



Mixed Reality ohne HoloLens
Entwicklung eines Prototypen für SLAM

Bachelor Thesis

von

Andreas Lakus

Matrikelnummer: 11012049

12.09.2021

SRH Hochschule Heidelberg
Fakultät für Information, Medien & Design
Studiengang „Virtuelle Realitäten“
Schwerpunkt „Game Development“

Gutachter

Prof. Dr.-Ing. Daniel Görlich

Dr. Andreas Jäger

Ich widme diese Arbeit meiner Frau Julia, als Gegenleistung für die verlorene gemeinsame Zeit. Es hat sich gelohnt!

Meinen Großvätern Günther und Karl-Heinz, die einen wesentlichen Meilenstein meines Lebens nicht mehr miterleben durften.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die Kapitel der Arbeit, für die ich als Verfasser genannt werde, selbständig verfasst habe, dass ich keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt habe und dass ich diese Arbeit bei keinem anderen Prüfungsverfahren vorgelegt habe.

Heidelberg, den 14.09.2021 Unterschrift _____

Abstract

Mixed Reality (MR) mixes physical with virtual reality. The Microsoft HoloLens interprets the physical environment through optical sensors and thus realizes spatial computing. Spatial Computing comprises the utilization of spatial geometry as input data for algorithms. This enables marker-less, inside-out tracking. To do this, however, the HoloLens must be localized at the same time as the spatial geometry is interpreted. The problem is known in robotics as Simultaneous Localization and Mapping (SLAM). If the robot is replaced by an operator with a head-mounted display, solutions for SLAM can also be transferred to MR. The aim of this work is to check whether the SLAM problem can also be solved without the HoloLens using just one camera. For this purpose, ORB-SLAM2 is identified and implemented as a valid visual SLAM solution strategy. To enable the subsequent development of an MR application, ORB-SLAM2 is integrated into the Robot Operating System (ROS). This enables the connection to the Unity Engine and thus the utilization of the SLAM process for the development of MR-capable applications. The successful integration of the SLAM process into the Unity Engine could be demonstrated. Subsequent tests revealed a difficult initialization phase of the algorithm and the problematic handling of pure rotations, which, however, could be minimized by integrating the algorithm in ROS. This work shows that MR can also be solved without the HoloLens and Windows Mixed Reality.

Zusammenfassung

Mixed Reality (MR) vermischt die physische mit der virtuellen Realität. Die Microsoft HoloLens interpretiert die physische Umgebung durch optische Sensoren und realisiert so Spatial Computing. Spatial Computing umfasst die Nutzbarmachung der Raumgeometrie als Eingabedaten für Algorithmen. Dies ermöglicht marker-less, inside-out Tracking. Dazu muss die Lokalisierung der HoloLens allerdings zeitgleich zur Interpretation der Raumgeometrie erfolgen. Das Problem ist in der Robotik als Simultaneous Localization and Mapping (SLAM) bekannt. Ersetzt man den Roboter durch einen Operator mit Head-Mounted Display, können Lösungen für SLAM auch auf MR übertragen werden. Diese Arbeit verfolgt das Ziel zu überprüfen, ob sich das SLAM-Problem auch ohne die HoloLens mittels nur einer Kameras lösen lässt. Dazu wird ORB-SLAM2 als valide visual SLAM-Lösungsstrategie identifiziert und implementiert. Um die anschließende Entwicklung einer MR-Applikation zu ermöglichen, wird ORB-SLAM2 in das Robot Operating System (ROS) integriert. Dies ermöglicht die Verbindung zur Unity Engine und damit die Nutzbarmachung des SLAM-Verfahrens für die Entwicklung MR-fähiger Applikationen. Die erfolgreiche Integration des SLAM-Verfahrens in die Unity Engine konnte demonstriert werden. Anschließende Tests ergaben eine schwierige Initialisierungsphase des Algorithmus und den problematischen Umgang mit puren Rotationen, was jedoch durch die Integration des Algorithmus in ROS minimiert werden konnte. Diese Arbeit zeigt, dass sich MR auch ohne die HoloLens und Windows Mixed Reality lösen lässt.

Danksagung

An dieser Stelle möchte ich mich zunächst bei meinem Betreuer und ersten Gutachter Herrn Prof. Dr.-Ing. Daniel Görlich für die interessante Aufgabenstellung und die herausragende Betreuung bedanken. Seine Ideen und Vorschläge waren bei der Ausarbeitung dieser Arbeit immer eine wesentliche Hilfe.

Des Weiteren möchte ich mich bei Herrn Dr. Andreas Jäger, für die nicht selbstverständliche Übernahme der Aufgabe des Zweitgutachters bedanken.

Ein besonderer Dank gilt meiner Ehefrau, die mir in dieser schwierigen Phase des Studiums den Rücken freihält und mir dadurch den notwendigen Freiraum geschaffen hat, diese Arbeit anzufertigen.

Inhalt

1	Einleitung und Motivation	12
1.1	Zielsetzung	13
1.2	Methodik	13
2	Grundlagen.....	15
2.1	Mixed Reality.....	15
2.1.1	Reality-Virtuality-Continuum	16
2.1.2	Virtual Reality.....	17
2.1.3	Menschliche Wahrnehmung in virtuellen Realitäten.....	19
2.1.4	Augmented Virtuality.....	20
2.1.5	Augmented Reality.....	21
2.2	Spatial Computing.....	22
2.3	Head-Mounted Displays.....	24
2.4	Simultaneous Localization and Mapping	26
2.5	Kamerasysteme und -modelle	29
2.5.1	Technischer Aufbau RGB-Kamera	31
2.5.2	Das Lochkameramodell	32
2.5.3	Kamerakalibrierung.....	34
2.5.4	Perspektivische Projektion	36
2.6	Visuelles Tracking	39
2.6.1	Tracking mit synthetischen Markern	39
2.6.2	Marker-less Tracking	40
2.7	Verwendete Software	40
2.7.1	Open Computer Vision Library	40
2.7.2	Robot Operating System	41
2.7.3	Unity Engine	41

2.8	Zusammenfassung.....	42
3	Stand der Technik	44
3.1	Interpretation der Mixed Reality	44
3.2	Microsoft HoloLens	47
3.3	Visual SLAM	50
3.3.1	Abgrenzung vSLAM von Visual inertial SLAM	50
3.3.2	Taxonomie moderner vSLAM-Methoden.....	51
3.3.3	Visual-SLAM-Framework	55
3.3.4	Visuelle Odometrie	56
3.3.5	Kartenkonstruktion.....	58
3.4	Zusammenfassung.....	59
4	Lösungskonzept	60
4.1	Umgebungsdefinition und Einsatzgebiet	60
4.2	Anforderungserhebung und -analyse	61
4.2.1	Funktionale Anforderungen	61
4.2.2	Nicht-funktionale Anforderungen	62
4.3	Identifikation einer SLAM-Lösungsstrategie.....	62
4.4	Systemaufbau	63
5	ORB-SLAM	65
6	Realisierung	69
6.1	Sensoren.....	69
6.2	Hardware	70
6.3	Software	70
6.4	Kamerakalibrierung.....	70
6.5	Implementierung ORB-SLAM2-Native.....	71
6.6	Anpassung des AR-Demo für ORBSLAM2	72

6.7	ORBSLAM2-ROS-Integration	73
6.8	Anbindung Kamera	74
6.9	Unity Engine	75
6.9.1	Unity-Robotics-Hub	75
6.9.2	TCP-Bridge	75
6.9.3	Unity Projekt	76
6.9.4	Subscriber für die Kamerapose	76
6.9.5	Script zur Nachverfolgung der Trajektorie	77
6.9.6	Subscriber zur Integration der Karteninformationen	78
6.10	Zusammenfassung	78
7	Test	79
7.1	Testkriterien	79
7.2	Testaufbau	80
7.2.1	Test TUM-Dataset	80
7.2.2	Entfernungstest	80
7.2.3	Live-Mapping	81
7.2.4	Rekonstruktionstest	81
7.2.5	Rotationstest	82
7.2.6	Test: RPY-Live	82
7.2.7	Test: XYZ-Live	82
7.2.8	ROS-Unity-Live-Test	82
7.3	Ergebnispräsentation und Auswertung	82
7.3.1	Test TUM-Dataset	83
7.3.2	Ergebnisse TUM-RPY-Dataset	84
7.3.3	Ergebnisse TUM-XYZ-Dataset	84
7.3.4	Vergleich der Testergebnisse	85

7.3.5	Entfernungstest.....	86
7.3.6	ORB-SLAM2 – Live Mapping	87
7.3.7	ORB-SLAM2 - Rekonstruktion.....	88
7.3.8	ORB-SLAM2 - Rotationstest.....	89
7.3.9	ORB-SLAM2 - RPY-Live	90
7.3.10	ORB-SLAM2 - XYZ-Live	91
7.3.11	Zusammenfassung der Ergebnisse von ORB-SLAM-Nativ	91
7.3.12	Zusammenfassung der Ergebnisse von ORB-SLAM-ROS.....	92
7.3.13	AR-DEMO	92
7.3.14	ROS-Unity	93
7.4	Zusammenfassung.....	94
8	Evaluation	95
9	Fazit und Ausblick	98
	Anhang	100
	Anhang 1 - Taxonomie moderner vSLAM-Verfahren.....	100
	Anhang 2 - Komponentendiagramm.....	100
	Anhang 3 - Data-Flow-Diagramm	101
	Anhang 4 - ORB-Feature Extraktor	102
	Anhang 5 – Verwendete Software-Versionen.....	103
	Anhang 6 - Kamerakalibrierung.....	104
	Anhang 7 – Ergebnisse Kamerakalibrierung	107
	Anhang 8 - ORB-SLAM2 – MonoAR.cc	108
	Anhang 9 - Unity-ROS – Subscriber Lokationsdaten.....	112
	Anhang 10 - Unity-ROS – Bread Crumbs	113
	Anhang 11 - Unity-ROS – Subscriber Point Cloud	114
	Anhang 12 - Testaufbau Entfernungstest.....	116

Anhang 13 - Testaufbau Entfernungstest - Kameraperspektive.....	117
Anhang 14 - ORB-SLAM-Native – Entfernungstest	118
Anhang 15 – ORB-SLAM-Native - Rekonstruktion.....	119
Anhang 16 - Test – ORBSLAM2 Nativ – TUM rpy.....	120
Test Nr.1.....	120
Test Nr. 2.....	123
Test Nr. 3.....	126
Test Nr. 4.....	129
Test Nr. 5.....	132
Anhang 17 - Test – ORBSLAM2 Nativ – TUM xyz	135
Test Nr. 1.....	135
Test Nr. 2.....	139
Test Nr. 3.....	142
Test Nr. 4.....	145
Test Nr. 5.....	148
Anhang 18 - Inhalt beiliegender DVDs	151
Literatur.....	152
Abbildungsverzeichnis.....	169
Tabellenverzeichnis	171
Listingverzeichnis	172
Formelverzeichnis	173
Abkürzungsverzeichnis.....	174

1 Einleitung und Motivation

Mixed Reality (MR) vermischt die physische Realität mit einer computergenerierten Alternativrealität, um virtuelle Objekte in die Perspektive des Nutzers zu integrieren (Coutrix und Nigay, 2006). Dadurch kann die Interaktion zwischen dem Nutzer und den virtuellen Objekten mittels bspw. eines Head-Mounted Displays (HMD) ermöglicht werden (Lalanne und Kohlas, 2009, Taheri und Xia, 2021). Die Interoperabilität zwischen der physischen und der virtuellen Realität, kann durch eine genaue Selbstlokalisierung, relativ zur Raumgeometrie gewährleistet werden. Aufgrund der Kenntnis über die Position, sowohl realer als auch virtueller Objekte, können Interaktionen zwischen beiden Welten realisiert werden. Jedoch müssen die eigene Pose, sowie die Geometrie des Raumes zum gleichen Zeitpunkt ermittelbar sein (Dörner et al., 2013).

