den direkten Sensorpunkt geschlossen werden. Ein weiterer Vorteil der erweiterten Kameramatrix  $K$  ist, dass sie die Pixeldichte  $k_x, k_y$  und die Brennweite  $\zeta$  beinhaltet. Diese Parameter

werden im Folgenden als intrinsische Kameraparameter bezeichnet. Die Koordinatentransformationsmatrix  $R$  wird im Gegensatz aus den sogenannten extrinsischen Kameraparameter, der Kameraposition und Orientierung, definiert. Sind sowohl die intrinsischen wie auch extrinsischen Kameraparameter vorbestimmt kann  $P$  bestimmt und mit dem hier beschriebenen Lösungsweg das Bild konstruiert werden.

# 3 Geometrische Beziehungen zwischen Punktekorrespondenzen

Das Ziel dieser Masterarbeit ist eine 3D Szene im dreidimensionalen Raum aus einer stereoskopischen Aufnahme zweier Kameras zu rekonstruieren. Bisher wurde die Bildaufnahme einer einzigen Kamera betrachtet. Jedoch kann eine Kamera allein nicht räumlich sehen. Um dreidimensionale Szenen aus Bildern zu rekonstruieren, müssen mindestens zwei Aufnahmen der gleichen Szene aus unterschiedlichen Blickwinkeln aufgenommen werden. Innerhalb dieser Aufnahmen müssen Punktekorrespondenzen gesucht werden. Korrespondierende Punkte zeichnen sich dadurch aus, dass sie die Abbildungen desselben Ursprungspunktes im Raum sind. Für diese Punkte muss eine gemeinsame Abbildungsvorschrift aufgestellt werden. Die Abbildungsvorschrift wird in einer  $3 \times 3$ -Matrix  $H$  ausgedrückt, welche die Transformationsmatrizen sowie die Kameramatrizen zusammenfasst. Die  $3 \times 3$ -Matrix, kann aus gegebenen Punktekorrespondenzen abgeleitet werden. Aus  $H$  können Rückschlüsse auf die Kamera-parameter der beiden Kameras gezogen werden.

Es seien  $m_\tau = (m_{\tau x}, m_{\tau y}, m_{\tau z})^T$  die homogenen Koordinaten eines Punktes auf der Bildebene  $(I, \tau)$  und  $m'_{\tau'} = (m'_{\tau' x}, m'_{\tau' y}, m'_{\tau' z})^T$  der dazu korrespondierende Punkt der Bildebene  $(I', \tau')$ , vergleiche hierzu Abbildung 3.1. Gesucht wird eine Abbildungsvorschrift welche als Matrix  $H$  ausgedrückt wird

$$m'_{\tau'} = Hm_\tau \quad (3.1)$$

$$Hm_\tau = \begin{bmatrix} h_1^T \cdot m_{\tau x} \\ h_2^T \cdot m_{\tau y} \\ h_3^T \cdot m_{\tau z} \end{bmatrix} \quad (3.2)$$

$$\rightsquigarrow m'_{\tau'} = Hm_\tau = \begin{bmatrix} h_{11}m_{\tau x} + h_{12}m_{\tau y} + h_{13}m_{\tau z} \\ h_{21}m_{\tau x} + h_{22}m_{\tau y} + h_{23}m_{\tau z} \\ h_{31}m_{\tau x} + h_{32}m_{\tau y} + h_{33}m_{\tau z} \end{bmatrix} \quad (3.3)$$

$$\rightsquigarrow H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \quad (3.4)$$

In den folgenden Unterkapiteln werden zwei Herleitungen der Abbildungsvorschriften zwei unterschiedlicher Fälle gesucht. Im ersten Fall wird vorausgesetzt, dass die 3D-Punkte im Raum auf einer Ebene liegen und auf die Bildebenen von zwei zueinander verschobenen und rotierten Kameras abgebildet werden. Im zweiten Fall werden die Punkte eines komplexeren 3D-Objektes auf die beiden Bildebenen abgebildet. Im letzten Abschnitt wird die Herleitungen der entstehenden Matrizen beider Fälle anhand von Punktekorrespondenzen aufgezeigt.

## 3.1 Korrespondenzen planarer Punktmengen mit Homographien

Eine Abbildungsvorschrift kann in bestimmten Fällen eindeutig bestimmt werden. In diesen Fällen nennt man die Abbildung zwischen beiden zweidimensionalen Bildern Homographie[2, 4, 9]. In diesem Kapitel wird der beispielhafte Fall behandelt, dass Punkte auf der  $x, y$ -Ebene im Weltkoordinatensystem auf zwei unterschiedlichen Kameras  $C$  und  $C'$  abgebildet werden. Dies ist in Abbildung 3.1 dargestellt.

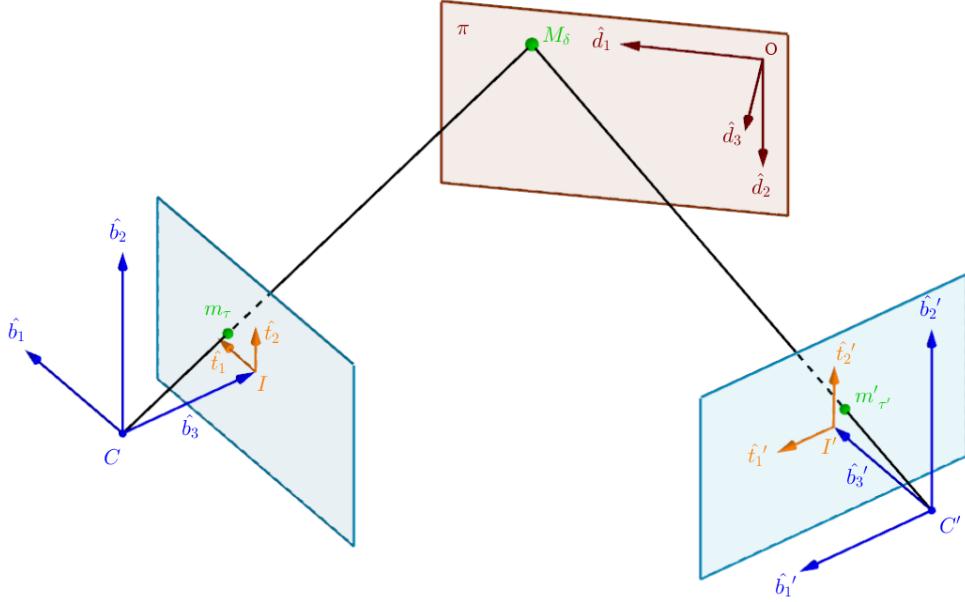


Abbildung 3.1: In der Abbildung sind die beiden Kameras  $C$  und  $C'$  mit ihren Bildebenen  $I$  und  $I'$  zu sehen. Ein Objektpunkt  $M_\delta$  bezüglich eines Weltkoordinatensystems  $(O, \delta)$  befindet sich auf der Ebene  $\pi$ , welche durch die Achsen  $\hat{d}_1$  und  $\hat{d}_2$  aufgespannt wird.  $M_\delta$  wird auf  $I$  und  $I'$  projiziert. Es entstehen die Bildpunkte  $m_\tau$  und  $m'_\tau$ .

Um die Abbildungsvorschrift herzuleiten wird bei Bildpunkt  $m_\tau = (m_{\tau x}, m_{\tau y}, m_{\tau z})^T$  mit  $m_{\tau z} = 1$  begonnen. Während in der Bildaufnahme ein Punkt  $m_\beta$  durch Division mit der  $m_{\beta, z}$ -Komponenten eindeutig zu  $m_\tau$  wird ist die Rückrichtung nicht eindeutig. Alle Punkte auf den Geraden von  $C$  und  $m_\tau$ , siehe Abbildung 3.2, werden auf denselben Bildpunkt projiziert. Die Projektion von  $m_\tau$  auf  $m_\beta$  ist demnach nicht eindeutig. Die Gerade mit allen möglichen Punkten  $m_\beta$  kann als  $\gamma m_\tau$  mit der freien Variablen  $\gamma > 0$  ausgedrückt werden.

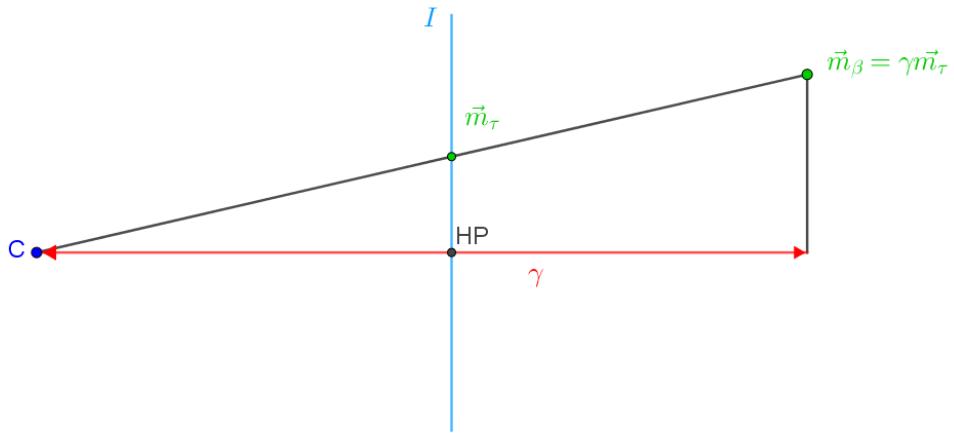


Abbildung 3.2: Auf der Geraden durch  $C$  und  $\vec{m}_\tau$ , befinden sich alle möglichen Punkte für  $m_\beta$ .  $m_\beta$  wird auf Grund seiner Unbestimmtheit als  $\gamma m_\tau$  bezeichnet

Für zwei korrespondierende Bildpunkte  $m_\tau$  und  $m'_\tau$ , kann für alle möglichen Punkte  $m_\beta$  und  $m'_\beta$  die folgende Projektionsvorschrift hergeleitet werden [4].

$$\gamma \vec{m}_\tau = P \begin{bmatrix} \vec{M}_\delta \\ 1 \end{bmatrix} = [KR| - KR\vec{C}_\delta] \cdot \begin{bmatrix} \vec{M}_\delta \\ 1 \end{bmatrix} = KR(\vec{M}_\delta - \vec{C}_\delta) \quad (3.5)$$

$$\gamma' \vec{m}'_{\tau'} = P' \begin{bmatrix} \vec{M}'_\delta \\ 1 \end{bmatrix} = [K'R'| - K'R'\vec{C}'_\delta] \cdot \begin{bmatrix} \vec{M}'_\delta \\ 1 \end{bmatrix} = K'R'(\vec{M}'_\delta - \vec{C}'_\delta). \quad (3.6)$$

Mit einem Ursprungspunkt  $M_\delta = (x_\delta, y_\delta, 0)^T$  auf der  $x, y$ -Ebene im Weltkoordinatensystem und einer unbekannten Projektionsmatrix  $P$  mit

$$P = \begin{bmatrix} p_{11} & p_{21} & p_{31} & p_{41} \\ p_{12} & p_{22} & p_{32} & p_{42} \\ p_{13} & p_{23} & p_{33} & p_{43} \end{bmatrix} = [p_1 \ p_2 \ p_3 \ p_4], \quad (3.7)$$

kann die folgende Gleichung aufgestellt werden [4]

$$\gamma m_\tau = P \cdot \begin{bmatrix} M_\delta \\ 1 \end{bmatrix} = [p_1 \ p_2 \ p_3 \ p_4] \cdot \begin{bmatrix} x_\delta \\ y_\delta \\ 0 \\ 1 \end{bmatrix} = [p_1 \ p_2 \ p_4] \cdot \begin{bmatrix} x_\delta \\ y_\delta \\ 1 \end{bmatrix} = G \vec{m}_\delta \quad (3.8)$$

$$\gamma' m'_{\tau'} = P \cdot \begin{bmatrix} M'_\delta \\ 1 \end{bmatrix} = [p'_1 \ p'_2 \ p'_3 \ p'_4] \cdot \begin{bmatrix} x_\delta \\ y_\delta \\ 0 \\ 1 \end{bmatrix} = [p'_1 \ p'_2 \ p'_4] \cdot \begin{bmatrix} x_\delta \\ y_\delta \\ 1 \end{bmatrix} = G' \vec{m}_\delta. \quad (3.9)$$

Aus  $\gamma m_\tau = G \cdot m_\delta$  und  $\gamma' m'_{\tau'} = G' \cdot m_\delta$  kann dann Folgendes abgeleitet werden[4]

$$\gamma' m'_{\tau'} = G' G^{-1} \gamma m_\tau. \quad (3.10)$$

Mit  $\lambda = \frac{\gamma'}{\gamma}$ , kann Gleichung 3.10 dann wieder umformuliert und in die Bedienungsgleichung der Homographie mit  $H = G' G^{-1}$  umgeformt werden

$$\lambda m'_{\tau'} = H m_\tau. \quad (3.11)$$

Die entstandene Homographie  $H$  ist somit eine Abbildungsvorschrift welche zwei korrespondierende Punkte in Verbindung setzt. Diese Homographiebedingung stellt ein Gleichungssystem mit neun unbekannten, welche in Kapitel 3.3 gelöst wird[2].

## 3.2 Korrespondenzanalyse für beliebige Punkte im Raum (Epipolare Geometrie)

Für Bilder von komplexeren, dreidimensionalen Objekten, bei denen die Punkte auf verschiedenen Ebenen im Raum liegen können, kann keine Homographiebedingung hergestellt werden um die Kameraparameter zu bestimmen. Jedoch kann auf geometrische Bedingungen zurückgegriffen werden, um die Abbildungsvorschrift zwischen den Bildern auszunutzen und die Kameraparameter beider Kameras zu bestimmen. Ein Ursprungspunkt  $M_\delta$  wird wieder mit zwei Kameras  $C$  und  $C'$  aufgenommen. In Abbildung 3.3 ist das stereoskopische System dargestellt.

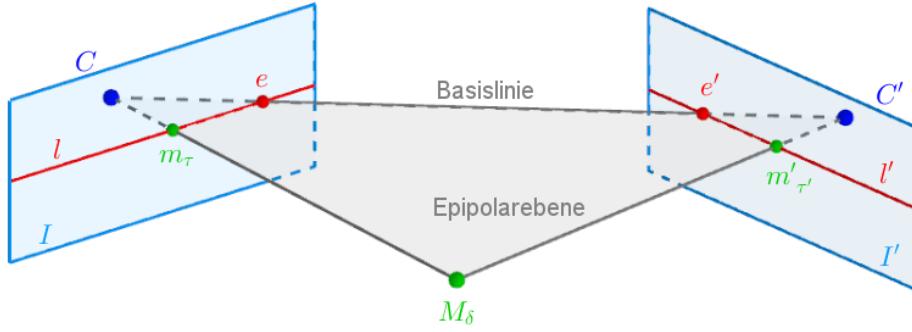


Abbildung 3.3:  $C$  und  $C'$  sind die Projektionszentren zweier Kameras. Beide Kameras besitzen jeweils eine Bildebene. Die Basislinie verbindet die Projektionszentren der Kameras. Die Punkte an welchen die Basislinie die Bildebenen schneidet werden als Epipole  $e$  und  $e'$  bezeichnet. Durch einen Epipol verlaufen alle Epipolarlinien des Bildes.  $M_\delta$  ist der Objektpunkt im 3D-Raum und  $m_\tau$  und  $m'_{\tau'}$  sind die jeweiligen Abbildungen dieses Punktes auf den Bildebenen. Die Verbindungsvektoren zwischen  $C, C'$  und  $M_\delta$  bilden die sogenannte Epipolarebene[10, 11, 2, 5].

Es werden hier einige geometrische Definitionen eingeführt um die danach folgende mathematische Herleitung genauer zu verstehen. Die Vektoren  $\overrightarrow{CM} = (\vec{M}_\delta - \vec{C}_\delta)$ ,  $\overrightarrow{C'M} = (\vec{M}_\delta - \vec{C}'_\delta)$  und  $\overrightarrow{CC'} = (\vec{C}'_\delta - \vec{C}_\delta)$  definieren die Epipolarebene, die durch das schwarze Dreieck in Abbildung 3.3 gekennzeichnet ist. Die Schnittpunkte der Geraden zwischen  $C$  und  $C'$  mit der jeweiligen Bildebene  $I$  und  $I'$  werden als Epipole  $e$  und  $e'$  bezeichnet. Die Schnittgerade der Epipolarebene mit  $I$  und  $I'$  bilden die sogenannten Epipolarlinien  $l$  und  $l'$ [2, 12, 13, 11, 4].

Ein Bildpunkt  $m_i$  auf der Bildebene  $I$  wird zuerst auf die Gerade, die durch  $m_i$  und  $C$  geht abgebildet. Die Gerade stellt alle möglichen Ursprungspunkte zu  $m_i$  dar. Dies ist durch die drei möglichen Punkte  $M_1, M_2, M_3$  in Abbildung 3.4 dargestellt. Jeder dieser Punkte wird nun wiederum auf  $I'$  projiziert. Die so entstandenen Punkte liegen alle auf der Epipolarlinie  $l'$ [2].

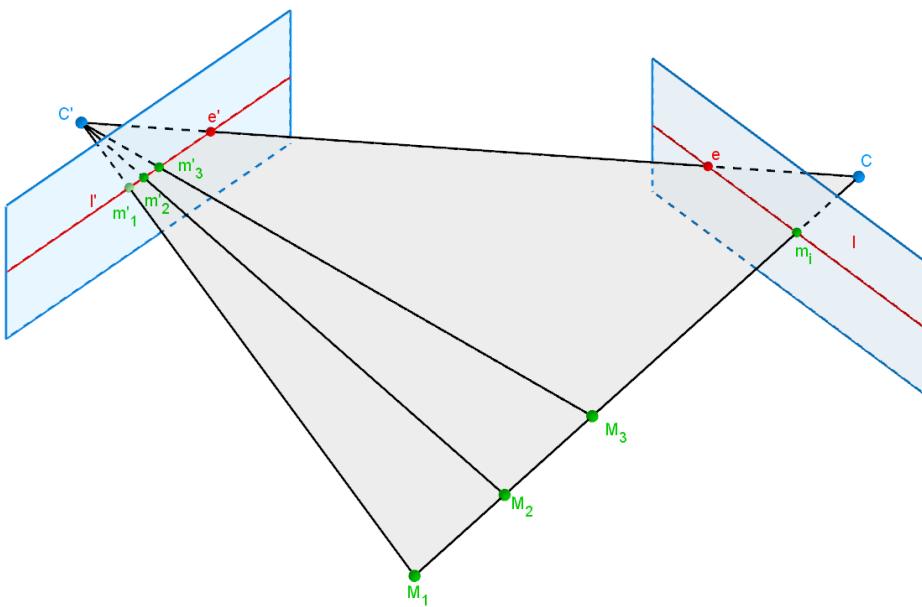


Abbildung 3.4: Die Objektpunkte  $M_1, M_2$  und  $M_3$  werden in  $I'$  als  $m'_1, m'_2$  und  $m'_3$  abgebildet, während sie in  $I$  immer den selben Bildpunkt  $m_1$  ergeben.

Die hier gezeigte Abbildung von  $m_i$  auf  $l'$  wird nun genauer betrachtet. Es werden wieder die Gleichungen für  $m_\tau$  und  $m'_{\tau'}$ , wie in Gleichung 3.6, aufgestellt

$$\gamma \vec{m}_\tau = P \begin{bmatrix} \vec{M}_\delta \\ 1 \end{bmatrix} = [KR| - KR \vec{C}_\delta] \cdot \begin{bmatrix} \vec{M}_\delta \\ 1 \end{bmatrix} = KR(\vec{M}_\delta - \vec{C}_\delta) \quad (3.12)$$

$$\gamma' \vec{m}'_{\tau'} = P' \begin{bmatrix} \vec{M}'_\delta \\ 1 \end{bmatrix} = [K'R'| - K'R' \vec{C}'_\delta] \cdot \begin{bmatrix} \vec{M}'_\delta \\ 1 \end{bmatrix} = K'R'(\vec{M}'_\delta - \vec{C}'_\delta). \quad (3.13)$$

Die Gleichungen 3.12 und 3.13 werden nach  $(\vec{M}_\delta - \vec{C}_\delta)$  und  $(\vec{M}'_\delta - \vec{C}'_\delta)$  aufgelöst.

$$\gamma R^T K^{-1} \vec{m}_\tau = (\vec{M}_\delta - \vec{C}_\delta) \quad (3.14)$$

$$\gamma' R'^T K'^{-1} \vec{m}'_{\tau'} = (\vec{M}'_\delta - \vec{C}'_\delta). \quad (3.15)$$

Wie in Abbildung 3.3 gezeigt bilden die Vektoren  $(\vec{M}_\delta - \vec{C}_\delta)$ ,  $(\vec{M}'_\delta - \vec{C}'_\delta)$  und  $(\vec{C}'_\delta - \vec{C}_\delta)$  ein Dreieck. Für dieses Dreieck kann die folgende Gleichung aufgestellt werden

$$(\vec{C}'_\delta - \vec{C}_\delta) = (\vec{M}_\delta - \vec{C}_\delta) - (\vec{M}'_\delta - \vec{C}'_\delta). \quad (3.16)$$

$(\vec{M}_\delta - \vec{C}_\delta)$  und  $(\vec{M}'_\delta - \vec{C}'_\delta)$  können durch die Ausdrücke in den Gleichungen 3.14 und 3.15 ersetzt werden um folgende Gleichung zu erhalten

$$(\vec{C}'_\delta - \vec{C}_\delta) = \gamma' R^T K^{-1} \vec{m}_\tau - \gamma R'^T K'^{-1} \vec{m}'_{\tau'}. \quad (3.17)$$

Durch Vektoridentitäten können  $\gamma$  und  $\gamma'$  eliminiert und folgende Bedingung aus Gleichung 3.17 hergeleitet werden [14, 4]

$$\vec{m}'^T K'^{-T} T R' [\vec{C}'_\delta - \vec{C}_\delta]_x R^T K^{-1} \vec{m}_\tau = \vec{m}'^T F \vec{m}_\tau = 0, \quad (3.18)$$

mit

$$[\vec{C}'_\delta - \vec{C}_\delta]_x = \begin{bmatrix} 0 & -(C'_{z\delta} - C_{z\delta}) & C'_{y\delta} - C_{y\delta} \\ C'_{z\delta} - C_{z\delta} & 0 & -(C'_{x\delta} - C_{x\delta}) \\ -(C'_{y\delta} - C_{y\delta}) & C'_{x\delta} - C_{x\delta} & 0 \end{bmatrix}. \quad (3.19)$$

In Gleichung 3.18 wurde die Bedingungsgleichung für die sogenannte Fundamentalmatrix  $F$  definiert [10]. Sind die Kameraparameter und dadurch die Kameramatrix  $K$  bekannt, so wird die essentielle Matrix  $E$  mit  $E = R' [\vec{C}'_\delta - \vec{C}_\delta]_x R^T$  definiert. Gleichung 3.18 kann zu einer Bedingung für  $E$  umgeformt werden.

$$\vec{m}'^T K'^{-T} E K^{-1} \vec{m}_\tau = 0. \quad (3.20)$$

Die Gleichungen 3.18 und 3.20 definieren die sogenannte epipolare Bedingung[2, 10] und können verwendet werden um die Fundamentalmatrix oder die essentielle Matrix aus bekannten Korrespondierenden Punkten zu bestimmen. Für die essentielle Matrix müssen zuvor noch die Koordinaten in der Form

$$\vec{m}'_\tau = \vec{m}'^T K'^{-T} \quad (3.21)$$

$$\vec{m}_\tau = K^{-1} \vec{m}_\tau, \quad (3.22)$$

zu normierten Bildebenenkoordinaten umgerechnet werden[2, 15]. Der verwendete Algorithmus zur Bestimmung von  $F$  wird in Kapitel 3.3 näher beschrieben.

Wenn die Fundamentalmatrix bekannt ist können auch die Epipole  $e$  und  $e'$  und die Epipolarlinien  $l$  und  $l'$  aus Eigenschaften der Fundamentalmatrix bestimmt werden [2, 11, 16, 5, 15]. Um die Epipole  $e$  zu erhalten, wird der rechte Kern von  $F$  bestimmt und für  $e'$  muss der linke Kern von  $F$  bestimmt werden[2, 11, 16, 5, 15]. Es gilt also

$$Fe = 0 \quad (3.23)$$

$$F^T e' = 0. \quad (3.24)$$

Um die zu  $m$  oder  $m'$  korrespondierende Epipolarlinie  $l'$  oder  $l$  zu bestimmten kann die folgende Transformation verwendet werden[2, 11, 16, 5, 15]

$$l' = Fm \quad (3.25)$$

$$l = F^T m'. \quad (3.26)$$

Die Matrizen  $F$  und  $E$  sind mit diesen Eigenschaften wichtige Instrumente für die Bestimmung der extrinsischen und intrinsischen Kameraparameter und ihre Eigenschaften werden in den folgenden Kapiteln ausgenutzt um effiziente Rekonstruktionalgorithmen für die Szene zu implementieren.

### 3.3 Bestimmung von Homographie und Fundamentalmatrix aus Punktekorrespondenzen

Im Folgenden wird gezeigt, wie beispielsweise eine Homographie, Fundamentalmatrix und dement sprechend auch eine essentielle Matrix aus Punktekorrespondenzen gewonnen werden können. Für essentielle Matrizen gilt das selbe Verfahren wie für die Fundamentalmatrizen, nur sind hier die Punkte in der Form wie in Gleichung 3.22 gezeigt dargestellt. Die Herleitung selbst wird am Beispiel der Fundamentalmatrix aufgezeigt.

Es wird davon ausgegangen, dass die Transformationsmatrizen  $R$  und  $R'$  sowie die Kameramatrizen  $K$  und  $K'$  nicht bekannt sind. Des Weiteren wird vorausgesetzt, dass zuvor mindestens acht korrespondierende Punkte aus den jeweiligen Bildpaaren detektiert wurden. Im Realfall, werden hierfür bestimmte Detektionsalgorithmen verwendet, wie beispielsweise der SURF-Algorithmus[17], welche markante Bildpunkte in beiden Bildern suchen.

Um eine Homographiematrix mit

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \quad (3.27)$$

zu erhalten werden die Punkte beider Kameras in eine Koeffizientenmatrix  $A$  eingetragen, welche sich nach dem folgenden Schema aufstellen lässt[2, 4]. Ausgehend von der Abbildungsvorschrift aus Gleichung 3.11 gilt

$$Hm_\tau = \lambda m'_{\tau'} \quad (3.28)$$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} m_\tau \\ m'_{\tau'} \end{bmatrix} = \begin{bmatrix} \lambda m'_{\tau'} \\ \vdots \\ \lambda m'_{\tau'z} \end{bmatrix} \quad (3.29)$$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} m_{\tau x} \\ m_{\tau y} \\ m_{\tau z} \end{bmatrix} = \begin{bmatrix} \lambda m'_{\tau'x} \\ \lambda m'_{\tau'y} \\ \lambda m'_{\tau'z} \end{bmatrix}. \quad (3.30)$$

Aus Gleichung 3.30 lässt sich das folgende Gleichungssystem aufstellen.

$$h_{11} m_{x\tau} + h_{12} m_{y\tau} + h_{13} m_{z\tau} = \lambda m'_{x\tau'} \quad (3.31)$$

$$h_{21} m_{x\tau} + h_{22} m_{y\tau} + h_{23} m_{z\tau} = \lambda m'_{y\tau'} \quad (3.32)$$

$$h_{31} m_{x\tau} + h_{32} m_{y\tau} + h_{33} m_{z\tau} = \lambda m'_{z\tau'}. \quad (3.33)$$

Da mit zweidimensionalen homogenen Bildkoordinaten gearbeitet wird und somit  $m_{\tau z}$  und  $m'_{\tau'z} = 1$  ist, ergibt sich für die letzte Zeile  $h_{31} m_{\tau x} + h_{32} m_{\tau y} + h_{33} m_{\tau z} = \lambda$ . Setzt man diesen Ausdruck anstelle von  $\lambda$  in die anderen beiden Gleichungen ein, so ergeben sich

$$h_{11} m_{\tau x} + h_{12} m_{\tau y} + h_{13} m_{\tau z} = (h_{31} m_{\tau x} + h_{32} m_{\tau y} + h_{33} m_{\tau z}) \cdot m'_{\tau'x} \quad (3.34)$$

$$h_{21} m_{\tau x} + h_{22} m_{\tau y} + h_{23} m_{\tau z} = (h_{31} m_{\tau x} + h_{32} m_{\tau y} + h_{33} m_{\tau z}) \cdot m'_{\tau'y}. \quad (3.35)$$

Für den Aufbau von  $A$  werden beide Ausdrücke nach Null aufgelöst, sodass sich pro korrespondierendem Punktpaar zwei Gleichungen nach 3.36 und 3.37 ergeben.

$$h_{11} m_{\tau x} + h_{12} m_{\tau y} + h_{13} m_{\tau z} - (h_{31} m_{\tau x} + h_{32} m_{\tau y} + h_{33} m_{\tau z}) \cdot m'_{\tau'x} = 0$$

$$h_{21} m_{\tau x} + h_{22} m_{\tau y} + h_{23} m_{\tau z} - (h_{31} m_{\tau x} + h_{32} m_{\tau y} + h_{33} m_{\tau z}) \cdot m'_{\tau'y} = 0$$

$$\rightsquigarrow h_{11} m_{\tau x} + h_{12} m_{\tau y} + h_{13} m_{\tau z} - h_{31} m_{\tau x} \cdot m'_{\tau'x} - h_{32} m_{\tau y} \cdot m'_{\tau'x} - h_{33} m_{\tau z} \cdot m'_{\tau'x} = 0 \quad (3.36)$$

$$\rightsquigarrow h_{21} m_{\tau x} + h_{22} m_{\tau y} + h_{23} m_{\tau z} - h_{31} m_{\tau x} \cdot m'_{\tau'y} - h_{32} m_{\tau y} \cdot m'_{\tau'y} - h_{33} m_{\tau z} \cdot m'_{\tau'y} = 0. \quad (3.37)$$

Die entstandenen Gleichungen werden dann nach folgendem Schema in die Koeffizientenmatrix  $A$  eingetragen[4, 2, 18, 6]

$$A \cdot x = 0 \quad (3.38)$$

$$\begin{pmatrix} m_{\tau x} & m_{\tau y} & 1 & 0 & 0 & 0 & m_{\tau x} m'_{\tau'x} & m_{\tau y} m'_{\tau'x} & 1 \cdot m'_{\tau'x} \\ 0 & 0 & 0 & m_{\tau x} & m_{\tau y} & 1 & m_{\tau x} m'_{\tau'y} & m_{\tau y} m'_{\tau'y} & 1 \cdot m'_{\tau'y} \\ & & & & & & \vdots & \vdots & \vdots \\ m_{\tau x,i} & m_{\tau y,i} & 1 & 0 & 0 & 0 & m_{\tau x,i} m'_{\tau'x,i} & m_{\tau y,i} m'_{\tau'x,i} & 1 \cdot m'_{\tau'x,i} \\ 0 & 0 & 0 & m_{\tau x,i} & m_{\tau y,i} & 1 & m_{\tau x,i} m'_{\tau'y,i} & m_{\tau y,i} m'_{\tau'y,i} & 1 \cdot m'_{\tau'y,i} \end{pmatrix} \cdot \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_i \end{pmatrix} = 0. \quad (3.39)$$

Gesucht wird ein Vektor  $\vec{x}$ , für den gilt, dass  $A \cdot x = 0$ . Besitzt Matrix  $A$  einen Rang von 8, so entspricht der gesuchte Vektor  $\vec{x}$  dem Kern der Koeffizientenmatrix und ist ein Spaltenvektor mit insgesamt neun Einträgen, welche in die 3x3-Homographiematrix eingetragen werden können[2, 18].

Das Verfahren mit welchem sowohl  $F$  als auch  $E$  geschätzt werden können ähnelt in seinem Aufbau dem der Bestimmung der Homographiematrix. Das Verfahren wird hier allgemein als der 8-Punkte-Algorithmus bezeichnet[2, 3, 12]. Der 8-Punkte-Algorithmus ist eine lineare Technik, welche angewandt wird, um die Fundamentalmatrix aus  $n \geq 8$  Punkten schätzen zu können. Der Algorithmus benötigt  $n \geq 8$  Punkte, um ein valides Ergebnis zu liefern [2, 12, 3]. Das Ergebnis und jedes seiner Vielfachen ist eine mögliche Lösung für  $F$ . Der Algorithmus wird am Beispiel für die Bestimmung von  $F$  veranschaulicht. Zunächst wird eine Koeffizientenmatrix  $A$  aus Punktkorrespondenzen gebildet. Hierzu wird sich auf die für  $F$  hergeleitete Gleichung 3.18 bezogen.

$$\begin{aligned}
m'^T_{\tau'} \cdot F \cdot m_{\tau} &= 0 \\
F = \begin{bmatrix} f_{11} & f_{122} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \\
[m'_{\tau'x} & m'_{\tau'y} & 1] \cdot \begin{bmatrix} f_{11} & f_{122} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \cdot \begin{bmatrix} m_{\tau x} \\ m_{\tau y} \\ 1 \end{bmatrix} &= 0 \\
f_{11} m_{\tau x} m'_{\tau'x} + f_{12} m_{\tau y} m'_{\tau'x} + f_{13} m'_{\tau'x} + f_{21} m_{\tau x} m'_{\tau'y} \\
+ f_{22} m_{\tau y} m'_{\tau'y} + f_{23} m'_{\tau'y} + f_{31} m_{\tau x} + f_{32} m_{\tau y} + f_{33} &= 0 \tag{3.40} \\
(m_{\tau x} m'_{\tau'x}, m_{\tau y} m'_{\tau'x}, m'_{\tau'x}, m_{\tau x} m'_{\tau'y}, m_{\tau y} m'_{\tau'y}, m'_{\tau'x}, m_{\tau x}, m_{\tau y}, 1) \cdot f &= 0 \\
\begin{bmatrix} m_{\tau x} m'_{\tau'x} & m_{\tau y} m'_{\tau'x} & m'_{\tau'x} & m_{\tau x} m'_{\tau'y} & m_{\tau y} m'_{\tau'y} & m'_{\tau'y} & m_{\tau x} & m_{\tau y} & 1 \\ \vdots & \vdots \\ m_{\tau x,i} m'_{\tau'x,i} & m_{\tau y,i} m'_{\tau'x,i} & m'_{\tau'x,i} & m_{\tau x,i} m'_{\tau'y,i} & m_{\tau y,i} m'_{\tau'y,i} & m'_{\tau'y,i} & m_{\tau x,i} & m_{\tau y,i} & 1 \end{bmatrix} \cdot \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} &= 0 \\
A \cdot f &= 0
\end{aligned}$$

Gesucht wird ein Vektor  $\vec{f}$ , für den gilt, dass  $A \cdot f = 0$ . Besitzt Matrix  $A$  einen Rang von 8, so entspricht der gesuchte Vektor  $\vec{x}$  auch hier dem Kern von  $A$  und ist ein Spaltenvektor mit insgesamt neun Einträgen, welche in die 3x3-Fundamentalmatrix eingetragen werden können[2, 5].

Bei der Homographie, wie auch bei der Fundamentalmatrix, kann es zu überbestimmten Systemen kommen. Ein System gilt als überbestimmt, wenn es durch mehr Gleichungen als Unbekannte beschrieben wird[18, 19]. Für die Koeffizientenmatrix für  $F$  und  $H$  hätte das zur Folge, dass sie in ihrem Rang steigt. Die Bestimmung des Kerns würde in beiden Fällen kein eindeutiges Ergebnis mehr liefern[2, 18].

Für die Lösung überbestimmter Systeme wird durch ein *Least-Square*-Verfahren, mit Hilfe der Singulärwertszerlegung einer Matrix  $A$ , eine Lösung für einen Vektor  $\vec{x}$  gesucht, so dass  $\| A \cdot x \|$  minimal wird [2, 19, 18]. Die Singulärwertzerlegung von  $A$  ist eine Faktorisierung der Matrix  $A \in \mathbb{R}^{m \times n}$  der Form  $A = U \cdot \Sigma \cdot V^T$  mit orthogonalen Matrizen  $U \in \mathbb{R}^{m \times n}$  und  $V \in \mathbb{R}^{m \times n}$  sowie mit einer Diagonalmatrix  $\Sigma$

$$S = \begin{pmatrix} s_1 & \dots & 0 & 0 & \dots & 0 \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ 0 & \dots & s_r & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \cdot & & \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot & & \cdot \\ 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}. \quad (3.41)$$

Die Diagonalmatrix  $\Sigma$  beinhaltet die Singulärwerte der Matrix. Dabei soll für die diagonalen Singulärwerte in  $\Sigma$  mit  $s_1$  bis  $s_r$  gelten, dass  $s_1 \geq s_2 \geq \dots \geq s_r \geq 0$ [19]. Die Spalte der Matrix  $V^T$ , welche mit dem kleinsten Singulärwert von  $\Sigma$  korrespondiert, ergibt den Vektor  $\vec{x}$ , für den  $\| A \cdot x \|$  minimal wird.

## 4 Synthetische Rekonstruktion

Anhand der erarbeiteten mathematischen Grundlagen ist ein Algorithmus für die Rekonstruktion einer Szene aus einer Stereobildaufnahme entstanden. Der Algorithmus wurde mit dem Ziel der Kamerakalibrierung und der Szenenrekonstruktion aus Bildquellen unterschiedlicher Auflösungen entwickelt, da Stereokalibrierungsverfahren einiger Computer-Vision-Applikationen keine unterschiedlichen Auflösungen von Kameras berücksichtigen. Der entwickelte Algorithmus ist in der Lage aus einem Stereobildpaar extrinsische Kameraparameter zu bestimmen und anhand dessen die 3D-Szene zu rekonstruieren, jedoch unter der Voraussetzung, dass die intrinsischen Kameraparameter beider Kameras bekannt sind.

Im Folgenden wird der Algorithmus anhand eines synthetischen Beispiels erklärt. Dabei werden die einzelnen Schritte des Aufbaus einer virtuellen 3D-Szene, der Bestimmung der extrinsischen Kameraparameter und der Rekonstruktion der virtuellen 3D-Szene beschrieben. Abbildung 4.1 fasst den Arbeitsprozess des Szenenrekonstruktionsalgorithmus für das virtuelle Beispiel zusammen.

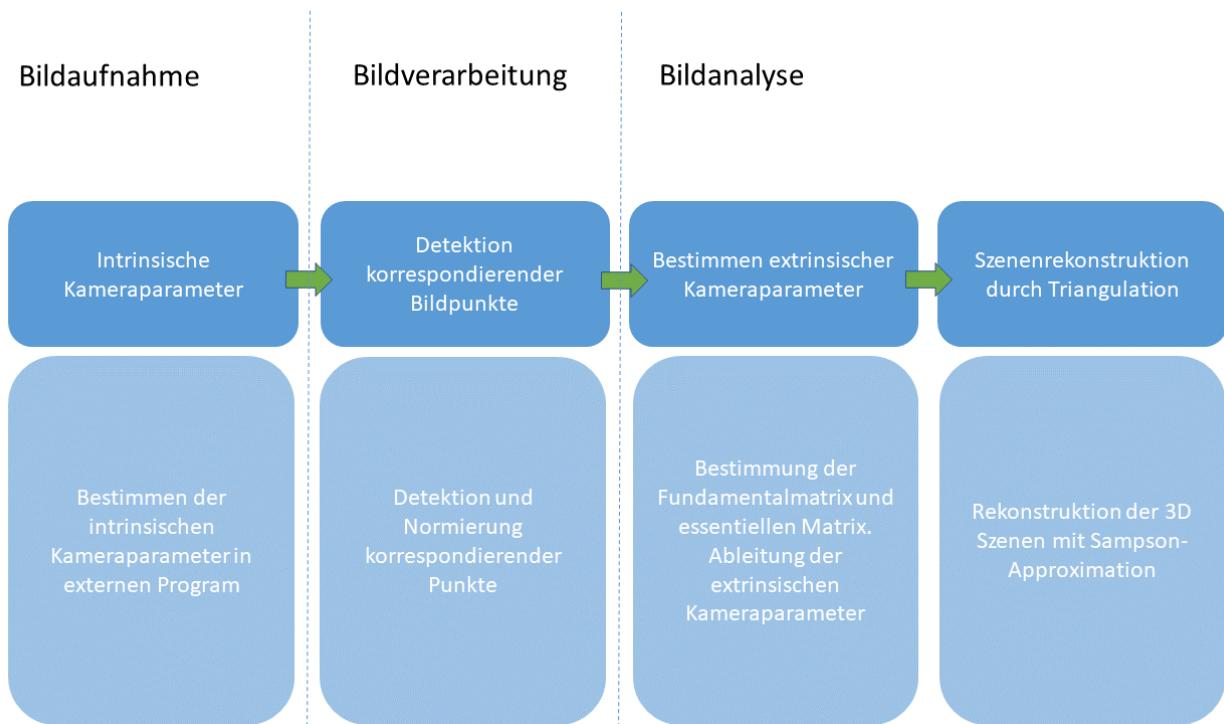


Abbildung 4.1: Ablaufdiagramm für das synthetische Beispiel

## 4.1 Simulierte Bildaufnahme einer virtuellen Szene

Als 3D-Objekt wurde ein Quader, in ein Weltkoordinatensystem  $(O, \delta)$  mit  $\delta = (\hat{d}_1, \hat{d}_2, \hat{d}_3)$  positioniert. Es werden zwei Kameras  $(C, \beta)$  mit  $\beta = (\hat{b}_1, \hat{b}_2, \hat{b}_3)$  und  $(C', \beta')$  mit  $\beta' = (\hat{b}'_1, \hat{b}'_2, \hat{b}'_3)$  in  $(O, \delta)$  platziert. Das Weltkoordinatensystem  $(O, \delta)$  und das Kamerakoordinatensystem  $(C, \beta)$  sind deckungsgleich.  $C'$  ist relativ zu  $C$  verschoben und rotiert. Die zwei Bildebene  $(I, \tau)$  mit  $\tau = (\hat{t}_1, \hat{t}_2, \hat{t}_3)$  und  $(I', \tau')$  mit  $\tau' = (\hat{t}'_1, \hat{t}'_2, \hat{t}'_3)$  sind vor  $C$  und  $C'$  positioniert. Die Sensorkoordinatensysteme  $(S, \sigma)$  und  $(S', \sigma')$  wurden gleich den Bildebenekoordinatensystemen  $(I, \tau)$  und  $(I', \tau')$  gesetzt. Es wird von zwei identischen Kameras ausgegangen und somit werden für den hier diskutierten Fall ausschließlich mit dem vereinfachten Kameramatrizen  $K_0$  gerechnet. Der schematische Aufbau der Szenen ist in den Abbildungen 4.2, 4.3 und 4.4 dargestellt.

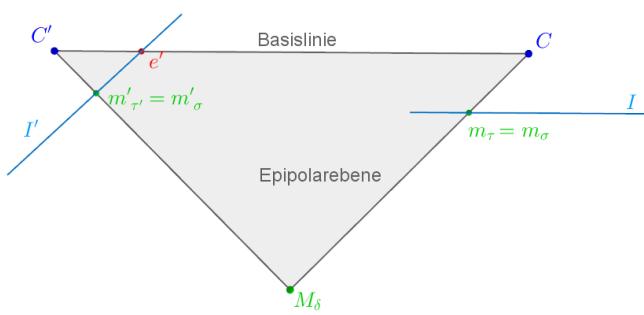


Abbildung 4.2: In der Abbildung ist der vereinfachte Stereoaufbau in einer Top-Down-Ansicht zu sehen

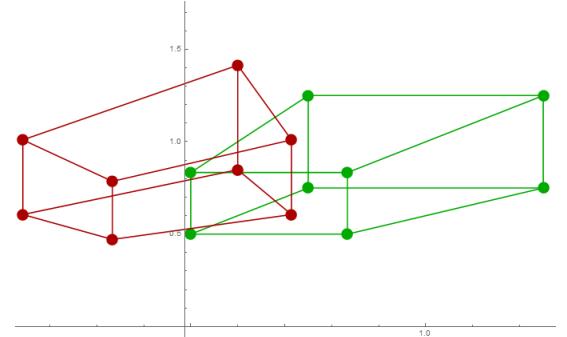


Abbildung 4.3: Simulierte Abbildung des Quaders auf die Kamera  $C$  in Grün und auf  $C'$  in Rot

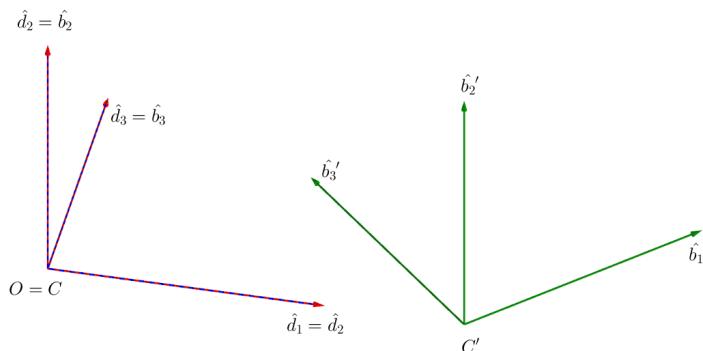


Abbildung 4.4: Abbildung der verschiedenen Kamerakoordinatensysteme, das Weltkoordinatensystem  $O$  ist zum Kamerakoordinatensystem  $C$  deckungsgleich und  $C'$  verschoben und rotiert.

Um die Eckpunkte des Quaders auf die Bildebene von  $C$  und  $C'$  abbilden zu können, werden zunächst die Projektionsmatrizen  $P$  und  $P'$  aufgestellt.  $C$  ist deckungsgleich mit dem Weltkoordinatensystem  $(O, \delta)$ . Es gilt also

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (4.1)$$

$$C = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.2)$$

$$T = R[I] - C \quad (4.3)$$

$$T = \begin{bmatrix} 1 & 0 & 0 & C_x \\ 0 & 1 & 0 & C_y \\ 0 & 0 & 1 & C_z \end{bmatrix}. \quad (4.4)$$

$C'$  dagegen ist gegenüber  $C$  verschoben und rotiert. Als Beispiel wird eine Rotation um die  $\hat{b}_2'$  Achse für  $C'$  bestimmt. Somit gilt für  $P'$ :

$$R' = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \quad (4.5)$$

$$\vec{C}' = \begin{pmatrix} 1 & 0 & 0 & -C'_x \\ 0 & 1 & 0 & -C'_y \\ 0 & 0 & 1 & -C'_z \end{pmatrix} \quad (4.6)$$

$$T' = R'[I] - C \quad (4.7)$$

$$T' = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -C'_x \\ 0 & 1 & 0 & -C'_y \\ 0 & 0 & 1 & -C'_z \end{pmatrix} \quad (4.8)$$

$$T' = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & -C'_x \cos(\alpha) + C'_z \sin(\alpha) \\ 0 & 1 & 0 & -C'_y \\ \sin(\alpha) & 0 & \cos(\alpha) & -C'_x \sin(\alpha) - C'_z \cos(\alpha) \end{pmatrix} \quad (4.9)$$

Der Quader hat insgesamt acht Punkte, welche auf die Bildebenen der Kameras projiziert werden. Neben den Punkten des Quaders wird noch ein weiterer Punkt außerhalb des Quaders platziert und ebenfalls auf die Bildebenen projiziert. Mit insgesamt neun Punkten bei der Bestimmung der Fundamentalmatrix, wird die Wahrscheinlichkeit ein unterbestimmtes System aus der Koeffizientenmatrix  $A$  zu bekommen minimiert. Ist  $A$  unterbestimmt so besitzt sie Rang 7 und  $F$  kann nicht eindeutig durch einen Sieben-Punkte-Algorithmus bestimmt werden[2, 20]. In dem hier berechneten Beispiel werden deswegen neun Punkte benutzt um sicherzugehen, dass  $A$  Rang 8 besitzt und somit eindeutig bestimmt werden kann. Zur Bestimmung von  $F$ , wird der in Kapitel 3 aufgeführte Acht-Punkte-Algorithmus angewandt.

Um die neun Punkte auf die Bildebenen  $(I, \tau)$  und  $(I', \tau')$  zu projizieren, müssen neben den Transformationsmatrizen  $R$  und  $R'$  noch die Kameramatrizen  $K_0$  und  $K'_0$  festgelegt werden.

$$K_0 = \begin{bmatrix} \zeta_C & 0 & 0 \\ 0 & \zeta_C & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.10)$$

$$K'_0 = \begin{bmatrix} \zeta_{C'} & 0 & 0 \\ 0 & \zeta_{C'} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (4.11)$$

Da zunächst von gleichen Kameraauflösungen ausgegangen wird, gilt  $\zeta = \zeta'$ . Sind  $R, R', K_0$  und  $K'_0$  bekannt, können die Projektionsmatrizen gebildet und anschließend die Punkte auf die Bildebene projiziert werden. Die entstandenen Bilder sind in Abbildung 4.3 zu sehen.

$$P = K_0 \cdot R \quad (4.12)$$

$$P = \begin{bmatrix} \zeta_C & 0 & 0 \\ 0 & \zeta_C & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.13)$$

$$P' = K'_0 \cdot R' \quad (4.14)$$

$$P' = \begin{bmatrix} \zeta_{C'} \cos(\alpha) & 0 & \zeta_{C'} \sin(\alpha) & -\zeta_{C'}(v'_1 \cos(\alpha) + v'_3 \sin(\alpha)) \\ 0 & 1 & 0 & \zeta_{C'} - v'_2 \\ \zeta_{C'} \sin(\alpha) & 0 & \zeta_{C'} \cos(\alpha) & -\zeta_{C'}(v'_1 \sin(\alpha) + v'_3 \cos(\alpha)) \end{bmatrix}. \quad (4.15)$$

## 4.2 Bildanalyse

Der Szenenrekonstruktionsalgorithmus für das synthetische Beispiel ist in drei Abschnitte unterteilt. Zuerst wird aus den Punktekorrespondenzen die Fundamentalmatrix und die essentielle Matrix geschätzt. Mit Hilfe der essentiellen Matrix werden die extrinsischen Kameraparameter bestimmt, um so im letzten Schritt die Szenenpunkte durch Rückprojektion der Bildpunkte mit Hilfe der Kameraparameter rekonstruiert.

### 4.2.1 Bestimmung der extrinsischen Kameraparameter

Zur Bestimmung der extrinsischen Kameraparameter werden in dem hier berechneten Beispiel neun korrespondierende Punkte bestimmt. Mittels der epipolaren Bedingung aus Gleichung 3.18 wird der Acht-Punkt-Algorithmus angewandt um die Fundamentalmatrix zu bestimmen.

$$0 = \vec{m}'_{\tau'}^T K'^{-T} R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T K^{-1} \vec{m}_{\tau} \quad (4.16)$$

$$= \vec{m}'_{\tau'}^T F \vec{m}_{\tau}. \quad (4.17)$$

Wie in Kapitel 3.2 hergeleitet, bildet sich die essentielle Matrix aus

$$\vec{m}'_{\tau'}^T K'^{-T} R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T K^{-1} \vec{m}_{\tau} = 0 \quad (4.18)$$

$$\rightsquigarrow E = R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T. \quad (4.19)$$

Um also  $E$  aus  $F$  zu bestimmen, gilt

$$E = K'^T F K \quad (4.20)$$

$$E = K'^T (K'^{-T} R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T K^{-1}) K \quad (4.21)$$

$$E = R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T. \quad (4.22)$$

Es wird davon ausgegangen, dass für  $T = [R] - RC$  von  $C$  gilt, dass  $T = [I] - 0$  ist. Die aus  $E$  zu ermittelnde Matrix  $T'$  beschreibt dann die Transformation von  $C'$  relativ zu  $C[2, 3]$ . Somit kann  $E$  umformuliert werden zu

$$E = R' [\vec{C}'_{\delta} - \vec{C}_{\delta}]_{\times} R^T \quad (4.23)$$

$$E = R' [\vec{C}'_{\delta} - 0]_{\times} I^T \quad (4.24)$$

$$E = R' [\vec{C}'_{\delta}]_{\times}. \quad (4.25)$$

Um  $R'$  und  $[\vec{C}'_\delta]_\times$  zu bestimmen wird zunächst die essentielle Matrix  $E$ , mit Hilfe der Singulärwertszerlegung, in drei Matrizen zerlegt.

$$E = U\Sigma V^T. \quad (4.26)$$

Die Singulärwerte  $\text{diag}(\sigma_1, \sigma_2, \sigma_3)$  der Matrix  $\Sigma$  müssen die Bedingung erfüllen, dass  $\Sigma = \text{diag}(1, 1, 0)[2, 3]$ . Wenn diese Bedingung nicht erfüllt ist, so wird sie erzwungen. Dazu wird Matrix  $\Sigma$  aus der Singulärwertszerlegung aus  $E$  modifiziert[2, 3].

$$E' = U\text{diag}(1, 1, 0)V^T. \quad (4.27)$$

$[\vec{C}'_\delta]_\times$  ist schiefsymmetrisch und kann in  $UZU^T$  zerlegt werden, wobei  $U$  eine orthogonale Matrix ist und  $Z$  eine block-diagonale Matrix[2].  $R'$  wird in  $UWV^T$  und  $UW^TV^T$  zerlegt, wobei  $W$  eine schiefsymmetrische Matrix ist[3, 2, 15].

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \quad (4.28)$$

Somit lassen sich die folgenden Lösungsmöglichkeiten für  $[\vec{C}'_\delta]_\times$  und  $R_1$  und  $R_2$  aufstellen[2, 3].

$$[\vec{C}'_\delta]_\times = \pm UZU^T \quad (4.29)$$

$$R'_1 = UW^TV^T \quad R'_2 = UWV^T. \quad (4.30)$$

$[\vec{C}'_\delta]_\times$  ist eine schiefsymmetrische Matrix, welche die Information für den noch gesuchten Translationsanteil  $v = -R'C'$  beinhaltet. Ohne zusätzliche Informationen kann  $v$  nur bis zu einer Skaleninvarianz genau bestimmt werden[2, 3, 15]. Durch die Modifizierung der Singulärwerte von  $E$  gilt für  $\|v\| = 1[2, 3]$ . Das bedeutet, dass es sich bei dem Translationsvektor  $v$  lediglich um den normierten Richtungsvektor zwischen  $C$  und  $C'$  handelt[21]. Um  $v$  aus  $[\vec{C}'_\delta]_\times$  zu extrahieren, wird der Kern von  $[\vec{C}'_\delta]_\times$  bestimmt.

$$[\vec{C}'_\delta]_\times \cdot v = v \times v = 0. \quad (4.31)$$

Die Skaleninvarianz bewirkt, dass es bei der Rekonstruktion die Größe der Objekte von ihrer Originalgröße abweichen, da es sich bei  $v$  nur um den normierten Richtungsvektor der ursprünglichen Strecke handelt. Die Abbildung 4.8 zeigt die Auswirkungen der Skaleninvarianz auf die später rekonstruierte Szene.

Letztendlich können für die Rekonstruktion der extrinsischen Kameraparameter vier mögliche Lösungen für  $T$  in Form von  $T = R[I] - C$ , wie in Gleichung 2.13 in Kapitel 2 definiert, gefunden werden[2, 3, 15].  $\lambda v$  heißt dabei, dass sowohl  $v$  also auch alle Vielfache von  $v$  Lösungen sein können, was durch die Skaleninvarianz der Resultate bedingt ist[2, 3, 15].

$$T' = [UWV^T] + \lambda v \quad \text{oder} \quad [UW^TV^T] + \lambda v \quad (4.32)$$

$$\text{oder} \quad [UWV^T] - \lambda v \quad \text{oder} \quad [UW^TV^T] - \lambda v \quad (4.33)$$

Die Abbildungen 4.5 und 4.6 stellen schematisch die vier verschiedenen Transformationsmöglichkeiten von  $T'$  dar. Die richtige Lösung  $T$  ist diejenige, bei der das Abbild der Objekte vor den Kameras liegt.

