

H T  
W I  
G N



**Hochschule Konstanz**  
Fakultät Informatik  
Institut für Optische Systeme

Eingereicht von  
Lukas Hornung  
Lukas Luschin  
Moritz Schmidt  
Timmo Waller-Ehrat

# Teamprojekt

## Mehrbildkamerasytem zur räumlichen Detektion von Modellhubschraubern

# Extended Abstract

Thema: Mehrbildkamerasystem zur räumlichen Detektion von Modellhubschraubern

Teammitglieder: Lukas Hornung, Lukas Luschin, Moritz Schmidt, Timmo Waller-Ehrat

Betreuer: Hochschule für Technik, Wirtschaft und Gestaltung  
HTWG Konstanz, Institut für Optische Systeme  
Prof. Dr. Georg Umlauf, Prof. Dr Matthias O. Franz

Ziel des Projekts war das räumliche Detektieren eines Modellhubschraubers. Die Detektion soll unter Laborbedingungen stattfinden, das heißt, der Helikopter befindet sich vor einer weißen Wand. Zur Lokalisierung des Helikopters im Raum soll zunächst eine 3D-Punktwolke der Szene generiert und daraus mit Hilfe des Clustering-Algorithmus k-Means die Position des Helikopters bestimmt werden. Auch die Tiefe (Entfernung zur Kamera) des Helikopters soll ermittelt werden. Die Detektion soll mit Hilfe von zwei oder mehr Kameras des Herstellers Point Grey, die über eine FireWire-Schnittstelle mit einem Computer verbunden sind, stattfinden.

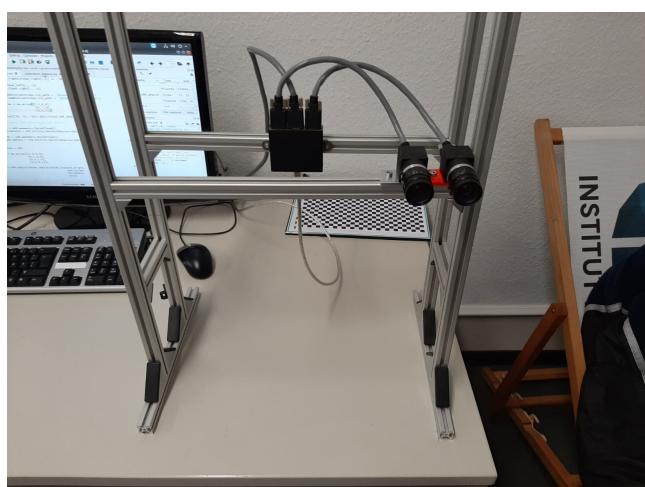
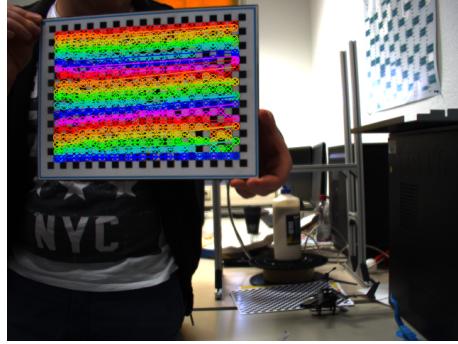
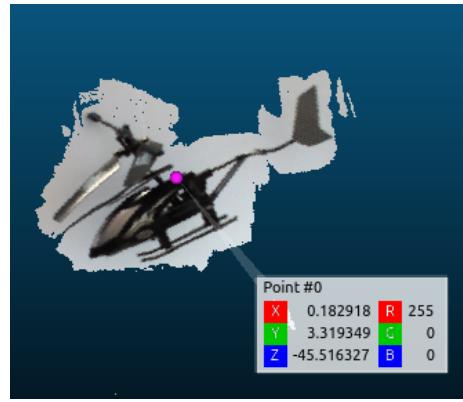


Abbildung 1: Kamera-System



(a) Kamera-Kalibrierung



(b) Helikopter-Punktewolke mit berechneter Raumposition

Abbildung 2: Kamera-Kalibrierung und Punktewolke

Mittels zwei Kameras, die auf einer Ebene angebracht sind (Stereonormalfall), kann der Mittelpunkt des Helikopters und dessen Abstand zur Kamera ermittelt werden. Zunächst müssen dafür die Kameras einzeln und anschließend zu einander. Das Kalibrieren erfolgt über ein Schachbrett-Muster. Sind die Kameras zueinander kalibriert, kann mittels eines Feature-Detektors eine Punktewolke der Szene generiert und der Mittelpunkt des Helikopters berechnet werden.

# Abstract

Ziel des Projekts war das räumliche Detektieren eines Modellhubschraubers. Die Detektion soll unter Laborbedingungen stattfinden, das heißt, der Helikopter befindet sich vor einer weißen Wand. Zur Lokalisierung des Helikopters im Raum soll zunächst eine 3D-Punktwolke der Szene generiert und daraus mit Hilfe des Clustering-Algorithmus k-Means die Position des Helikopters bestimmt werden. Auch die Tiefe (Entfernung zur Kamera) des Helikopters soll ermittelt werden. Die Detektion soll mit Hilfe von zwei oder mehr Kameras des Herstellers Point Grey, die über eine FireWire-Schnittstelle mit einem Computer verbunden sind, stattfinden.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	
1.1	Aufgabenstellung und Zielsetzung	.....
1.2	Motivation	.....
1.3	Aufbau	.....
<b>2</b>	<b>Projektplan</b>	
<b>3</b>	<b>Technologien</b>	
3.1	Software	.....
3.1.1	OpenCV	.....
3.1.2	scikit-learn	.....
3.1.3	Open3D	.....
3.1.4	PyCapture	.....
3.1.5	Spyder	.....
3.2	Hardware	.....
3.2.1	Kameras	.....
<b>4</b>	<b>Bildverarbeitung und Umsetzung</b>	
4.1	Kalibrierung	.....
4.2	Genauigkeitsabschätzung der Kalibrierung	.....
4.3	Stereokalibrierung	.....
4.3.1	Entzeichnung	.....
4.3.2	Rektifizierung	.....
4.4	Generierung einer Tiefenkarte	.....
4.5	Punktwolken-Generierung	.....
4.6	Lokalisierung des Helikopters	.....
4.6.1	k-Means	.....
<b>5</b>	<b>Experimente</b>	
5.1	Vereinfachte Kalibrierung	.....
5.2	Tiefe und Stereokalibrierung	.....
5.3	Fehlerfortpflanzung und Tiefenvalidierung	.....

## *INHALTSVERZEICHNIS*

**6 Probleme**

**7 Fazit**

# 1

## Einleitung

### 1.1. Aufgabenstellung und Zielsetzung

Im Rahmen dieses Teamprojekts stand die Entwicklung eines Mehrbildkamerasytems zur räumlichen Detektion eines Modellhubschraubers. Dies beinhaltet sowohl das Erkennen des Helikopters, als auch die Abstandsmessung von diesem. Dies sollte mit Hilfe von Bilderverarbeitungs- und Machine Learning-Techniken, sowie der Verwendung von zwei oder mehr Kameras umgesetzt werden.

Die Lernziele umfassten das Erlernen des Umgangs mit Kameras für die industrielle Bildverarbeitung, sowie ein Verständnis für die Grundlagen industrieller Signalverarbeitung zu schaffen. Zudem sollten grundlegende KI-Verfahren erlernt werden.