Das Problem der zeitgleichen Lokalisation und Kartierung in unbekanntem Raum, ist in der Robotik als Simultaneous Localization and Mapping (SLAM) bekannt und ein aktives Forschungsgebiet (Borthwick und Durrant-Whyte, 1994, Smith und Cheeseman, 1986, Taheri und Xia, 2021, Frese, 2010). Hierbei steht ein autonomes System vor dem Problem, sich selbst in einer sich zur Programmlaufzeit entwickelnden Karte zu lokalisieren. Diese Problemstellung besteht auch in Mixed-Reality-Anwendungsfällen, in denen markerless-tracking praktiziert wird (Vlaminck, Luong und Philips, 2017, Liu et al., 2020). Hierbei werden durch Kameras natürlich vorkommende Bildmerkmale identifiziert und deren Pose für die eigene Lokalisation genutzt (Dörner et al., 2013, Rokhsaritalemi, Sadeghi-Niaraki und Choi, 2020). Die Lösungsstrategien hinsichtlich SLAM können auf MR transferiert werden, indem das autonome System durch ein HMD inklusive Operator ersetzt wird (Chen et al., 2018a).

Microsoft (MS) gilt als Vorreiter für MR und bietet mit der HoloLens das erste autarke HMD an, welches die Umgebung durch Kameras interpretierbar macht und somit SLAM lösen kann (Evans et al., 2017). Durch Windows-Mixed-Reality (Windows-MR) stellt MS zudem ein MR-Ökosystem bereit, welches Entwickertools und abgestimmte Softwarelösungen für unterschiedliche Anwendungsfälle bietet (Wojo, 2021). MS löst – mit der HoloLens und Windows-MR – das SLAM-Problem, jedoch ist der algorithmische Systemablauf in seiner konkreten Umsetzung durch Patente geschützt und nicht öffentlich verfügbar (Khoshelham, Tran und Acharya, 2019).

Aufgrund der Annahme, MS löse das SLAM-Problem mittels aktuellen Forschungsansätzen, setzt sich diese Arbeit zum Ziel, das SLAM-Problem nach dem Vorbild der HoloLens mittels Kameras zu lösen und hierfür einen Prototypen zu entwickeln.

1.1 Zielsetzung

Diese Arbeit verfolgt das Ziel einen Prototypen zur Lösung des SLAM-Problems für MR-Anwendungen zu entwickeln und diesen zu testen.

1.2 Methodik

Die vorliegende Arbeit umfasst insgesamt 10 Kapitel. In Kapitel zwei werden die Grundlagen des RV-Continuum nach Milgram hergeleitet (Milgram und Kishino, 1994). Weiter wird Spatial Computing als Nutzbarmachung der physischen Umgebung als Eingabedaten für Computerprogramme dargelegt. Spatial Computing bedingt die Informationsextraktion durch entsprechende Sensorik (Greenwold, 2003). Dazu werden Head-mounted Displays vorgestellt, welche mit optischer Sensorik die Umgebung wahrnehmen können (Sutherland, 1968). Simultaneous Localization and Mapping (SLAM) wird als Problemstellung aus der Robotik vorgestellt (Smith und Cheeseman, 1986) und der Bezug zu Mixed Reality (MR) und Spatial Computing erläutert (Cöltekin et al., 2020). Weiter werden Kameras als Sensoren für die sensorische Interpretation der Raumgeometrie vorgestellt (Zeller, 2021). Die Grundlagen des Lochkameramodells und der perspektivischen Projektion erläutern die Ermittlung der Pose (Hartley und Zisserman, 2015). Danach werden Verfahren zum visuellen Tracking vorgestellt (Dörner et al., 2013). Abschließend zu Kapitel 2, wird die in dieser Arbeit verwendete Software zur Herstellung des Prototypen vorgestellt.

Das dritte Kapitel dient der Ermittlung des aktuellen Stand der Technik. Das RV-Continuum erlaubt die Interpretation unterschiedlicher Variationen der MR (Milgram und Kishino, 1994). Die in dieser Arbeit fokussierte Interpretation – Windows Mixed Reality – wird vorgestellt (Bray, 2021). Anschließend wird der State of the Art, bezogen auf die Ein- und Ausgabegeräte für Mixed Reality, mit der HoloLens repräsentiert (Evans et al., 2017). Anschließend werden visual SLAM-Verfahren erläutert (Chen et al., 2018b).

Einleitung und Motivation

Kapitel 4 dient der Konzeption der verfolgten Lösungsstrategie für den Prototypen. Die Umgebung und die darin enthaltenen Dynamik, spielt eine wesentliche Rolle im Zuge der Identifikation einer validen Lösungsstrategie für SLAM (Hentschel und Wagner, 2011, Stachniss, Leonard und Thrun, 2016). Daher werden Umgebungen klassifiziert und vorgestellt. Weiter wird die Anforderungserhebung bzgl. der Realisierung des Prototypen durchgeführt. Das Ziel der Lösungskonzeption, wird durch die Identifikation einer validen Lösungsstrategie für das SLAM-Problem, respektive des Systementwurfs erreicht.

In Kapitel 5 wird ORB-SLAM2 als Lösungsstrategie zur Umsetzung in einem Prototypen vorgestellt (Mur-Artal und Tardos, 2017). Dazu werden in Kapitel 6 die Teilschritte zur Realisierung des Prototypen präsentiert.

Kapitel 7 enthält die Testkriterien und die Testdurchführung. Hierbei wird zunächst der native ORB-SLAM2-Algorithmus anhand den für SLAM üblichen Datasets (TUM) getestet (Sturm et al., 2012). Hierbei werden Datasets ausgewählt, welche die natürlichen Bewegungsabläufe einer Kopfbewegung nachbilden und einem MR-Szenario ähneln. Anschließend werden – begründet durch den Mangel eines AR-Benchmarks (Jinyu et al., 2019) – verschiedene Echtzeit-Testszenarien durchgeführt, um den Algorithmus und das System zu überprüfen. Abschließend werden die Daten präsentiert und die Ergebnisse ausgewertet.

Das achte Kapitel dient der Evaluation der gewonnen Erkenntnisse und Forschungsleistung. Die Arbeit schließt mit dem Fazit und Ausblick auf weitere Forschung in Kapitel 9 ab.

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für die zugrundeliegende Arbeit näher betrachtet. Mixed Reality wird als Spektrum vorgestellt und die Teilbereiche des MR-Spektrums näher beleuchtet. Um die menschliche Wahrnehmung im Kontext virtueller Realitäten darzulegen, werden Virtual Reality, Augmented Virtuality und Augmented Reality vorgestellt. Danach wird Spatial Computing als wissenschaftliche Disziplin zum Einbezug des umgebenden physischen Raumes als Eingabedaten für Computerprogramme präsentiert (Greenwold, 2003) und der Bezug zu MR erläutert. Zudem werden für MR-Erfahrungen eingesetzte Mensch-Maschine-Schnittstellen betrachtet und dazu Head-Mounted Displays präsentiert.

Weiter wird Simultaneous Localization and Mapping (SLAM) als Problemstellung in der Robotik vorgestellt und auf MR übertragen. Darauffolgend wird das Lochkameramodell dargelegt und Kamerakalibrierung näher betrachtet. Danach wird ein Einblick in im Kontext AR/MR eingesetzte visuelle Tracking-Verfahren gegeben.

Anschließend werden die zur Entwicklung des Prototypen eingesetzten Softwarelösungen und Bibliotheken vorgestellt. Beginnend mit der Open Computer Vision Library als Werkzeug zur Bildverarbeitung und -analyse, wird anschließend, das Robot Operating System (ROS) als eingesetzte Middleware näher betrachtet. Anschließend folgt ein Einblick in die Unity Engine, welche als Zielplattform des Prototypen definiert wird.

2.1 Mixed Reality

Mixed Reality zählt zu den sich rasant entwickelnden immersiven Technologien (Aithal und Aithal, 2015). Große Technologieunternehmen – wie Microsoft – haben diesen Trend frühzeitig erkannt und mit der HoloLens maßgeblich zur Entwicklung heutiger MR-Systeme beigetragen (Noor, 2016). Microsofts Interpretation der Mixed Reality beruht auf den bereits Mitte der 1990er Jahre veröffentlichten Erkenntnissen von Milgram und Kishino (Bray, 2021).

Paul Milgram ist Professor für Mechanical und Industrial Engineering an der Universität von Toronto (Univ. Toronto, 2021) und Fumio Kishino hat einen Lehrstuhl an der

Universität von Osaka inne (IEEE Explore, 2021). Beide forschen seit über 30 Jahren an Themen der Mensch-Maschinen-Interaktion, sowie virtuellen- und augmentierten Realitäten. Sie definierten zuerst den Begriff Mixed Reality (Milgram et al., 1995). Seit der Publikation des Aufsatzes „*Augmented Reality: A class of displays on the reality-virtuality continuum*“ zählen die Veröffentlichungen des Autorenteams zu den Grundlagenwerken im Bereich Mixed- und Extended Reality (Dörner et al., 2013). Dementsprechend werden in dieser Arbeit Milgrams und Kishino's Erkenntnisse als Basis zur Vorstellung der Mixed Reality verwendet.

2.1.1 Reality-Virtuality-Continuum

Um das Spektrum der Mixed Reality in den Kontext aller bekannten Realitäten einordnen zu können, haben Milgram et al. das Konzept des Reality-Virtuality Continuum (RV-Continuum) entwickelt. Dadurch kann das Spektrum von der physischen Realität bis zur vollständig virtuellen Realität beschrieben werden (Milgram et al., 1995).

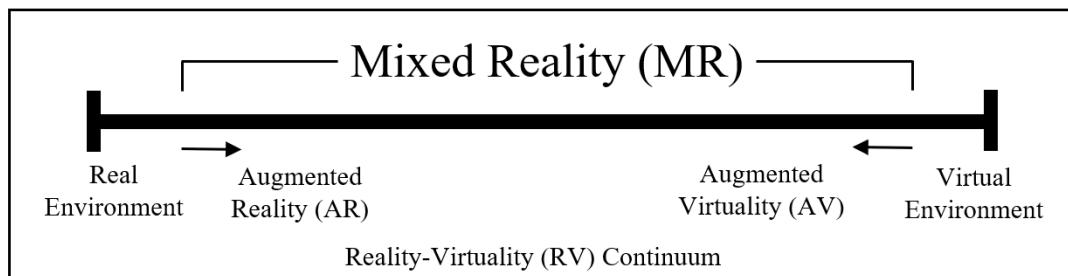


Abbildung 1: Reality-Virtuality Continuum in Anlehnung an (Milgram et al., 1995)

Abbildung 1 zeigt das RV-Continuum. Die äußersten Extrema des Spektrums stellen die reale, sowie die vollständig virtuelle Umgebung dar. Mixed Reality ist somit – laut Milgram et al. – ein Spektrum, welches die Augmented Reality (AR), sowie die Augmented Virtuality (AV) einschließt (Milgram und Kishino, 1994). Da MR als Spektrum definiert ist, lassen sich unterschiedliche Interpretationen der MR formulieren. Je nachdem, ob virtuelle oder reale Anteile der Szene überwiegen, kann eine neue Interpretation der MR erfolgen (Bray, 2021).