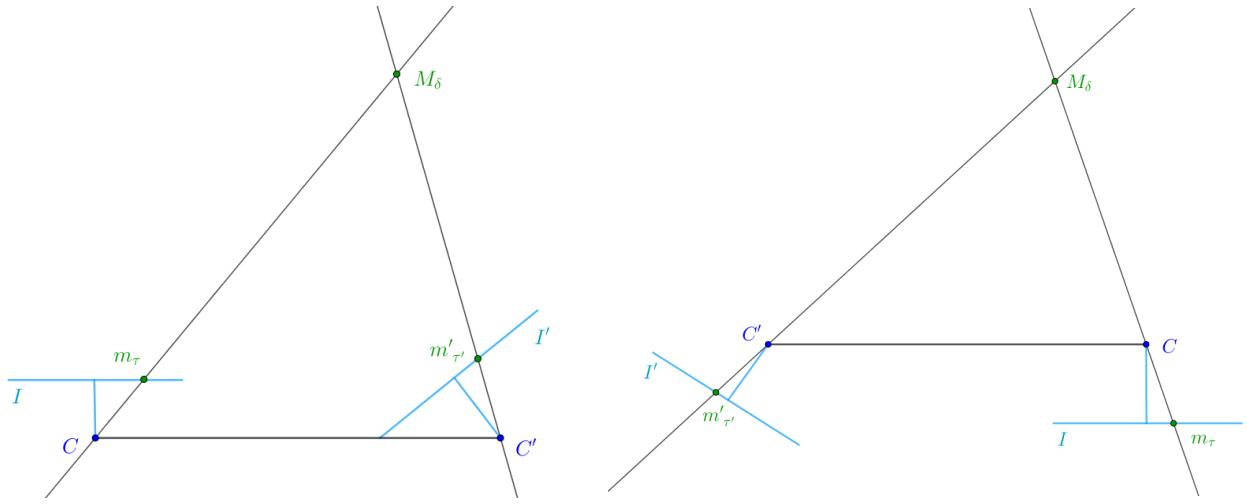


Abbildung 4.5: In den ersten beiden Abbildungen kommt es zu einer Umkehrung der Baseline.

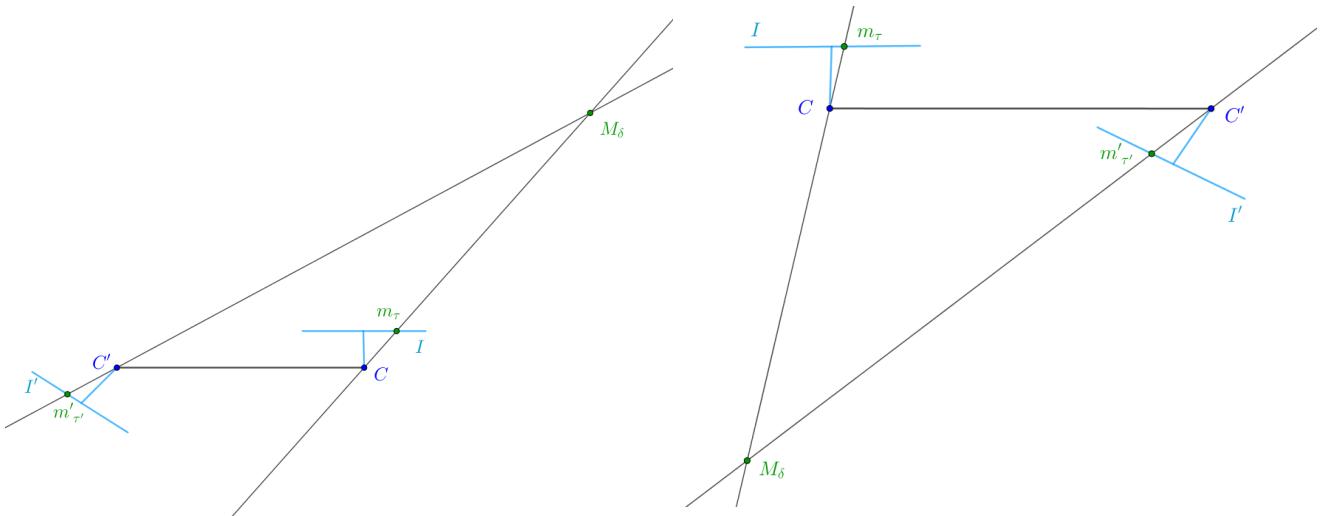


Abbildung 4.6: In den beiden Abbildungen wird  $C'$  und  $180^\circ$  gedreht

#### 4.2.2 Szenenrekonstruktion durch Triangulation

Als Triangulierung wird in der Computer Vision die Bestimmung eines 3D-Objektpunktes aus korrespondierenden Bildpunkten bezeichnet. Als Voraussetzung für die Rekonstruktion müssen die jeweiligen korrespondierenden Bildpunkte und die Kameraparameter der einzelnen Kameras bekannt sein. Die Triangulierung funktioniert wie eine umgekehrte Projektion der Bildpunkte auf der Bildebene in einen Objektpunkt im Raum. Zwei Geraden, welche jeweils durch die Projektionszentren und den zu rekonstruierenden Bildpunkten gehen, treffen sich im Raum. Der Schnittpunkt beider Geraden bildet den zu den Bildpunkten gehörenden Ursprungspunkt, wie in Abbildung 4.7 schematisch dargestellt ist.

Im synthetischen Beispiel wird mit reinen Daten gearbeitet. Das heißt, die Bildpunkte wurden mathematisch von ihrem Ursprungspunkt im Raum berechnet und sind somit frei von Verfälschungen durch äußere Einflüsse. Ein zum Bildpunkt  $m_\tau$  korrespondierender Bildpunkt  $m'_{\tau'}$  liegt genau auf der zu  $m_\tau$  korrespondierenden Epipolarlinien  $l'$ . Somit ist garantiert, dass sich die Bildpunkte  $m_\tau$  und  $m'_{\tau'}$  bei einer Rückprojektion in einem Punkt  $M_{\delta,0}$  im Raum treffen. Durch die zuvor erwähnte Skaleninvarianz der extrinsischen Kameraparameter handelt es sich bei  $M_{\delta,0}$  jedoch noch nicht um den eigentlichen Ursprungspunkt  $M_\delta$ .

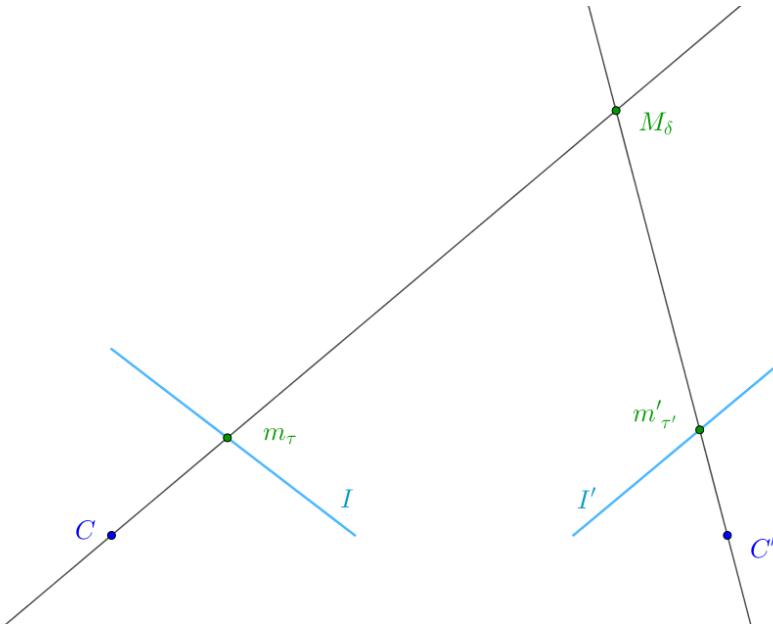


Abbildung 4.7: Optimale Triangulierung: Beide Geraden Treffen sich in einem Punkt im 3D-Raum

Vor der Bestimmung der extrinsischen Kameraparameter wurde festgesetzt, dass  $T = [I| - 0]$  die Translationsmatrix von  $C$  ist. Somit gilt für die Projektionsmatrix von  $C$ , dass  $P = K_0 T = K_0[I| - 0]$ . Die Projektionsmatrix  $P'$  für  $C'$  setzt sich aus einer der Lösungen von  $T'$  und der Kameramatrix  $K'_0$  zusammen, sodass gilt  $P' = K'_0 T' = K[R'| - RC]$ .

Um eine Gerade von den Projektionszentren  $C$  und  $C'$  durch die jeweiligen Bildpunkte bilden zu können, müssen die Positionen von  $C$  und  $C'$  bekannt sein. Da die Koordinatensysteme  $(C, \beta)$  und  $(O, \delta)$  deckungsgleich sind, und  $P = K_0[I| - 0]$  ist, gilt  $C = (0, 0, 0)^T$ . Um  $C'$  aus  $T' = R'[R'| - R'C']$  zu bestimmen wird der Translationsvektor  $-R'C'$  aus  $T'$  mit dem transponierten Rotationsmatrix  $R'^T$  aus  $T'$  multipliziert.

$$C' = R'^T \cdot -R'C' \quad (4.34)$$

Die zweidimensionalen Bildpunkte werden mit den bekannten Brennweiten  $\zeta$  und  $\zeta'$  aus  $K_0$  und  $K'_0$  zu einer dreidimensionalen Koordinate erweitert.

$$\begin{pmatrix} m_{\tau x} \\ m_{\tau y} \\ \zeta \end{pmatrix} \rightsquigarrow \begin{pmatrix} m_{\tau x} \\ m_{\tau y} \\ \zeta \end{pmatrix} \quad (4.35)$$

$$\begin{pmatrix} m'_{\tau' x} \\ m'_{\tau' y} \\ \zeta' \end{pmatrix} \rightsquigarrow \begin{pmatrix} m'_{\tau' x} \\ m'_{\tau' y} \\ \zeta' \end{pmatrix} \quad (4.36)$$

Danach werden für die Rückprojektion zwei Geradengleichungen aufgestellt. Eine Gerade geht durch  $C$  und  $\begin{pmatrix} m_{\tau x} \\ m_{\tau y} \\ \zeta \end{pmatrix}$  die zweite Gerade geht durch die Punkte  $C'$  und  $\begin{pmatrix} m'_{\tau' x} \\ m'_{\tau' y} \\ \zeta' \end{pmatrix}$ . Anschließend wird aus den zwei Geraden der Schnittpunkt  $M_{\delta,0}$  im Raum bestimmt.  $C$  und  $C'$  sind aus Sicht des Weltkoordinatensystems  $(O, \delta)$  definiert.  $m'_{\tau'}$  wird in Koordinaten bezüglich des Kamerakoordinatensystem  $(C, \beta)$  transformiert mit  $m'_{\beta} = [R[I| - C']]^{-1} \cdot m'_{\tau'}$ .

$$g := \vec{C} + t \cdot \begin{pmatrix} m_{\beta x} \\ m_{\beta y} \\ \zeta \end{pmatrix} \quad (4.37)$$

$$g' := \vec{C}' + t \cdot \begin{pmatrix} m'_{\beta x} \\ m'_{\beta y} \\ \zeta' \end{pmatrix} \quad (4.38)$$

Bei den zuvor ermittelten extrinsischen Kameraparametern ist der Translationsvektor skaleninvariant. Dies führt dazu, dass der rekonstruierte Objektpunkt  $M_{\delta,0}$  nach der Szenenrekonstruktion noch nicht dem Ursprünglichen  $M_\delta$  entsprechen muss. Dementsprechend wird als letzter Schritt die rekonstruierte Szenen anhand einer bekannten Referenzgröße skaliert. Als Referenzgröße kann beispielsweise ein zuvor abgemessener Abstand zwischen zwei Punkten in der Szene dienen. Im synthetischen Aufbau sind beispielsweise die Abstände zwischen den Originalbildpunkten bekannt. Die Abbildung 4.8 zeigt die Szene des Quaders mit unterschiedlichen Skalierungen. Die Abbildung 4.9 zeigt die rekonstruierte Szene des synthetischen Beispiels auf ihre Ursprungsgröße skaliert.

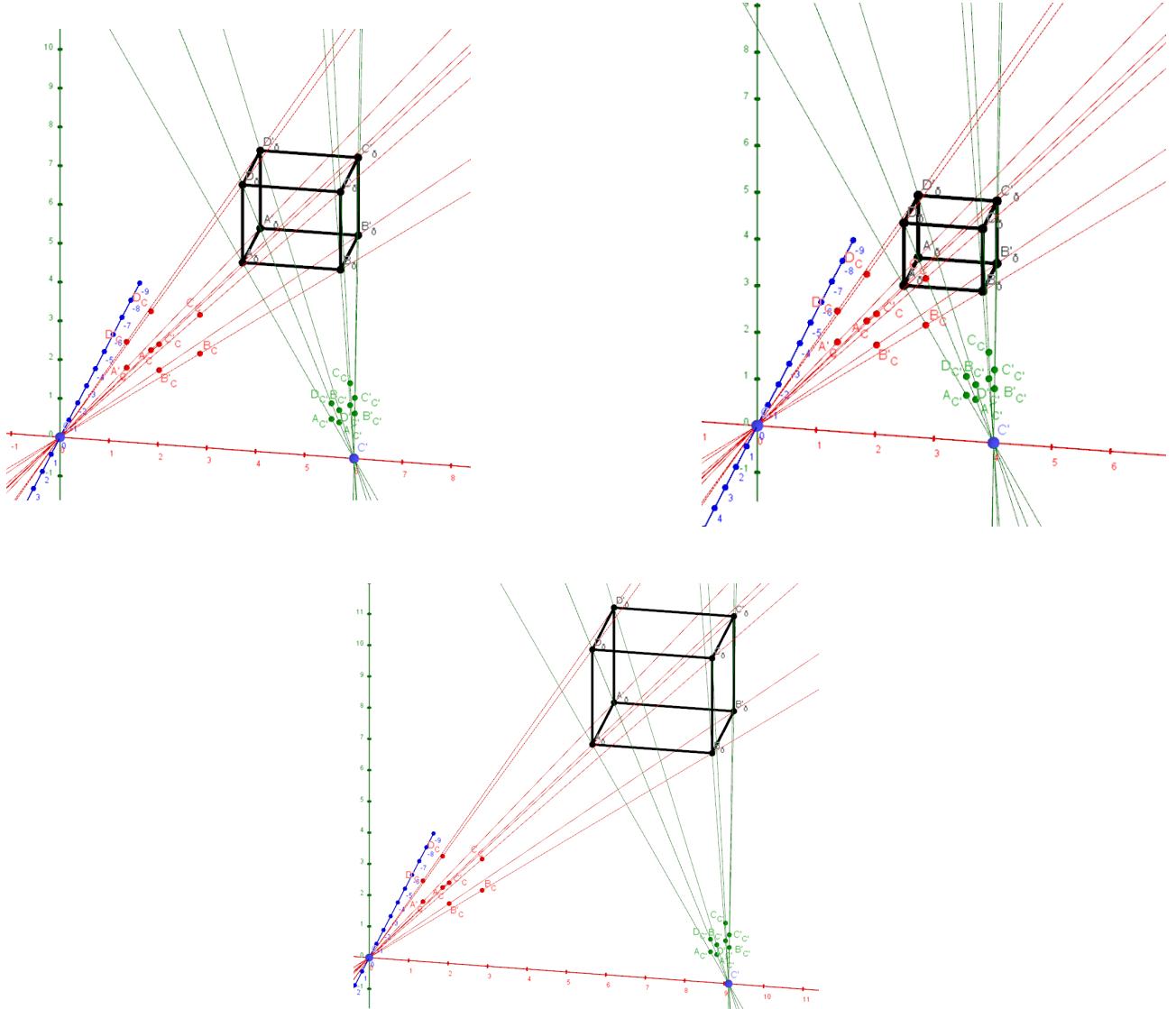


Abbildung 4.8: Veranschaulichung der Skaleninvarianz und dessen Auswirkung auf die geometrische Form und Größe der Objekte

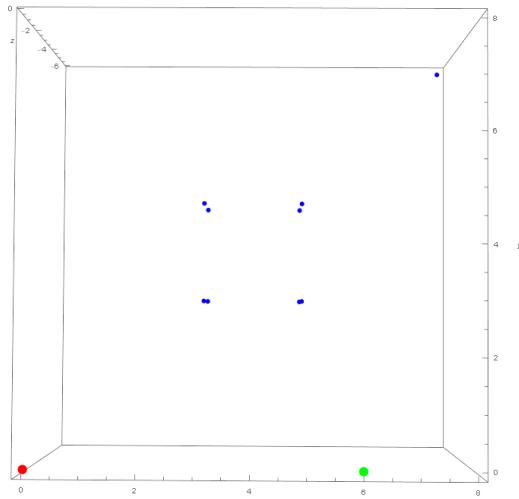
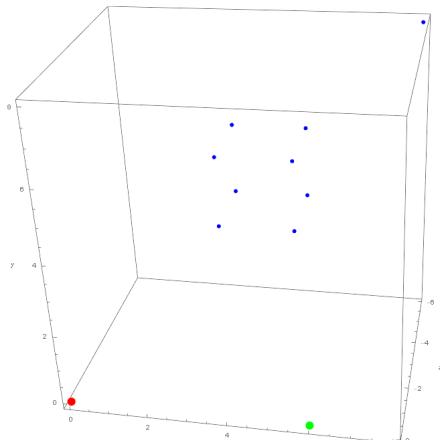


Abbildung 4.9: Der rote Punkt stellt die Postion von  $C$  dar, der grüne steht für die Position von  $C'$  relativ zu  $C$ . Die blauen Punkte stellen den rekonstruierten Quader und den extern platzierten neunten Punkt da. Die Abbildungen entstand aus dem in *Mathematica*[22] implementierten Algorithmus.

## 4.3 Auswirkungen von unterschiedlichen Kameraauflösungen

Der entstandene Szenenrekonstruktionsalgorithmus wurde anhand von Kameras gleicher Auflösung implementiert und bereits validiert. Im folgenden soll nun getestet werden, ob der entwickelte Algorithmus auch im Stande ist für Kameras unterschiedlicher Auflösung eine Szene richtig zu Rekonstruieren. Zunächst wird beschrieben, welche Modifizierungen auf einem Sensor bei Veränderung der Auflösung stattfinden. Anschließend wird analysiert, wie sich die Auflösungsänderung auf das in Kapitel 2 beschriebene Kameramodell ändert und welche Einflüsse sie auf die in Kapitel 3 hergeleiteten Epipolaren Bedingungen und somit auf die Bestimmung der extrinsischen Kameraparameter und der folgenden Szenenrekonstruktion hat. Zum Schluss werden die Ergebnisse der synthetischen Rekonstruktion mit unterschiedlichen Kameraauflösungen präsentiert und validiert.

## 4.4 Geometrie eines Sensors

Die Geometrie eines Sensors kann als eine  $M \times N$ -Matrix, bestehend aus  $M \times N$  Sensorelementen dargestellt werden[8]. Die Auflösung eines Sensors hängt von den horizontalen und vertikalen Abständen der Sensorelemente ab. Abbildung 4.10 zeigt den schematischen Aufbau eines Sensors (CMOS).

Ein Sensor hat eine maximale Auflösung. Die maximale Anzahl der Sensorelemente auf einem Sensor beschränkt die maximale Auflösung. Verschiedene Kameras können aus diesem Grund verschiedene Auflösungen besitzen. Die Anzahl und Größe der einzelnen Sensorelemente variiert mit den Größen der Sensorchips. Je größer ein Sensor ist und je mehr Sensorelemente er besitzt, desto besser ist die Bildqualität[8]. Bei maximaler Auflösung definiert genau ein Sensorelement einen Pixel. Ein Pixel wiederum entspricht einem Bildpunkt[8]. Auch ist es möglich die Auflösung eines Sensors digital zu verändern. Wird eine Auflösung kleiner der maximalen Auflösung eingestellt, desto geringer wird die Anzahl der Pixel. Der Prozess, welcher hier stattfindet, gehört zu den Nachbarschaftsoperationen. Benachbarte Pixel werden hier zu einem neuen Pixel definiert[8].

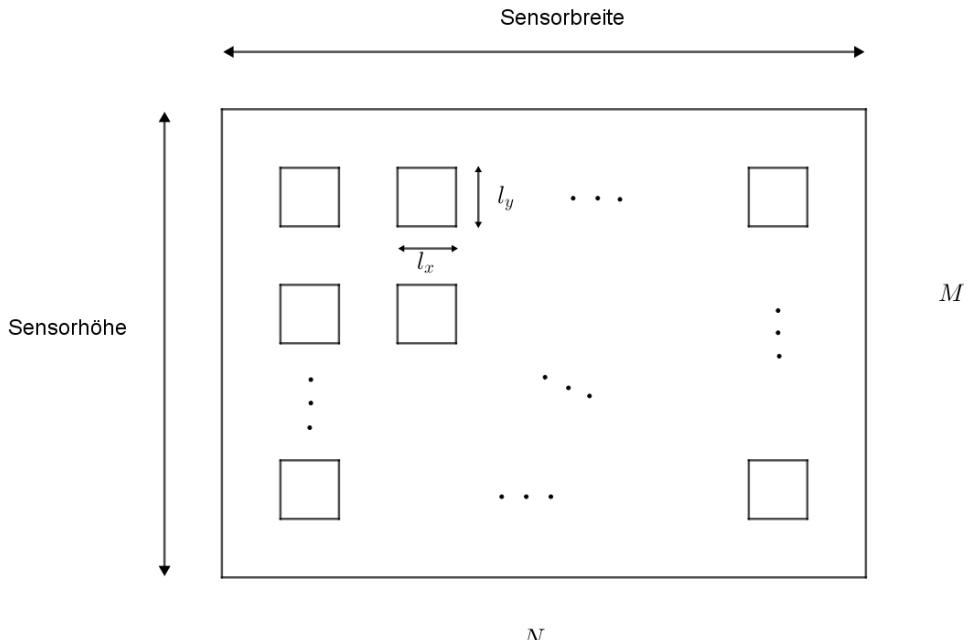


Abbildung 4.10: Rechteckiger Bildsensor mit darauf sich befindenden quadratischen Sensorelementen. Vergleiche [8]

Eine Veränderung der Auflösung kann auch eine Änderung der Seitenverhältnisse mit einschließen. Ändert sich das Seitenverhältnis so wird der Bereich der lichtempfindlichen Fläche auf dem Sensor beschränkt[8]. Dies führt dazu, dass sich die Bildausschnitte ändern. Abbildung 4.11 stellt schematisch da wie sich die lichtempfindlichen Bereiche auf dem Sensor bei unterschiedlichen Auflösungen ändert.

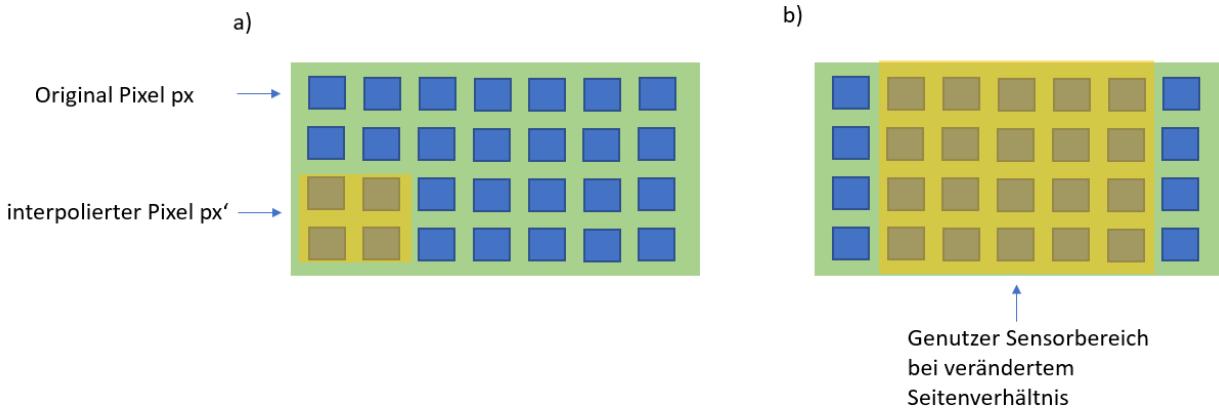


Abbildung 4.11: Bild a) zeigt den Zusammenschluss mehrerer benachbarter Pixel zu einem neuen Pixel. Bild b) zeigt in gelb markiert, den aktiven lichtempfindlichen Bereich des Sensors, wenn das Seitenverhältnis geändert wird und nicht mehr der komplette Sensor genutzt wird.

## 4.5 Auswirkungen auf die Szenenrekonstruktion

Im folgenden wird zunächst analysiert, welche Änderungen sich im Lochkameramodell bei veränderter Auflösung ergeben und was für Auswirkungen diese auf die in Kapitel 3 aufgestellten Epipolaren Bedingungen hat.

Im Lochkameramodell hat eine Änderung der Kameraauflösung lediglich eine Auswirkung auf die Skalierung der Sensorkoordinatenachsen. Mit der Auflösung, ändern sich die Anzahl und die Größe der Pixel. Die Längeneinheiten des Sensorkoordinatensystems orientieren sich, wie in Kapitel 2 beschrieben, an genau diesen Längenskalierung der Pixelkanten  $l_x$  und  $l_y$ . Folglich kommt es zu einer Skalierung des Sensorkoordinatensystems. Alle anderen Koordinatensysteme bleiben unverändert. Durch Nachbarschaftsopterationen werden aus mehreren benachbarten Pixel ein neuer, jedoch bleibt der Ort des Pixel der gleiche[23]. Die Abbildungen 4.12 und 4.13 zeigen, dass sich zwar die Projektion von Bildbenennkoordinatensystem auf das Sensorkoordinatensystem für den Punkt  $m_\sigma$  ändert, jedoch wird der Bildpunkt  $m_{\tau'}$  an der selben Position des Sensors wie zuvor auch abgebildet.

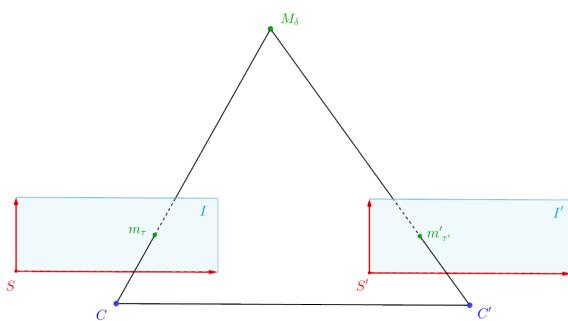


Abbildung 4.12:  $C$  und  $C'$  haben die selbe Auflösung eingestellt

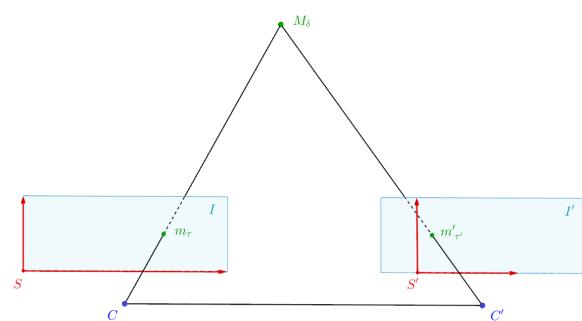


Abbildung 4.13:  $C$  und  $C'$  haben unterschiedliche Auflösungen eingestellt

Eine Skalierung der Sensorkoordinaten bedeutet, dass sich die Brennweite in Pixeleinheiten gegeben ändert, jedoch ändert sich nicht die effektive Brennweite in Millimeter. Anhand des Aufbaus der Kameramatrix aus Kapitel 2 kann das nochmal verdeutlicht werden

$$K = \begin{bmatrix} k_x \zeta & 0 & V_{\sigma x} \\ 0 & k_y \zeta & V_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix}. \quad (4.39)$$

Wie bereits bekannt, kommt es bei der Transformation von Bildebeneenkoordinaten  $m_\tau$  auf Sensorkoordinaten  $m_\sigma$  zu einer Skalierung der Bildebeneenkoordinaten in Millimeter auf Sensorkoordinaten in Pixel.  $\zeta_x$  und  $\zeta_y$  stehen für die Brennweite. Durch die Multiplikation mit  $k_x$  und  $k_y$  wird die Brennweite auf Pixeleinheiten skaliert. Die ursprüngliche Brennweite beträgt  $\zeta_x = \zeta_y = 1$ . Kommt jetzt eine Skalierung von  $k_x = k_y = 2$  dazu, ergibt sich die folgende Matrix

$$K_0 = \begin{bmatrix} \zeta & 0 & V_{\tau x} \\ 0 & \zeta & V_{\tau y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & V_{\tau x} \\ 0 & 1 & V_{\tau y} \\ 0 & 0 & 1 \end{bmatrix} \quad (4.40)$$

$$K = \begin{bmatrix} k_x \zeta & 0 & k_x V_{\tau x} \\ 0 & k_y \zeta & k_y V_{\tau y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 0 & 2 \cdot V_{\tau x} \\ 0 & 2 \cdot 1 & 2 \cdot V_{\tau y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & V_{\sigma x} \\ 0 & 2 & V_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix}. \quad (4.41)$$

Die Veränderung der Kameraauflösung hat in diesem Beispiel zur Folge, dass es so wirkt, als wäre die Brennweite verdoppelt worden. Das würde bedeuten, dass sich die Kamera von der Bildebene entfernt hätte, jedoch verändert weder Kamera noch Bildebene ihre Position. Dennoch vergrößert oder verkleinert sich durch die Skalierung der Pixel effektiv die Bildgröße. Um zu überprüfen, ob die Änderung der Kameraauflösung auch eine Änderung der Epipolaren Bedingungen mit sich führt, werden wieder die Gleichungen der Epipolaren Bedingungen in Kapitel 3 genauer betrachtet.

Die Fundamentalmatrix beinhaltet, sowohl die intrinsischen als auch die extrinsischen Parameter, um von  $F$  auf  $E$  zu kommen, müssen die intrinsischen Kameraparameter bekannt sein. Mit bekannten  $K$  und  $K'$  gilt, dass

$$F = K'^{-T} R' \left[ \vec{C}'_\delta - \vec{C}_\delta \right]_\times R^T K^{-1} \quad (4.42)$$

$$E = K'^T F K \quad (4.43)$$

$$E = K'^T (K'^{-T} R' \left[ \vec{C}'_\delta - \vec{C}_\delta \right]_\times R^T K^{-1}) K \quad (4.44)$$

$$\rightsquigarrow E = R' \left[ \vec{C}'_\delta - \vec{C}_\delta \right]_\times R^T. \quad (4.45)$$

Da bei der Bestimmung von  $E$  aus  $F$  die intrinsischen Kameraparameter eliminiert werden, haben unterschiedliche Auflösungen keine Auswirkung auf die essentielle Matrix. Folglich sollte das Ergebnis bei der Bestimmung der extrinsischen Kameraparameter unverändert sein. Um die Aufgestellte Theorie zu überprüfen, wurde im synthetischen Beispiel die Kameramatrix  $K'$  von  $C'$  modifiziert. Für  $C$  wurde  $\zeta_x = \zeta_y = 1$  definiert, so das für Kameramatrix  $K$  gilt

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (4.46)$$

Für die Kameramatrix  $K'$  von  $C'$  galt ursprünglich, dass  $K = K'$ . Die Auflösung von  $C'$  wird geändert indem die Skalierungsfaktoren  $k_x$  und  $k_y$  mit  $\zeta'_x$  und  $\zeta'_y$  multipliziert werden.

Für das Beispiel wurden drei verschiedenen Auflösungen getestet.  $K'_1$  mit  $\zeta'_x \cdot 2$  und  $\zeta'_y \cdot 2$ ,  $K'_2$  mit  $\zeta'_x \cdot 3.2$  und  $\zeta'_y \cdot 1.2$  und  $K'_e$  mit  $\zeta'_x \cdot 0.5$  und  $\zeta'_y \cdot 4.3$ . Da ursprünglich galt, dass  $\zeta'_x = \zeta'_y = 1$ , ergeben sich die folgenden Kameramatrizen für  $K'$ .

$$K'_1 = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (4.47)$$

$$K'_2 = \begin{bmatrix} 3.2 & 0 & 0 & 0 \\ 0 & 1.2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (4.48)$$

$$K'_3 = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 4.3 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (4.49)$$

Die Unterschiede der entstehenden Abbildungen in  $C'$  sind in den Abbildungen 4.14, 4.15, 4.16 und 4.17 zu sehen.

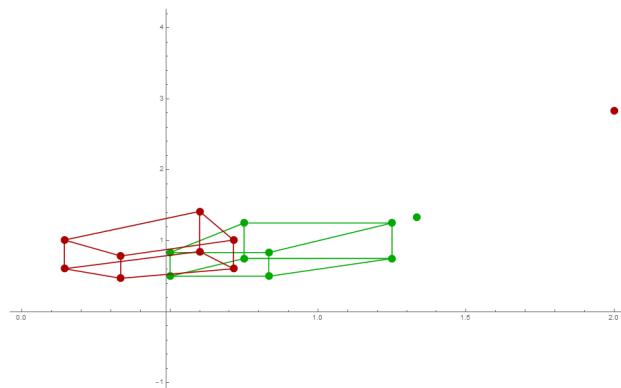


Abbildung 4.14:  $C$  und  $C'$  haben die selbe Auflösung eingestellt

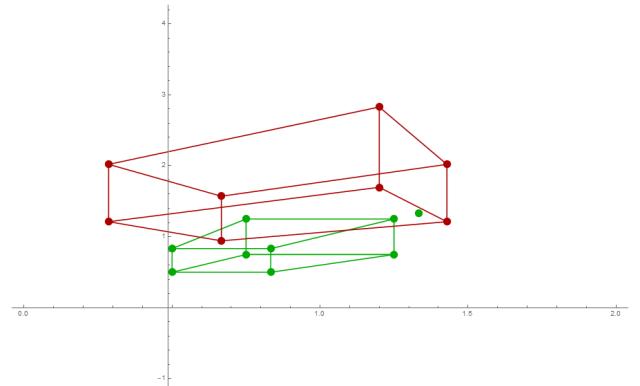


Abbildung 4.15:  $C$  und  $C'$  haben unterschiedliche Auflösungen eingestellt.  $C$  mit  $K$  und  $C'$  mit  $K'_1$

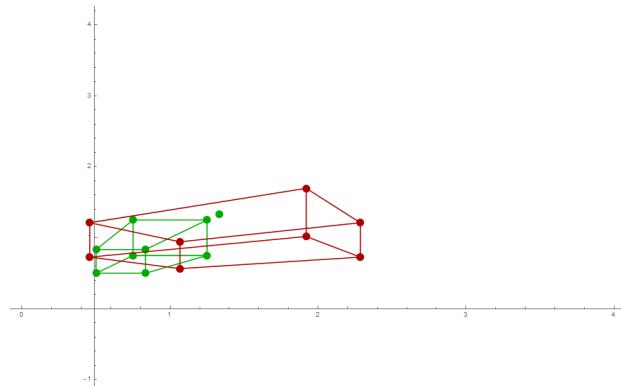


Abbildung 4.16:  $C$  mit  $K$  und  $C'$  mit  $K'_2$

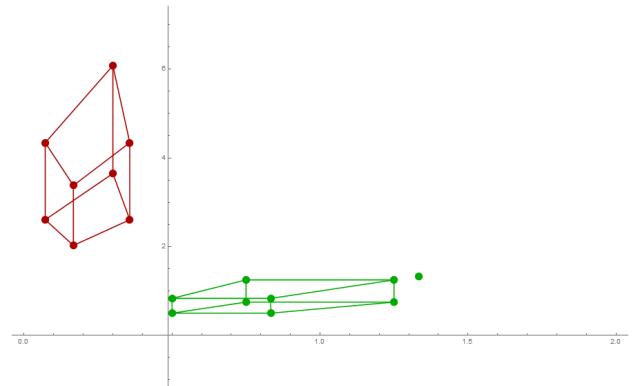


Abbildung 4.17:  $C$  mit  $K$  und  $C'$  mit  $K'_3$

Wird das synthetische Beispiel jeweils mit den drei verschiedenen modifizierten  $K'$  durchgerechnet, so ergeben sich für die essentielle Matrix folgende Ergebnisse.

$$\begin{aligned} \zeta'_x = 1, \zeta'_y = 1 : \quad E &= \begin{pmatrix} 0 & -0.5 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0.5 & 0 \end{pmatrix} | : 0.5 \rightsquigarrow E = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix} \\ \zeta'_x = 2, \zeta'_y = 2 : \quad E &= \begin{pmatrix} 0 & 0.756 & 0 \\ 0 & 0 & 1.069 \\ 0 & -0.756 & 0 \end{pmatrix} | : -0.756 \rightsquigarrow E = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix} \\ \zeta'_x = 3.2, \zeta'_y = 1.2 : \quad E &= \begin{pmatrix} 0 & 0.634 & 0 \\ 0 & 0 & 1.069 \\ 0 & -0.634 & 0 \end{pmatrix} | : -0.634 \rightsquigarrow E = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix} \\ \zeta'_x = 0.5, \zeta'_y = 4.3 : \quad E &= \begin{pmatrix} 0 & 0.442 & 0 \\ 0 & 0 & 1.069 \\ 0 & -0.442 & 0 \end{pmatrix} | : -0.442 \rightsquigarrow E = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix}. \end{aligned}$$

Wie beobachtet werden kann, werden trotz unterschiedlicher Kameraauflösungen für  $K'$ , immer die selbe essentielle Matrix im Algorithmus bestimmt. Zu Erinnerung, in Kapitel 3 wurde gezeigt, dass jedes Vielfache von  $E$  eine gültige Lösung ist. Somit gilt die Behauptung, dass die Kameraauflösung keine Auswirkung auf die Bestimmung der extrinsischen Kameraparameter hat, im synthetischen Beispiel als bestätigt. Als Vergleich kann die Abbildung 4.18, welche das Ergebnis der Szenenrekonstruktion mit  $K'_3$  als Kameramatrix für  $C'$  veranschaulicht, mit der rekonstruierten Szene in Abbildung 4.9 aus dem ersten Beispiel, betrachtet werden. Für die anderen Varianten von  $K'$  wurden ebenfalls die selbe 3D-Szene rekonstruiert.

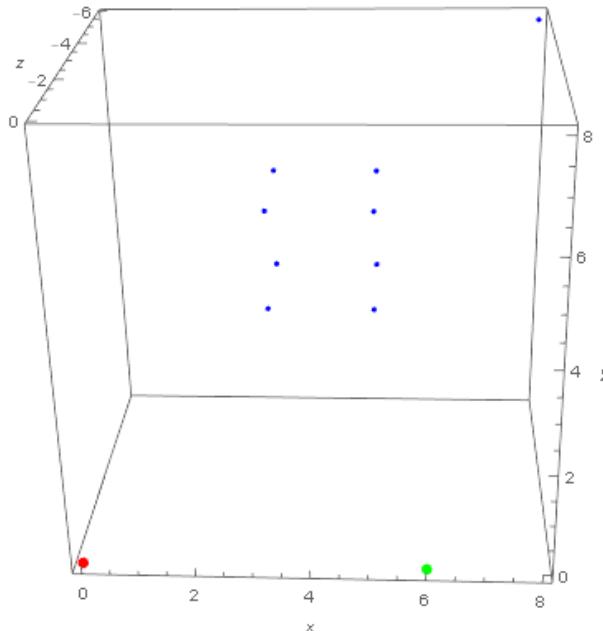


Abbildung 4.18: Die Abbildung zeigt die rekonstruierte Szenen des synthetischen Beispiels mit  $K'_3$  als intrinsische Parameter für  $C'$ .

# 5 Reelle Rekonstruktion

Der entwickelte Szenenrekonstruktionsalgorithmus wird im Folgenden anhand eines realen Stereoaufbaus getestet. Anders als bei den synthetischen Bilddaten im Beispiel zuvor, muss beim arbeiten mit reellen Stereobildern mit einer Fehleranfälligkeit der Bilddaten gerechnet werden. Liegt beispielsweise bei der Bestimmung von korrespondierenden Punkten eine Ecke zwischen zwei Pixel, so kann optisches Rauschen, welches in realen Aufnahmen präsent ist, dazu führen dass eine Ecke in gleichen Bildern an verschiedenen Pixel erkannt wird. Diese Abweichung der Punktekorrespondenz führt dazu, dass die in Kapitel 3.2 definierten epipolaren Bedingungen nicht mehr zur Gänze erfüllt werden. In diesem Kapitel wird hauptsächlich darauf eingegangen, welche Auswirkungen diese Ungenauigkeiten auf die Bestimmung der Fundamentalmatrix, der essentiellen Matrix, der extrinsischen Kameraparameter und der anschließenden Rekonstruktion der Szene durch Triangulierung haben und wie man ihnen entgegenwirken kann. In Abbildung 5.1, ist der Arbeitsprozess für den reellen Fall zu sehen. Der Ablauf im Allgemeinen bleibt beständig.

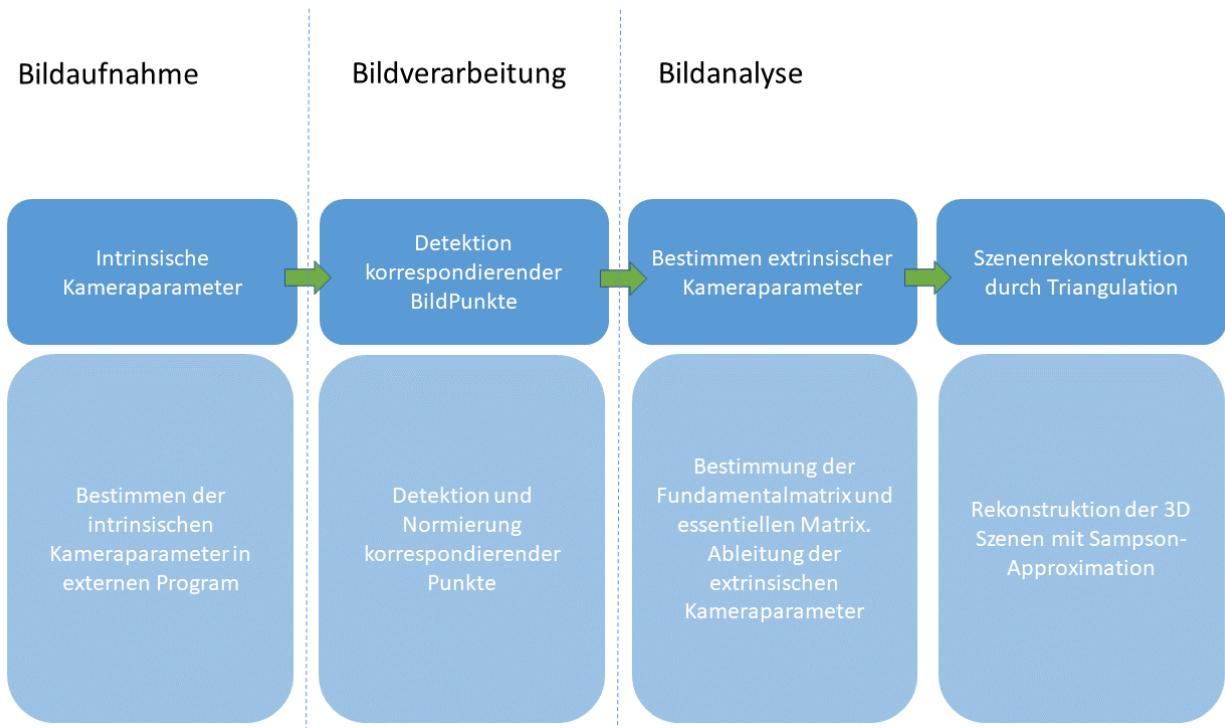


Abbildung 5.1: Ablaufdiagramm für die reelle Rekonstruktion

Zu Beginn wird der Stereoaufbau vorgestellt. Danach folgt die Korrespondenzanalyse, in welcher zwei Möglichkeiten aufgeführt werden, wie Punktekorrespondenzen aus den Stereoaufnahmen gewonnen werden können. Der Normierte-Acht-Punkt-Algorithmus, stellt eine für reale Bilddaten leicht veränderte Fassung des bereits bekannten Acht-Punkt-Algorithmus vor. Mit dessen Hilfe ist es möglich die Auswirkungen der Abweichungen in den Punktekorrespondenzen auf ein Minimum zu reduzieren. Nach der Bestimmung der Fundamentalmatrix durch den Normierten-Acht-Punkt-Algorithmus, wird diese auf ihre Gültigkeit kontrolliert. Hierzu werden vor allem die Singulärwerte der Fundamentalmatrix genauer betrachtet und welche Auswirkungen sie auf die epipolaren Bedingungen haben. Als nächstes wird die aus der Fundamentalmatrix bestimmte essentielle Matrix überprüft. Auch hier wird auf die Eigenschaften der Singulärwerte der essentiellen Matrix eingegangen.

Danach wird eine Triangulierungsmethode nach *Hartley & Zisserman* [2] vorgestellt, die über ein Näherungsverfahren eine Triangulation zwischen ungenauen korrespondierenden Punkten ermöglicht. Der letzte Abschnitt des Kapitels befasst sich dann mit der reellen Rekonstruktion bei unterschiedlichen Kameraauflösungen.

## 5.1 Stereoaufbau

Für die Stereobildaufnahme wurde eine Szene vor zwei nebeneinander platzierten Kameras aufgebaut. Beide Kameras wurden auf die Szene gerichtet. Abbildung 5.2 zeigt den entstandenen Stereoaufbau.



Abbildung 5.2: Kamera eins  $C$  befindet im Bild links, Kamera zwei  $C'$  befindet sich rechts im Bild. Auf dem Tisch zwischen den Kameras ist die in den Abbildungen 5.4 und 5.3 abgebildete Szene zu sehen. Beide Kameras sind auf die Szene gerichtet.

Die auf Abbildung 5.2 zu sehende linke Kamera wurde als primäre Kamera definiert. Die extrinsischen Kameraparameter für  $C'$ , werden dahingehend realiv zu  $C$  bestimmt. Das Kamerakoordinatensystem  $(C, \beta)$  entspricht somit gleich dem Weltkoordinatensystem  $(O, \delta)$ . Kamera zwei mit  $(C', \beta')$  befindet sich auf der Abbildung rechts. Für beide Kameras wurden in einem externen Programm die intrinsischen Kameraparameter  $K$  und  $K'$  bestimmt. An den Stereoaufnahmen der Szene wird eine Korrespondenzanalyse durchgeführt.

## 5.2 Korrespondenzanalyse

Für die Detektion von Punktekorrespondenzen bei Stereoaufnahmen einer dreidimensionalen Szene wurde eine bereits existierende Funktion von Mathematica genutzt[22]. Die Funktion basiert auf dem Prinzip eines SURF-Algorithmus. SURF ist die Kurzform für *Speeded Up Robust Features* und ist ein rotations- und skalenvarianter Punkte Detektor und Deskriptor[17, 24]. Es werden Punkte an markanten Stellen in beiden Bildern detektiert, wie beispielsweise Eckpunkte oder Kanten. Die Umgebung eines jeden gefundenen Punktes wird durch einen Merkmalsvektor, dem Deskriptor, beschrieben. Die Deskriptoren beider Bilder werden abgeglichen und gleiche Punkte werden als korrespondierende Punkte gekennzeichnet[17, 24]. Die Abbildungen 5.4 und 5.3 zeigen die Ergebnisse nach der Anwendung des SURF-Algorithmus auf das Stereobildpaar.

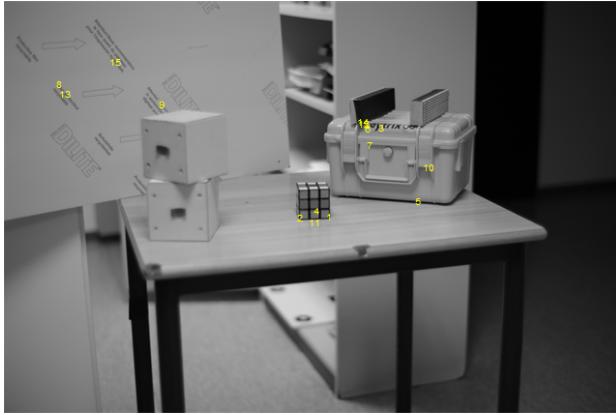


Abbildung 5.3: Aufnahme von Kamera  $C$

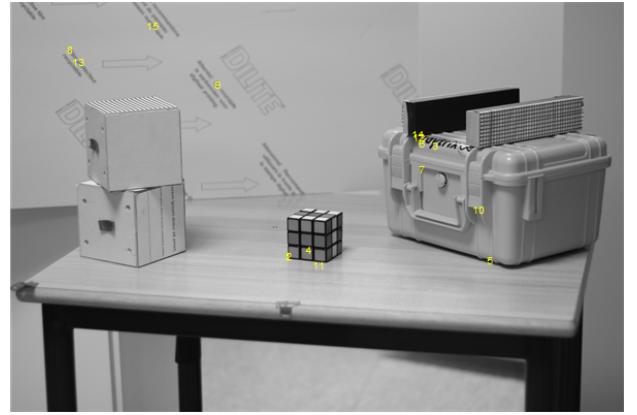


Abbildung 5.4: Aufnahme von Kamera  $C'$

Die mit dem *SURF*-Algorithmus gefundenen Punkte sind mit den gelben Ziffern in den Abbildungen 5.4 und 5.3 gekennzeichnet. Die Abbildung 5.4 zeigt das Bild von  $C$ , die Abbildung 5.3 zeigt das Bild von  $C'$ .

Die Detektion von Punktekorrespondenzen mit Detektionsalgorithmen, wie beispielsweise dem angewendeten *SURF*-Algorithmus, können immer Fehler und Abweichungen mit sich bringen. Die Ursprünge der Fehler können sowohl durch den Algorithmus als auch durch Fehler, wie Bildrauschen, in den Aufnahmen selbst entstehen. Diese Fehler wirken sich auf die Bestimmung der Abbildungsvorschriften  $F$  und  $E$  aus und somit auch auf die Genauigkeit der Szenenrekonstruktion[2]. Im Folgenden werden sowohl die Fehler als auch Methoden für deren Minimierung vorgestellt.

Eine eigens implementierte alternative für die Korrespondenzanalyse zwischen Stereoaufnahmen eines zweidimensionalen Schachbretts wird in Kapitel 7 vorgestellt.

### 5.3 Normierter-Acht-Punkt-Algorithmus

Trotz das der Acht-Punkt-Algorithmus eine einfache Methode zur Bestimmung der Fundamentalmatrix bietet, ist er sehr instabil sobald Fehler wie Ungenauigkeiten in Punktekorrespondenzen auftreten[2, 25].

Die Ausmaße der Fehler lässt sich anhand der Kondition der Koeffizientenmatrix  $A$  genauer feststellen. Als Kondition wird die Abhängigkeit der Lösung eines Problems von der Störung der Eingangsdaten beschrieben[16, 26, 27]. Die Kondition lässt sich durch Bestimmung des kleinsten Eigenvektors der Matrixmultiplikation der Koeffizientenmatrix  $A$  mit ihrer Transponierten  $A^T$  herausfinden. Die Matrix  $AA^T$  wird in die Matrizen  $UDU^T$  zerlegt, wobei  $U$  eine orthogonale und  $D$  eine diagonale Matrix ist. Die Diagonaleinträge von  $D$  sind in einer nicht ansteigenden Reihenfolge, woraus resultiert, dass der kleinste Singulärwert von  $D$  mit der letzten Spalte von  $U^T$  korrespondiert. Die letzte Spalte von  $U^T$  ist gleich dem kleinsten Eigenvektor von  $AA^T$ [16, 26]. Wird angenommen, dass  $AA^T$  eine  $9 \times 9$ -Matrix ist, so ergeben die Diagonaleinträge  $d_1/d_9$  den Wert der Kondition. Je größer die Kondition ist, desto größer wirken sich schon kleinste Abweichungen der reinkommenden Bilddaten, auf die aus  $A$  bestimmte Matrix  $F$  aus. Erkenntlich wird eine schlechte Kondition zum Beispiel an einem Ungleichgewicht der Matrixeinträge von  $F$ [16]. Das bedeutet, dass sich die Matrixeinträge stark in ihren Größeneinheiten unterscheidet.

$$F = \begin{pmatrix} 10^{-8} & 10^{-6} & 10^{-4} \\ 10^{-7} & 10^{-8} & 10^{-3} \\ 10^{-4} & 10^{-3} & 10^{-2} \end{pmatrix}. \quad (5.1)$$

Das führt dazu, dass die epipolare Bedingung mit  $m_\sigma^T F m_\sigma = 0$  schon bei kleinsten Abweichungen der Punktekorrespondenzen große Fehler errechnet. Die schlechte Kondition liegt laut *Hartley & Zisserman* an der weitflächigen Verteilung der homogenen Bildkoordinaten im Raum. Die heutigen Kameras können Bilder mit bis zu mehreren tausend Pixel aufnehmen. Ein Bildpunkt kann somit beispielsweise wie folgt aussehen

$$m_\sigma = \begin{pmatrix} 1150 \\ 2193 \\ 1 \end{pmatrix}. \quad (5.2)$$

$m_{\sigma x}$  und  $m_{\sigma y}$  liegen in einem Bereich der sich um mehr als 1000 Pixel unterscheidet. Das bedeutet, dass sich die ersten beiden Koordinateneinträge um einen viel größeren Zahlbereich befinden als ihre dritte Komponente. Um die Kondition möglichst klein zu halten, werden die Bildkoordinaten beider Bilder normiert[16, 25]. Normierte Koordinaten befinden sich in einem ausgeglicheneren Wertebereich und liegen nicht mehr weit verteilt im Raum[16]. Ihre Einträge entsprechen dem folgenden Beispiel

$$m_\sigma = \begin{pmatrix} 1.5 \\ 1.193 \\ 1 \end{pmatrix}. \quad (5.3)$$

Die Einträge der entstehende Fundamentalmatrix haben dann die Form

$$F = \begin{pmatrix} 10^{-3} & 10^{-2} & 10^{-2} \\ 10^{-2} & 10^{-3} & 10^{-2} \\ 10^{-2} & 10^{-2} & 10^{-2} \end{pmatrix}. \quad (5.4)$$

Das hat zur Folge, dass die epipolare Bedingung weniger stark auf ungenaue Punktekorrespondenzen reagiert. Die in Literaturquellen, vorgeschlagene Methode zur Normierung besteht darin den Schwerpunkt aller Punkte in den Ursprung des Sensorskoordinatensystems zu verschieben und die Bildpunkt so zu skalieren, dass ihr durchschnittlicher Abstand zum Ursprung  $\sqrt{2}$  beträgt[2, 3, 25].

