

---

# 3D zu 2D Drohnenflug Visualisierung

---

## BACHELORARBEIT

Author: Nina Suette, 11912463



---

Supervisor: Univ.-Prof. Dr. Stephan Michael Weiss

Institut für Intelligente Systemtechnologien

Klagenfurt, September 2022

## Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- die eingereichte wissenschaftliche Arbeit selbstständig verfasst und andere als die angegebenen Hilfsmittel nicht benutzt habe,
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, einschließlich signifikanter Betreuungshinweise, vollständig offengelegt habe,
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder sinngemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben (z. B. in Fußnoten) ersichtlich gemacht habe,
- die Arbeit bisher weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und
- zur Plagiatskontrolle eine digitale Version der Arbeit eingereicht habe, die mit der gedruckten Version übereinstimmt. Ich bin mir bewusst, dass eine tatsächlichenwidrige Erklärung rechtliche Folgen haben wird.



Klagenfurt, 12. September 2022

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>1</b>
<b>Abbildungsverzeichnis</b>	<b>4</b>
<b>Abstract</b>	<b>6</b>
<b>Vorwort</b>	<b>7</b>
<b>1 Einleitung</b>	<b>8</b>
1.1 Zielsetzung . . . . .	10
1.2 Gliederung der Arbeit . . . . .	10
<b>2 Grundlagen</b>	<b>11</b>
2.1 Kamera und Parameter . . . . .	11
2.1.1 Funktion der Kamera . . . . .	11
2.1.2 Kameramodell . . . . .	11
2.1.3 Kameraparameter . . . . .	12
2.2 Verzerrungskoeffizienten . . . . .	14
2.3 Rotations- und Translationskoeffizienten . . . . .	15
2.4 Kalibrierung . . . . .	18
2.4.1 Kalibrierungsmethode . . . . .	18
2.5 Kalibrieren der Kamera . . . . .	19
2.6 Projektion . . . . .	21
2.6.1 Homogene Transformationsmatrix . . . . .	21
2.7 perspektivische Transformation . . . . .	22
2.7.1 Entfernen der Verzerrung . . . . .	24
2.8 Projektion einer Trajektorie . . . . .	26
<b>3 Methodik</b>	<b>26</b>
3.1 Ubuntu 20.04 Virtualbox . . . . .	26
3.2 ROS . . . . .	27
3.3 OpenCV . . . . .	27
3.4 Kalibrierung der Kamera . . . . .	28
3.5 Generieren einer Trajektorie mittels MATLAB . . . . .	28
3.6 ROS Package . . . . .	29
3.7 Programm und Projektion . . . . .	30
3.8 Experiment in der Drohnenhalle . . . . .	31

<b>4</b>	<b>Ergebnisse</b>	<b>31</b>
4.1	Kalibrierung der Kamera . . . . .	32
4.2	Projektion eines einzelnen Punktes . . . . .	33
4.3	Generieren einer Trajektorie . . . . .	34
4.4	Projektion einer Trajektorie . . . . .	36
4.5	Live-Projektion in der Drohnenhalle . . . . .	37
<b>5</b>	<b>Diskussion</b>	<b>38</b>
<b>6</b>	<b>Fazit</b>	<b>39</b>
6.1	Zusammenfassung . . . . .	39
6.2	Ausblick . . . . .	40
<b>7</b>	<b>Nachwort</b>	<b>41</b>

## Abbildungsverzeichnis

1	Brennweite [12] . . . . .	12
2	Vertauschen der Brennpunkt- und Bildebene [12] . . . . .	13
3	Rotation and Translation [12] . . . . .	15
4	Veränderung der Koordinaten durch Rotation in 2D [12] . . . . .	17
5	Setup und Koordinatensysteme . . . . .	18
6	Schachbrettmuster [13] . . . . .	20
7	Lochkameramodell [17] . . . . .	24
8	Beispiel für Verzerrung [17] . . . . .	25
9	Ergebnisse Kalibration im YAML-File . . . . .	32
10	Projektion eines einzelnen Punktes in der Nahansicht . . . . .	33
11	Projektion eines einzelnen Punktes Fernansicht . . . . .	33
12	in MATLAB generierte Trajektorie mit 100 Wegpunkten . . . . .	34
13	in MATLAB generierte Trajektorie mit 1000 Wegpunkten . . . . .	35
14	Projektion einer Trajektorie mit 100 Wegpunkten . . . . .	36
15	Projektion einer Trajektorie mit 1000 Wegpunkten . . . . .	36
16	in der Drohnenhalle live aufgenommene Seitenansicht der Trajektorie	37
17	in der Drohnenhalle live aufgenommene Nahansicht der Trajektorie .	38

## **Abkürzungsverzeichnis**

OpenCV - Open Computer Vision  
ROS - Robot Operating System  
MoCap – Motion Capture  
Ext4 - fourth extended filesystem (Linux Kernel)  
Btrfs - B Tree File System  
ReiserFS - Reiser File System  
ZFS-Partitionen - Zettabyte File System (Pseudo-Akronym)  
AMD - Advanced Micro Devices  
MMX - Multi Media Extension  
SSE - Streaming SIMD Extensions  
STL - Standard Template Library  
IMU - Inertial Measurement Unit  
CSV - Comma-separated values  
YAML - yet another markup language  
TXT - Text  
XML - extensible markup language  
CPP - C++ (C Plus Plus, Programmiersprache)  
MATLAB - matrix laboratory  
BAG - Bathymetric Attributed Grid  
AR - Augmented Reality  
UWB – Ultra wide band

# Abstract

Ziel der vorliegenden Arbeit ist die Entwicklung eines Tools, welches durch eine an einer Drohne angebrachte Kamera aufgenommene 3D Punkte in der realen Welt zu 2D Punkten in einem Kamerabild visualisiert. Zu diesem Zwecke muss geklärt werden, wie es möglich ist vom Zustandsschätzer und Planer geplante, vom Regler geflogene und vom Schätzer geschätzte Pfade im Drohnenflug zu einem bestimmten Zeitpunkt im Kamerabild direkt zu visualisieren. Dazu müssen mittels Kalibrierung der Kamera die intrinsischen und die extrinsischen Parameter identifiziert werden. In dieser Bachelorarbeit werden Parameter Brennweite, optische Zentren, radiale und tangentiale Verzerrung, sowie Rotations- und Translationskoeffizienten eingeführt und beschrieben, wie mittels dieser Parameter dreidimensionale Punkte der realen Welt in zweidimensionale Bildpunkte transformiert und anschließend auf ein Kamerabild projiziert werden. Folgend soll eine Trajektorie generiert und im Code eingelesen werden. Mittels der oben genannten Parameter sollen die Wegpunkte der Trajektorie in Bildpunkte transferiert und am Kamerabild abgebildet werden. Die Implementierung des Tools in umfangreichere Forschungsprojekte, um diese zu unterstützen, kann zukünftig eines seiner Hauptaufgaben werden. Das Resultat der Arbeit ist gut visuell darstellbar und kann in weiteren Projekten in der Drohnenforschung genutzt werden.

The goal of this thesis is to develop a tool that directly visualizes 3D points in the real world captured by a camera attached to a drone to 2D points in an image. For this purpose it must be clarified how it is possible to directly visualize paths planned by the state estimator and planner, flown by the controller, and estimated by the estimator in drone flight at a particular point in time in the image? For this purpose, the intrinsic and extrinsic parameters must be identified by means of camera calibration. In this bachelor thesis, parameters such as focal length, optical centers, radial and tangential distortion, as well as rotation and translation coefficients are introduced. It is described how three-dimensional points of the real world are transformed into two-dimensional image points by means of these parameters and subsequently projected onto a camera image. Furthermore, a trajectory is to be generated and read into the Code. By means of the parameters mentioned above, the waypoints of the trajectory are to be transferred into image points and mapped onto the camera image. The implementation of the tool in more extensive research projects to support them can become one of his main tasks in the future. The result of the work is well visualizable and can be used in many projects in drone research.

# Vorwort

Die vorliegende Bachelorarbeit behandelt die Fragestellung „Wie ist es möglich, die vom Planer geplanten, vom Regler geflogenen, und vom Schätzer geschätzten Pfade zu einem Zeitpunkt im Kamerabild zu visualisieren?“. Um die Forschungsfrage zu klären, wurde zusätzlich zu dieser Arbeit ein Projekt realisiert, welches die Umsetzung eines Programms, das diese Überlagerung von einer Trajektorie in die reale Welt möglich macht, darstellt. Ziel war es, die im Trackingsystem bekannten 3D Pfade zu visualisieren, indem diese Punkte auf das momentane zweidimensionale Kamerabild projiziert werden. Diese und zwei weitere Fragestellungen wurden von meinem Betreuer Herrn Univ.Prof. Dr. Stephan Michael Weiss, als Vorschlag für meine Arbeit zur Verfügung gestellt. Da mein Interesse an der Arbeit mit Drohnen schon seit Beginn meines Studiums im Herbst 2019 geweckt wurde, habe ich mich für dieses Thema entschieden. Durch seine breit gefächerten Fachkenntnisse in diesem Themengebiet, wurden mir wertvolle Einblicke in die Materie verschafft und auch schon eine Einleitung in die Thematik folgender Mastervorlesungen ermöglicht.

Bei der Kalibrierung der Kamera ergaben sich Startschwierigkeiten, diese konnten jedoch nach genauer Fehleranalyse mit Unterstützung meines Betreuers, gelöst werden. Auch für unzählige weitere Fragen und reichlich Input bezüglich der Vorgehensweise, für einen erfolgreichen Abschluss meiner Arbeit, standen mir Herr Univ.Prof. Dr. Weiss und Herr MEng. Brommer stets bei Seite.

Diese Arbeit stellt den Abschluss meines Bachelorstudiums Informationstechnik, im Zweig Wirtschaftsingenieurwesen an der Alpen-Adria-Universität Klagenfurt dar und war gleichzeitig Hauptbestandteil meiner Tätigkeit als studentische Mitarbeiterin am Institut für intelligente Systemtechnologien.

Ich möchte meinen Betreuern für ihre gelungene Anleitung und ihre Unterstützung während dieses Prozesses danken.

# 1 Einleitung

Aktuell sind Drohnen aufgrund ständiger Weiterentwicklungen der Technologien in vielen Bereichen unserer Gesellschaft zu einem gängigen und leistungsstarken Werkzeug geworden. Die unbemannten Luftfahrzeuge können aufgrund ihrer praktischen Größe, ihrer Flexibilität und ihres ständig wachsenden Potenzials in verschiedenen Bereichen eingesetzt werden. In Betracht des Überbegriffes, der projektiven Geometrie finden Drohnen zum heutigen Stand der Technik in Bereichen wie Photogrammetrie, Visual tracking oder Augmented Reality ihren Einsatz. Die digitale Photogrammetrie machte es möglich, photogrammetrische Vermessungen mit einem PC oder einer Workstation durchzuführen, ohne aufwendigere Instrumente wie beispielsweise Plotter verwenden zu müssen. Das Ziel der Photogrammetrie ist die Extraktion metrischer Informationen aus Bildern. Das Hauptanwendungsgebiet der Photogrammetrie ist die Kartenerstellung, aber auch in Bereichen wie in der Mikroskopie, industriellen Messtechnik, Architekturphotogrammetrie und medizinischen Bildgebung wird sie verwendet [1].

Im Sektor des Visual trackings wird zwischen dem visuellen 3D-Tracking und dem visuellen 2D-Tracking unterschieden. Während das visuelle 2D-Tracking darauf abzielt, die Größe, den Schwerpunkt oder die Trajektorie des Objekts im Bild kontinuierlich wiederherzustellen, verfügt das visuelle 3D-Tracking noch über zusätzliche Funktionen, wie die Wiederherstellung der 3D-Position des Objekts [2]. Zu visuellen 3D-Tracking-Systemen, die die 3D-Position des Objekts feststellen und aufzeichnen können, zählt auch Optitrack, welches in dieser Arbeit bei der Durchführung des Projektes zum Tracking verwendet wird.

Ein anderer Teilbereich der projektiven Geometrie ist Augmented Reality, auch AR genannt. Augmented Reality ist eine hochmoderne Technologie. Sie ist eine Variante von Virtual Reality. Bei Virtual Reality-Technologien kann der Benutzer die reale Welt um ihn herum nicht mehr wahrnehmen, da dieser vollständig in die virtuelle Umgebung eintaucht. Die Verwendung von Augmented Reality aber, gewährt dem Benutzer, im Gegensatz zu Virtual Reality, Einblick in eine Überlagerung der realen und der virtuellen Welt. Hierzu werden Objekte der virtuellen Welt in die reale Welt abgebildet, was den Benutzer glauben lässt, dass sich Objekte der virtuellen und der realen Welt im selben Raum befinden [3]. AR-Anwendungen werden aktuell in Bereichen wie Architektur, Ingenieurwesen, Bauwesen und Facility Management eingesetzt [4]. Hierbei wird mithilfe der Abbildung von Objekten der virtuellen Ebene auf die reale Ebene Visualisierungen von beispielsweise Bauobjekten ermöglicht. Dies kann helfen, das Benutzerverständnis zu verbessern, oder die Ausbildung zum Bedienern schwerer Geräte erleichtern [4]. Der Einsatz von AR-Technologien kann

aber auch im Bereich der Medizin Anwendung finden. Augmented Reality kann von Ärzten als Visualisierungs- und Trainingshilfe für Operationen genutzt werden [3]. Zu den genannten Einsatzgebieten kommen ständig neue hinzu, Wissenschaftler und Unternehmer entdecken die Technik für sich und adaptieren sie für völlig neue Anwendungsfälle. Für alle genannte Teilbereiche der projektiven Geometrie ist die Visualisierung von 3D zu 2D Punkten allenfalls ein notwendiger Bestandteil.