Nachfolgend wird daher zunächst Virtual Reality und anschließend die menschliche Wahrnehmung in virtuellen Realitäten betrachtet. Darauffolgend werden AR und AV näher erläutert und voneinander abgegrenzt.

2.1.2 Virtual Reality

Die heutige VR-Technik basiert auf den Erkenntnissen des ehemaligen Harvard-Professor Ivan Sutherland (Wikipedia, 2021b) und seinem 1965 veröffentlichten Konzept des Ultimate Display in der Forschungsarbeit „*The ultimate display*“ (Sutherland, 1965). Drei Jahre später veröffentlichte Sutherland den Aufsatz „*A head-mounted three dimensional display*“, welcher den Grundstein für heutige HMDs legte (Sutherland, 1968). Im Laufe der letzten Jahre führten technologische Fortschritte – wie die Entwicklung performanterer Hardware und Sensorik – zu einer sprunghaften Weiterentwicklung von VR-Technologien (Aithal und Aithal, 2015, Serafin et al., 2018).

Die virtuelle Realität beschreibt eine computergenerierte alternative Realität, in die ein Nutzer mittels einer Mensch-Maschinen-Schnittstelle eintauchen kann (Dörner et al., 2013). Diese virtuelle Realität ist nur durch die Kreativität des Entwicklers und der zur Verfügung stehenden Technologie limitiert (Graessler und Taplick, 2019). Heute sind einige Methoden und technische Geräte bereits im Stande den Tastsinn anzusprechen oder Berührungen zu simulieren (Al-Sada et al., 2020). Jedoch ist die vollständige Verlagerung der gesamten menschlichen Wahrnehmung in eine alternative Realität noch Science-Fiction (Dörner et al., 2013).



Abbildung 2: Beispiel einer VR-Anwendung (Sattler, 2021)

Abbildung 2 zeigt eine VR-Anwendung, welche vom Max-Plack-Institut für Bildungsforschung in unterschiedlichen Studien eingesetzt wird (Sattler, 2021). Hierbei nutzt ein Studienteilnehmer ein HMD, um audiovisuelle Inhalte wahrzunehmen. Zudem werden die Kopfbewegungen über interne Messinstrumente in die virtuelle Welt übertragen.

Dadurch wird die Blickrichtung des Nutzers in die Computersimulation inkludiert und somit die Eingrenzung der virtuellen Szene auf den aktiven virtuellen Teilabschnitt ermöglicht (Sattler, 2021).

Heute existiert keine einheitliche Definition der VR (Boyd und Koles, 2019). Vielmehr werden die Definitionen in Abhängigkeit der Perspektive formuliert. Sutherland identifiziert eher technische Merkmale und formuliert dementsprechend technologische Aspekte – bezogen auf den Einsatz von HMDs im Kontext VR – und geht weniger auf die User Experience bzw. Immersion ein (Sutherland, 1968, Sutherland, 1965).

Dörner et al. beschreiben ein VR-System als Computersystem, bestehend aus geeigneter Hard- und Software, um eine digitale synthetische Realität zu erschaffen. Dörner et al. bezeichnet die durch das VR-System dargestellten Inhalte als Virtuelle Welt. Konkret umfasste die Virtuelle Welt das Verhalten und die physikalischen Gegebenheiten aller virtuellen Objekte in der künstlichen Realität. Die Kombination der Virtuellen Welt und dem VR-System wird durch Dörner et. al als Virtuelle Umgebung bezeichnet (Dörner et al., 2013). Dörner et al. stellen technologische Aspekte zwar in den Vordergrund, binden allerdings die User Experience und Immersion mit ein. Die Definition beinhaltet zum einen, die für eine VR-Erfahrung notwendige Hardware in Form eines geeigneten Wiedergabemediums und zum anderen die funktionalen Anforderungen an die VR-Umgebung (Dörner et al., 2013).

Bryson et al. beschreiben die VR als ein System, welches sich auf den Einsatz virtueller, dreidimensionaler Objektdarstellungen und dazugehöriger Interaktionsmedien spezialisiert (Bryson, 1993). Das System ermögliche die Erkundung computergenerierter Umgebungen in Echtzeit. Demnach basiere VR zwar auf der Computergraphik, fokussiere allerdings Echtzeitfähigkeit, um die Immersion nicht zu brechen und somit die Präsenz des Nutzers in der synthetischen Welt nicht zu stören (Bryson, 1993).

Zusammenfassend korrelieren die Definitionen von Bryson und Dörner, jedoch fällt die Einbindung der User Experience im Vergleich zu Sutherlands Definition auf. Im weiteren Verlauf der Arbeit werden die Definition bzw. Merkmale der VR von Dörner et al. angenommen und in Kapitel 4 zur Anforderungsdefinition des Prototypen herangezogen.

2.1.3 Menschliche Wahrnehmung in virtuellen Realitäten

Grundsätzlich nehmen Menschen die Realität unterschiedlich wahr (Dörner et al., 2013). Wir verfügen zwar in der Regel über die gleichen Möglichkeiten die Außenwelt visuell wahrzunehmen, verarbeiten und Filtern diese Informationen jedoch meist unterschiedlich (Snowden, Thompson und Troscianko, 2012, Hubel und Wiesel, 2005). Um nun die menschliche Wahrnehmung im Kontext virtueller Realitäten näher zu analysieren, stellen Ralf Döner et al. die in VR-Erfahrungen bereits berücksichtigten bzw. in die VR verlagerten Sinne vor (Dörner et al., 2013).

Ralf Döner ist seit 2004 Professor für virtuelle Realitäten und graphische Datenverarbeitung an der Hochschule RheinMain in Wiesbaden und hat die Monographie „*Virtual und Augmented Reality*“ mitverfasst (Hochschule RheinMain, 2021). Döner stellt den aktuellen Stand der Technik, bezogen auf die Virtualisierung der menschlichen Sinne dar. Hierbei werden laut Döner zu der synthetischen Stimulation der akustischen und visuell empfänglichen menschlichen Sinne, zudem der haptische bzw. der Tastsinn bereits erfolgreich angesprochen (Döner et al., 2013). Al-Sada et al. haben dem weiteres hinzugefügt und mit der Vorstellung ihres Systems bereits erfolgreich demonstriert, dass sich weitere Sinne durch tragbare Vorrichtungen ansprechen lassen (Al-Sada et al., 2020).

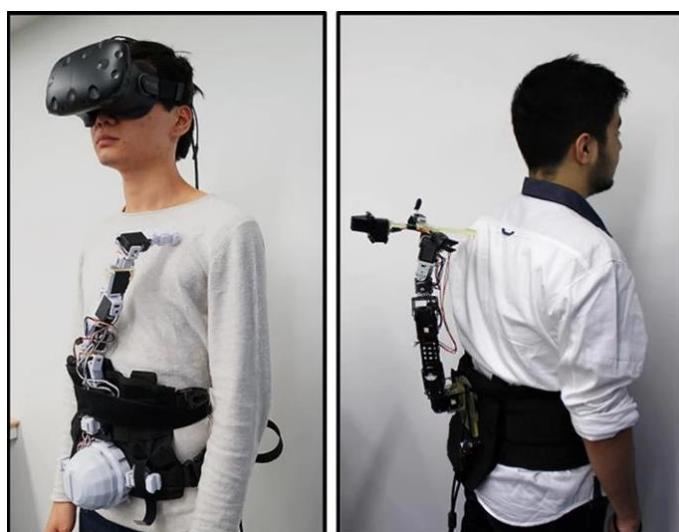


Abbildung 3: Haptic Snake (Al-Sada et al., 2020)

Abbildung 3 zeigt die von Al-Sada et al. entwickelte Haptic Snake, die es ermöglichen soll, unterschiedliche Körperteile durch generierbaren Luftzug auf die Haut zu stimulieren

(Al-Sada et al., 2020). So können bspw. hektische Bewegungsabläufe oder der Fahrtwind einer rasanten Autofahrt simuliert werden. Durch den Roboterarm können zudem Berührungen aus der virtuellen in die reale Welt übertragen werden.

Nichtsdestotrotz gilt die vollständig immersive Virtuelle Realität aktuell als nicht realisierbar (Dörner et al., 2013). Hierbei müsste das gesamte Wahrnehmungsvermögen und eine physische Repräsentation des Menschen in eine computergenerierte Realität transferiert werden können. Somit müssten zudem alle physischen Handlungen und Bewegungen in äquivalente virtuelle Auswirkungen transferiert werden können und umgekehrt (Dörner et al., 2013). Um die Illusion aufrecht zu erhalten, darf keine Unterscheidung zwischen physischer und künstlicher Realität mehr möglich sein, andernfalls würde ein Bruch zwischen beiden Realitäten entstehen (Dörner et al., 2013). Sutherland warnt schon in den 60er Jahren von den Gefahren vollständiger Immersion in virtuellen Realitäten (Sutherland, 1965).

Im weiteren Verlauf der Arbeit wird unter der virtuellen Realität, respektive gemischten Realitäten, lediglich die Wahrnehmung visueller Inhalte verstanden und die Simulation weiterer Sinneseindrücke vernachlässigt.

2.1.4 Augmented Virtuality

Innerhalb des RV-Continuum nach Milgram bezeichnet die Augmented Virtuality eine computergenerierte virtuelle Welt – ähnlich einer VR – welche durch die Projektion von Teilen der physischen Realität angereichert wird (Milgram et al., 1995). Hierbei ist das Verhältnis zwischen virtuellen und realen Objekten in der Szene entscheidend, wobei die synthetischen Elemente überwiegen müssen (Milgram et al., 1995). Nur dann handelt es sich um AV (Milgram et al., 1995). Oft werden Gliedmaßen oder ganze Personen in eine künstliche Realität projiziert (Bruder et al., 2009).



Abbildung 4: Beispiel Augmented Virtuality (Bruder et al., 2009)

Abbildung 4 zeigt ein AV-Szenario, indem die virtuelle Welt durch physische Elemente in Form von den Händen des Nutzers angereichert wird. Bruder et al. nutzen dazu ein HMD mit zusätzlichen externen Kameras zur Erfassung reale Objekte (Bruder et al., 2009).

2.1.5 Augmented Reality

Augmented Reality stellt den Gegenpol zu Augmented Virtuality im RV-Continuum nach Milgram dar (Milgram und Kishino, 1994). Laut Milgram wir das natürliche Feedback an den Bediener durch simulierte Hinweise verstärkt und über optical see-through displays als Ausgabemedium erfahren (Milgram et al., 1995). Dörner beschreibt AR als interaktive und echtzeitfähige Erweiterung der Wahrnehmung physischer Umgebungen durch virtuelle Inhalte (Dörner et al., 2013).



Abbildung 5: Beispiel Augmented Reality (Dörner et al., 2013)

Abbildung 5 zeigt eine Augmented-Reality-Anwendung, welche einen physischen Marker verwendet. Dadurch wird die Projektionsfläche identifizierbar und die Kamerapose kann in Relation zur erfassten Pose des Markers ermittelt werden. Dies ermöglicht die Positionierung und damit Projektion des virtuellen Objektes in die Realität. Die physische Realität wird um virtuelle Objekte erweitert (Dörner et al., 2013).

Laut Azuma ermöglicht AR dem Benutzer die reale Welt über ein geeignetes Medium wahrzunehmen (Azuma, 1997). Das Medium ermöglicht es, die Realität mit virtuellen Objekten zu überlagern oder dadurch zu erweitern und nicht wie in VR-Szenarien zu ersetzen (Azuma, 1997). Ronald Azuma ist Teamleiter bei Intel Labs und forscht an Schlüsseltechnologien für neue Medien – insbesondere VR und AR. Azuma gilt als einer der Pioniere in der Erforschung von AR-Technologien (Azuma, 2021).