Für die Normierung wird pro Bild eine Transformationsmatrix  $T$  und  $T'$  definiert. Die Matrizen beinhalten sowohl eine Skalierung als auch eine Translation. Die Bestimmung der Matrix  $T$  wird im Folgenden aufgezeigt. Zuerst wird der Schwerpunkt  $s$  mit  $s = \begin{pmatrix} s_x \\ s_y \end{pmatrix}$  der Punktemenge  $p_n$  mit  $p_n = \begin{pmatrix} p_{nx} \\ p_{ny} \end{pmatrix}$  berechnet, indem der Mittelwert aller Punkte  $p_n$  berechnet wird.

$$\begin{pmatrix} s_x \\ s_y \end{pmatrix} = \frac{1}{n} \sum_{i=1}^n \begin{pmatrix} p_{ix} \\ p_{iy} \end{pmatrix} \quad (5.5)$$

Danach wird  $s$  in den Ursprung verschoben. Die Punkte  $p_n$  werden ebenfalls um den Wert von  $s$  verschoben  $p'_n = p_n - s$ . Der Mittelwert aus den um  $s$  verschobenen Punkten  $x'_n$  ergibt den neuen Schwerpunkt  $s_0$  im Koordinatenursprung. Als nächstes wird die Distanz jedes Punktes von  $p'_n$  zu  $s_0$  berechnet und der Mittelwert aller Distanzen, hier mit  $d$  bezeichnet, berechnet. Die Matrix  $T$  und  $T'$  haben dann die folgende Form:

$$T = \begin{pmatrix} \frac{\sqrt{2}}{d} & 0 & -s_x \\ 0 & \frac{\sqrt{2}}{d} & -s_y \\ 0 & 0 & 1 \end{pmatrix} \quad (5.6)$$

$$T' = \begin{pmatrix} \frac{\sqrt{2}}{d'} & 0 & -s'_x \\ 0 & \frac{\sqrt{2}}{d'} & -s'_y \\ 0 & 0 & 1 \end{pmatrix} \quad (5.7)$$

Die originalen Bildpunkte des Stereobildpaars, werden mit den Matrizen  $T$  und  $T'$  verrechnet. Mit den Normierten Bildkoordinaten wird dann mit dem Acht-Punkt-Algorithmus eine Fundamentalmatrix  $\hat{F}$  bestimmt[2, 16, 3, 25]. Ist  $\hat{F}$  aus den normierten Koordinaten bestimmt, wird sie mit  $T$  und  $T'$  wieder denormalisiert.

$$F = T'^T \hat{F} T \quad (5.8)$$

### 5.3.1 Singularität der Fundamentalmatrix

Die aus den Punktekorrespondenzen bestimmte Fundamentalmatrix muss eine singuläre Matrix mit Rang 2 sein[2, 5, 3]. Die Singularität der Fundamentalmatrix sorgt zum einen dafür das ihr rechter und linker Kern jeweils eine eindeutige Lösung für den Epipol des jeweiligen Bildes ergibt und die Epipolarlinien, welche mit  $l = Fm_\sigma$  und  $l' = F^T m'_\sigma$ , berechnet werden, auch alle durch diese Epipole verlaufen[2]. Durch Ungenauigkeiten in korrespondierenden Bildpunkten kann es dazu kommen, dass die aus dem Normierten-Acht-Punkt-Algorithmus bestimmte Fundamentalmatrix  $\hat{F}$  in ihrem Rang steigt und somit keine singuläre Matrix mehr ist. Matrizen sind singulär, wenn sie keine Inverse besitzen, was durch das verschwinden der Determinante nachgewiesen werden kann[28]. Die Determinante von  $\hat{F}$  muss Null sein, damit sie eine singuläre Matrix ist.

Ist  $\hat{F}$  durch den erhöhten Rang keine singuläre Matrix, so ergeben der linke und der rechte Kern von  $\hat{F}$  keine eindeutigen Lösungen mehr für  $e$  und  $e'$  und die Epipolarlinien in beiden Bildern verlaufen dementsprechend auch nicht mehr durch genau einen Punkt, wie man in den Abbildungen 5.5 und 5.6 erkennen kann. Die Abbildungen bilden Epipolarlinien aus dem Stereobildpaar 5.3 und 5.4 ab. Die Ursache des erhöhten Ranges von  $\hat{F}$  ist in ihren Singulärwerten von  $F$  zu finden.

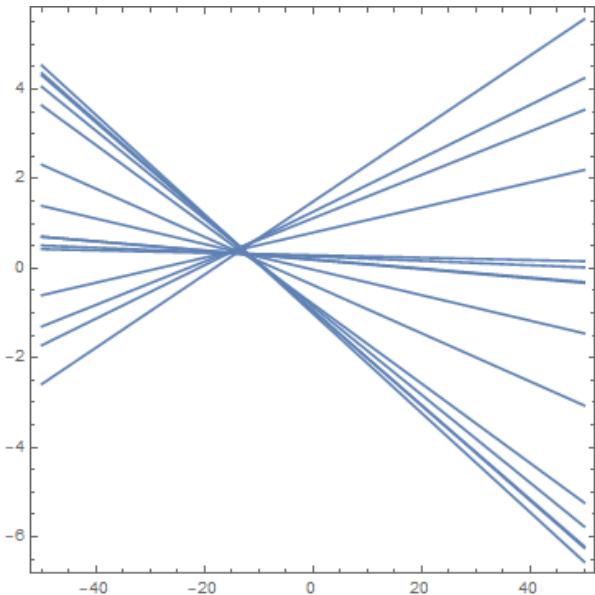


Abbildung 5.5: Epipolarlinien ohne *Epipolar constraint* im Bild der Canon 6D

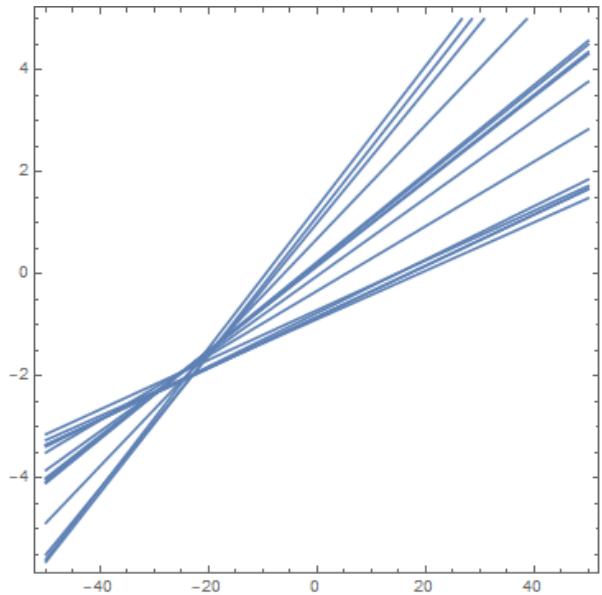


Abbildung 5.6: Epipolarlinien ohne *Epipolar constraint* im Bild der Canon 60D

Um mit dem Algorithmus weiter verfahren zu können, muss die Singularität in der noch normierten Fundamentalmatrix  $\hat{F}$  erzwungen werden[2]. Hierfür wird eine Singulärwertzerlegung an  $\hat{F}$  durchgeführt, so dass  $\hat{F} = U\Sigma V^T$  zerlegt wird.  $\Sigma$  beinhaltet in einer Diagonalmatrix die Singulärwerte  $D = \text{diag}(r, s, t)$ . Die Diagonaleinträge erfüllen die Bedingung, dass  $r \geq s \geq t$  gilt. Damit  $\hat{F}$  zu einer singulären Matrix wird, muss für die Diagonaleinträge jedoch gelten, dass  $\Sigma = \text{diag}(r, s, 0)$  ist. Diese Bedingung wird erzwungen, indem der letzte Eintrag  $t$  auf  $t = 0$  gesetzt wird. Die so modifizierte

Fundamentalmatrix mit  $\bar{F} = U \text{diag}(r, s, 0) V^T$  wird dann wieder zusammengesetzt. Die Fundamentalmatrix  $\bar{F}$  besitzt jetzt einen Rang 2 und ist singulär[2].  $\bar{F}$  minimiert die Frobenius Norm  $\| \hat{F} - \bar{F} \|$  und ist die nächste zum ursprünglichen  $\hat{F}$  liegende singuläre Matrix von Rang 2[2, 16, 28].

Werden aus  $\bar{F}$  der rechte und linke Kern bestimmt, so ergeben sich eindeutige Lösungen für  $e$  und  $e'$  und die Epipolarlinien  $l$  und  $l'$  verlaufen jeweils durch ihre entsprechenden Epipole[2]. Die Abbildungen 5.7 und 5.8 zeigen die Auswirkung der erzwungenen Singularität von  $\bar{F}$  auf dem Stereobildpaar 5.3 und 5.4. Die Abbildungen 5.9 und 5.10 zeigen die selben Epipolarlinien nur ist  $\bar{F}$  mit  $T$  und  $T'$  zu  $F$  denormiert worden.

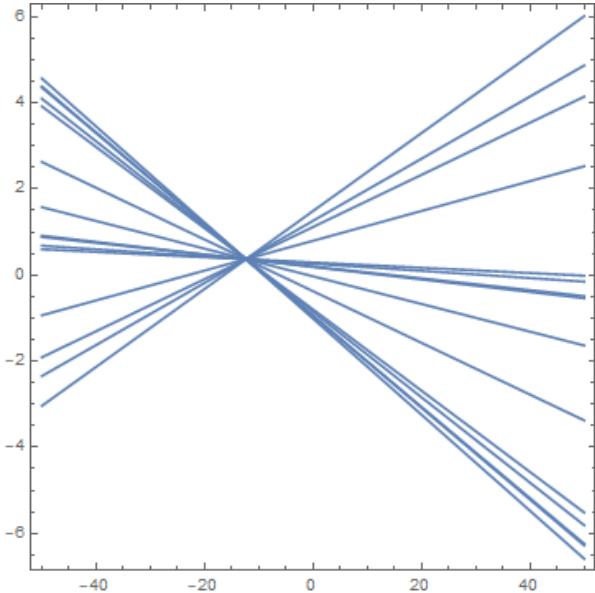


Abbildung 5.7: Die Abbildung zeigt, dass die Epipolarlinien auf der Aufnahme von  $C$ , nach dem Erzwingen der Singularität in der normierten  $\hat{F}$ , alle durch den Epipol  $e$  verlaufen

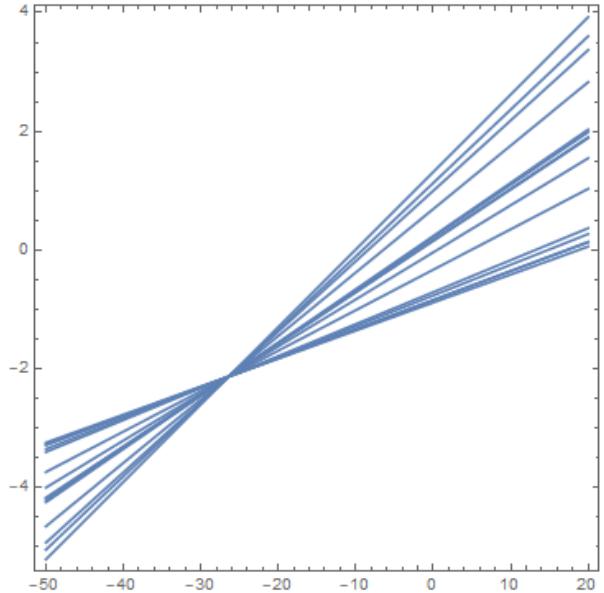


Abbildung 5.8: Die Abbildung zeigt, dass die Epipolarlinien auf der Aufnahme von  $C'$ , nach dem Erzwingen der Singularität in der normierten  $\hat{F}$ , alle durch den Epipol  $e'$  verlaufen

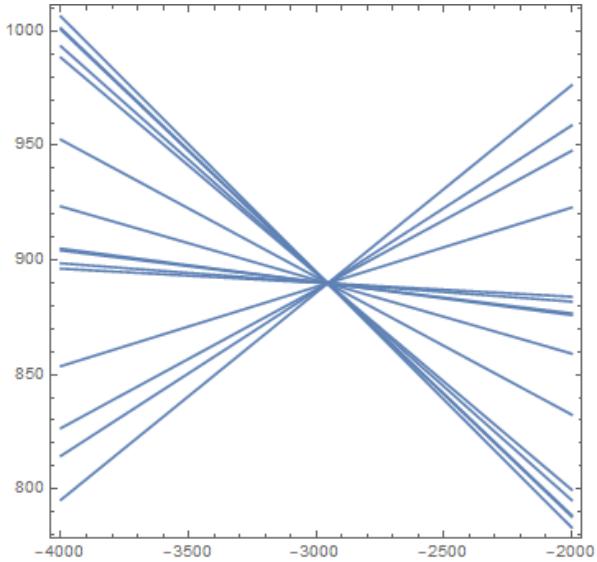


Abbildung 5.9: Die Abbildung zeigt die Epipolarlinien in  $C$  nachdem die Fundamentalmatrix  $\bar{F}$  mit  $F = T'\bar{F}T$  denormalisiert wurde

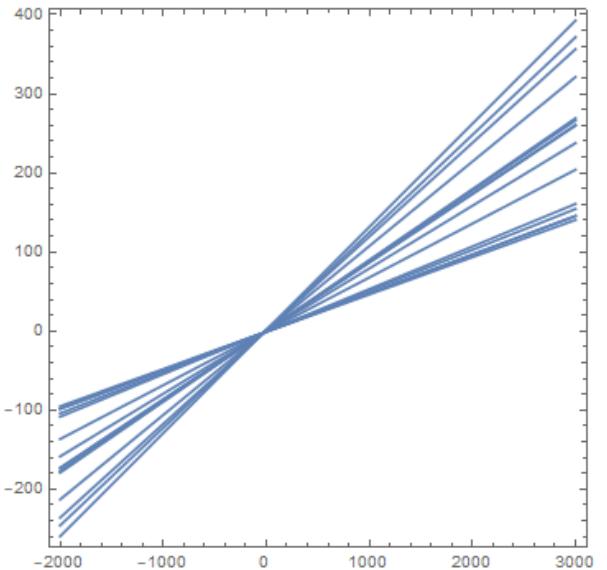


Abbildung 5.10: Die Abbildung zeigt die Epipolarlinien in  $C'$  nachdem die Fundamentalmatrix  $\bar{F}$  mit  $F = T'\bar{F}T$  denormalisiert wurde

### 5.3.2 Singulärwerte der essentiellen Matrix

Die essentielle Matrix wird im entwickelten Algorithmus aus der Fundamentalmatrix  $F$  bestimmt. Da zuvor die Singularität von  $F$  erzwungen wurde, ist die essentielle Matrix ebenfalls eine Matrix von Rang 2[2]. Im synthetischen Beispiel wurde gezeigt, dass für die Bestimmung der extrinsischen Kameraparameter für  $E$  gelten muss, dass für ihre Singulärwerte  $\Sigma = \text{diag}(1, 1, 0)$  sind.

Die Ungenauigkeit der Punktekorrespondenzen, kann auf  $E$  die Auswirkung haben, dass ihre Singulärwerte die Form  $\Sigma = \text{diag}(a, b, c)$  mit  $a \geq b \geq c$  annehmen. Eine Matrix gilt nur dann als essentielle Matrix, wenn zwei ihrer Singulärwerte gleich sind  $a = b$  und für den dritten gilt  $c = 0$ . Um diese Bedingung zu erzwingen, wird diejenige essentielle Matrix  $\hat{E}$  gesucht, welche sich laut der Frobenius Norm am nächsten an der ursprünglichen  $E$  befindet[2, 3]. Diese Matrix lässt sich aus  $E = U\Sigma V^T$  bestimmen, indem eine neue essentielle Matrix  $\hat{E} = U\hat{\Sigma}V^T$  mit  $\hat{\Sigma} = \text{diag}(\frac{a+b}{2}, \frac{a+b}{2}, 0)[2]$  gebildet wird.

Nach erzwingen der singulären Bedingung  $\hat{\Sigma} = \text{diag}(\frac{a+b}{2}, \frac{a+b}{2}, 0)$ , ist  $E$  wieder eine gültige essentielle Matrix und der Algorithmus kann mit der Bestimmung der extrinsischen Kameraparameter, wie in Kapitel 4 gezeigt, fortfahren.

Das Erzwingen der singulären Bedingungen für  $F$  und  $E$  sind Näherungsverfahren, in welchen die Auswirkungen der Fehlerhaften Punktekorrespondenzen minimiert werden. Nur wenn  $F$  und  $E$  ihre singulären Bedingungen erfüllen, können die extrinsischen Kameraparameter bestimmt und die Szene rekonstruiert werden[2].

## 5.4 Szenenrekonstruktion mit Sampson-Approximation

Aufgrund der Ungenauigkeit der korrespondierenden Punkte ist es nicht möglich die 3D-Objektpunkte durch einfache Rückprojektion der Bildpunkte zu rekonstruieren. Liegt der zu  $m_\sigma$  korrespondierende Bildpunkt  $m'_{\sigma'}$  nicht ganz genau auf der zu  $m_\sigma$  korrespondierenden Epipolarlinie, so ist die in Kapitel 3 aufgestellte epipolare Bedingung aus Gleichung 3.18 nicht mehr erfüllt. Durch einsetzen der detektierten korrespondierenden Punkte  $m_\sigma$  und  $m'_{\sigma'}$  in die Gleichung

$$m'^T \mathbf{F} m_\sigma = 0, \quad (5.9)$$

kommt ein Wert  $\neq 0$  heraus. Je weiter der Wert von Null abweicht, desto ungenauer ist die Korrespondenz beider Bildpunkte. Dies führt dazu, dass sich bei der Rückprojektion die Linien der Bildpunkte  $m_\sigma$  und  $m'_{\sigma'}$  nicht im Raum treffen, sondern windschief zueinander liegen. Die Abbildungen 5.11 und 5.12 veranschaulichen die Konsequenz von ungenauen Punktekorrespondenzen.

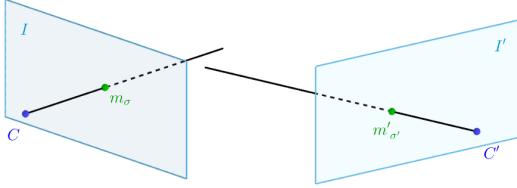


Abbildung 5.11: a) Windschiefe Geraden

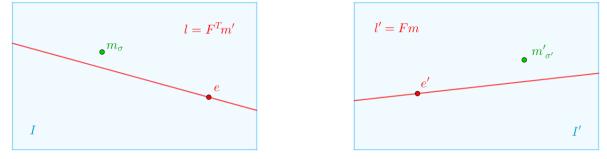


Abbildung 5.12: b) Epipolare Bedingungen werden nicht erfüllt

Abbildung 5.13: a) Die rückprojizierten Strahlen der ungenauen korrespondierenden Punkte  $m_\sigma$  und  $m'_{\sigma'}$ , sind Windschief zueinander und treffen sich nicht in einem Punkt  $M_{\delta,0}$  im Raum.  
 b) Die korrespondierenden Bildpunkte  $m_\sigma$  und  $m'_{\sigma'}$ , erfüllen nicht die epipolare Bedingung. Die Epipolarlinie  $l' = F m$  ist die korrespondierende Epipolarlinie zu  $m_\sigma$  und  $l = F^T m'$  ist die korrespondierende Epipolarlinie zu  $m'_{\sigma'}$ . Da weder  $m_\sigma$  noch  $m'_{\sigma'}$  auf der Epipolarlinie zum jeweils korrespondierenden Punkt liegen, kommt es zu keinem Schnittpunkt der rückprojizierten Strahlen

Um trotz der ungenauen korrespondierenden Punkte eine Triangulation zu ermöglichen, wird ein Verfahren voran geschaltet, welches zwei Punkte  $\hat{m}_\sigma$  und  $\hat{m}'_{\sigma'}$  sucht, die möglichst nah an den ursprünglichen Punkten  $m_\sigma$  und  $m'_{\sigma'}$  liegen und gleichzeitig die epipolare Bedingung  $\hat{m}'_{\sigma'}^T \mathbf{F} \hat{m}_\sigma = 0$  erfüllen.  $\hat{m}_\sigma$  und  $\hat{m}'_{\sigma'}$  sollen durch Minimierung einer Funktion  $C$  bestimmt werden, welche die Distanz  $d$  zwischen  $m_\sigma$  und  $\hat{m}_\sigma$  und  $m'_{\sigma'}$  und  $\hat{m}'_{\sigma'}$  minimiert. Für die Minimierung wird das Verfahren der Sampson-Approximation gewählt[2].

$$C(m, m') = d(m, \hat{m})^2 + d(m', \hat{m}')^2 \quad (5.10)$$

Die optimalen Punkte  $\hat{m}$  und  $\hat{m}'$  liegen auf den korrespondierenden Epipolarlinien  $\hat{l}$  und  $\hat{l}'$  am Fuße des Lots, welches von den ursprünglich projizierten Punkten  $m$  und  $m'$  auf die Epipolarlinien  $\hat{l}$  und  $\hat{l}'$  gefällt wird[2], wie in Abbildung 5.14 zu sehen ist. Jedes andere Punktpaar auf  $\hat{l}$  und  $\hat{l}'$  würde die epipolare Bedingung erfüllen, jedoch minimieren nur  $m_{\sigma\perp}$  und  $m'_{\sigma'\perp}$  die quadratischen Distanzen  $d(m_\sigma, \hat{m}_\sigma)^2$  und  $d(m'_{\sigma'}, \hat{m}'_{\sigma'})^2$  in der Funktion  $C$ . Gesucht wird also der geringste Abstand von  $m_\sigma$  zu  $\hat{l}$  und  $m'_{\sigma'}$  zu  $\hat{l}'$ . Die Funktion  $C$  kann dementsprechend umformuliert werden in

$$C(m, m') = d(\hat{m}_\sigma, \hat{l})^2 + d(\hat{m}'_{\sigma'}, \hat{l}')^2. \quad (5.11)$$

Aus allen möglichen Epipolarlinien, welche  $\hat{l}$  und  $\hat{l}'$  annehmen können, entspricht immer der senkrechte Abstand von  $m_\sigma$  und  $m'_{\sigma'}$  zur jeweiligen Epipolarlinie der minimalen Distanz. Es soll jedoch genau das Epipolarlinienpaar gewählt werden, welches die Funktion  $C$  minimal werden lässt[2].

Im ersten Schritt werden die Epipolarlinien parametrisiert, so dass eine Linie als  $\hat{l}(t)$  geschrieben werden kann. Durch die Parametrisierung der Epipolarlinien, kann die Funktion  $C$  als eine Funktion von  $t$  umformuliert werden.

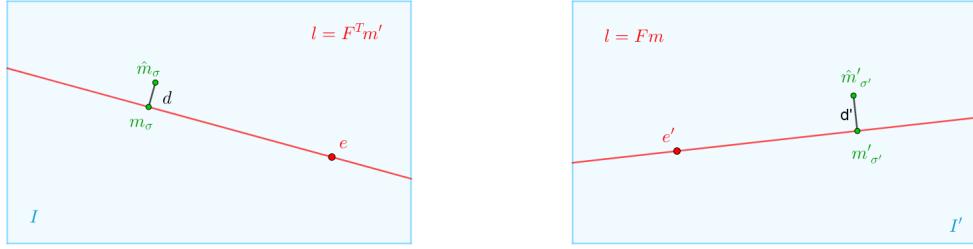


Abbildung 5.14: Die Abbildung zeigt die zwei korrespondierenden Epipolarlinien  $\hat{l}$  und  $\hat{l}'$  mit den gesuchten Punkten  $\hat{m}_\sigma$  und  $\hat{m}'_{\sigma'}$

$$C(m_\sigma, m'_{\sigma'}) = d(m_\sigma, \hat{l}(t))^2 + d(m'_{\sigma'}, \hat{l}'(t))^2 \quad (5.12)$$

Um die Minimierung zu vereinfachen, werden zu Beginn die Bildpunkte  $m_\sigma$  und  $m'_{\sigma'}$  mit jeweils einer Matrix  $T$  und  $T'$  in den Ursprung  $(0, 0, 1)^T$  verschoben.

$$T = \begin{bmatrix} 1 & 0 & -m_{\sigma x} \\ 0 & 1 & -m_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix} \rightsquigarrow \bar{m}_\sigma = T \cdot m_\sigma \quad (5.13)$$

$$T' = \begin{bmatrix} 1 & 0 & -m'_{\sigma' x} \\ 0 & 1 & -m'_{\sigma' y} \\ 0 & 0 & 1 \end{bmatrix} \rightsquigarrow \bar{m}'_{\sigma'} = T' \cdot m'_{\sigma'} . \quad (5.14)$$

Die Fundamentalmatrix  $F$  wird ebenfalls mit  $T$  und  $T'$  transformiert, sodass sie an die verschobenen Punkte  $\bar{m}_\sigma$  und  $\bar{m}'_{\sigma'}$  angepasst ist

$$\bar{F} = T'^{-T} F T^{-1} . \quad (5.15)$$

Der rechte und linke Kern von  $\bar{F}$  ergeben die Epipole  $\bar{e}$  und  $\bar{e}'$ . Angenommen  $f$  und  $f'$  seien genau Null, so liegen die Epipole  $e = (1, 0, f)^T$  und  $e' = (1, 0, f')^T$  im unendlichen. Ist dies der Fall so hat  $\bar{F}$  für welche dann gilt, dass  $\bar{F}(1, 0, f)^T = (1, 0, f')\bar{F} = 0$  eine spezielle Form[2].

$$\bar{F} = \begin{pmatrix} ff'd & -f'c & -f'd \\ -fb & a & b \\ -fd & c & d \end{pmatrix} \quad (5.16)$$

$$\begin{pmatrix} ff'd & -f'c & -f'd \\ -fb & a & b \\ -fd & c & d \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ f \end{pmatrix} = \begin{pmatrix} ff'd + (-ff'd) \\ -fb + fb \\ -fd + fd \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (5.17)$$

$$(1 \ 0 \ f') \cdot \begin{pmatrix} ff'd & -f'c & -f'd \\ -fb & a & b \\ -fd & c & d \end{pmatrix} = \begin{pmatrix} ff'd + (-ff'd) \\ -f'c + f'c \\ -f'd + f'd \end{pmatrix} = (0 \ 0 \ 0) . \quad (5.18)$$

Im Realfall sind die Werte der Epipole  $\bar{e}$  und  $\bar{e}'$  nicht genau  $\bar{e} = (1, 0, f)^T$  und  $\bar{e}' = (1, 0, f')^T$ , sondern weichen in ihren Richtungen leicht ab. Des Weiteren sind die Epipole auch noch nicht normiert. Es folgt

zunächst die Normierung der Epipole, sodass  $e_1^2 + e_2^2 = 1$  und  $e_1'^2 + e_2'^2 = 1$  gilt. Danach werden  $e$  und  $e'$  mit zwei Rotationsmatrizen  $R$  und  $R'$  auf  $Re = (1, 0, e_3) = (1, 0, f)$  und  $R'e' = (1, 0, e'_3) = (1, 0, f')$  rotiert

$$R = \begin{bmatrix} e_1 & e_2 & 0 \\ -e_2 & e_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.19)$$

$$R' = \begin{bmatrix} e'_1 & e'_2 & 0 \\ -e'_2 & e'_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.20)$$

$\bar{F}$  wird dann mit  $\bar{F}_{Rot} = R'FR^T$  ersetzt. Die Einträge in  $\bar{F}_{Rot}$  haben nun die Form wie in Gleichung 5.16, mit  $f = e_3$ ,  $f' = e'_3$ ,  $a = \bar{F}_{Rot,22}$ ,  $b = \bar{F}_{Rot,23}$ ,  $c = \bar{F}_{Rot,32}$  und  $d = \bar{F}_{Rot,33}$ .

$$\bar{F}_{Rot} = \begin{pmatrix} e_3 e'_3 \bar{F}_{Rot,33} & -e'_3 \bar{F}_{Rot,32} & -e'_3 \bar{F}_{Rot,33} \\ -e_3 \bar{F}_{Rot,23} & \bar{F}_{Rot,22} & \bar{F}_{Rot,23} \\ -e_3 \bar{F}_{Rot,33} & \bar{F}_{Rot,32} & \bar{F}_{Rot,33} \end{pmatrix} \quad (5.21)$$

Verläuft eine Epipolarlinie durch einen Punkt  $(0, t, 1)^T$  und dem Epipol  $\bar{e} = (1, 0, f)^T$ , wird diese Epipolarlinie mit  $\hat{l}(t)$  bezeichnet. Das Kreuzprodukt dieser beiden Punkte beschreibt die Epipolarlinie  $\hat{l}(t)[2]$ .

$$\hat{l}(t) = \begin{pmatrix} 0 \\ t \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ f \end{pmatrix} = \begin{pmatrix} tf \\ 1 \\ -t \end{pmatrix} \quad (5.22)$$

Die quadratische Distanz dieser Linie zum Ursprung wird dann bezeichnet mit:

$$d(\bar{m}_\sigma, \hat{l}(t))^2 = \frac{t^2}{1 + (tf)^2} \quad (5.23)$$

Nach Hartley & Zisserman [2] werden Linien durch Vektoren der Form  $\vec{v} = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$  dargestellt.

Übersetzt in eine Koordinatengleichung ergibt sich

$$Ax + By + C = 0. \quad (5.24)$$

Die Gleichung 5.23 wird anhand der Koordinatengleichung hergeleitet. Für die Herleitung wird die Koordinatenform der Geraden zunächst in Normalform umgeschrieben

$$\vec{n} \cdot (\vec{x} - \vec{p}) = 0 \quad (5.25)$$

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix} \cdot \left( \vec{x} - \begin{pmatrix} 0 \\ C \\ B \end{pmatrix} \right) = 0 \quad (5.26)$$

Der Abstand  $\|\vec{v}\|$  eines Punktes zur Geraden 5.26 kann folgendermaßen berechnet werden.

$$\vec{v} = \frac{\vec{p} \cdot \vec{n}}{\vec{n} \cdot \vec{n}} \cdot \vec{n} \rightsquigarrow \frac{-C}{A^2 + B^2} \cdot \begin{pmatrix} A \\ B \end{pmatrix} \quad (5.27)$$

$$\|\vec{v}\| = \frac{|\vec{p} \cdot \vec{n}|}{\|\vec{n}\|^2} \cdot \|\vec{n}\| \rightsquigarrow \|\vec{v}\| = \frac{|\vec{p} \cdot \vec{n}|}{\|\vec{n}\|} \quad (5.28)$$

$$\Rightarrow |C| = |\vec{p} \cdot \vec{n}| \quad (5.29)$$

$$\Rightarrow \sqrt{A^2 + B^2} = \|\vec{n}\| \quad (5.30)$$

$$\|\vec{v}\| = \frac{|C|}{\sqrt{A^2 + B^2}} \quad (5.31)$$

Werden nun  $A, B$  und  $C$  mit den Werten der Geraden  $(tf, 1, -t)^T$  ersetzt, kann Gleichung 5.23 rekonstruiert werden.

$$A = tf, B = 1, C = -t, \vec{v} = d \quad (5.32)$$

$$d^2 = \frac{t^2}{\sqrt{((tf)^2 + 1^2)^2}} = \frac{t^2}{(tf)^2 + 1^2} = \frac{t^2}{1 + (tf)^2} \quad (5.33)$$

Um die zu  $\hat{l}(t)$  korrespondierende Epipolarlinie  $\hat{l}'(t)$  zu bestimmen, wird der Beispunkt  $(0, t, 1)^T$  und die Fundamentalmatrix  $\bar{F}_{Rot}$  multipliziert. Für die Übersichtlichkeit der Matrix  $\bar{F}_{Rot}$ , wird die Symbolschreibweise aus Gleichung 5.16 genutzt. Für die Einträge gelten die Definitionen aus Gleichung 5.21.

$$l'(t) = \bar{F}_{Rot}(0, t, 1)^T = (-f'(ct + d), at + b, ct + d)^T. \quad (5.34)$$

Für die quadratische Distanz  $d(\bar{m}'_{\sigma'}, l'(t))^2$  ergibt sich dann

$$d(\bar{m}'_{\sigma'}, l'(t))^2 = \frac{(ct + d)^2}{(at + b)^2 + f'^2(ct + d)^2}. \quad (5.35)$$

Die Funktion  $C$  kann jetzt in eine Funktion von  $s(t)$  umformuliert werden.

$$s(t) = \frac{t^2}{1 + (tf)^2} + \frac{(ct + d)^2}{(at + b)^2 + f'^2(ct + d)^2}. \quad (5.36)$$

Ein Minimum für  $s(t)$  kann beispielsweise durch Bestimmung der Minima und Maxima mit  $s'(t) = 0$  gefunden werden.

$$s'(t) = \frac{2t}{(1 + f^2 t^2)^2} - \frac{2(ad - bc)(at + b)(ct + t)}{((at + b)^2 + f'^2(ct + d)^2)^2}. \quad (5.37)$$

Werden die beiden Terme in  $s'(t)$  auf einen gemeinsamen Nenner gebracht und der Zähler dann gleich Null gesetzt, ergibt sich der folgende Ausdruck  $g(t)[2]$

$$g(t) = t((at + b)^2 + f'^2(ct + d)^2)^2 - (ad - bc)(1 + f^2 t^2)^2(at + b)(ct + d) \quad (5.38)$$

Funktion  $g(t)$  ist ein Polynom vom Grad 6. Das Minimum für  $s(t)$  ergibt sich aus einer der sechs möglichen Lösungen für  $t$  aus  $g(t)$ . Für die Bestimmung des Minimums werden nur die reellen Lösungen

für  $t$  in Betracht gezogen. Die reellen Lösungen für  $t$  aus  $g(t)$ , werden dann wieder in  $s(t)$  eingesetzt. Das  $t$ , welches durch einsetzen in  $s(t)$  den kleinsten Wert ergibt, ist das gesuchte Minimum  $t_{min}$ .

Mit  $t_{min}$  können die Epipolarlinien  $\hat{l}(t_{min}) = (t_{min}f, 1, -t)$  und  $\hat{l}'(t_{min}) = \bar{F}_{Rot}(0, t_{min}, 1)^T$  berechnet werden. Danach werden die zwei neuen Punkte  $\hat{m}_{\sigma Rot}$  und  $\hat{m}'_{\sigma' Rot}$  auf den Epipolarlinien  $\hat{l}(t_{min})$  und  $\hat{l}'(t_{min})$  bestimmt.  $\hat{m}_{\sigma Rot}$  und  $\hat{m}'_{\sigma' Rot}$  sind die Punkte auf der Epipolarlinie welche dem Ursprung am nächsten sind. Zur Erinnerung die Bildpunkte  $m_\sigma$  und  $m'_{\sigma'}$  wurden zu Beginn in den Ursprung verschoben. Der Punkt, welcher vom Ursprung aus am nächsten auf einer Linie  $(\lambda, \mu, v)$  liegt, kann mit  $(-\lambda \cdot v, -\mu \cdot v, \lambda^2 + \mu^2)$  berechnet werden[2]. Somit gilt

$$\hat{l} = (tf, 1, -t) \quad (5.39)$$

$$\hat{m}_{\sigma Rot} = (-(tf) \cdot v, -1 \cdot v, (tf)^2 \cdot 1^2). \quad (5.40)$$

Nachdem zu beiden Linien  $\hat{l}$  und  $\hat{l}'$  der jeweils nächste Punkte  $\hat{m}_{Rot}$  und  $\hat{m}'_{Rot}$  vom Ursprung aus gefunden wurde, werden diese mit  $T$ ,  $T'$ ,  $R$  und  $R'$  wieder an ihre Ausgangsposition zurück transformiert.

$$\hat{m} = T^{-1} R^T \hat{m}_{Rot} \quad (5.41)$$

$$\hat{m}' = T'^{-1} R'^T \hat{m}'_{Rot} \quad (5.42)$$

Für diese neu berechneten korrespondierenden Punkte ist die epipolare Bedingung aus Gleichung 3.18 erfüllt und es ist gewährleistet, dass sich ihre jeweiligen Rückprojektionen in einem Punkt im Raum treffen.

Für die Rückprojektion der einzelnen Bildpunkte wurde ein lineares Triangulationsverfahren gewählt[2]. Dieses ist für mehrere Punkte rechentechnisch günstiger als das geometrische Verfahren, welches im synthetischen Beispiel in Kapitel 4 genutzt wurde.

Für die Rückprojektion werden pro korrespondierendem Punktpaar zunächst die Projektionsgleichungen  $\hat{m}_\sigma = PM_\delta$  und  $\hat{m}'_{\sigma'} = P'M_\delta$  aufgestellt. Diese werden so in eine Koeffizientenmatrix  $A$  eingetragen dass gilt  $A \cdot x = 0$ . Durch die Verwendung des Kreuzproduktes, wird die homogene Komponente eliminiert[2].

$$\hat{m}_\sigma \times (PM_\delta) = 0 \quad (5.43)$$

$$\hat{m}'_{\sigma'} \times (PM_\delta) = 0 \quad (5.44)$$

Was ausgeschrieben für  $\hat{m}$  und  $\hat{m}'$  zu den folgenden drei Gleichungen führt.  $\hat{m}_\sigma \times (PM_\delta) = 0$  ergibt ausgeschrieben

$$m_{\sigma x}(p^{3T} M_\delta) - (p^{1T} M_\delta) = 0 \quad (5.45)$$

$$m_{\sigma y}(p^{3T} M_\delta) - (p^{2T} M_\delta) = 0 \quad (5.46)$$

$$m_{\sigma x}(p^{2T} M_\delta) - m_{\sigma y}(p^{1T} M_\delta) = 0, \quad (5.47)$$

und Für  $\hat{m}'_{\sigma'} \times (PM_\delta) = 0$  gelten

$$m'_{\sigma' x}(p^{3T} M_\delta) - (p^{1T} M_\delta) = 0 \quad (5.48)$$

$$m'_{\sigma' y}(p^{3T} M_\delta) - (p^{2T} M_\delta) = 0 \quad (5.49)$$

$$m'_{\sigma' x}(p^{2T} M_\delta) - m'_{\sigma' y}(p^{1T} M_\delta) = 0. \quad (5.50)$$

$p^{iT}$  bezeichnet hier jeweils die Reihen der Projektionsmatrix  $P$  beziehungsweise  $P'$ . Zwei der drei Gleichungen sind linear unabhängig und werden in die Koeffizientenmatrix  $A$  geschrieben

$$A = \begin{bmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{bmatrix}. \quad (5.51)$$

Die zwei Wege eine solche Matrix zu lösen wurden in Kapitel 3 vorgestellt. Zum einen kann die inhomogene Methode angewandt werden in welcher die Bestimmung des Kerns das Ergebnis für  $M_{\delta 0}$  bietet oder es kann das homogene Verfahren angewandt werden, welches die Methode der Singulärwertzerlegung beinhaltet.

Die rekonstruierten Punkte  $M_{\delta 0}$  sind wie in Kapitel 4 auch nur bis auf einen Skalierungsfaktor genau bestimmt. Die Abbildungen 5.15 und 5.16 zeigen die aus den Bildern 5.4 und 5.3 rekonstruierten Punkte im Raum. Der rote Punkt steht für die Position von  $C$ , der grüne für die Position von  $C'$ . Die blauen Punkte sind die rekonstruierten Punkte, der beiden Bilder 5.3 und 5.4. Somit wurde gezeigt, dass der Algorithmus, welcher für die synthetische Rekonstruktion entwickelt wurde, mit gewissen Modifizierungen auch für ein reelles Stereobildpaar angewandt werden kann. Durch die Näherungen, sind gewissen Abweichungen von der Originalszene nicht zu vermeiden.

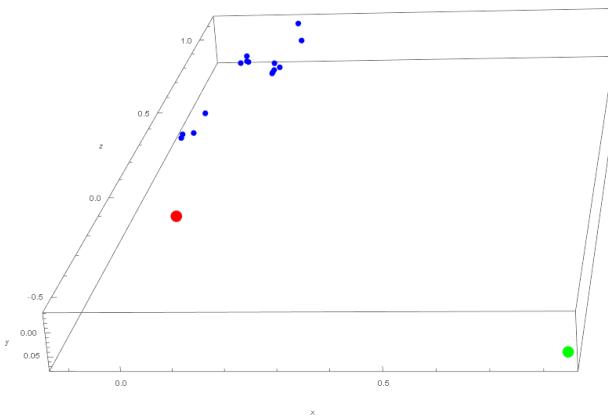


Abbildung 5.15: Rekonstruierte Szene, unskaliert in Pixeleinheiten

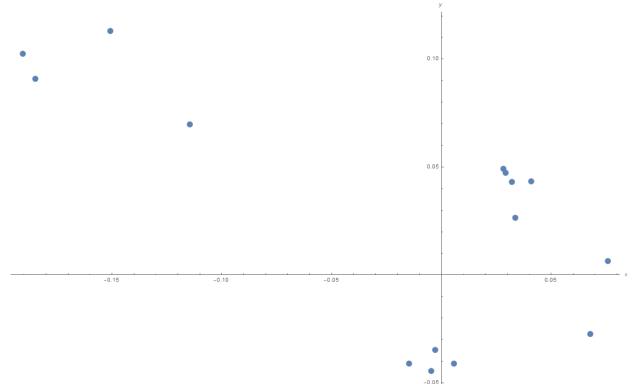


Abbildung 5.16: Rekonstruierte Szene, unskaliert, in Pixeleinheiten und in einem 2D-Plot angezeigt

## 5.5 Ergebnisse einer Stereoanalyse mit Kameras unterschiedlicher Auflösung

Für den Test, ob die Szeneriekonstruktion im Realbeispiel auch mit unterschiedlichen Kameraauflösungen funktioniert, wird die Kameramatrix  $K'$  von  $C'$  künstlich skaliert. Wie aus Kapitel 2, hat diese die Form

$$K'_{[1:1]} = \begin{bmatrix} k_x \zeta & s & V_{\sigma x} \\ 0 & k_y \zeta & V_{\sigma y} \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.52)$$

Um die Auflösung von  $K'$  zu verändern, werden die Matrixeinträge  $k_x\zeta$  und  $k_y\zeta$  jeweils noch um eine beliebige Skalierung erweitert. Für den Test wird  $K'$  drei mal unterschiedlich skaliert und zwar mit den Verhältnissen  $[5 : 2]$ ,  $[1 : 2]$  und  $[1.2 : 2.3]$ .

$$K'_{[5:2]} = \begin{bmatrix} k_x\zeta \cdot 5 & s & V_{\sigma x} \cdot 5 \\ 0 & k_y\zeta \cdot 2 & V_{\sigma y} \cdot 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K'_{[1:2]} = \begin{bmatrix} k_x\zeta \cdot 1 & s & V_{\sigma x} \cdot 1 \\ 0 & k_y\zeta \cdot 2 & V_{\sigma y} \cdot 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K'_{[1.2:2.3]} = \begin{bmatrix} k_x\zeta \cdot 1.2 & s & V_{\sigma x} \cdot 1.2 \\ 0 & k_y\zeta \cdot 2.3 & V_{\sigma y} \cdot 2.3 \\ 0 & 0 & 1 \end{bmatrix}$$

Von den detektierten korrespondierenden Bildpunkten des SURF-Algorithmus werden die Bildpunkte des zweiten Bildes von  $C'$  auch um die selben Verhältnisse skaliert.

$$m_{\sigma[5:2]} = m_{\sigma} \cdot \begin{pmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.53)$$

$$m_{\sigma[1:2]} = m_{\sigma} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.54)$$

$$m_{\sigma[1.2:2.3]} = m_{\sigma} \cdot \begin{pmatrix} 1.2 & 0 & 0 \\ 0 & 2.3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.55)$$

Der Szenenrekonstruktionsalgorithmus wird für jede Kameraauflösung getestet. Die Abbildungen 5.21, 5.21 und 5.23 zeigen jeweils die vier Lösungen für  $T'$ , welche sich bei der Bestimmung der extrinsischen Parameter bei unterschiedlichen Kameraauflösungen ergaben. Zu beobachten ist, dass sich die Lösungen bei unterschiedlichen Auflösungen nicht unterscheiden. Die geringen Abweichungen in den Nachkommastellen sind auf die Ungenauigkeiten der Bilddaten zurückzuführen. Durch die Skalierung der Bildkoordinaten in die neuen Sensorkoordinatensysteme werden auch deren Abweichungen mit skaliert.

$$\begin{aligned} T1 &= \begin{pmatrix} 0.991092 & 0.120891 & 0.0558752 & -0.812277 \\ -0.120366 & 0.992649 & -0.0126719 & 0.0389145 \\ -0.0569964 & 0.00583352 & 0.998357 & 0.581973 \end{pmatrix} \\ T2 &= \begin{pmatrix} 0.991092 & 0.120891 & 0.0558752 & 0.812277 \\ -0.120366 & 0.992649 & -0.0126719 & -0.0389145 \\ -0.0569964 & 0.00583352 & 0.998357 & -0.581973 \end{pmatrix} \\ T3 &= \begin{pmatrix} 0.378236 & -0.0296342 & -0.925235 & -0.812277 \\ 0.0547644 & -0.997021 & 0.0543212 & 0.0389145 \\ -0.924088 & -0.0712161 & -0.375487 & 0.581973 \end{pmatrix} \\ T4 &= \begin{pmatrix} 0.378236 & -0.0296342 & -0.925235 & 0.812277 \\ 0.0547644 & -0.997021 & 0.0543212 & -0.0389145 \\ -0.924088 & -0.0712161 & -0.375487 & -0.581973 \end{pmatrix} \end{aligned}$$

Abbildung 5.17: Zeigt die rekonstruierte Matrix  $T'$  bei unveränderter Auflösung. Die Auflösungen von  $C_\delta$  und  $C'_\delta$  sind die selben.

$$\begin{aligned} T1 &= \begin{pmatrix} 0.991142 & 0.121017 & 0.0546967 & 0.812113 \\ -0.120498 & 0.992632 & -0.0126975 & -0.0388039 \\ -0.0558303 & 0.00599422 & 0.998422 & -0.582208 \end{pmatrix} \\ T2 &= \begin{pmatrix} 0.991142 & 0.121017 & 0.0546967 & -0.812113 \\ -0.120498 & 0.992632 & -0.0126975 & 0.0388039 \\ -0.0558303 & 0.00599422 & 0.998422 & 0.582208 \end{pmatrix} \\ T3 &= \begin{pmatrix} 0.376619 & -0.0296193 & -0.925895 & 0.812113 \\ 0.0551443 & -0.996999 & 0.0543246 & -0.0388039 \\ -0.924725 & -0.0715175 & -0.373856 & -0.582208 \end{pmatrix} \\ T4 &= \begin{pmatrix} 0.376619 & -0.0296193 & -0.925895 & -0.812113 \\ 0.0551443 & -0.996999 & 0.0543246 & 0.0388039 \\ -0.924725 & -0.0715175 & -0.373856 & 0.582208 \end{pmatrix} \end{aligned}$$

Abbildung 5.18: Zeigt die rekonstruierte Matrix  $T'$  wenn  $K'$  mit einem Verhältnis von  $[5 : 2]$  skaliert wurde

$$\begin{aligned}
T1 &= \begin{pmatrix} 0.990947 & 0.120272 & 0.0596553 & -0.810768 \\ -0.119751 & 0.992728 & -0.0122401 & 0.0387536 \\ -0.0606937 & 0.0049855 & 0.998144 & 0.584083 \end{pmatrix} \\
T2 &= \begin{pmatrix} 0.990947 & 0.120272 & 0.0596553 & 0.810768 \\ -0.119751 & 0.992728 & -0.0122401 & -0.0387536 \\ -0.0606937 & 0.0049855 & 0.998144 & -0.584083 \end{pmatrix} \\
T3 &= \begin{pmatrix} 0.37685 & -0.0292569 & -0.925812 & -0.810768 \\ 0.0543725 & -0.997079 & 0.0536412 & 0.0387536 \\ -0.924677 & -0.0705534 & -0.374158 & 0.584083 \end{pmatrix} \\
T4 &= \begin{pmatrix} 0.37685 & -0.0292569 & -0.925812 & 0.810768 \\ 0.0543725 & -0.997079 & 0.0536412 & -0.0387536 \\ -0.924677 & -0.0705534 & -0.374158 & -0.584083 \end{pmatrix}
\end{aligned}$$

Abbildung 5.19: Zeigt die rekonstruierte Matrix  $T'$  wenn  $K'$  mit einem Verhältnis von  $[1 : 2]$  skaliert wurde

$$\begin{aligned}
T1 &= \begin{pmatrix} 0.990963 & 0.12034 & 0.0592445 & -0.81095 \\ -0.119818 & 0.99272 & -0.0122887 & 0.0387766 \\ -0.060292 & 0.0050791 & 0.998168 & 0.583829 \end{pmatrix} \\
T2 &= \begin{pmatrix} 0.990963 & 0.12034 & 0.0592445 & 0.81095 \\ -0.119818 & 0.99272 & -0.0122887 & -0.0387766 \\ -0.060292 & 0.0050791 & 0.998168 & -0.583829 \end{pmatrix} \\
T3 &= \begin{pmatrix} 0.377058 & -0.0293027 & -0.925726 & -0.81095 \\ 0.0544044 & -0.997073 & 0.0537206 & 0.0387766 \\ -0.92459 & -0.0706194 & -0.37436 & 0.583829 \end{pmatrix} \\
T4 &= \begin{pmatrix} 0.377058 & -0.0293027 & -0.925726 & 0.81095 \\ 0.0544044 & -0.997073 & 0.0537206 & -0.0387766 \\ -0.92459 & -0.0706194 & -0.37436 & -0.583829 \end{pmatrix}
\end{aligned}$$

Abbildung 5.20: Zeigt die rekonstruierte Matrix  $T'$  wenn  $K'$  mit einem Verhältnis von  $[1.2 : 2.3]$  skaliert wurde

Die Abbildungen 5.21, 5.22 und 5.23 zeigen einmal die Rekonstruierten Punkte in einem 3D-Plot und daneben den 2D-Plot. Auch hier kann beobachtet werden, dass es immer zu den gleichen rekonstruierten Szenen kommt. In Abbildung 5.22 ist die Rekonstruktion in einem links drehendem Koordinatensystem geplottet, weshalb die Rekonstitution spiegelverkehrt wirkt.

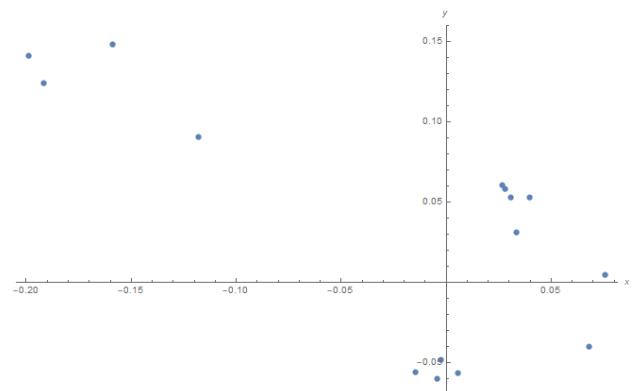
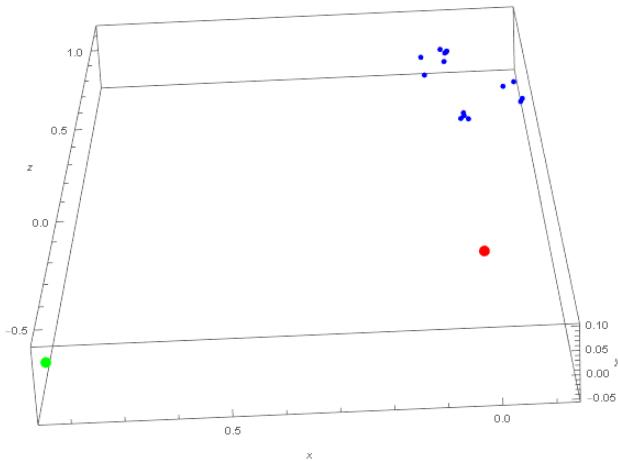


Abbildung 5.21: Rekonstruierte Szene, wenn  $K'$  mit einem Verhältnis von  $[5 : 2]$  skaliert wurde

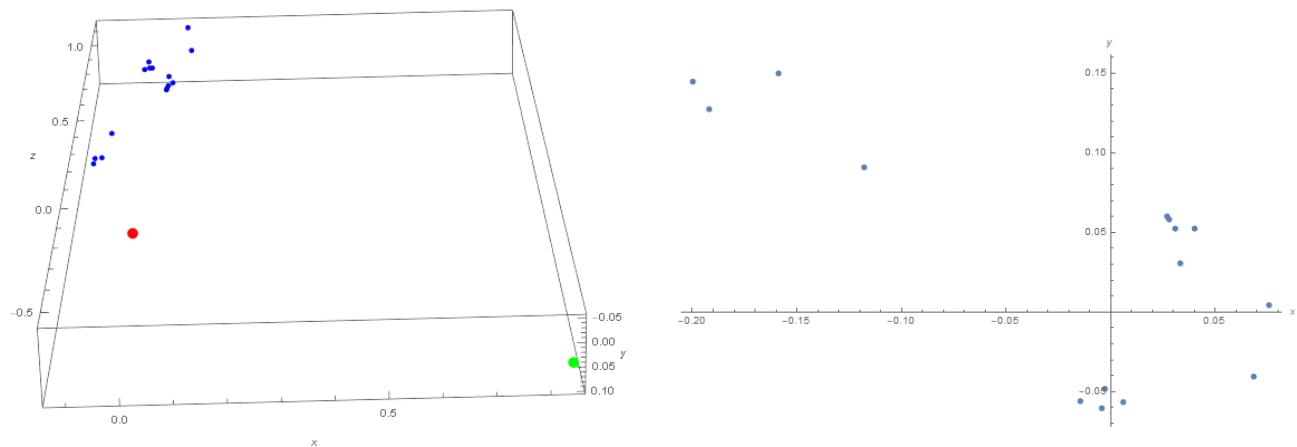


Abbildung 5.22: Rekonstruierte Szene, wenn  $K'$  mit einem Verhältnis von  $[1 : 2]$  skaliert wurde

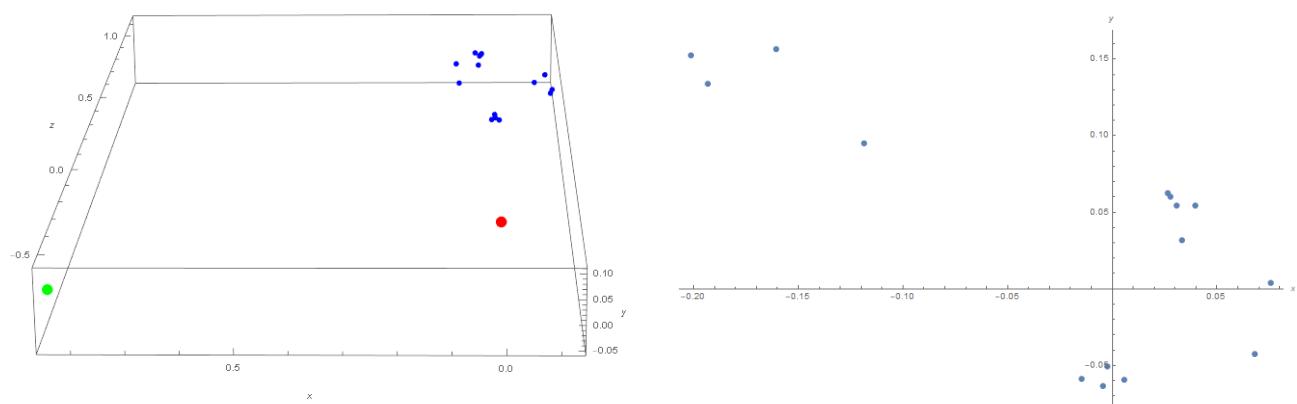


Abbildung 5.23: Rekonstruierte Szene, wenn  $K'$  mit einem Verhältnis von  $[1.2 : 2.3]$  skaliert wurde

# 6 Szenenrekonstruktion durch Rektifizierung

Ein verbreiteter Ansatz der Stereobildanalyse basiert auf zuvor rektifizierten Bildern. Rektifizierte Bilder zeichnen sich durch ins unendliche projizierte Epipole aus. Dies hat zu Folge, dass die jeweiligen Epipolarlinien der Bilder parallel zueinander verlaufen. Anschließend werden die Epipole noch so rotiert, dass die Epipolarlinien in einheitlichen Scanlinien über beide Bilder verlaufen. In Abbildung 6.1 ist das graphisch dargestellt[29, 30, 31, 32, 15].