### 1.2. Motivation

*„Computer vision, or the ability of artificially intelligent systems to see like humans, has been a subject of increasing interest and rigorous research for decades now.“*

— Naveen Joshi[9]

Das maschinelle Sehen gewinnt in den letzten Jahren immer mehr an Popularität. Sei es in der Forschung oder z.B. in der Spieleentwicklung mittels Augmented Reality. Durch die steigende Relevanz in der Praxis wurde auch unser Interesse für dieses Themengebiet geweckt. Es ist spannend zu verstehen, wie komplex die Dinge, die für uns

### **1.3. Aufbau**

Menschen selbstverständlich erscheinen, wie zum Beispiel das räumliche Sehen, eigentlich sind. Ein weiterer Anreiz für das Projekt waren die verschiedenen angewandten Technologien. Wir alle interessieren und sehr für das Programmieren. Viel Erfahrung in der Programmiersprache Python hatte aber anfangs keines der Teammitglieder. Somit war das Erlernen dieser Sprache eine weitere Motivation.

Auch die zum Großteil verwendete und weit verbreitete Bibliothek für Computer Vision-Anwendungen in Python "OpenCV" hat das Interesse an das Projekt geweckt.

## **1.3. Aufbau**

Die Ausarbeitung des Teamprojekts besteht aus drei Teilen.

Anfangs wird kurz auf die angewandten Technologien eingegangen. Anschließend wird die eigentliche Umsetzung und das Vorgehen erläutert. Zuletzt wird auf die aufgetretenen Probleme eingegangen und ein Fazit gezogen.

# 2

## Projektplan

Das Arbeiten in einem Team, indem es keine Hierarchie geben soll, ist verbunden mit agilem Entwickeln. Wichtig für keine strenge Rolleneinteilungen ist die permanente Dokumentation und das verständliche Entwickeln, damit die Weiterentwicklung von allen Teammitgliedern zu jeder Zeit möglich ist. Die Kommunikation der Mitglieder und der Austausch von neu erlerntem Wissen funktionierten azyklisch. Terminierte Treffen wurden einberufen um unsere Fortschritte zu besprechen und neue Anforderungen des Projektbetreuers zu bekommen. Einen Zeitplan mit einigen Meilensteine, haben wir nach der ersten Einarbeitungsphase erstellt. Anhand von diesem können wir gut unsere Fortschritte und unsere Misserfolge darstellen. Die Anpassung der Zwischenschritte wird in Kapitel Probleme genauer dargestellt.

Die Abbildung ?? zeigt den genaueren Zeitplan, mit mehreren Arbeitspaketen, welche einen Zeiteinschätzung haben und an denen man erkennen kann welche Pakete von welchen Abhängen. Die Zwischenziele sind als Meilensteine aufgetragen. Das erfolgreiche Abschließen der Arbeitspakete und Meilensteine wurde bis zum Zwischentermin abgeschlossen. Die Einarbeitungsphase und die Lieferung der Hardware haben viel Zeit eingenommen, werden aber nicht weiter thematisiert. Bis zur Zwischenpräsentation sollten das wichtige Arbeitspaket der Kamerakalibrierung abgeschlossen sein. Des Weiteren sollte die Generierung der Punktewolke stattfinden

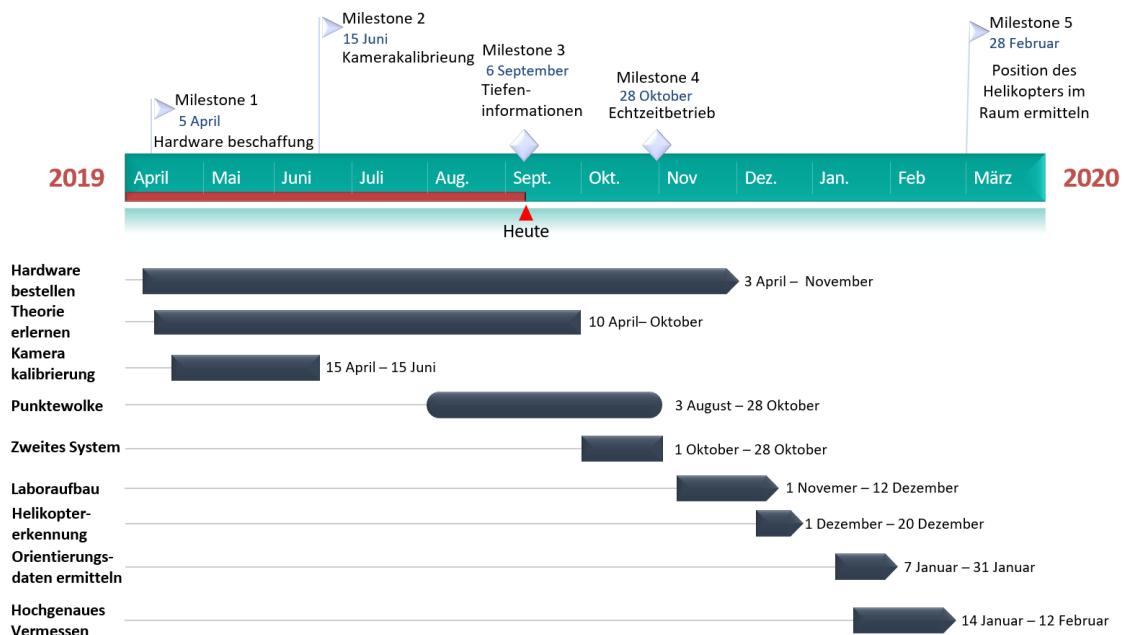


Abbildung 2.1: Projektplan Zwischenergebnis

In dem endgültigen Plan mussten einige Anpassungen getroffen werden. Dies ist in der Abbildung 2.2 rot markiert. Der nicht eingehaltene Meilenstein 4: Echtzeitbetrieb, wurde vorerst verschoben und dann als nicht relevant für die Aufgabenstellung und die Algorithmus Entwicklung beschlossen. Das Arbeitspaket Kamerakalibrierung hat sich sehr in die Länge gezogen und ist auch noch präsent bei dem Abschluss des Projektes. Sie führt immer wieder zu Ungenauigkeiten und wird im Kapitel Probleme behandelt. Das Arbeitspaket Orientierungsdaten ermitteln, wurde in der Art geändert, dass der Helikopter erkannt wird aber nicht in seiner Ausrichtung im Raum. Die Bearbeitung nach unserem Projektplan hat zu einem Ergebnis geführt, das in den nächsten Kapiteln dargestellt wird.