Azumas Definition der AR korreliert mit der Milgrams, sowie Dörners. Dadurch wird eine zusammenfassende Betrachtung aller Erkenntnisse bzgl. augmentierter Realitäten möglich. Wird Milgrams Definition der MR als Spektrum angenommen, stellen AR und VR Bestandteile dieses Spektrums dar (Milgram und Kishino, 1994). Daher lassen sich die geltenden Anforderungen für AR und VR auch auf MR übertragen (Dörner et al., 2013).

Die daraus resultierenden Anforderungen werden im weiteren Verlauf für MR angenommen und in Kapitel 4 zusammengetragen bzw. auf dieser Grundlage der Entwurf des Prototypen vorgenommen. In Kapitel 3.1 wird die in dieser Arbeit fokussierte Interpretation der MR vorgestellt.

2.2 Spatial Computing

Simon Greenwold stellte den Begriff Spatial Computing in seiner Abschlussarbeit „*Spatial Computing*“ an der Yale University am Massachusetts Institute of Technology im Jahre 1995 erstmals vor, worauf 2003 die Veröffentlichung der Thesis folgte (Greenwold, 2003). Hierbei stellt der Forscher und Systemdesigner für Mensch-Maschine-Interaktion (Greenwold, 2021) Spatial Computing als Extraktion und Nutzbarmachung von räumlicher Information bspw. zur visuellen Representation vor (Greenwold, 2003). Dadurch wird die Verschmelzung der physischen Umgebung mit den Informationen der virtuellen Welt ermöglicht (Greenwold, 2003, Çöltekin et al., 2020).

Spatial Computing nutzt den umgebenden Raum als Eingabemedium, um virtuelle Objekte in die lokale physische Umgebung zu integrieren, sowie die Physis in die Virtualität einfließen zu lassen (Greenwold, 2003). Dies bedingt den Einsatz von geeigneten Sensoren für die Extraktion der Umgebungsdaten (Greenwold, 2003). Durch die geringe Bauform, Kosteneffizienz und der heute vorhandenen Rechenleistung, eignen sich Kameras aufgrund der reichhaltigen Bildinformation besonders (Greenwold, 2003, Sualeh und Kim, 2019).

Im Gegensatz zu MR wird bei Spatial Computing nicht das Medium bzw. die virtuelle Interpretation ins Zentrum gestellt, sondern das Problem aus einer allgemeinen bzw. technischen Perspektive behandelt (Çöltekin et al., 2020). Demnach ist Spatial Computing als Überbegriff für eine eigene Disziplin oder Problemkategorie zu verstehen, unter dieser MR als Werkzeug zur Visualisierung und als Realitätsdefinition verstanden werden kann (Greenwold, 2003).

Spatial-Computing-Technologien werden heute bspw. in Kombination unterschiedlicher Sensoren, Recheneinheiten und Zusatzgeräten eingesetzt. So können bspw. mittels haptischen Geräten – wie in Kapitel 2.1.3 vorgestellt – und optischer Sensorik Roboter gesteuert werden (Tian et al., 2021). Hierbei generiert das haptische Gerät entsprechendes Berührungsfeedback und eine in den Roboter angebrachte Kamera ermöglicht die Erfassung der Raumgeometrie (Tian et al., 2021). Diese Kombination ermöglicht es den Nutzern ein Gefühl der realitätsnahen Präsenz zu erfahren (Tian et al., 2021). Durch die Fernsteuerung können Operatoren ortsabhängig agieren und in Echtzeit haptisch unterstützt arbeiten (Tian et al., 2021).

Spatial Mapping bezeichnet den Prozess der Erfassung und das Verständnis der physischen Gegebenheiten und Darstellung der extrahierten Information durch eine persistente Karte (Greenwold, 2003, Park et al., 2020, Kress, 2020). Die Umsetzung erfolgt hierbei oft über die Identifikation markanter Bildmerkmale und die Erstellung einer Karte des Raumes (Çöltekin et al., 2020). Im Allgemeinen wird der Prozess der Kartierung (Spatial Mapping) auch als Teilbereich von Simultaneous Localization and Mapping verstanden (Moezzi et al., 2020) und daher in Kapitel 2.4 vorgestellt. MR-Anwendungen benötigen geeignete Ausgabegeräte (Dörner et al., 2013) und Spatial Computing entsprechende Sensoren zur Umgebungsanalyse (Greenwold, 2003), daher werden in Kapitel 2.3 Head Mounted Displays als Ein- und Ausgabegeräte im Kontext MR vorgestellt.

2.3 Head-Mounted Displays

Zuvor wurde Mixed Reality als Spektrum näher betrachtet und die Teilbereiche AR, AV und VR voneinander abgegrenzt. Eine Gemeinsamkeit der unterschiedlichen Realitäten liegt bei den verwendeten Ausgabegeräten (Dörner et al., 2013). Spatial Computing nutzt vorhandene Umgebungsinformationen als Eingabedaten und macht so den Raum für den Computer interpretierbar (Greenwold, 2003). MR bedingt ein Ausgabemedium, welches oft in Form eines HMDs vorliegt (Costanza, Kunz und Fjeld, 2009). Spatial-Computing-Technologien benötigen Sensoren, welche die Umgebung erfassen können (Greenwold, 2003). Diese werden oft in HMDs in Form von Kameras verbaut (Evans et al., 2017). Nachfolgend werden daher HMDs als Ein- sowie Ausgabegeräte für MR-Anwendungen im Kontext Spatial Computing vorgestellt.

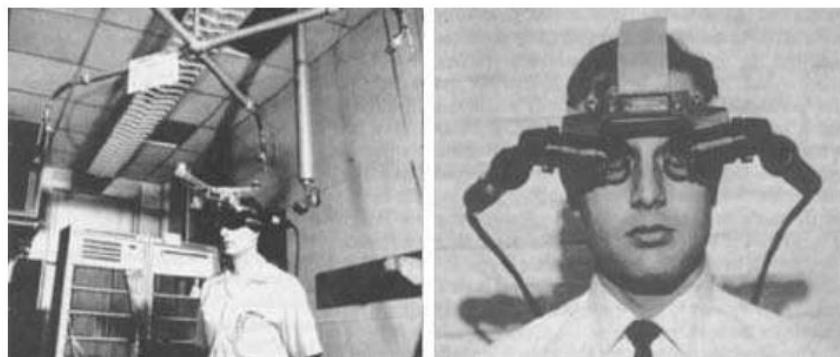


Abbildung 6: "Sword of Damocles" (Sutherland, 1968)

Ivan E. Sutherland gilt nicht nur als Pionier der Computergrafik, sondern auch der virtuellen Realität, sowie als Urvater der HMDs (Wikipedia, 2021b). Mit der Entwicklung des „*Sword of Damocles*“ (Abbildung 6) konstruierte Sutherland den ersten Prototypen der heutigen HMDs (Sutherland, 1968). Es handelte sich um eine der ersten AR-Erfahrungen, welche es ermöglicht, virtuelle Objekte in das Sichtfeld des Nutzers zu projizieren (Sutherland, 1968). Die Kopfbewegungen des Nutzers werden als Eingabe erfasst (Sutherland, 1968). Dadurch bewegen sich virtuelle Objekte synchron zur Kopfbewegung des Nutzers. Die künstlich erzeugten Objekte, werden über Kathodenröhren zu beschichteten Spiegeln transportiert und von dort direkt ins Auge des Nutzers projiziert (Sutherland, 1968).

HMDs können in zwei grundlegende Kategorien klassifiziert werden (Dörner et al., 2013). Video see-through Displays zeigen dem Nutzer die Realität über ein in das HMD

integriertes Display (Dörner et al., 2013). Die einfachste Variante der Video see-through Displays stellen Videobrillen dar, welche lediglich virtuelle Inhalte über eine in das HMD integrierte Displayeinheit anzeigen (Dörner et al., 2013). Virtual Reality Displays (VR-Display) stellen die Weiterentwicklung der Videobrillen dar und ermitteln mit eingebauten Sensoren die Kopfposition (Dörner et al., 2013). Dadurch kann die Blickrichtung in die virtuelle Welt übertragen werden (Dörner et al., 2013).

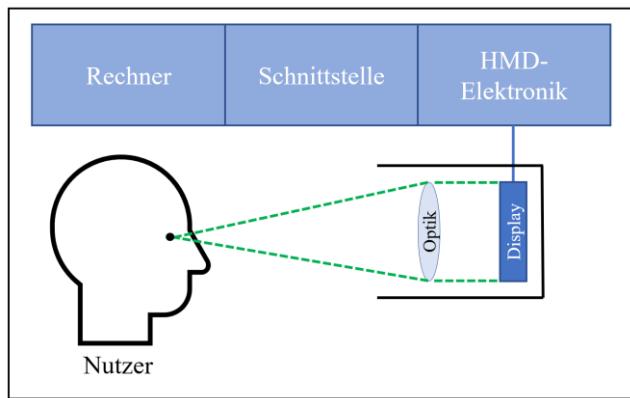


Abbildung 7: Video see-through Display in Anlehnung an (Dörner et al., 2013)

Abbildung 7 zeigt eine vereinfachte Darstellung eines Virtual Reality Displays (Dörner et al., 2013). Die Recheneinheit kann hierbei intern verbaut oder an das VR-Display angeschlossen werden (Dörner et al., 2013). Letzteres limitiert die Bewegungsfreiheit, sowie die Einsatzreichweite des Geräts. Die Verbindung zwischen allen elektronischen Komponenten des VR-Displays werden über unterschiedliche Schnittstellen gewährleistet (Dörner et al., 2013). Dadurch kann die durch die Recheneinheit aufbereitete Information über ein Display ausgegeben werden (Dörner et al., 2013). Der Nutzer blickt hierbei über eine optische Vorrichtung auf das in der Brille verbaute Display und kann so die virtuellen Objekte visuell wahrnehmen (Dörner et al., 2013). Für den weiteren Verlauf der Arbeit werden Video see-through Displays nicht weiter berücksichtigt.

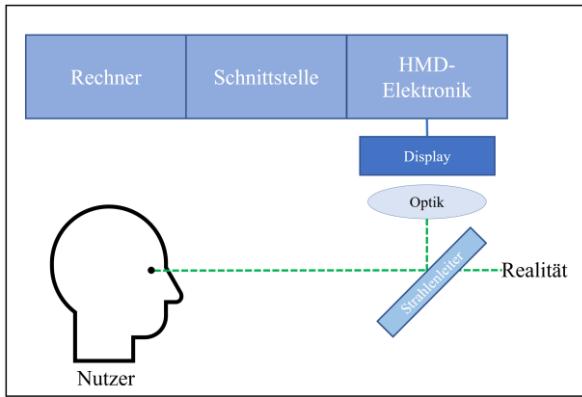


Abbildung 8: Optical see-through Display in Anlehnung an (Dörner et al., 2013)

In Abbildung 8 wird ein generalisiertes Modell eines optical see-through Displays dargestellt (Dörner et al., 2013). Optical see-through Displays ermöglichen es dem Nutzer, die physische Realität direkt wahrzunehmen, ohne den Umweg über eine Displayeinheit zu gehen. Dies ermöglicht die direkte Wahrnehmung der realen Umgebung in unveränderter Qualität und in der anatomisch möglichen Bildwiederholungsrate, die nicht durch Berechnungen beeinflusst wird (Dörner et al., 2013).