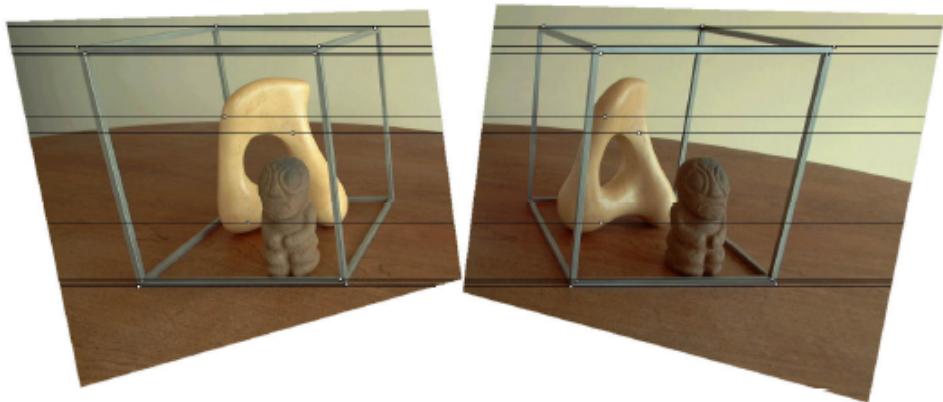


Abbildung 6.1: Beispiel eines rektifizierten Bildes. Die Epipole beider Bilder sind ins unendliche projiziert worden. Des Weiteren wurden die Epipole so rotiert, dass die Epipolarlinien zu einheitlichen Scanlinien über beide Bilder verlaufen. Quelle: [31]

Anhand der so entstandenen Scanlinien wird die Suche nach weiteren korrespondierenden Punkten auf eine eindimensionale Suche beschränkt. Der korrespondierende Punkt zu einem Punkt auf einem Bild, wird nur noch entlang der entsprechenden korrespondierenden Epipolarlinie gesucht. Die Rektifizierung ermöglicht eine effiziente Analyse von Punktekorrespondenzen ganzer Bilder mit geringstem Rechenaufwand[31, 29, 30]. Jedoch verlangen viele Rektifizierungsansätze als Voraussetzung, dass das verwendetet Stereobildpaar die selbe Auflösung besitzt.

Im Verlauf des Kapitels wird zunächst der Arbeitsprozess der Stereoanalyse auf Basis von Bildrektifizierungen erläutert. Anschließend wird ein Rektifizierungsalgorithmus nach *Charles Loop & Zhengyou Zhang*[31] vorgestellt, welcher nur durch Vorwissen von neun korrespondierenden Punkten und der daraus resultierenden Fundamentalmatrix eine Rektifizierung zweier Bilder ermöglicht. Zuerst werden zwei Bilder gleicher Auflösung rektifiziert, danach werden zwei Tests mit Bildern unterschiedlicher Auflösung durchgeführt. Die Resultate der rektifizierten Bilder mit unterschiedlichen Auflösungen werden im Anschluss analysiert.

## 6.1 Szenenrekonstruktion mit Rektifizierung

Bestimmte Formen der Rektifizierung benötigen keine vorherige Kalibrierung der Kameras und werden in den meisten gängigen Szenenrekonstruktionsalgorithmen eingesetzt. [29, 30, 33]. Der Arbeitsprozess für die Szenenrekonstruktion auf Basis von rektifizierten Bildern, welcher in Abbildung 6.2 schematisch dargestellt ist, unterscheidet sich etwas zu den bereits bekannten Arbeitsprozessen, welche in den Abbildungen 4.1 und 5.1 zusammengefasst sind.

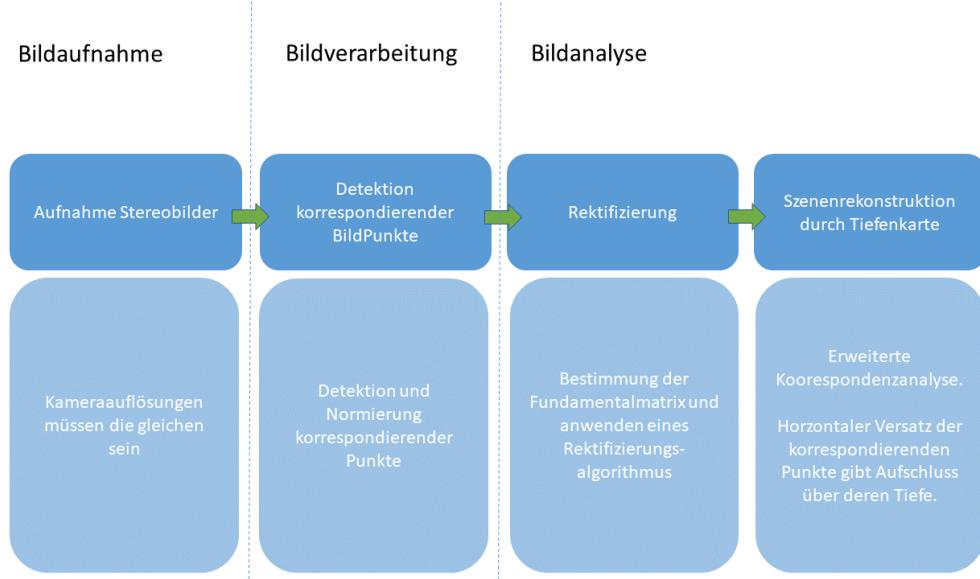


Abbildung 6.2: Ablaufdiagramm der Szenenrekonstruktion basierend auf einem Rektifizierungsansatz

Der entscheidende Unterschied zwischen den Ansätzen liegt im Abschnitt der Bildanalyse. Die Bilder werden hier vor der Szenenrekonstruktion rektifiziert. Die Rektifizierung benötigt keine extrinsischen oder intrinsischen Kameraparameter und erlaubt die Rekonstruktion einer Szene mit minimalen vorher bestimmten Informationen[31, 30, 29, 15].

Zunächst werden acht bis neun korrespondierende Punkte in einem Stereobildpaar detektiert. Aus den zuvor noch normierten korrespondierenden Punkten wird eine Fundamentalmatrix  $F$  bestimmt, wie in Kapitel 5 beschrieben. Mit den korrespondierenden Punkten und der Fundamentalmatrix  $F$  wird ein Rektifizierungsalgorithmus implementiert, welcher zwei Stereobilder transformiert, sodass ihre Epipolarlinien zu horizontal ausgerichteten Scanlinien werden, wie in Abbildung 6.1 zu sehen ist. Mit Hilfe dieser entstandenen Scanlinien, wird die Suche nach weiteren korrespondierenden Punkten von einer zweidimensionalen auf eine eindimensionale Suche vereinfacht.[31, 29, 30].

Nach der Rektifizierung ist zwischen zwei korrespondierenden Punkten ein horizontaler Versatz zu verzeichnen, welcher durch die unterschiedlichen Perspektiven der Bilder entsteht. Dieser Versatz entsteht durch den Abstand der beiden Kameras zueinander. Der Versatz zwischen den beiden korrespondierenden Punkten wird als Disparität bezeichnet und ist ein Maß für die Tiefe der Punkte in der Szene[30, 29]. Weit von den Kameras entfernte Punkte werden durch die unterschiedliche Perspektive der beiden weniger stark beeinflusst als Punkte, welche sich nah an den Kameras befinden. Die Disparität lässt sich als Disparitätskarte oder Tiefenkarke für stereoskopische Bilder veranschaulichen[30]. Der Disparität wird ein Grauwert zugeordnet. In Abbildung 6.3 werden Punkten mit einer kleinen Disparität ein dunkler Wert und Punkten mit einer großen Disparität ein heller Wert zugeordnet. Die Rekonstruktion durch eine Disparitätskarte wird auch als *dense rectification* bezeichnet und bietet eine schnelle und auch effiziente Möglichkeit ein Maß für die Tiefe einer Szene zu generieren.[30, 29, 31]. Eine weitere Möglichkeit für eine Szenenrekonstruktion auf Basis einer Rektifizierung ist, wie bereits bekannt anhand der korrespondierenden Punkte eine Triangulation durchzuführen[30].



Abbildung 6.3: In den oberen beiden Bildern sind in blau zwei korrespondierende Epipolarlinien zu sehen, die zusammen eine Scanlinie bilden. Im Bild darunter ist eine aus den Dispritäten zweier Bilder zueinander entstandene Disparitäts- beziehungsweise Tiefenkarthe zu sehen. Quelle: [30]

Das standard Rektifizierungsverfahren, welches im folgenden Kapitel eingeführt wird, ist für Bilder gleicher Auflösung definiert. Für den Fall ungleicher Auflösungen wird dieses Verfahren erweitert und anhand von Beispielen die Anwendbarkeit des Rektifizierungsalgorithmus überprüft.

## 6.2 Rektifizierung mit Homographien

Im Folgenden wird ein Rektifizierungsalgorithmus nach *Zhang*[31] vorgestellt. Diese Art der Rektifizierung zeichnet sich durch minimale Voraussetzungen an die Ursprungsbilder aus. Alle notwendigen Informationen zur Rektifizierung werden aus der Fundamentalmatrix gewonnen. Zusätzlich zur Fundamentalmatrix muss noch die Lage der jeweiligen Epipole bekannt sein[15]. Anhand des entstandenen Algorithmus wird dann getestet, ob Bilder unterschiedlicher Auflösung richtig rektifiziert werden können und was die Ergebnisse für den weiteren Verlauf der Szenenrekonstruktion bedeuten könnten.

Für den Rektifizierungsansatz wird pro Bild eine Homographiematrix  $H$  und  $H'$  aufgestellt.

$$\bar{m}_\sigma = H m_\sigma \quad (6.1)$$

$$\bar{m}'_{\sigma'} = H' m'_{\sigma'} \quad (6.2)$$

Die Fundamentalmatrix, welche aus den rektifizierten korrespondierenden Punkte resultiert, wird mit  $\bar{F}$  bezeichnet[31, 15]:

$$\bar{m}_{\sigma'}^T \bar{F} \bar{m}_{\sigma} = 0 \quad (6.3)$$

$$\rightsquigarrow m_{\sigma'}^T H'^T \bar{F} H m_{\sigma} = 0 \quad (6.4)$$

$$\rightsquigarrow F = H'^T [i] \times H \quad (6.5)$$

Die Homographien  $H$  und  $H'$  werden in die projektiven Komponenten  $H_p$  und  $H'_p$  und die affinen Komponente  $H_a$  und  $H'_a$  unterteilt. Die affine Komponente wird wiederum in zwei weitere Komponenten unterteilt.  $H_r$  steht für eine Ähnlichkeitstransformation und  $H_s$  bezeichnet eine Scherungstransformation [31, 15].

$$H = H_a H_p \rightsquigarrow H = H_s H_r H_p \quad (6.6)$$

$$H' = H'_a H'_p \rightsquigarrow H' = H'_s H'_r H'_p \quad (6.7)$$

$$(6.8)$$

$H_p$  und  $H'_p$  beinhalten eine projektive Transformationen, welche dafür sorgt, dass der Epipol  $e$  und der Epipol  $e'$  ins Unendliche projiziert wird und die Epipolarlinien parallel zueinander ausgerichtet sind[31, 15].  $H_r$  und  $H'_r$  sind Rotationsmatrizen, welche die Epipolarlinien parallel zur horizontalen Achse ausrichtet.  $H_s$  und  $H'_s$  sind Scherungsmatrizen, welche durch Minimierung versuchen die durch die vorherigen Transformationen  $H_p$  und  $H_r$  entstandenen projektiven Verzerrungen in den Bildern bestmöglich auszugleichen[31, 15].

Die Reihen der Homographiematrix  $H$  beschreibt drei Linien  $u$ ,  $v$  und  $w$  mit  $u = (u_a, u_b, u_c)^T$ ,  $(v_a, v_b, v_c)^T$  und  $(w_a, w_b, w_c)^T$ , welche jeweils durch den Epipol verlaufen. Selbiges gilt auch für  $H'$  mit den Linien  $u'$ ,  $v'$  und  $w'$ . Die Linien  $v$  und  $v'$  sowie  $w$  und  $w'$  sind korrespondierende Epipolarlinien. Dadurch entsteht eine geometrische Beziehung zwischen den beiden Bildern[31].

$$H = \begin{bmatrix} u^T \\ v^T \\ w^T \end{bmatrix} = \begin{bmatrix} u_a & u_b & u_c \\ v_a & v_b & v_c \\ w_a & w_b & w_c \end{bmatrix} \quad (6.9)$$

$$H' = \begin{bmatrix} u'^T \\ v'^T \\ w'^T \end{bmatrix} = \begin{bmatrix} u'_a & u'_b & u'_c \\ v'_a & v'_b & v'_c \\ w'_a & w'_b & w'_c \end{bmatrix} \quad (6.10)$$

Bevor die Matrizen  $H$  und  $H'$  in ihre projektiven und affinen Komponenten zerlegt werden, wird die letzte Komponenten  $w_c$  und  $w'_c$  durch Division eliminiert um somit skaleninvariante Matrizen  $H$  und  $H'$  zu erhalten[31, 15].

$$H = \begin{bmatrix} u^T \\ v^T \\ w^T \end{bmatrix} = \begin{bmatrix} u_a & u_b & u_c \\ v_a & v_b & v_c \\ w_a & w_b & 1 \end{bmatrix} \quad (6.11)$$

$$H' = \begin{bmatrix} u'^T \\ v'^T \\ w'^T \end{bmatrix} = \begin{bmatrix} u'_a & u'_b & u'_c \\ v'_a & v'_b & v'_c \\ w'_a & w'_b & 1 \end{bmatrix} \quad (6.12)$$

Die Matrizen  $H_p$  und  $H'_p$  beschreiben den projektiven Teil von  $H$  und  $H'$  und wirken sich auf den projektiven Teil eines Punktes aus. Sie werden dazu verwendet die Epipole ins unendliche zu projizieren[31, 15].

$$H_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ w_a & w_b & 1 \end{bmatrix} \quad (6.13)$$

Für die affinen Komponenten  $H_a$  und  $H'_a$  gilt

$$H_a = H \cdot H_p^{-1} = \begin{bmatrix} u_a - v_c w_b & v_c w_a - v_a & 0 \\ v_a - v_c w_a & v_b - v_c w_b & v_c \\ 0 & 0 & 1 \end{bmatrix} \quad (6.14)$$

Des Weiteren gilt für  $H_a$  und  $H'_a$ , dass sie jeweils nochmal in eine Rotationsmatrix  $H_r$  und  $H'_r$  und eine Scherungsmatrix  $H_s$  und  $H'_s$  zerlegt werden[31, 15].

$$H_a = H_s \cdot H_r \quad (6.15)$$

$$H_r = \begin{bmatrix} v_b - v_c w_b & v_a - v_c w_a & 0 \\ v_a - v_c w_a & v_b - v_c w_b & v_c \\ 0 & 0 & 1 \end{bmatrix} \quad (6.16)$$

$$H_s = \begin{bmatrix} u_a & u_b & u_c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.17)$$

$H_r$  und auch  $H'_r$  definieren eine Rotation und auch eine Verschiebung, welche die bereits parallelen Epipolarlinien horizontal ausrichtet. Durch die Verschiebung  $v_c$  wird der vertikale Versatz zwischen den Epipolarlinien beider Bilder ausgeglichen. Es entstehen die einheitlichen Scanlinien. Die Matrizen  $H_s$  und  $H'_s$  definieren eine Scherung. Sie gehören nicht zum eigentlichen Rektifizierungsprozess aber dienen dazu die horizontale Verzerrung der Bilder, welche durch die Rektifizierungstransformationen entstanden sind, wieder auszugleichen.

### 6.2.1 Projektive Transformation

Im Folgenden wird die Herleitung der Matrizen  $H_p$  und  $H'_p$  beschrieben. Die projektiven Matrizen  $H_p$  und  $H'_p$  werden aus den Linien  $w$  und  $w'$  gebildet.  $w$  und  $w'$  werden nicht willkürlich gewählt. Definiert werden sie durch eine Richtung  $z = [\lambda \ \mu \ 0]^T$ .  $z$  soll dabei so gewählt werden, dass die durch die Rektifizierung entstehenden Bildverzerrungen in beiden Bildern minimal bleibt. Die Linien  $w$  und  $w'$  werden wie folgt definiert.

$$w = [e]_x \cdot z \quad (6.18)$$

$$w' = F \cdot z \quad (6.19)$$

Die Minimierung beinhaltet den Versuch ein  $z$  zu finden, welches  $w = (w_a, w_b, w_c)^T$  und  $w' = (w'_a, w'_b, w'_c)^T$  so definiert, dass die Einträge  $w_a$  und  $w_b$  und auch  $w'_a$  und  $w'_b$  in  $H_p$  und  $H'_p$  nahezu null sind. Anders ausgedrückt es wird versucht die projektiven Matrizen  $H_p$  und  $H'_p$  so affin wie möglich zu machen[31].

$$H_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ w_a & w_b & 1 \end{bmatrix} \quad (6.20)$$

Sollte es der Fall sein, dass der Epipole  $e$  bereits im unendlichen ist, so sind  $w_a = 0$  und  $w_b = 0$ . In diesem Fall ist eine projektive Transformation für diesen Epipol nicht mehr nötig.

Für die Minimierung wird die Methode des Rayleigh-Verfahren angewandt. Die Methode der kleinsten Quadrate ist ein mathematisches Standardverfahren für eine Ausgleichsrechnung. Mit ihrer Hilfe soll aus der Menge der Bildpunkte beider Bilder ein Wert für  $z$  ermittelt werden[34]. Für  $z$  gilt mit  $z = [\lambda \ \mu \ 0]^T$  bereits die Bedingung, dass es sich um einen Punkt im unendlichen handeln soll.  $\lambda$  und  $\mu$ , sollen dabei einen Wert annehmen, der sich nah an den Gewichtungen der Punkteansammlungen beider Bilder befindet[31].

Um  $z$  zu ermitteln, werden zunächst die Gewichtungen der Punkte beider Bilder benötigt.  $p_i$  beinhaltet alle Punkte von Bild eins und  $p_j$  beinhaltet alle Punkte von Bild zwei. Ein Punkt  $p_{i1} = [p_{i1,u} \ p_{i1,v} \ 1]^T$  aus Bild eins soll zu einem Punkt  $p_{i1} = \left[\frac{p_{i1,u}}{w_i} \ \frac{p_{i1,v}}{w_i} \ 1\right]^T$  mit  $w_i$  gleich der Gewichtung

$$w_i = w^T p_i \quad (6.21)$$

transformiert werden. Dasselbe soll auch für die Punkte  $p_j$  im zweiten Bild geschehen. Sind die Gewichtungen beider Bilder gleich, so ergibt sich keine projektive Verzerrung und die Epipole sind bereits im unendlichen. Befindet sich einer oder beide Epipole nicht im unendlichen, so können die Gewichtungen der Punkte beider Bilder nicht gleich sein[31].

Das Ziel ist es einen Wert für  $z$  zu finden, der die Abweichung der Gewichtungen beider Bilder zueinander so gering wie möglich zu hält. Die Rektifizierung wurde anhand des synthetischen Beispiels in Kapitel 4 implementiert. Dem entsprechend bilden die Abbildungen der Eckpunkte des Quaders auf den Sensoren der virtuellen Kameras, die Punkte in  $p_i$  und  $p_j$  anhand welcher die Rektifizierung durchgeführt werden soll.

Der Wert  $p_c$  mit  $p_c = \frac{1}{n} \sum_{i=1}^n p_i$  ergibt sich aus dem Durchschnittswert aller Punkte eines Bildes und gibt den Bildmittelpunkt an.  $w_c$  ist die Gewichtung am Bildmittelpunkt und wird berechnet mit

$$w_c = w^T p_c \quad (6.22)$$

Die Abweichung der Gewichtungen wird bezüglich der Gewichtung am Bildzentrum  $w_c$  gemessen.

$$\sum_{i=1}^n \left[ \frac{w_i - w_c}{w_c} \right]^2 \quad (6.23)$$

$$\rightsquigarrow \sum_{i=1}^n \left[ \frac{w^T(p_i - p_c)}{w^T p_c} \right]^2 \rightsquigarrow \sum_{i=1}^n \left[ \frac{w^T(p_i - p_c)(p_i - p_c)^T w}{w^T p_c p_c^T w} \right] \quad (6.24)$$

Vereinfacht lässt sich das auch in einer Matrixgleichung in Form von

$$\frac{w^T P P^T w}{w^T p_c p_c^T w} \quad (6.25)$$

angeben, in welcher für  $P$  gilt

$$P = \begin{bmatrix} p_{1,u} - p_{c,u} & p_{2,u} - p_{c,u} & \dots & p_{i,u} - p_{c,u} \\ p_{1,v} - p_{c,v} & p_{2,v} - p_{c,v} & \dots & p_{i,v} - p_{c,v} \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad (6.26)$$

Für die Punkte  $p_j$  in Bild zwei ergibt sich dann entsprechend die Matrixgleichung

$$\frac{w'^T P' P'^T w'}{w'^T p'_c p'^T_c w'} \quad (6.27)$$

$w$  und  $w^T$  werden nun noch mit ihren Definitionen aus den Gleichungen 6.18 und 6.19 ersetzt. Die Gleichungen 6.25 und 6.27 werden somit als eine Funktion von  $z$  ausgedrückt[31].

$$\frac{z^T [e]_x^T P P^T [e]_x z}{z^T [e]_x^T p_c p_c^T [e]_x z} + \frac{z^T F^T P' P'^T F z}{z^T F^T p'_c p'^T_c F z} \quad (6.28)$$

Gleichung 6.28 wird noch vereinfach mit

$$A = [e]_x^T P P^T [e]_x \quad (6.29)$$

$$B = [e]_x^T p_c p_c^T [e]_x \quad (6.30)$$

$$A' = F^T P' P'^T F \quad (6.31)$$

$$B' = F^T p'_c p'^T_c F \quad (6.32)$$

$$\rightsquigarrow \frac{z^T A z}{z^T B z} + \frac{z^T A' z}{z^T B' F z} \quad (6.33)$$

$$f(z) = \frac{z^T A z}{z^T B z} + \frac{z^T A' z}{z^T B' F z} \quad (6.34)$$

Da bereits fest gesetzt ist das die dritte Komponente von  $z$  Null ist, wird  $z$  im Folgenden als zweidimensionaler Vektor mit  $z = \begin{pmatrix} \lambda \\ \mu \end{pmatrix}$  dargestellt.  $A, B, A'$  und  $B'$  sind 3x3-Matrizen, von denen dann nur noch der erste  $2 \times 2$ -Block wichtig ist.

Für eine nicht lineare Optimierung wird das gesamte Polynom 6.34 aufgeteilt. Als erstes, wird  $\frac{z^T A z}{z^T B z}$  minimiert und danach  $\frac{z^T A' z}{z^T B' z}$ . So entstehen zwei Lösungen  $\hat{z}_1$  und  $\hat{z}_2$ , welche über eine Mittelung eine erste Schätzung für  $z$  geben[31].

$$z = \frac{\frac{\hat{z}_1}{\|\hat{z}_1\|} + \frac{\hat{z}_2}{\|\hat{z}_2\|}}{2} \quad (6.35)$$

Da es sich um eine nicht lineare Optimierung handelt ist die Minimierung von  $\frac{z^T A z}{z^T B z}$  gleichzusetzen mit der Maximierung von  $\frac{z^T B z}{z^T A z}$ . Beide werden als eine Funktion von  $f(z)$  definiert. Matrix  $A$  wird mit der Choleskyzerlegung[28] in zwei höhere Dreiecksmatrizen zerlegt  $A = D^T D$ . Dies geht nur da  $A$  nachweislich eine symmetrische und positiv-definite Matrix ist[35, 28]. Des Weiteren wird definiert, dass  $y = Dz$  ist und  $f(z)$  wird zu  $\hat{f}(y)$ [31].

$$A = D^T D \quad (6.36)$$

$$y = Dz \rightsquigarrow z = D^{-1}y \quad (6.37)$$

$$f(z) = \frac{z^T B z}{z^T A z} \quad (6.38)$$

$$\rightsquigarrow f(z) = \frac{z^T B z}{z^T D^T D z} \quad (6.39)$$

$$\hat{f}(y) = \frac{y^T D^{-T} B D^{-1} y}{y^T y} \quad (6.40)$$

Da  $y$  bis auf einen Skalierungsfaktor bestimmt ist, kann angenommen werden, dass  $\|y\|=1$  gilt.  $\hat{f}(y)$  ist maximiert, wenn  $y$  gleich dem Eigenvektor von  $D^{-T}BD^{-1}$  ist, welcher mit dem größten Eigenwert von  $D^{-T}BD^{-1}$  assoziiert wird[31]. Für  $\hat{z}_1$  ergibt sich  $\hat{z}_1 = D^{-1}y$ . Exakt das selbe Verfahren wird für die Bestimmung von  $z_2$  mit  $\frac{z^T B' z}{z^T A' z}$  angewandt[31].

Die erste Schätzung von  $z$  bietet bereits eine akzeptable Lösung. Um das Ergebnis für  $z$  zu optimieren können die Werte  $z_1$  und  $z_2$  in Gleichung 6.34 eingesetzt und ein gemeinsames Minimum gesucht werden. das Minimum ergibt den neuen Wert für  $z$ .  $z$  wird in die Gleichungen 6.18 und 6.19 eingesetzt und  $w$  und  $w'$  werden berechnet. Die Einträge der Matrizen  $H_p$  und  $H'_p$  sind mit  $w = (w_a, w_b, w_c)^T$  und  $w' = (w'_a, w'_b, w'_c)^T$  bestimmt[31].

Im synthetischen Beispiel werden die Matrizen  $H_p$  und  $H'_p$  bestimmt und auf die Abbildungen des Quaders angewandt. Die Abbildung 6.4 zeigt die unrektifizierten Abbildungen der Quader in den Kameräen. In grün ist die Abbildung des Quaders in  $C$  zu sehen und in rot die Abbildung des Quaders in  $C'$ . Die Abbildung 6.5 zeigt in blau die Epipolarlinien beider Abbildungen.

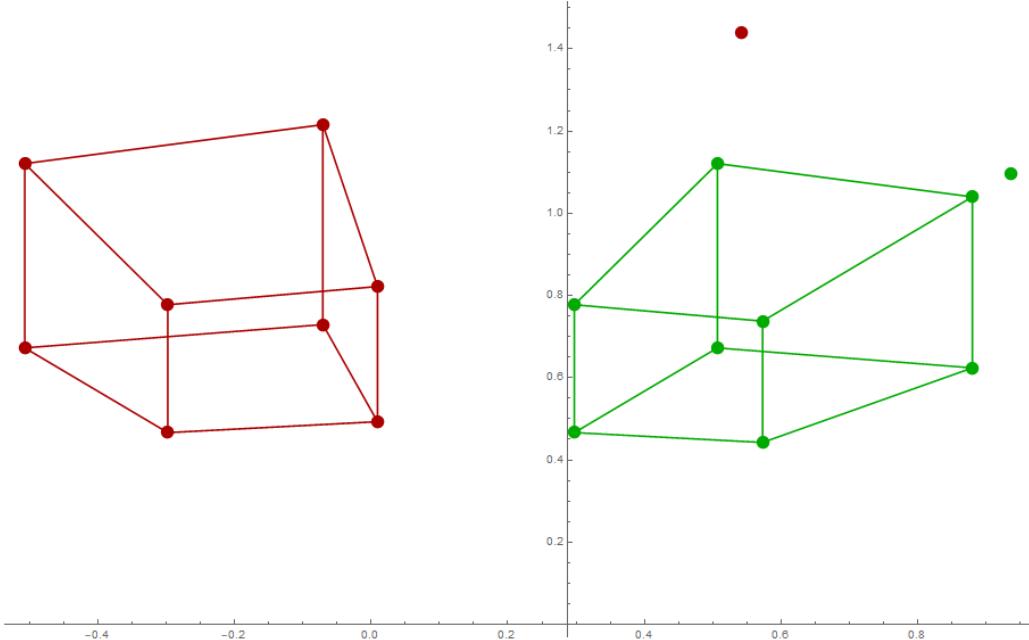


Abbildung 6.4: Aufnahmen zweier Kameräen mit den selben Auflösungen, Kamera eins(Grün) und Kamera(rot) zwei gelten jeweils  $\zeta = 1$

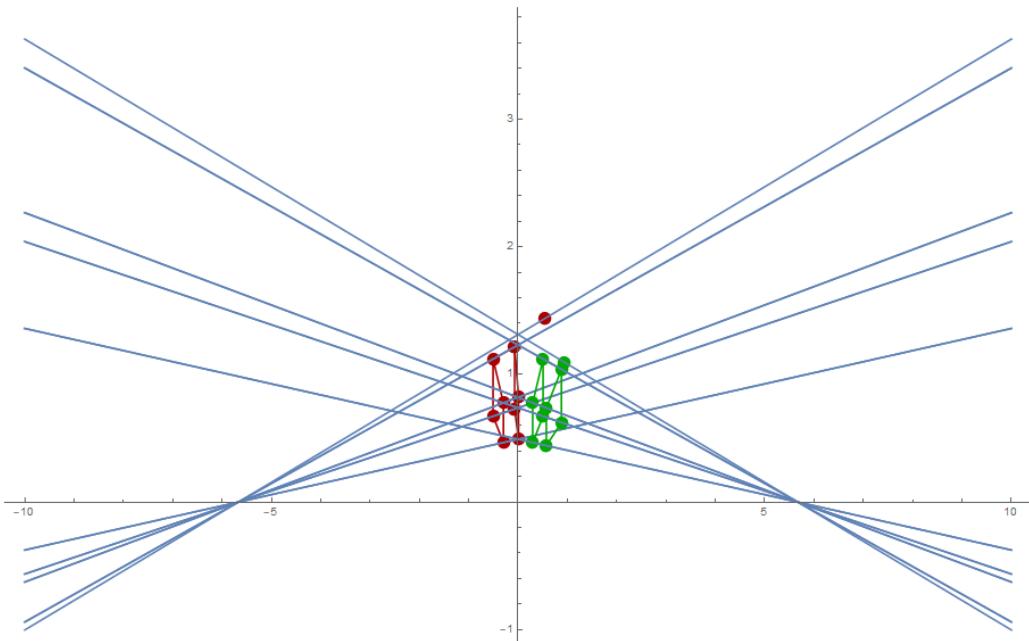


Abbildung 6.5: Epipole für Kamera eins und Kamera zwei vor der Rektifizierung

Nach der Transformation der Abbildungspunkte mit den Matrizen  $H_p$  und  $H'_p$  befinden sich die Epipole im Unendlichen. Die jeweiligen Epipolarlinien der Bilder verlaufen jetzt parallel zueinander, jedoch noch nicht zwingend parallel zur horizontalen Achse. In den Abbildungen 6.6 und 6.7 ist das Ergebnis der mit  $H_p$  und  $H'_p$  transformierten Bildpunkte zu sehen.

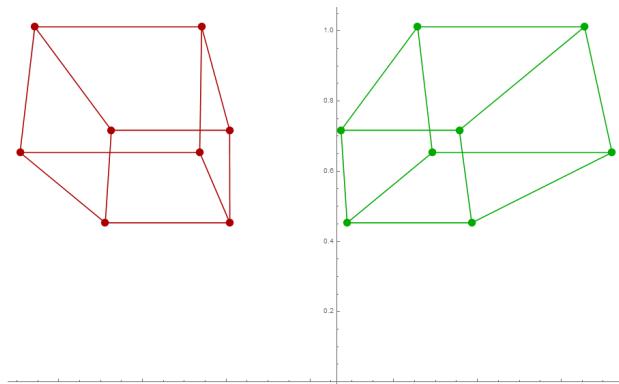


Abbildung 6.6: Abbildungen der Quader nach der Transformation mit  $H_p$  und  $H'_p$

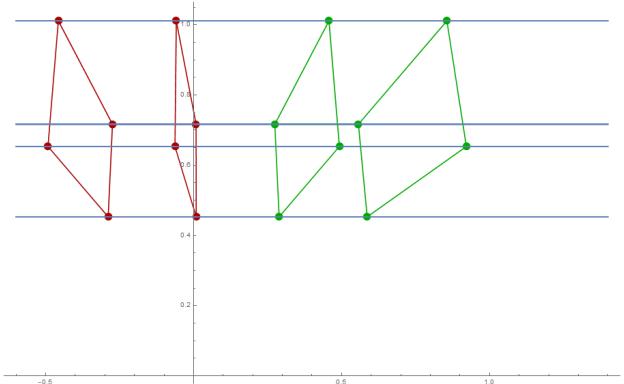


Abbildung 6.7: Abbildungen der Quader nach der Transformation mit  $H_p$  und  $H'_p$  mit eingezeichneten Epipolarlinien

## 6.2.2 Ähnlichkeitstransformation

Die Epipole der jeweiligen Bilder befinden sich nach der projektiven Transformation im Unendlichen. Die daraus resultierenden Epipolarlinien sind, wie in Abbildung 6.7 zu sehen, parallel zueinander angeordnet. Im Folgenden sollen die Epipole rotiert werden, sodass ihre Richtungen  $i = [1 \ 0 \ 0]$  betragen.  $H_r$  und  $H'_r$ , welche aus der Zerlegung von  $H_a$  und  $H'_a$  resultieren, sollen die Rotation ausführen.

$w$  und  $w'$  sind bereits bekannt. Mit Hilfe des bekannten  $F$ , können  $v_a$  und  $v_b$  ersetzt werden. Zur Erinnerung eine Voraussetzung für den hier aufgezeigten Rektifizierungsansatz ist, dass sowohl die Bildpunkte als auch die Fundamentalmatrix bekannt sein müssen. Die Epipole wurden ins unendliche projiziert und sollen nach der Rotation die Form  $e = [1 \ 0 \ 0]^T$  haben. Das heißt das die Fundamentalmatrix  $\bar{F}$  die folgende Form haben wird.

$$\bar{F} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (6.41)$$

Des Weiteren ist bekannt, dass  $F = H'^T \bar{F} H$

$$\bar{m}'^T \bar{F} \bar{m} = 0 \rightsquigarrow m'^T H'^T \bar{F} H m = 0 \quad (6.42)$$

$$\rightsquigarrow F = H'^T [i]_{\times} H \quad (6.43)$$

$$F = \begin{bmatrix} u'_a & v'_a & w'_a \\ u'_b & v'_b & w'_b \\ u'_c & v'_c & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_a & u_b & u_c \\ v_a & v_b & v_c \\ w_a & w_b & 1 \end{bmatrix} \quad (6.44)$$

$$F = \begin{bmatrix} v_a w'_a - v'_a w_a & v_b w'_a - v'_a w_b & v_c w'_a - v'_a \\ v_a w'_b - v'_b w_a & v_b w'_b - v'_b w_b & v_c w'_b - v'_b \\ v_a - v'_c w_a & v_b - v'_c w_b & v_c - v'_c \end{bmatrix} \quad (6.45)$$

Bisher sind  $w$ ,  $w'$  und die Fundamentalmatrix  $F$  aus den nicht-rektifizierten Punkten bekannt. Um  $v_a$ ,  $v_b$  und  $v_c$  zu erhalten, werden jeweils die Einträge der letzten Zeile in  $F$  umgeformt. Für  $v'_a$ ,  $v'_b$  und  $v'_c$  dient die letzte Spalte von  $F$ .

$$v_a = F_{31} + v'_c w_a \quad (6.46)$$

$$v_b = F_{32} + v'_c w_b \quad (6.47)$$

$$v_c = F_{33} + v'_c \quad (6.48)$$

$$v'_a = v_c w'_a - F_{13} \quad (6.49)$$

$$v'_b = v_c w'_b - F_{23} \quad (6.50)$$

$$v'_c = v_c - F_{33} \quad (6.51)$$

$F_{31}$ ,  $F_{32}$ ,  $F_{33}$ ,  $F_{13}$  und  $F_{23}$  stehen für die Einträge der Matrix  $F$ . Werden die Gleichungen 6.46 bis 6.51 in  $H_r$  und  $H'_r$  eingesetzt, so ergeben sich für  $H_r$  und  $H'_r$

$$H_r = \begin{bmatrix} v_b - v_c w_b & v_a - v_c w_a & 0 \\ v_a - v_c w_a & v_b - v_c w_b & v_c \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} F_{32} - w_b F_{33} & w_a F_{33} - F_{31} & 0 \\ F_{31} - w_a F_{33} & F_{32} - w_b F_{33} & F_{33} + v'_c \\ 0 & 0 & 1 \end{bmatrix} \quad (6.52)$$

$$H'_r = \begin{bmatrix} v'_b - v'_c w'_b & v'_a - v'_c w'_a & 0 \\ v'_a - v'_c w'_a & v'_b - v'_c w'_b & v'_c \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} w'_b F_{33} - F_{23} & F_{13} - w'_a F_{33} & 0 \\ w'_a F_{33} - F_{13} & w'_b F_{33} - F_{23} & v'_c \\ 0 & 0 & 1 \end{bmatrix} \quad (6.53)$$

Die gemeinsame unbekannte Variable  $v'_c$  beschreibt die geometrische Verbindung beider Bilder in ihrer Verschiebung entlang der vertikalen Richtung[31].  $F_{33}$  ist somit der horizontale Versatz, welcher benötigt wird, um die horizontalen Epipolarlinien beider Bilder zueinander auszurichten, sodass die gewünschten Scanlinien über beide Bilder entstehen.  $v'$  wird dabei so gewählt, dass die kleinste Koordinate der rektifizierten Bilder in vertikaler Achsenrichtung gleich null ist.[31].

Die Abbilder des Quaders sehen nach der Transformation mit  $H_r H_p$  und  $H'_r H'_p$  aus wie in den Abbildungen 6.8 und 6.9 dargestellt. Sollten die Epipolarlinien noch nicht parallel zur horizontalen Achse gewesen sein, so sind sie es nach der Transformation mit  $H_r$  und  $H'_r$ . Des Weiteren wurden beide Bilder durch  $v'_c$  so verschoben, dass der in vertikaler Richtung kleinste Bildpunkt beider Bilder jeweils auf der Horizontalen Koordinatenachse liegt.

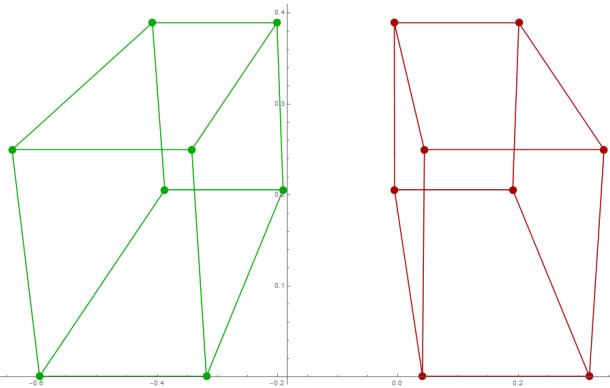


Abbildung 6.8: Abbildungen der Quader nach der Tranfromation mit  $H_rH_p$  und  $H'_rH'_p$

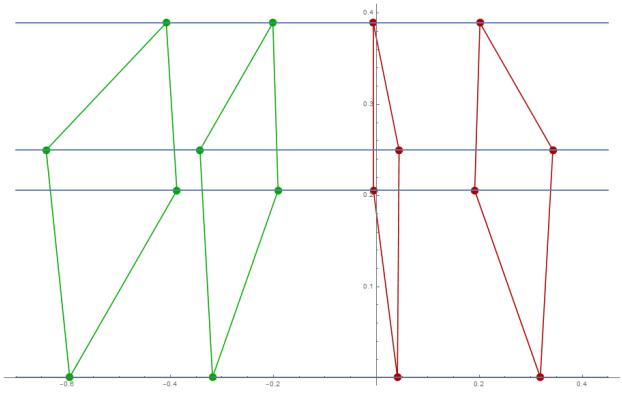


Abbildung 6.9: Abbildungen der Quader nach der Tranformation mit  $H_rH_p$  und  $H'_rH'_p$  mit eingezeichneten Epipolarlinien

### 6.2.3 Scherungstransformation

Im letzten Schritt soll die durch die Transformation mit  $H_p$  und  $H'_p$  entstandene horizontale Verzerrung in beiden Bilder reduziert werden. Unter der horizontalen Verzerrung versteht man die Verschiebung der gegenüberliegenden Bildkanten zueinander. Die Verbindungsline der gegenüberliegenden Bildkantenmittnen stehen nicht mehr orthogonal aufeinander, was eine Verzerrung des Bildes bewirkt. Abbildung 6.10 stellt diese Verzerrung schematisch dar.

Aus diesem Grund werden die Scherungsmatrizen  $H_s$  und  $H'_s$ , aus der Zerlegung der affinen Matrix  $H_a$  und  $H'_a$  benötigt. Die Einträge der ersten Zeile in  $H_s$  und  $H'_s$  haben nur noch Auswirkungen auf die horizontalen Koordinaten der Bildpunkte.

$$H_s = \begin{bmatrix} u_a & u_b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.54)$$

$$H'_s = \begin{bmatrix} u'_a & u'_b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.55)$$

Um die richtigen Werte für  $u_a, u'_a, u_b$  und  $u'_b$  zu erhalten, werden zunächst Punkte an den jeweiligen gegenüberliegenden Kanten der Bilder definiert. Da die Bilder des Quaders nicht aus mehreren Pixeln bestehen, wie ein reales Bild, sondern nur über dessen Eckpunkte bestimmt sind, wird eine Bildbreite  $w$  und  $w'$  und eine Bildhöhe  $h$  und  $h'$  um die Abbildungen des Quaders definiert.

Da es nicht möglich ist die horizontale Verzerrung gänzlich zu reduzieren, wird stattdessen die Orthogonalität der Verbindungslien  $\overline{bd}$  und  $\overline{ca}$  wieder hergestellt. In Abbildung 6.10 ist das Ziel grafisch dargestellt.

Für den nächsten Schritt werden im synthetischen Beispiel für die Abbildungen des Quaders jeweils Bildweite  $w$  und Bildhöhe  $h$  definiert. Da zunächst von gleichen Kameraauflösungen ausgegangen wird, sind die Bildbreiten und Höhen beider Bilder gleich. Danach werden die vier Mittelpunkte  $a, b, c$  und  $d$  der Bildkanten definiert mit  $a = [\frac{w}{2} \ 0 \ 1]^T, b = [w \ \frac{h}{2} \ 1]^T, c = [\frac{w}{2} \ h \ 1]^T, d = [0 \ \frac{h}{2} \ 1]^T$ .

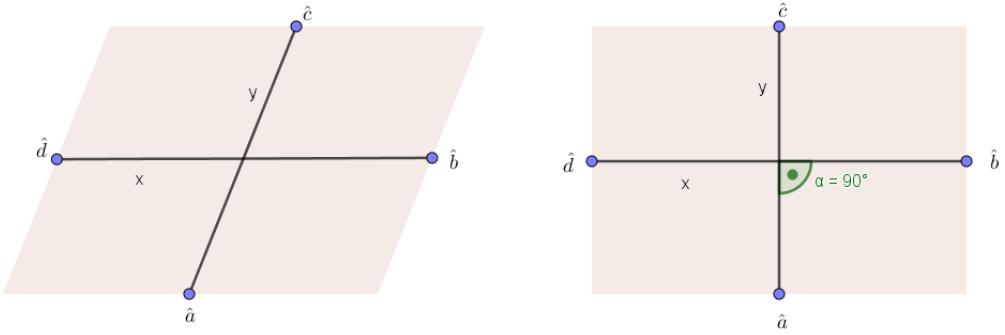


Abbildung 6.10: Die Verbindungslien  $\overline{bd}$  und  $\overline{ca}$  sollen so ausgerichtet werden, dass sie orthogonal zueinander stehen.

Die Punkte  $a, b, c, d$  und auch  $a', b', c', d'$  sind die Mittelpunkte der noch nicht rektifizierten Bildkanten. Diese werden mit den Matrizen  $H_r H_p$  und  $H'_r H'_p$  transformiert, um so die Position der Kantenmitten nach der Rektifizierung zu erhalten.

$$\hat{a} = H_r \cdot H_p \cdot a$$

$$\hat{b} = H_r \cdot H_p \cdot b$$

$$\hat{c} = H_r \cdot H_p \cdot c$$

$$\hat{d} = H_r \cdot H_p \cdot d$$

$$\hat{a}' = H'_r \cdot H'_p \cdot a'$$

$$\hat{b}' = H'_r \cdot H'_p \cdot b'$$

$$\hat{c}' = H'_r \cdot H'_p \cdot c'$$

$$\hat{d}' = H'_r \cdot H'_p \cdot d'$$

Danach können die Vektoren  $\vec{x}$  und  $\vec{y}$  der sich ursprünglich gegenüberliegenden Punkte gebildet werden.

$$x = \hat{b} - \hat{d} \quad (6.56)$$

$$y = \hat{c} - \hat{a} \quad (6.57)$$

$$x' = \hat{b}' - \hat{d}' \quad (6.58)$$

$$y' = \hat{c}' - \hat{a}' \quad (6.59)$$

$x$  und  $y$  sind Vektoren der euklidischen Bildebene[31]. Das heißt, sie sind genau dann orthogonal zueinander wenn gilt

$$(H_s x)^T (H_s y) = 0 \quad (6.60)$$

$$(H'_s x')^T (H'_s y') = 0 \quad (6.61)$$

Für die Erhaltung der Seitenverhältnisse gilt dann

$$\frac{(H_s x)^T (H_s x)}{(H_s y)^T (H_s y)} = \frac{w^2}{h^2} \quad (6.62)$$

$$\frac{(H'_s x')^T (H'_s x')}{(H'_s y')^T (H'_s y')} = \frac{w'^2}{h'^2}. \quad (6.63)$$

Anhand der Gleichungen 6.60, 6.61, 6.62 und 6.63 können die folgenden Ausdrücke für die Matrixeinträge  $u_a$  und  $u_b$  für  $H_s$  und  $H'_s$  aufgestellt werden[31, 36].

$$u_a = \frac{h^2 x_v^2 + w^2 + y_v^2}{hw(x_v y_u - x_u y_v)} \quad (6.64)$$

$$u_b = \frac{h^2 x_u x_v + w^2 y_u y_v}{hw(x_u y_v - x_v y_u)} \quad (6.65)$$

Die selben Gleichungen werden auch für  $u'_a$  und  $u'_b$  aufgestellt. Das Ergebnis der gesammten Transformation  $H$  mit  $H_s H_r H_p$  und  $H'$  mit  $H'_s H'_r H'_p$  ist in den Abbildungen 6.11 und 6.12 zu sehen.

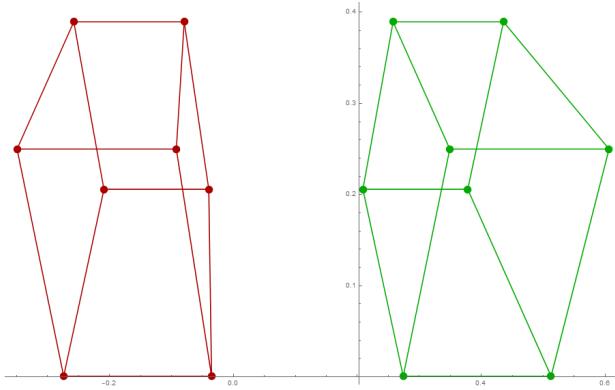


Abbildung 6.11: Abbildungen der Quader nach der Tranfromation mit  $H_s H_r H_p$  und  $H'_s H'_r H'_p$

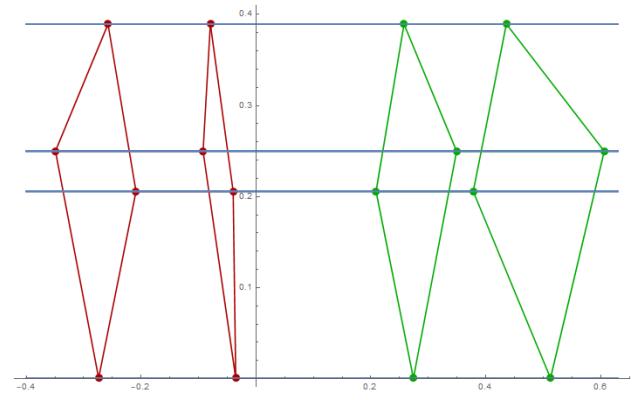


Abbildung 6.12: Abbildungen der Quader nach der Tranfromation mit  $H_s H_r H_p$  und  $H'_s H'_r H'_p$  mit eingezeichneten Epipolarlinien

### 6.3 Rektifizierung mit unterschiedlichen Kameraauflösungen

Im Folgenden wird der entstandene Rektifizierungsalgorithmus auf Bilder unterschiedlicher Auflösung angewandt. Im ersten Beispiel wird für  $C$  die Auflösung  $\zeta_x = \zeta_y = 1$  gewählt. Die Kameramatrix  $K$  lautet wie folgt

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.66)$$

Für  $C'$  wird eine kleinere Auflösung mit  $\zeta'_x = \zeta'_y = 0.5$  definiert. Die Kameramatrix  $K'$  lautet somit

$$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.67)$$

Die entstehenden Bilder des Quaders sind in Abbildung 6.13 zu sehen. Da  $K'$  nur eine halb so große Auflösung wie  $K$  besitzt, ist das resultierende Bild des roten Quaders auch nur halb so groß. Der grüne Quader zeigt das entstehende Bild von  $C'$ .

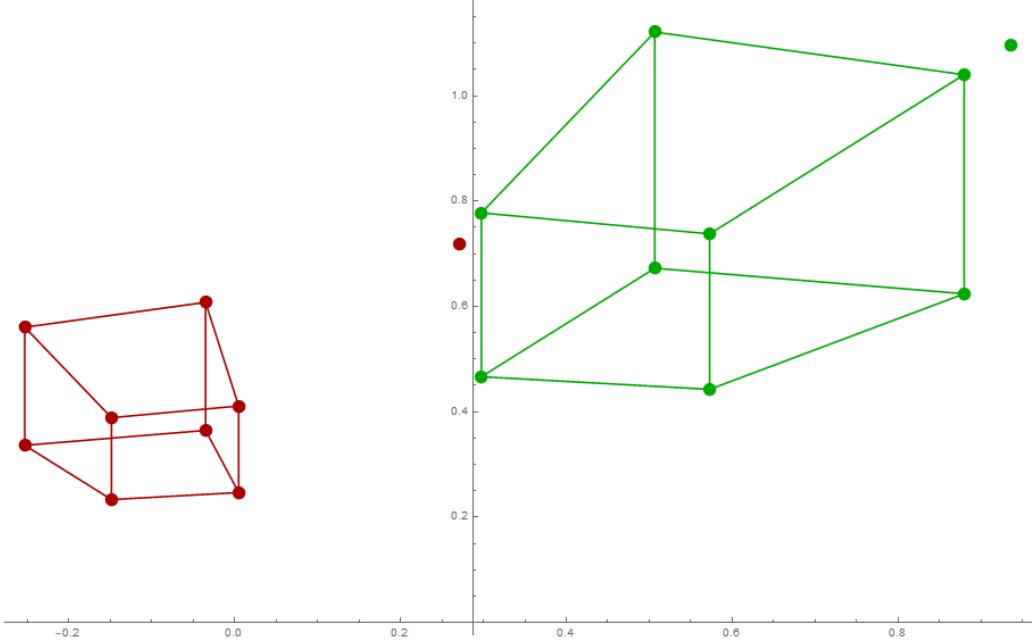


Abbildung 6.13: Aufnahmen zweier Kameras mit unterschiedlichen Auflösungen. Für Kamera eins(Grün) gilt  $\zeta_x = \zeta_y = 1$  und für Kamera zwei(rot) gilt  $\zeta'_x = \zeta'_y = 2$ .

Abbildung 6.14 zeigt die projektive Transformation der beiden Bilder mit  $H_p$  und  $H'_p$ . Die Epipolarlinien sind nach dieser Transformation jeweils parallel zueinander. In Abbildung 6.15 ist das Resultat zu sehen, wenn die Transformationen  $H_r$  und  $H'_r$  dazukommen. Die Epipolarlinien sind jetzt parallel zur horizontalen Achse und die Epipolarlinien von Bild eins und Bild zwei sind so zueinander ausgerichtet, dass sie zu einheitlichen Linien über zwei Bilder werden. Der rote Quader, welches das Bild mit der niedrigeren Auflösung repräsentiert, ist nach dieser Transformation vergrößert.

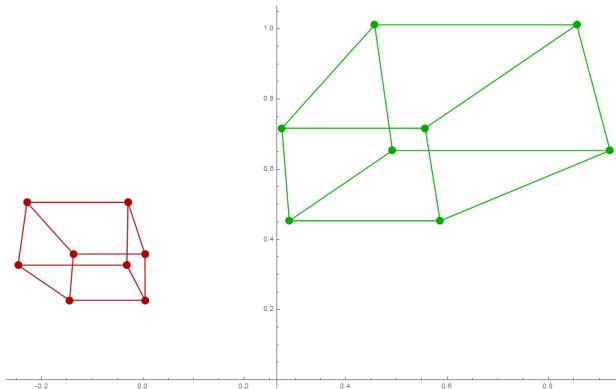


Abbildung 6.14: Transformation  $H_p$  und  $H'_p$  angewandt auf Bilder unterschiedlicher Auflösungen

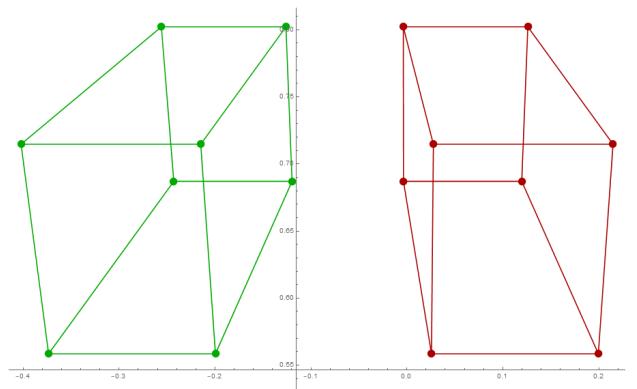


Abbildung 6.15: Transformation  $H_rH_p$  und  $H'_rH'_p$  angewandt auf Bilder unterschiedlicher Auflösungen

Die Abbildungen 6.16 und 6.17 zeigen die letzte Transformation mit  $H_s$  und  $H'_s$ . Die Bilder scheinen richtig rektifiziert worden zu sein. Dies wurde noch mit weiteren Vielfachen der Kameramatrix  $K$  ausprobiert. Für alle getesteten Fälle ergab sich dasselbe Ergebnis, wie es in den Abbildungen 6.16 und 6.17 zu sehen ist. Die Transformationen während der Rektifizierung können unterschiedliche Auflösungen ausgleichen und es ist folglich möglich eine Tiefenkarte zu erstellen um die 3D Szene zu rekonstruieren.

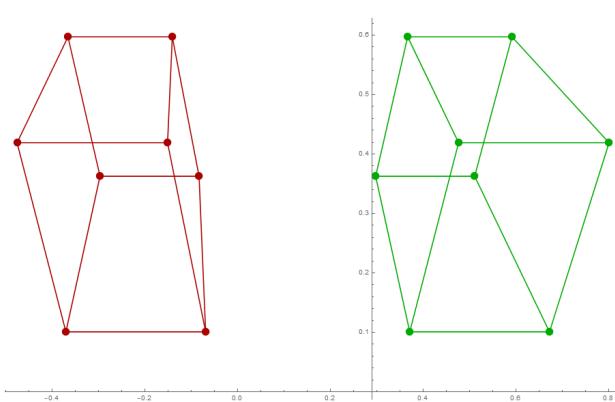


Abbildung 6.16: Transformation  $H_sH_rH_p$  und  $H'_sH'_rH'_p$  angewandt auf Bilder unterschiedlicher Auflösungen

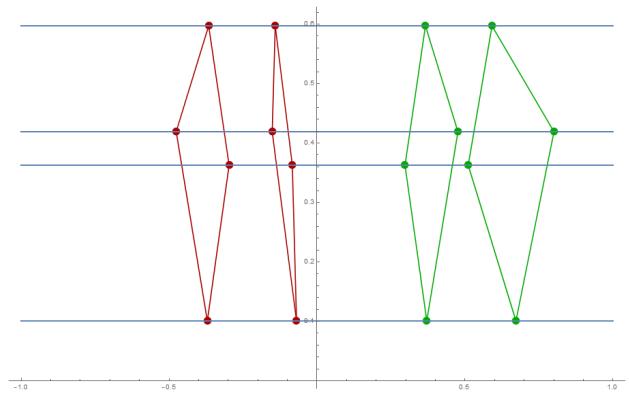


Abbildung 6.17: [Transformation  $H_sH_rH_p$  und  $H'_sH'_rH'_p$  angewandt auf Bilder unterschiedlicher Auflösungen mit Epipolarlinien

Als nächstes wurden die Auflösung von  $C'$  so verändert, dass  $\zeta'_x \neq \zeta'_y$  gilt. Es werden  $\zeta'_x = 2.3$  und  $\zeta'_y = 3.2$  definiert. Somit folgt für  $K'$  die folgende Matrix

$$K' = \begin{bmatrix} 2.3 & 0 & 0 \\ 0 & 3.2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.68)$$

Für  $K$  von  $C$  gilt weiterhin  $\zeta_x = \zeta_y = 1$ . Die entstandenen Bilder sind in Abbildung 6.18 zu sehen. Die horizontale Kantenlänge des roten Quaders ist im Verhältnis kürzer als ihre vertikale Kantenlänge. Wie in Abbildung 6.19 zu sehen ist, bleibt das ungleiche veränderte Seitenverhältnis des roten Quaders nach der Rektifizierung erhalten, weshalb der rote Quader schmäler ist als der grüne.