Hierbei spielt in erster Linie die Kamera und die Kalibrierung der Kamera eine Hauptrolle. Forscher setzen vielfältige Kameramodelle in Bereich der projektiven Geometrie ein. Bewährte Modelle sind neben der in dieser Arbeit eingesetzten Pinhole-Kamera beispielsweise Fisheye Lens Kameras [5] oder Katadioptrische Kameras [6]. Das Modell der Pinhole – oder auch Lochkamera besitzt den Vorteil, die Abbildung von 3D-Punkten der Szene auf die 2D-Bildebene mit Hilfe der projektiven Geometrie linear zu beschreiben. Viele Probleme, wie beispielsweise die Kamerakalibrierung, können durch diesen Ansatz mit Standardverfahren der linearen Algebra gelöst werden [7]. Angesichts der Kamerakalibrierung gibt es wiederum diverse Verfahren der Kalibration, wie beispielsweise nicht lineare Kalibrierungsmethoden [5,6], in der vorliegenden Arbeit kann aufgrund des simplen Pinhole-Kameramodells eine lineare Kalibrierungsmethode durchgeführt werden.

In dieser Arbeit wird die Visualisierung von 3D zu 2D Punkten behandelt. Es soll veranschaulicht werden, wie es möglich ist, die durch Visual tracking Systeme aufgenommene Trajektorien, mit der realen Welt zu überlagern. Hierzu wird das Visual tracking System Optitrack verwendet. Optitrack verfügt über Motion-Capture-Software, sowie über Hochgeschwindigkeits-Tracking-Kameras. Dies ermöglicht es, das Tracking während des Drohnenfluges durchzuführen und eine Trajektorie im Kamerabild live zu zeigen. Motion Capture wird der Vorgang genannt, bei welchem eine Bewegung, die gerade live durchgeführt wird, in eine digitale Version ihrer selbst übersetzt wird [8]. Laut Menache [8] erfolgt diese digitale Aufzeichnung durch Festhalten von Punkten im dreidimensionalen Koordinatensystem, welche diese Bewegung repräsentieren. Damit dies gelingt, müssen Punkte vorhanden sein, an welchen sich das System orientieren kann, häufig werden diese Punkte „key points“ genannt. Wie Menache [8] es in seinem Werk beschreibt, gibt es vielerlei Möglichkeiten Bewegungen digital zu erfassen. Häufig werden Kameras verwendet, welche die Bewegung aus verschiedenen Perspektiven aufzeichnen und im Anschluss die key points zusammenführen. Motion Capture Systeme ermöglichen daher das Realtime Tracking mit unbegrenzter Anzahl von Tackingpunkten und mit gleichzeitig minimaler Fehlerquote [8].

Für die digitale Bewegungserfassung haben sich im Gegenzug zu Marker orientierten Tracking Ansätzen wie Optitrack unter anderem auch Ansätze wie ultra-wideband band (UWB) [9] basierte Methoden, oder Radar Tracking basierte Methoden [10]

zur Bewegungserfassung bewährt. Da aber in dem 150 Quadratmeter großen und zehn Meter hohen Gebäude, der Drohnenhalle der Alpen-Adria-Universität Klagenfurt unter anderem an kamerabasierter Navigation ohne GPS gearbeitet wird, wird für die vorliegende Arbeit, das in der Drohnenhalle implementierte System Optitrack zur Bewegungserfassung verwendet. Die Halle bietet zur möglichst genauen Erfassung der Lage einer Drohne eine Ausstattung mit 37 hochpräzisen Kameras, die mit Infrarot-LEDs jede Bewegung im Raum verfolgen können [11].

Die durch das Tracking entstehenden dreidimensionalen Koordinaten-Punkte sollen ins zweidimensionale umgewandelt werden und anschließend im Kamerabild mit der realen Welt überlagert werden. Der Vorgang dieser Visualisierung beschreibt den Forschungsbereich der vorliegenden Arbeit, woraus sich folgende Forschungsfrage ergibt: „Wie ist es möglich, vom Zustandsschätzer und Planer geplante, vom Regler geflogene und vom Schätzer geschätzte Pfade im Drohnenflug zu einem Zeitpunkt im Kamerabild direkt zu visualisieren?“

## 1.1 Zielsetzung

Das Ziel der vorliegenden Arbeit ist es zu verstehen und zu implementieren, wie sich durch eine an der Drohne angebrachte Kamera aufgenommene 3D Punkte in der realen Welt zu 2D Punkten in einem Bild direkt visualisieren lassen. Hierfür soll eine Trajektorie generiert, transferiert und auf ein Kamerabild projiziert werden. So kann eine Überlagerung der transferierten Trajektorie mit dem realen Kamerabild möglich sein. Die Trajektorie soll somit im Raum abgebildet und von allen Seiten per laufender Kamera inspizierbar sein. Um die Umsetzbarkeit zu demonstrieren, soll eine Anwendung entwickelt werden, welche, anstatt direkte Aufnahmen der Drohne von 3D Punkten in die 2D Ebene zu projizieren, vorerst vorhandenes Videomaterial als Ressource verwendet. Mittels der vorhandenen Aufzeichnungen soll zuerst eine Kalibration der Kamera erfolgen, der darauffolgende Schritt stellt die Projektion auf das Bild dar. Wird die Kalibrierung einmal vorgenommen, kann diese für die jeweilige Kamera immer wieder verwendet werden, vorausgesetzt die Kameraparameter bleiben dieselben. Im Anschluss soll ein Experiment in der Drohnenhalle den Prozess veranschaulichen.

## 1.2 Gliederung der Arbeit

In Kapitel 2 werden die Grundlagen für die Kalibrierung und die Projektion eingeführt, dieses Kapitel beinhaltet Informationen über die Kamera, die Kameraparameter, die Kalibrierung der Kamera und die Projektion von 3D zu 2D. Im Kapitel

3 wird die Methodik beschrieben, die Komponenten, die für das Projekt verwendet wurden, sowie die Durchführung des Projektes selbst, werden erklärt. Kapitel 4 enthält die Ergebnisse der in Kapitel 3 beschriebenen Methodik zur Durchführung des Projektes. In Kapitel 5 werden die Ergebnisse diskutiert. Kapitel 6 biete ein Fazit mit einer kurzen Zusammenfassung und einem Ausblick. Kapitel 7 bietet noch ein kurzes Nachwort zur Arbeit.

## 2 Grundlagen

Dieses Kapitel beinhaltet die theoretischen Grundlagen, die nötig sind, um eine Projektion von einem Punkt in der realen Welt auf die 2D Ebene zu ermöglichen. Anfänglich wird das Modell der Kamera besprochen, anschließend wird auf deren intrinsische Parameter, sowie die extrinsischen Parameter eingegangen, die für die Projektion der Wegpunkte auf das Kamerabild benötigt werden.

### 2.1 Kamera und Parameter

#### 2.1.1 Funktion der Kamera

Wie Bradski und Kaehler in „Learning OpenCV“ beschreiben, muss, um Kalibration und Projektion verstehen zu können, zuerst betrachtet werden, wie das Sehen eines Objektes durch eine Kamera funktioniert. Licht, welches ein strahlender Körper, wie beispielsweise die Sonne, auf ein Objekt wirft, wird teilweise von diesem Objekt reflektiert. Das reflektierte Licht, welches zur Kamera gelangt wird auf dem Imager gesammelt. Dies geschieht bei einer Lochkamera wie folgt: reflektiertes Licht strahlt auf eine Wand, welche ein winziges Loch aufweist. Strahlen, die das sich in der Mitte der Wand befindende Loch nicht treffen, werden blockiert, so werden laut Bradski und Kaehler nur die Strahlen am Imager gesammelt, welche durch das Loch strahlen. Da ein Lochkameramodell nur wenig Licht sammelt und somit für das Aufnehmen von Bildern oder Videos nicht optimal geeignet ist, verwenden Kameras Linsen. Durch die Verwendung von Linsen treten jedoch andere Geometrien, sowie Verzerrungen auf [12]. Durch die Kalibrierung der Kamera können Abweichungen von einfachen Pinhole-Modellen korrigiert werden.

#### 2.1.2 Kameramodell

Zunächst wird noch einmal auf das oben genannte Kameramodell, die Lochkamera, oder auch Pinhole-Kamera genannt, eingegangen. Wie bereits erwähnt, werden alle

Strahlen, die das Loch der Pinhole-Kamera nicht treffen, blockiert, was dazu führt, dass nur der einzelne Strahl, welcher dieses Loch trifft, auf die Abbildungsfläche „projiziert“ wird. Dadurch ist das Bild auf dieser Bildebene immer im Fokus und die Größe des Bildes ist relativ zum entfernten Objekt. Sie wird durch die Brennweite der Kamera vorgegeben, welche genau der Abstand von der Lochblende zum Bildschirm entspricht. [12]

### 2.1.3 Kameraparameter

In Abbildung 1 ist die Brennweite der Lochkamera, englisch „focal length“ als „ $f$ “ zu sehen. Zudem stellen  $Z$  den Abstand von der Kamera zum Objekt,  $X$  die Länge des Objekts, und  $x$  das Bild des Objekts auf der Abbildungsebene dar. [12]

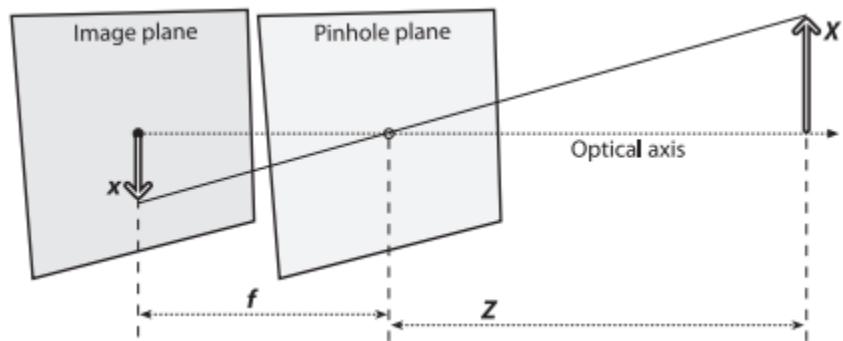


Abbildung 1: Brennweite [12]

Wie es in Abbildung 1 zu sehen ist, wird das Bild kopfüber projiziert. Um die Projektion nicht mehr um 180 Grad gespiegelt darzustellen und das Modell mathematisch zu vereinfachen, werden die Brennpunktebene und die Bildebene getauscht. Abbildung 2 veranschaulicht diesen Vorgang.

Der Punkt Q liegt, wie von Bradski und Kaehler beschrieben wird, aber oft nicht in der Mitte des Imagers, weil die einzelnen Pixel bei einem typischen Imager eher rechteckig als quadratisch sind. Um dies zu berücksichtigen, ist das Einführen zweier verschiedener Brennweiten  $f_x$  und  $f_y$  notwendig. Die Brennweiten  $f_x$  und  $f_y$  setzen sich aus deren physikalischer Brennweite ( $F$ ) und der Größe des Bildsensors ( $s_x$  und  $s_y$ ) zusammen. Zusätzlich wird das Modell noch um die Koordinaten des Zentrums des Bildsensors erweitert ( $c_x, c_y$ ). Das Ergebnis stellt einen Punkt Q in der physikalischen Welt, dessen Koordinaten ( $X, Y, Z$ ) sind, an einer Pixelposition, die durch  $(x_{screen}, y_{screen})$  gegeben ist, auf den Bildschirm projiziert dar [12]. Hierzu ergeben

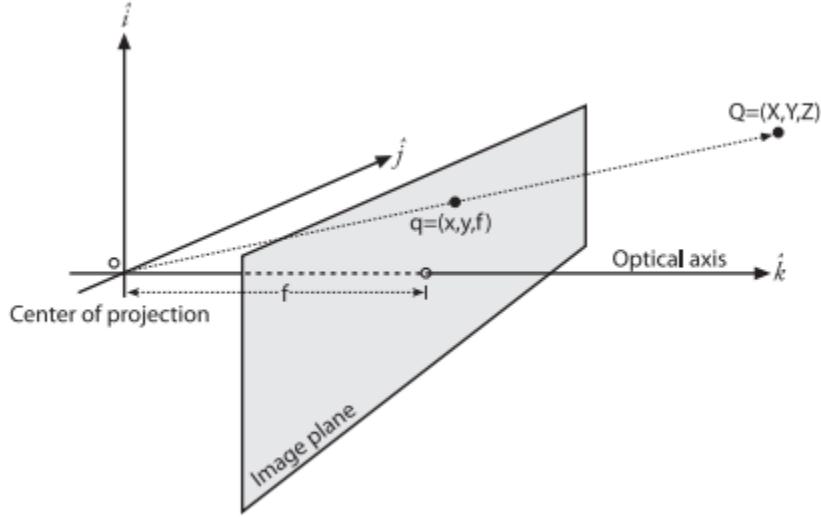


Abbildung 2: Vertauschen der Brennpunkt- und Bildebene [12]

sich folgende Gleichungen für die Koordinaten des abgebildeten Punkts:

$$x_{screen} = f_x \left( \frac{X}{Z} \right) + c_x, \quad y_{screen} = f_y \left( \frac{Y}{Z} \right) + c_y \quad (1a)$$

Um nun die Kameramatrix, die sogenannte intrinsische Matrix zu erstellen, werden statt der berechneten Koordinaten üblicherweise homogene Koordinaten verwendet. Daraus ergibt sich ein dreidimensionaler Vektor  $q = (q_1, q_2, q_3)$ . Das Resultat lässt sich wie folgt darstellen [12]:

$$q = MQ, \quad \text{mit} \quad q = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad M = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \quad Q = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (2)$$

Bradski und Kaehler beschreiben die Lochkamera deshalb als „nicht optimal“, da durch das Loch nur wenig Licht eintreten kann und daher das Entstehen des Bildes lange dauert. Um ein langes Warten zu verhindern, muss mehr Licht über eine größere Öffnung eintreffen können. Das kann durch die Verwendung einer Linse erreicht werden. Jedoch bietet die Verwendung einer Linse nicht nur eben genannte Vorteile, sondern leider auch den Nachteil auftretender Verzerrungen [12].