Diese Technik findet üblicherweise in AR/MR-Systemen Anwendung (Azuma, 1997). Hierbei werden alle im Szenenabschnitt ersichtlichen virtuellen Objekte in die dargestellte physische Realität projiziert (Azuma, 1997). Je nach Einsatzzweck können zusätzliche Sensoren bspw. zur Umgebungsobservation oder Lokalisierung des Geräts verbaut werden (Kress, 2020). Optical see-through Displays werden heute in unterschiedlichen Konfigurationen in Forschung und Technik genutzt. Der grundliegende Aufbau bleibt allerdings im Wesentlichen ähnlich (Kress, 2020). In Kapitel 3.2 wird die Microsoft HoloLens als Beispiel für ein aktuelles optical see-through Display vorgestellt.

2.4 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM), wurde von Randall Smith und Peter Cheeseman mit dem Aufsatz “*On the Representation and Estimation of Spatial Uncertainty*” 1986 vorgestellt (Smith und Cheeseman, 1986). Randal C. Smith war von 1986 bis 2008 als Projektleiter bei General Motors tätig und hierbei für die Weiterentwicklung und Forschung in den Bereichen Virtual Reality, immersiven Stereo-Displays und magnetischem Tracking zuständig (ACM-DL, 2021b). Peter Cheeseman forscht am NASA Ames Research

Center in Kalifornien an maschinellem Sehen und Künstlicher Intelligenz (ACM-DL, 2021a).

Das Akronym SLAM wurde erstmals 1996 von Durrant-Whyte et al. (Durrant-Whyte, Rye und Nebot, 2000) verwendet und hat sich in Forschung und Entwicklung als Sammelbegriff für unterschiedliche Lösungsstrategien hinsichtlich SLAM etabliert (Taheri und Xia, 2021). Durrant-White ist Professor für Maschinenbau und forscht an Themen der Robotik, sowie Maschinellem Lernen. Er gilt als Pionier im Forschungsgebiet SLAM (Wikipedia, 2021a).

Als Simultaneous Localization and Mapping wird die Problemstellung verstanden, in der ein Roboter vor der Aufgabe steht, sich in einer unbekannten Umgebung mittels den eigenen Sensoren selbst zu lokalisieren und sich dadurch im Raum zu orientieren (Durrant-Whyte und Bailey, 2006). Hierbei muss zeitgleich zur Selbstlokalisierung eine Karte des Raumes erstellt werden, um die Pose relativ zur Observation ermitteln zu können (Stachniss, 2009).

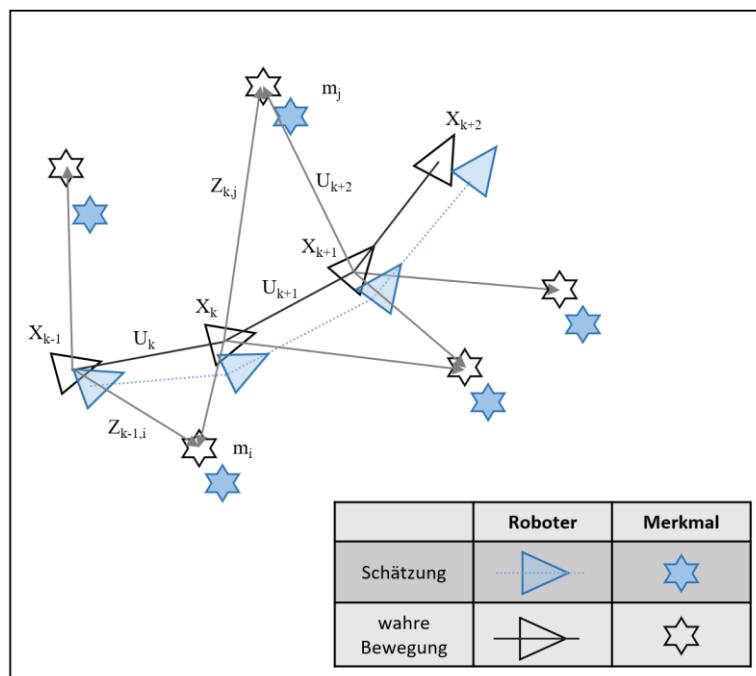


Abbildung 9: Das SLAM-Problem in Anlehnung an (Durrant-Whyte und Bailey, 2006)

Abbildung 9 zeigt die Essenz des SLAM-Problems (Durrant-Whyte und Bailey, 2006). Ein Roboter steht vor der Aufgabe sich selbst in einer sich erst entwickelten Karte zu orientieren. Zeitgleich wird die Karteninformation zur Selbstlokation verwendet (Durrant-Whyte und Bailey, 2006). Hierbei werden die Sensoren des Roboters genutzt, um markante

Raummerkmale zu detektieren. Die Pose der Merkmale wird relativ zur Pose des Roboters ermittelt und in eine Karte übertragen (Durrant-Whyte und Bailey, 2006). Dadurch lässt sich die zurückgelegte Wegstrecke mit der Zeit nachverfolgen, sowie die Position der Merkmale im Raum tracken (Bailey und Durrant-Whyte, 2006). Allerdings können sich die Positionsschätzungen u.U. deutlich von der wahren Fortbewegung abheben (Bailey und Durrant-Whyte, 2006). Demzufolge sind weitere Optimierungsschritte erforderlich, um die Lokationsdaten zu präzisieren (Durrant-Whyte und Bailey, 2006, Thrun, Burgard und Fox, 2005).

In Abbildung 9 wird die geschätzte Pose in blau und die sog. Ground Truth – also die tatsächliche Trajektorie zum Zeitpunkt k – in schwarz visualisiert. Letztere wird häufig durch externe physische Messungen ermittelt (Wulf et al., 2007). Der Roboter hat üblicherweise während der Systemlaufzeit ausschließlich Zugang zur eigenen Schätzung in blau. X_k stellt hierbei die aktuelle Pose des Roboters dar. U_k ist der Kontrollvektor, welcher die Trajektorie zum nächsten Messpunkt X_{k+1} zeigt. Jedes Merkmal wird als Landmark bezeichnet, sobald die dreidimensionale Position bekannt bzw. eine erfolgreiche Observation erfolgt ist (Durrant-Whyte und Bailey, 2006). Die Pose der Landmarks wird mit m_j zeitunabhängig beschrieben. Z_{ik} beschreibt die Observation eines Merkmals der Szene zum Zeitpunkt k (Durrant-Whyte und Bailey, 2006).

SLAM-Systeme können unterschiedliche Aufgaben wahrnehmen und werden daher auch für verschiedene Anwendungen verwendet (Huang Baichuan, Zhao und Liu, 2019). SLAM hat seinen Ursprung in der Robotik und wird demnach primär zur autonomen Navigation verwendet (Reitmayr et al., 2010). Dadurch kann die Wegplanung zur Navigation in unbekanntem Umfeld oder die Kollisionsvermeidung mit Hinderniserkennung realisiert werden (Cadena et al., 2016, Bryson und Sukkarieh, 2005).

Im Kontext AR/MR und in Verbindung mit einem Ausgabegerät, wird SLAM neben der Lokalisierung, unterstützend zur Identifikation physischer Objekte oder zum Tracking von Merkmalen eingesetzt (Vlaminck, Luong und Philips, 2017). Zudem ist es durch SLAM möglich, bei AR-/MR-Anwendungen, auf die Verwendung künstlicher Marker zu verzichten und natürlich in der Umgebung vorkommende Merkmale zu identifizieren und tracken (La Viola et al., 2017). Im Bereich des 3D-Scannings werden SLAM-Methoden zur Erstellung

von dreidimensionalen Modellen für die Kartierung von bspw. archäologischen Fundorten genutzt (Pierzchała, Giguère und Astrup, 2018, Gautier et al., 2020).

Zusammenfassend betrachtet kann SLAM mittels unterschiedlichen Sensoren realisiert werden (Sualeh und Kim, 2019). Da jedoch MR in Verbindung mit einem HMD eingesetzt wird, können nur Sensoren eingesetzt werden, welche eine geringe Versorgungsspannung benötigen. Außerdem werden die Sensoren üblicherweise in die HMDs fest integriert, weshalb die Sensoren kompakt sein müssen (Evans et al., 2017). Das HMD wird i.d.R. mehrere Minuten auf dem Kopf getragen und sollte daher nicht sehr schwer sein (Kress, 2020, Dörner et al., 2013).

Zudem werden reichhaltige Informationen zur Positionsbestimmung für die SLAM-Algorithmen benötigt. Kameras bieten sich zur Informationsextraktion aus der Umgebung besonderes an (Tang und Cao, 2020), weshalb Kamerasysteme in Kapitel 2.5 vorgestellt werden. In Kapitel 3.3 wird dementsprechend visual SLAM als SLAM-Variation vorgestellt, welche sich auf die Verwendung von Kameras als Primärsensor spezialisieren (Filipenko und Afanasyev, 2018).

2.5 Kamerasyteme und -modelle

Aktuelle Kameras verwenden complementary metal oxide semiconductor sensors (CMOS), die auf einstrahlendes Licht reagieren und durch angeschlossene Transistoren entsprechende Signale schalten (Schmidt, 2013). CMOS-Sensoren bestehen aus auf einer Platine angeordneten licht-reaktiven Fotodioden, welche einfallende Lichtstrahlen in digitale Signale umwandeln. Abbildung 10 zeigt die Sensorplatine, welche etliche Sensoren vereint (Schmidt, 2013). Hierbei können die aufgeladenen Dioden direkt adressiert werden, was die Zuordnung eines Lichtstrahles zu dem jeweiligen Sensor ermöglicht und als ein einzelner Pixel interpretiert werden kann (Schmidt, 2013). Daher kann ein Koordinatensystem genutzt werden, um den jeweils stimulierten Sensor einem Pixel zuzuordnen. Üblicherweise werden die Koordinatensysteme in der Bildanalyse und -Verarbeitung in einer der Ecken des Bildes platziert (Jähne, 2005). Das Zentrum des Sensors wird mit o bezeichnet und stellt die Koordinaten des optischen Zentrums dar (Schreer, 2005). Das optische Zentrum des Kamerakalibrierung wird durch die Kamerakalibrierung ermittelt und in Kapitel 2.5.3 vorgestellt.