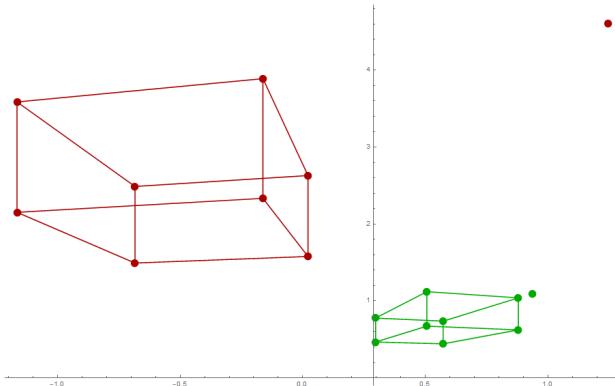


Abbildung 6.18: Aufnahmen zweier Kameras mit unterschiedlichen Auflösungen. Für Kamera eins(grün) gilt  $\zeta_x = \zeta_y = 1$  und für Kamera zwei(rot) gilt  $\zeta'_x = 2.3$   $\zeta'_y = 3.2$

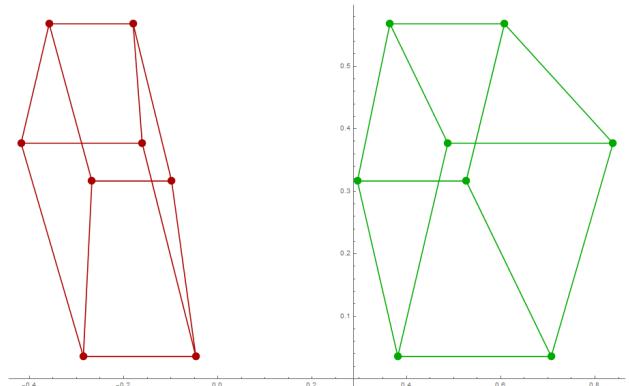


Abbildung 6.19: Nach der Rektifizierung stimmen die horizontalen Koordinaten nicht überein, es würde zu Fehlern in der Tiefenkarte und somit in der gesamten 3D-Rekonstruktion geben.

Diese Beispiele zeigen, dass das hier vorgestellte Rektifizierungsverfahren auf Bilder verschiedener Auflösungen mit demselben Seitenverhältnis der Pixel anwendbar ist. Jedoch können Bilder mit verschiedenen Seitenverhältnissen mit dem Verfahren nicht rektifiziert werden. Eine mögliche Lösung wäre eine Erweiterung des Modells, welche die Bildlängen bestimmt und schließlich die relativen Seitenverhältnisse ausgleicht.

## 7 Punktesortierung in Schachbrettmustern

Um Punktekorrespondenzen in stereoskopischen Bildaufnahmen von zweidimensionalen Schachbrettern zu ermitteln, ist ein Algorithmus entwickelt worden, welcher zuvor detektierte Eckpunkte eines Schachbretts sortiert und eindeutig identifiziert. Jeder Punkt beinhaltet nach der Sortierung die Information in welcher Reihe  $j$  und in welcher Spalte  $i$  er sich befindet. Jeder Punkt ist somit eindeutig durch die zwei Indizes  $i$  und  $j$  identifiziert. Dem Sortierungsalgorithmus ist ein Algorithmus zu Detektion der Eckpunkte eines Schachbretts voran geschaltet. Werden beide Algorithmen auf die Stereoaufnahme zweier Schachbretter angewandt, so können korrespondierende Punkte anhand der zugewiesenen Indizes ausgemacht werden. Die Schachbretter können dabei sowohl Kissen- als auch Tonnennverzeichnungen aufweisen und oder perspektivisch verzerrt sein.

Aus den zuvor detektierten Eckpunkten des Schachbretts wird ein Startpunkt ermittelt. Der Startpunkt wird so bestimmt, dass es immer die linke unterste Ecke des Schachbretts ist. Somit ist gewährleistet, dass die Indizes der Eckpunkte beider Bilder gleich sind. Ist der Startpunkt bestimmt, bekommt dieser die Indizes  $i = 1$  und  $j = 1$ .  $i$  steht für die jeweilige Reihen in welcher sich ein Punkt befindet und  $j$  steht für die jeweilige Spalte. Ist der Startpunkt bestimmt, wird der erste Punkt entlang der unteren Schachbrettkante in  $j$ -Richtung und der erste Punkt entlang der linken Randkante in  $i$ -Richtung gesucht. Anhand dieser Punkte lassen sich vom Startpunkt aus Richtungsvektoren definieren. Entlang des Richtungsvektors wird ein Suchbereich definiert. In Abbildung 7.1 ist dieser Suchbereich als das blaue Dreieck dargestellt. Ist ein neuer Punkt gefunden, so wird der Richtungsvektor anhand des neuen und des vorherigen Punktes neu ausgerichtet. Der dynamische Suchbereich ermöglicht es, dass selbst bei stark verzerrten Schachbrettern, Punkte der einzelnen Reihen und Spalten geordnet werden können, trotz dass diese nicht mehr auf einer Gerade liegen.

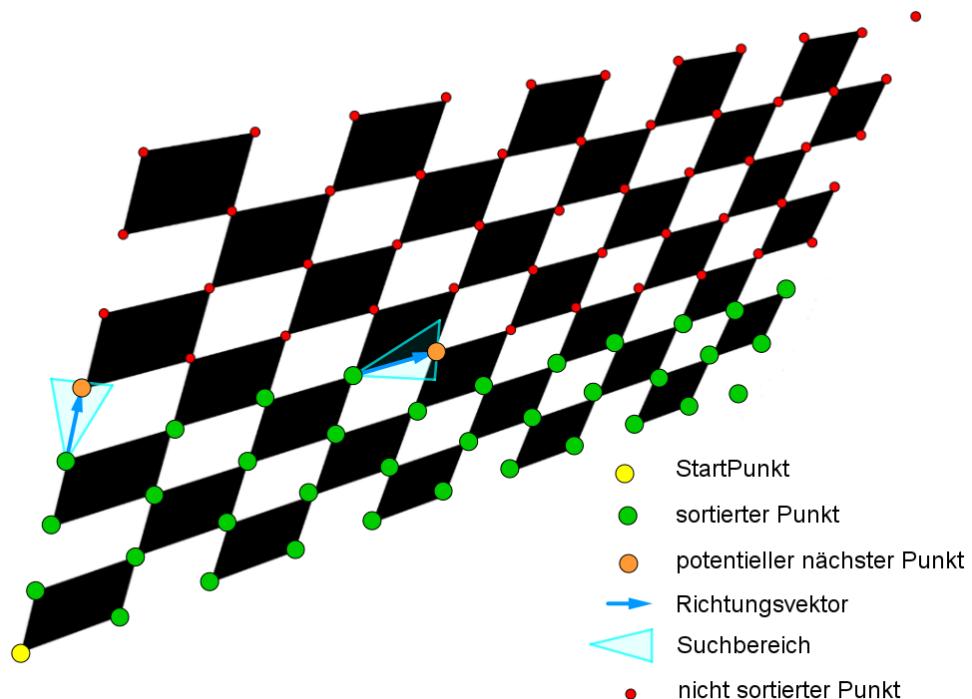


Abbildung 7.1: Schematik der Vorgehensweise des Sortieralgorithms

## 7.1 Sortierungsalgorithmus

Im Folgenden soll der Ablauf des implementierten Sortierungsalgorithmus beschrieben und dessen Funktionsweise an Beispielen von unterschiedlichen Schachbrettern demonstriert werden.

Der Sortierungsalgorithmus nimmt eine Liste aus unsortierten Eckpunkten eines Schachbretts entgegen. Von den Punkten in dieser Liste wird eine grobe Vorsortierung vorgenommen. Um das Schachbrett herum wird ein Rahmen definiert. Die Punkte mit der maximalen  $y$ -Koordinate und der minimalen  $y$ -Koordinate begrenzen die oberen und unteren Kanten des Rahmens. Die Punkte mit der minimalen  $x$ -Koordinate und der Punkt mit der maximalen  $x$ -Koordinate begrenzen die vertikalen Kanten des Rahmens. In Abbildung 7.2 sind die begrenzenden Kanten in rot um das Schachbrett zu sehen.

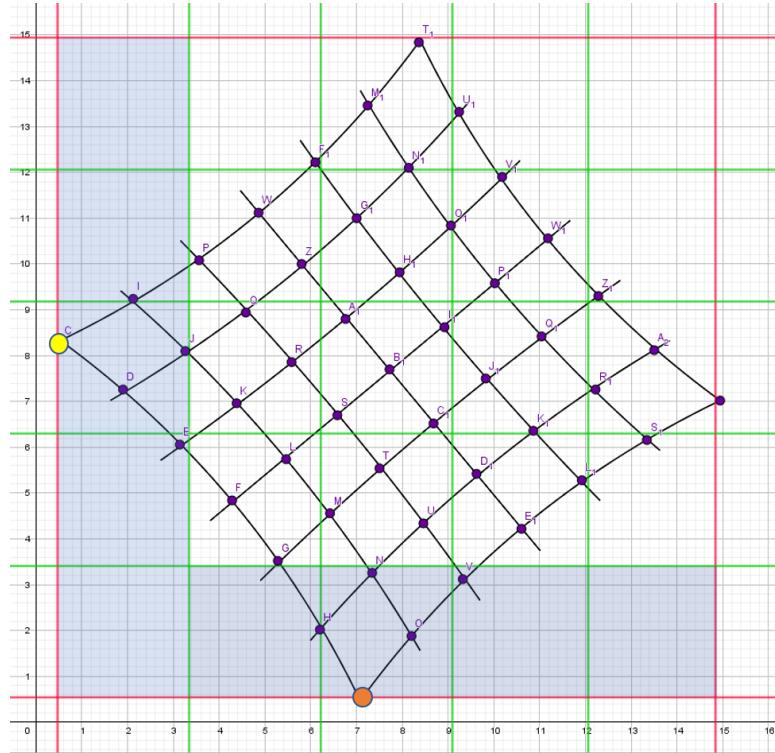


Abbildung 7.2: Die in blau markierten Bereiche beinhalten die möglichen Startpunkte. Der Bereich entlang der horizontalen  $j$ -Achse bildet die erste Suchfensterreihe in  $i$ -Richtung. Der blaue Bereich entlang der vertikalen  $i$ -Achse bildet die erste Suchfensterreihe in  $j$ -Richtung. Der gelbe Punkt steht für den Punkt welcher als  $VecJ$  bezeichnet wird und der orangefarbene Punkt ist derjenige Punkt, welcher als  $VecI$  bestimmt wird

Der durch den Rahmen begrenzte Bereich, welcher das gesamte Schachbrett einschließt, wird in mehrere Zellen eingeteilt. Diese werden durchgezählt und mit den Indizes  $i$  und  $j$  eindeutig bestimmt.  $i$  beschreibt die Reihennummer der Zelle und  $j$  beschreibt die entsprechende Spalte. In Abbildung 7.2 hätte die Zelle links unten im Eck somit die Indizes  $i = 1$  und  $j = 1$ , die zweite rechts daneben die Indizes  $i = 1$  und  $j = 2$ .

In zwei *ConstantArrays* namens *JSplits* und *ISplits* werden die Begrenzungen der Zellen in  $x$ - und  $y$ -Richtung gespeichert. Die Größe und Anzahl der Zellen werden berechnet, indem die Distanz zwischen den vertikalen und horizontalen Kanten des Rahmens durch die gewünschte Anzahl an Zellen geteilt wird. Im nächsten Schritt wird überprüft in welcher Zelle jeder Punkt liegt. Für jeden Punkt wird eine Liste *Aus Associations* angelegt. *Associations* weisen Werten Schlüssel zu. Anhand dieser Schlüssel sind Werte eindeutig identifiziert[22]. Pro Punkt werden jeweils die  $x$ - und  $y$ -Koordinaten sowie die Indizes der Zellen in Schlüssel gespeichert.

$$Punkt = \{ < | Coordx \rightarrow x_u, Coordy \rightarrow y_u, CellI \rightarrow i_u, CellJ \rightarrow j_u | > \}$$

Nach der Vorsortierung wird der Startpunkt für die Rekonstruktion des Schachbretts gesucht. Von diesem Startpunkt aus soll das Schachbrettgitter rekonstruiert werden. Alle Punkte innerhalb der Zellen der ersten Reihe mit den Indizes  $i = 1$  und der Spalten  $j \leq j_{all}$  und die Punkte der ersten Spalte mit Indizes  $i \leq i_{all}$  und  $j = 1$  werden als mögliche Startpunkte gekennzeichnet. In Abbildung 7.2 befinden sich die möglichen Startpunkte innerhalb des blau hinterlegten Bereichs.

Die jeweiligen horizontalen Zellen  $i = 1$  und  $j \leq j_{max}$  werden getrennt von den vertikalen  $i \leq i_{all}$  und  $j = 1$  untersucht. Innerhalb der horizontalen Zellen wird zunächst nach dem Punkt mit der kleinsten  $y$ -Koordinate gesucht und als erster möglicher Startpunkt in  $VecI$  gespeichert. Danach wird überprüft, ob es einen Punkt gibt, dessen  $x$ -Koordinate kleiner ist als die des momentan gesetzten  $VecI$ . Ist dies der Fall so wird noch überprüft, ob dessen  $y$ -Koordinate kleiner gleich der von  $VecI$  plus einem Pufferwert ist. Der Pufferwert wird so definiert, dass die  $y$ -Koordinate des neuen möglichen  $VecI$  zwar größer als die des momentanen sein kann, jedoch eine bestimmte Schwelle nicht übertreten darf, da es sich sonst um einen Punkt der zweiten  $i$ -Reihe handeln könnte. Für eine mathematische Schätzung des Pufferwerts existieren Ansätze, jedoch wird dieser momentan noch selbstständig gesetzt. In Abbildung 7.2 ist  $VecI$  als oranger Punkt abgebildet.

Innerhalb des vertikalen Suchbereichs bestehend aus den Zellen  $i \leq i_{all}$  und  $j = 1$  wird derjenige Punkt als  $VecI$  gesetzt, dessen  $x$ -Koordinate minimal ist. Im nächsten Schritt wird ein Punkt innerhalb des Bereichs gesucht, dessen  $y$ -Koordinate kleiner ist als die des momentanen  $VecJ$  und dessen  $x$ -Koordinaten kleiner gleich der  $x$ -Koordinate von  $VecJ$  plus einem Pufferwert ist. Ist so ein Punkt gefunden, wird dieser als neuer  $VecJ$  bestimmt.  $VecJ$  ist in Abbildung 7.2 als gelber Punkt dargestellt.

Je nachdem wie das Schachbrett rotiert ist oder welche Art der Verzerrungen es aufweist, kann es sein dass  $VecI$  und  $VecJ$  bereits den selben Punkt ergeben haben, was den Startpunkt *StartPoint* eindeutig identifiziert, wie es in den Beispielen 7.11 oder auch 7.15 der Fall ist. Andererseits kann es auch sein, dass  $VecI$  und  $VecJ$  sich unterscheiden, wie es in Abbildung 7.2 zu sehen ist. In solchen Fällen wurde  $VecJ$  als Standard Startpunkt festgelegt. Es kann auch  $VecI$  gewählt werden, da in diesen Fällen nicht eindeutig gesagt werden kann, welches die linke unterste Ecke des Schachbretts ist. Zu beachten ist dabei, dass bei beiden Bildern der selbe Punkt als Startpunkt gewählt wird.

Die *Assiciation*-Liste des Startpunktes wird um zwei weitere Schlüssel erweitert. Die Schlüssel *NeighbourI* und *NeighbourJ* speichern ab in welcher Reihe  $i$  und Spalte  $j$  des Schachbrettgitters sich der gefundenen Punkt befindet. Der Startpunkt *StartPoint* ist dann wie folgt definiert

$$\begin{aligned} StartPoint = \{ &< | Coordx \rightarrow x_{-u}, Coordy \rightarrow y_u, \\ &CellI \rightarrow i_u, CellJ \rightarrow j_u, NeighbourI \rightarrow 1, NeighbourJ \rightarrow 1 | > \} \end{aligned}$$

Anhand der Schlüssel *NeighbourI* und *NeighbourJ*, ist die Position des Punktes innerhalb des Schachbrettgitters eindeutig identifiziert. In den Abbildungen 7.9 und 7.10 wurde in beiden Bildern alle Punkte abgefragt, deren Schlüssel *NeighbourI* = 3 ist. Die grün gefärbten Punkte liefern das Ergebnis.

Der Startpunkt *StartPoint* wird danach in zwei unterschiedliche Listen geschrieben. Die Liste *SortedPoints* speichert alle fertig sortierten Punkte. Die Liste *CheckList* speichert eine verkürzte *Associtaion*-Liste der Punkte, welche nur aus den Schlüsseln *Coordx* und *Coordy* besteht. Mit Hilfe der in *Checklist* ge-

speicherten Koordinaten eines Punktes, soll bei der weiteren Suche verhindert werden, dass ein Punkt mehrmals in *SortedPoints* sortiert wird.

Nachdem der Startpunkt gefunden ist werden die jeweils nächsten Punkte in *i*- und *j*-Richtung gesucht. Diese Punkte werden dann in die Variablen *NextI* und *NextJ* gespeichert. In Abbildung 7.3 ist *NextI* der violette Punkte und *NextJ* ist der rote Punkte.

Für die Bestimmung von *NextI* wird zuerst initial  $NextI = < |Coordx \rightarrow 100000, Coordy \rightarrow 100000| >$  gesetzt. Für den Punkt *NextI* wird die Zelle in welchem sich *StartPoint* befindet und die jeweiligen Zellen eins darüber und darunter durchsucht. In Abbildung 7.3 ist dieser Bereich um *StartPoint* gelb hinterlegt.

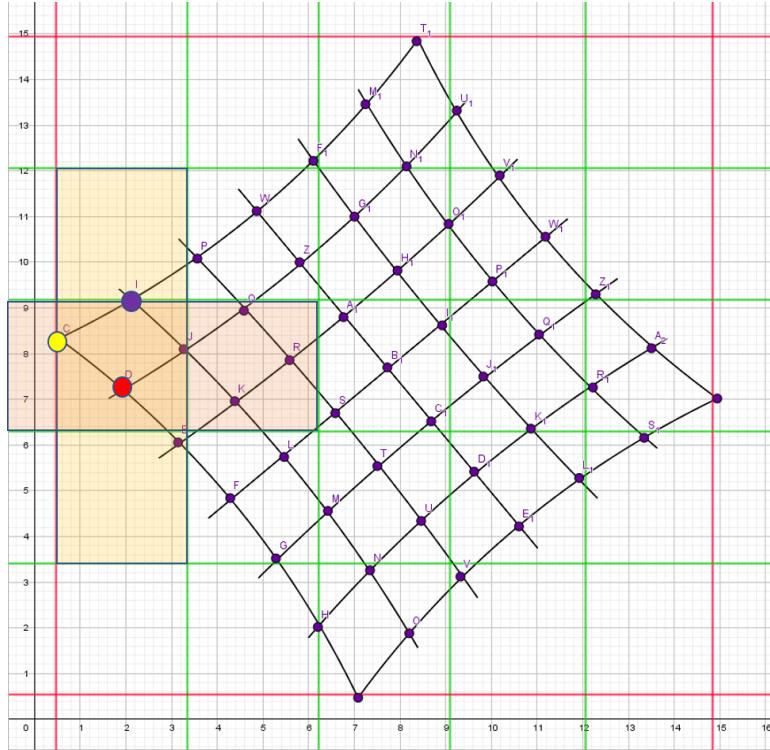


Abbildung 7.3: Der in gelb markierte Bereich beinhaltet die für *NextI* möglichen Punkte. der orange markierte Bereich beinhaltet die für *NextJ* möglichen Punkte. Der violett gefärbte Punkt ist der gesuchte *NextI*, der rot gefärbte Punkt ist der gesuchte *NextJ*

Innerhalb des Bereichs wird der Punkt mit der zu *StartPoint* nächst höheren *y*-Koordinate ermittelt. Dieser Punkt wird dann als vorläufiges *NextI* festgelegt. Danach wird geprüft, ob das neu bestimmte *NextI* bereits das letztendliche *NextI* ist. Dazu werden im gelben Bereich alle Punkte nochmals durchgegangen um zu testen, ob es einen Punkt gibt, welcher einen kleineren *x*-Koordinatenabstand vom Startpunkt besitzt als *NextI*. Gleichzeitig wird geprüft, ob der *y*-Koordinatenabstand vom Startpunkt aus zum neu gefundenen Punkt kleiner ist als der *y*-Koordinatenabstand vom Startpunkt zu *NextI*. Ist dies der Fall, so wird der Punkt auf den beides zutrifft als neues *NextI* bestimmt. In Abbildung 7.4 ist die letzte Abfrage noch einmal grafisch dargestellt. Ist *NextI* ermittelt, kann dieser Wert im Folgenden nicht mehr als *NextJ* bestimmt werden. Das kann in Situationen wie in Abbildung 7.3 verhindern, dass für *NextI* und *NextJ* der selbe Punkt ermittelt wird.

Die Suche nach dem nächsten Punkt in *j*-Richtung erfolgt nach dem gleichen Prinzip. Für *NextJ* wird der in Abbildung 7.3 rot hinterlegte Bereich untersucht. Innerhalb des roten Bereichs, wird derjenige Punkt ermittelt der zu *StartPoint* den nächst höheren *x*-Wert besitzt. Dieser wird dann als vorläufiger *NextJ* gespeichert. Danach werden alle Punkte im roten Bereich noch einmal durchgegangen und geprüft, ob es einen Punkt gibt, dessen *y*-Koordinatenabstand zum Startpunkt kleiner ist. Gleichzeitig wird abgefragt, ob der *x*-Koordinatenabstand vom Startpunkt aus zum neu gefundenen Punkt kleiner

ist als der momentane  $x$ -Koordinatenabstand vom Startpunkt zu  $NextJ$ .

Die beiden Punkte  $NextI$  und  $NextJ$  werden dann wie  $StartPoint$  um die zwei Schlüssel  $NeighbourI$  und  $NeighbourJ$  erweitert. Die Punkte sind dann wie folgt definiert und werden in die Liste  $SortedPoints$  gespeichert.

$$NextI = \{ < |CoordI \rightarrow i - \text{Koordinate}, CoordJ \rightarrow j - \text{Koordinat}, \\ CellI \rightarrow i - \text{Zelle}, CellJ \rightarrow j - \text{Zelle}, NeighbourI \rightarrow 2, NeighbourJ \rightarrow 1| > \}$$

$$NextJ = \{ < |CoordI \rightarrow i - \text{Koordinat}, CoordJ \rightarrow j - \text{Koordinat}, \\ CellI \rightarrow i - \text{Zelle}, CellJ \rightarrow j - \text{Zelle}, NeighbourI \rightarrow 1, NeighbourJ \rightarrow 2| > \}$$

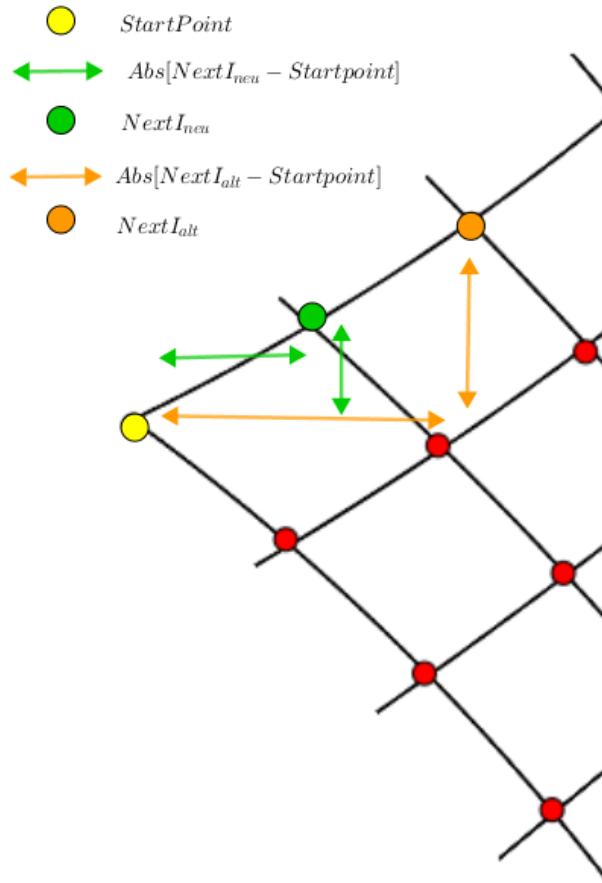


Abbildung 7.4: Um zu überprüfen, ob der momentan  $NextI$ , in der Abbildung der orange Punkt, wirklich der nächste Punkt ist, wird sein Abstand zum Startpunkt in  $j$ - und  $i$ -Richtung, mit den anderen noch möglichen Punkten verglichen. Gibt es einen dessen beide Abstände kleiner sind, wie beispielsweise der grüne Punkt, so wird dieser zum neuen  $NextI$  bestimmt.

Mit den Punkten  $StartPoint$ ,  $NextI$  und  $NextJ$  können die ersten Richtungsvektoren in  $i$ -Richtung mit  $NextIDir = NextI - StartPoint$  und  $j$ -Richtung mit  $NextJDir = NextJ - StartPoint$  gebildet werden. Anhand der Richtungsvektoren werden Suchbereiche definiert, in welchen der jeweils nächste Punkt in entsprechender Richtung  $i$  oder  $j$  gesucht werden soll.

Für beide Richtungen werden zwei Listen  $IList$  und  $JList$  angelegt. In den Listen werden jeweils die Punkte der ersten beiden Zellen in  $j$ - und  $i$ -Richtung gespeichert. In Abbildung 7.5 entspricht  $IList$  dem vertikalen und  $JList$  dem horizontalen blauen Bereich.

Der Algorithmus geht dann nach folgendem Schema vor. Zuerst wird die untere Schachbrettkante vervollständigt. Hierzu wird eine Schleife implementiert, welche den im Folgenden beschriebenen Suchvorgang so lange durchgeht, bis keine Punkte mehr gefunden werden. Danach wird der Nächste Punkt der linken Randkante in  $i$ -Richtung gesucht, von welchem aus wieder die nächste Reihe in  $j$ -Richtung vervollständigt wird.

Für die Vervollständigung der ersten Reihe in  $j$ -Richtung, wird der Richtungsvektor  $NextDirJ$  um einen Puffer erweitert. Dieser Puffer berechnet sich aus einem drittel der Länge von  $NextIDir$ . Neben dem Richtungsvektor  $NextJDir$  wird noch ein Vektor  $SearchAreaJ$  definiert, welcher den  $i$ -Koordinatenabstand zwischen  $StartPoint$  und  $NextJ$  beinhaltet und den Suchbereich um den Richtungsvektor  $NextJDir$  aufspannt.  $SearchAreaJ$  wird ebenfalls um einen Pufferwert erweitert. Dieser besteht aus einem drittel der Länge des  $i$ -Koordinatenabstands von  $Startpoint$  zu  $NextJ$ . In Abbildung 7.1 ist dieser Puffer als Blauer Kegel um den Richtungsvektor zu sehen. Dieser Kegel definiert den Suchbereich von einem bereits bekannten Punkt zu einem noch unbekannten Punkt. In Abbildung 7.5 auf dem linken Bild wird der Richtungsvektor  $NextJDir$  als Pfeil vom gelben  $StartPoint$  zum roten  $NextJ$  dargestellt. Die Pfeile welche in  $i$ -Richtung von  $NextJ$  aus ausgehen, bilden die  $SearchAreaJ$ . Der Bereich den die Vektoren aufspannen bildet den Suchbereich für den auf  $NextJ$  folgenden Punkt.

$$NextJDir = (NextI - StartPoint) + Puffer$$

$$SearchAreaJ = (NextI[CoordI] - StartPoint[CoordI]) + Puffer$$

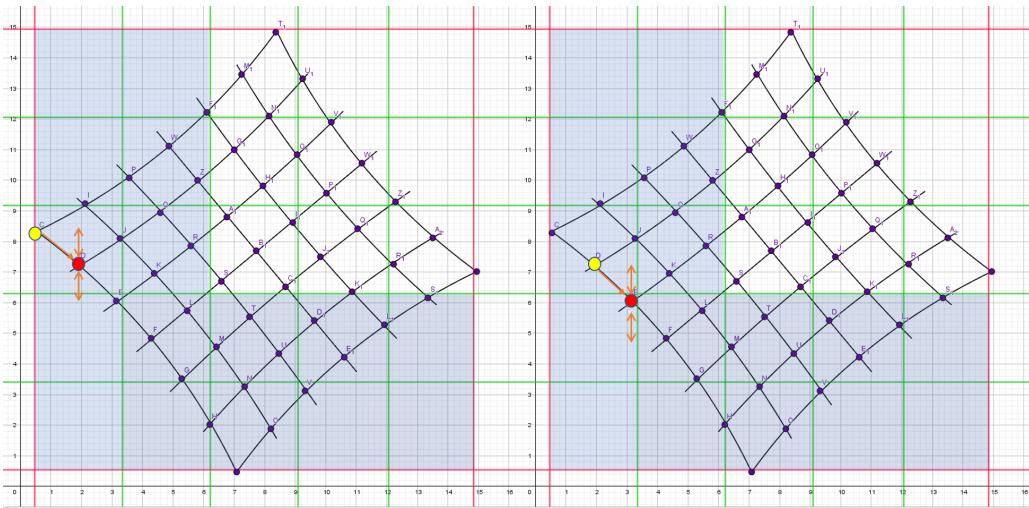


Abbildung 7.5: Im linken Bild ist der Startpunkt in gelb dargestellt und der bereits gefundenen Punkt  $NextJ$  ist in rot eingefärbt. Anhand der beiden Punkte wird ein Richtungsvektor definiert und zwei Pufferwerte oberhalb und unterhalb. Diese bilden einen Suchbereich. Der zuvor definierte Suchbereich wird vor  $NextJ$  gesetzt und es wird nach einem potentiellen nächsten Punkt in der Reihe gesucht. Ist ein Punkt gefunden, wird  $NextJ$  zum neuen  $StartPoint$  umdefiniert und der neu gefundene Punkt wird zu  $NextJ$ .

Der Ablauf der Schleife zur Vervollständigung einer Reihe wird im Folgenden genauer beschrieben. Der definierte Suchbereich bestehend aus *NextJDir* und der *SearchAreaJ* wird vor *NextJ* definiert. Es werden alle Punkte von *JList* durchgegangen. Zunächst wird geprüft, ob es einen Punkte innerhalb der Liste gibt, der nicht *StartPoint* ist und dessen *x*-Koordinate größer ist die *x*-Koordinate von *NextI*. Gleichzeitig wird abgefragt, ob dessen *y*-Koordinate kleiner gleich oder größer gleich der von *NextJ* ist.

Ein solcher Punkt wird als potentieller nächster Punkt weiter überprüft. Es wird sowohl die Distanz *distanceJ* des Richtungsvektors *NextJDir* als auch die Distanz von *NextJ* zum gefundenen potentiellen Punkt berechnet *distanceNextPotPointJ*. Um zu überprüfen, ob der gefundenen Punkt innerhalb des Suchbereiches liegt, welcher von *NextJ* und *SearchArea* aufgespannt ist, wird der potentielle Punkt auf folgendes getestet. Die *y*-Koordinatenposition des potentiellen Punktes soll zwischen der *y*-Koordinate von *NextJ* plus dem Wert von *SearchArea* und der *y*-Koordinate von *NextJ* minus dem Wert der *SearchArea* liegen. Des Weiteren darf seine Distanz *distanceNextPotPointJ* nicht größer werden als *distanceJ* plus einem Puffer, welcher ein drittel von *distanceJ* beinhaltet. Zudem darf seine Distanz aber auch nicht kleiner sein als die Hälfte von *dinstanceJ*. Als letztes wird mit der *CheckList* noch sichergestellt, dass der gefundenen Punkt nicht bereits in der *SortedList* gespeichert wurde.

Ist ein solcher Punkt gefunden, so wird *NextJ* zum neuen *Startpunkt* und der potentielle Punkt wird zum neuen *NextJ* deklariert. Es werden der neue Richtungsvektoren *NextJDir* aus dem neuen *StartPoint* und *NextJ* berechnet, sowie eine neue *SearchAreas* aus der *i*-Koordinatendifferenz der beiden neuen Punkte *StartPoint* und *NextJ*. *NextJ* wird dann noch mit den neuen Schlüsseln *NeighbourJ* und *NeighbourI* in die Liste *SortedPoints* gespeichert. Danach beginnt die Schleife mit der Suche nach dem nächsten Punkt in *j*-Richtung. Für jeden weiteren Punkt entlang der Reihe, wird der Bereich der möglichen Punkte neu definiert. Es werden alle Punkte innerhalb der Zellen um den neuen *NextJ* als neue Potentielle Punkte angesehen. Durch diese Begrenzung des Suchbereichs wird der Suchaufwand auf wenige Punkte reduziert.

Sind alle Punkte in einer Reihe sortiert, so wird der nächste Punkt *NextI* entlang der äußeren Kante gesucht. Vom neuen *NextI* aus startet dann wieder die Schleife, welche die Reihe in *j*-Richtung vervollständigt. Die Suche nach *NextI* verfährt nach dem selben Verfahren wie *NextJ* nur wird nach einem neuen gefundenen Punkt für *NextI* die Funktion für die Vervollständigung der Reihe dazwischen geschaltet.

Bevor die Suche entlang der einer Reihe oder auch der äußeren Kante als beendet gilt, treten noch zwei Sicherheitsfunktionen in Kraft. Die erste wird als *Saftylist* bezeichnet. *Saftylist* sorgt dafür, dass alle Punkte der unteren und der linken Kante des Schachbretts vollständig detektiert werden. Wird beispielsweise kein weiterer Punkt bei der Detektierung der außen Kanten innerhalb der Bereiche von *JList* oder *IList* gefunden, so wird der Suchbereich kurzfristig erweitert. Um den letzten gefundenen Punkt innerhalb der Listen werden, sofern vorhanden, alle noch nicht abgesuchten Zellen abgesucht. Gibt es in einer darum liegenden Zelle noch einen Punkt, der in den Suchbereich für den nächsten Punkt fällt, wird dieser noch mit aufgenommen. Die Funktion der *SaftyList* kommt beispielsweise genau dann zum Einsatz, wenn die Punkte eines Schachbretts wie in 7.6 sortiert werden. In der linken Abbildung ist zu sehen, dass der Bereich der *IList* endet, es aber noch weitere Punkte gibt. Auf der rechten Abbildung ist dann zu sehen, wie der rot hinterlegte Bereich noch hinzugenommen wird, sodass die restlichen Punkte der Kante auch noch gefunden werden.

Die zweite erwähnte Sicherheitsfunktion ist die *PlaceSyntheticPoint*. Zu Beginn des Sortierungsalgoritmus nimmt dieser eine Punkteliste des voran geschalteten Eckpunktendektionsalgoritmus entgegen. Bei der Detektion der Eckpunkte, kann es vorkommen, dass Punkte nicht erkannt wurden und somit Lücken im Schachbrettgitter vorhanden sind. Trifft der Sortierungsalgoritmus auf eine solche Lücke, könnte er keinen weiteren Punkt detektieren und würde mit der Suche in der entsprechenden Reihe aufhören. Jedoch können sich hinter der Lücke befinden noch weitere Punkte befinden,

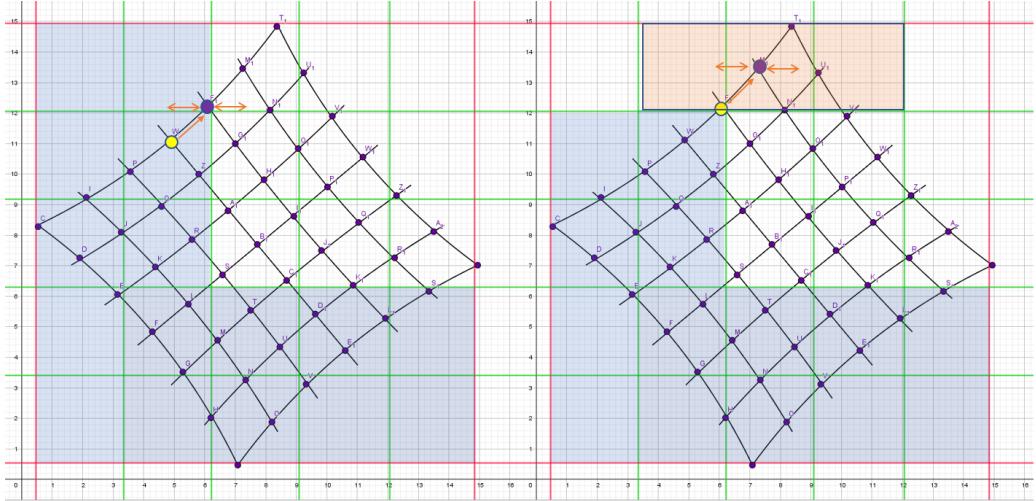


Abbildung 7.6: Wird der kein weiterer Punkt innerhalb von *IList* oder *JList* gefunden, Wird ein kleiner Bereich, hier in rot hinterlegt, abgesucht, ob es doch noch potentielle nächste Punkte gibt, die aufgrund der Lage des Schachbretts nich in *IList* oder *JList* mit enthalten waren

welche aber nicht mehr gefunden und sortiert werden. Diese Punkte würden dem entsprechend nicht in der *SortedList* gespeichert und der Algorithmus würde am Ende die Nachricht ausgeben, dass die Sortierung unvollständig sei, da noch unsortierte Punkte übrig sind.

Um dem entgegen zu wirken wurde die Sicherheitsfunktion *PlaceSyntheticPoint* entwickelt. Sollte vorerst kein Punkt im Suchbereich entdeckt werden, so setzt die Sicherheitsfunktion einen synthetischen Punkt und sucht ausgehend von diesem weiter nach Punkten. Sollte daraufhin ein weiterer Punkt gefunden werden, so bleibt der synthetische Punkt bestehen und die Suche wird normal fortgesetzt. Der synthetische Punkt wird nicht in die Liste *SortedList* mit aufgenommen, da er später nicht als möglicher korrespondierender Punkt in betracht gezogen werden soll. Er dient lediglich dazu alle real vorhandenen Punkte zu finden und richtig zu sortieren. Wird nach dem setzen des synthetischen Punkts kein weiterer Punkt gefunden, gilt die Suche in der Reihe beziehungsweise Spalte für beendet und der synthetische Punkt wird wieder gelöscht.

In Abbildung 7.7 und 7.8 wird ein Schachbrett aus zwei unterschiedlichen Blickwinkeln dargestellt. Die roten Punkte markieren die Eckpunkte, deren Koordinaten an den Sortierungsalgorithmus übergeben werden. In den Abbildungen 7.9 und 7.10 werden alle Punkte, welche in der Liste *SortedPoints* gespeichert wurden in einem Plot ausgegeben. Alle Punkte deren Schlüssel *NeighbourI* = 3 ist, sind in grün dargestellt. Das Ergebnis dieser Abfrage ist in den Abbildungen 7.9 und 7.10 zu sehen. Die dargestellten Bilder könnten zwei stereoskopische Abbildungen eines Schachbrettes sein. Der entwickelte Sortierungsalgorithmus kann auf diese Schachbretter angewendet werden und sortiert die Punkte im Schachbrett, sodass korrespondierende Punkte bestimmt werden können und die in den vorherigen Kapiteln beschriebene Szenenrekonstruktion angewandt werden kann.

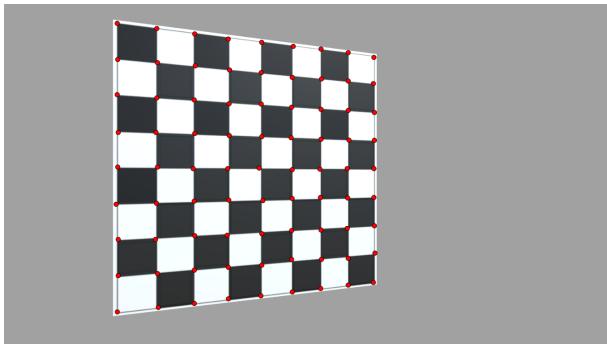


Abbildung 7.7: Die Kamera, welche das Schachbrett abbildet steht links versetzt zum Schachbrett und ist rotiert

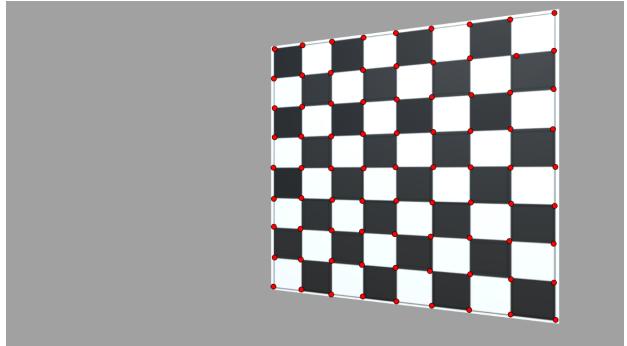


Abbildung 7.8: Die Kamera, welche das Schachbrett abbildet steht rechts versetzt zum Schachbrett und ist rotiert

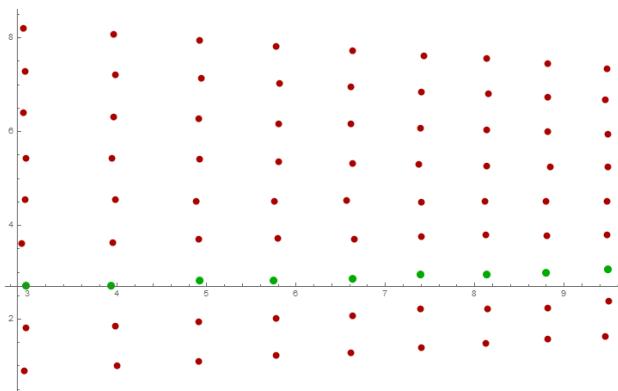


Abbildung 7.9: Die Abbildung zeigt das Ergebnis der Sortierung des linken Schachbretts. In grün ist die Reihe an Punkten zu sehen, welche den Schlüssel  $NeighbourI = 3$  haben.

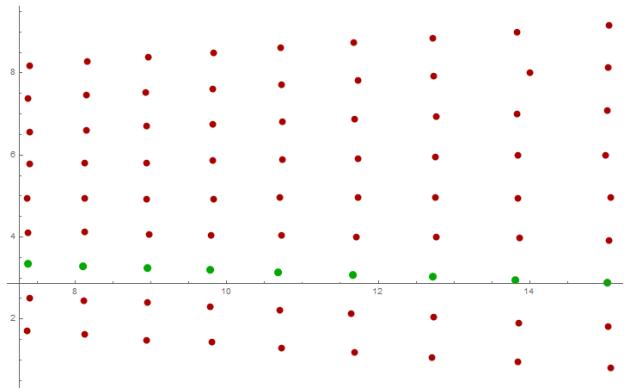


Abbildung 7.10: Die Abbildung zeigt das Ergebnis der Sortierung des rechten Schachbretts. In grün ist die Reihe an Punkten zu sehen, welche den Schlüssel  $NeighbourI = 3$  haben.

## 7.2 Resultate bei stark verzerrten Schachbrettern

Der entstandenen Sortierungsalgorithmus wird an stark perspektivisch verzerrten oder durch Bildfehler, wie Verzeichnungen, betroffenen Schachbrettern getestet. Die Möglichkeit herauszufinden, welche Eckpunkte in einem stark verzerrten Bild eines Schachbretts in eine Reihe oder Spalte gehören, kann bei der mathematischen Korrektur von Bilder Anwendung finden. Dies war ein weiterer Grund für die Implementierung dieses Algorithmus und soll in Folgearbeiten weiter studiert werden.

In den folgenden Beispielen ist jeweils das Originalbild des Schachbretts und daneben die Ausgabe des Algorithmus zu sehen. Der Plot zeigt in rot alle Punkte an, welche in *SortedPoints* aufgenommen wurden. Alle Punkte, welche als Schlüssel  $NeighbourI = 3$  haben, wurden im folgenden grün eingefärbt.

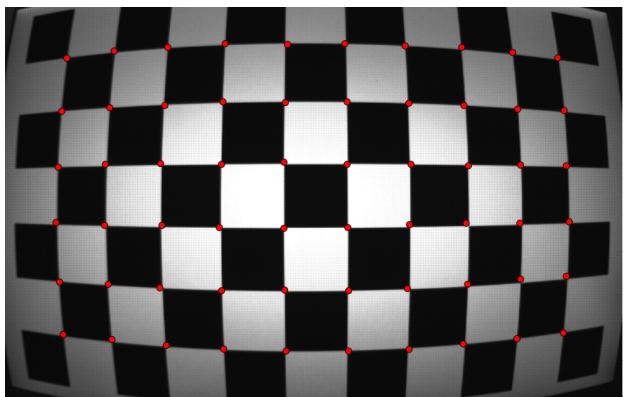


Abbildung 7.11: Schachbrett mit Tonnenverzeichnung

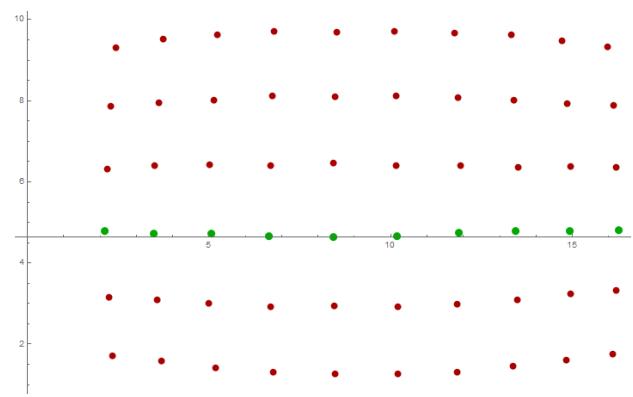


Abbildung 7.12: Ergebnis des Sortierungsalgoritmus.

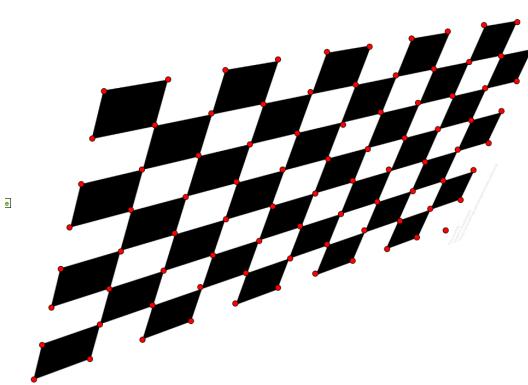


Abbildung 7.13: Perspektivisch stark verzerrtes Schachbrett

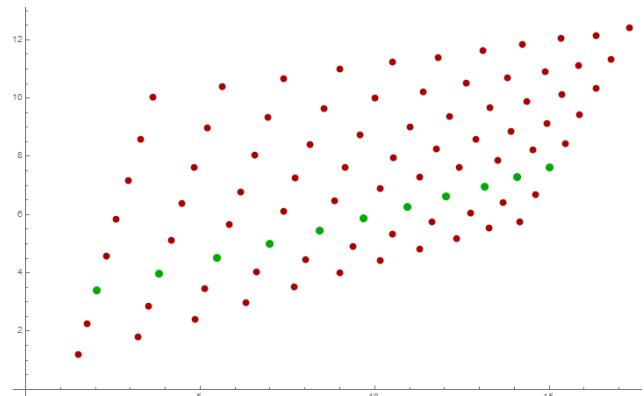


Abbildung 7.14: Ergebnis des Sortierungsalgoritmus.

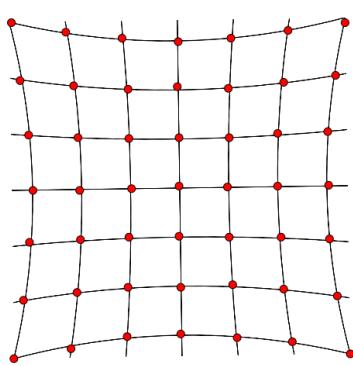


Abbildung 7.15: Schachbrett mit Kissenverzeichnung

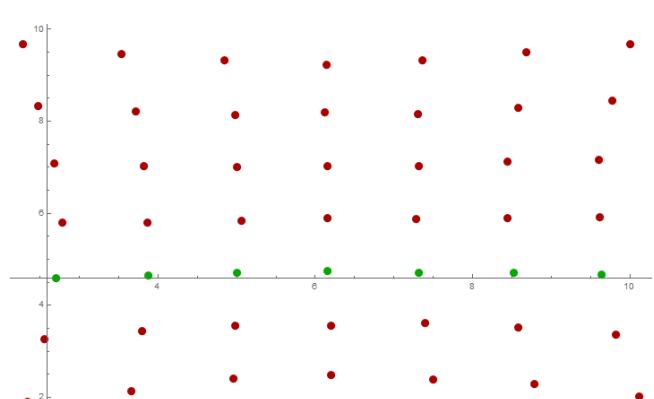


Abbildung 7.16: Ergebnis des Sortierungsalgoritmus.

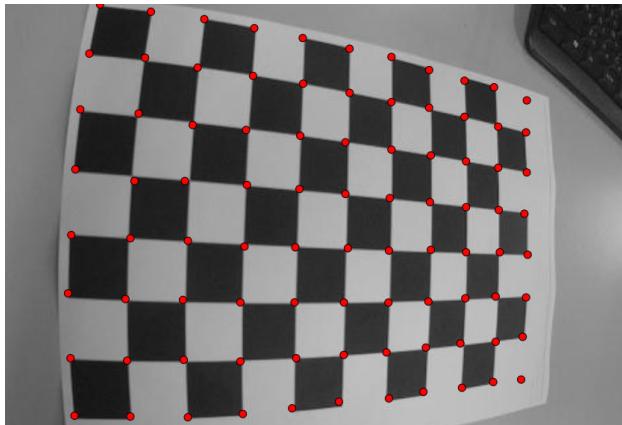


Abbildung 7.17: Perspektivisch verzerrten Schachbrett mit Tonnenvorzeichnung

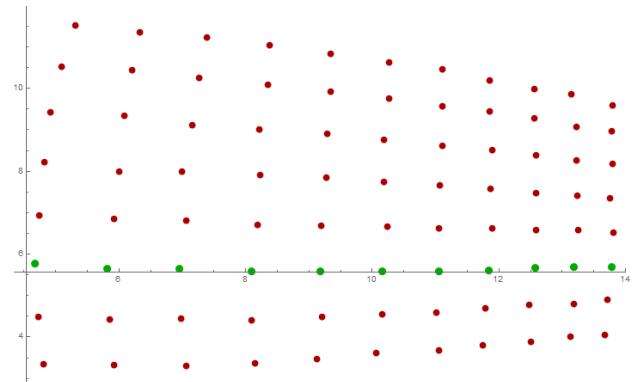


Abbildung 7.18: Ergebnis des Sortieralgorithmus.

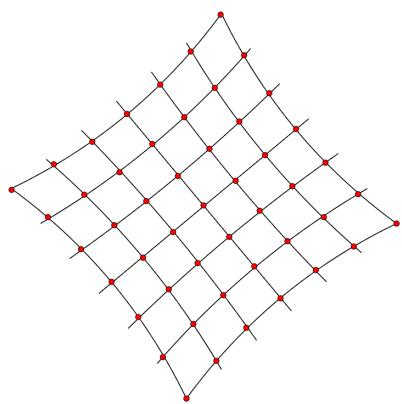


Abbildung 7.19: Bild eines Tonnenförmig verzeichnetem leicht perspektivisch verzerrtem Schachbretts

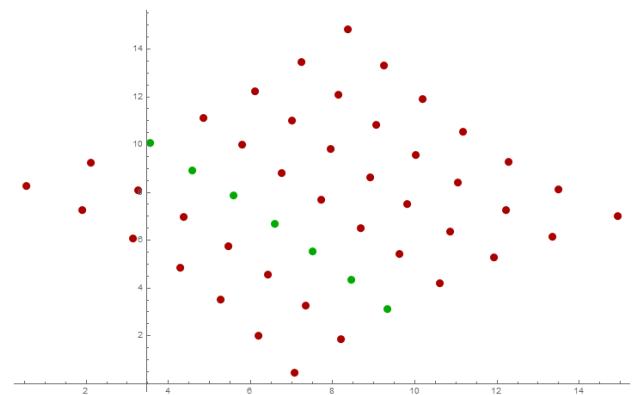


Abbildung 7.20: Algorithmisch detektierte Linie der dritten i-Reihe

# Zusammenfassung und Ausblick

In dieser Arbeit ist ein Szenenrekonstruktionsalgorithmus für stereoskopische Bildaufnahmen mit unterschiedlicher Auflösung entstanden. Dieser ist mit zuvor bekannten intrinsischen Kameraparametern im Stande die extrinsischen Kameraparameter zu bestimmen und die 3D-Szene zu rekonstruieren.

Für die Entwicklung des Algorithmus wurde ein synthetischer Szenenaufbau implementiert. Die grundlegenden Funktionen für die Bestimmung der extrinsischen Kameraparameter und der darauf folgenden Rekonstruktion der Szene, wurden anhand dieses Szenenaufbaus entwickelt und validiert. Der entstandene Algorithmus wurde dann auf realen Stereoaufnahmen angewandt. Die auf Grund ungenauer Punktekorrespondenzen entstandenen Fehler, konnten mit Hilfe des synthetischen Beispiels lokalisiert werden. Ein möglicher Lösungsansatz wurde mit Zuhilfenahme von Literaturquellen entwickelt und der Algorithmus entsprechend modifiziert. Der entwickelte modifizierte Algorithmus kann aus stereoskopischen Bildquellen unterschiedlicher Auflösung von vorcharakterisierten intrinsischen Kameraparametern eine Rekonstruktion der aufgenommenen Szene durchführen.

Eine zukünftig angedachte Modifikation dieses Szenenrekonstruktionsalgorithmus ist die Ableitung der Kameraparameter durch die Fundamentalmatrix. Durch die Bestimmung der Fundamentalmatrix sind die für die Triangulation nötigen Informationen bereits vorhanden. Die Ableitung der nötigen Kameraparameter aus der Fundamentalmatrix müsste genauer analysiert werden und der Rekonstruktionsalgorithmus dementsprechend angepasst werden. Mit dieser Modifikation würde es erlaubt sein, Kameras ohne eine Vorkalibrierung für die Szenenrekonstruktion zu verwenden.