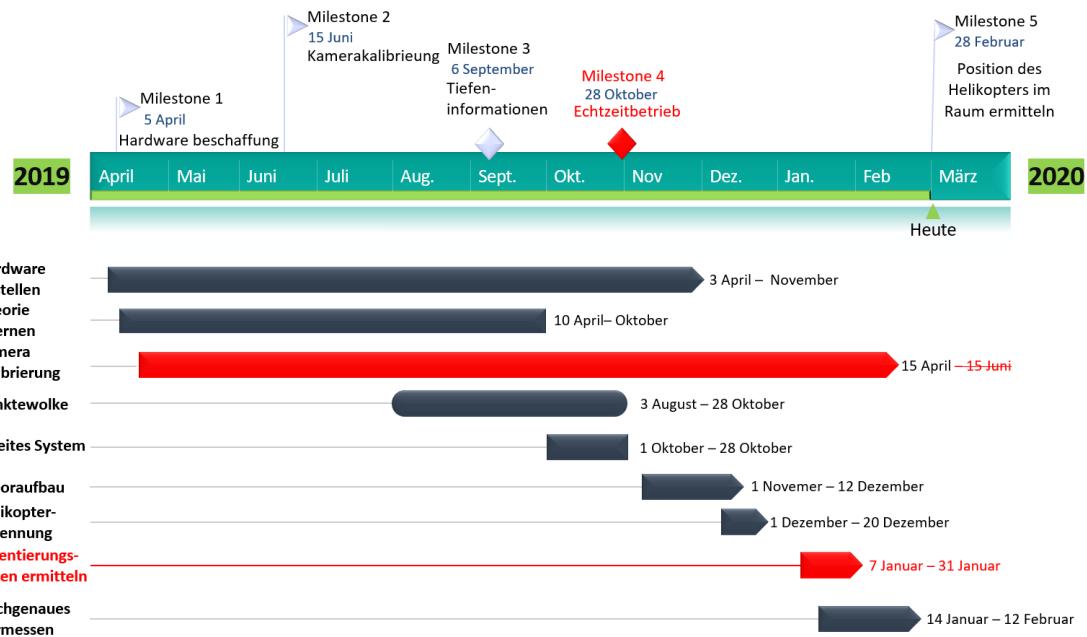


Abbildung 2.2: Endgültiger Zeitplan

# 3

## Technologien

### 3.1. Software

#### 3.1.1. OpenCV

OpenCV ist eine Open-Source-Bibliothek, die über Algorithmen für maschinelles Sehen und Bildverarbeitung verfügt [19].



Abbildung 3.1: OpenCV

Quelle: [https://de.wikipedia.org/wiki/OpenCV#/media/Datei:OpenCV\\_Logo\\_with\\_text.png](https://de.wikipedia.org/wiki/OpenCV#/media/Datei:OpenCV_Logo_with_text.png)

#### 3.1.2. scikit-learn

Scikit-learn ist eine freie, plattformunabhängige Python-Bibliothek, die für das maschinelle Lernen konzipiert ist. Die Software ist unter BSD lizenziert [20]. Von dieser Bibliothek wird lediglich die Implementierung des k-Means-Algorithmus verwendet.

### 3.2. *Hardware*



Abbildung 3.2: scikit

Quelle: [https://de.wikipedia.org/wiki/Scikit-learn#/media/File:Scikit\\_learn\\_logo\\_small.svg](https://de.wikipedia.org/wiki/Scikit-learn#/media/File:Scikit_learn_logo_small.svg)

#### 3.1.3. Open3D

Open3D ist eine Open-Source Bibliothek, die diverse Algorithmen für das Verarbeiten von 3D-Daten bereitstellt.

#### 3.1.4. PyCapture

Mittels PyCapture werden die Kameras angesteuert. Diese Bibliothek liefert 15 Bilder pro Sekunde.

#### 3.1.5. Spyder

Als Entwicklungsumgebung wurde Spyder verwendet. Hauptsächlich, weil die IDE auf wissenschaftliche Programmierung in Python ausgelegt ist [21].

## 3.2. *Hardware*

### 3.2.1. Kameras

Bei den Kameras handelt es sich um das Modell FL2-14S3C des Herstellers Point Grey. Das Objektiv ist das Modell DF6HA-1B der Marke Fujinon. Die Kameras sind horizontal zu einander verschoben und auf dem selben Stativ befestigt.

# 4

## Bildverarbeitung und Umsetzung

Im Groben lässt sich der Ablauf unseres Programms, das das Problem der Lokalisierung eines Helikopters im Raum löst, folgendermaßen beschreiben:

Anfangs werden die Kameras separat kalibriert. Ist der berechnete Fehler dieser Kalibrierung gering, werden anschließend die Kameras zu einander kalibriert. Wird dann mit beiden Kameras ein Bild der Szene aufgenommen, werden diese Bilder zuerst rektifiziert und anschließend entzeichnet. Mit den neuen Bildern ist es möglich, eine Tiefenkarte und anschließend eine Punktwolke zu generieren. Von dieser Punktwolke wird dann der Mittelpunkt und die Koordinaten dieses Punktes bestimmt. Somit gelangen wir an die  $x$ -,  $y$ - und  $z$ -Koordinaten des Helikopters.

Diese Schritte werden im Folgenden detailliert beschrieben.

### 4.1. Kalibrierung

Für eine Messung, bei der der Fehler minimiert werden soll, ist das Kalibrieren der Kameras unumgänglich. Durch die Linse einer Kamera entsteht eine tonnenförmige Verzeichnung. Diese Fehler sind meist so klein, dass sie vom menschlichen Auge nicht erfasst werden können [2] [8]. Durch die Kalibrierung der Kamera können diese kompensiert werden.

#### 4.1. Kalibrierung

## VERZEICHNUNGSEIGENSCHAFTEN

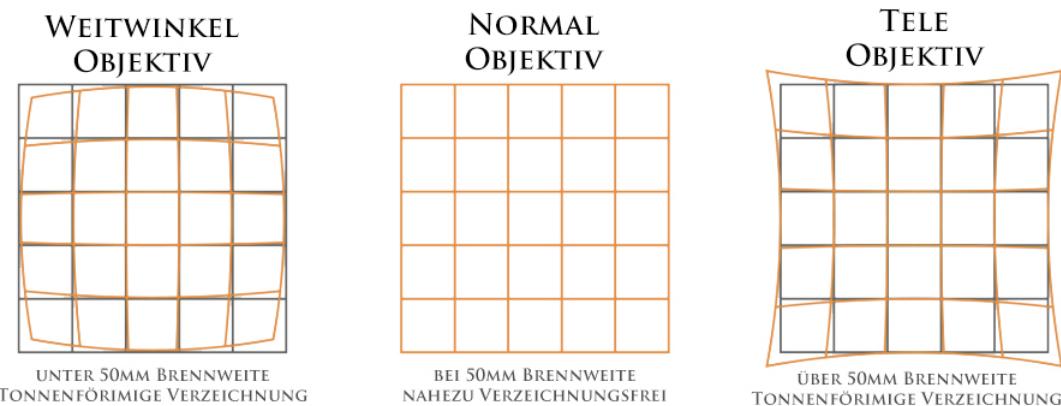


Abbildung 4.1: Verzeichnung

quelle: <http://www.fotokurs-bremen.de/wp-content/uploads/2016/11/Objektiv-Verzeichnung.jpg>

Durch die Kamerakalibrierung werden folgende Parameter bestimmt:

**Intrinsische Parameter** Bezeichnen die Abbildung von 3D-Punkten im Kamerakoordinatensystem auf den 2D-Sensor der Kamera. Es sind Informationen der Kamera selbst, die unabhängig davon sind, wo sich die Kamera befindet und wie diese ausgerichtet ist [**Intr**].

**Extrinsische Parameter** Die räumliche Lage und Orientierung der Kamera zu einem Referenzkoordinatensystem, d.h. die Rotation und Translation [10] [16].

Da es sich bei dem System um ein Stereokamera-System handelt, ist die Kalibrierung von diesem etwas komplizierter.

Zuerst müssen die Kameras gesondert kalibriert werden. Dies wird mit der Funktion `calibrateCamera` von OpenCV durchgeführt. Für die Kalibrierung wird ein Schachbrett-Muster verwendet. Wichtig ist, dass bei der Kalibrierung beide Kameras dasselbe Bild verwenden. Für die Erkennung des Schachbretts wird die OpenCV-Funktion `findChessboardCorners` verwendet. Diese liefert die Objekt- und Bild-Punkte der Aufnahme. Bei den Objekt-Punkten handelt es sich um die 3D-Punkte des Bildes, bei den Bild-Punkten um die 2D-Punkte [12].