## 2.2 Verzerrungskoeffizienten

Zahlreiche Verzerrungskoeffizienten können beim Einsatz einer Linse auftreten und je nach eingesetzter Linse variieren. In Bezug auf die vorliegende Arbeit genügt es aber, die zwei häufigsten Arten der Verzerrung zu diskutieren, die radiale Verzerrung und die tangentiale Verzerrung. Hierbei kann zwischen den beiden Typen der Verzerrung wie durch OpenCV beschrieben, unterschieden werden: Die radiale Verzerrung, welche gerade Linien gekrümmmt erscheinen lässt und mit zunehmender Entfernung von der Bildmitte wächst. In den OpenCV Camera Calibration Tutorials lässt sich nachlesen, dass radiale Verzerrung der x- und y-Koordinaten auch wie folgt beschrieben werden kann [13]:

$$x_{distorted} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (3)$$

$$y_{distorted} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (4)$$

$$\text{mit } r^2 = x'^2 + y'^2 \quad (5)$$

$$\text{und } \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} X_c/Z_c \\ Z_c/Z_c \end{pmatrix} \quad (6)$$

Wobei  $X_c$ ,  $Y_c$  und  $Z_c$  die dreidimensionalen Kamerakoordinaten darstellen. Auf die Berechnung der dreidimensionalen Kamerakoordinaten wird im Kapitel „Projektion“ genauer eingegangen.

Die zweite Art der Verzerrung, die häufig auftritt, ist die tangentiale Verzerrung. Sie kann bewirken, dass einige Bereiche des Bildes weiter entfernt scheinen als andere, da das Aufnahmegerät oft nicht perfekt parallel zur Abbildungsebene ausgerichtet ist. OpenCV schlägt folgende Darstellung der tangentialen Verzerrung vor [13]:

$$x_{distorted} = x + (2p_1xy + p_2(r^2 + 2x^2)) \quad (7)$$

$$y_{distorted} = y + (p_1(r^2 + 2y^2) + 2p_2xy) \quad (8)$$

Nachdem die beiden relevanten Verzerrungen bekannt sind, können diese zu einem Vektor zusammengefasst werden, welcher aus 5 Komponenten besteht und auch häufig „Distortion Coefficients“ genannt wird. Dargestellt werden dieses 5 Parameter laut OpenCV wie folgt [13]:

$$Distortion coefficients = (k_1 \ k_2 \ p_1 \ p_2 \ k_3) \quad (9)$$

Um beim Kalibrieren der Kamera die Verzerrungskoeffizienten ausfindig zu machen und diese zu korrigieren, müssen noch weitere Parameter bekannt sein. Für die Kalibrierung fehlen noch das Verwenden eines wohldefinierten Musters, welches das Festlegen spezifischer Punkte in der realen Welt ermöglichen soll, sowie die extrinsische Matrix, welche die Rotations- und Translationskoeffizienten beinhaltet. Auf die Rotations- und Translationskoeffizienten wird im nächsten Kapitel genauer eingegangen.

## 2.3 Rotations- und Translationskoeffizienten

Rotations- und Translationskoeffizienten werden benötigt, um die Beziehung zum Kamerabild in den Koordinaten der realen Welt zu definieren. Die geometrische Beziehung zwischen einem Punkt in den Koordinaten der realen Welt und seiner Entsprechung im Bild in Pixeln hängt von der Starrkörperbewegung der Kamera ab [14].

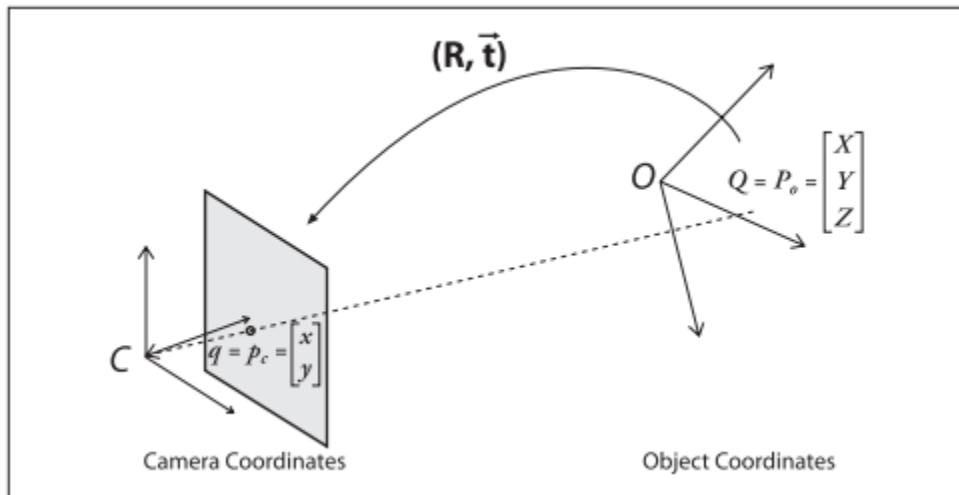


Abbildung 3: Rotation and Translation [12]

Eine Rotationsmatrix ist eine quadratische Matrix, die beispielsweise im Zweidimensionalen eine Größe von 2x2 und im Dreidimensionalen eine Größe von 3x3

aufweist. Für jede vorhandene Koordinate (x,y oder x,y,z) ist wiederum eine eigene Matrix definiert. Rotationsmatrizen für den dreidimensionalen Raum werden wie folgt beschrieben [12]:

$$R_x(\psi) \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \psi & \sin \psi \\ 0 & -\sin \psi & \cos \psi \end{pmatrix} \quad (10)$$

$$R_y(\varphi) \begin{pmatrix} \cos \varphi & 0 & -\sin \varphi \\ 0 & 1 & 0 \\ \sin \varphi & 0 & \cos \varphi \end{pmatrix} \quad (11)$$

$$R_z(\vartheta) \begin{pmatrix} \cos \vartheta & \sin \vartheta & 0 \\ -\sin \vartheta & \cos \vartheta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (12)$$

Wird nun eine Rotation des Koordinatensystems um den Winkel verursacht, ist dies mit einer Gegendrehung des Zielpunktes um den Ursprung dieses Koordinatensystems um denselben Winkel gleichzusetzen. Die gesamte Rotationsmatrix R ergibt sich aus dem Produkt der oben angeführten Matrizen  $R_x(\psi)$ ,  $R_y(\varphi)$  und  $R_z(\vartheta)$  [12].

Der Translationsvektor beschreibt eine örtliche Verschiebung des Koordinatensystems, mit anderen Worten ist dieser der Versatz vom Ursprung des Koordinatensystems zum Ursprung des anderen Koordinatensystems. Für besseres Verständnis wird hier noch einmal auf Abbildung 3 verwiesen. Dort ist zu erkennen, dass der Translationsvektor der Subtraktion des Kameraursprungs vom Objektursprung entspricht. [12] Fügt man nun das Wissen über Rotation und Translation zusammen, so kann man eine Transformation von Welt- zu Kamerakoordinaten wie folgt beschreiben:

$$P_c = R(P_o - T) \quad (13)$$

$P_c$  stellt die Kamerakoordinaten dar, R ist die Rotation, T die Translation und  $P_o$  stellt die Weltkoordinaten dar.

Da in dieser Arbeit ein besonderes Setup verwendet wird, sollen dieses gemeinsam mit dessen Koordinatensystemen folgend genauer beschrieben werden. Abbildung 3

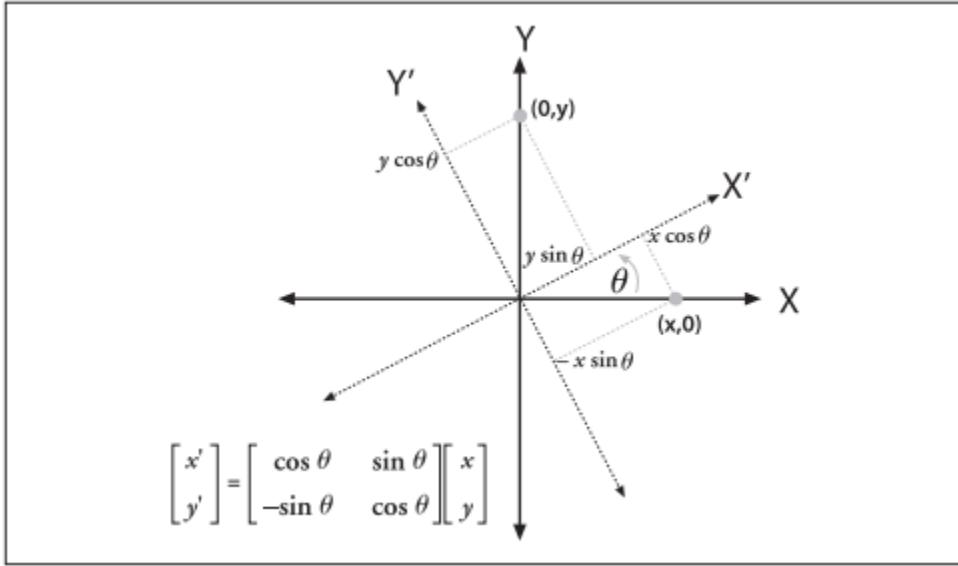


Abbildung 4: Veränderung der Koordinaten durch Rotation in 2D [12]

zeigt die Situation, in welcher der Punkt im Weltsystem ist und dieser zuerst in das Kamera-System transformiert werden muss. Tatsächlich wird in der vorliegenden Arbeit der Punkt aber erst in das Optitrack-Objektsystem, dann ins IMU-System, und dann das Kamera-System transformiert, wie es in Abbildung 5 zu sehen ist. Der Zwischenschritt über das Optitrack-Objekt ist notwendig, da das Tracking System nur die reflektierenden Marker erkennt, nicht das eigentliche Kamera-System direkt. Der Zwischenschritt über die IMU wird verwendet, da Kalibr die IMU-Kamera Transformation berechnet und die Optitrack-IMU Transformation leicht anderweitig, nämlich über Winkelgeschwindigkeiten, berechenbar ist.

In Abbildung 5 stellen  $R_w$  und  $T_w$  die Rotation und Translation der Weltkoordinaten zu den Opitrack-Objektsystem Koordinaten dar,  $R_o$  und  $T_o$  sollen die Rotation und Translation der Opitrack-Objektsystem Koordinaten zu den Koordinaten der IMU darstellen.  $R_i$  und  $T_i$  sollen schlussendlich die Rotation und Translation der IMU Koordinaten zu den Koordinaten des Kamerasytems darstellen.