Im zweiten Abschnitt dieser Masterarbeit wurde auf den in vielen Computer-Vision-Applikationen genutzten Ansatz für eine effiziente Szenenrekonstruktion, welcher keine zusätzliche Kamerakalibrierung beinhaltet, eingegangen. Der Ansatz basiert auf der vorherigen Rektifizierung von Stereoaufnahmen, was eine Vereinfachung der Korrespondenzanalyse mit sich bringt. Anhand dieser Korrespondenzen können Tiefenkarten erstellt werden, die eine direkte Abschätzung der Szenentiefe darstellen. Da für die Anwendung dieser Applikationen meist gleiche Kameraauflösungen vorausgesetzt werden, wurde in dieser Arbeit ein Rektifizierungsalgorithmus implementiert und auf Bildquellen verschiedener Auflösungen angewandt. Es wurde festgestellt, dass eine Ändern der Proportionen einzelner Pixel zwischen beiden Kameras im implementierte Algorithmus zu Streckung oder Stauchung der rekonstruierten Szene führt. Sind die Proportionen der Pixel dieselben und die Kameras haben nur unterschiedliche Auflösungen kann der implementierte Algorithmus angewandt werden und die Szene rekonstruiert werden.

Die meisten Kameras arbeiten mit quadratischen Pixel. Stereoskopische Aufnahmen mit unterschiedlichen Auflösungen solcher Kameras können mit dem implementierten Algorithmus rekonstruiert werden. Für Aufnahmen von rechteckigen Pixel mit unterschiedlichen Proportionen könnte eine Funktion entwickelt werden, welche die unterschiedlichen Pixelproportionen anhand spezieller Bildpunkte erkennt und die Bilder in ein Koordinatensystem mit gleichen Pixelproportionen transformiert. Auf die transformierten Bilder könnte der implementierte Algorithmus angewandt werden und somit aus stereoskopischen Bildern unterschiedlicher Auflösung mit unterschiedlichen Pixelproportionen eine Szene rekonstruiert werden.

# **Anhang**

## Programmcode synthetische Rekonstruktion

```
In[10]:= (*Define synthetic stereo setup and computation of Fundamentalmatrix*)
(*-----*)

(*Moduls for camera orientation-----*)
RotationsE1[alpha_] := Module[{Re1},
  [Modul
  Re1 = {{1, 0, 0}, {0, Cos[alpha Degree], -Sin[alpha Degree]}, {0, Sin[alpha Degree], Cos[alpha Degree]}},
    [Kosinus] [Grad] [Sinus] [Grad]
    [Sinus] [Grad] [Kosinus] [Grad]
  Return[Re1]
  [gib zurück
];
RotationsE2[alpha_] := Module[{Re2},
  [Modul
  Re2 = {{Cos[alpha Degree], 0, Sin[alpha Degree]}, {0, 1, 0}, {-Sin[alpha Degree], 0, Cos[alpha Degree]}},
    [Kosinus] [Grad] [Sinus] [Grad]
    [Sinus] [Grad] [Kosinus] [Grad]
  Return[Simplify[Re2]]
  [gib zur... vereinfache
];
RotationsE3[alpha_] := Module[{Re3},
  [Modul
  Re3 = {{Cos[alpha Degree], -Sin[alpha Degree], 0}, {Sin[alpha Degree], Cos[alpha Degree], 0}, {0, 0, 1}};
    [Kosinus] [Grad] [Sinus] [Grad]
    [Sinus] [Grad] [Kosinus] [Grad]
  Return[Re3]
  [gib zurück
];
(*Start of Computation-----*)
StartComputation[] := Module[{},
  [Modul
  ComputeTranslationAndRotation[Rot1, Rot2, Oc2, zeta2];
];
(*Place Cameras in 3D Scene-----*)
[leite ab
ComputeTranslationAndRotation[Rotation_, Rotations2_, Oc2_, zeta2_] :=
Module[{V, V2, M, PM1, PM2, R, R2, M2},
  [Modul
  PM2 = {{zeta2, 0, 0, 0}, {0, zeta2, 0, 0}, {0, 0, zeta2, 0}, {0, 0, 1, 0}};
  PM1 = {{zeta1, 0, 0, 0}, {0, zeta1, 0, 0}, {0, 0, zeta1, 0}, {0, 0, 1, 0}};
  Print["PM1 = ", PM1];
  [gib aus
  Print["PM2 = ", PM2];
  [gib aus
  PM2 = A.PM2;
```

```

Print["PM2 = ", PM2];
[gib aus

R = Transpose[Rotation];
[transponiere

R2 = Transpose[Rotations2];
[transponiere

Print["OC2 = ", Oc2];
[gib aus

V = {{1, 0, 0, -Oc[[1]]}, {0, 1, 0, -Oc[[2]]}, {0, 0, 1, -Oc[[3]]}};
M = R.V;
M = {{M[[1, 1]], M[[1, 2]], M[[1, 3]], M[[1, 4]]}, {M[[2, 1]], M[[2, 2]], M[[2, 3]], M[[2, 4]]}, {M[[3, 1]], M[[3, 2]], M[[3, 3]], M[[3, 4]]}, {0, 0, 0, 1}};

V2 = {{1, 0, 0, -Oc2[[1]]}, {0, 1, 0, -Oc2[[2]]}, {0, 0, 1, -Oc2[[3]]}};
M2 = R2.V2;
M2 = {{M2[[1, 1]], M2[[1, 2]], M2[[1, 3]], M2[[1, 4]]}, {M2[[2, 1]], M2[[2, 2]], M2[[2, 3]], M2[[2, 4]]}, {M2[[3, 1]], M2[[3, 2]], M2[[3, 3]], M2[[3, 4]]}, {0, 0, 0, 1}};

Print["M of C2=", MatrixForm[Simplify[M]]];
[gib aus [Matrizenform [vereinfache

Print["M2 of C1 =", MatrixForm[N[M2]]];
[gib aus [Matrizenform [numerischer Wert

ComputeProjectionMtx[a, b, c, d, aPrime,
  bPrime, cPrime, dPrime, d2Prime, Oc, Oc2, M, PM1, PM2, M2];
];

(*Compute Projectionmatrices_____*)
ComputeProjectionMtx[a_, b_, c_, d_, aPrime_, bPrime_,
  cPrime_, dPrime_, d2Prime_, Oc_, Oc2_, M_, PM1_, PM2_, M2_] :=
Module[{t, ProjectionMtxCamera1, ProjectionMtxCamera2},
[Modul

  ProjectionMtxCamera1 = PM1.M;
  ProjectionMtxCamera2 = PM2.M2;

  Print["ProjectionMtxCamera1", MatrixForm[ProjectionMtxCamera1]];
[gib aus [Matrizenform

  Print["ProjectionMtxCamera2", MatrixForm[N[ProjectionMtxCamera2]]];
[gib aus [Matrizenform [numerischer Wert

  ComputeProjectedPointsCamera1And2[ProjectionMtxCamera1, ProjectionMtxCamera2];
];

(*Project Points to Imageplanes in cameracoordinates_____*)
ComputeProjectedPointsCamera1And2[ProjectionMtxCamera1_, ProjectionMtxCamera2_] :=
Module[{CameraProjectedPointsK1, CameraProjectedPointsK2,
[Modul

  GraphicPointsC1, GraphicPointsC2, G1, G2},

```

```

CameraProjectedPointsK1 = Map[ProjectionMtxCamera1.# &,
  [wende an
 {a, b, c, d, aPrime, bPrime, cPrime, dPrime, d2Prime}];
CameraProjectedPointsK2 = Map[ProjectionMtxCamera2.# &,
  [wende an
 {a, b, c, d, aPrime, bPrime, cPrime, dPrime, d2Prime}];

Print["CameraProjectedPointsK1 = ", MatrixForm[CameraProjectedPointsK1]];
[gib aus [Matritzenform
Print["CameraProjectedPointsK2 = ", MatrixForm[N[CameraProjectedPointsK2]]];
[gib aus [Matritzenform [numerischer Wert

For[i = 1, i ≤ 9, i++,
[For-Schleife
 CameraProjectedPointsK1[[i]] =
 CameraProjectedPointsK1[[i]] / CameraProjectedPointsK1[[i, 4]];
 CameraProjectedPointsK2[[i]] = CameraProjectedPointsK2[[i]] /
 CameraProjectedPointsK2[[i, 4]];

];

Print["homogeneous CameraProjectedPointsK1 = ",
[gib aus
 MatrixForm[CameraProjectedPointsK1]];
[Matritzenform
Print["homogeneous CameraProjectedPointsK2 = ",
[gib aus
 MatrixForm[N[CameraProjectedPointsK2]]];
[Matritzenform [numerischer Wert

Print[
[gib aus
"Begin construct Epipol-----"];
[beginne Kontext
if[alpha == 45, ConstructEpipole[0c, 0c2, RotationsE2[alpha],
 CameraProjectedPointsK2[[1]], ProjectionMtxCamera2]];

Print["End construct Epipol-----"];
[gib aus [beende Kontext

GraphicPointsC1 = Map[{#[[1]], #[[2]]} &, CameraProjectedPointsK1];
[wende an
GraphicPointsC2 = Map[{#[[1]], #[[2]]} &, CameraProjectedPointsK2];
[wende an
G1 = Show[ListPlot[GraphicPointsC1[[1 ;; 9]], PlotStyle → Darker[Green]],
 [zeig... [listenbezogene Graphik [Darstellungsstil [dunkler [grün
ListLinePlot[{GraphicPointsC1[[4, All]], GraphicPointsC1[[1, All]],
 [listenbezogene Liniengraphik [alle [alle
GraphicPointsC1[[2, All]], GraphicPointsC1[[3, All]],
 [alle [alle
GraphicPointsC1[[4, All]], GraphicPointsC1[[8, All]],
 [alle [alle
GraphicPointsC1[[7, All]], GraphicPointsC1[[6, All]], GraphicPointsC1[[5
 [alle [alle

```

```

, All]], GraphicPointsC1[[8, All]], PlotStyle -> Darker[Green]],
[alle] [alle] [Darstellungsstil dunkler grün
ListLinePlot[{GraphicPointsC1[[1, All]], GraphicPointsC1[[5, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Green],
[Darstellungsstil dunkler grün
ListLinePlot[{GraphicPointsC1[[2, All]], GraphicPointsC1[[6, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Green],
[Darstellungsstil dunkler grün
ListLinePlot[{GraphicPointsC1[[3, All]], GraphicPointsC1[[7, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Green]]];
[Darstellungsstil dunkler grün
G2 = Show[ListPlot[GraphicPointsC2[[1 ;; 9]], PlotStyle -> Darker[Red]],
[zeige... listenbezogene Graphik] [Darstellungsstil dunkler rot
ListLinePlot[{GraphicPointsC2[[4, All]], GraphicPointsC2[[1, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
GraphicPointsC2[[2, All]], GraphicPointsC2[[3, All]],
[alle] [alle]
GraphicPointsC2[[4, All]], GraphicPointsC2[[8, All]],
[alle] [alle]
GraphicPointsC2[[7, All]], GraphicPointsC2[[6, All]], GraphicPointsC2[[5
[alle] [alle]
, All]], GraphicPointsC2[[8, All]], PlotStyle -> Darker[Red]],
[alle] [alle] [Darstellungsstil dunkler rot
ListLinePlot[{GraphicPointsC2[[1, All]], GraphicPointsC2[[5, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Red],
[Darstellungsstil dunkler rot
ListLinePlot[{GraphicPointsC2[[2, All]], GraphicPointsC2[[6, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Red],
[Darstellungsstil dunkler rot
ListLinePlot[{GraphicPointsC2[[3, All]], GraphicPointsC2[[7, All]]},
[listenbezogene Liniengraphik] [alle] [alle]
PlotStyle -> Darker[Red]]];
[Darstellungsstil dunkler rot
Print>Show[G1, G2, PlotRange -> All]];
[gib aus zeige an] [Koordinatenb... alle
ComputeProjectedPointsOnImagePlane1And2[
CameraProjectedPointsK1, CameraProjectedPointsK2, G1, G2];
];
(*Norm Points to Imageplanes-----*)
[Norm
ComputeProjectedPointsOnImagePlane1And2[CameraProjectedPointsK1_,
CameraProjectedPointsK2_, G1_, G2_] := Module[{ImagePlaneC1Points,
[Modul
ImagePlaneC2Points, SensorProjectionMtx1, SensorProjectionMtx2, PixelPitch},
(*In this case the Image plane Point is equal to the sensor points
[eingegebenes Objekt Bild] [Punkt
which is nessecary for the derivation of the fundamental matrix.

```

```

→ Take Pitchel pitch of 1 means that only the third
  [entferne
    element of the Vector is replaced by the forth*)
PixelPitch = 100;

ImagePlaneC1Points = Map[{#[[1]], #[[2]], #[[4]]} &, CameraProjectedPointsK1 ];
  [wende an
ImagePlaneC2Points = Map[{#[[1]], #[[2]], #[[4]]} &, CameraProjectedPointsK2 ];
  [wende an

Print["ImagePlaneC1Points = ", MatrixForm[Simplify[ImagePlaneC1Points]]];
  [gib aus          [Matrizenform] [vereinfache
Print["ImagePlaneC2Points = ", MatrixForm[N[ImagePlaneC2Points]]];
  [gib aus          [Matrizenform] [numerischer Wert

ComputeFundamentalMatrix[ImagePlaneC1Points, ImagePlaneC2Points, G1, G2];
];

(*Compute Fundamentalmatrix with 8-Point-Algorithm______)
  [Punkt
ComputeFundamentalMatrix[ImagePlaneC1Points_, ImagePlaneC2Points_, G1_, G2_] :=
Module[{CoefficientMtx, ns, F, lC1, lPrimeC1,
  [Modul
    lC2, lPrimeC2, e, K1, K2, EMtx, ePrime, EpipoleLines},
CoefficientMtx = ConstantArray[0, {9, 9}];
  [konstantes Array
For[i = 1, i ≤ 9, i++,
  [For-Schleife
    CoefficientMtx[[i]] =
      {ImagePlaneC2Points[[i, 1]] * ImagePlaneC1Points[[i, 1]],
       ImagePlaneC2Points[[i, 1]] * ImagePlaneC1Points[[i, 2]], ImagePlaneC2Points[[i, 1]],
       ImagePlaneC2Points[[i, 2]] * ImagePlaneC1Points[[i, 1]],
       ImagePlaneC2Points[[i, 2]] * ImagePlaneC1Points[[i, 2]], ImagePlaneC2Points[[i, 2]],
       ImagePlaneC1Points[[i, 1]], ImagePlaneC1Points[[i, 2]], 1};
  ];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMtx]]];
  [gib aus          [Matrizenform] [numerischer Wert
Print["MatrixRank[CoefficientMtx]", MatrixRank[CoefficientMtx]];
  [gib aus          [Rang der Matrix
(*Print[
  [gib aus
    "RowReduce CoefficientMtx = "[MatrixForm[RowReduce[N[CoefficientMtx]]]];*)
  [reduziere Zellen          [Matrizenform] [reduziere Ze] [numerischer Wert

Flatten[ns = NullSpace[N[CoefficientMtx]]];
  [ebne ein      [Nullraum] [numerischer Wert
Print["ns = ", ns];
  [gib aus
F = {{ns[[1, 1]], ns[[1, 2]], ns[[1, 3]]},
     {ns[[1, 4]], ns[[1, 5]], ns[[1, 6]]}, {ns[[1, 7]], ns[[1, 8]], ns[[1, 9]]}};
Print["F = ", MatrixForm[N[F]]];
  [gib aus          [Matrizenform] [numerischer Wert

```

```

Lyn aus      Liniarization mit numerischer Wert

lC1 = ConstantArray[0, {9, 3}];
  ↘konstantes Array
lPrimeC1 = ConstantArray[0, {9, 3}];
  ↘konstantes Array

For[i = 1, i ≤ 9, i++,
  ↗For-Schleife
    lC1[[i]] = N[F.ImagePlaneC1Points[[i, All]]];
      ↗numerischer Wert   ↗alle
    lPrimeC1[[i]] = N[Transpose[F].ImagePlaneC2Points[[i, All]]];
      ↗... ↗transponiere   ↗alle
  ];
Print["lC1 = ", lC1];
  ↗gib aus
Print["lPrimeC1 = ", N[lPrimeC1]];
  ↗gib aus   ↗numerischer Wert
e = Flatten[NullSpace[F]];
  ↗ebne ein   ↗Nullraum
ePrime = Flatten[NullSpace[Transpose[F]]];
  ↗ebne ein   ↗Nullraum   ↗transponiere

Print["e = ", N[e]];
  ↗gib aus   ↗numerischer Wert
Print["e' = ", N[ePrime]];
  ↗gib aus   ↗numerischer Wert

EpipoleLines = Map[Cross[#, ePrime] &, ImagePlaneC2Points];
  ↗w... ↗Kreuzprodukt
EpipoleLines = Map[# / #[[3]] &, EpipoleLines];
  ↗wende an

Print["EpipoleLines = ", MatrixForm[N[EpipoleLines]]];
  ↗gib aus   ↗Matrizenform   ↗numerischer Wert

Print>Show[G2, ContourPlot[lC1[[1]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗gib aus ↗zeige an   ↗Konturgraphik
  ContourPlot[lC1[[2]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[3]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[4]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[5]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[6]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[7]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[8]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik
  ContourPlot[lC1[[9]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}], G1,
  ↗Konturgraphik
  ContourPlot[lPrimeC1[[1]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
  ↗Konturgraphik

```

```

L<Konturgraphik
ContourPlot[lPrimeC1[[2]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[3]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[4]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[5]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[6]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[7]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[8]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
ContourPlot[lPrimeC1[[9]].{x, y, 1} == 0, {x, -10, 10}, {y, -5, 5}],
|Konturgraphik
PlotRange -> All]];
|Koordinatenbe...|alle

ComputeEssentialMtxFromFormular[F, ImagePlaneC1Points, ImagePlaneC2Points];
Rectification[F, e, ePrime, ImagePlaneC1Points, ImagePlaneC2Points];
NewRectification[F, e, ePrime, ImagePlaneC1Points, ImagePlaneC2Points];

(*ComputeEssentialMtx[ImagePlaneC1Points,ImagePlaneC2Points];*)

];

```

```

(*Computation essential Matrix (2 different ways) and extrinsic Cameraparameters*)
(*-----*)

(*1 Method:
 [Methode

Compute essential matrix with 8-Point-Algorithm and normalized Image coordinates*)
[ Punkt [Bild

ComputeEssentialMtx[PC1_, PC2_] :=
Module[{EMtx, normC1, normC2, K1, K2, n, CoefficientMtx, SVDE},
[Modul

Print["Begin Computing essential
[gib aus [beginne Kontext
Matrix_-----"];
Print["PC1 = ", PC1];
[gib aus
K1 = {{zeta1, 0, 0}, {0, zeta1, 0}, {0, 0, 1}};
K2 = AK.{{zeta2, 0, 0}, {0, zeta2, 0}, {0, 0, 1}};

normC1 = Map[Inverse[K1] .# &, PC1];
[w... [inverse Matrix
normC2 = Map[Inverse[K2] .# &, PC2];
[w... [inverse Matrix

Print["normalized Coordinates K1 = ", MatrixForm[normC1]];
[gib aus [Matrizenform
Print["normalized Coordinates K2 = ", MatrixForm[N[normC2]]];
[gib aus [Matrizenform [numerischer Wert

CoefficientMtx = ConstantArray[0, {9, 9}];
[konstantes Array
For[i = 1, i ≤ 9, i++,
[For-Schleife
CoefficientMtx[[i]] =
{normC2[[i, 1]] * normC1[[i, 1]], normC2[[i, 1]] * normC1[[i, 2]], normC2[[i, 1]],
normC2[[i, 2]] * normC1[[i, 1]], normC2[[i, 2]] * normC1[[i, 2]],
normC2[[i, 2]], normC1[[i, 1]], normC1[[i, 2]], 1};
];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMtx]]];
[gib aus [Matrizenform [numerischer Wert
n = NullSpace[N[CoefficientMtx]];
[Nullraum [numerischer Wert
Print["ns = ", n];
[gib aus
EMtx = {{n[[1, 1]], n[[1, 2]], n[[1, 3]]},
{n[[1, 4]], n[[1, 5]], n[[1, 6]]}, {n[[1, 7]], n[[1, 8]], n[[1, 9]]}};
Print["EMtx = ", MatrixForm[Simplify[EMtx]]];
[gib aus [Matrizenform [vereinfache

Print["SVD E = ", SingularValueDecomposition[EMtx]];
[gib aus [Expo... [Singulärwertzerlegung
EMtx = EMtx / EMtx[[2, 3]];
Print["EMtx = ", MatrixForm[Simplify[EMtx]]];
[gib aus [Matrizenform [vereinfache

```

```

Lyn aus          Liniarzentrum Everimache

Print["EigenValue for testing", Eigenvalues[EMtx.Transpose[EMtx]]];
|gib aus      |Eigenwerte |transponiere

ComputingCamerasFromE[EMtx, PC1, PC2];
];

(*2.Method: Compute essential matrix from Fundamentalmatrix____*)
|Metode

ComputeEssentialMtxFromFormular[F_, PC1_, PC2_] := Module[{K1, K2, EMtx},
|Modul

Print["Begin Computing essential
|gib aus |beginne Kontext
Matrix_____"];
(*Compute EssentialMatrix with EMtx = Transpose[K2].F.K1;*)
|transponiere

K1 = {{zeta1, 0, 0}, {0, zeta1, 0}, {0, 0, 1}};
K2 = AK.{{zeta2, 0, 0}, {0, zeta2, 0}, {0, 0, 1}};

EMtx = Transpose[K2].F.K1;
|transponiere

Print["EMtx = ", MatrixForm[N[EMtx]]];
|gib aus |Matrizenform |numerischer Wert
ComputingCamerasFromE[EMtx, PC1, PC2, F, K1, K2];
];

(*Compute extrinsic Cameraparameters____*)
ComputingCamerasFromE[EMtx_, PC1_, PC2_, F_, K1_, K2_] :=
Module[{W, Z, U, V, Sigma, S1, S2, R1, R2, i, t, P21, P22, P23, P24},
|Modul

{U, Sigma, V} = SingularValueDecomposition[EMtx];
|Singulärwertzerlegung

Print["U of E = ", N[U]];
|gib aus |Expo... |numerischer Wert
Print["Sigma of E = ", N[Sigma]];
|gib aus |Expo... |numerischer Wert
Print["V of E = ", N[V]];
|gib aus |Expo... |numerischer Wert

If[Det[U] == -1,
|... |Determinante
U = U * -1;
];

If[Det[V] == -1,
|... |Determinante
V = V * -1;
];

```

```

W = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
Z = {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}};

S1 = -U.Z.Transpose[U];
    \transponiere
S2 = U.Z.Transpose[U];
    \transponiere
R1 = U.Transpose[W].Transpose[V];
    \transponiere \transponiere
R2 = U.W.Transpose[V];
    \transponiere
Print["S1 = ", MatrixForm[N[S1]]];
\gib aus \Matrizenform \numerischer Wert
Print["S2 = ", MatrixForm[N[S2]]];
\gib aus \Matrizenform \numerischer Wert
Print["R1 = ", MatrixForm[N[R1]]];
\gib aus \Matrizenform \numerischer Wert
Print["R2 = ", MatrixForm[N[R2]]];
\gib aus \Matrizenform \numerischer Wert
(*Proof if R1 and R2 are possible rotations*)
i = IdentityMatrix[3];
    \Einheitsmatrix
Print["Test R1 is rotation =", Transpose[N[R1]].N[R1]];
\gib aus \transponiere \numerischer Wert
Print["Test R2 is rotation =", Transpose[N[R2]].N[R2]];
\gib aus \transponiere \numerischer Wert
(*Map[If[SameQ[Transpose[#+],i],
    \w... \... \identi... \transponiere
        Print[#, " is Rotation"], Print[N[#]," is no Rotation"]]&,{R1,R2}];*)
\gib aus \numerischer Wert
(*Map[Print[Det[#]]&,{R1,R1}];*)
\w... \gib aus \Determinante
(*Compute t from S1&S2*)

Print["Check if t of S1, S2 is equal = ", Map[NullSpace[#] &, {S1, S2}]];
\gib aus \prüfe \w... \Nullraum
t = Flatten[NullSpace[S1]];
    \ebne ein \Nullraum
Print["t = ", N[t]];
    \numerischer Wert
(*t is missing a scale factor of lamda set lamda to -
1 and 1 and you get four different solutions*)

P21 = U.W.Transpose[V];
    \transponiere
P21 = {{P21[[1, 1]], P21[[1, 2]], P21[[1, 3]], -1*t[[1]]},
    {P21[[2, 1]], P21[[2, 2]], P21[[2, 3]], -1*t[[2]]},
    {P21[[3, 1]], P21[[3, 2]], P21[[3, 3]], -1*t[[3]]}};

Print["P21 = ", MatrixForm[N[P21]]];
\gib aus \Matrizenform \numerischer Wert

P22 = U.Transpose[W].Transpose[V];
    \transponiere \transponiere
P22 = {{P22[[1, 1]], P22[[1, 2]], P22[[1, 3]], -1*t[[1]]},

```

```

{P22[[2, 1]], P22[[2, 2]], P22[[2, 3]], -1*t[[2]]},
{P22[[3, 1]], P22[[3, 2]], P22[[3, 3]], -1*t[[3]]}};

Print["P22 = ", MatrixForm[N[P22]]];
[gib aus] [Matrizenform] [numerischer Wert]

P23 = U.W.Transpose[V];
[transponiere]

P23 = {{P23[[1, 1]], P23[[1, 2]], P23[[1, 3]], 1*t[[1]]},
{P23[[2, 1]], P23[[2, 2]], P23[[2, 3]], 1*t[[2]]},
{P23[[3, 1]], P23[[3, 2]], P23[[3, 3]], 1*t[[3]]}};

Print["P23 = ", MatrixForm[N[P23]]];
[gib aus] [Matrizenform] [numerischer Wert]

P24 = U.Transpose[W].Transpose[V];
[transponiere] [transponiere]

P24 = {{P24[[1, 1]], P24[[1, 2]], P24[[1, 3]], 1*t[[1]]},
{P24[[2, 1]], P24[[2, 2]], P24[[2, 3]], 1*t[[2]]},
{P24[[3, 1]], P24[[3, 2]], P24[[3, 3]], 1*t[[3]]}};

Print["P24 = ", MatrixForm[N[P24]]];
[gib aus] [Matrizenform] [numerischer Wert]

Print["End Computing essential
[gib aus] [beende Kontext
Matrix_____"]];

PList = {};

AppendTo[PList, P21];
[hänge an bei]

AppendTo[PList, P22];
[hänge an bei]

AppendTo[PList, P23];
[hänge an bei]

AppendTo[PList, P24];
[hänge an bei]

Print["PList = ", PList];
[gib aus]

Print["Length PList = ", Length[PList]];
[gib aus] [Länge] [Länge]

Print[
[gib aus]

End Reconstruction of Rotation and
[beende Kontext
Translation_____]

"];

```

```

For [uu = 1, uu ≤ Length[PList], uu++,
  For-Schleife      | Länge
    RecMtx = PList[[uu]];

    RForOK2 = {{RecMtx[[1, 1]], RecMtx[[1, 2]], RecMtx[[1, 3]]},
               {RecMtx[[2, 1]], RecMtx[[2, 2]], RecMtx[[2, 3]]},
               {RecMtx[[3, 1]], RecMtx[[3, 2]], RecMtx[[3, 3]]}};
    RForOK2 = Transpose[RForOK2];
    | transponiere

    tForOK2 = {RecMtx[[1, 4]], RecMtx[[2, 4]], RecMtx[[3, 4]]};

    StructureComputation[F, PList[[uu]], PC1, PC2, K1, K2, RForOK2, tForOK2];

(*If[beta== 0,
  Wenn
  CreateTriangulation[PC1,PC2,PList[[uu]]];
  ];

  If[beta≠ 0,
  Wenn
  CreateTriangulation[PC1,PC2,PList[[uu]]]
  ];*)

];

Clear[PList];
| Lösche
];

```

```

(*geometric way for Triangulation-----*)
(*-----*)

CreateTriangulation[IPC1_, IPC2_, P22_] :=
Module[{LinesC1, LinesC2, PMW, O22D, O2D, V, ImagePlaneC2PointsWorld,
Modul
  ImagePlaneC1PointsWorld, solve, ReconstructedPointsC1, ReconstructedPointsC2,
  C20c2, W0c2, test, ResizedPointsC1, ResizedPointsC2, scaleValueC1,
  scaleValueC2, Ro2, GraphicPointsC1, GraphicPointsC2, G1, G2},
Print["Triangulation: WorldPoint reconstruction
[gib aus
-----"];

O2D = {0c[[1]], 0c[[2]], 0c[[3]]};
C20c2 = {P22[[1, 4]], P22[[2, 4]], P22[[3, 4]]};

Print["C20c2 = ", C20c2];
[gib aus

If[beta != 0,
[wenn
  Ro2 = {{P22[[1, 1]], P22[[1, 2]], P22[[1, 3]]}, {P22[[2, 1]],
    P22[[2, 2]], P22[[2, 3]]}, {P22[[3, 1]], P22[[3, 2]], P22[[3, 3]]}};
  Ro2 = Transpose[Ro2];
  [transponiere
  W0c2 = -Ro2.C20c2;
  Print["Ro2 = ", MatrixForm[Ro2]];
  [gib aus
  Print["W0c2 = ", W0c2];
  [gib aus
  PMW = {{Ro2[[1, 1]], Ro2[[1, 2]], Ro2[[1, 3]], W0c2[[1]]},
    {Ro2[[2, 1]], Ro2[[2, 2]], Ro2[[2, 3]], W0c2[[2]]},
    {Ro2[[3, 1]], Ro2[[3, 2]], Ro2[[3, 3]], W0c2[[3]]}, {0, 0, 0, 1}};
];
  If[beta == 0,
  [wenn
    W0c2 = -Rot1.C20c2;
    Print["-Rot1 = ", MatrixForm[-Rot1]];
    [gib aus
    Print["W0c2 = ", W0c2];
    [gib aus
    PMW = {{Rot1[[1, 1]], Rot1[[1, 2]], Rot1[[1, 3]], W0c2[[1]]},
      {Rot1[[2, 1]], Rot1[[2, 2]], Rot1[[2, 3]], W0c2[[2]]},
      {Rot1[[3, 1]], Rot1[[3, 2]], Rot1[[3, 3]], W0c2[[3]]}, {0, 0, 0, 1}};
];
  ImagePlaneC2PointsWorld = Map[{#[[1]], #[[2]], zeta2, #[[3]]} &, IPC2];
  [wende an
  ImagePlaneC1PointsWorld = Map[{#[[1]], #[[2]], zeta1, #[[3]]} &, IPC1];
  [wende an
  For[i = 1, i <= 9, i++,
  [For-Schleife
    ImagePlaneC2PointsWorld[[i, All]] = PMW.ImagePlaneC2PointsWorld[[i, All]];
    [alle

```

```

]; Länge

Print["ImagePlaneC1PointsWorld =", Länge
  | gib aus
  MatrixForm[Simplify[ImagePlaneC1PointsWorld]]]; Länge
  | Matrizenform | vereinfache

Print["ImagePlaneC2PointsWorld =", Länge
  | gib aus
  MatrixForm[Simplify[ImagePlaneC2PointsWorld]]]; Länge
  | Matrizenform | vereinfache

ImagePlaneC1PointsWorld =
Map[{#[[1]], #[[2]], #[[3]]} &, ImagePlaneC1PointsWorld];
| wende an

ImagePlaneC2PointsWorld = Map[{#[[1]], #[[2]], #[[3]]} &,
  | wende an
  ImagePlaneC2PointsWorld];

LinesC1 = Map[# + t (# - O2D) &, ImagePlaneC1PointsWorld];
| wende an

LinesC2 = Map[# + t2 (# - W0c2) &, ImagePlaneC2PointsWorld];
| wende an

Print["LinesC1 = ", LinesC1];
| gib aus

Print["LinesC2 = ", Simplify[LinesC2]];
| gib aus | vereinfache

solve = ConstantArray[0, {9, 1}];
| konstantes Array

For[i = 1, i ≤ 9, i++,
| For-Schleife
  solve[[i]] = Solve[{LinesC1[[i, 1]] == LinesC2[[i, 1]] &&
    | löse
    LinesC1[[i, 2]] == LinesC2[[i, 2]]}, {t, t2}]
];

Print["t & t2 = ", Flatten[Simplify[solve]]];
| gib aus | ebne ein | vereinfache

ReconstructedPointsC1 = ConstantArray[0, {8, 3}];
| konstantes Array

ReconstructedPointsC2 = ConstantArray[0, {8, 3}];
| konstantes Array

For[i = 1, i ≤ 8, i++,
| For-Schleife
  ReconstructedPointsC1[[i]] = N[LinesC1[[i]] /. t → {t} /. solve[[i]]];
| numerischer Wert
  ReconstructedPointsC2[[i]] = N[LinesC2[[i]] /. t2 → {t2} /. solve[[i]]];
| numerischer Wert
];

Print["ReconstructedPointsC1 = ", ReconstructedPointsC1];
| gib aus

```

```

 $\text{gib aus}$ 
Print["ReconstructedPointsC2 = ", ReconstructedPointsC2];
 $\text{gib aus}$ 

ResizedPointsC1 = ConstantArray[0, {8, 3}];
 $\text{konstantes Array}$ 
ResizedPointsC2 = ConstantArray[0, {8, 3}];
 $\text{konstantes Array}$ 

scaleValueC1 =
ObjectSize / (ReconstructedPointsC1[[2, 1, 1]] - ReconstructedPointsC1[[1, 1, 1]]);

Print["scaleValueC1 = ", scaleValueC1];
 $\text{gib aus}$ 
For[i = 1, i ≤ 8, i++,
 $\text{For-Schleife}$ 
ResizedPointsC1[[i]] = ReconstructedPointsC1[[i]] * scaleValueC1;
];

Print["ResizedPointsC1 = ", ResizedPointsC1];
 $\text{gib aus}$ 

scaleValueC2 =
ObjectSize / (ReconstructedPointsC2[[2, 1, 1]] - ReconstructedPointsC2[[1, 1, 1]]);

Print["scaleValueC2 = ", scaleValueC2];
 $\text{gib aus}$ 
For[i = 1, i ≤ 8, i++,
 $\text{For-Schleife}$ 
ResizedPointsC2[[i]] = ReconstructedPointsC2[[i]] * scaleValueC2;
];

Print["ResizedPointsC2 = ", ResizedPointsC2];
 $\text{gib aus}$ 

GraphicPointsC1 = Map[{#[[1, 1, 1]], #[[1, 2, 1]]} &, ResizedPointsC1];
 $\text{wende an}$ 
GraphicPointsC2 = Map[{#[[1, 1, 1]], #[[1, 2, 1]]} &, ResizedPointsC2];
 $\text{wende an}$ 

G1 = Show[ListPlot[GraphicPointsC1[[1 ;; 8]], PlotStyle → Darker[Green]],
 $\text{zeigt... listenbezogene Graphik}$   $\text{Darstellungsstil dunkler grün}$ 
ListLinePlot[{GraphicPointsC1[[4, All]], GraphicPointsC1[[1, All]],
 $\text{listenbezogene Liniengraphik}$   $\text{alle}$   $\text{alle}$ 
GraphicPointsC1[[2, All]], GraphicPointsC1[[3, All]],
 $\text{alle}$   $\text{alle}$ 
GraphicPointsC1[[4, All]], GraphicPointsC1[[8, All]],
 $\text{alle}$   $\text{alle}$ 
GraphicPointsC1[[7, All]], GraphicPointsC1[[6, All]], GraphicPointsC1[[5
 $\text{alle}$   $\text{alle}$ 
, All]], GraphicPointsC1[[8, All]], PlotStyle → Darker[Green]],
 $\text{alle}$   $\text{Darstellungsstil dunkler grün}$ 
ListLinePlot[{GraphicPointsC1[[1, All]], GraphicPointsC1[[5, All]]}],
 $\text{listenbezogene Liniengraphik}$   $\text{alle}$   $\text{alle}$ 

```

```

ListLinePlot[ {GraphicPointsC1[[2, All]], GraphicPointsC1[[6, All]]},
  |listenbezogene Liniengraphik |alle |alle
  PlotStyle -> Darker[Green],
  |Darstellungsstil |dunkler |grün
ListLinePlot[ {GraphicPointsC1[[3, All]], GraphicPointsC1[[7, All]]},
  |listenbezogene Liniengraphik |alle |alle
  PlotStyle -> Darker[Green],
  |Darstellungsstil |dunkler |grün
ListLinePlot[ {GraphicPointsC1[[1 ; 8]], PlotStyle -> Darker[Red]},
  |zeige... |listenbezogene Graphik |Darstellungsstil |dunkler |rot
G2 = Show[ ListPlot[GraphicPointsC2[[1 ; 8]], PlotStyle -> Darker[Red]],
  |alle |alle
ListLinePlot[ {GraphicPointsC2[[4, All]], GraphicPointsC2[[1, All]]},
  |listenbezogene Liniengraphik |alle |alle
  GraphicPointsC2[[2, All]], GraphicPointsC2[[3, All]],
  |alle |alle
  GraphicPointsC2[[4, All]], GraphicPointsC2[[8, All]],
  |alle |alle
  GraphicPointsC2[[7, All]], GraphicPointsC2[[6, All]], GraphicPointsC2[[5
  |alle |alle
  , All]], GraphicPointsC2[[8, All]], PlotStyle -> Darker[Red],
  |alle |alle |Darstellungsstil |dunkler |rot
ListLinePlot[ {GraphicPointsC2[[1, All]], GraphicPointsC2[[5, All]]},
  |listenbezogene Liniengraphik |alle |alle
  PlotStyle -> Darker[Red],
  |Darstellungsstil |dunkler |rot
ListLinePlot[ {GraphicPointsC2[[2, All]], GraphicPointsC2[[6, All]]},
  |listenbezogene Liniengraphik |alle |alle
  PlotStyle -> Darker[Red],
  |Darstellungsstil |dunkler |rot
ListLinePlot[ {GraphicPointsC2[[3, All]], GraphicPointsC2[[7, All]]},
  |listenbezogene Liniengraphik |alle |alle
  PlotStyle -> Darker[Red]];
  |Darstellungsstil |dunkler |rot
Print[Show[G1, G2, PlotRange -> All]];
  |gib aus |zeige an |Koordinatenb...|alle
];

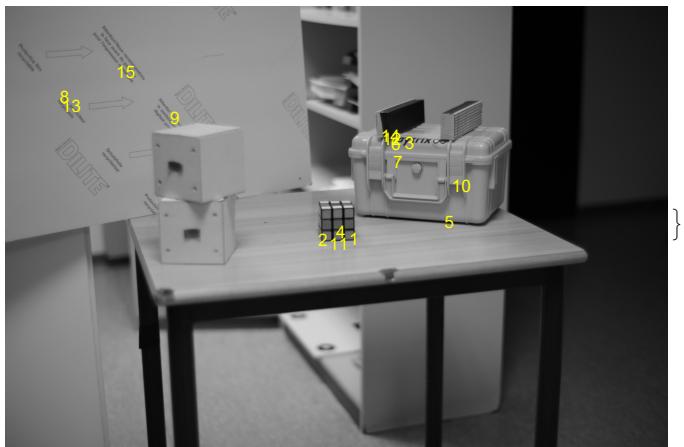
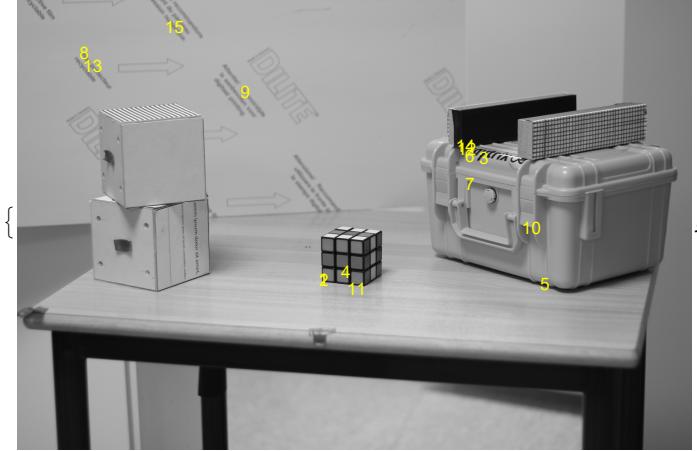
```

## Programmcode reelle Rekonstruktion

(\*Cooresponding points are found with Mathematica method \*)

```
imagesRGB = {};  
  
images = ColorConvert[imagesRGB, "Grayscale"];  
(matches = ImageCorrespondingPoints @@ images;  
MapThread[Show[#, Graphics[{Yellow, MapIndexed[Inset[#[[1]], #1] &, #[[2]]]}]] &,  
{images, matches}])  
  
Print["matches = ", matches];  
PointListK1 = matches[[2]]; (*[[2,2;;All]]*)  
PointListK2 = matches[[1]] (*[[1,2;;All]]*);  
  
Print["PointListK1 = ", PointListK1];  
Print["PointListK2 = ", PointListK2];  
  
PC1 = Map[{#[[1]], #[[2]], 1} &, PointListK1];  
PC2 = Map[{#[[1]], #[[2]], 1} &, PointListK2];  
A = {{1.2, 0, 0}, {0, 2.3, 0}, {0, 0, 1}};  
  
PC2 = Map[A.# &, PC2];  
  
SizeImage1 = ImageDimensions[];  
SizeImage2 = ImageDimensions[];  
  
Print["SizeImage1 = ", SizeImage1];  
Print["SizeImage1 = ", SizeImage1];  
piWidth = SizeImage1[[1]];  
piHeight = SizeImage1[[2]];  
pjWidth = SizeImage2[[1]];  
pjHeight = SizeImage2[[2]]
```

```
Print["PC1 = ", MatrixForm[PC1]];
 $\downarrow$  gib aus
 $\downarrow$  Matrizenform
Print["PC2 = ", MatrixForm[PC2]];
 $\downarrow$  Matrizenform
```



```

NormalizeCoordinates[PC1_, PC2_] :=
Module[{CentroidPC1, CentroidPC2 (*,projectedPointsK1,projectedPointsK2*)},  

  Modul

  Print["PC1 = ", MatrixForm[PC1]];
  |gib aus |Matrizenform
  Print["PC2 = ", MatrixForm[PC2]];
  |gib aus |Matrizenform

(*Normalize Image coordinates----- *)
  |normalisiere |Bild

CentroidPC1 = Mean[PC1];
  |arithmetisches Mittel
CentroidPC2 = Mean[PC2];
  |arithmetisches Mittel
Print["CentroidPC1 = ", N[CentroidPC1]];
  |gib aus |numerischer Wert
Print["CentroidPC2 = ", N[CentroidPC2]];
  |gib aus |numerischer Wert

(*move centroid and Points to origin*)
IntermediatePC1 = Map[ #- CentroidPC1[[1;;2]] &, PC1[[All, 1;;2]]];
  |wende an |alle
IntermediatePC2 = Map[ #- CentroidPC2[[1;;2]] &, PC2[[All, 1;;2]]];
  |wende an |alle
Print["Mean[IntermediatePC1] = ", N[Mean[IntermediatePC1]]];
  |gib aus |arithmetisches Mittel |...|arithmetisches Mittel
Print["Mean[IntermediatePC2] = ", N[Mean[IntermediatePC2]]];
  |gib aus |arithmetisches Mittel |...|arithmetisches Mittel

(*Get Men Distance*)
  |erhalte
DistanceVectorPC1 = Map[{0, 0} - # &, IntermediatePC1];
  |wende an
DistancesPC1 = Map[Sqrt[#[[1]]^2 + #[[2]]^2] &, DistanceVectorPC1];
  |w...|Quadratwurzel
MeanDistPC1 = Mean[DistancesPC1];
  |arithmetisches Mittel
Print["MeanDistPC1 = ", N[MeanDistPC1]];
  |gib aus |numerischer Wert
DistanceVectorPC2 = Map[{0, 0} - # &, IntermediatePC2];
  |wende an
DistancesPC2 = Map[Sqrt[#[[1]]^2 + #[[2]]^2] &, DistanceVectorPC2];
  |w...|Quadratwurzel
MeanDistPC2 = Mean[DistancesPC2];
  |arithmetisches Mittel
Print["MeanDistPC2 = ", N[MeanDistPC2]];
  |gib aus |numerischer Wert

(*Create Matrix out of Results
  to Move and Scale all Points and get RMS to Sqrt(2)*)
  |skaliere |Quadratwurzel
T = {{Sqrt[2] / MeanDistPC1, 0, (Sqrt[2] / MeanDistPC1) * -CentroidPC1[[1]]}, {0,
  |Quadratwurzel |Quadratwurzel

```

```

 $\text{Quadratwurzel}$   $\text{Quadratwurzel}$ 
 $\text{Sqrt}[2] / \text{MeanDistPC1}, (\text{Sqrt}[2] / \text{MeanDistPC1}) * -\text{CentroidPC1}[[2]], \{0, 0, 1\};$ 
 $\text{Quadratwurzel}$ 

Print[MatrixForm[N[T]]];
 $\text{gib aus }$   $\text{Matritzenform }$   $\text{numerischer Wert}$ 
PC1Norm = Map[T.# &, PC1];
 $\text{wende an}$ 
PC1Norm = Map[{#[[1]], #[[2]]} &, PC1Norm];
 $\text{wende an}$ 
Print["Matrix normalization PC1= ", N[PC1Norm]];
 $\text{gib aus }$   $\text{numerischer Wert}$ 
Print["RMS distance=", N[Mean[Map[Norm[#] &, PC1Norm]]]];
 $\text{gib aus }$   $\text{arit... }$   $\text{w... }$   $\text{Norm}$ 

TPrime = {{Sqrt[2] / MeanDistPC2, 0, (Sqrt[2] / MeanDistPC2) * -CentroidPC2[[1]]}, {0,
 $\text{Quadratwurzel}$   $\text{Quadratwurzel}$ 
 $\text{Sqrt}[2] / \text{MeanDistPC2}, (\text{Sqrt}[2] / \text{MeanDistPC2}) * -\text{CentroidPC2}[[2]], \{0, 0, 1\};$ 

Print[MatrixForm[N[TPrime]]];
 $\text{gib aus }$   $\text{Matritzenform }$   $\text{numerischer Wert}$ 

PC2Norm = Map[TPrime.# &, PC2];
 $\text{wende an}$ 
PC2Norm = Map[{#[[1]], #[[2]]} &, PC2Norm];
 $\text{wende an}$ 
Print["Matrix normalization PC2 = ", N[PC2Norm]];
 $\text{gib aus }$   $\text{numerischer Wert}$ 
Print["RMS distance=", N[Mean[Map[Norm[#] &, PC2Norm]]]];
 $\text{gib aus }$   $\text{arit... }$   $\text{w... }$   $\text{Norm}$ 

(*Compute Fundamentalmatrix with normalized pixel points PC1Norm and PC2Norm*)
FundamentalMtxBestimmung[PC1, PC2, PC1Norm, PC2Norm, T, TPrime(*, K1, K2*)];
];

(*Compute Fundamentalmatrix with normalized-8-Point_Algorithm_____*)

FundamentalMtxBestimmung[PC1_, PC2_, PC1Norm_, PC2Norm_, T_, TPrime_(*, K1_, K2_*)] :=
Module[{CoefficientMatx = {}}, UDVErgebnis, LängeV, FinF11, FinF12,
Modul
FinF13, FinF21, FinF22, FinF23, FinF31, FinF32, FinF33, FFin, SVDF},

Print[
 $\text{gib aus }$ 

Begin Computation of
 $\text{beginne Kontext}$ 
Fundamentalmatrix-----
];


```

```

For [qq = 1, qq ≤ Length[PC1Norm], qq++,
  For-Schleife      Länge
    AppendTo[CoefficientMatx, {PC2Norm[[qq, 1]] * PC1Norm[[qq, 1]],
      PC2Norm[[qq, 1]] * PC1Norm[[qq, 2]], PC2Norm[[qq, 1]],
      PC2Norm[[qq, 2]] * PC1Norm[[qq, 1]], PC2Norm[[qq, 2]] * PC1Norm[[qq, 2]],
      PC2Norm[[qq, 2]], PC1Norm[[qq, 1]], PC1Norm[[qq, 2]], 1}];
  ];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMatx]]];
  gib aus          Matrizenform numerischer Wert
UDVERgebnis = SingularValueDecomposition[N[CoefficientMatx]];
  Singulärwertzerlegung numerischer Wert
Print["Rank CoefficientMatx = ", MatrixRank[CoefficientMatx]];
  gib aus          Rang der Matrix

LängeV = Length[UDVERgebnis[[3]]];
  Länge
Print["LängeV ", LängeV];
  gib aus

FinF11 = UDVERgebnis[[3, 1, LängeV]];
FinF12 = UDVERgebnis[[3, 2, LängeV]];
FinF13 = UDVERgebnis[[3, 3, LängeV]];
FinF21 = UDVERgebnis[[3, 4, LängeV]];
FinF22 = UDVERgebnis[[3, 5, LängeV]];
FinF23 = UDVERgebnis[[3, 6, LängeV]];
FinF31 = UDVERgebnis[[3, 7, LängeV]];
FinF32 = UDVERgebnis[[3, 8, LängeV]];
FinF33 = UDVERgebnis[[3, 9, LängeV]];

FFin =
{{FinF11, FinF12, FinF13}, {FinF21, FinF22, FinF23}, {FinF31, FinF32, FinF33}};

Print["FFin = ", MatrixForm[N[Chop[FFin]]];
  gib aus          Matrizenform ersetze kleine Zahlen mit 0
Print["FFin Rank = ", MatrixRank[FFin]];
  Rang der Matrix

PC2NormTest = Map[{#[[1]], #[[2]], 1} &, PC2Norm];
  wende an
PC1NormTest = Map[{#[[1]], #[[2]], 1} &, PC1Norm];
  wende an
Print["PC2NormTest = ", N[PC2NormTest]];
  numerischer Wert
Print["PC1NormTest = ", N[PC1NormTest]];
  gib aus          numerischer Wert
l = {};
lPrime = {};

AppendTo[l, Map[FFin.# &, PC1NormTest]];
  hänge an bei  wende an
AppendTo[lPrime, Map[Transpose[FFin].# &, PC2NormTest]];
  hänge an bei  w... transponiere

```

```

Print[
  gib aus
  Show[ContourPlot[l.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 10}], PlotRange -> All]];
  Konturgraphik                                     Koordinatenbe... alle

Print[Show[
  gib ... zeige an
  ContourPlot[lPrime.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 5}], PlotRange -> All]];
  Konturgraphik                                     Koordinatenbe... alle

SVDF = SingularValueDecomposition[FFin];
  Singulärwertzerlegung

SVDF[[2]] = {SVDF[[2, 1]], SVDF[[2, 2]], {0, 0, 0}};

FPrime = SVDF[[1]].SVDF[[2]].Transpose[SVDF[[3]]];
  transponiere

Print["FPrime MatrixRank = ", MatrixRank[FPrime]];
  gib aus      Rang der Matrix      Rang der Matrix

Print["FPrime = ", MatrixForm[N[FPrime]]];
  gib aus      Matrizenform      numerischer Wert

Print[MatrixRank[FPrime]];
  gib aus Rang der Matrix

lRank2 = {};
lPrimeRank2 = {};

AppendTo[lRank2, Map[FPrime.# &, PC1NormTest]];
  hänge an bei      wende an

AppendTo[lPrimeRank2, Map[Transpose[FPrime].# &, PC2NormTest]];
  w... transponiere

Print[Show[
  gib aus zeige an
  ContourPlot[lRank2.{x, y, 1} == 0, {x, -50, 50}, {y, -10, 10}], PlotRange -> All]];
  Konturgraphik                                     Koordinatenbe... alle

Print[Show[ContourPlot[lPrimeRank2.{x, y, 1} == 0,
  gib aus zeig... Konturgraphik
  {x, -50, 20}, {y, -10, 4}], PlotRange -> All]];
  Koordinatenbe... alle

(*Denormalize F*)
F = Transpose[TPrime].FPrime.T;
  transponiere

Print["Demnornalized FPrime -> F =", MatrixForm[N[F]]];
  gib aus      Matrizenform      numerischer Wert

Print["F Rank = ", MatrixRank[F]];
  gib aus      Rang der Matrix

For[tt = 1, tt < Length[PC1], tt++,
  For-Schleife      Länge

Print[N[PC2[[tt]].F.PC1[[tt]]]];
  gib aus numerischer Wert

```

```

Lyn aus numerischer Wert
];

lRank2De = {};
lPrimeRank2De = {};

AppendTo[lRank2De, Map[F.# &, PC1]];
|hänge an bei |wende an
AppendTo[lPrimeRank2De, Map[Transpose[F].# &, PC2]];
|w...|transponiere

Print[Show[ContourPlot[lRank2De.{x, y, 1} == 0,
|gib ...|zeig...|Konturgraphik
{x, -4000, -2000}, {y, -1280, 1280}], PlotRange -> All]];
|Koordinatenbe...|alle

Print[Show[ContourPlot[lPrimeRank2.{x, y, 1} == 0, {x, -2000, 3000},
|zeig...|Konturgraphik
{y, -3000, 600}], PlotRange -> All]];
|Koordinatenbe...|alle

epipole = Flatten[NullSpace[F]];
|ebne ein |Nullraum
epipolePrime = Flatten[NullSpace[Transpose[F]]];
|ebne ein |Nullraum |transponiere
testEipolPrime = Transpose[F].epipolePrime;
|transponiere
testEipol = F.epipole;

Print["epipole =", Chop[epipole]];
|gib aus |ersetze kleine Zahlen mit 0
Print["epipolePrime =", Chop[epipolePrime]];
|gib aus |ersetze kleine Zahlen mit 0
Print["Test F^T.e' =", Chop[testEipolPrime]];
|gib aus |ersetze kleine Zahlen mit 0
Print["Test F.e = ", Chop[testEipol]];
|gib aus |ersetze kleine Zahlen mit 0

NewRectification[F, epipole, epipolePrime, images];
Print[
|gib aus

End Computation of
|beende Kontext
Fundamentalmatrix-----
"];

(*EssentialMtxAlgorithm[F,PC1Norm,
PC2Norm,T,TPrime,projectedPointsK1,projectedPointsK2];*)
EssentialMtx[F, PC1, PC2];

```

```

];

(*1. Method: Compute essential Matrix with 8-Pint-Algorithm*)
[Methode

EssentialMtxAlgorithm[F_, PC1Norm_, PC2Norm_,
T_, TPrime_, projectedPointsK1_, projectedPointsK2_] :=
Module[{normPC1, normPC2, K1, K2, UDVERgebnis, LängeV, FinE11, FinE12, FinE13,
[Modul
FinE21, FinE22, FinE23, FinE31, FinE32, FinE33, EFin, SVDE, CoefficientMtx = {}},

Print["
[gib aus

Begin Computation of Essential Matrix
[beginne Kontext
with 8-Point-Algorithm-----
[Punkt

"]];

(*Different Cameramatrices from different set ups*)
(*K1={{17.3158028,0,6.146589863},{0,17.31981867,4.600615527},{0,0,1}};;
K2={{18.60732121,0,4.145650968},{0,18.58796099,3.22706539},{0,0,1}};*)

K1 = {{18.530, 0, 6.094}, {0, 18.537, 3.994}, {0, 0, 1}};
K2 = K1;

(*px to mm for both imagePoints*)

(*PC1mm = Map[(#*6.5)/1000 &,PC1];
[wende an
PC2mm = Map[(#*4.29)/1000 &,PC2];*)
[wende an

PC1mm = Map[(# * 6.5) / 1000 &, PC1];
[wende an
PC2mm = Map[(# * 4.29) / 1000 &, PC2];
[wende an

(*PC1mm = Map[{#[[1]],#[[2]],1}&,PC1mm];
[wende an
PC2mm = Map[{#[[1]],#[[2]],1}&,PC2mm];*)
[wende an

Print["PC1 in mm = ", MatrixForm[N[PC1mm]]];
[gib aus [Matrizenform [numerischer Wert
Print["PC2 in mm = ", MatrixForm[N[PC2mm]]];
[gib aus [Matrizenform [numerischer Wert

normPC1 = Map[Inverse[K1].# &, PC1mm];
[... [inverse Matri