## 4.2. Genauigkeitsabschätzung der Kalibrierung

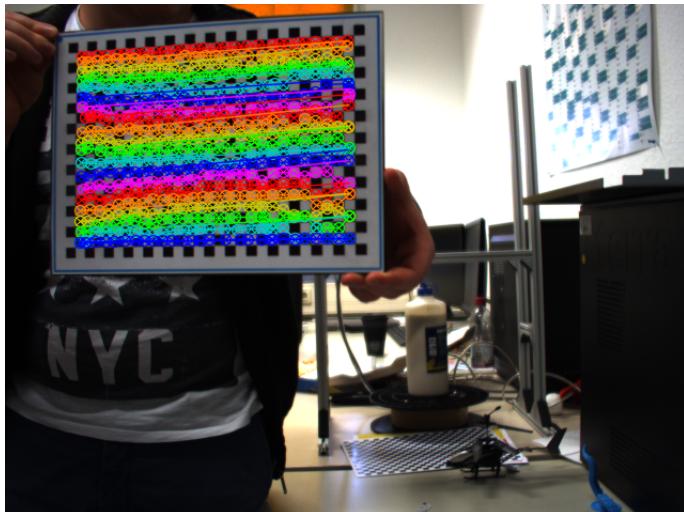


Abbildung 4.2: Kalibrierung

Für eine möglichst genaue Kalibrierung werden 50 Bilder verwendet. Anhand dieser wird jede Kamera mittels *calibrateCamera* kalibriert.

Die Funktion liefert die intrinsischen Parameter in Form von einer  $3 \times 3$  Kamera-Matrix.

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Wobei  $f_x$  und  $f_y$  die Brennweite in Pixeln und  $c_x$   $c_y$  ein Hauptpunkt, der normalerweise in der Bildmitte liegt, ist.

Die Ergebnisse dieses Vorgangs werden auf dem Computer gespeichert, sodass dieser nicht wiederholt werden muss. Anschließend wird das Ergebnis der Kalibrierungen an die OpenCV-Funktion *stereoCalibrate* übergeben.

Mit Hilfe der Stereo-Kalibrierung kann der Zusammenhang zwischen den Kameras ermittelt werden: Es werden von dem Bezugsbild, welches zum Ursprung des Koordinatensystems wird, die Objekt-Punkte verwendet. Von beiden Kamerasyttemen werden die Bild-Punkte, die jeweiligen Kameramatrizen und die Verzeichnungskoeffizienten verwendet. Die für uns wichtigsten Ergebnisse der Stereo-Kalibrierung sind die Rotation und Translation der beiden Kameras zueinander.

## 4.2. Genauigkeitsabschätzung der Kalibrierung

Der Reprojection Error ist ein qualitatives Maß für die Genauigkeit. Der Reprojection Error ist die Distanz zwischen einem detektierten Punkt in dem kalibrierten Bild und

### 4.3. Stereokalibrierung

dem korrespondierendem Weltpunkt projiziert in dasselbe Bild. Die Reprojection Error Methode ist ein Diagnosewerkzeug um zu erkennen ob eine Verbesserung gemacht werden muss oder nicht. Dies wird in 4.3 visuell dargestellt.

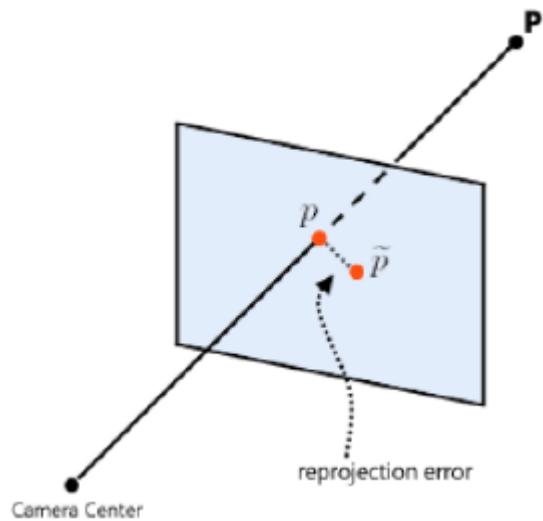


Abbildung 4.3: Reprojection Error

Quelle: [https://www.researchgate.net/figure/The-reprojection-error-is-the-distance-between-the-projected-image-point-p-and-the\\_fig3\\_267557759](https://www.researchgate.net/figure/The-reprojection-error-is-the-distance-between-the-projected-image-point-p-and-the_fig3_267557759)

### 4.3. Stereokalibrierung

Wie auf dem Bild 4.4 zu sehen ist, handelt es sich bei unserem Aufbau um ein Stereosystem. Dies ist notwendig für das Berechnen von Tiefeinformationen. Liegen alle Referenzpunkte auf einer geraden, so kann mittels Triangulation, wie auf 4.5 zu sehen ist, der  $z$ -Achsen Wert der Punkte berechnet werden.

### 4.3. Stereokalibrierung

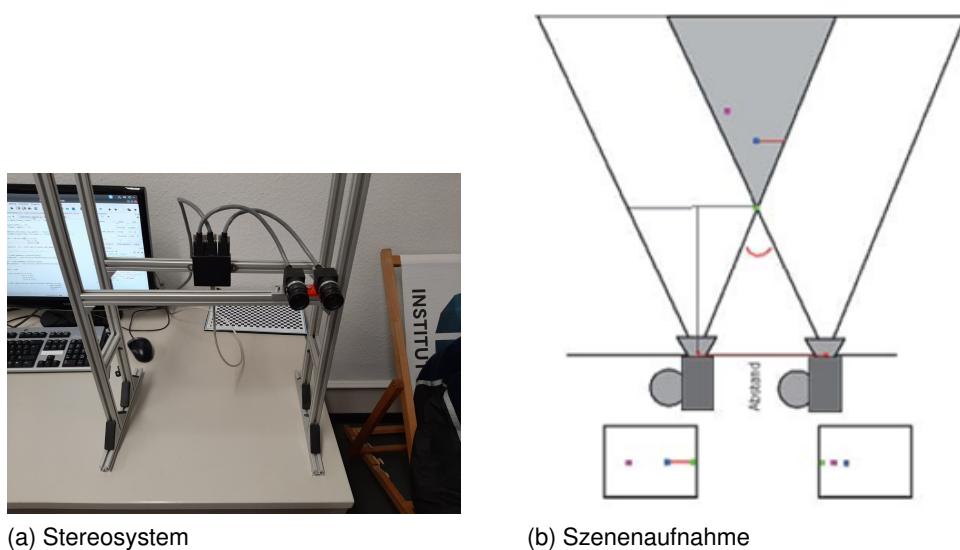


Abbildung 4.4: Stereo-System

Quelle: [https://me.efi.th-nuernberg.de/interaktion/index.php5/Bearbeitung\\_und\\_Gewinnung\\_von\\_Tiefeninformation\\_durch\\_die\\_Kopplung\\_zweier\\_Kameras](https://me.efi.th-nuernberg.de/interaktion/index.php5/Bearbeitung_und_Gewinnung_von_Tiefeninformation_durch_die_Kopplung_zweier_Kameras)

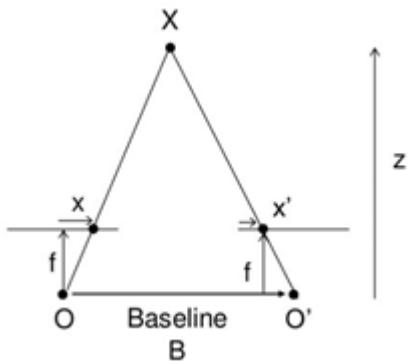


Abbildung 4.5: Z-Achsen Berechnung

Quelle: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_calib3d/py\\_depthmap/py\\_depthmap.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html)