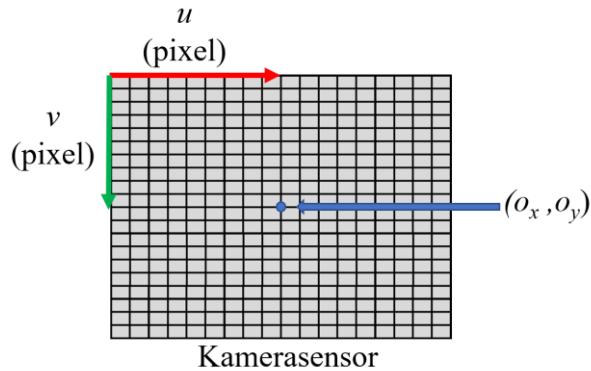


Abbildung 10: CMOS-Sensor in Anlehnung an (Gonzalez und Woods, 2007, Schreer, 2005)

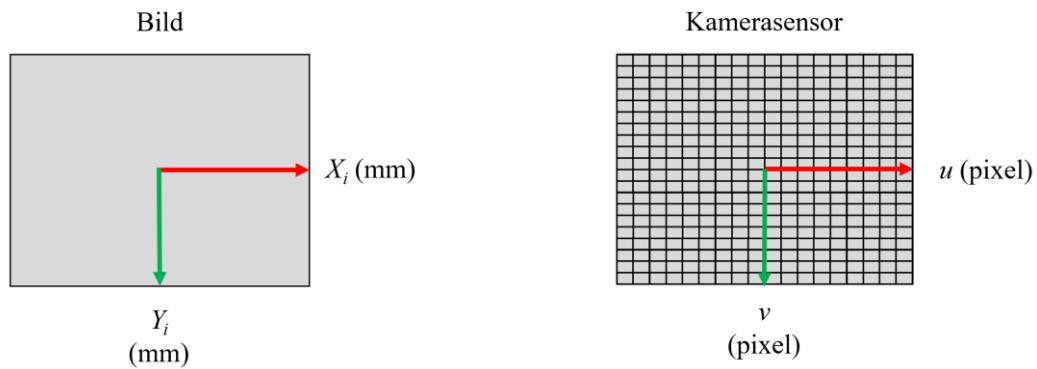


Abbildung 11: Beispielhafte Darstellung eines Bildsensors in Anlehnung an (Hartley und Zisserman, 2015, Gonzalez und Woods, 2007)

Abbildung 11 zeigt die beispielhafte Darstellung eines Bildsensors (rechts) und die aus den Sensordaten ermittelte Bildebene (links). Die linke Abbildung zeigt die Bildebene mit dem optischen Zentrum in der Mitte. In der Realität bildet das optische Zentrum nicht den exakten Ursprung des Koordinatensystems ab (Beyerer, Puente León und Frese, 2016). Es ist notwendig das optische Zentrum über ein entsprechendes Kalibrierungsverfahren zu verlagern, um die Berechnung der Rückprojektion zu vereinfachen (Beyerer, Puente León und Frese, 2016). Die Achsen werden hierbei in mm angegeben. Durch die Kalibrierung wird u.a. die Umrechnung von Pixeln in die metrische Korrespondenz ermöglicht (Beyerer, Puente León und Frese, 2016).

Im rechten Abschnitt der Darstellung (Abbildung 11) wird der Kamerasensor visualisiert. Ausgehend von der Annahme, dass die Pixel nicht quadratisch sein müssen, sondern auch im Einzelfall rechteckig, werden die Sensorpositionen durch zwei Koordinaten – u und v –

angegeben (Schmidt, 2013). Durch die Kalibrierung der Kamera kann das optische Zentrum der Kamera auf das Zentrum der Bild- bzw. Sensorebene verschoben werden und bildet dadurch die Grundlage für die Ermittlung der Pixelkorrespondenzen (Beyerer, Puente León und Frese, 2016).

2.5.1 Technischer Aufbau RGB-Kamera

Abbildung 12 zeigt den generalisierten Aufbau einer Kamera, bestehend aus dem Sensor in Form einer matrizenartigen Anordnung der Fotodioden und der Optik in Verbindung mit der Linse, welche den Brennpunkt beinhaltet (Schmidt, 2013). Zudem zeigt Abbildung 12 die Bildebene, welche den observierten Szenenabschnitt beherbergt. Das einfallende Licht wird hierbei durch die Optik gebündelt und auf die Sensormatrix projiziert (Schmidt, 2013). Kameras unterscheiden sich zunächst in der Bauform und dem Einsatzzweck (Schmidt, 2013). In diesem Teilkapitel werden jedoch RGB-Kameras fokussiert (Schmidt, 2013).

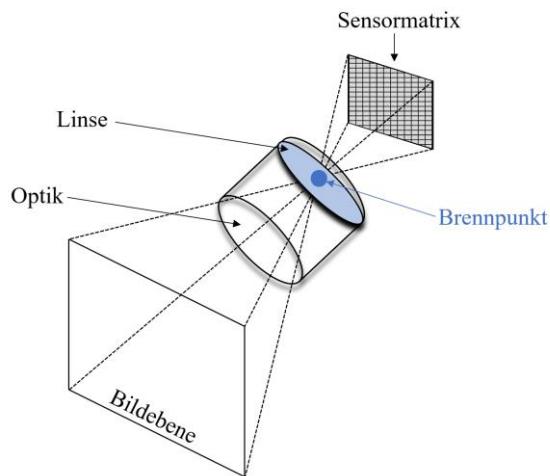


Abbildung 12: Beispiel einer Kamera in Anlehnung an (Schreer, 2005)

RGB-Kameras unterschieden sich in Bildwiederholungsrate, sowie der Qualität der produzierten Bilddaten (Schmidt, 2013). Die Bildqualität muss als ausreichend gut für die Extraktion validierter Merkmale aus den Bildinformationen klassifizierbar, insbesondere die Bildwiederholungsrate muss dazu entsprechend hoch sein (Jähne, 2005).

2.5.2 Das Lochkameramodell

Das Lochkameramodell beschreibt die vereinfachte Abbildung eines dreidimensionalen Objektes über das optische Zentrum auf eine zweidimensionale Bildebene (Schreer, 2005). Abbildung 13 zeigt das Lochkameramodell, welches seinen Namen dem sich in der Brennebene befindlichen Loch (Brennpunkt) verdankt (Schreer, 2005). Die Brennebene stellt hierbei zudem die Fokalebene dar und definiert so den Fokalabstand zur Bildebene (Schreer, 2005).

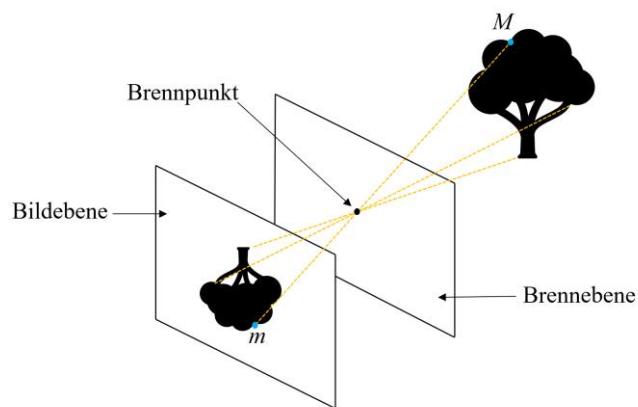


Abbildung 13: Dreidimensionale Projektion auf die Bildebene in Anlehnung an (Schreer, 2005, Hartley und Zisserman, 2015)

Abbildung 13 zeigt demnach die Projektion eines realen Objekts M auf die Bildebene. Die daraus resultierende Projektion wird durch die Abbildung gespiegelt und auf die Bildebene als Abbildung m projiziert. Steht das Projektionszentrum vor der Bildebene erzeugt dies ein horizontal und vertikal gespiegeltes Abbild (Schreer, 2005).

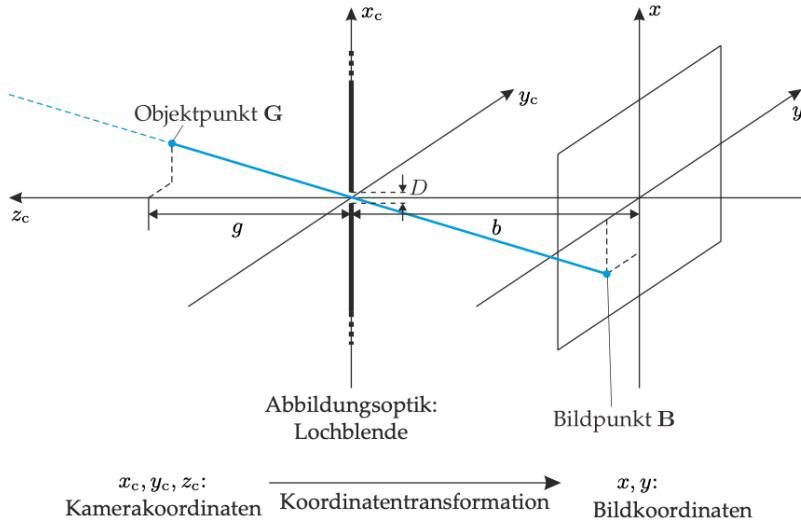


Abbildung 14 zeigt das schematische Modell einer Lochkamera und visualisiert die Spiegelung eines Bildpunktes am Projektionszentrum (Schreer, 2005). Dadurch wird die Abbildung des Objektpunktes G spiegelverkehrt auf die Bildebene als Bildpunkt B projiziert (Schreer, 2005). Die Abbildungsoptik beherbergt die Lochblende, welche den Ursprung des Kamerakoordinatensystems darstellt (X_c, Y_c, Z_c) und zeigt demnach von der Kamera weg. Die Koordinaten der Bildebene werden mit X und Y bezeichnet (Schreer, 2005).

Formel 1 zeigt die Zentralprojektion und verdeutlicht die Abbildung des Objektpunktes mit den Koordinaten X_c und Y_c auf einen Punkt auf der Bildebene und folgt aus dem geometrischen Strahlensatz (Schreer, 2005):

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \mapsto \begin{pmatrix} X \\ Y \end{pmatrix} = -\frac{b}{Z_c} \begin{pmatrix} X_c \\ Y_c \end{pmatrix}$$

Formel 1: Zentralprojektion (Hartley und Zisserman, 2015, Nayar, 2021, Schreer, 2005)

Das Lochkameramodell stellt ein idealisiertes System dar, wird jedoch oft für die Modellierung von Kamerasystemen herangezogen (Schreer, 2005). In der Theorie, ist die Schärfentiefe der Lochkamera unendlich (Schreer, 2005). Dies erlaubt die Annahme, ein Lichtstrahl ließe sich einem konkreten Pixel zuzuordnen (Schreer, 2005). Die Ermittlung der Parameter eines solchen Systems, wird auch als Kamerakalibrierung bezeichnet (Schreer, 2005) und in Kapitel 2.5.3 vorgestellt. Hierbei wird zwischen intrinsischen und extrinsischen

Parametern unterscheiden (Schreer, 2005). Die intrinsischen Parameter beziehen sich auf die internen Eigenschaften des Sensors bzw. der Kamera und die externen beschreiben die Lage und Orientierung (Pose) der Kamera im Raum (Schreer, 2005).

Generell muss das Lochkameramodell zur praktischen Anwendung erweitert werden (Schreer, 2005). Es werden bereits das Kamera- sowie das Bildkoordinatensystem entsprechend berücksichtigt (Schreer, 2005). Zur Rekonstruktion einer weltlichen Korrespondenz fehlt das Weltkoordinatensystem und die Relation zum Kamera- bzw. Bildkoordinatensystem (Schreer, 2005). Hierbei wird die Lage des Weltkoordinatensystems als statisch beschrieben, jedoch kann dies an jeder beliebigen Position in der Welt generiert werden (Schreer, 2005). Die Transformation der Welt- in Kamerakoordinaten, kann mittels einer Rotation und anschließender Translation beschreiben werden (Schreer, 2005). Formel 2 beschreibt die Relation des Welt- zum Kamerakoordinatensystem (Schreer, 2005):

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + t$$

Formel 2: Relation Welt- zu Kamerakoordinaten (Hartley und Zisserman, 2015, Nayar, 2021, Schreer, 2005)

2.5.3 Kamerakalibrierung

Kalibrierungsverfahren lassen sich in zwei Phasen unterteilen. Zunächst werden Bildpaare bestehend aus den ermittelten Korrespondenzen zwischen Welt und Bildpunkten identifiziert und extrahiert (Wilhelm Burger, 2016, Zhang, 2000). Anschließend werden aus den erhobene korrespondierenden Daten die entsprechenden Kameraparameter errechnet (Wilhelm Burger, 2016, Zhang, 2000).