```

```

Lvv... Inverse Matrix
normPC2 = Map[Inverse[K2].# &, PC2mm];
  [w... inverse Matrix
normPC1 = Map[{#[[1]], #[[2]], 1} &, normPC1];
  [wende an
normPC2 = Map[{#[[1]], #[[2]], 1} &, normPC2];
  [wende an

Print["normalized Coordinates K1 = ", MatrixForm[N[normPC1]]];
[gib aus] [Matrizenform] [numerischer Wert
Print["normalized Coordinates K2 = ", MatrixForm[N[normPC2]]];
[gib aus] [Matrizenform] [numerischer Wert

For[qq = 1, qq ≤ Length[normPC1], qq++,
[For-Schleife] [Länge
  AppendTo[CoefficientMtx, {normPC2[[qq, 1]] * normPC1[[qq, 1]],
    [hänge an bei
      normPC2[[qq, 1]] * normPC1[[qq, 2]], normPC2[[qq, 1]],
      normPC2[[qq, 2]] * normPC1[[qq, 1]], normPC2[[qq, 2]] * normPC1[[qq, 2]],
      normPC2[[qq, 2]], normPC1[[qq, 1]], normPC1[[qq, 2]], 1}];
];
Print["CoefficientMtx = ", MatrixForm[N[CoefficientMtx]]];
[gib aus] [Matrizenform] [numerischer Wert
UDVErgebnis = SingularValueDecomposition[N[CoefficientMtx]];
  [Singulärwertzerlegung] [numerischer Wert
Print["Rank CoefficientMtx = ", MatrixRank[CoefficientMtx]];
[gib aus] [Rang der Matrix

LängeV = Length[UDVErgebnis[[3]]];
[Länge
Print[LängeV];
[gib aus]

FinE11 = UDVErgebnis[[3, 1, LängeV]];
FinE12 = UDVErgebnis[[3, 2, LängeV]];
FinE13 = UDVErgebnis[[3, 3, LängeV]];
FinE21 = UDVErgebnis[[3, 4, LängeV]];
FinE22 = UDVErgebnis[[3, 5, LängeV]];
FinE23 = UDVErgebnis[[3, 6, LängeV]];
FinE31 = UDVErgebnis[[3, 7, LängeV]];
FinE32 = UDVErgebnis[[3, 8, LängeV]];
FinE33 = UDVErgebnis[[3, 9, LängeV]];

EFin =
{{FinE11, FinE12, FinE13}, {FinE21, FinE22, FinE23}, {FinE31, FinE32, FinE33}};
Print["EFin = ", N[Chop[EFin]]];
[gib aus] [.. ersetze kleine Zahlen mit 0
Print["EFin = ", MatrixRank[EFin]];
[gib aus] [Rang der Matrix

SVDE = SingularValueDecomposition[EFin];
  [Singulärwertzerlegung
Print["SVDE = ", SVDE];
[gib aus]

SVD = SingularValueDecomposition[EFin];
  [Singulärwertzerlegung

```

```

 $\text{Singularwertdecomposition}$ 
Print["SVD of EMtx = ", SVD];
 $\text{gib aus}$ 
SVD[[2]] = DiagonalMatrix[
 $\text{Diagonalmatrix}$ 
{ (SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, (SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, 0 }];
Print["new SVD = ", SVD];
 $\text{gib aus}$ 

EPrime = SVD[[1]].SVD[[2]].Transpose[SVD[[3]]];
 $\text{transponiere}$ 
Print["EPrime = ", EPrime];
 $\text{gib aus}$ 
Print["EPrime = ", MatrixRank[EPrime]];
 $\text{Rang der Matrix}$ 

Print["
 $\text{gib aus}$ 

End Computation of Essential Matrix
 $\text{beende Kontext}$ 
with 8-Point-Algorithm_
 $\text{-----}$ 
 $\text{Punkt}$ 

"];
ExtractRotationAndTranslation[EFin, PC1, PC2];

];

(*2. Method: Compute essential matrix with Fundamentalmatrix*)
 $\text{Methode}$ 
EssentialMtx[F_, PC1_, PC2_] :=
Module[{K1, K2, EsMtx, SVD, normPC1, normPC2 (*,PC1mm,PC2mm*) },
 $\text{Modul}$ 

Print["
 $\text{gib aus}$ 

Begin Computation of Essential Matrix
 $\text{beginne Kontext}$ 
with K1 and K2_
 $\text{-----}$ 

];
(*Different Cameramatrices from different set ups*)
(*60 D and 6D First Try*)
 $\text{leite ab } \dots \text{ erstes Element}$ 
(*K1={{17.3158028,0,6.146589863},{0,17.31981867,4.600615527},{0,0,1}};*
K2={{18.60732121,0,4.145650968},{0,18.58796099,3.22706539},{0,0,1}};*)
(*K1={{2663.969662,0,945.6292097},{0,2664.587487,707.7870042},{0,0,1}};*
K2={{4337.370912,0,966.352207},{0,4332.858039,752.2296946},{0,0,1}};*)

```

```

(*same Cameras 6D*)
    \leite ab
(*K1={{18.530,0,6.094},{0,18.537,3.994},{0,0,1}};
K2=K1;
K1={{2850.913,0,937.6861},{0,2851.852,614.5411},{0,0,1}};
K2=K1;*)

(*6D and 6D with Mathematica
    \leite ab \leite ab
Correspondong detection different Cameras same resolution*)
(*K1={{18.17617,0,6.1583},{0,18.18014,4.568642},{0,0,1}};
K2={{19.146558,0,4.229572},{0,19.00778,3.125653},{0,0,1}};*)
K1 = {{4436.0715295011, 0, 985.9143}, {0, 4430.717, 728.5904}, {0, 0, 1}};
K2 = A.{{2796.335, 0, 947.4308}, {0, 2796.944, 702.8681}, {0, 0, 1}};

(*6D and 6D with Mathematica
    \leite ab \leite ab
Correspondong detection and different Camera resolutions*)
(*K1={{2825.059,0,927.4028},{0,2809.878,599.5711},{0,0,1}};
K2={{6423.957,0,1108.976},{0,6394.091,657.8209},{0,0,1}};*)

Print["PC1 = ", PC1];
\gib aus
PC1mm = PC1;

PC2mm = PC2;

Print["PC1mm = ", MatrixForm[PC1mm]];
\gib aus \Matrizenform
Print["PC2mm = ", MatrixForm[PC2mm]];
\gib aus \Matrizenform
normPC1 = Map[Inverse[K1] .# &, PC1mm];
\w... \inverse Matrix
normPC2 = Map[Inverse[K2] .# &, PC2mm];
\w... \inverse Matrix
normPC1 = Map[{#[[1]], #[[2]], 1} &, normPC1 ];
\wende an
normPC2 = Map[{#[[1]], #[[2]], 1} &, normPC2 ];
\wende an

Print["normalized Coordinates K1 = ", MatrixForm[N[normPC1]]];
\gib aus \Matrizenform \numerischer Wert
Print["normalized Coordinates K2 = ", MatrixForm[N[normPC2]]];
\gib aus \Matrizenform \numerischer Wert

EsMtx = Transpose[K2].F.K1;
\transponiere
Print["EsMtx = ", EsMtx];
\gib aus
Print[MatrixRank[EsMtx]];
\gib aus \Rang der Matrix

SVD = SingularValueDecomposition[EsMtx];
\Singulärwertzerlegung

```

```

Print["SVD of EMtx =", SVD];
|gib aus
SVD[[2]] = DiagonalMatrix[
  |Diagonalmatrix
  {(SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, (SVD[[2, 1, 1]] + SVD[[2, 2, 2]]) / 2, 0}];

Print["new SVD = ", SVD];
|gib aus

EPrime = SVD[[1]].SVD[[2]].Transpose[SVD[[3]]];
|transponiere

Print["EPrime = ", EPrime];
|gib aus
Print[MatrixRank[EPrime]];
|gib aus |Rang der Matrix

For[zz = 1, zz < Length[normPC1], zz++,
|For-Schleife |Länge
  Print[normPC2[[zz]].EPrime.normPC1[[zz]]];
  |gib aus
];

Print["
|gib aus

End Computation of Essential Matrix
|beende Kontext
  with K1 and K2-----
-----];

Print["
|gib aus

Begin Computation of extern Cameraparameters
|beginne Kontext
  with K1 and K2-----
-----];

(*Compute extrinsic Cameraparameters-----*)

ExtractRotationAndTranslation[EPrime, F, PC1mm, PC2mm, K1, K2];
];

(*Compute extrinsic Cameraparameters-----*)

ExtractRotationAndTranslation[EPrime_, F_, PC1mm_, PC2mm_, K1_, K2_] :=
Module[{W, Z, U, V, Sigma, S1, S2, R1, R2, i, t, P21, P22, P23, P24, NewE},
|Modul

```

```

Lviuuui
Print["
|gib aus

Begin Reconstruction of Rotation and
|beginne Kontext
    Translation-----

"];

Print["E = ", MatrixForm[EPrime]];
|gib aus |Expo... |Matritzenform

{U, Sigma, V} = SingularValueDecomposition[EPrime];
|Singulärwertzerlegung

Print["U of E = ", U];
|gib aus |Exponentielle E

Print["Sigma of E = ", Sigma];
|gib aus |Exponentielle E

Print["V of E = ", V];
|gib aus |Exponentielle E

Sigma = {{1, 0, 0}, {0, 1, 0}, {0, 0, 0}};

ETest1 = U.Sigma.Transpose[V];
|transponiere

Print["ETest1 = ", ETest1];
|gib aus

{U, Sigma, V} = SingularValueDecomposition[ETest1];
|Singulärwertzerlegung

Print["U of E = ", U];
|gib aus |Exponentielle E

Print["Sigma of E = ", Sigma];
|gib aus |Exponentielle E

Print["V of E = ", V];
|gib aus |Exponentielle E

If[Det[U] == -1,
|... |Determinante
    U = U * -1;
];

If[Det[V] == -1,
|... |Determinante
    V = V * -1;
];

ETest2 = U.Sigma.Transpose[V];
|transponiere

Print["ETest2 = ", ETest2];
|gib aus

```

```

Lyn aus

W = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
Z = {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}};

S1 = -U.Z.Transpose[U];
    |transponiere
S2 = U.Z.Transpose[U];
    |transponiere
R2 = U.Transpose[W].Transpose[V];
    |transponiere |transponiere
R1 = U.W.Transpose[V];
    |transponiere
Print["S1 = ", MatrixForm[S1]];
|gib aus |Matrizenform
Print["S2 = ", MatrixForm[S2]];
|gib aus |Matrizenform
Print["R1 = ", MatrixForm[R1]];
|gib aus |Matrizenform
Print["R2 = ", MatrixForm[R2]];
|gib aus |Matrizenform

Print[Det[U]];
|gib aus |Determinante
Print[Det[V]];
|gib aus |Determinante
Print["Det E =", Det[EPrime]];
|gib aus |De... |Exp... |Determinante
Print["Det F =", Det[F]];
|gib aus |Determina... |Determinante

Print["Check if t of S1, S2 is equal = ", Map[NullSpace[#] &, {S1, S2}]];
|gib aus |prüfe |w... |Nullraum
Print["R1 is Rotation = ", Chop[Transpose[R1].R1]];
|gib aus |erse... |transponiere
Print["R2 is Rotation =", Chop[Transpose[R2].R2]];
|gib aus |erse... |transponiere

Print["Determinant of R1 = ", Det[R1]];
|gib aus |Determinante
Print["Determinant of R2 = ", Det[R2]];
|gib aus |Determinante

LeftEprime = NullSpace[Transpose[EPrime]];
    |Nullraum |transponiere
t = Flatten[NullSpace[S1]];
    |ebne ein |Nullraum

(*t={U[[3,1]],U[[3,2]],U[[3,3]]};*)

Print["t =", t];
|gib aus
LeftS2 = NullSpace[Transpose[S2]];
    |Nullraum |transponiere

```

```

Print["Left E und S = ", LeftEprime, " , ", LeftS2];
|gib aus |links |Exponentielle Konstante E

(*P1=R1;
P2=R1;
P3=R2;
P4=R2;*)

P1 = U.W.Transpose[V];
|transponiere
P2 = U.W.Transpose[V];
|transponiere
P3 = U.Transpose[W].Transpose[V];
|transponiere |transponiere
P4 = U.Transpose[W].Transpose[V];
|transponiere |transponiere

P1 = {{P1[[1, 1]], P1[[1, 2]], P1[[1, 3]], 1*t[[1]]}, {P1[[2, 1]], P1[[2, 2]], P1[[2, 3]], 1*t[[2]]}, {P1[[3, 1]], P1[[3, 2]], P1[[3, 3]], 1*t[[3]]}};
P2 = {{P2[[1, 1]], P2[[1, 2]], P2[[1, 3]], -1*t[[1]]}, {P2[[2, 1]], P2[[2, 2]], P2[[2, 3]], -1*t[[2]]}, {P2[[3, 1]], P2[[3, 2]], P2[[3, 3]], -1*t[[3]]}};
P3 = {{P3[[1, 1]], P3[[1, 2]], P3[[1, 3]], 1*t[[1]]}, {P3[[2, 1]], P3[[2, 2]], P3[[2, 3]], 1*t[[2]]}, {P3[[3, 1]], P3[[3, 2]], P3[[3, 3]], 1*t[[3]]}};
P4 = {{P4[[1, 1]], P4[[1, 2]], P4[[1, 3]], -1*t[[1]]}, {P4[[2, 1]], P4[[2, 2]], P4[[2, 3]], -1*t[[2]]}, {P4[[3, 1]], P4[[3, 2]], P4[[3, 3]], -1*t[[3]]}};

Print["T1 = ", MatrixForm[P1]];
|gib aus |Matrizenform
Print["T2 = ", MatrixForm[P2]];
|gib aus |Matrizenform
Print["T3 = ", MatrixForm[P3]];
|gib aus |Matrizenform
Print["T4 = ", MatrixForm[P4]];
|gib aus |Matrizenform

(*R2=R2*-1;
Test = {{0,-t[[3]],t[[2]]},{t[[3]],0,-t[[1]]},{-t[[2]],t[[1]],0}}.R2;
Test = S2.R2;
Print[MatrixForm[Test]];*)
|gib aus |Matrizenform

PList = {};

AppendTo[PList, P1];
|hängt an bei
AppendTo[PList, P2];
|hängt an bei
AppendTo[PList, P3];
|hängt an bei
AppendTo[PList, P4];
|hängt an bei

Print["PList = ", PList];
|nicht ausgeführt

```

```

Lyn aus
Print["Length PList = ", Length[PList]];
|gib aus |Länge
|Länge

Print[
|gib aus

End Reconstruction of Rotation and
|beende Kontext
Translation-----
];

For[uu = 1, uu <= Length[PList], uu++,
|For-Schleife |Länge
RecMtx = PList[[uu]];

RForOK2 = {{RecMtx[[1, 1]], RecMtx[[1, 2]], RecMtx[[1, 3]]},
{RecMtx[[2, 1]], RecMtx[[2, 2]], RecMtx[[2, 3]]},
{RecMtx[[3, 1]], RecMtx[[3, 2]], RecMtx[[3, 3]]}};
RForOK2 = Transpose[RForOK2];
|transponiere

tForOK2 = {RecMtx[[1, 4]], RecMtx[[2, 4]], RecMtx[[3, 4]]};

StructureComputation[F, PList[[uu]], PC1, PC2, K1, K2, RForOK2, tForOK2];
];

];

```

```

(*Reconstruct Scene with Sampson
Approximation-----*)
StructureComputation[F_, P3_, PC1_, PC2_, K1_, K2_, RForOK2_, tForOK2_] :=
Module[{T, TPrime, StructureF, R, RPrime, RotatedF, f, fPrime, a, b,
Modul
c, d, rootMin, l, lPrime, K2Ex, K1Ex, P, PPrime, CoefficientMtx = {}, 
ee, ii, SpacePoints = {}, SpacePointsMMNotScaled = {}},

For[vv = 1, vv < Length[PC1mm], vv++,
For-Schleife      |Länge
K2Ex = P3;

(*Translate Points to origin*)
|verschiebe
T = {{1, 0, -PC1[[vv, 1]]}, {0, 1, -PC1[[vv, 2]]}, {0, 0, 1}};
TPrime = {{1, 0, -PC2[[vv, 1]]}, {0, 1, -PC2[[vv, 2]]}, {0, 0, 1}};

StructureF = Transpose[Inverse[TPrime]].F.Inverse[T];
|transponiere |inverse Matrix |inverse Matrix

(*Normalize epipoles*)
|normalisiere
epipole = Flatten[NullSpace[StructureF]];
|ebne ein |Nullraum
epipolePrime = Flatten[NullSpace[Transpose[StructureF]]];
|ebne ein |Nullraum |transponiere

Teste = epipole[[1]]^2 + epipole[[2]]^2;
TestePrime = epipolePrime[[1]]^2 + epipolePrime[[2]]^2;

ScaleE = 1 / Teste;
ScaleEPrime = 1 / TestePrime;

epipole =
{epipole[[1]]^2 * ScaleE, epipole[[2]]^2 * ScaleE, epipole[[3]]^2 * ScaleE};
epipole = {Sqrt[epipole[[1]]], Sqrt[epipole[[2]]], Sqrt[epipole[[3]]]};
|Quadratwurzel |Quadratwurzel |Quadratwurzel

epipolePrime = {epipolePrime[[1]]^2 * ScaleEPrime,
epipolePrime[[2]]^2 * ScaleEPrime, epipolePrime[[3]]^2 * ScaleEPrime};
epipolePrime = {Sqrt[epipolePrime[[1]]], Sqrt[epipolePrime[[2]]],
|Quadratwurzel |Quadratwurzel
Sqrt[epipolePrime[[3]]]};

Teste = epipole[[1]]^2 + epipole[[2]]^2;
TestePrime = epipolePrime[[1]]^2 + epipolePrime[[2]]^2;

(*Rotate Epipoles to (1,0,0^T) *)
|drehe
R =
{{epipole[[1]], epipole[[2]], 0}, {-epipole[[2]], epipole[[1]], 0}, {0, 0, 1}};
RPrime = {{epipolePrime[[1]], epipolePrime[[2]], 0},
{-epipolePrime[[2]], epipolePrime[[1]], 0}, {0, 0, 1}};

RTest = Transpose[R].R;
|transponiere

```

```

LUDWIGSPUNIERE
RPrimeTest = Transpose[RPrime].RPrime;
    |transponiere
Rep = R.epipole;
RePrime = RPrime.epipolePrime;

(*Replace F with RPrime.StructureF.R^T*)
    |ersetze

RotatedF = RPrime.StructureF.Transpose[R];
    |transponiere

(*Set variables f, fPrime, a, b, c, d for further computations*)
    |weise zu

f = epipole[[3]];
fPrime = epipolePrime[[3]];
a = RotatedF[[2, 2]];
b = RotatedF[[2, 3]];
c = RotatedF[[3, 2]];
d = RotatedF[[3, 3]];

(*Form the polynomial g(t)*)

roots = Solve[root ((a*root + b)^2 + fPrime^2 (c*root + d)^2)^2 -
    |löse
    (a*d - b*c) * (1 + f^2 * root^2)^2 * (a*root + b) * (c*root + d) == 0, root];

rootsRealis =
Solve[root ((a*root + b)^2 + fPrime^2 (c*root + d)^2)^2 - (a*d - b*c) * -
    |löse
    (1 + f^2 * root^2)^2 * (a*root + b) * (c*root + d) == 0, root, Reals];
    |Menge reeller Zahlen

tInfinity = 1/f^2 + c^2/(a^2 + fPrime^2 * c^2);

(*evaluate min cost function of real root values in s(t)*)
st = {};

For[oo = 1, oo <= Length[rootsRealis], oo++,
    |For-Schleife      |Länge
    AppendTo[st, (root^2 / (1 + f^2 * root^2)) + (c*root + d)^2 / -
        |hängt an bei
        ((a*root + b)^2 + fPrime^2 * (c*root + d)^2) /. rootsRealis[[oo]]];
    ];

rootMin = Min[st];
    |kleinstes Element
l = {rootMin*f, 1, -rootMin};

lPrime = {-fPrime*(c*rootMin + d), a*rootMin + b, c*rootMin + d};

```

```

closestPointX = {-l[[1]] * l[[3]], -l[[2]] * l[[3]], l[[1]]^2 + l[[2]]^2};
closestPointX = Inverse[T].Transpose[R].closestPointX;
    [inverse Matrix] [transponiere]
closestPointX = closestPointX / closestPointX[[3]];

closestPointXPrime = {-lPrime[[1]] * lPrime[[3]],
- lPrime[[2]] * lPrime[[3]], lPrime[[1]]^2 + lPrime[[2]]^2};
closestPointXPrime = Inverse[TPrime].Transpose[RPrime].closestPointXPrime;
    [inverse Matrix] [transponiere]
closestPointXPrime = closestPointXPrime / closestPointXPrime[[3]];

Print["closestPointX = ", closestPointX];
    [gib aus]
Print["closestPointXPrime= ", closestPointXPrime];
    [gib aus]

K1Ex = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}};

P = K1.K1Ex;
PPrime = K2.K2Ex;

(* Reconstruct Points with linear Triangulation Method*)
    [Methode]

A = {closestPointX[[1]] * P[[3, All]] - P[[1, All]],
    [alle] [alle]
    closestPointX[[2]] * P[[3, All]] - P[[2, All]],
    [alle] [alle]
    closestPointXPrime[[1]] * PPrime[[3, All]] - PPrime[[1, All]],
    [alle] [alle]
    closestPointXPrime[[2]] * PPrime[[3, All]] - PPrime[[2, All]]];
    [alle] [alle]

SVDTest = SingularValueDecomposition[A];
    [Singulärwertzerlegung]
SpacePointPx =
{SVDTest[[3, 1, 4]], SVDTest[[3, 2, 4]], SVDTest[[3, 3, 4]], SVDTest[[3, 4, 4]]};

SpacePointPx = SpacePointPx / SpacePointPx[[4]];

AppendTo[SpacePointsMMNotScaled, SpacePointPx];
    [hänge an bei]
SpacePointsMMNotScaled = Map[#/#[[4]] &, SpacePointsMMNotScaled];
    [wende an]

Clear[st];
    [lösche]
];
GraficPoints = Map[{#[[1]], #[[2]], #[[3]]} &, SpacePointsMMNotScaled];
    [wende an]
GraficPoints2 = Map[{#[[1]], #[[2]], #[[3]]} &, SpacePointsMMNotScaled];
    [wende an]
plrange = {-1, 1};

```

```

Print[ListPointPlot3D[GraficPoints, AxesLabel → {x, y, z}, BoxRatios → 1,
  |gib aus |listenbezogenes 3D-Streudiagramm |Achsenbeschriftung |Seitenverhältnis der Box
  PlotRange → (*{plrange,plrange,plrange}*) All, PlotStyle → PointSize[0.023]];
  |Koordinatenbereich der Graphik |alle |Darstellungsstil |Punktgröße

Print["RForOk2 = ", RForOk2];
|gib aus
Print["tForOK2 = ", tForOK2];
|gib aus
t = RForOk2.tForOK2;
t = {t[[1]], t[[2]], t[[3]]};
Print["t = ", t];
|gib aus

Print["Reconstructed Points 3D = ", GraficPoints];
|gib aus |leite ab
G1 = Graphics3D[{Red, PointSize[0.02], Point[{0, 0, 0}]}];
  |3D-Graphik |rot |Punktgröße |Punkt
G2 = Graphics3D[{Blue, PointSize[0.01], Point[GraficPoints]}];
  |3D-Graphik |blau |Punktgröße |Punkt
G3 = Graphics3D[{Green, PointSize[0.02], Point[t]}];
  |3D-Graphik |grün |Punktgröße |Punkt
Print[Show[G1, G3, G2, Axes → True,
  |zeige an |Axen |wahr
  PlotRange → All, AxesLabel → {x, y, z} (*,BoxRatios→1*)];
  |alle |Achsenbeschriftung |Seitenverhältnis der Box

TwoDGraphicPoints = SpacePointsMMNotScaled;

TwoDGraphicPoints = Map[#[[1]] / #[[3]], #[[2]] / #[[3]]] &, TwoDGraphicPoints];
  |wende an
TwoDGraphicPoints = Map[{#[[1]], #[[2]]} &, TwoDGraphicPoints];
  |wende an
Print[ListPlot[TwoDGraphicPoints,
  |listenbezogene Graphik
  AxesLabel → {x, y}, PlotRange → All, PlotStyle → PointSize[0.01]];
  |Koordinatenb... |alle |Darstellungsstil |Punktgröße

];

```

## Programmcode Rektifizierung

```
(*Rektificationalgorithm Charles Loop & Zhengyou Zhang-----*)
(*-----*)

NewRectification[F_, e_, ePrime_, ImagePlaneC1Points_, ImagePlaneC2Points_] :=
Module[{Hp, HpPrime, HrPrime, Hr, w, wPrime, eInf, ePrimeInf, Vc,
Modul
ePrimeHorizontal, eHorizontal, RecPointsC2, RecPointsC1, RecGraphicPointsC1,
RecGraphicPointsC2, G2, G1, P, PPrime, pc, pcPrime, piWidth, piHeight,
pjWidth, pjHeight, pi = {}, pj = {}, n, PP, PPPPrime, pcpc, pcpcPrime,
A, APrime, B, BPrime, ex, ePrimex, z, z1, z2, piA, piB, piC, piD, piSA,
piSB, Hs, HsPrime, pjA, pjB, pjC, pjD, pjSA, pjSB, eL, eLPrime, zGuess},
Print["
[gib aus
Begin New Rectification with Distortion minimization
[beginne Kontext
criterion_
"];
For[i = 1, i ≤ Length[ImagePlaneC1Points], i++,
[For-Schleife      [Länge
AppendTo[pi, ImagePlaneC1Points[[i]]];
[hänge an bei
AppendTo[pj, ImagePlaneC2Points[[i]]];
[hänge an bei
];
(*Distortion minimization Criterion for finding z and w and w' *)

n = Length[pi];
[Länge
Print["pi = ", N[pi]];
[gib aus      [numerischer Wert
Print["pj = ", N[pj]];
[gib aus      [numerischer Wert
Print["n = ", n];
[gib aus

minXPi = Min[Map[#[[1]] &, pi]];
[kle.. [wende an
maxXPi = Max[Map[#[[1]] &, pi]];
[gr.. [wende an
minYPi = Min[Map[#[[2]] &, pi]];
[kle.. [wende an
maxYPi = Max[Map[#[[2]] &, pi]];
[gr.. [wende an

minXPj = Min[Map[#[[1]] &, pj]];
[kle.. [wende an
maxXPj = Max[Map[#[[1]] &, pj]];
[gr.. [wende an
minYPj = Min[Map[#[[2]] &, pj]];
[kle.. [wende an
```

```

maxYPj = Max[Map#[[2]] &, pj];
|gr...|wende an

Print["minXpi = ", N[minXPi]];
|gib aus      |numerischer Wert
Print["maxXPi = ", N[maxXPi]];
|gib aus      |numerischer Wert
Print["minYpi = ", N[minYPi]];
|gib aus      |numerischer Wert
Print["maxYpi = ", N[maxYPi]];
|gib aus      |numerischer Wert
Print["minXPj = ", N[minXPj]];
|gib aus      |numerischer Wert
Print["maxXPj = ", N[maxXPj]];
|gib aus      |numerischer Wert
Print["minYPj = ", N[minYPj]];
|gib aus      |numerischer Wert
Print["maxYPj = ", N[maxYPj]];
|gib aus      |numerischer Wert

piWidth = EuclideanDistance[Wxmax, Wxmin];
|euklidischer Abstand
piHeight = EuclideanDistance[Hymax, Hymin];
|euklidischer Abstand
pjWidth = EuclideanDistance[Wxmax, Wxmin];
|euklidischer Abstand
pjHeight = EuclideanDistance[Hymax, Hymin];
|euklidischer Abstand

Print["piWidth = ", N[piWidth]];
|gib aus      |numerischer Wert
Print["piHeight = ", N[piHeight]];
|gib aus      |numerischer Wert
Print["pjWidth = ", N[pjWidth]];
|gib aus      |numerischer Wert
Print["pjHeight = ", N[pjHeight]];
|gib aus      |numerischer Wert

(*projective transformation H_p and H_p'-----*)
(*Find w and w' by minimizing z-----*)
|finde

P = ConstantArray[0, {3, n}];
|konstantes Array
PPrime = ConstantArray[0, {3, n}];
|konstantes Array

pc = {0, 0};
pcPrime = {0, 0};
For[i = 1, i ≤ Length[pi] - 1, i++,
|For-Schleife   |Länge

pc = Total[pi];
|Gesamtsumme

```

```

    Gesamtsumme
pcPrime = Total[pj];
    |Gesamtsumme
];

pc = pc / n;
pcPrime = pcPrime / n;

Print["pc = ", MatrixForm[N[pc]];
|gib aus      |Matrizenform |numerischer Wert
Print["pcPrime = ", MatrixForm[N[pcPrime]]];
|gib aus      |Matrizenform |numerischer Wert

For[i = 1, i ≤ Length[pi], i++,
|For-Schleife |Länge
  P[[1, i]] = (pi[[i, 1]] - pc[[1]]);
  P[[2, i]] = (pi[[i, 2]] - pc[[2]]);

  PPrime[[1, i]] = (pj[[i, 1]] - pcPrime[[1]]);
  PPrime[[2, i]] = (pj[[i, 2]] - pcPrime[[2]]);
];
Print["P = ", N[P]];
|gib aus      |numerischer Wert
Print["PPrime = ", N[PPrime]];
|gib aus      |numerischer Wert
PP = P.Transpose[P];
|transponiere
Print["PP = ", N[PP]];
|gib aus      |numerischer Wert
PPPPrime = PPrime.Transpose[PPPrime];
|transponiere
Print["PPPPrime = ", N[PPPPrime]];
|gib aus      |numerischer Wert
Print["pc = ", N[pc]];
|gib aus      |numerischer Wert
pcpc = {{pc[[1]]}, {pc[[2]]}, {pc[[3]]}}.{{pc[[1]], pc[[2]], pc[[3]]}};

Print["pcpc = ", N[pcpc]];
|gib aus      |numerischer Wert
pcpcPrime = {{pcPrime[[1]]}, {pcPrime[[2]]}, {pcPrime[[3]]}}.
  {{pcPrime[[1]], pcPrime[[2]], pcPrime[[3]]}};

Print["pcpcPrime = ", N[pcpcPrime]];
|gib aus      |numerischer Wert

ex = {{0, -e[[3]], e[[2]]}, {e[[3]], 0, -e[[1]]}, {-e[[2]], e[[1]], 0}};
ePrimex = {{0, -ePrime[[3]], ePrime[[2]]},
  {ePrime[[3]], 0, -ePrime[[1]]}, {-ePrime[[2]], ePrime[[1]], 0}};

A = Transpose[ex].PP.ex;
|transponiere
B = Transpose[F].pcpc.F;
|transponiere

```

```

 $\text{APrime} = \text{Transpose}[\text{ePrimex}].\text{PPPPrime.ePrimex};$ 
 $\quad \text{[transponiere inverse Matrix]}$ 
 $\text{BPrime} = \text{Transpose}[F].\text{pcpcPrime.F};$ 
 $\quad \text{[transponiere inverse Matrix]}$ 

 $\text{Print["A = ", MatrixForm[N[A]]];}$ 
 $\quad \text{[gib aus Matritzenform numerischer Wert]}$ 
 $\text{Print["B = ", MatrixForm[N[B]]];}$ 
 $\quad \text{[gib aus Matritzenform numerischer Wert]}$ 
 $\text{Print["APrime = ", MatrixForm[N[APrime]]];}$ 
 $\quad \text{[gib aus Matritzenform numerischer Wert]}$ 
 $\text{Print["BPrime = ", MatrixForm[N[BPrime]]];}$ 
 $\quad \text{[gib aus Matritzenform numerischer Wert]}$ 

 $\text{Print["\{A[[1,1;;2]],A[[2,1;;2]]\}=", N[\{A[[1,1;;2]],A[[2,1;;2]]\}]]};$ 
 $\quad \text{[numerischer Wert]}$ 
 $\text{Print["\{APrime[[1,1;;2]],APrime[[2,1;;2]]\}=",}$ 
 $\quad \text{[gib aus numerischer Wert]}$ 
 $\quad N[\{APrime[[1,1;;2]],APrime[[2,1;;2]]\}]]};$ 
 $\quad \text{[numerischer Wert]}$ 

 $\text{DD = CholeskyDecomposition[\{A[[1,1;;2]],A[[2,1;;2]]\}];}$ 
 $\quad \text{[Cholesky-Zerlegung inverse Matrix]}$ 
 $\text{DDPrime = CholeskyDecomposition[\{APrime[[1,1;;2]],APrime[[2,1;;2]]\}];}$ 
 $\quad \text{[Cholesky-Zerlegung inverse Matrix]}$ 

 $\text{Print["DD =", N[DD]];}$ 
 $\quad \text{[gib aus numerischer Wert]}$ 
 $\text{Print["DDPrime =", N[DDPrime]];}$ 
 $\quad \text{[gib aus numerischer Wert]}$ 

 $\text{Print["\{B[[1,1;;2]],B[[2,1;;2]]\}=", N[\{B[[1,1;;2]],B[[2,1;;2]]\}]]};$ 
 $\quad \text{[numerischer Wert]}$ 
 $\text{Print["\{BPrime[[1,1;;2]],BPrime[[2,1;;2]]\}=",}$ 
 $\quad \text{[gib aus numerischer Wert]}$ 
 $\quad N[\{BPrime[[1,1;;2]],BPrime[[2,1;;2]]\}]]};$ 
 $\quad \text{[numerischer Wert]}$ 

 $\text{DTBD = }$ 
 $\quad \text{Eigensystem}[\text{Transpose}[\text{Inverse}[DD]].\{B[[1,1;;2]],B[[2,1;;2]]\}.\text{Inverse}[DD]];$ 
 $\quad \text{[Eigenwert transponiere inverse Matrix inverse Matrix]}$ 
 $\text{DTBPrimeD = Eigensystem}[\text{Transpose}[\text{Inverse}[DDPrime]].$ 
 $\quad \text{[Eigenwert transponiere inverse Matrix inverse Matrix]}$ 
 $\quad \{BPrime[[1,1;;2]],BPrime[[2,1;;2]]\}.\text{Inverse}[DDPrime];$ 
 $\quad \text{[inverse Matrix]}$ 

 $\text{Print["DTBD[[2,1]] = ", N[DTBD[[2,1]]]];}$ 
 $\quad \text{[gib aus numerischer Wert]}$ 
 $\text{Print["Eigensystem DTB1 = ", N[DTBD]];}$ 
 $\quad \text{[gib aus Eigenwert numerischer Wert]}$ 
 $\text{Print["Eigensystem DTBPrimeD = ", N[DTBPrimeD]];}$ 
 $\quad \text{[gib aus Eigenwert numerischer Wert]}$ 

```

```

gib aus  Eigensystem          numerischer Wert
z1 = Inverse[DD].DTBD[[2, 1]];
  [inverse Matrix
Print["z1 first = ", Inverse[DD].DTBD[[2, 1]]];
  [gib aus      [inverse Matrix
If[DTBD[[1, 2]] ≥ DTBD[[1, 1]],
  [wenn
    z1 = Inverse[DD].DTBD[[2, 2]];
      [inverse Matrix
Print["z1 second = ", z1];
  [gib aus
];

Print["z1= ", N[z1]];
  [gib aus      [numerischer Wert

z2 = Inverse[DDPrime].DTBPrimeD[[2, 1]];
  [inverse Matrix
Print["z2 first = ", z2];
  [gib aus
If[DTBPrimeD[[1, 2]] ≥ DTBPrimeD[[1, 1]],
  [wenn
    z2 = Inverse[DDPrime].DTBPrimeD[[2, 2]];
      [inverse Matrix
Print["z2 second = ", z2];
  [gib aus
];

Print["z2= ", N[z2]];
  [gib aus      [numerischer Wert

z = (z1 / Normalize[z1] + z2 / Normalize[z2]);
  [normalisiere      [normalisiere

z = {z[[1]], z[[2]], 0};
Print["z =", N[z]];
  [gib aus      [numerischer Wert

(*Similarity Transformation Hr and Hr' *)

w = {{0, -e[[3]], e[[2]]}, {e[[3]], 0, -e[[1]]}, {-e[[2]], e[[1]], 0}}.z;
wPrime = F.z;

Print["w = ", N[w]];
  [gib aus      [numerischer Wert
Print["wPrime = ", N[wPrime]];
  [gib aus      [numerischer Wert
wPrime = wPrime / wPrime[[3]];
Print["wPrime = ", N[wPrime]];
  [gib aus      [numerischer Wert
w = w / w[[3]];
Print["w = ", N[w]];
  [gib aus      [numerischer Wert

```

```

HpPrime = {{1, 0, 0}, {0, 1, 0}, {wPrime[[1]], wPrime[[2]], wPrime[[3]]}};
Print["HpPrime = ", MatrixForm[N[HpPrime]]];
|gib aus |Matrizenform |numerischer Wert

Hp = {{1, 0, 0}, {0, 1, 0}, {w[[1]], w[[2]], w[[3]]}};
Print["Hp = ", MatrixForm[N[Hp]]];
|gib aus |Matrizenform |numerischer Wert
ePrimeInf = HpPrime.ePrime;
eInf = Hp.e;

Print["ePrime inf = ", N[ePrimeInf]];
|gib aus |numerischer Wert
Print["e inf = ", N[eInf]];
|gib aus |numerischer Wert

Vc = 0.705; (*Wie bekomme ich Vc raus??*)
Hr = {{F[[3, 2]] - w[[2]] * F[[3, 3]], w[[1]] * F[[3, 3]] - F[[3, 1]], 0}, {F[[3, 1]] -
    w[[1]] * F[[3, 3]], F[[3, 2]] - w[[2]] * F[[3, 3]], F[[3, 3]] + Vc}, {0, 0, 1}};

HrPrime = {{wPrime[[2]] * F[[3, 3]] - F[[2, 3]], F[[1, 3]] - wPrime[[1]] * F[[3, 3]], 0}, {wPrime[[1]] * F[[3, 3]] - F[[1, 3]],
    wPrime[[2]] * F[[3, 3]] - F[[2, 3]], Vc}, {0, 0, 1}};

Print["HrPrime = ", MatrixForm[N[HrPrime]]];
|gib aus |Matrizenform |numerischer Wert
Print["Hr = ", MatrixForm[N[Hr]]];
|gib aus |Matrizenform |numerischer Wert

ePrimeHorizontal = HrPrime.ePrimeInf;
Print["ePrimeHorizontal = ", N[ePrimeHorizontal]];
|gib aus |numerischer Wert
eHorizontal = Hr.eInf;
Print["eHorizontal = ", N[eHorizontal]];
|gib aus |numerischer Wert

(*Shearing Transformation H_s and H_s'*)

piA = {(piWidth) / 2, 0, 1};
piB = {piWidth, (piHeight) / 2, 1};
piC = {(piWidth) / 2, piHeight, 1};
piD = {0, (piHeight) / 2, 1};

piA = Hr.Hp.piA;
piB = Hr.Hp.piB;
piC = Hr.Hp.piC;
piD = Hr.Hp.piD;

Print["piA = ", piA];
|gib aus
Print["piB = ", piB];
|gib aus
Print["piC = ", piC];
|gib aus
Print["piD = ", piD];
|gib aus

```

```

Lyn aus

piX = piB - piD;
piY = piC - piA;

Print["piX = ", piX];
|gib aus
Print["piY = ", piY];
|gib aus

piSA = (piHeight^2 * piX[[2]]^2 + piWidth^2 * piY[[2]]^2) /
((piHeight * piWidth) * (piX[[2]] * piY[[1]] - piX[[1]] * piY[[2]]));
piSB = (piHeight^2 * piX[[1]] * piX[[2]] + piWidth^2 * piY[[1]] * piY[[2]]) /
((piHeight * piWidth) * (piX[[1]] * piY[[2]] - piX[[2]] * piY[[1]]));
Print["piSA = ", N[piSA]];
|gib aus          |numerischer Wert
Print["piSB = ", N[piSB]];
|gib aus          |numerischer Wert

Hs = {{piSA, piSB, 0}, {0, 1, 0}, {0, 0, 1}};

pjWidth = pjWidth * 1;
pjHeight = pjHeight * 1;

pjA = {(pjWidth) / 2, 0, 1};
pjB = {pjWidth, (pjHeight) / 2, 1};
pjC = {(pjWidth) / 2, pjHeight, 1};
pjD = {0, (pjHeight) / 2, 1};

pjA = HrPrime.HpPrime.pjA;
pjB = HrPrime.HpPrime.pjB;
pjC = HrPrime.HpPrime.pjC;
pjD = HrPrime.HpPrime.pjD;

Print["pjA = ", N[pjA]];
|gib aus          |numerischer Wert
Print["pjB = ", N[pjB]];
|gib aus          |numerischer Wert
Print["pjC = ", N[pjC]];
|gib aus          |numerischer Wert
Print["pjD = ", N[pjD]];
|gib aus          |numerischer Wert
pjX = pjB - pjD;
pjY = pjC - pjA;

Print["pjX = ", N[pjX]];
|gib aus          |numerischer Wert
Print["pjY = ", N[pjY]];
|gib aus          |numerischer Wert

```

```

pjSA = (pjHeight^2 * pjX[[2]]^2 + pjWidth^2 * pjY[[2]]^2) /
      ((pjHeight * pjWidth) * (pjX[[2]] * pjY[[1]] - pjX[[1]] * pjY[[2]]));
pjSB = (pjHeight^2 * pjX[[1]] * pjX[[2]] + pjWidth^2 * pjY[[1]] * pjY[[2]]) /
      ((pjHeight * pjWidth) * (pjX[[1]] * pjY[[2]] - pjX[[2]] * pjY[[1]]));
Print["pjSA = ", N[pjSA]];
gib aus numerischer Wert
Print["pjSB = ", N[pjSB]];
gib aus numerischer Wert

HsPrime = {{pjSA, pjSB, 0}, {0, 1, 0}, {0, 0, 1}};

eL = ConstantArray[0, {8, 3}];
konstantes Array
eLPrime = ConstantArray[0, {8, 3}];
konstantes Array

RecPointsC2 = ConstantArray[0, {8, 3}];
konstantes Array
RecPointsC1 = ConstantArray[0, {8, 3}];
konstantes Array

For[i = 1, i ≤ 8, i++,
For-Schleife

  RecPointsC1[[i]] = (*Hs.Hr.*) Hp.pi[[i]];
  eL[[i]] = N[Cross[eInf, RecPointsC1[[i, All]]]];
.. Kreuzprodukt alle
  RecPointsC1[[i]] = RecPointsC1[[i]] / RecPointsC1[[i, 3]];

  RecPointsC2[[i]] = (*HsPrime.HrPrime.*) HpPrime.pj[[i]];
  eLPrime[[i]] = N[Cross[ePrimeInf, RecPointsC2[[i, All]]]];
.. Kreuzprodukt alle
  RecPointsC2[[i]] = RecPointsC2[[i]] / RecPointsC2[[i, 3]];

];
Print["RecPointsC1 = ", MatrixForm[N[RecPointsC1]]];
gib aus Matrizenform numerischer Wert
Print["RecPointsC2 = ", MatrixForm[N[RecPointsC2]]];
gib aus Matrizenform numerischer Wert

RecGraphicPointsC1 = Map[{#[[1]], #[[2]]} &, RecPointsC1];
wende an
RecGraphicPointsC2 = Map[{#[[1]], #[[2]]} &, RecPointsC2];
wende an

G1 = Show[ListPlot[RecGraphicPointsC1 [[1 ;; 8]], PlotStyle → Darker[Green]],
zeig... listenbezogene Graphik Darstellungsstil dunkler grün
ListLinePlot[{RecGraphicPointsC1 [[4, All]]},
listenbezogene Liniengraphik alle
]

```

```

ListLinePlot[RecGraphicPointsC1[[1, All]], RecGraphicPointsC1[[2, All]],
            |alle| alle
ListLinePlot[RecGraphicPointsC1[[3, All]], RecGraphicPointsC1[[4, All]],
            |alle| alle
ListLinePlot[RecGraphicPointsC1[[8, All]], RecGraphicPointsC1[[7, All]],
            |alle| alle
ListLinePlot[RecGraphicPointsC1[[6, All]], RecGraphicPointsC1[[5
            |alle| alle
            , All]], RecGraphicPointsC1[[8, All]]], PlotStyle -> Darker[Green],
            |alle| Darstellungsstil dunkler grün
ListLinePlot[{RecGraphicPointsC1[[1, All]], RecGraphicPointsC1[[5, All]]},
            |listenbezogene Liniengraphik| alle| alle
            PlotStyle -> Darker[Green],
            |Darstellungsstil dunkler grün
ListLinePlot[{RecGraphicPointsC1[[2, All]], RecGraphicPointsC1[[6, All]]},
            |listenbezogene Liniengraphik| alle| alle
            PlotStyle -> Darker[Green],
            |Darstellungsstil dunkler grün
ListLinePlot[{RecGraphicPointsC1[[3, All]], RecGraphicPointsC1[[7, All]]},
            |listenbezogene Liniengraphik| alle| alle
            PlotStyle -> Darker[Green]];
            |Darstellungsstil dunkler grün

G2 = Show[ListPlot[RecGraphicPointsC2[[1 ;; 8]], PlotStyle -> Darker[Red]],
          |zeig...| listenbezogene Graphik| Darstellungsstil dunkler rot
ListLinePlot[{RecGraphicPointsC2[[4, All]]},
            |listenbezogene Liniengraphik| alle
            RecGraphicPointsC2[[1, All]], RecGraphicPointsC2[[2, All]],
            |alle| alle
            RecGraphicPointsC2[[3, All]], RecGraphicPointsC2[[4, All]],
            |alle| alle
            RecGraphicPointsC2[[8, All]], RecGraphicPointsC2[[7, All]],
            |alle| alle
            RecGraphicPointsC2[[6, All]], RecGraphicPointsC2[[5
            |alle|
            , All]], RecGraphicPointsC2[[8, All]]], PlotStyle -> Darker[Red],
            |alle| alle| Darstellungsstil dunkler rot
ListLinePlot[{RecGraphicPointsC2[[1, All]]},
            |listenbezogene Liniengraphik| alle
            RecGraphicPointsC2[[5, All]], PlotStyle -> Darker[Red],
            |alle| Darstellungsstil dunkler rot
ListLinePlot[{RecGraphicPointsC2[[2, All]], RecGraphicPointsC2[[6, All]]},
            |listenbezogene Liniengraphik| alle| alle
            PlotStyle -> Darker[Red],
            |Darstellungsstil dunkler rot
ListLinePlot[{RecGraphicPointsC2[[3, All]], RecGraphicPointsC2[[7, All]]},
            |listenbezogene Liniengraphik| alle| alle
            PlotStyle -> Darker[Red]];
            |Darstellungsstil dunkler rot
Print[Show[G1, G2, PlotRange -> All]];
|gib aus| zeige an| Koordinatenb...| alle

yAx = 1.4;

```

```

xAx = -0.6;

Print[Show[G2, ContourPlot[eL[[1]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |gib aus |zeige an |Konturgraphik
  ContourPlot[eL[[2]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[3]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[4]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[5]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[6]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[7]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eL[[8]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], G1,
  |Konturgraphik
  ContourPlot[eLPrime[[1]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[2]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[3]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[4]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[5]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[6]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[7]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}], 
  |Konturgraphik
  ContourPlot[eLPrime[[8]].{x, y, 1} == 0, {x, xAx, yAx}, {y, xAx, yAx}],
  |Konturgraphik
  PlotRange → Automatic]];
  |Koordinatenb... |automatisch

Print["
  |gib aus
End New Rectification with Disortion minimization
|beende Kontext
  criterion_-----
"];
];

```

## Programmcode Sortierungsalgorithmus

```
(*SortPoints Algorithm-----  
-----*)  
  
FindMinMax[PointList_] :=  
Module[{imin, imax, jmin, jmax, iSplits, jSplits, iDistance, jDistance},  
Modul  
imin = 10000;  
imax = 0;  
jmin = 10000;  
jmax = 0;  
  
Print["Splits = ", splits];  
gib aus  
For[i = 1, i ≤ Length[PointList], i++,  
Länge  
If[imin ≥ PointList[[i, 2]], imin = PointList[[i, 2]]];  
wenn  
If[jmin ≥ PointList[[i, 1]], jmin = PointList[[i, 1]]];  
wenn  
If[imax ≤ PointList[[i, 2]], imax = PointList[[i, 2]]];  
wenn  
If[jmax ≤ PointList[[i, 1]], jmax = PointList[[i, 1]]];  
wenn  
];  
  
Print["imin = ", imin, " imax = ", imax, " jmin = ", jmin, " jmax = ", jmax];  
gib aus  
  
jSplits = ConstantArray[0, {1, splits + 1}];  
konstantes Array  
jSplits[[1, 1]] = jmin;  
  
jDistance = jmax - jmin;  
  
For[i = 2, i ≤ splits + 1, i++,  
For-Schleife  
jSplits[[1, i]] = jSplits[[1, i - 1]] + (jDistance / splits)  
];  
  
Print["jSplits = ", jSplits];  
gib aus  
iSplits = ConstantArray[0, {1, splits + 1}];  
konstantes Array  
iSplits[[1, 1]] = imin;  
  
iDistance = imax - imin;  
  
For[i = 2, i ≤ splits + 1, i++,  
For-Schleife  
iSplits[[1, i]] = iSplits[[1, i - 1]] + (iDistance / splits)  
];  
  
Print["iSplits = ", MatrixForm[iSplits]];  
gib aus  
Matrixform
```

```

Lyn aus          Linienelement
GoThroughConvexHulls[iSplits, jSplits, PointList];
];

GoThroughConvexHulls[iSplits_, jSplits_, PointList_] := Module[{ConvexHull},
  (*ConvexHull = ConstantArray[0,{splits,1}];*)
  ConvexHull = {};
  For[ii = 1, ii < Length[Flatten[iSplits]] - 1, ii++,
    AppendTo[ConvexHull, FindPointsInConvexHull[
      iSplits[[1, ii]], iSplits[[1, ii + 1]], jSplits, PointList]];
  ];
  (*Print["ConvexHull =", MatrixForm[Flatten[ConvexHull]]];*)
  ConvexHull = Flatten[ConvexHull];
  FindStartVectors[ConvexHull];
];

FindPointsInConvexHull[iSplitsBottom_, iSplitsTop_, jSplits_, PointList_] :=
Module[{ConvexHullCell, ConvexHullCellList, ConvexHullCellKeys},
  ConvexHullCell = {};
  ConvexHullCellKeys = <||>;
  For[b = 1, b < Length[Flatten[jSplits]] - 1, b++,
    For[i = 1, i < Length[PointList], i++,
      If[PointList[[i, 2]] >= iSplitsBottom &&
        PointList[[i, 2]] <= iSplitsTop && PointList[[i, 1]] >= jSplits[[1, b]] &&
        PointList[[i, 1]] <= jSplits[[1, b + 1]],
        AssociateTo[ConvexHullCellKeys, CoordJ > PointList[[i, 1]]];
        AssociateTo[ConvexHullCellKeys, CoordI > PointList[[i, 2]]];
        AssociateTo[ConvexHullCellKeys, CellJ > b];
        AssociateTo[ConvexHullCellKeys, CellI > ii];
        AppendTo[ConvexHullCell, ConvexHullCellKeys];
      ];
    ];
  ];
  Return[ConvexHullCell];
]

```

[Lynx zurück](#)

```

];
```

```

FindStartVectors[ConvexHull_] := Module[{StartPointCloud = {}},
  Modul
  StartPointCloudKeys = <| |>, VecI, VecJ, counti, countj, Start, nextI, nextJ|,
  Modul
  For[rr = 0, rr < Length[ConvexHull] - 1, rr++,
    For-Schleife      Länge
    If[ConvexHull[[rr]][CellI] == 1 && ConvexHull[[rr]][CellJ] <= splits,
      wenn
      AssociateTo[StartPointCloudKeys,
        assoziere mit
        {CoordJ -> ConvexHull[[rr]][CoordJ], CoordI -> ConvexHull[[rr]][CoordI],
         CellJ -> ConvexHull[[rr]][CellJ], CellI -> ConvexHull[[rr]][CellI]}];
      AppendTo[StartPointCloud, StartPointCloudKeys]];
      hänge an bei
    ];
    Wenn
    If[ConvexHull[[rr]][CellI] <= splits && ConvexHull[[rr]][CellJ] == 1,
      Wenn
      AssociateTo[StartPointCloudKeys,
        assoziere mit
        {CoordJ -> ConvexHull[[rr]][CoordJ], CoordI -> ConvexHull[[rr]][CoordI],
         CellJ -> ConvexHull[[rr]][CellJ], CellI -> ConvexHull[[rr]][CellI]}];
      AppendTo[StartPointCloud, StartPointCloudKeys]];
      hänge an bei
    ];
  ];
  ];
  counti = 100;
  countj = 100;
  VecI = 0;
  VecJ = 0;

  For[aa = 0, aa < Length[StartPointCloud], aa++,
    For-Schleife      Länge
    If[StartPointCloud[[aa]][CoordI] <= counti &&
      Wenn
      StartPointCloud[[aa]][CellJ] <= splits && StartPointCloud[[aa]][CellI] == 1,
      counti = StartPointCloud[[aa]][CoordI];
      VecI = StartPointCloud[[aa]];
    ];
    Wenn
    If[StartPointCloud[[aa]][CoordJ] <= countj &&
      Wenn
      StartPointCloud[[aa]][CellI] <= splits && StartPointCloud[[aa]][CellJ] == 1,
      countj = StartPointCloud[[aa]][CoordJ];
      VecJ = StartPointCloud[[aa]];
    ];
  ];

```

```

];
];

Print["VecI = ", VecI];
|gib aus

Print["VecJ = ", VecJ];
|gib aus

For[yy = 0, yy < Length[StartPointCloud], yy++,
|For-Schleife |Länge
  If[ StartPointCloud[[yy]][CellI] == 1 && StartPointCloud[[yy]][CellJ] < splits &&
|wenn
    StartPointCloud[[yy]][CoordJ] <= VecI[CoordJ] &&
    StartPointCloud[[yy]][CoordI] <= VecI[CoordI] + 0.8,
    (*statt plus 0.8 vllt einen Offset aus distanzen bestimmen...
      |Versatz
      nur wie wenn andere Nachbarn noch unbekannt?*)

    VecI = StartPointCloud[[yy]];
  ];
];

For[yy = 0, yy < Length[StartPointCloud], yy++,
|For-Schleife |Länge
  If[ StartPointCloud[[yy]][CellJ] == 1 && StartPointCloud[[yy]][CellI] <= splits &&
|wenn
    StartPointCloud[[yy]][CoordI] <= VecJ[CoordI] &&
    StartPointCloud[[yy]][CoordJ] <= VecJ[CoordJ] + 0.5,

    VecJ = StartPointCloud[[yy]];
  ];
];

Print["VecI = ", VecI, " VecJ =", VecJ];
|gib aus

If[VecI == VecJ, Start = VecI, Start = VecJ];
|wenn
Print["Start = ", Start];
|gib aus
(*Find NextI and NextJ *)
|finde
nextI = <|CoordJ → 100000, CoordI → 100000|>;
nextJ = <|CoordJ → 100000, CoordI → 100000|>

Print["StartPointCloud = ", StartPointCloud];
|gib aus
For[bb = 0, bb < Length[StartPointCloud], bb++,
|For-Schleife |Länge

  If[StartPointCloud[[bb]][CellI] == Start[CellI] ||
|wenn
    StartPointCloud[[bb]][CellI] == Start[CellI] + 1 ||
    StartPointCloud[[bb]][CellI] == Start[CellI] - 1,