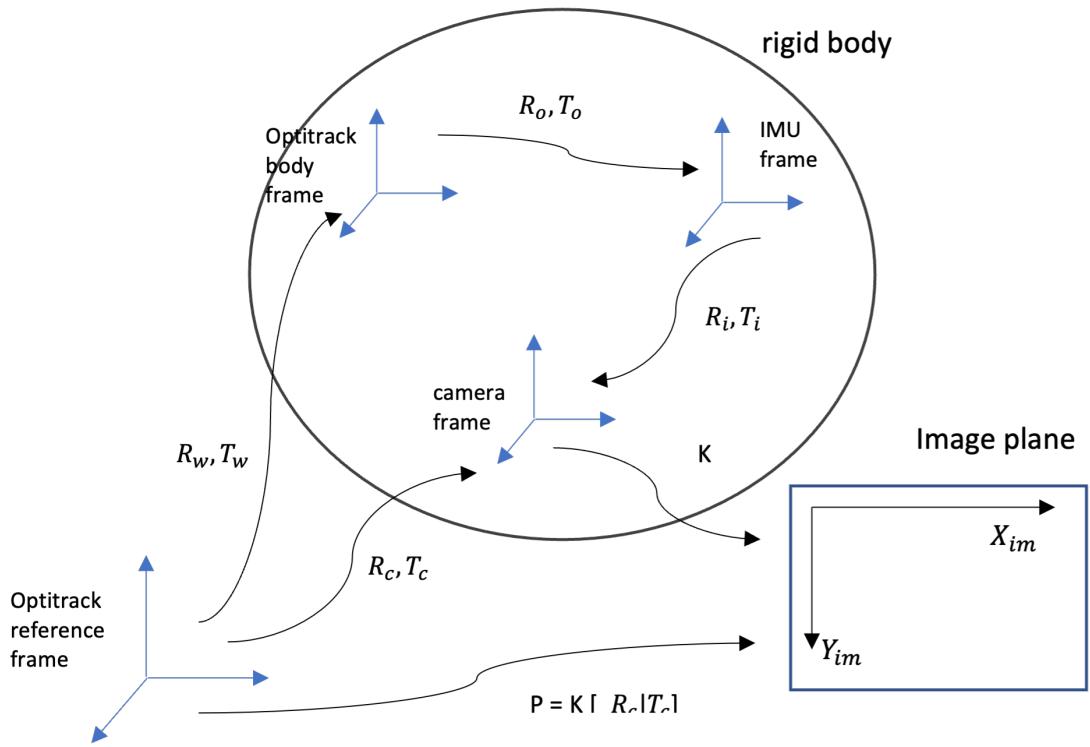


Abbildung 5: Setup und Koordinatensysteme

## 2.4 Kalibrierung

Nach Aufklärung über verwendete Matrizen muss nun noch definiert werden, woher die für die Kalibrierung notwendigen Parameter stammen. Wie Sadekar und Mallick in ihrem Post schreiben, wird bei der Kalibrierung schrittweise vorgegangen. Mithilfe einer Sammlung von Bildern, deren 2D-Bildkoordinaten und 3D-Weltkoordinaten bekannt sind, sollen die intrinsischen und extrinsischen Parameter ausfindig gemacht werden [15]. In folgenden Kapiteln werden die Kalibrierungsmethode und der Vorgang der Kalibrierung beschrieben.

### 2.4.1 Kalibrierungsmethode

Soll eine Kamerakalibration durchgeführt werden, so wird üblicherweise eine Sammlung von Bildmaterial mit „Ecken“ oder „Punkten“ verwendet, deren zweidimensionale und dreidimensionalen Koordinaten bekannt sind [15]. Je nach verwendetem Material gibt es verschiedene Ansätze und Methoden der Kamerakalibration. Derzeit

unterstützt OpenCV drei Arten von Objekten für die Kalibrierung: ein klassisches schwarz-weißes Schachbrett, ein symmetrisches Kreismuster und ein asymmetrisches Kreismuster. Unabhängig davon, welches Objekt gewählt wird, müssen mit der zu kalibrierenden Kamera Aufnahmen des Musters gemacht werden. Das Muster soll von verschiedenen Positionen aufgenommen werden, zudem sollen genügend Aufnahmen vorhanden sein, um ein gutes Ergebnis zu erzielen [16].

In dieser Arbeit wird ein Schachbrett zur Kalibrierung der Kamera verwendet. Schachbrettmuster werden deshalb häufig für diesen Vorgang verwendet, da sie eindeutig sind. Ebenso als Vorteil erweisen sich die Ecken der Quadrate (oder Rechtecke) des Bretts, welche besonders leicht zu lokalisieren sind [15]. Beim Kalibrieren mit einem Schachbrettmuster werden mehr Aufnahmen benötigt als bei der Verwendung anderer Methoden. Grundsätzlich wäre hierbei eine Kalibrierung mit zwei Aufnahmen schon möglich, für ein zufriedenstellendes Ergebnis jedoch, werden rund 10 Bilder des Schachbrettmusters empfohlen [16].

## 2.5 Kalibrieren der Kamera

In diesem Kapitel erfolgt die Kalibrierung, wie von Sadekar und Mallick [15] empfohlen. Bekannt sind anfänglich nur die Maße des Schachbretts. Aus den Maßen des Schachbrettes sollen nach Abschluss der Kalibrierung die Kameraparameter, welche intrinsische Matrix genannt werden, die Verzerrungskoeffizienten, sowie die extrinsischen Parameter, also die Rotations- und Translationskoeffizienten gefunden werden.

Zuerst müssen die 3D Weltkoordinaten festgestellt werden. Hierzu werden, wie bereits im vorherigen Kapitel beschrieben, Aufnahmen des Schachbrettmusters, dessen Maße bekannt sind, benötigt. Die x-Achse und die y-Achse stellen die Länge beziehungsweise die Breite des Musters dar. Die z-Achse ist null, da eine Aufnahme eines Bildes durch eine Kamera nur zwei Dimension hat. Die Weltkoordinaten sind nun an das Schachbrett gebunden und alle Eckpunkte liegen auf der x-y-Ebene. Aufgrund dessen lassen sich die Koordinaten jedes 3D-Punktes leicht definieren. Es wird ein Koordinatenursprung festgelegt, in Bezug auf diesen Referenzpunkt können nun alle weiteren Punkte definiert werden [16]. Im nächsten Schritt beschreiben Sadekar und Mallick [15] das Aufnehmen mehrerer Bilder mit der Kamera. Ob die Kamera statisch ist und das Schachbrett bewegt wird oder umgekehrt, spielt keine große Rolle. Wichtig ist, dass das Schachbrett von verschiedenen Standpunkten aus aufgenommen wird, zudem sollen genügend Aufnahmen gemacht werden (bestenfalls rund 10 Bilder). Wenn nun genügend Aufnahmen vorhanden sind und die 3D-Koordinaten der Punkte im Schachbrettmuster bekannt sind, werden nur noch die 2D-Koordinaten benötigt.



Abbildung 6: Schachbrettmuster [13]

Die Punkte, deren 3D-Koordinaten bekannt sind, sind jeweils die Ecken der einzelnen Quadrate im Muster [16].

Das Kalibrieren der Kamera ist einerseits im Code möglich, wofür OpenCV eingebaupte Funktionen bietet. Hierfür werden zuerst die Funktion „findChessboardCorners“ zur Verfügung gestellt, welche ein Schachbrett erkennt und die Koordinaten dessen Eckpunkte ausfindig macht. Danach werden die 3D-Punkte in Weltkoordinaten und ihre 2D-Positionen in allen Bildern an die „calibrateCamera“-Methode von OpenCV übergeben, welche schlussendlich die Kameramatrix, Verzerrungskoeffizienten und Rotations- und Translationskoeffizienten bestimmt [16].

Für die vorliegenden Arbeit wird für das Kalibrieren der Kamera jedoch ein anderes Tool verwendet, welches aber grundsätzlich ähnlich funktioniert. Das Tool wird „Kalibr Docker Image“ genannt und soll auf möglichst einfachem Weg die intrinsischen und extrinsischen Parameter eines Kamerasytems generieren.

Da die Kalibrierung nur einmal pro Kamera durchgeführt werden muss, ist es sinnvoll, die entstandenen Parameter nach erfolgreicher Kalibrierung zu speichern. Auf diese Weise können diese Werte später einfach in das Programm geladen werden. Daher wird üblicherweise zuerst die Kalibrierung durchgeführt und nach Erhalt der Parameter werden diese händisch in ein YAML File geschrieben und abgespeichert. Der Vorgang der Projektion wird in den folgenden Kapiteln der Arbeit erläutert.

## 2.6 Projektion

Das Ziel der vorliegenden Arbeit ist es darzustellen und zu implementieren, wie sich durch eine an der Drohne angebrachte Kamera aufgenommene 3D Punkte in der realen Welt zu 2D Punkten in einem Bild direkt visualisieren lassen. Dazu muss mit den intrinsischen und extrinsischen Parametern, sowie den Verzerrungskoeffizienten der vorherigen zwei Kapiteln gearbeitet werden. Benötigt werden die Kamera-Matrix (intrinsische Matrix), die Verzerrungskoeffizienten und die Rotationsmatrix und den Translationsvektor (extrinsische Matrix). Damit zu den dreidimensionalen Koordinaten passende zweidimensionale Koordinaten im Bild berechnet werden können, sind Matrixmultiplikationen notwendig. Um diese durchführen zu können, muss eine homogene Transformationsmatrix eingeführt werden.

### 2.6.1 Homogene Transformationsmatrix

In der projektiven Geometrie werden häufig homogene Koordinaten verwendet, da diese im Vergleich mit Kartesischen Koordinaten die Darstellung und Berechnung mathematischen Formeln sehr vereinfachen [17].

Soll nun das vorliegende dreidimensionale Koordinatensystem in homogene Koordinaten transformiert werden, so kann die Matrix wie folgt durch einen vierten Wert erweitert werden. Die Erweiterung erfolgt mit einer 1 [18].

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \rightarrow \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \quad (14)$$

Wird hingegen eine zweidimensionale Matrix verwendet, so werden die Werte x- und y-Achse durch den Wert der z-Achse dividiert [18].

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \rightarrow \begin{pmatrix} X/W \\ Y/W \end{pmatrix} \quad (15)$$

Um nun die Rotations- und Translationskoeffizienten, die in der extrinsischen Matrix zusammengefasst werden, als homogene Transformationsmatrix darstellen zu können, wird die Matrix wie folgt erweitert [18]:

$$(R|t) = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (17)$$

## 2.7 perspektivische Transformation

Die perspektivische Transformation stellt einen 3D-Punkt der realen Welt als Pixel im Bild dar. Wie es von Zhang [18] beschrieben wird, kann die projektive Transformation wie folgt dargestellt werden:

$$sp = A[(R|t)]P_w \quad (18)$$

Im Bezug auf die angegebene Formel ist s die Skalierung der projektiven Transformation, welche in häufigen Fällen nicht berücksichtigt wird, so auch in der vorliegenden Arbeit. A ist die Kamera Matrix, die intrinsische Matrix, p ist ein 2D-Pixel in der Bildebene, R und t sind Rotations- und Translationskoeffizienten, die die Änderung der Koordinaten von Welt zu Kamera beschreiben und  $P_w$  ist ein 3D-Punkt mit Koordinaten der realen Welt [17].

Das zweidimensionale Pixel p auf der Bildebene lässt sich durch die Multiplikation der intrinsischen Kameramatrix mit 3D-Punkten im Kamerakoordinatensystem berechnen [17]. Die Kameramatrix ist nach Durchführen der Kalibrierung bekannt und kann direkt als diese verwendet werden. Der Pixel-Punkt wird, genau wie die intrinsische Kameramatrix um den Wert 1 erweitert, um mit der dreidimensionalen Matrix des Kamerakoordinatensystems arbeiten zu können [18].

$$A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad p = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (19)$$

Daraus ergibt sich die Multiplikation der Matrizen wie folgt [18]:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \quad (20)$$

Die Koordinaten  $X_c$ ,  $Y_c$  und  $Z_c$  des Kamerakoordinatensystems sind noch unbekannt, jedoch ist die Berechnung deren Werte mittels der 3D-Koordinaten des Punktes in der realen Welt und mittels Rotations- und Translationskoeffizienten, die ebenfalls nach der Kalibrierung bekannt sein sollten, möglich. Werden Rotations- und Translationskoeffizienten gemeinsam in eine 3x4 Matrix gefüllt, so ergibt sich die Transformationsmatrix. Wird diese wie in Kapitel 2.6.1 beschrieben, erweitert, entsteht die 4x4 homogene Transformationsmatrix, mit welcher die 3D-Punkte des Kamerakoordinatensystems berechnet werden können.

$$P_c = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} P_w \quad (21)$$

*oder*

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (22)$$

Hier wird, wie in OpenCV [17] beschrieben, die Multiplikation eines Punktes der realen Welt mit der homogenen Transformationsmatrix, was das Ergebnis der Kamerakoordinaten zur Folge hat, dargestellt.

Die Herleitung der Formel beinhaltet also folgende Schritte: Einsatz von bekannten 3D Weltkoordinaten, durch die mithilfe von Rotations- und Translationskoeffizienten die 3D Kamerakoordinaten bestimmt werden können, sowie das Erzeugen der Pixelkoordinaten durch die Multiplikation der intrinsischen Kameramatrix mit den Kamerakoordinaten. Die perspektivische Transformation kann, wie in OpenCV [17] demonstriert, in einer gesamten Formel veranschaulicht werden.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (23)$$

Sollen nun noch die Pixelkoordinaten zweidimensional dargestellt werden, kann dies, wie in Kapitel 3.1 beschrieben mittels Division der x- und der y-Achse durch die z-Achse erreicht werden.