Da die Kameras horizontal verschoben sind, erhalten wir den Stereonormalfall, der wie folgt beschrieben wird: *Das achsparallele Stereosystem zeichnet sich durch zwei Kameras aus, die nur horizontal verschoben und deren Koordinatensysteme nicht gegeneinander verdreht sind [14].*

Bei der Stereokalibrierung werden sowohl inneren Kameraparameter für das Stereo-kamerapaar, als auch die geometrische Lage zwischen den Kameras berechnet [7]. OpenCV bietet hierfür die Funktion `stereoCalibrate`. Diese Funktion erwartet die vorher berechneten Kameramatrizen und Verzeichnungskoeffizienten beider Kameras. Zudem werden Von beiden Kameras Bildpunkte benötigt, die auf die selben Objektpunkte ab-

### 4.3. Stereokalibrierung

bilden. Demnach müssen beide Kameras auf das selbe Kalibrierobjekt kalibriert werden [13]. Im Programm wird mit beiden Kameras ein Foto der Szene gemacht und anschließend versucht, das Schachbrett muster zu detektieren. War die Detektion bei beiden Bildern erfolgreich, werden die Bild- und Objekt-Punkte gespeichert und der Vorgang 50 mal wiederholt. Nach 50 erfolgreichen Detektierungen werden die Kameras zueinander kalibriert.

Durch die Stereokalibrierung erhalten wir die Rotation und Translation beider Kameras zueinander. Mit diesen Werten ist es möglich, Informationen zur Tiefe zu ermitteln. Wie die Werte validiert wurden, wird in 5.2 erläutert.

#### 4.3.1. Entzeichnung

Wie in 4.1 erwähnt, dient die Kalibrierung die Kompensation der Verzeichnung. Dieser Vorgang wird Entzeichnung genannt. Die Verzeichnung entsteht durch die Bauform der Linse und nimmt mit der Brennweite zu [17]. Das Entzeichnen eines Bildes erfolgt in OpenCV mittels der Funktion *initUndistortRectifyMap*.

#### 4.3.2. Rektifizierung

Auch wenn es sich bei unserem Aufbau um den Stereonormalfall handelt, existieren dennoch Verschiebungen auf der Y-Achse von Referenzpunkten.



Abbildung 4.6: Stereo Aufnahme

#### 4.4. Generierung einer Tiefenkarte



Abbildung 4.7: Stereo Vergrößerung

4.6 zeigt eine Aufnahme der Kameras. Vergrößert man diese wie in 4.7 sieht man, dass Referenzpunkte nicht immer auf einer Geraden liegen. Dies liegt an der Rotation um die  $x$ -Achse der beiden Kameras. Diese entsteht zum einen durch das nicht exakte Stativ, zum anderen durch die Montur der Kameras an diesem. Aufgrund dieser Rotation ist es nicht möglich Tiefeninformationen der ganzen Szene zu ermitteln. Um dieses Problem zu beheben müssen die Bilder rektifiziert werden [7]. Mittels der Rektifizierung wird die Rotation beider Kameras berechnet und auf den jeweiligen Aufnahmen kompensiert, sodass Referenzpunkte auf einer geraden Linie liegen. In der Epipolargeometrie wird diese Gerade die Epipolarlinie genannt [13] [18]. Nach der Rektifizierung sind alle diese Epipolarlinien parallel. Die Rektifizierung wird in OpenCV mittels der Funktion `stereoRectify` erreicht.

#### 4.4. Generierung einer Tiefenkarte

Die Tiefenkarte, aus der die Punktewolke generiert wird, wird mit Hilfe des Blockmatching Algorithmus berechnet. Dabei handelt es sich um einen objektorientierter Algorithmus, der zwei gleichgroße Blöcke aus verschiedenen Bildern vergleicht. Es wird ein Bereich im ersten Bild mit allen gleichgroßen Bereichen im zweiten verglichen. Dieser Bereich ist eine  $n \times m$ -Matrix. Der mittlere Quadratische Fehler (MSE) dient als vergleichsmaß, welches sie wie folgt berechnet [3]:

$$MSE(x, y, \Delta) = \frac{1}{n - m} \sum_{i=-k}^k \sum_{j=-k}^k (|G_L(x + i, y + j) - G_R(x + i + \Delta, y + j)|^2) \quad (4.1)$$

$G_L$  und  $G_R$  sind die Bilder als Matrizen in Grauwerten abgespeichert. Der Offset  $\Delta$  ist der Indikator für die Sprungweite im zweiten Bild. Nun ist es möglich, die Disparität  $d$

#### 4.5. Punktewolken-Generierung

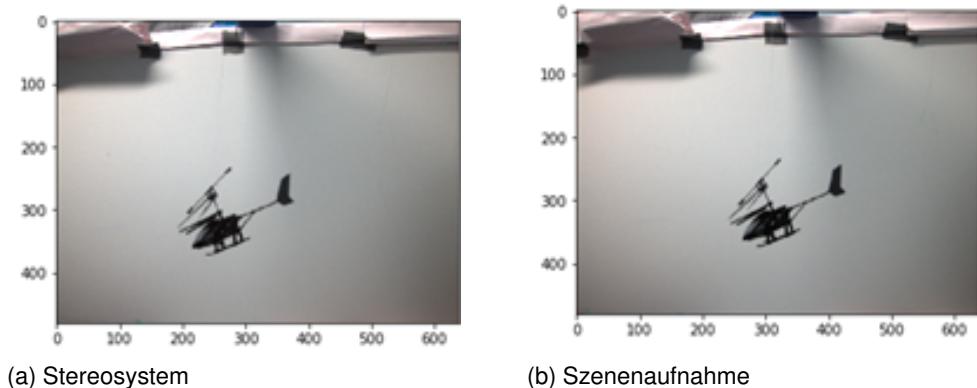


Abbildung 4.8: Aufnahmen eines Helikopters

zu bestimmen (die Disparität bezeichnet die horizontale Differenz zwischen zwei korrespondierenden Bildpunkten [1]). Dies beschränkt sich allerdings auf die Bereiche, in der der MSE minimal ist [3]:

$$D = \min_{|\Delta| \leq d_{max}} \{MSE(x, y\Delta)\} \quad (4.2)$$

Durch eine Pixelselektion kann eine Tiefenkarte über das ganze Bild erstellt werden.

4.8 zeigt Aufnahmen des Helikopters von beiden Kameras.