```

```

If[StartPointCloud[[bb]][CoordI] ≥ Start[CoordI] && StartPointCloud[[bb]][
| wenn
    CoordJ] ≤ nextI[CoordJ] && StartPointCloud[[bb]] ≠ Start,
    nextI = StartPointCloud[[bb]];
    Print["nextI first = ", nextI];
    | gib aus
];
];

If[StartPointCloud[[bb]][CellJ] == Start[CellJ] ||
| wenn
    StartPointCloud[[bb]][CellJ] == Start[CellJ] + 1 ||
    StartPointCloud[[bb]][CellJ] == Start[CellJ] - 1,
    If[StartPointCloud[[bb]][CoordJ] ≥ Start[CoordJ] && StartPointCloud[[bb]][
| wenn
    CoordI] ≤ nextJ[CoordI] && StartPointCloud[[bb]] ≠ Start,
    nextJ = StartPointCloud[[bb]];
    Print["nextJ first = ", nextJ];
    | gib aus
];
];
];

(*Check if NextI and NextJ are final Values*)
| Werte

For[bb = 0, bb ≤ Length[StartPointCloud], bb++,
| For-Schleife | Länge

If[Abs[nextJ[CoordI] - Start[CoordI]] ≥
| ... | Absolutwert
    Abs[StartPointCloud[[bb]][CoordI] - Start[CoordI]] &&
    | Absolutwert
    StartPointCloud[[bb]] ≠ Start && Abs[nextJ[CoordJ] - Start[CoordJ]] >
    | Absolutwert
        Abs[StartPointCloud[[bb]][CoordJ] - Start[CoordJ]],
    | Absolutwert
    nextJ = StartPointCloud[[bb]];
];
];

If[Abs[nextI[CoordJ] - Start[CoordJ]] ≥
| ... | Absolutwert
    Abs[StartPointCloud[[bb]][CoordJ] - Start[CoordJ]] &&
    | Absolutwert
    StartPointCloud[[bb]] ≠ Start && Abs[nextI[CoordI] - Start[CoordI]] >
    | Absolutwert
        Abs[StartPointCloud[[bb]][CoordI] - Start[CoordI]],
    | Absolutwert
    nextI = StartPointCloud[[bb]];
];
];

```

```

Print["nexti = ", nextI];
[gib aus
Print["nextj = ", nextJ];
[gib aus
CreatePossiblePointsListsIAndJ[nextI, nextJ, Start, ConvexHull];
];

CreatePossiblePointsListsIAndJ[nextI_, nextJ_, Start_, ConvexHull_] :=
Module[{IList = {}, JList = {}, IDir, JDir, distance, cache, PotNextI, PotNextJ},
Modul
aj = 2;
restJ = splits;

For[rr = 0, rr < Length[ConvexHull], rr++,
For-Schleife      [Länge

If[ConvexHull[[rr]][CellJ] <= aj && ConvexHull[[rr]][CellI] <= splits,
[wenn
AppendTo[IList, ConvexHull[[rr]]];
[hänge an bei
];
If[ConvexHull[[rr]][CellI] <= aj && ConvexHull[[rr]][CellJ] <= splits,
[wenn
AppendTo[JList, ConvexHull[[rr]]];
[hänge an bei
];
];
Print["IList = ", IList];
[gib aus
Print["JList = ", JList];
[gib aus
FindNeighbours[IList, JList, Start, nextI, nextJ, ConvexHull];
];

```

```

FindNeighbours[IList_, JList_, Start_, nextI_, nextJ_, ConvexHull_] :=
Module[{SortedPointsKeys = <||>, SortedPoints = {}, proportion},
Modul
proportionI, jtemp, itemp, PotNextJDir, distanceNextPotPointJ, PotNextIDir ,
distanceNextPotPointI, NeighbourNumberJ, NeighbourNumberI, distanceJ,
distanceI, NextJDir, NextIDir, StartPropJForFirstCompleteGridJ},

Print["
[gib aus

Detect I and J rows of
[imaginäre Einheit I
Points_-----


"]];

Clear[CheckPoints];
[lösche
CheckPoints = {};
StartPoint = Start;
StartPointI = Start;
StartPointJ = Start;
NextPointI = nextI;
NextPointJ = nextJ;

Print["Start = ", Start];
[gib aus
Print[" nextI = ", nextI];
[gib aus
Print["nextJ = ", nextJ];
[gib aus
IDirStart = {nextI[CoordI] - Start[CoordI], nextI[CoordJ] - Start[CoordJ]};
JDirStart = {nextJ[CoordI] - Start[CoordI], nextJ[CoordJ] - Start[CoordJ}];

Print["IDir = ", IDirStart];
[gib aus
Print["JDir = ", JDirStart];
[gib aus
StartProportionI = nextI[CoordJ] - Start[CoordJ];
StartProportionJ = nextJ[CoordI] - Start[CoordI];

StartDistanceI = Sqrt[Abs[IDirStart[[1]]^2 + IDirStart[[2]]^2]];
[Qua... [Absolutwert
StartDistanceJ = Sqrt[Abs[JDirStart[[1]]^2 + JDirStart[[2]]^2]];
[Qua... [Absolutwert

AssociateTo[StartPoint, {NeighbourJ → 1, NeighbourI → 1}];
[assoziere mit
AssociateTo[NextPointI, {NeighbourJ → 1, NeighbourI → 2}];
[assoziere mit
AssociateTo[NextPointJ, {NeighbourJ → 2, NeighbourI → 1}];
[assoziere mit
AppendTo[SortedPoints, {StartPoint, NextPointI, NextPointJ}];
[hänge an bei

```

```

AppendTo[CheckPoints, Start];
|hänge an bei

AppendTo[CheckPoints, nextJ];
|hänge an bei

AppendTo[CheckPoints, nextI];
|hänge an bei

NeighbourNumberJ = 2;
NeighbourNumberI = 2;
aI = 2;
aJ = 2;
NextJDir = JDirStart;
distanceJ = StartDistanceJ;

proportionJ = StartProportionJ;
StartPropJForFirstCompleteGridJ = StartProportionJ;

For[pp = 1, pp < Length[JList] * 2, pp++,
[For-Schleife   |Länge
  For[tt = 1, tt < Length[JList], tt++,
    [For-Schleife   |Länge
      If[JList[[tt]][CoordJ] != StartPointJ[CoordJ] &&
       |wenn
        JList[[tt]][CoordJ] >= NextPointJ[CoordJ] && JList[[tt]][CoordI] >=
        NextPointJ[CoordI] || JList[[tt]][CoordI] <= NextPointJ[CoordI],
        PotNextJDir = {JList[[tt]][CoordJ] - NextPointJ[CoordJ],
                      JList[[tt]][CoordI] - NextPointJ[CoordI]};
        distanceNextPotPointJ = Sqrt[Abs[PotNextJDir[[1]]^2 + PotNextJDir[[2]]^2]];
        |Qua...|Absolutwert
      ];
      If[JList[[tt]][CoordJ] != NextPointJ[CoordJ] &&
       |wenn
        JList[[tt]][CoordJ] != StartPointJ[CoordJ] &&
        (JList[[tt]][CoordI]) <= NextPointJ[CoordI] + proportionJ + 0.3 &&
        (JList[[tt]][CoordI]) >= NextPointJ[CoordI] + proportionJ - 0.3 &&
        distanceNextPotPointJ <= (distanceJ + 0.5) &&
        JList[[tt]][CoordJ] >= NextPointJ[CoordJ] + (distanceJ / 2),
        StartPointJ = NextPointJ;
        NextPointJ = JList[[tt]];
      ];
      Print["Test J = ", JList[[tt]]];
      |gib aus
      NextJDir = {NextPointJ[CoordJ] - StartPointJ[CoordJ],
                  NextPointJ[CoordI] - StartPointJ[CoordI]};
      distanceJ = Sqrt[Abs[NextJDir[[1]]^2 + NextJDir[[2]]^2]];
      |Qua...|Absolutwert
      proportionJ = NextPointJ[CoordI] - StartPointJ[CoordI];
      aJ = aJ + 1;
      AppendTo[CheckPoints, JList[[tt]]];
      |hänge an bei
    ];
  ];
];

```

```

AssociateTo[SortedPointsKeys,
  assoziiere mit
  { JList[[tt]], NeighbourJ → aJ, NeighbourI → 1 }];
AppendTo[SortedPoints, SortedPointsKeys];
  hänge an bei
LastJPointsCellI = JList[[tt]][CellI];
CheckLastPointJ = JList[[tt]];
AssociateTo[CheckLastPointJ, {NeighbourJ → aJ, NeighbourI → 1}];
  assoziiere mit
];
];
];
(*];*)

AppendTo[SortedPoints, SaftyListJ[Start, CheckLastPointJ,
  hänge an bei
  proportionJ, LastJPointsCellI, ConvexHull, distanceJ, NextJDir]];

AppendTo[SortedPoints, CompleteJGrid[ nextI, ConvexHull, StartDistanceJ,
  hänge an bei
  StartPropJForFirstCompleteGridJ, Start , NeighbourNumberJ, aI]];

proportionI = StartProportionI;
distanceI = StartDistanceI;
NextIDir = IDirStart;

For[pp = 1, pp ≤ Length[IList] * 2, pp++,
  For-Schleife   Länge
  For[tt = 1, tt ≤ Length[IList], tt++,
    For-Schleife   Länge
    If[IList[[tt]][CoordI] ≠ StartPointI[CoordI] &&
      wenn
        IList[[tt]][CoordJ] ≥ NextPointI[CoordJ] || IList[[tt]][CoordJ] ≤
        NextPointI[CoordJ] && IList[[tt]][CoordI] ≥ NextPointI[CoordI],
        PotNextIDir = {IList[[tt]][CoordJ] - NextPointI[CoordJ],
                      IList[[tt]][CoordI] - NextPointI[CoordI]};
        distanceNextPotPointI = Sqrt[Abs[PotNextIDir[[1]]^2 + PotNextIDir[[2]]^2]];
          Qua... Absolutwert
    ];
    If[IList[[tt]][CoordI] ≠ NextPointI[CoordI] &&
      wenn
        IList[[tt]][CoordI] ≠ StartPointI[CoordI] &&
        (IList[[tt]][CoordJ]) ≤ NextPointI[CoordJ] + proportionI + 0.3 &&
        (IList[[tt]][CoordJ]) ≥ NextPointI[CoordJ] + proportionI - 0.3 &&
        distanceNextPotPointI ≤ distanceI + 0.5 &&
        IList[[tt]][CoordI] ≥ NextPointI[CoordI] + (distanceI / 3),
        Print["Test I = ", IList[[tt]]];
        gib aus   imaginäre Einheit I
        StartPointI = NextPointI;
        NextPointI = IList[[tt]];
    ];
  ];
];
];

```

```

NextIDir = {NextPointI[CoordJ] - StartPointI[CoordJ],
            NextPointI[CoordI] - StartPointI[CoordI]};
distanceI = Sqrt[Abs[NextIDir[[1]]^2 + NextIDir[[2]]^2]];

$$\sqrt{\text{Quadrat der Summe der Quadrate}}$$


proportionI = NextPointI[CoordJ] - StartPointI[CoordJ];
(*propJForGrind = Abs[NextPointI[CoordJ] - StartPointI[CoordJ]];*)

$$\text{Absolutwert}$$


propJForGrind = Abs[StartProportionJ];

$$\text{Absolutwert}$$


aI = aI + 1;

AppendTo[CheckPoints, IList[[tt]]];

$$\text{hängt an bei}$$


AssociateTo[SortedPointsKeys,

$$\text{assoziieren mit}$$

{ IList[[tt]], NeighbourJ \rightarrow 1, NeighbourI \rightarrow aI}]];
AppendTo[SortedPoints, SortedPointsKeys];

$$\text{hängt an bei}$$


LastIPointsCellJ = IList[[tt]][CellJ];
CheckLastPointI = IList[[tt]];
AssociateTo[CheckLastPointI, {NeighbourJ \rightarrow 1, NeighbourI \rightarrow aI}];

$$\text{assoziieren mit}$$


NeighbourNumberJ = NeighbourNumberJ;
AppendTo[SortedPoints, SaftyListI[Start, CheckLastPointI,

$$\text{hängt an bei}$$

proportionI, LastIPointsCellJ, ConvexHull, distanceI, NextIDir]];

AppendTo[SortedPoints, CompleteJGrid[IList[[tt]],

$$\text{hängt an bei}$$

ConvexHull, distanceJ, propJForGrind, Start, NeighbourNumberJ, aI]];

];

];

];

Print["Länge ConvexHull = ", Length[ConvexHull]];

$$\text{gibt aus}$$

Print["Länge SortedPoints = ", Length[Flatten[SortedPoints]]];

$$\text{gibt aus}$$

If[Length[Flatten[SortedPoints]] \neq Length[ConvexHull] \&& splits \leq 8,

$$\dots \text{Länge} \text{ ebne ein}$$


$$\text{Länge}$$

splits = splits + 1;
Clear[CheckPoints];

$$\text{löschen}$$

CheckPoints = {};
Print[

$$\text{gibt aus}$$

Another
round-----

```

```

"];
  FindMinMax[PointList],
  Print["
    gib aus
End and
[beende Kontext
Result_-----


"];
  Print["SortedPoints = ", Flatten[SortedPoints]];
  [gib aus      [ebne ein
  DrawGraph[SortedPoints];
  splits = 2;
];
];
[Modul

SaftyListJ[Start_, CheckLastPointJ_, proportionJ_,
  LastJPointsCellI_, ConvexHull_, distanceJ_, NextJDir_] :=
Module[{SaftyList = {}, SaftyKeys = <||>, SaftyKeysList = {}},
propJ, lastDirJ, lastdistanceJ, PotNextJDir, tempj,
distanceNextPotPointJ, nextNeighbourNumber, StartAtThisJPoint},

If[Length[SaftyKeysList] != 0,
[... [Länge
  Clear[SaftyKeysList];
  [lösche
];
StartAtThisJPoint = CheckLastPointJ;
nextNeighbourNumber = CheckLastPointJ[NeighbourJ] + 1;
propJ = proportionJ;
lastDirJ = NextJDir;
lastdistanceJ = distanceJ;

For[ii = 1, ii <= Length[ConvexHull], ii++,
[For-Schleife      [Länge

  If[ConvexHull[[ii]][CellI] == CheckLastPointJ[CellI] + 1 ||
[wenn
    ConvexHull[[ii]][CellI] == CheckLastPointJ[CellI] - 1 ||
    ConvexHull[[ii]][CellI] == CheckLastPointJ[CellI] &&
    ConvexHull[[ii]][CellJ] >= CheckLastPointJ[CellJ],

    AppendTo[SaftyList, ConvexHull[[ii]]];
    [hängt an bei
];
];
];

Print["SaftyList J = ", SaftyList];
[gib aus

```

```

Lyn aus
For[ll = 1, ll < Length[SaftyList], ll++,
|For-Schleife |Länge
  If[SaftyList[[ll]][CoordJ] != Start[CoordJ] &&
  |wenn
    SaftyList[[ll]][CoordJ] >= StartAtThisJPoint[CoordJ] &&
    SaftyList[[ll]][CoordI] >= StartAtThisJPoint[CoordI] ||
    SaftyList[[ll]][CoordI] <= StartAtThisJPoint[CoordI] ,
    PotNextJDir = {StartAtThisJPoint[CoordJ] - SaftyList[[ll]][CoordJ],
      StartAtThisJPoint[CoordI] - SaftyList[[ll]][CoordI]};
    distanceNextPotPointJ = Sqrt[Abs[PotNextJDir[[1]]^2 + PotNextJDir[[2]]^2]];
    |Qua...|Absolutwert
  ];
  If[SaftyList[[ll]][CoordJ] != Start[CoordJ] &&
  |wenn
    SaftyList[[ll]][CoordJ] != StartAtThisJPoint[CoordJ] &&
    SaftyList[[ll]][CoordJ] >= StartAtThisJPoint[CoordJ] &&
    SaftyList[[ll]][CoordI] <= StartAtThisJPoint[CoordI] + propJ + 0.3 &&
    SaftyList[[ll]][CoordI] >= StartAtThisJPoint[CoordI] + propJ - 0.3 &&
    distanceNextPotPointJ < lastdistanceJ + 0.5 &&
    SaftyList[[ll]][CoordJ] >= StartAtThisJPoint[CoordJ] + lastdistanceJ / 3
    (**(3/4)* ) && MemberQ[CheckPoints, SaftyList[[ll]]] == False,
    |enthalten? |falsch
    Print["SaftyListPoint = ", SaftyList[[ll]]];
    |gib aus
    AppendTo[CheckPoints, SaftyList[[ll]]];
    |hänge an bei
    AssociateTo[SaftyKeys, {SaftyList[[ll]]},
    |assoziere mit
      NeighbourJ → nextNeighbourNumber , NeighbourI → CheckLastPointJ[NeighbourI}}];
    AppendTo[SaftyKeysList, SaftyKeys];
    |hänge an bei

    lastDirJ = {StartAtThisJPoint[CoordJ] - SaftyList[[ll]][CoordJ],
      StartAtThisJPoint[CoordI] - SaftyList[[ll]][CoordI]};
    lastdistanceJ = Sqrt[Abs[lastDirJ[[1]]^2 + lastDirJ[[2]]^2]];
    |Qua...|Absolutwert
    propJ = SaftyList[[ll]][CoordI] - StartAtThisJPoint[CoordI];

    StartAtThisJPoint = SaftyList[[ll]];
    Print["StartAtThisJPoint = ", StartAtThisJPoint];
    |gib aus
    nextNeighbourNumber = nextNeighbourNumber + 1;

    For[ii = 1, ii < Length[ConvexHull], ii++,
    |For-Schleife |Länge
      If[ConvexHull[[ii]][CellI] == StartAtThisJPoint[CellI] + 1 ||
      |wenn
        ConvexHull[[ii]][CellI] == StartAtThisJPoint[CellI] - 1 ||
        ConvexHull[[ii]][CellI] == StartAtThisJPoint[CellI] &&
        ConvexHull[[ii]][CellI] >= StartAtThisJPoint[CellJ],
        AppendTo[SaftyList, ConvexHull[[ii]]];
      |hänge an bei
    ];
  ];

```

```

        Länge an bei
    ];
];
];
];

Clear[SafetyList];
| Lösche
Return[SafetyKeysList];
| gib zurück
];

SafetyListI[Start_, CheckLastPointI_, proportionI_,
CheckCellJForI_, ConvexHull_, distanceI_, NextIDir_] :=
Module[{SafetyList = {}, SafetyKeys = <>, SafetyKeysList = {}, propI,
| Modul
lastDirI, lastdistanceI, PotNextIDir, tempI, distanceNextPotPointI ,
nextNeighbourNumber, StartAtThisIPoint},

If[Length[SafetyKeysList] ≠ 0,
| ... | Länge
Clear[SafetyKeysList];
| Lösche
];
StartAtThisIPoint = CheckLastPointI;
nextNeighbourNumber = CheckLastPointI[NeighbourI] + 1;

propI = proportionI;
lastDirI = NextIDir;
lastdistanceI = distanceI;

For[kk = 1, kk ≤ Length[ConvexHull], kk++,
| For-Schleife | Länge

If[ConvexHull[[kk]][CellJ] == CheckLastPointI[CellJ] + 1 ||
| wenn
ConvexHull[[kk]][CellJ] == CheckLastPointI[CellJ] - 1 &&
ConvexHull[[kk]][CellI] ≥ CheckLastPointI[CellI],

AppendTo[SafetyList, ConvexHull[[kk]]];
| hänge an bei
];
];

Print["SafetyList I = ", SafetyList];
| gib aus | imaginäre Einheit I
For[uu = 1, uu ≤ Length[SafetyList], uu++,
| For-Schleife | Länge
(*Print["Start At This JPoint = ", StartAtThisJPoint];*)
| gib aus
If[SafetyList[[uu]][CoordI] ≠ Start[CoordI] &&
| wenn
SafetyList[[uu]][CoordI] ≥ StartAtThisIPoint[CoordI] &&
SafetyList[[uu]][CoordJ] ≥ StartAtThisIPoint[CoordJ] ||

```

```

SaftyList[[uu]][CoordJ] ≤ StartAtThisIPoint[CoordJ] ,  

PotNextIDir = {StartAtThisIPoint[CoordI] - SaftyList[[uu]][CoordI],  

StartAtThisIPoint[CoordJ] - SaftyList[[uu]][CoordJ]};  

distanceNextPotPointI = Sqrt[Abs[PotNextIDir[[1]]^2 + PotNextIDir[[2]]^2]];  

Qua... Absolutwert  

];  

If[SaftyList[[uu]][CoordI] ≠ Start[CoordI] &&  

| wenn  

  SaftyList[[uu]][CoordI] ≠ StartAtThisIPoint[CoordI] &&  

  SaftyList[[uu]][CoordI] ≥ StartAtThisIPoint[CoordI] &&  

  SaftyList[[uu]][CoordJ] ≤ StartAtThisIPoint[CoordJ] + propI + 0.3 &&  

  SaftyList[[uu]][CoordJ] ≥ StartAtThisIPoint[CoordJ] + propI - 0.3 &&  

  distanceNextPotPointI ≤ lastdistanceI + 0.5 &&  

  SaftyList[[uu]][CoordI] ≥ StartAtThisIPoint[CoordI] + lastdistanceI / 3  

  (**(3/4)* ) && MemberQ[CheckPoints, SaftyList[[uu]]] == False,  

  | enthalten? | falsch  

Print["SaftyListPoint I = ", SaftyList[[uu]]];  

| gib aus | imaginäre Einheit I  

AppendTo[CheckPoints, SaftyList[[uu]]];  

|hänge an bei  

AssociateTo[SaftyKeys,  

| assoziiere mit  

{ SaftyList[[uu]], NeighbourJ → 1 , NeighbourI → nextNeighbourNumber}];  

AppendTo[SaftyKeysList, SaftyKeys];  

|hänge an bei  

lastDirI = {StartAtThisIPoint [CoordI] - SaftyList[[uu]] [CoordI],  

StartAtThisIPoint [CoordJ] - SaftyList[[uu]] [CoordJ]};  

lastdistanceI = Sqrt[Abs[lastDirI [[1]]^2 + lastDirI [[2]]^2]];  

Qua... Absolutwert  

propI = SaftyList[[uu]][CoordJ] - StartAtThisIPoint[CoordJ];  

StartAtThisIPoint = SaftyList[[uu]];  

nextNeighbourNumber = nextNeighbourNumber + 1;  

AppendTo[SaftyKeysList, CompleteJGrid[StartAtThisIPoint,  

|hänge an bei  

  ConvexHull, lastdistanceI, propI, Start, 2 , nextNeighbourNumber - 1]];  

For[ii = 1, ii ≤ Length[ConvexHull], ii++,  

| For-Schleife | Länge  

If[ConvexHull[[ii]][CellJ] == StartAtThisIPoint[CellJ] + 1 ||  

| wenn  

  ConvexHull[[ii]][CellJ] == StartAtThisIPoint[CellJ] - 1 ||  

  ConvexHull[[ii]][CellJ] == StartAtThisIPoint[CellJ] &&  

  ConvexHull[[ii]][CellJ] ≥ StartAtThisIPoint[CellJ],  

AppendTo[SaftyList, ConvexHull[[ii]]];  

|hänge an bei  

];
];
];
];

```

```

Clear[SaftyList];
[lösche
Return[SaftyKeysList];
[gib zurück
];

CompleteJGrid[StartPointI_, ConvexHull_, StartDistanceJ_, proportionJ_,
Start_, NeighbourNumberJ_, aI_] := Module[{PossiblePointsListJ = {}, 
[Modul
SortedPointsKeys = <||>, SaftyPossiblePointsListJ = {}, propJ, StartPointForJGrid ,
distanceJ , NextNeighbourNumbrtJ, distanceNextPotGridPointJ , tempj,
NextPointJDir, PotNextJDir , NextJDir, CheckPointJ, CheckCellForJ},

propJ = proportionJ;
StartPointForJGrid = StartPointI;
distanceJ = StartDistanceJ;
NextNeighbourNumbrtJ = NeighbourNumberJ;

Print["proportionJ Complete Grid = ", proportionJ];
[gib aus [Gitter

For[aa = 1, aa < Length[ConvexHull], aa++,
[For-Schleife [Länge
If[ConvexHull[[aa]][CellI] == StartPointForJGrid[CellI] + 1 ||
[wenn
ConvexHull[[aa]][CellI] == StartPointForJGrid[CellI] - 1 ||
ConvexHull[[aa]][CellI] == StartPointForJGrid[CellI] && ConvexHull[[aa]][
CellJ] < splits && MemberQ[CheckPoints, ConvexHull[[aa]]] == False,
[enthalten? [falsch
AppendTo[PossiblePointsListJ, ConvexHull[[aa]]];
[hänge an bei
];
];
];

Print["PossiblePointList = ", PossiblePointsListJ];
[gib aus

For[pp = 1, pp < Length[PossiblePointsListJ], pp++,
[For-Schleife [Länge
If[PossiblePointsListJ[[pp]][CoordJ] != StartPointForJGrid[CoordJ] &&
[wenn
PossiblePointsListJ[[pp]][CoordJ] >= StartPointForJGrid[CoordJ] &&
PossiblePointsListJ[[pp]][CoordI] <= StartPointForJGrid[CoordI] ||
PossiblePointsListJ[[pp]][CoordI] >= StartPointForJGrid[CoordI],
PotNextJDir = {PossiblePointsListJ[[pp]][CoordJ] - StartPointForJGrid[CoordJ],
PossiblePointsListJ[[pp]][CoordI] - StartPointForJGrid[CoordI]};
distanceNextPotGridPointJ = Sqrt[Abs[PotNextJDir[[1]]^2 + PotNextJDir[[2]]^2]];
[Qua...[Absolutwert
];
If[PossiblePointsListJ[[pp]][CoordJ] != StartPointForJGrid[CoordJ] &&
[wenn
PossiblePointsListJ[[pp]][CoordJ] != Start[CoordJ] &&

```

```

PossiblePointsListJ[[pp]][CoordJ] ≥ StartPointForJGrid[CoordJ] &&
PossiblePointsListJ[[pp]][CoordI] ≤ StartPointForJGrid[CoordI] + propJ + 0.3 &&
PossiblePointsListJ[[pp]][CoordI] ≥
  StartPointForJGrid[CoordI] + propJ - (*1.0*) 3.0 &&
PossiblePointsListJ[[pp]][CoordJ] ≥ StartPointForJGrid[CoordJ] + distanceJ / 3 &&
distanceNextPotGridPointJ ≤ (distanceJ + 0.9) &&
MemberQ[CheckPoints, PossiblePointsListJ[[pp]]] == False,
  [enthalten? falsch]
Print["[PossiblePointsListJ[[pp]]] = ", PossiblePointsListJ[[pp]]];
  [gib aus]

NextJDir = { PossiblePointsListJ[[pp]][CoordJ] - StartPointForJGrid[CoordJ],
  PossiblePointsListJ[[pp]][CoordI] - StartPointForJGrid[CoordI]};
distanceJ = Sqrt[Abs[NextJDir[[1]]^2 + NextJDir[[2]]^2]];
  [Qua... Absolutwert]

propJ = PossiblePointsListJ[[pp]][CoordI] - StartPointForJGrid[CoordI];

StartPointForJGrid = PossiblePointsListJ[[pp]];

AppendTo[CheckPoints, PossiblePointsListJ[[pp]]];
  [hänge an bei]

AssociateTo[SortedPointsKeys, {PossiblePointsListJ[[pp]],
  [assoziere mit
    NeighbourJ → NextNeighbourNumbrtJ, NeighbourI → aI}];
AppendTo[SaftyPossiblePointsListJ, SortedPointsKeys];
  [hänge an bei]

AssociateTo[StartPointForJGrid,
  [assoziere mit
    {NeighbourJ → NextNeighbourNumbrtJ, NeighbourI → aI}];
NextNeighbourNumbrtJ = NextNeighbourNumbrtJ + 1;
];

];

CheckPointJ = StartPointForJGrid;
CheckCellForJ = StartPointForJGrid[CellJ];

AppendTo[SaftyPossiblePointsListJ, SaftyListJ[Start, StartPointForJGrid,
  [hänge an bei
    propJ, StartPointForJGrid[CellJ], ConvexHull, distanceJ, NextJDir]];

Clear[PossiblePointsListJ];
  [lösche

Print["Checklist = ", CheckPoints];
  [gib aus

Print["NEXT"];
  [gib aus

Return[SaftyPossiblePointsListJ];
  [gib zurück

];

CreateSyntheticPointsForFurtherChecks[] := Module[{},
  [Modul

```

];

## Modulübersicht Sortierungsalgorithmus

Module	Parameter	Lokale Variablen	Funktion
FindMinMax	Pointlist	Imin, imax, jmin, jmax, iSplits, jSplits, iDistance, jDistance	<ul style="list-style-type: none"> <li>Die minimas und maximas der i und j-Werte der Koordinaten werden gesucht, um den „Rahmen“ des Gitters um das Schachbrett festzulegen</li> <li>In den ConstantArrays JSplits und ISplits werden die Zellen des Gitters gespeichert.</li> <li>Diese werden über die Distanz der jeweilien Minimalwerte und Maximalwerte geteilt durch die gewünschte Anzahl an Zellen geteilt.</li> </ul>
SortPointList	iSplits, jSplits, Pointlist	pi,pj	<ul style="list-style-type: none"> <li>Die Eckpunkte werden zunächst der Größe nach nach ihren i-Werten Sortiert. Die sortierte Liste wird dann durchgezählt, so dass jeder Punkt seinen Indexwert in I-Richtung bekommt</li> <li>Danach werden die Eckpunkte der Größe nach nach ihren J-Werten sortiert und bekommen hier ebenfalls einen Index zugeordnet</li> <li>(Diese Sortierung ist nach jetzigem Stand des Algorithmus vlt nicht mehr zwingend notwendig)</li> </ul>
GoThroughConvex Hulls	iSplits, jSplits, pj	ConvexHull	<ul style="list-style-type: none"> <li>Nun wird herausgefiltert, welcher Punkt in welche Zelle des erstellten Gitters gehört, somit wird eine grobe Vorsortierung der Punkte für den weiteren Verlauf vorgenommen.</li> <li>In einer For-Schleife welche alle iSplits durchzählt wird die Funktion FindPointsInConvexHull bei jedem Durchgang aufgerufen welche eine Liste mit Associations in die Liste ConvexHull hinzufügt.</li> <li>Der Funktion werden die momentanen iSplits der Durchzählung übergeben und alle jSplits. Des Weiteren wird die nach J sortierte Punkteliste übergeben</li> </ul>
FindPointsInConvexHull	iSplits[[1,i]], iSplits[[1,i+1]], jSplits, pj	ConvexHullCell={}, ConvexHullList,ConvexHullCellKeys = <  >	<ul style="list-style-type: none"> <li>Eine Liste namens ConvexHullCell und eine Association nachmes ConvexHullKeys wird angelegt</li> <li>Zwei For-Schleifen werden gestartet. Die erste läuft durch alle Jsplits, die zweite geht alle Punkte von pj durch.</li> <li>Innerhalb der For-Schleife wird dann überprüft, welche Punkte aus pj sich innerhalb der übergebenen Isplits und den dazugehörigen jSplits befinden.</li> <li>Die Koordinaten, die Indizes und die Zellenbezeichnung werden dann in Keys in die Association ConvexHullCellKeys gespeichert und er Liste ConvexHullCell angehängt. Diese Liste wird dann und die Liste ConvexHull angehängt</li> <li>Wiederholung des Vorganges mit neuen iSplits.</li> </ul>

FindStartVectors	ConvexHull	<pre>StartPointCloud Keys={}&gt;, VecI, VecJ, countI, countJ, Start, nextI, nextJ,</pre>	<ul style="list-style-type: none"> <li>• Die Punkte der Zellen (<math>i = 1, j = \text{All}</math>) und (<math>i = \text{all}, j = 1</math>) werden in eine neue Liste namens StartPointCloud gespeichert.</li> <li>• Die Liste wird zweimal durchlaufen <ul style="list-style-type: none"> <li>• Alle Punkte welche sich in den Zellen <math>j = 1</math> und <math>i = \text{all}</math> aufhalten. Aus ihnen wird der geringste <math>i</math>-Wert ermittelt (VecI)</li> <li>• Alle Punkte welche sich in den Zellen <math>j = \text{all}</math> und <math>i = 1</math> aufhalten. Aus ihnen wird der geringste <math>j</math>-Wert ermittelt (VecJ)</li> <li>• Die Punkte mit den geringsten Werten werden in VecI und VecJ gespeichert.</li> </ul> </li> <li>• Jetzt wird die Liste nochmals zweimal durchgegangen. <ul style="list-style-type: none"> <li>• Alle Punkte welche sich in den Zellen <math>j = 1</math> und <math>i = \text{all}</math> aufhalten werden durchgegangen. Aus ihnen wird derjenige Wert ermittelt, welcher einen Wert für <math>j</math> besitzt der kleiner ist als der momentan <math>j</math>-Wert von VecI und dessen <math>i</math>-Wert kleiner ist als der <math>i</math>-Wert von VecI plus einem Offset. Dieser Wert ist das neue VecI</li> <li>• Alle Punkte welche sich in den Zellen <math>j = \text{all}</math> und <math>i = 1</math> aufhalten. Aus ihnen wird derjenige Wert ermittelt, welcher einen Wert für <math>i</math> besitzt der kleiner ist als der momentan <math>i</math>-Wert von VecJ und dessen <math>j</math>-Wert kleiner ist als der <math>j</math>-Wert von VecI plus einem Offset. Dieser Wert ist das neue VecJ</li> <li>• VecI und VecJ ergeben den gleichen Punkt und somit ist der Startwert gesetzt.</li> <li>• Nun sollen die ersten Punkte in <math>i</math>- und <math>j</math>-Richtung vom Startpunkt aus gefunden werden.</li> <li>• Es wird ein nextI und ein nextJ definiert, dessen Koordinaten sehr groß angefangen</li> <li>• Es wird wieder die StartPointListe zweimal durchlaufen <ul style="list-style-type: none"> <li>• Es werden Punkte gesucht, welche sich in der selben Zelle <math>i</math> wie der Startpunkt befinden und auch die Zellen <math>+1</math> und <math>-1</math> drum herum. Sollte es ein Punkt geben, der kleiner ist als das momentane nextI und größer als der Startpunkt, jedoch nicht gleich dem Startpunkt. So nimmt nextI dessen Wert an.</li> <li>• Danach muss geprüft werden, ob das potentielle nextI auch wirklich das richtige nextI ist. Hierzu wird eine neue For-Schleife gestartet, welche wieder die StartPointCloud durchgeht und überprüft ob es einen Punkt gibt dessen <math>j</math>-Koordinatenabstand zum Startpunkt kleiner ist also der <math>j</math>-Koordinatenabstand des momentanen nextI zum Startpunkt und ob dessen <math>i</math>-Koordinatenabstand zum Startpunkt kleiner ist als der momentane <math>i</math>-Koordinatenabstand von nextI zum Startpunkt.</li> </ul> </li> </ul> </li> </ul>
------------------	------------	--	---

			<ul style="list-style-type: none"> <li>• Ist dies der Fall so wird dieser Punkt zum neuen nextJ.</li> </ul>
CreatePossiblePoint – ListsIAndJ	nextJ, nextI, Start, ConvexHull	IIList={ }, JList={ }, IDir, JDir, distance, cache, PotNextI, PotNextJ	<ul style="list-style-type: none"> <li>• Mit dem potentiellen nextJ wird ebenso verfahren.</li> <li>• IDir und JDir sind die Richtungsvektoren vom Startpunkt aus in beide Kantenrichtungen des Schachbretts.</li> <li>• Danach werden die ersten beiden Spalten in I- und J-Richtung jeweils durchlaufen, und in IList und JList gespeichert.</li> <li>• Diese Listen enthalten weitere potentielle Punkte entlang der gesuchten Kante.</li> <li>• Die Kanten können natürlich durch die perspektivische Verzerrung mancher Bilder auch noch weiter in die Zellen hineinragen. Hierum kümmert sich dann im späteren Algorithmus die SaftyIList[] und SaftyJList[] Funktionen</li> </ul>
FindNeighbours	IIList, JList , Start, nextI, nextJ, ConvexHull	SortedPointsKeys = <  >, Sortedpoints = {}, proportionI, proportionJ, Itemp, itemp, PotNextIDir, distanceNextPotPointJ, PotNextIDir, distanceNextPotPointI, PotNextIDir, NeighbourNumberJ, NeighbourNumberI, distanceJ, distanceI, NextIDir, NextIDir, StartPropJForFirstCompleGrids	<ul style="list-style-type: none"> <li>• StartPoint und NextPointI und NextPointJ werden die Keys NeighbourI und NeighbourJ gegeben mit StartPoint(NeighbourJ → 1, NeighbourI → 1), NextPointI(NeighbourJ → 1, NeighbourI → 2) und NextPointJ(NeighbourJ → 2, NeighbourI → 1).</li> <li>• Diese drei bereits bekannten Punkte werden dann auch in eine angelegte CheckPointList gespeichert, diese wird für das spätere Prüfen von weiteren Punkten benötigt.</li> <li>• Nun wird zunächst in einer For-Schleife die Punkte von StartPoint und NextPointJ aus gesucht. <ul style="list-style-type: none"> <li>• Anmerkung: Für die Punkte in I-Richtung des Schachbretts wird das selbe Verfahren angewandt.</li> <li>• Benötigt wird die Distanz zwischen dem momentanen StartPointJ, welcher nach jedem Durchlauf der Schleife den Wert des momentanen NextPointJ bekommt und einem momentanen NextPointI, welcher nach jedem Durchlauf der Schleife den Wert des gerade neu gefundenen nächsten Punktes bekommt.</li> <li>• Die Schleife selbst durchläuft alle Punkte, welche in der für die Richtung entsprechenden Richtung Liste sind. In diesem Fall die JList</li> <li>• Es wird außerdem bei der Suche den nächsten Punktes in j-Richtung eine Distanz namens proportion berechnet, welcher die Distanz i zwischen StartPoint und NextPoint beinhaltet.</li> <li>• Innerhalb der durchlaufenden Liste wird derjenige Punkt gesucht welcher zum NextPointJ den geringsten Abstand in J Richtung hat und dessen Abstand in I-Richtung &lt;= der i-Koordinate des NextPointJ + proportion+noch einen Puffer ist und &gt;= der i-Koordinate des NextPointJ – proportion+noch einen Puffer.</li> </ul> </li> </ul>

		<ul style="list-style-type: none"> <li>• Ist der nächste Punkt gefunden, so wird dieser der SortedPointsList und der der CheckPointsList übergeben mit den passendem Neighbour und NeighbourJ associationKey.</li> <li>• Des Weiteren bekommt für den nächsten Schleifendurchlauf StartPointJ die Werte von NextPointJ und NextPointJ in wird der neu gefundenen Punkt aus der JListe gespeichert.</li> <li>• Im Anschluss werden noch in           <pre>AppendTo[SortedPoints, SaftyList[[Start, CheckPointJ, proportionY, CheckCellForJ, ConvexHull, distanceI]]; AppendTo[SortedPoints, CompleteJGrid[ nextI, ConvexHull, StartDistanceJ, StartProportionJ, Start, {p, all}]]</pre>           Weitere Punkte zur SortedList in J-Richtung hinzugefügt, bei ersterem nur in bestimmten Fällen. Mehr zu den Funktionen folgt.</li> <li>• Nicht zu vergessen: selbiges wie Oben wird auch mit den Punkten in I-Richtung vollzogen, bis auf die CompleteGrid Funktion</li> </ul>	
SaftyList	<pre>Start, CheckLastPointJ , proportionJ, LastPointsCell, ConvexHull, NextJDir</pre>	<ul style="list-style-type: none"> <li>• SaftyList = {}, SaftyKeys = &lt;&gt;, SaftyKeysList = {}, propJ, lastDir, lastdistanceJ</li> <li>• Der Funktion werden die Parameter CheckLastPointJ und LastPointCell mitgegeben. Diese stammen aus der Funktion FindNeighbours und es handelt sich um den letzten Punkt der innerhalb der JListe ermittelt wurde und dessen i-Zelle in welcher sich dieser befindet.</li> <li>• Da die I- bzw die JListe in jede Richtung nur die Punkte der ersten beiden Zellen beinhaltet, kann es bei einem rotierten Schachbrett sein, dass sich noch weitere Punkte in Zellen weiter oben/unten befinden</li> <li>• Die Funktion SaftyList, erstellt eine Liste aus möglichen weiteren Punkten, indem sie die in diesem Falle I-Zelle des letzten Punktes nimmt und diese so wie die unter und oberhalb dieser Zelle und alle deren J-Zellen aufwärts auf einen möglichen nächsten Punkt untersucht. → Dies geschieht nach dem selben Verfahren wie in FindNeighbours.</li> <li>• Sollte es noch einen geben wird dieser ebenfalls der CheckPointList und der SortedPointsList zugewiesen, ansonsten passiert nichts.</li> </ul>	
CompleteJGrid	<pre>StartPointI, ConvexHull, StartDistanceJ, proportionJ, Start,</pre>	<ul style="list-style-type: none"> <li>PossiblePointsList = {}, SortedPointsKeys = &lt;&gt;, SaftyPossiblePointsListJ = {}, propJ, StartPointForJGrid, distanceJ,</li> <li>Nachdem die äußersten Punkte der linken und unteren Kante des Schachbretts gefunden wurden, muss nun das restliche Grid des Schachbretts detektiert und mit den richtigen Neighbour und NeighbourJ werten versehen werden.</li> <li>Jeder Punkt der in I-Richtung als „Rahmenpunkt“ detektiert wurde, wird einmal als Startpunkt gesetzt, von ihm aus wird dann in einem sehr ähnlichem Verfahren wie schon zuvor der nächste Punkt in J-Richtung gesucht und wenn nötig tritt auch hier</li> </ul>	

	NeighbourNumberJ, al	NextNeighbourNumberJ, distanceNextPointGridPointJ, tempJ, NextPointJDir, NextJDir, CheckPointJ, CheckCellForJ	nochmal die SaftyList Funktion in kraft um auch wirklich alle Punkte jeder Reihe ausfindig zu machen
--	----------------------	---	--

## **Eidesstattliche Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht auch nicht auszugsweise, bereits für andere Prüfungen angefertigt wurde.

Furtwangen, den 29.Mai 2018

# Abbildungsverzeichnis

2.1	Lochkameramodell . . . . .	7
2.2	Lochkameramodell Querschnitt . . . . .	8
2.3	Koordinatentransformation . . . . .	9
2.4	Koordinatensysteme im Überblick . . . . .	11
3.1	Abbildungsvorschrift in Homographien . . . . .	15
3.2	Veranschaulichung der freien Variablen $\gamma$ . . . . .	15
3.3	Epipolar Geometrie . . . . .	17
3.4	Abbildungsvorschrift . . . . .	17
4.1	Ablaufdiagramm . . . . .	23
4.2	Synthetisches Beispiel Top-Down-Ansicht . . . . .	24
4.3	Simulierte Abbildung eines Quaders auf zwei Kameras . . . . .	24
4.4	Koordinatensysteme von $C$ und $C'$ . . . . .	24
4.5	Bestimmung extrinsischer Kameraparameter Lösung eins und zwei . . . . .	28
4.6	Bestimmung extrinsischer Kameraparameter Lösung eins und zwei . . . . .	28
4.7	Einfache Triangulation . . . . .	29
4.8	Skalierung der Rekonstruierten Szene . . . . .	30
4.9	Programmplot der rekonstruierten Szene . . . . .	31
4.10	Aufbau CMOS Sensor . . . . .	32
4.11	Nachbarschaftsoperation auf Sensoren . . . . .	33
4.12	Sensorkoordinatensystem bei Kameras gleicher Auflösung . . . . .	33
4.13	Sensorkoordinatensystem bei Kameras unterschiedlicher Auflösung . . . . .	33
4.14	Abbildungen bei $K' = K$ . . . . .	35
4.15	Abbildung bei $K'_1$ . . . . .	35
4.16	Abbildungen bei $K'_2$ . . . . .	35
4.17	Abbildungen bei $K'_3$ . . . . .	35
4.18	Rekonstruierte Szene bei unterschiedlichen Kameraauflösungen . . . . .	36
5.1	Ablaufdiagramm für die reelle Rekonstruktion . . . . .	37
5.2	Stereoaufbau im Überblick . . . . .	38
5.3	Aufnahme von Kamera $C$ . . . . .	39
5.4	Aufnahme von Kamera $C'$ . . . . .	39
5.5	Epipolarlinien $C$ ohne Singularität . . . . .	41
5.6	Epipolarlinien $C'$ ohne Singularität . . . . .	41
5.7	Epipolarlinien in $C$ aus singulärer Fundamentalmatrix . . . . .	42
5.8	Epipolarlinien in $C'$ aus singulärer Fundamentalmatrix . . . . .	42
5.9	Epipolarlinien in $C$ aus singulärer denormalisierter Fundamentalmatrix . . . . .	43
5.10	Epipolarlinien in $C$ aus singulärer denormalisierter Fundamentalmatrix . . . . .	43
5.11	Windschiefe Geraden . . . . .	44
5.12	Epipolare Bedingung wird nicht erfüllt . . . . .	44
5.13	Problemstellung für die Triangulation im reellen Beispiel . . . . .	44
5.14	Sampson Approximation aufstellen der Kostenfunktion . . . . .	45
5.15	Rekonstruierte Szene 3D . . . . .	49
5.16	Rekonstruierte Szene 2D . . . . .	49
5.17	Vier Lösungen für $T$ bei gleicher Kameraauflösung . . . . .	50
5.18	Vier Lösungen für $T$ bei $K'$ mit 5:2 . . . . .	50

5.19	Vier Lösungen für $T$ bei $K'$ mit 1:2 . . . . .	51
5.20	Vier Lösungen für $T$ bei $K'$ mit 1.2 : 2.3 . . . . .	51
5.21	Rekonstruierte Szene bei $K' = 5:2$ . . . . .	51
5.22	Rekonstruierte Szene bei $K' = 1:2$ . . . . .	52
5.23	Rekonstruierte Szene bei $K' = 1.2:2.3$ . . . . .	52
6.1	Beispiel eines Rektifizierten Bildes mit Scanlinien . . . . .	53
6.2	Arbeitsprozess der Szenenrekonstruktion mit Rektifizierung . . . . .	54
6.3	Entstehung einer Disparitätskarte . . . . .	55
6.4	virtuelle Aufnahme eines Quaders für die Rektifizierung . . . . .	60
6.5	Epipolarlinien und Epipole vor der Rektifizierung . . . . .	61
6.6	$H_p$ und $H'_p$ Transformation . . . . .	61
6.7	$H_p$ und $H'_p$ Transformation mit Epipolarlinien . . . . .	61
6.8	$H_rH_p$ und $H'_rH'_p$ Transformation . . . . .	63
6.9	$H_rH_p$ und $H'_rH'_p$ Transformation mit Epipolarlinien . . . . .	63
6.10	Wiederherstellung der Orthogonalität . . . . .	64
6.11	$H_sH_rH_p$ und $H'_sH'_rH'_p$ Transformation . . . . .	65
6.12	$H_sH_rH_p$ und $H'_sH'_rH'_p$ Transformation mit Epipolarlinien . . . . .	65
6.13	virtuelle Aufnahme mit unterschiedlichen Auflösung für die Rektifizierung . . . . .	66
6.14	Transformation $H_p$ und $H'_p$ angewandt auf Bilder unterschiedlicher Auflösungen . . . . .	66
6.15	Transformation $H_rH_p$ und $H'_rH'_p$ angewandt auf Bilder unterschiedlicher Auflösungen . . . . .	66
6.16	Transformation $H_sH_rH_p$ und $H'_sH'_rH'_p$ bei unterschiedlicher Auflösungen . . . . .	67
6.17	Transformation $H_sH_rH_p$ und $H'_sH'_rH'_p$ mit Epipolarlinien . . . . .	67
6.18	Rektifizierung mit $\zeta'_x = 2.3$ und $\zeta'_y = 3.2$ . . . . .	67
6.19	Rektifizierung mit $\zeta'_x = 2.3$ und $\zeta'_y = 3.2$ . . . . .	67
7.1	Funktionsübersicht des Sorieralgorithmus . . . . .	68
7.2	Startpunktsuche in Schachbrettpunkten . . . . .	69
7.3	Finden der Startvektoren in Schachbrettpunkten . . . . .	71
7.4	Überprüfung des gefundenen $NextI$ . . . . .	72
7.5	Suche nach $NextJ$ . . . . .	73
7.6	Sicherheitsfunktion <i>SaftyList</i> . . . . .	75
7.7	Korrespondezsuche mit Sortierungsalgorithmus. Schachbrett auf linker Seite . . . . .	76
7.8	Korrespondezsuche mit Sortierungsalgorithmus. Schachbrett auf rechter Seite . . . . .	76
7.9	Ergebnis des Sortierungsalgorithmus. Schachbrett auf linker Seite . . . . .	76
7.10	Ergebnis des Sortierungsalgorithmus. Schachbrett auf linker Seite . . . . .	76
7.11	Schachbrett mit Tonnenverzeichnung . . . . .	77
7.12	Sortierte Punkte eines Schachbretts mit Tonnenverzeichnung . . . . .	77
7.13	perspektivisch stark verzerrtes Schachbrett . . . . .	77
7.14	Sortierte Punkte eines perspektivisch verzerrten Schachbretts . . . . .	77
7.15	Schachbrett mit Kissenverzeichnung . . . . .	77
7.16	Sortierte Punkte eines Schachbretts mit Kissenverzeichnung . . . . .	77
7.17	Perspektivisch verzerrtes Schachbrett mit Tonnenverzeichnung . . . . .	78
7.18	Sortierte Punkte eines perspektivisch verzerrten Schachbretts mit Tonnenverzeichnung . . . . .	78
7.19	Bild eines Tonnenförmig verzeichnetem leicht perspektivisch verzerrtem Schachbretts . . . . .	78
7.20	Algorithmisch detektierte Linie der dritten i-Reihe . . . . .	78

# Literaturverzeichnis

- [1] Lutz Priese. *Computer Vision, Einführung in die Verarbeitung und Analyse digitaler Bilder.* Springer-Verlag Berlin Heidelberg, 2014.
- [2] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in computer vision.* Cambridge, 2004, Second Edition.
- [3] Ferid Bajrmovic. *Self- Calibration of Multi- Camera Systems.* Logos Verlag Berlin GmbH, 2010.
- [4] Tomas Pajdla. *Elements of Geometry for Computer Vision,* 2013, überarbeitet am 27.2.2017. "<http://cmp.felk.cvut.cz/pajdla/gvg/GVG-2017-Lecture.pdf>".
- [5] Zhengyou Zhang Gang Xu. *Epipolar Geometry in Stereo, Motion and Object Recognition: A Unified Approach.* Springer-Science and Business Media, 1996.
- [6] Christian Heipke. *Photogrammetrie und Fernerkundung.* 2017 Springer, 1. Auflage.
- [7] Ramalingam Tardif S.Gasparini J.Barreto R.Sturm, S. Camera models and fundamental concepts used in geometric computer vision. Mitsubishi Electric Research Laboratories, 2011.
- [8] Rolf Martin Ekbert Hering. *Photonik, Grundlagen, Technologien und Anwendung.* Springer-Verlag Berlin Heidelberg, 2006.
- [9] Dipl.-Ing. Martin Roser. *Modellbasierte und positionsgenaue Erkennung von Regentropfen in Bildfolgen zur Verbesserung von viedeoisierten Fahrerassistenzfunktionen.* 1986, 1994 Springer Basel AG, KIT Scientific Publishing.
- [10] Christoph Stiller Thao Dang, Christian Hoffmann. Continuous stereo self-calibration by camera parameter tracking.
- [11] Branislav Micusik. *Two-View Geometry of Omnidirectional Cameras.* Dissertation, Technische Universität Prag.
- [12] Zhengyou Zhang. *Epipolar Geometry,* pages 247–258. Springer US, Boston, MA, 2014.
- [13] Zhengyou Zhang. Determining the epipolar geometry and its uncertainty: A review. Received July 16, 1996; Accepted February 13, 1997.
- [14] K.A. Semendjajew I.N. Bronstein. *Taschenbuch der Mathematik,* volume 5. Auflage. First Published 1962.
- [15] Sascha jockel. *3-dimensionale Rekonstruktion einer Tischszene aus monokularen Handkamera-Bildsequenzen im Kontext automotiver Serviceroboter.* Dissertation, Fakultät für Mathematik, Informatik und Naturwissenschaften, Universität Hamburg.
- [16] Richard I. Hartley. In defence of the 8-point algorithm. GE-Corporate Research and Development, Schenectady, NY, 12309.
- [17] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [18] Norbert Köckler Hans Rudolf Schwarz. *Numerische Mathematik.* 2011, Springer Verlag, 8. Auflage.

- [19] Daniel Scholz. *Numerik interaktiv, Grundlagen verstehen, Modelle erforschen und Verfahren anwenden mit taramath*. 2016, Springer Verlag.
- [20] LongQuan. *Image Based Modeling*, volume 1. Auflage. Springer US, 2010.
- [21] Bernd KITT. *Effiziente Schätzung dichter Bewegungsvektorfelder*, volume Band 027. KIT Scientific Publishing, 2013.
- [22] Wolfram Research, Inc. Mathematica, Version 11.1.1. Champaign, IL, 2018.
- [23] Olaf Dössel. *Bildgebende Verfahren in der Medizin*, volume 2. Auflage. Springer-Verlag Berlin Heidelberg, 2016.
- [24] Emmanuel Habets Nicolas Paparoditis Xiaozhi Qu, Bahman Soheilian. Evaluation of sift and surf for vision based localization. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XLI-B3, 2016.
- [25] Antin van den Hengel Darren Gawey Wojciech Chojnacki, Michael J. Brooks. Revisiting hartley's normalized eight-point-algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume: 25, 2003.
- [26] Jarosław Bylina Anna Pyzara, Beata Bylina. The influence of a matrix condition number on iterative methods convergence. *Proceedings of the Federated Conference on Computer Science and Information Systems*, pp. 459–464, 2011.
- [27] James Demmel Dinesh Manocha. Algorithms for intersecting parametric and algebraic curves 1. *ACM Transactions on Graphics (TOG)*, Volume 13 Issue 1, Pages 73-100, 1994.
- [28] Jürgen Adamy Christian Voigt. *Formelsammlung der Matrizenrechnung*. Oldenburg Verlag München Wien, 2007.
- [29] Luca Irsara Andrea Fusiello. Euclidean epipolar rectification of uncalibrated images. Eurac researc, IT.
- [30] Carlos Villagrá Arnedor Antonio Javier Gallego Sanchez, Rafael Molina Carmona. *Scene reconstruction and geometrical rectification from stereo images*. Januar 2005, uploaded by Antonio Javier Gallego Sánchez on 21 May 2014, ResearchGate.
- [31] Charles Loop and Zhengyou Zhang. Computing rectifying homographies for stereo vision. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Vol.1, pages 125–131, June 23-25, 1999. Fort Collins, Colorado, USA, 1999 Errors corrected on June 6, 2001.
- [32] MathWorks. Mathworks documentation, rectify stereo images.
- [33] Richard I. Hartley. Euclidean reconstruction from uncalibrated views. G.E. CRD, Schenectady, NY, 12301.
- [34] Dongqing Li, editor. *Encyclopedia of Microfluidics and Nanofluidics*, pages 999–999. Springer US, Boston, MA, 2008.
- [35] William T. Vetterling Brian P. Flannery William H. Press, Saul A. Teukolsky. *Numerical Recipes in Fortran 77 The Art Of Scientific Computing*. Copyright Numerical Recipes Software 1986, 1992, 1997 All Rights Reserved., Reprinted with corrections 1997, Volume 1 of Fortran Numerical Recipes.
- [36] James Demmel Dinesh Manocha. Algorithms for interesting parametric and algebraic curves 1: Simple intersections. *ACM Transactions of Graphics*:73–100, 1994.