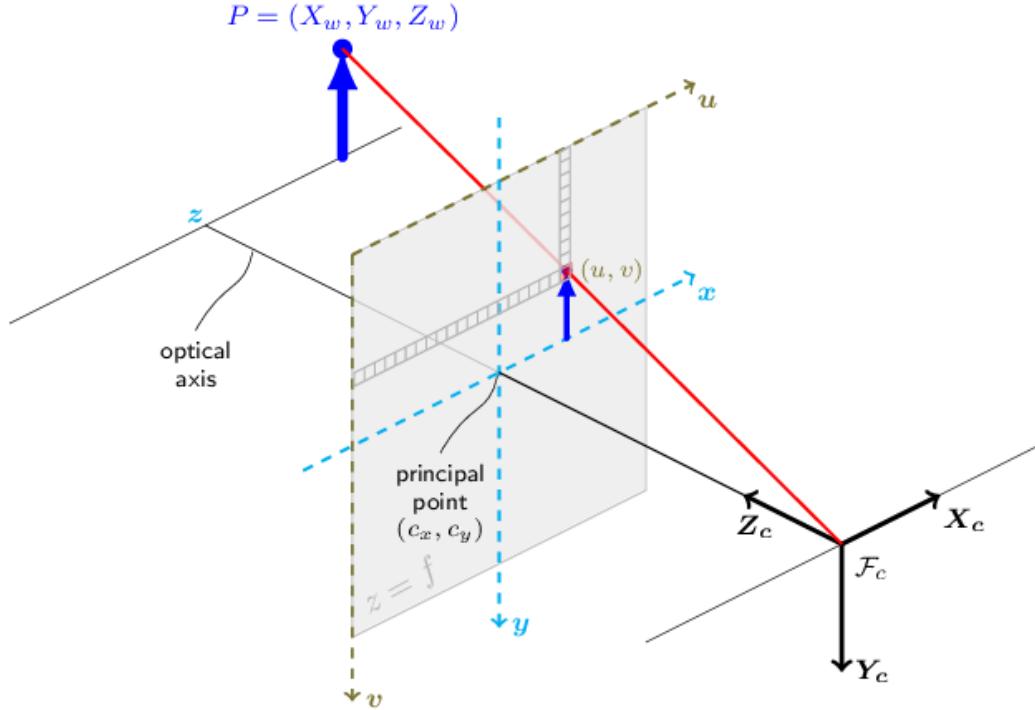


Abbildung 7: Lochkameramodell [17]

### 2.7.1 Entfernen der Verzerrung

Beim Verwenden einer Linse treten oftmals Verzerrungen des Bildes auf, wie sie schon in Kapitel 2.2 besprochen wurden. Um Verzerrungen zu entfernen, werden die bei der Kalibrierung der Kamera entstehenden Verzerrungskoeffizienten berücksichtigt. [19] schlägt vor, das Modell wie folgt zu erweitern:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{pmatrix} \quad (24)$$

Dabei lassen sich die Parameter x“ und y“ durch den Einsatz der Verzerrungskoeffizienten berechnen [19].

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} x^{\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}} & +2p_1x'y' + p_2 & (r^2 + 2x'^2) & +s_1r^2 + s_2r^4 \\ y^{\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}} & +p_1 & (r^2 + 2y'^2) & +2p_2x'y' + s_3r^2 + s_4r^4 \end{pmatrix} \quad (26)$$

mit

$$r^2 = x'^2 + y'^2 \quad (27)$$

und

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} X_c/Z_c \\ Y_c/Z_c \end{pmatrix} \quad (28)$$

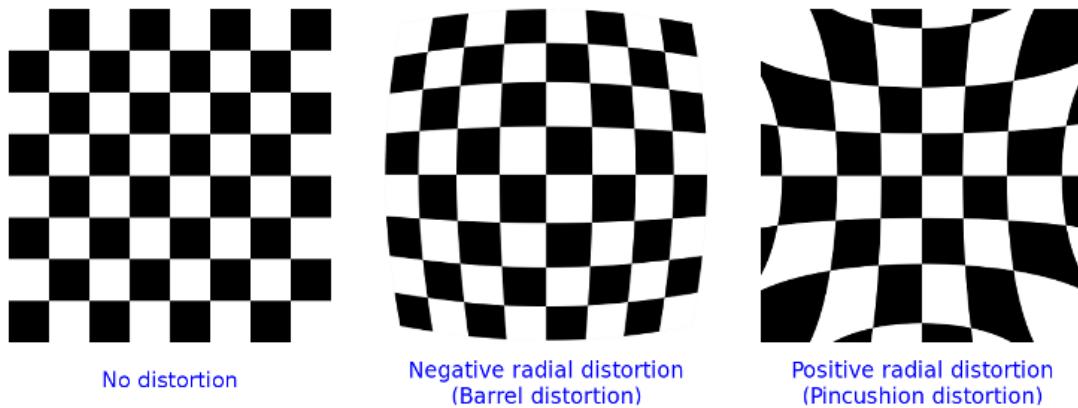


Abbildung 8: Beispiel für Verzerrung [17]

## 2.8 Projektion einer Trajektorie

Im vorherigen Kapitel wird veranschaulicht, wie ein einzelner Punkt der realen Welt auf ein Kamerabild projiziert werden kann. Ziel der Arbeit ist es, eine Trajektorie, also eine Sammlung beliebig vieler Punkte der realen Welt auf die zweidimensionale Ebene abbilden zu können. Soll also eine bereits generiert Trajektorie im Kamerabild projiziert werden, so kann eine Datei, welche die 3D Punkte der Trajektorie beinhaltet, im Code eingelesen werden. Da Kameramatrix, Verzerrungskoeffizienten, Rotation- und Translationskoeffizienten bereits für die Projektion eines einzelnen Punktes bekannt sind, können diese auch für die Projektion mehrerer Punkte verwendet werden. So kann die dreidimensionale Trajektorie mittels bekannter Parameter ins Zweidimensionale konvertiert und auf das Kamerabild projiziert werden.

## 3 Methodik

Für das durchgeführte Experiment wurden Ubuntu 20.04 Virtualbox als Betriebssystem verwendet. Das System Robotic Operating System (ROS) wurde eingesetzt, um die Kommunikation zwischen Prozessen zu verwirklichen. Zusätzlich wurde Open Computer Vision, OpenCV als Open-Source-Softwarebibliothek genutzt. Das Programm MATLAB wurde für das Generieren der Trajektorien eingesetzt. Anschließend werden die verwendeten Komponenten erklärt und die einzelnen Schritte des Versuchsaufbaus von der Kalibrierung der Kamera über die Herstellung einer Trajektorie, bis hin zur Punktprojektion auf dem Kamerabild erläutert.

### 3.1 Ubuntu 20.04 Virtualbox

Für die Umsetzung des Projektes wurde das Betriebssystem Ubuntu 20.04 Virtualbox verwendet. Laut [20] ist VirtualBox ein leistungsstarkes x86- und AMD64 / Intel64-Virtualisierungsprodukt für den privaten und privaten Gebrauch. Es gewährt dem Host-Betriebssystem keinen direkten Zugriff auf die Hardware des verwendeten Computers. Alternativ werden hardwarebezogene Anweisungen über die Treiber des Host-Betriebssystems von Virtualbox weitergeleitet. Dies ist machbar, da VirtualBox ein "GastBetriebssystem in einem Fenster des Host-Betriebssystems ausführen kann. Durch die Verwendung von VirtualBox ist es möglich, alle Versionen von Windows, Linux und viele andere x86- und AMD / Intel-basierte 32- und 64-Bit-Betriebssysteme entweder als Host-Betriebssystem oder als Gast-Betriebssystem auszuführen. Ubuntu 20.04 arbeitet mit dem Linux-Kernel 5.4 und unterstützt neben den gängigen Linux-Dateisystemen wie Ext4, Btrfs und ReiserFS auch ZFS-Partitionen.

Standard ist der Ubuntu-Desktop, der auf Gnome basiert. Ausgestattet ist Ubuntu 20.04 mit einer Auswahl an gängigen Programmen wie LibreOffice, Firefox und Thunderbird, die sich durch die Software-Verwaltung beliebig erweitern lässt [21].

### 3.2 ROS

Robotic Operating System (ROS) wurde im Projekt verwendet, um die Kommunikation zwischen Prozessen zu verwirklichen. ROS ist ein Open Source, Meta-Betriebssystem und stellt unter anderem folgende Dienste zur Verfügung: Hardwareabstraktion, Gerätetreiber, Utilityfunktionen, Interprozesskommunikation und Paketmanagement und läuft zurzeit nur auf Unix-basierten Plattformen. Des Weiteren sind Werkzeuge und Bibliotheken für das Beziehen, Builden, Schreiben und Ausführen von Code über mehrere Computer vorhanden. ROS ist kein Echtzeit Framework, obwohl es möglich ist, ROS mit Echtzeitkomponenten zu vereinen. Der Laufzeitgraph, über welchen ROS verfügt, ist ein Peer-to-Peer Netzwerk von Prozessen, welche via die ROS Kommunikationsinfrastruktur lose gekoppelt sind. Die synchrone Kommunikation über Services, asynchrones Streamen über Topics, sowie Datenspeicher auf dem Parameter Server sind Beispiele für Kommunikationsarten, die ROS zur Verfügung stellt.

Robotic Operating System ist ein verteiltes System von Prozessen, die „Nodes“ genannt werden. Dadurch wird die lose Kopplung von individuellen Komponenten ermöglicht. Durch die Möglichkeit des Zusammenschlusses von Code Repositories, ermöglicht das System die Zusammenarbeit über verteilte Infrastrukturen, sowie unabhängige Entscheidungen bezüglich Entwicklung und Implementierung. [22].

### 3.3 OpenCV

Open Source Computer Vision Library, OpenCV, ist eine Open-Source-Softwarebibliothek für Computer Vision und maschinelles Lernen und verfügt über rund 2500 optimierte Algorithmen. Diese umfassen sowohl klassische als auch hochmoderne Computer Vision- und Machine Learning-Algorithmen. Genutzt wird OpenCV hauptsächlich, um Objekte zu identifizieren, Gesichter zu erkennen, Kamerabewegungen oder sich bewegende Objekte zu verfolgen, 3D-Modelle von Objekten zu extrahieren, 3D-Punktwolken aus Stereokameras zu erzeugen, aber auch um Bilder zusammenzufügen, rote Augen aus Bildern zu entfernen, die mit Blitz aufgenommen wurden und sogar zum Erkennen von Landschaften und Erstellen von Markierungen, um sie mit Augmented Reality zu überlagern [23]. Die Open Source Computer Vision Library unterstützt die gängigsten Betriebssysteme, wie Linux, Android, Windows und Mac

OS, zudem verfügt OpenCV über Schnittstellen, wie C++, Python, Java und MATLAB. Die Open-Source-Softwarebibliothek für Computer Vision basiert hauptsächlich auf Echtzeit-Bildverarbeitungsanwendungen und verwendet MMX- und SSE-Befehle, sofern verfügbar. OpenCV ist lokal in C++ geschrieben und verfügt über eine voreingebaute Schnittstelle, die nahtlos mit STL-Containern zusammenarbeiten kann [23].

### 3.4 Kalibrierung der Kamera

Die Umsetzung des Experiments begann mit der Kalibrierung der Kamera. Ziel der Kamera-Kalibrierung war es, die intrinsischen Kameraparameter, die extrinsischen Kameraparameter und die Verzerrungskoeffizienten für die verwendete Kamera ausfindig zu machen. Im durchgeführten Experiment wurde für die Kalibrierung eine Kamera des Modells „pinhole-radtan“ mit der Auflösung von 2056x1542 verwendet, „pinhole“ bedeutet, dass es sich um eine Lochkamera handelt, „radtan“, dass mit radialen und tangentialen Verzerrungskoeffizienten gearbeitet wird. Für die Umsetzung der Kalibration wurde ein Tool verwendet, welches sich Kalibr Docker Image nennt. Dieses Tool wurde eingesetzt, um zu vermeiden, dass individuell alle Library dependencies installiert werden müssen, oder Software Versionen inkompatibel werden (z.B. OpenCV in Ubuntu 20.04). Dieser Docker Container bot somit die Möglichkeit, ein Virtuelles System lediglich mit einer Kalibr Installation zu verwenden. Um es einfacher zu machen, Daten zwischen dem Host System und dem Docker Container auszutauschen, wurde Volume-Mapping verwendet. Damit konnten alle Daten aus dem Ordner, in dem der Docker ausgeführt wurde, im Ordner innerhalb des Docker Containers gefunden werden. Diese Daten wurden Bi-Direktional im Container und auf dem Host System geändert. Zusätzlich wurden noch ein Schachbrettmuster, sowie ein BAG-File mit Videomaterial, auf welches später auch projiziert werden kann, verwendet. Folgend konnten mit dem Kalibr Tool durch Ausführen von Befehlen auf der Kommandozeile zuerst die intrinsischen und dann die extrinsischen Parameter generiert und automatisch in einem File abgespeichert werden, aus welchem die entstandenen Parameter im folgenden Schritt zu entnehmen sind. Die entnommenen Parameter wurden händisch in ein YAML-File übertragen, welches später im Code eingelesen werden konnte.