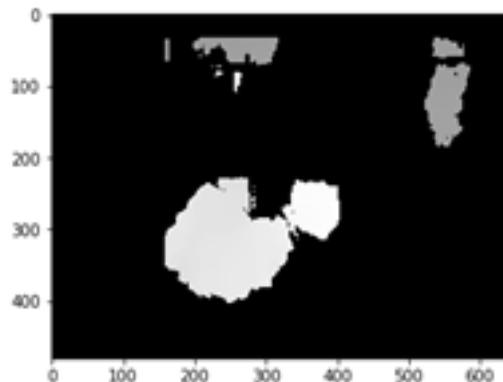


Abbildung 4.9: Tiefenkarte

#### 4.5. Punktewolken-Generierung

Um aus der Tiefenkarte 3D-Koordinaten zu bestimmen, werden zunächst die  $x$ - und  $y$ - Werte derjenigen Pixel, die einen Wert größer 0 in der Tiefenkarte aufweisen, mit dem dazugehörigen Disparitätswert  $d$  in homogener Schreibweise, d.h. mit einer ange-

## 4.6. Lokalisierung des Helikopters

hängten 1, in einen Vektor geschrieben. Außerdem werden die Farbwerte dieser Pixel in einer Matrix gespeichert. Der Vektor  $(x, y, d, 1)$  wird dann mit der  $4 \times 4$ -Perspektiv-Transformations-Matrix  $Q$ , die durch die Rektifizierung bestimmt wurde multipliziert. Die Matrix  $Q$  enthält aktualisierte Informationen über die Brennweite, die Hauptpunktsverschiebung und die Länge der Baseline nach der Rektifizierung. Zum Schluss muss lediglich der Resultatsvektor durch den Wert des letzten Elements geteilt werden, um die Homogenisierung rückgängig zu machen und die gewünschten  $x$ -,  $y$ -, und  $z$ -Koordinaten in kartesischer Form zu erhalten. Diese werden mit den gespeicherten Farbwerten in eine *.ply*-Datei geschrieben, um beispielweise mit CloudCompare anschließend visualisiert werden zu können.

Die Berechnung könnte auch innerhalb einer geschachtelten Schleife, die über sämtliche Pixelwerte iteriert, vorgenommen werden. Der vektorbasierte Ansatz bietet in Python jedoch deutliche Performancevorteile, weshalb die Berechnung auf diese Weise umgesetzt wurde. Das Speichern der Resultate ist optional, es könnte auch direkt mit der Lokalisierung fortgefahrene werden, ist jedoch zur Überprüfung der Resultate sehr hilfreich. [4.10](#) visualisiert das Ergebnis, visualisiert mit CloudCompare.

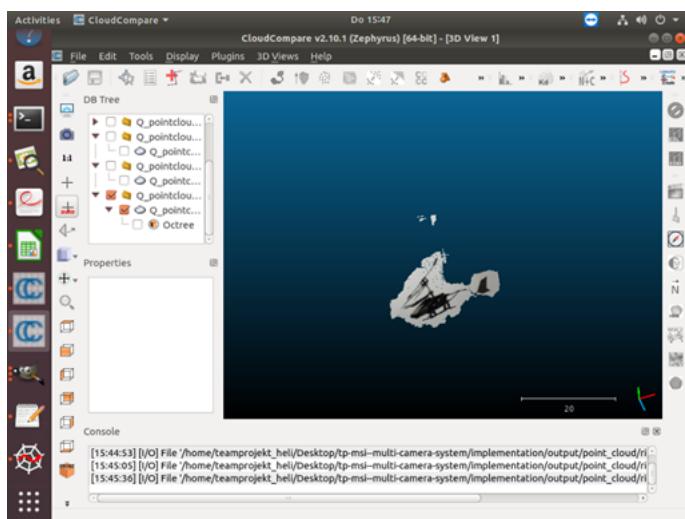


Abbildung 4.10: 3D-Punktwolke des Modellhubschraubers vor weißem Hintergrund in CloudCompare

## 4.6. Lokalisierung des Helikopters

### 4.6.1. k-Means

Der k-Means Algorithmus ist eine Clusteranalyse, welche verwendet wird, um einen Clustermittelpunkt zu ermitteln. Dafür ist die Clusteranzahl statisch davor zu bestim-

#### 4.6. Lokalisierung des Helikopters

men. Da wir unsere Detektion unter Laborbedingungen durchführen kann davon ausgegangen werden, dass es nur ein Cluster gibt, nämlich die Punktewolke des Helikopters. Demnach wird die Clustergröße auf eins gesetzt, um den Mittelpunkt des Helikopters zu ermitteln. Der Algorithmus partitioniert die Datenpunkte so, dass die summierte Varianz innerhalb jedes Clusters minimiert wird [4].

Die Problematik in einem lokalen Minimum hängen zu bleiben entfällt, da es nur ein Cluster gibt. Die euklidische Distanz von jedem Punkt zu dem optimalen clustermittelpunkt wird bestimmt. Der Mittelpunkt des Clusters wird mit folgender Formel berechnet:

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist(c_i, x)^2 \quad (4.3)$$

SSE (sum of squared error) soll minimiert werden, was zu einem besseren Mittelpunkt des Cluster führt. Dafür ist es Notwendig *dist* zu minimieren. Dabei handelt es sich um den euklidischen Abstand von dem Clustermittelpunkt zu dem jeweiligen Punkt. Eine Quadrierung wird vorgenommen um das Vorzeichen zu eliminieren [11]. Durch die gegebenen Umstände ist eine Vereinfachung möglich:

$$SSE = \sum_{x \in C_i} dist(c_i, x)^2 \sum_{i=1}^K \quad (4.4)$$

Der Anpassungsschritt, in dem der neue Clustermittelpunkt gewählt wird, wird durch das arithmetisches Mittel ausgerechnet:

$$\mu_k = \frac{\sum_n x_n}{n} \quad (4.5)$$

#### 4.6. Lokalisierung des Helikopters

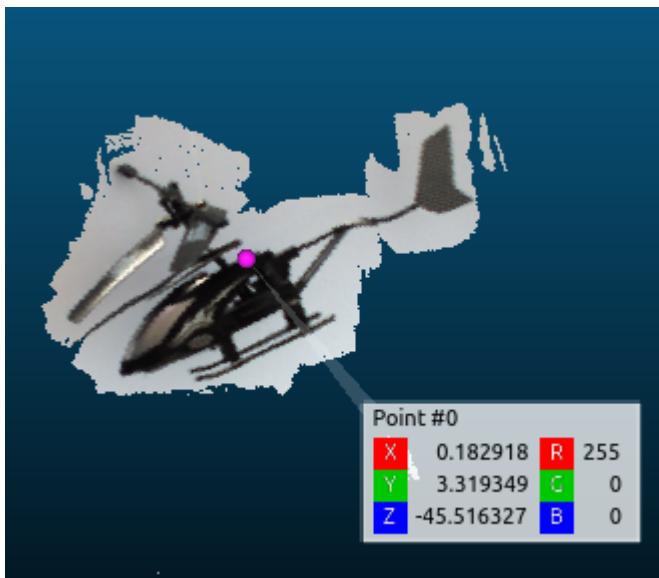


Abbildung 4.11: Helikopter Cluster

In 4.11 sieht man den angewandten K-Means Algorithmus auf eine generierte Punktwolke aus unserem Kamerasystem. Gut zu erkennen ist, dass der Mittelpunkt des Helikopterclusters auch der tatsächliche Mittelpunkt des Helikopters ist.

# 5

## Experimente

### 5.1. Vereinfachte Kalibrierung

Eine andere Möglichkeit, als in 4.1 erläutert, eine Kamera zu kalibrieren, ist mit Hilfe eines rechteckigen Kalibrierobjektes. In unserem Fall handelt es sich, wie in 5.1 zu sehen ist, um die Verpackung von Heftklammern.



Abbildung 5.1: Kalibrierobjekt

Die Formel zur Berechnung der Brennweite  $f_x$  und  $f_y$  ergibt sich aus den Formeln

$$f_x = \frac{\Delta x'}{x} z \quad (5.1)$$

und

## 5.2. Tiefe und Stereokalibrierung

$$f_y = \frac{\Delta y'}{y} z \quad (5.2)$$

bei  $\Delta x$  und  $\Delta y$  handelt es sich um die jeweiligen Seitenlängen des Kalibrierobjekts.  $\Delta x'$  und  $\Delta y'$  sind die Längen der Seiten des Kalibrierobjekts in Pixelwerten.  $z$  ist der Abstand zwischen Linse und Objekt [15] [5].