Es ist erforderlich dem System eine weltliche Korrespondenz mit bekannten Abmessungen zu präsentieren (Zhang, 2000). Hierzu werden oft Schachbrettmuster verwendet (Abbildung 15), da diese eine regelmäßige Struktur aufweisen (Kaehler und Bradski, 2015). OpenCV biete hierzu nicht nur entsprechende Kalibrierungsfunktionen an, sondern kann zudem ein Kalibrierungsmuster in Form eines Schachbretts erstellen (Kaehler und Bradski, 2015). Die Dimensionen des Schachbrettmusters und demnach die Abmessungen der Quadrate sind bekannt. Die Lokalisation der Eckpunkte des Schachbrettmusters erfolgt dabei durch OpenCV durch Aufruf der entsprechende Funktion (Kaehler und Bradski, 2015).

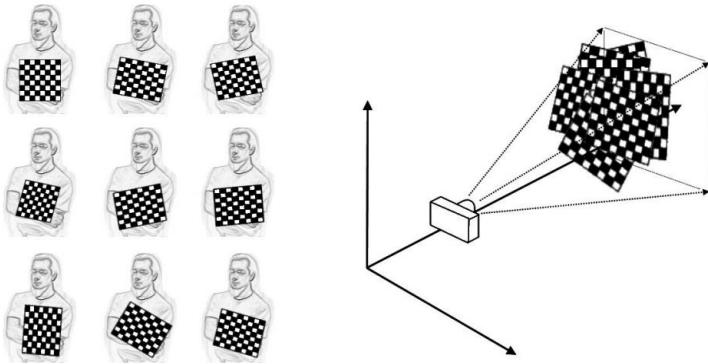


Abbildung 15: Darstellung der Kamerakalibrierung (Kaehler und Bradski, 2015)

Die Kenntnis über die Pose der Objektpunkte allein genügt nicht zur Kalibrierung der Kamera. Hierzu bedarf es zusätzlich der Ermittlung der Kameraparameter (Zhang, 2000). Dazu können mehrere Aufnahmen der Szene aus unterschiedlichen Perspektiven erstellt werden (Kaehler und Bradski, 2015). Dabei sollten die Aufnahmen zudem aus unterschiedlichen Entfernung zum Kalibrierungsmuster erstellt werden (Kaehler und Bradski, 2015). Zudem müssen ausreichend Korrespondenzen zur Berechnung zur Verfügung stehen (Kaehler und Bradski, 2015). OpenCV bietet dazu die Möglichkeit an, zunächst die entsprechenden Eckpunkte zu identifizieren und mit der maximal verfügbaren Anzahl der durch das Muster präsentierten Eckpunkte zu vergleichen (Kaehler und Bradski, 2015). Demnach ist es programmatisch möglich, eine gültige Aufnahme an die maximal verfügbare Korrespondenzmenge zu binden (Kaehler und Bradski, 2015).

Zur Kalibrierung werden der Kamera die Schachbrettmuster aus unterschiedlichen Entfernung und Perspektiven präsentiert (Kaehler und Bradski, 2015). Abbildung 15 fasst den Prozess zusammen. Entweder muss die Kamera oder das Muster statisch befestigt werden (Kaehler und Bradski, 2015). Es empfiehlt sich, das Muster in gedruckter Form statisch zu befestigen und die bestmögliche Representation zu erreichen. Im Idealfall sind beide Systeme statisch, d.h. die Kamera sowie das Muster an entsprechenden Vorrichtungen befestigt. Dies kann ideale Aufnahmebedingungen schaffen und reduziert verzerzte Bilder durch Bewegungen (Kaehler und Bradski, 2015).

Die Linsenverzerrung wird durch den OpenCV-Algorithmus automatisch berücksichtigt. Es werden mehrere Aufnahmen benötigt, sodass die Schätzung der Verzerrungsparameter und intrinsischen, sowie extrinsischen Parameter möglichst präzise erfolgt (Kaehler und

Bradski, 2015). Die Anzahl der Aufnahmen ist nicht klar definiert. Es werden 15-20 empfohlen, jedoch genügen dem Algorithmus bereits zwei (Kaehler und Bradski, 2015).

2.5.4 Perspektivische Projektion

Abbildung 16 zeigt das lineare Kameramodell, welches die Transformation eines dreidimensionalen Punktes P innerhalb des Weltkoordinatensystems W in das Bildkoordinatensystem I realisiert (Hartley und Zisserman, 2015, Wilhelm Burger, 2016). Das Kamerakoordinatensystem C befindet sich hierbei innerhalb von W . Die Z -Achse des Kamerakoordinatensystems vereint sich hierbei mit der optischen Achse. Dies ist durch den Wegfall der Tiefeninformation bei der Rückprojektion von 3D auf 2D zu begründen (Hartley und Zisserman, 2015). Der effektive Fokalabstand f stellt hierbei die Distanz zwischen der Bildecke und dem Projektionszentrum dar.

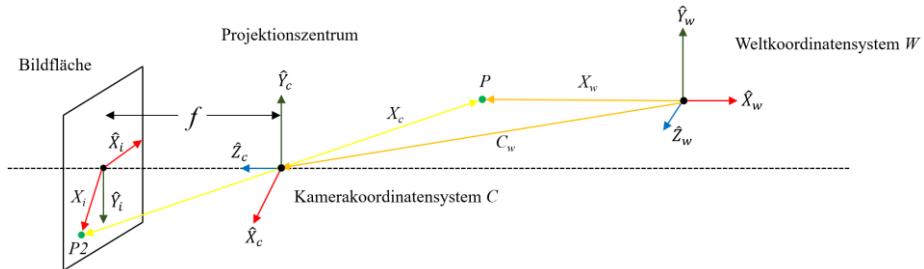


Abbildung 16: Lineares Kameramodell in Anlehnung an (Hartley und Zisserman, 2015, Nayar, 2021)

Es wird angenommen, dass die relativen Positionen und Orientierungen C_w von W bekannt sind. Dadurch kann die Transformation von einem Weltpunkt P bis hin zu dessen Projektion P_2 auf der Bildecke beschrieben werden. In Abbildung 17 wird der Ablauf der perspektivischen Rückprojektion eines Punktes im Weltkoordinatensystem auf einen Bildpunkt zusammengefasst visualisiert (Hartley und Zisserman, 2015).

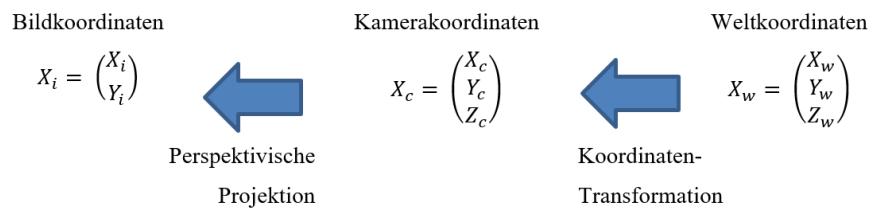


Abbildung 17: Projektion von Objektkoordinaten in Anlehnung an (Nayar, 2021, Hartley und Zisserman, 2015)

Die Perspektivische Projektion stellt die Transformation eines Weltpunktes auf die Bildfläche dar. Angenommen die Bildkoordinaten des Punkt $P2$ aus Abbildung 16 sind durch $\frac{x_i}{f} = \frac{x_c}{Z_c}$ und $\frac{y_i}{f} = \frac{y_c}{Z_c}$ bekannt. Abbildung 11 aus Kapitel 2.5.1 zeigt die Interpretation der Koordinaten in mm, jedoch liegen diese zunächst in Pixel vor (Hartley und Zisserman, 2015). Die Umrechnung von mm in Pixel erfolgt hierbei durch die Formeln 3 und 4:

$$u = m_x X_i = m_x f \frac{X_c}{Z_c}$$

$$v = m_y Y_i = m_y f \frac{Y_c}{Z_c}$$

Formel 3: Umrechnung Pixel in mm für u (Hartley und Zisserman, 2015)

Formel 4: Umrechnung Pixel in mm für v (Hartley und Zisserman, 2015)

Die Pixeldichte wird jedoch erst bei der Kamerakalibrierung exakt ermittelt. Hierbei wird angenommen die Pixeldichte, sowie die Position des Projektionszentrums (o_x, o_y) sei bereits bekannt, daher werden diese Parameter als unbekannte Variablen behandelt (Hartley und Zisserman, 2015).

Die Gleichungen lassen sich anschließend wie folgt vereinfachen (Formel 5 und 6), insbesondere die Lokation des Projektionszentrum o berücksichtigen:

$$u = f_x \frac{X_c}{Z_c} + o_x$$

$$v = f_y \frac{Y_c}{Z_c} + o_y$$

Formel 5: Vereinfachung von Formel 3 (Hartley und Zisserman, 2015)

Formel 6: Vereinfachung von Formel 4 (Hartley und Zisserman, 2015)

Der Parameter f stellt hierbei den Fokalabstand dar. Gesetz dem Falle, die Pixel seien nicht quadratisch, werden hierfür zwei Parameter verwendet (Hartley und Zisserman, 2015, Nayar, 2021). Die internen oder intrinsischen Parameter beziehen sich auf den Fokalabstand und die Position des Projektionszentrums und werden auch als die interne Geometrie bzw. intrinsische Parameter der Kamera bezeichnet (Hartley und Zisserman, 2015, Nayar, 2021). Durch Homogene Koordinaten lässt sich das System in einen linearen Ausdruck (Formel 7) verwandeln und als Matrix ausdrücken:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \equiv \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix}$$

Formel 7: Formel 3 und 4 mit homogenisierten Koordinaten

Der Ausdruck lässt sich weiter in die Kalibrierungsmatrix K (Formel 8), welche eine trianguläre Matrix darstellt und die Intrinsische Matrix M_{int} (Formel 9) zerlegen (Hartley und Zisserman, 2015, Nayar, 2021).

$$K = \begin{pmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{pmatrix}$$

Kalibrierungsmatrix:

Formel 8: Kalibrierungsmatrix (Hartley und Zisserman, 2015)

$$M_{int} = [K|0] = \begin{pmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Intrinsische Matrix:

Formel 9: Intrinsische Matrix (Hartley und Zisserman, 2015)

Daraus ergibt sich, dass die M_{int} die Translation einer homogenisierten Representation des Punktes \tilde{u} im Kamerakoordinatensystem in 3D, auf das zweidimensionale Bildkoordinatensystem ermöglicht (Hartley und Zisserman, 2015, Nayar, 2021). Dies kann als $\tilde{u} = [K|0]\tilde{X}_c = M_{int}\tilde{X}_c$ vereinfacht ausgedrückt werden.