### 3.5 Generieren einer Trajektorie mittels MATLAB

Soll ein einzelner Punkt der realen Welt auf das Kamerabild abgebildet werden, so kann dieser im Code selbst generiert werden. Dieser Ansatz eignete sich gut, um

Tests durchzuführen. Soll jedoch eine Trajektorie auf das Videomaterial projiziert werden, ist es sinnvoll, diese Sammlung beliebig vieler Punkte der realen Welt zuerst zu generieren und dann einzulesen.

In der vorliegenden Arbeit wurde für das Generieren einer Trajektorie das Programm MATLAB verwendet. MATLAB ist eine Plattform für Programmierung und numerische Berechnungen. Für die jeweils durchgeführten Tests wurde eine Trajektorie generiert, welche ein spiralförmiges Fliegen der Drohne in Raum von unten nach oben veranschaulichen soll. Für die beiden Testtrajektorien wurden einmal einhundert und einmal eintausend Wegpunkte generiert. Die Sammlung der Wegpunkte der beiden Trajektorien wurden jeweils in ein CSV File gespeichert, welches die Koordinaten der x-, y- und z-Achse der generierten Punkte zeilenweise beinhaltete.

### 3.6 ROS Package

Für den praktischen Teil der Arbeit wird das Open Source, Meta-Betriebssystem ROS Noetic Ninjemys verwendet. Der Workspace, in welchem gearbeitet wurde, nennt sich ROS Package. In diesem Package befinden sich folgende Bestandteile, welche alle für die Durchführung des Experiments notwendig sind:

- Nodes: Ein Node ist eine ausführbare Datei, die ROS verwendet, um mit anderen Nodes zu kommunizieren [24]. Diese sind im vorliegenden Experiment CPP Dateien im Source-Ordner, welche das Hauptprogramm und das Programm für die Projektion sind, oder auch die Header-Files, die sich im Include-Ordner befinden. Sie beinhalten Initialisierungen und binden benötigte Libaries ein.
- Messages: Ein ROS-Datentyp, der beim Abonnieren oder Veröffentlichen eines Topics verwendet wird [24]. Im Experiment die Datei, die für Messages verwendet wird, befindet sich im Messages-Ordner, sie beinhaltet lediglich den Datentyp der kommunizierten Daten.
- Topics: Nodes können Messages zu einem Topic veröffentlichen, sowie ein Topic abonnieren, um Messages zu empfangen. Im Allgemeinen wissen Nodes nicht, mit wem sie kommunizieren. Stattdessen abonnieren Nodes, die an Daten interessiert sind, das relevante Topic. Nodes, die Daten generieren, veröffentlichen zum relevanten Topic [24].
- Master: Ein Namensdienst für ROS, er hilft den Nodes dabei, sich zu lokalisieren. Sobald sich diese Nodes gefunden haben, kommunizieren sie miteinander Peer-to-Peer [24]. Beim Ausführen des Programms muss der Master im Hintergrund immer laufen.

- Lauch: Das Ausführen des Lauch-Files, welches sich im Launch-Ordner des Experiments befindet, startet Nodes wie im Lauch-File definiert. Im Launch-File werden die zu startenden Files und Namespaces definiert.
- Services: Services sind eine weitere Möglichkeit, mit der Nodes miteinander kommunizieren können. Sie ermöglichen es Nodes, eine Anfrage zu senden und eine Antwort zu empfangen [24]. Die verantwortliche Datei des Experiments befindet sich im Service-Ordner und beinhaltet die Datentypen der „Requests“, also Anfragen und die der „Answers“, also Antworten.
- Bag: Im Bag-Ordner befindet sich das Bag-File, welches das Kameramaterial für das Experiment beinhaltet, mit welchem die Kamera Kalibration durchgeführt und auf welches die Trajektorie projiziert wird.
- Makelist: Die Datei CMakeLists.txt ist die Eingabe in das CMake-Buildsystem zum Erstellen von Softwarepaketen. Ein Package kann eine oder mehrere dieser Dateien enthalten. Sie beschreiben, wie der Code erstellt wird und wo er installiert wird [24]. Die Makelist befindet sich im Package des Experiments.
- Package-Datei: Die XML-Datei namens package.xml, muss im Stammordner jedes Pakets enthalten sein. Diese Datei definiert Eigenschaften über das Paket wie den Paketnamen, Versionsnummern, Autoren, Betreuer und Abhängigkeiten [24]. Die Package-Datei befindet sich ebenfalls im Package des Experiments.

Zudem befanden sich nach Kalibrieren der Kamera und Abspeichern der Parameter noch ein YAML-File, welches die intrinsischen und extrinsischen Parameter und die Distorsionparameter beinhaltet und das CSV-File, in welchem sich die generierte Trajektorie befindet, im Package.

### **3.7 Programm und Projektion**

Das Programm zur Projektion der Parameter befindet sich gemeinsam mit dem Hauptprogramm im Source-Ordner und wurde in der Programmiersprache C++ verfasst. Zu Beginn wurden die Parameter aus dem YAML-File mittels Nodehandler eingelesen. Dazu zählen eine Matrix der intrinsischen Kameraparameter, sowie zwei Matrizen der extrinsischen Parameter. Die beiden Matrizen der extrinsischen Parameter ergeben sich jeweils aus den Rotation- und Translationskoeffizienten von Kamera zur Inertial Measurement Unit (IMU) und von Optitrack zu IMU. Anschließend wurden die Wegpunkte des CSV-Files der Trajektorie im Codes eingelesen.

Folgend wurde die aktuelle Position der Drohne benötigt, diese konnte durch die aktuellen Rotations- und Translationsparameter bestimmt werden. Die aktuellen Parameter wurden in einer eigenen Methode durch das Empfangen von Messages laufend eingelesen.

In einer weiteren Methode wurde die totale Position bestimmt. Dies geschah mittels Errechnung der Rotations- und Translationsmatrizen der IMU, des Optitracks und der Drohne. Im Anschluss wurde mit einer von OpenCV bereitgestellten Funktion mittels totaler Rotation und Translation, sowie der intrinsischen Kameraparameter und der Verzerrungskoeffizienten die dreidimensionalen Wegpunkte der Trajektorie in zweidimensionale Bildpunkte transformiert. Der Vorgang wurde im Kapitel 2 genau beschrieben und im Code für jeden Punkt der Trajektorie durchgeführt.

### 3.8 Experiment in der Drohnenhalle

Nach Erfolgreichem Überlagern eines voraufgenommenen Videos mit der transformierten Trajektorie wurde ein Experiment in der Drohnenhalle geplant. Um das funktionsfähige Programm zu testen, konnte ein Termin in der Drohnenhalle im Lakeside Park in Klagenfurt mit Unterstützung von Herrn Univ.Prof. Dr. Weiss und Herrn Jantos vereinbart werden. Damit eine Trajektorie auf ein laufendes Kamerabild projiziert werden und diese Trajektorie mittels Schwenken der Kamera von allen Blickwinkeln aus betrachten werden konnte, musste zuerst eine Kamera bereitgestellt werden, welche durch das Subscriben von Topics im Code mit dem ROS-Package verbunden wurde. Im Anschluss konnte die in den Code eingelesene Trajektorie auf das laufende Kamerabild projiziert werden. Das Betrachten der eingelesenen Wegpunkte war problemlos von allen Perspektiven möglich. Um dies veranschaulichen und auswerten zu können, wurde ein BAG-File aufgenommen, welches die Trajektorie von allen Seiten zeigt. Im folgenden Kapitel wird veranschaulicht, wie die projizierte Trajektorie aus verschiedenen Blickwinkeln aussieht.

## 4 Ergebnisse

Dieses Kapitel zeigt die Ergebnisse der in Kapitel 3 beschriebene Methodik zur Kalibrierung einer Kamera, der Projektion eines einzelnen Punktes auf dem Kamerabild, sowie auch die Projektion einer künstlich generierten Trajektorie, sowohl mit hundert als auch mit tausend Wegpunkten.

## 4.1 Kalibrierung der Kamera

```
# lens calibration: fu, fv, pu, pv, k1, k2, r1, r2
cam_intrinsics: [1129.9331937554641, 1130.9895755901139, 1044.2720522161956, 741.7471990254357,
-0.2840863427773136, 0.08521145025642786, -0.00040481857154512014, 0.000981699472593185]
# transformation from imu to cam (column major): r11, r21, r31, 0, r12, r22, r32, 0, r13, r23, r33, 0, t1,
t2, t3, 1
cam_imu_extrinsics: [-0.99953564, -0.02175933, 0.02133164, 0.0, -0.0218677, 0.99974906, -0.00486018,
0.0, -0.02122054, -0.00532439, -0.99976064, 0.0, -0.23926146, -0.05014366, -0.01954729, 1.0]
# transformation from imu to optitrack (column major): r11, r21, r31, 0, r12, r22, r32, 0, r13, r23, r33, 0,
t1, t2, t3, 1
opti_imu_extrinsics: [0.9979, 0.0531, -0.0377, 0.0, -0.0549, 0.9972, -0.0503, 0.0, 0.0349, 0.0523, 0.9980,
0.0, -0.1000, 0.0, 0.06, 1.0000]
bool_example: True
int_example: 111
double_example: 11.1
enum_example: EX2
```

Abbildung 9: Ergebnisse Kalibration im YAML-File

Die Abbildung 9 zeigt ein händisch gefülltes YAML-File, welches die nach der Kalibrierung der Kamera mittels Kalibr Docker entstandenen Parameter beinhaltet. Die erste Matrix „camera\_intrinsics“ beinhaltet die intrinsischen Kameraparameter. Die Brennweiten stehen an den ersten beiden Stellen ( $f_u, f_v$ ), gefolgt von den optischen Zentren ( $p_u, p_v$ ). Im Anschluss involviert die Matrix noch die Verzerrungskoeffizienten der radialen und der tangentialen Verzerrung ( $k_1, k_2, r_1, r_2$ ). Die anderen beiden Matrizen stellen jeweils die Rotationskoeffizienten ( $r_{11}, r_{21}, r_{31}, r_{12}, r_{22}, r_{32}, r_{13}, r_{23}, r_{33}$ ) und die Translationskoeffizienten ( $t_1, t_2, t_3$ ) von Kamera zu IMU und von Optitrack zu IMU dar.

## 4.2 Projektion eines einzelnen Punktes

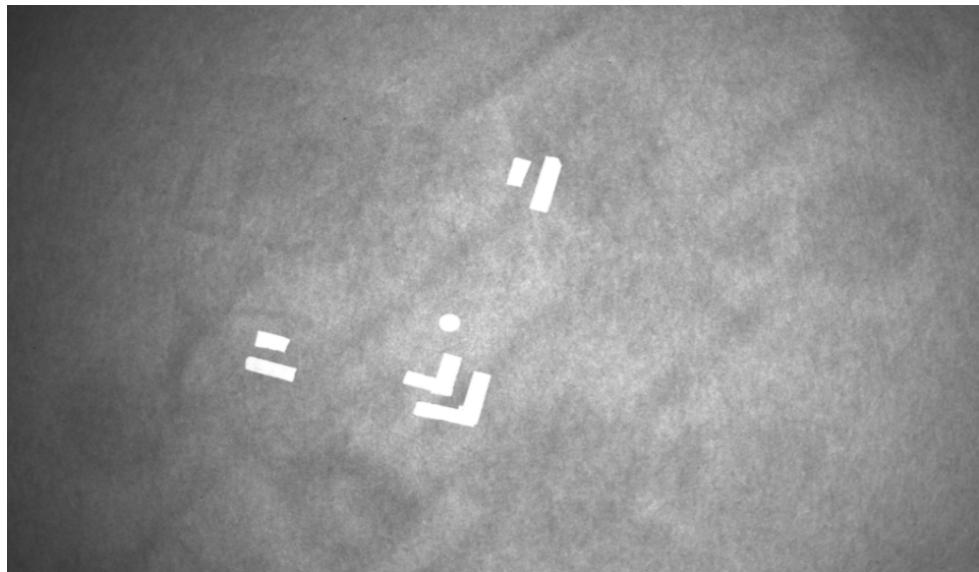


Abbildung 10: Projektion eines einzelnen Punktes in der Nahansicht

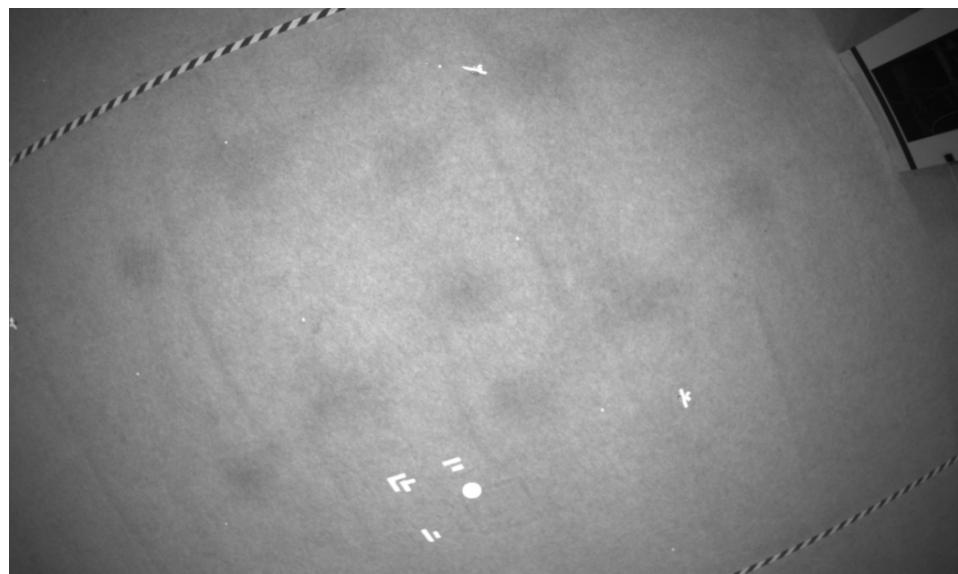


Abbildung 11: Projektion eines einzelnen Punktes Fernansicht

In Abbildung 10 und in Abbildung 11 ist ein weißer Punkt zu erkennen. Dies ist ein Punkt, dessen reale Welt Koordinaten zu Testzwecken im Code festgelegt sind. Wie in Kapitel 2 beschrieben, werden seine Bildkoordinaten berechnet und die Projektion wird durchgeführt. Bei laufendem Video ist zu erkennen, dass sich der Punkt kaum von der Stelle bewegt.