Abbildung 5.2: Vereinfachte Kalibrierung

Quelle: Solem 2012

Wie in 5.2 zu sehen ist, muss das Kalibrierobjekt senkrecht auf eine plane Oberfläche gestellt werden. So auch die Kamera. Die Kamera muss dann so ausgerichtet werden, dass das Objekt im Bild genau zentriert und entlang der Bildzeilen und -spalten ausgerichtet ist [5].

Obwohl diese Kalibrierung sehr ungenau erscheint, wurde ein dennoch relativ geringer Reprojection-Error errechnet. Der wie in 4.1 ermittelte Wert war dennoch besser.

## 5.2. Tiefe und Stereokalibrierung

Um die Werte, die wir von der Stereokalibrierung erhalten zu validieren, wurde folgendes Experiment durchgeführt:

Mit dem Stereo-System wurden zwei Aufnahmen einer Szene aufgenommen (5.3 und 5.4). Auf dem seitlichen Bild 5.5 sieht man die tatsächliche Tiefe der Objekte im Bild.

Ziel war es, erstmals Tiefeninformationen der Szene zu ermitteln und parallel, wenn möglich, die Stereokalibrierung zu validieren. Auf den zwei Bildern der Szene wurden hier vorerst händisch, dann mittels des SIFT-Algorithmus Referenzpunkte ermittelt, zu denen die Tiefe berechnet werden sollte.

Für die Tiefenberechnung wurde Formel 5.3 verwendet.

### 5.3. Fehlerfortpflanzung und Tiefenvalidierung



Abbildung 5.3: Tiefe links



Abbildung 5.4: Tiefe rechts



Abbildung 5.5: Tiefe seitlich

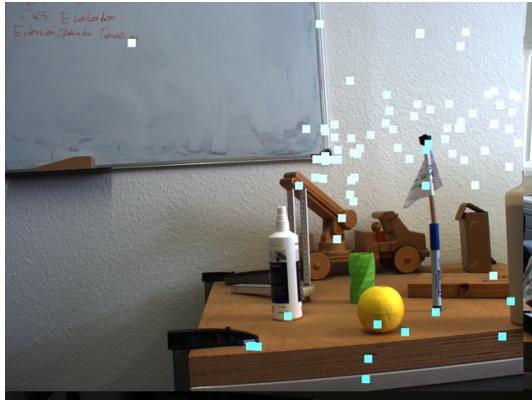


Abbildung 5.6: Tiefeninformation Punkte

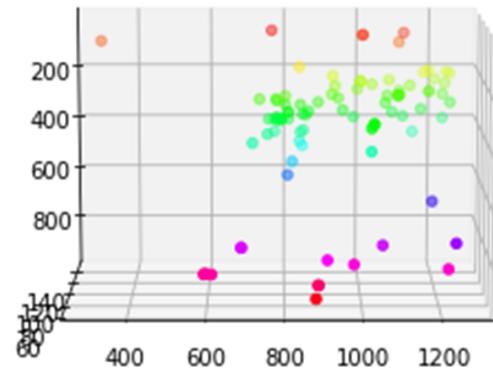


Abbildung 5.7: Tiefeninformation 3D-Plot

$$Z = \frac{f * B}{x - x'} \quad (5.3)$$

Wobei  $f$  die Brennweite,  $B$  die Baseline und  $x$  bzw.  $x'$  die Referenzpunkte sind. Als  $B$  wird die in der Stereokalibrierung erhaltene Translation verwendet.

Mittels eines selbst geschriebenem Algorithmus, der Werte mit im Mittel großem  $Z$  weißlich und Werte mit im Mittel kleinem  $Z$  hellblau einfärbt, kann man sich die Informationen zur Tiefe auf einem Bild darstellen.

In 5.6 kann man gut erkennen, dass Objekte, die näher an der Kamera stehen, mit einem hellen blau markiert sind. Objekte, die weiter entfernt sind, mit einem weißlichem Farbton. Durch dieses Experiment war es uns möglich zu Validieren, dass unser Ansatz der Richtige war.

### 5.3. Fehlerfortpflanzung und Tiefenvalidierung

Bei der Tiefenberechnung mit Hilfe eines Stereokamerasystems wird die Raumposition nicht gemessen, sondern aus den beiden vom Kamerasystem erfassten Bildern be-

### 5.3. Fehlerfortpflanzung und Tiefenvalidierung

rechnet. Dazu wird der gleiche Raumpunkt in beiden Bildern detektiert. Die räumliche Rekonstruktion ist im Stereonormalfall besonders einfach, da die beiden Kameras keine Rotation zueinander aufweisen. Dadurch vereinfachen sich die allgemeinen Formeln zur räumlichen Rekonstruktion [6]

$$x = x_0 + (z - z_0) \frac{r_{11}(x' - x'_0) + r_{12}(y' - y'_0) - r_{13}f}{r_{31}(x' - x'_0) + r_{32}(y' - y'_0) - r_{33}f} \quad (5.4)$$

$$y = y_0 + (z - z_0) \frac{r_{21}(x' - x'_0) + r_{22}(y' - y'_0) - r_{23}f}{r_{31}(x' - x'_0) + r_{32}(y' - y'_0) - r_{33}f} \quad (5.5)$$

zu

Rechtes Bild:

$$x = -z \frac{x'_1}{f} \quad (5.6)$$

$$y = -z \frac{y'_1}{f} \quad (5.7)$$

Linkes Bild:

$$x = B - z \frac{x'_2}{f} \quad (5.8)$$

$$y = -z \frac{y'_2}{f} \quad (5.9)$$

Aus den vereinfachten Formeln 5.6, 5.7, 5.8 und 5.9, ist es ersichtlich, dass ein Raumpunkt in beiden Bildern die gleiche  $y$ -Koordinate aufweist, sich jedoch normalerweise in der  $x$ -Koordinate unterscheidet. Diese Differenz wird als  $x$ -Parallaxe  $\rho_x = x_1 - x_2$  oder Disparität  $D$  bezeichnet.

Durch Gleichsetzen der Formeln 5.6 – 5.9 für linkes und rechtes Bild, erhält man folgende Formel zur Berechnung der Tiefeninformation:

$$z = -\frac{f * B}{x'_1 - x'_2} = -\frac{f * B}{\rho x'} \quad (5.10)$$

Da die Brennweite  $f$  und der Abstand der beiden Kameras zueinander (Baseline  $B$ ) konstant sind, ist die Entfernung allein von der  $x$ -Parallaxe abhängig. Dabei ist ein Punkt näher, je größer die Parallaxe ist.