### 4.3 Generieren einer Trajektorie

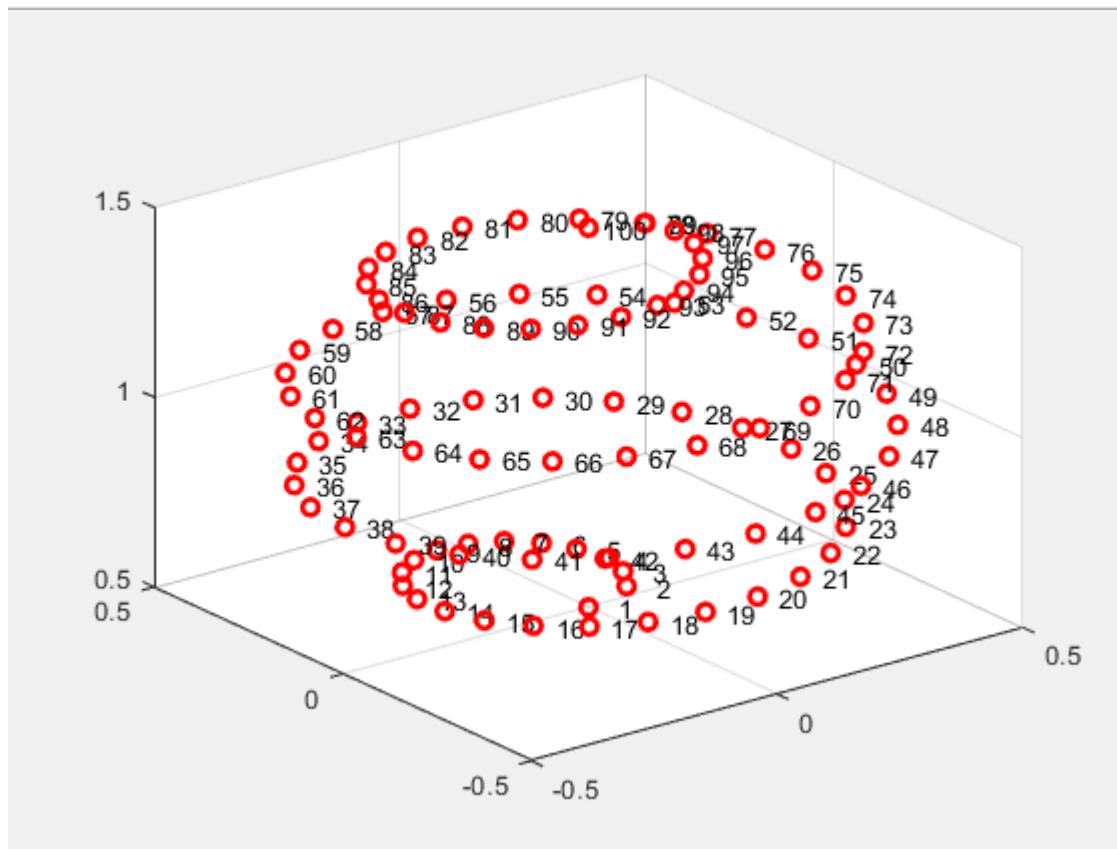


Abbildung 12: in MATLAB generierte Trajektorie mit 100 Wegpunkten

In Abbildung 12 ist eine in MATLAB generierte Trajektorie zu erkennen. Zu sehen ist ein Startpunkt mittig am Boden des Raumes. Spiralförmig fliegt die Drohne von unten nach oben, die Aufzeichnung der Wegpunkte endet oben mittig im Raum.

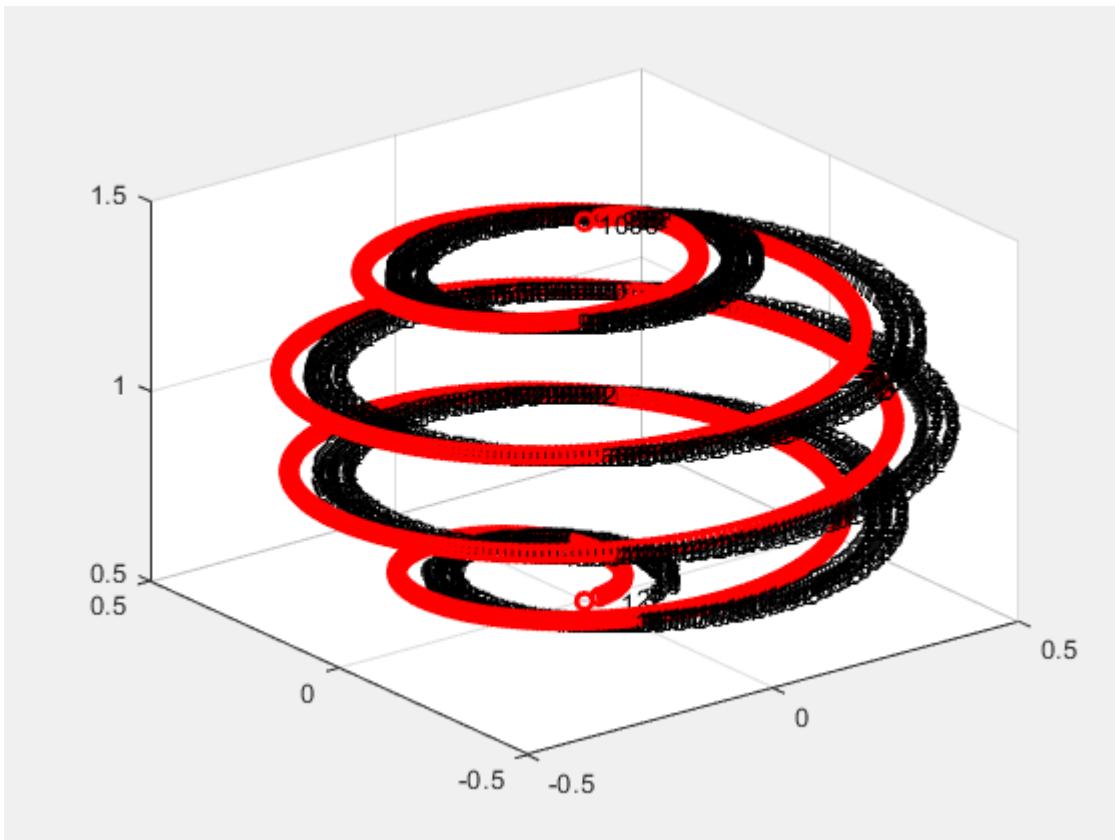


Abbildung 13: in MATLAB generierte Trajektorie mit 1000 Wegpunkten

In Abbildung 13 ist dieselbe Trajektorie, generiert mittels MATLAB, mit eintausend Wegpunkten. Aufgrund der höheren Anzahl von Wegpunkten ist die Abgeflogene Strecke besser zu erkennen.

## 4.4 Projektion einer Trajektorie

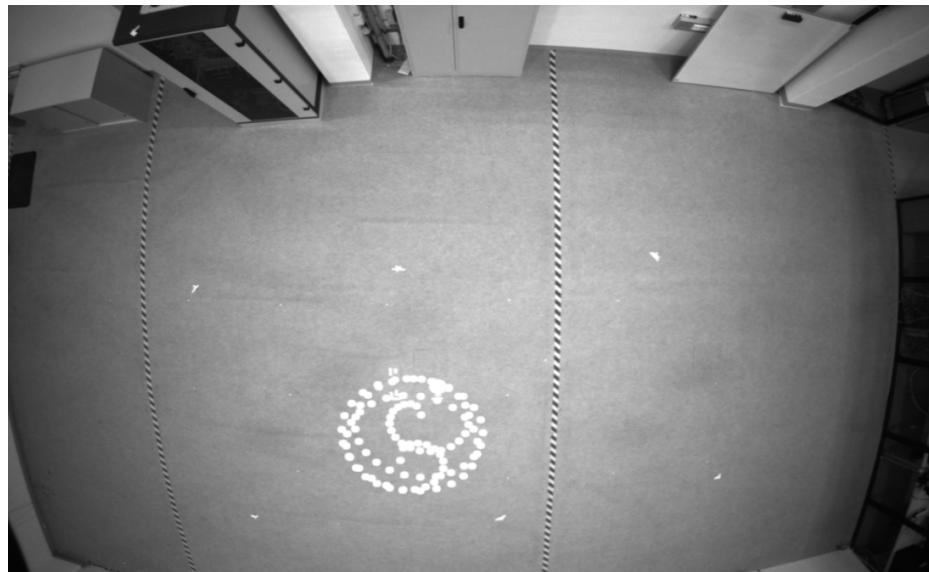


Abbildung 14: Projektion einer Trajektorie mit 100 Wegpunkten



Abbildung 15: Projektion einer Trajektorie mit 1000 Wegpunkten

Die beiden Projektionen in Abbildung 14 und in Abbildung 15 stellen die in Abbildung 12 und Abbildung 13 generierten Trajektorien im Kamerabild dar. Das Ergebnis ist beim Versuch mit einhundert Wegpunkten in Abbildung 14 bereits gut sichtbar, jedoch ist das Erkennen der Trajektorie mit tausend Wegpunkten in Abbildung 15 leichter.

## 4.5 Live-Projektion in der Drohnenhalle

Nach Fertigstellung des funktionsfähigen Programms, konnte dieses in der Drohnenhalle im Lakeside Park in Klagenfurt mit Unterstützung von Herrn Univ.Prof. Dr. Weiss und Herrn Jantos live getestet werden. In folgenden Abbildungen wird veranschaulicht, wie die projizierte Trajektorie aus verschiedenen Blickwinkeln aussieht:

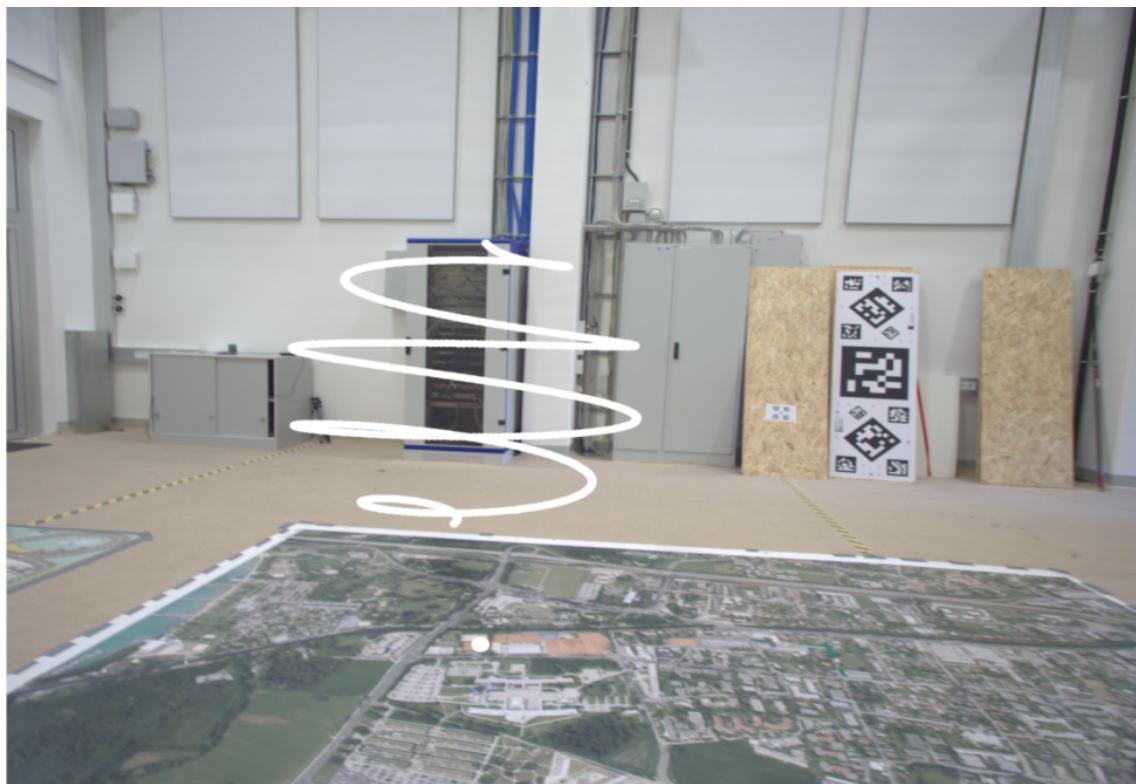


Abbildung 16: in der Drohnenhalle live aufgenommene Seitenansicht der Trajektorie

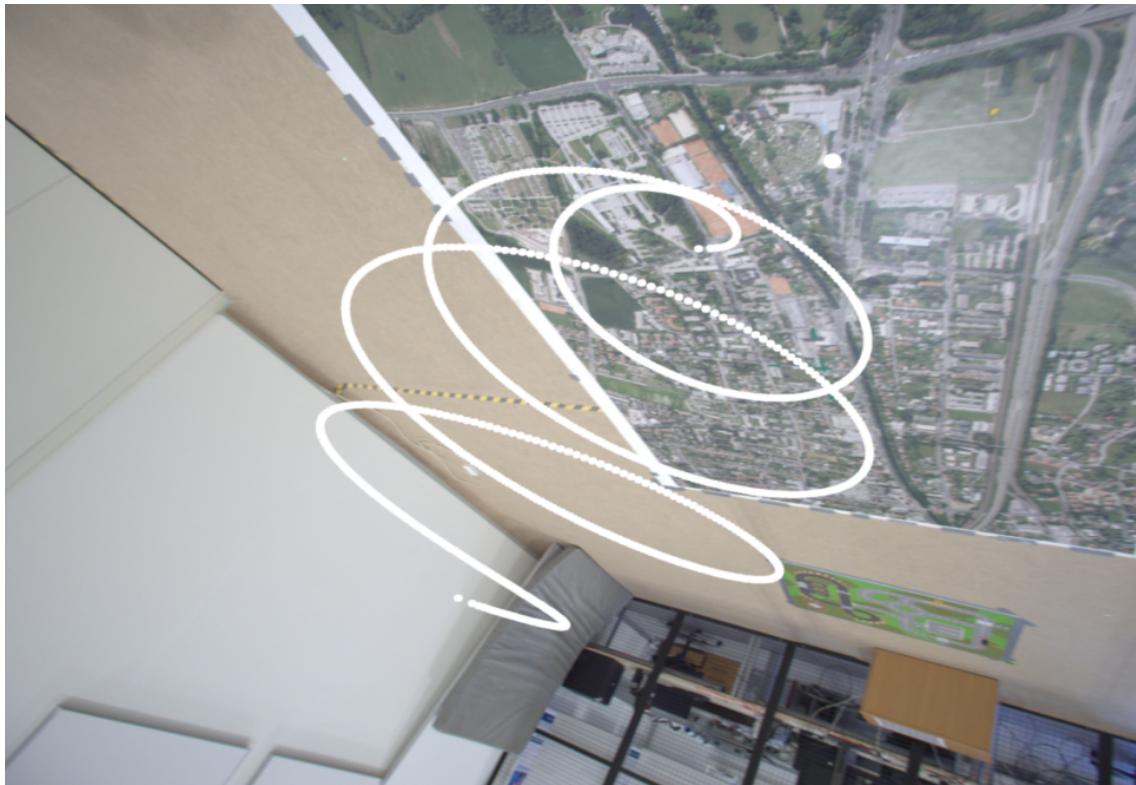


Abbildung 17: in der Drohnenhalle live aufgenommene Nahansicht der Trajektorie

Bei vorherigen Versuchen wurden die Wegpunkte auf bereits bestehendes Kameramaterial abgebildet und konnten so hauptsächlich nur aus der Vogelperspektive betrachtet werden. Mit eigenen Aufnahmen konnte die Trajektorie aus allen Perspektiven, von nah und von fern, veranschaulicht werden. In Abbildung 16 ist eine Seitenansicht gegeben. Hier ist die geflogene Spiralform von mittig unten nach mittig oben klar ersichtlich. In Abbildung 17 wird eine Nahansicht veranschaulicht, welche die einzelnen Wegpunkte besser erkennen lässt.

## 5 Diskussion

Um eine Trajektorie, die den Flug einer Drohne in dreidimensionalen Koordinaten darstellt, auf ein Kamerabild, welches zweidimensionale Koordinaten aufweist, abbilden zu können, wurden zuallererst Literaturrecherche, gefolgt von Implementierung des Codes und eine Vielzahl von Versuchen und Experimenten durchgeführt. Dafür

wurde mittels Kalibr Docker Tool die verwendete Kamera kalibriert und anschließend deren entstandene Parameter zur Konvertierung der dreidimensionalen Weltpunkte zu zweidimensionalen Bildpunkten eingesetzt.

Diese Bachelorarbeit hat erste Erkenntnisse zum Themenbereich 3D zu 2D Drohnenflug Visualisierung geliefert. Theoretisch soll sich die projizierte Trajektorie bei Bewegung der Kamera immer am selben Ort befinden. Versuchsergebnisse an vorhandenen Aufnahmen weichen nur leicht von der Theorie ab. Im Versuch hat sich ergeben, dass die Trajektorie am Kamerabild bei Bewegung des Bildes annähernd am selben Ort bestehen bleibt, was zu einem zufriedenstellenden Forschungsergebnis führt. Die leichten Abweichungen der praktischen Durchführung von der Theorie ergeben sich durch das Rauschen, auch Bildrauschen genannt. Bildrauschen entsteht durch den Bildsensor und liefert unerwünschte Informationen in digitalen Bildern. Grund dafür kann oft der Lichteinfall sein.

Auch bei den Live-Versuchen in der Drohnenhalle bleibt die Trajektorie wie gewünscht annähernd am selben Ort im Raum. Kleinere Abweichungen, wie leichte Bewegung der Trajektorie sind auch hier auf das Bildrauschen zurückzuführen.

Durch die Methode, eine generierte Trajektorie fix einzulesen, wurde der gewünschte Effekt bereits erzielt und ein Grundstein für weitere Einsatzgebiete gelegt. Anhand weiterer Versuchsdurchführungen mit live geflogenen Wegpunkten könnte die gerade abgeflogene Trajektorie auch tatsächlich sofort projiziert und untersucht werden.

## 6 Fazit

Das Ziel der vorliegenden Arbeit ist es zu verstehen und zu implementieren, wie sich durch eine an der Drohne angebrachte Kamera aufgenommene 3D Punkte in der realen Welt zu 2D Punkten in einem Bild direkt visualisieren lassen. Mit anderen Worten soll es möglich sein, eine Trajektorie in das Programm einzulesen, welche aus dreidimensionalen Welt-Koordinaten besteht, diese in zweidimensionale Bild-Koordinaten zu konvertieren und schließlich auf das Kamerabild zu projizieren. Das fertige Tool ermöglicht genau diesen Vorgang. Vorausgesetzt eine Trajektorie ist vorhanden, beziehungsweise, wird künstlich generiert, kann diese auf das laufende Kamerabild projiziert werden.

### 6.1 Zusammenfassung

Zusammenfassend kann gesagt werden, dass das Ziel der Arbeit erreicht und auch der ungefähre Zeitplan wie gewünscht eingehalten wurde. Die Kalibrierung für die

verwendete Kamera, sowie ein funktionsfähiges Programm für das Einlesen einer Trajektorie und die Projektion der Wegpunkte wurden erstellt und können verwendet, oder auch in weitere Projekte implementiert werden.

## 6.2 Ausblick

Voraussichtlich wird das entworfene Tool in weiteren Projekten eingesetzt. Beispielsweise soll es zukünftig für Debug-Prozesse und Visualisierungen der entstehenden Forschungsergebnisse verwendet werden.

## 7 Nachwort

Das Schreiben dieser Arbeit habe ich als sehr lehrreich empfunden, vor allem das Programmieren des praktischen Teiles der vorliegenden Arbeit hat mir einen profitablen Einblick in die Praxis gewährt.

Aufgrund eines Fehlers beim Implementieren der Kalibration/Projektion, der zur Halbzeit des Projektes aufgetreten ist, durfte ich auch einige Einblicke in die Fehler suche erleben. Die Arbeit und das Projekt haben mein Wissen nicht nur in deren Themenbereiche erweitert, sondern haben mir darüber hinaus auch das Verfassen wissenschaftlicher Arbeiten, das richtige Zitieren, sowie Literaturrecherche und auch das Arbeiten mit neuen Programmen, wie ROS oder dem Image Kalibr Docker nähergebracht.

Ich möchte mich an dieser Stelle bei meinem Betreuer Herr Univ.Prof. Dr. Weiss bedanken, der mich bei Entscheidungen hinsichtlich meiner Vorgehensweise unterstützt und mir wichtige Denkanstöße für die Untersuchung gegeben hat.

Nina Suette

Klagenfurt, September 2022

## Literatur

- [1] G. Forlani, R. Roncella and C. Nardinocchi, Where is photogrammetry heading to? State of the art and trends. 2015
- [2] F. Qiang, C. Xiang-Yang, H. Wei, A Survey on 3D Visual Tracking of Multicopters, 2019
- [3] R. T. Azuma, A Survey of Augmented Reality, Presence: teleoperators & virtual environments 6.4, 1997.
- [4] C. Hung-Lin, K. Shih-Chung and W. Xiangyu, Research trends and opportunities of augmented reality applications in architecture, engineering, and construction, 2013.
- [5] E. Schwalbe, Geometric modelling and calibration of Fisheye Lens Camera systems, 2005
- [6] C. Geyer and K. Daniilidi, Catadioptric Camera Calibration, 1999
- [7] D. Stricker, Computer-Vision-basierte Tracking- und Kalibrierungsverfahren fur Augmented Reality, 2002
- [8] A. Menache, Understanding Motion Capture for Computer Animation. Motion Capture and Performance Animation. Elsevier, 2011.
- [9] A. Masiero, F. Fissorea, R. Antonellob, A. Cenedeseb and A. Vettore, A comparison of UWB and motion capture UAV indoor positioning, 2019
- [10] Z. Peijun, C. Xiaoxuan Lu, W. Bing, N. Trigoni and A. Markham, 3D Motion Capture of an Unmodified Drone with Single-chip Millimeter Wave Radar, 2021
- [11] Der Standard. (2021, August 13). Universität Klagenfurt bietet Drohnenforschung eigene Flughalle. [Online]. Available: <https://www.derstandard.at/story/2000112431569/universitaet-klagenfurt-bietet-drohnenforschung-eigene-flughalle>.
- [12] G. Bradski and A. Kaehler, Learning OpenCV. Computer Vision with the OpenCV Library. Sebastopol: O'Reilly Media, Inc., 2008.
- [13] Open Source Computer Vision. (2021, August 20). Camera Calibration. [Online]. Available: [https://docs.opencv.org/master/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html).

- [14] D. Wischounig-Strucl. (2013). Resource aware and incremental mosaics of wide areas from small-scale UAVs (Dissertation, Technischen Wissenschaften). Alpen-Adria Universitat Klagenfurt. [Online]. Available: <https://netlibrary.aau.at/obvuklhs/download/pdf/2411488?originalFilename=true>
- [15] K. Sadekar and S. Mallick. (2020, Februar 25). Camera Calibration using OpenCV. [Online]. Available: <https://learnopencv.com/camera-calibration-using-opencv/>
- [16] Open Source Computer Vision. (2020, Oktober 11). Camera calibration With OpenCV. [Online]. Available: [https://docs.opencv.org/4.5.0/d4/d94/tutorial\\_camera\\_calibration.html](https://docs.opencv.org/4.5.0/d4/d94/tutorial_camera_calibration.html)
- [17] Open Source Computer Vision (2020, Oktober 11). Camera Calibration and 3D Reconstruction. [Online]. Available: [https://docs.opencv.org/4.5.0/d9/d0c/group\\_\\_calib3d.html](https://docs.opencv.org/4.5.0/d9/d0c/group__calib3d.html)
- [18] Z. Zhang, A Flexible New Technique for Camera Calibration. Transactions on pattern analysis and machine intelligence. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.
- [19] H. Louhichi, T. Fournel, J. M. Lavest, and H. Ben Aissia, Self-calibration of scheimpflug cameras: an easy protocol. Meas. Sci. Technol., 2007.
- [20] WebSetNet (2020, August 5). So installieren Sie VirtualBox unter Linux Ubuntu 20.04. [Online]. Available: <https://websetnet.net/de/how-to-install-virtualbox-on-linux-ubuntu-20-04/>
- [21] Computer Bild (2021, August). Ubuntu LTS (Long-Term-Support). [Online]. Available: <https://www.computerbild.de/download/Ubuntu-LTS-Long-Term-Support-29171319.html>
- [22] ROS.org (2013, Juli 12). Was ist ROS? [Online]. Available: <http://wiki.ros.org/de/ROS/Introduction>
- [23] OpenCV (2021). About. [Online]. Available: <https://opencv.org/about/>
- [24] ROS.org (2021, September 8). Understanding ROS Nodes. [Online]. Available: <http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>