# 6

## Probleme

Trotz dem Erfolgreichen lokalisieren unseres Helikopters, hatten wir einige langwierigen Schwierigkeiten, die auch nicht bei Beendigung des Projektes gelöst wurden. Das Hauptproblem, mit dem wir nicht gerechnet hatten, war die Kalibrierung und die vielen Folgeprobleme, die sich daraus entwickelten. Nach dem Arbeitspaket Kalibrierung, in dem wir uns eingelesen, selbst kalibriert haben und unsere Methode mit den Methoden von OpenCV verglichen haben, sind wir davon ausgegangen das Thema sei abgeschlossen. Aber eine plausible Validierung aus den Daten war für uns nicht möglich. Die erhaltenen Werte konnten nicht eingeordnet werden, ob diese gut oder schlecht sind. Zwar versuchten wir immer wieder mit neuen Messungen dies zu bestätigen, aber ein konsistenter Beleg, dass unsere Kalibrierung exakt ist, war nicht möglich. So dachten wir oft wir hätten dieses Themen erfolgreich beendet, aber bei Fehlschlägen in weiterführenden Themen, wie große Unstimmigkeiten in den Resultaten, konnte durch eine exaktere Kalibrierung bessere Ergebnisse erreicht werden. Die Ergebnisse schwankten auch sehr stark, wie die Lichteinstrahlung zu Zeiten der Messung im Labor war. So war es oft am besten abends Aufnahmen zu machen, da kein direktes Licht auf unseren Versuchsaufbau fiel. Trotz des abhängen des Fensters, war es nicht möglich dieses Schwanken zu eliminieren. Dies führte auch bei Erfolgen schnell zu einer Demotivation weiter zu machen, da eine andere Fehlerquelle nicht auszuschließen war.

Wir haben mit der Kalibrierung von zwei Kameras zueinander gestartet, was viel Zeit in Anspruch genommen hat. Als wir unseren Aufbau um ein weiteres Kamerasystem erweitert haben, war es nicht mehr möglich Bilder aufzunehmen. Dieses Problem konnte durch eine Reduzierung der Bildgröße behoben werden. Es resultierte, dass alles neu kalibriert werden musste.

Des Weiteren war es uns oft nicht möglich aus der OpenCV Dokumentation die richtigen Schlüsse zu ziehen, da sie lückenhaft und teilweise veraltet war. Daraus resultierten Fehler, welche laut der Dokumentation nicht passieren sollten.

Die Library der Kameras um diese ansteuern zu können wurde von Flycap geliefert. Diese hatte aber keine vernünftige Dokumentation. Um Flycap zu verwenden wurde Spyder als Entwicklungsumgebung ausgewählt. Die Debug Möglichkeiten von Spyder sind sehr begrenzt und es ist nicht möglich größere Matrizen anzeigen zu lassen.

# 7

Fazit

# Literatur

- [1] Matthias Behnisch. *Stereovision: Grundlagen*. URL: \url{https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwi8s6WlwfLnAhXGeZoKHetUDj8QFjAurl=https%3A%2F%2Fwww.techfak.uni-bielefeld.de%2F~rhaschke%2Flehre%2FWS04%2Fhumanoids%2Fausarbeitung%2FStereoalgorithmen1.pdf&usg=A0vVaw2TnHMxL7VlIVDFFF} (besucht am 26. 02. 2020).
- [2] fotokurs bremen. *Objektive Brennweite und Verzeichnung*. URL: <http://www.fotokurs-bremen.de/objektive-brennweite-und-verzeichnung/> (besucht am 22. 02. 2020).
- [3] Gregory Föll. *Stereo Vision: Vergleich verschiedener Algorithmen zur Lösung des Korrespondenzproblems*. URL: <https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-aw2/foell/bericht.pdf> (besucht am 27. 02. 2020).
- [4] Matthias Franz. „Dimensionsreduktion“. 2019.
- [5] Matthias Franz. „ransac“. 2019.
- [6] Matthias Franz. „Zweibildauswertung“. 2019.
- [7] ZBS Ilmenau. *Rektifizierung der Stereobilder*. URL: [https://zbs-ilmenau.de/intern/vip/3D-Daten/VIP\\_3D\\_Rektifizierung\\_Stereobilder.html](https://zbs-ilmenau.de/intern/vip/3D-Daten/VIP_3D_Rektifizierung_Stereobilder.html) (besucht am 25. 02. 2020).
- [8] Ingmar Jahr. *Kamerakalibrierung*. URL: <https://www.invision-news.de/allgemein/kamerakalibrierung/> (besucht am 23. 02. 2020).
- [9] Naveen Joshi. *The Present And Future Of Computer Vision*. URL: <https://www.forbes.com/sites/cognitiveworld/2019/06/26/the-present-and-future-of-computer-vision/#490553c0517d> (besucht am 22. 02. 2020).
- [10] Georg Rupert Müller. *Kalibrierung von Kameras*. URL: <https://www.unibw.de/tas/forschung/kalibrierung-von-kameras/view> (besucht am 23. 02. 2020).
- [11] TU München. *Clusteranalyse*. URL: <https://www-m9.ma.tum.de/material/felix-klein/clustering/Methoden/K-Means.php> (besucht am 27. 02. 2020).
- [12] OpenCV. *Calibration Tutorial*. URL: [https://docs.opencv.org/3.4/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html) (besucht am 22. 02. 2020).

## LITERATUR

- [13] OpenCV. *Camera Calibration and 3D Reconstruction*. URL: [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html) (besucht am 26.02.2020).
- [14] Robert Sablatnig und Sebastian Zambanini. *Stereo and Motion*. URL: <https://www.cg.tuwien.ac.at/courses/EinfVisComp/Skriptum/SS13/EVC-18%20Stereo%20und%20Motion.pdf> (besucht am 24.02.2020).
- [15] Jan Erik Solem. *Programming Computer Vision with Python. Tools and algorithms for analyzing images*. 2012.
- [16] Springer. *Das Kameramodell*. URL: [https://link.springer.com/content/pdf/10.1007%2F3-540-27473-1\\_3.pdf](https://link.springer.com/content/pdf/10.1007%2F3-540-27473-1_3.pdf) (besucht am 23.02.2020).
- [17] Wikipedia. *Entzerrung (Fotografie)*. URL: [https://de.wikipedia.org/wiki/Entzerrung\\_\(Fotografie\)](https://de.wikipedia.org/wiki/Entzerrung_(Fotografie)) (besucht am 26.02.2020).
- [18] Wikipedia. *Eipolargeometrie*. URL: <https://de.wikipedia.org/wiki/Eipolargeometrie#8-Punkt-Algorithmus> (besucht am 27.02.2020).
- [19] Wikipedia. *OpenCV*. URL: <https://de.wikipedia.org/wiki/OpenCV> (besucht am 22.02.2020).
- [20] Wikipedia. *Scikit-learn*. URL: <https://de.wikipedia.org/wiki/Scikit-learn> (besucht am 22.02.2020).
- [21] Wikipedia. *Spyder (Software)*. URL: [https://de.wikipedia.org/wiki/Spyder\\_\(Software\)](https://de.wikipedia.org/wiki/Spyder_(Software)) (besucht am 27.02.2020).

# Abbildungsverzeichnis

1	Kamera-System	.
2	Kamera-Kalibrierung und Punktewolke	.
2.1	Projektplan Zwischenergebnis	.
2.2	Endgültiger Zeitplan	.
3.1	OpenCV	.
3.2	scikit	.
4.1	Verzeichnung	.
4.2	Kalibrierung	.
4.3	Reprojection Error	.
4.4	Stereo-System	.
4.5	Z-Achsen Berechnung	.
4.6	Stereo Szenenaufnahme	.
4.7	Stereo Vergrößerung	.
4.8	Aufnahmen eines Helikopters	.
4.9	Tiefenkarte	.
4.10	3D-Punktewolke des Modellhubschraubers vor weißem Hintergrund in CloudCompare	.
4.11	Helikopter Cluster	.
5.1	Kalibrierobjekt	.
5.2	Vereinfachte Kalibrierung	.
5.3	Tiefe links	.
5.4	Tiefe rechts	.
5.5	Tiefe seitlich	.
5.6	Tiefeninformation Punkte	.
5.7	Tiefeninformation 3D-Plot	.