Abbildung 16 zeigt Position der Kamera in Relation zum Weltkoordinatensystem. Diese wird mit dem Translationsvektor C_w , sowie durch die Rotationsmatrix R ergänzt. Zur Ermittlung der Relation werden die externen Parameter verwendet. Die externen Parameter beschreiben die Rotation und Translation des Kamerakoordinatensystems in Relation zum Weltkoordinatensystem (Hartley und Zisserman, 2015, Nayar, 2021). Hierbei ist $t \in \mathbb{R}^3$ und \mathbb{R}^{3x3} eine orthogonale Rotationsmatrix (Hartley und Zisserman, 2015, Nayar, 2021). Dabei gilt $R^T R = I$, wobei R und t die extrinsischen Parameter bereits enthalten (Hartley und Zisserman, 2015, Nayar, 2021). Demnach kann die extrinsische Matrix M_{ext} (Formel 10 und 11) wie folgt ausgedrückt werden:

$$M_{ext} = \begin{pmatrix} R_{3x3} & t \\ 0_{1x3} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Formel 10: Extrinsische Matrix (Hartley und Zisserman, 2015)

$$\tilde{X}_c = M_{ext}\tilde{X}_w$$

Formel 11: Formel 10 vereinfacht (Hartley und Zisserman, 2015)

Werden M_{int} und M_{ext} nun zusammengefasst (Formel 12), lässt sich die vollständige Projektionsmatrix formen (Formel 13) (Nayar, 2021, Hartley und Zisserman, 2015).

$$\tilde{u} = M_{int}M_{ext}\tilde{X}_w = P\tilde{X}_w$$

Formel 12: Zusammenfassung von intrinsischer und extrinsischer Matrix (Hartley und Zisserman, 2015)

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

Formel 13: Projektionsmatrix (Hartley und Zisserman, 2015)

2.6 Visuelles Tracking

In Kapitel 2.1.5 wird AR vorgestellt. Zur Schätzung der Kamerapose, respektive des observierten Szenenabschnitts, ist das Tracking einer Representation der physischen Umgebung, insbesondere des HMDs notwendig. Hierbei wird zwischen zwei Herangehensweisen unterschieden. Das Tracking mittels synthetischen Markern, welche vorab in der Szene platziert werden müssen und das Tracking von natürlichen Markern. Beide Ansätze werden in Kapitel 2.6.1, insbesondere 2.6.2 näher betrachtet.

2.6.1 Tracking mit synthetischen Markern

Synthetische Marker werden in unterschiedlichen Ausführungen zum Tracken einer weltlichen Korrespondenz verwendet (Dörner et al., 2013). Abbildung 18 zeigt einige Beispiele künstlich erzeugter Marker. Dies weisen i.d.R. harte Kanten auf, welche sich von der Umgebung abheben sollen und sich demensprechend leicht erfassen bzw. verfolgen lassen (Dörner et al., 2013). Durch optische Sensoren können die entsprechend platzierten Marker erfasst werden (Dörner et al., 2013). Dadurch können die relativen Transformationen zwischen dem Objekt- und dem Kamera- bzw. Weltkoordinatensystem ermittelt werden (Dörner et al., 2013). So lässt sich die Pose der Kamera, aus der die Aufnahme des Frames erfolgte, rekonstruieren (Dörner et al., 2013). Diese Arbeit fokussiert das optische Tracking natürlicher Merkmale, weshalb synthetische Marker nicht weiter berücksichtigt werden. In Kapitel 2.6.2 erfolgt daher Vorstellung des marker-less Trackings.



Abbildung 18: Marker für visuelles Tracking (Dörner et al., 2013)

2.6.2 Marker-less Tracking

Das Tracking natürlicher Marker kann durch unterschiedliche Ansätze realisiert werden (Dörner et al., 2013). In dieser Arbeit wird unter markerless-Tracking, die Extraktion von Merkmalen aus der physischen Umgebung durch Kameras verstanden. Hierbei werden Bildmerkmale in Form von Kanten identifiziert und daraus die Pose rekonstruiert (Dörner et al., 2013). Werden im folgenden Frame bereits bekannte Merkmale identifiziert, kann aus den korrespondierenden Merkmalen beider Frames die Trajektorie der Kamera zwischen beiden Frames rekonstruiert werden (Dörner et al., 2013). Die Merkmalsextraktion wird in Kapitel 5 im Zuge der Vorstellung des identifizierten SLAM-Verfahrens näher betrachtet.

2.7 Verwendete Software

Im nachfolgenden Teilkapitel werden die zur Erstellung des Prototypen genutzten Softwarelösungen vorgestellt. Für die Bildanalyse und Bearbeitung wird die Open Computer Vision Library eingesetzt. Als Middleware und Kommunikationsschnittstelle zwischen SLAM und der Unity Engine, wird das Robot Operating System verwendet. Abschließend wird die Unity Engine als Authoring-Tool zur Entwicklung von MR-Applikationen vorgestellt.

2.7.1 Open Computer Vision Library

Computer Vision bezeichnet das Forschungsgebiet des Maschinellen Sehens, insbesondere der Nutzbarmachung optischer Daten für Computerprogramme (Peters, 2017). Die Open Computer Vision Library (OpenCV), wird nativ in C++ entwickelt, kann jedoch auch in Kombination mit anderen Programmiersprachen genutzt werden. Aktuell werden zudem JavaScript, Java und Python unterstützt (Kaehler und Bradski, 2015).

Ursprünglich wurde OpenCV von Gary Bradski, während seiner Zeit bei der Intel Corporation, entwickelt. Bradskis Absicht war es leichteren Zugang zu Technologien der Computer Vision, respektive Künstlichen Intelligenz zu ermöglichen. OpenCV folgt dem Open-Source-Gedanken und wird gemeinschaftlich weiterentwickelt (Kaehler und Bradski, 2015).

Zusammenfassend stellt OpenCV eine Computer-Vision-Library dar, welche die wesentlichen Anforderungen für eine Entwicklung im Bereich des maschinellen Sehens bietet und zudem interessante Werkzeuge zur Lösung vSLAM-basierter Probleme bietet (Kaehler und Bradski, 2015). So sind Methoden zur Merkmalsextraktion, Kamerakalibrierung und Bildmanipulation bereits implementiert (Kaehler und Bradski, 2015).

Hinsichtlich der Flexibilität bietet die Bibliothek eine nahezu freien Wahl der Programmiersprache an. Demnach eignet sich OpenCV, nicht nur bezogen auf Flexibilität und Kompatibilität, sondern auch aufgrund der stetigen Weiterentwicklung für die Entwicklung eines Prototypen (Kaehler und Bradski, 2015).

2.7.2 Robot Operating System

Das Robot Operating System (ROS) wird zur Entwicklung komplexer Systeme – primär im Kontext der Robotik – verwendet (Koubaa, 2016). Konkret können komplexe Bewegungsabläufe und das Verhalten autonomer Systeme realisiert werden (Koubaa, 2016). Dazu zählen die in dieser Arbeit fokussierten SLAM-Verfahren und die maschinelle Wahrnehmung. Mittels ROS können demnach unterschiedliche Szenarien realisiert werden (Koubaa, 2016). Zudem bietet ROS die Möglichkeit als Middleware zu fungieren und dadurch die Kommunikation zwischen dem Betriebssystem und weiteren Applikationen zu realisieren (Koubaa, 2016). Im Fokus der Zielstellung dieser Arbeit, wird ROS als Middleware genutzt, um die Kommunikation zwischen dem verwendeten SLAM-Algorithmus und der Unity Engine zu realisieren (Hussein, Garcia und Olaverri-Monreal, 2018).

2.7.3 Unity Engine

Die Unity Engine ist ein professionelles Entwicklungstool, primär für die Entwicklung von digitalen Spielen (Hocking, 2015). Als Authoring-Tool bietet die Unity-Engine, gegenüber

der manuellen Programmierung von Spielen, einen geführten Development-Workflow an (Hocking, 2015). In der Forschung haben sich in der Simulationstechnik neue Anwendungsgebiete für die Unity Engine ergeben (Miller, Gibson und Navarro, 2021). Demensprechend bietet die Unity Engine in Verbindung mit dem Unity-Robotics-Hub eine Schnittstelle zur Robotik an und ist dadurch in der Lage erzeugte Daten mit einer ROS-Instanz auszutauschen (Miller, Gibson und Navarro, 2021).

2.8 Zusammenfassung

Kapitel 2 bietet einen Einblick in die theoretischen Grundlagen der zugrundeliegenden Arbeit. Angefangen mit dem Reality-Virtuality-Continuum nach Milgram et al. werden die Teilmengen des RV-Continuum näher erläutert (Milgram und Kishino, 1994). AR beschreibt eine Realitätsform, basierend auf der physischen Realität (Dörner et al., 2013). Hierbei kann der Nutzer die reale Welt, angereichert mit computergenerierten Objekten wahrnehmen (Dörner et al., 2013). Die virtuellen Objekte werden mittels eines HMD in das Sichtfeld des Nutzers projiziert (Dörner et al., 2013).

Spatial Computing beschreibt Methoden zur Interpretation des physischen Raumes für Computerprogramme (Greenwold, 2003). Hierbei müssen entsprechende Sensoren vorhanden sein, welche die Umgebung erfassen können (Greenwold, 2003). Oft werden hierfür Kameras eingesetzt (Dörner et al., 2013). Zur Interpretation der Umgebung und Nutzbarmachung der ermittelten Informationen, wird SLAM als Werkzeug zur Realisierung von Spatial Computing vorgestellt (Greenwold, 2003, Çöltekin et al., 2020).

Um aus der Umgebung Information zu extrahieren, werden oft Kameras eingesetzt . Durch das Lochkameramodell und die perspektivische Projektion können dreidimensionale Posen aus zweidimensionalen Bilddaten ermittelt werden. Hierzu müssen Kameras jedoch kalibriert werden, um die intrinsischen und extrinsischen Parameter der Kamera zu ermitteln.

Zur Ermittlung der Kamerapose, müssen entsprechende Korrespondenzen in den erfassten Frames in Form von Merkmalen identifizierbar sein. Hierfür muss entweder eine physische Markierung für die lagerichtige Projektion vorhanden sein oder – um markerless-tracking zu nutzen – entsprechende Algorithmen eingesetzt werden. Letztere versuchen natürliche Bildmerkmale zu identifizieren und daraus die Pose zu ermitteln (Dörner et al.,

Grundlagen

2013). Abschließend werden in Kapitel 2 die verwendeten Softwarelösungen und Bibliotheken vorgestellt, welche im Kontext SLAM, MR und der Bildverarbeitung eingesetzt werden.

Das nachfolgende Kapitel 3 stellt die von Microsoft vorgestellte Interpretation der MR vor und präsentiert den technischen Aufbau der Microsoft HoloLens. Beides dient als Referenz für das Realisierungsziel des Prototypen und stellt dementsprechend auch den aktuellen Stand der Technik dar. Anschließend wird in Kapitel 3 ein Einblick in den Stand der Technik visueller SLAM-Verfahren gegeben und Lösungsstrategien für das SLAM-Problem in Verbindung mit visueller Sensorik vorgestellt.

3 Stand der Technik

In Kapitel 2 werden die essentiellen Grundlagen des RV-Continuum nach Milgram vorgestellt. Dies dient dem Verständnis der einzelnen Bestanteile künstlicher Realitäten. Dabei zeigen sich die Gemeinsamkeiten zwischen AR und MR. Diese Gemeinsamkeiten ermöglichen eine gemeinsame Betrachtung von AR- und MR-Anwendungen, hinsichtlich der Anforderungen an die Systeme (Milgram und Kishino, 1994).

Spatial Computing wird als Überbegriff für Technologien eingeführt, welche die Interpretation der Umwelt ermöglichen (Greenwold, 2003). Dies ermöglicht die Extraktion und Verarbeitung von realen Umgebungsinformationen, welche für die Lokalisierung notwendig sind (Chen et al., 2018b).

Anschließend werden HMDs als geeignete Ein- und Ausgabegeräte für MR-Anwendungen vorgestellt (Sutherland, 1968). Zur Lokalisierung und der Ermittlung der Pose des Nutzers, respektive HMDs in der Realität, ist es notwendig, Merkmale aus der Umgebung zu extrahieren (Dörner et al., 2013). Dies bedingt den Einsatz von geeigneter Sensorik (Chen et al., 2018b). Mixed Reality, sowie Spatial Computing setzen die Extraktion von Umgebungsinformation aus der Realität voraus. Selbst unter Verwendung physischer Marker müssen visuelle Daten verarbeitbar sein (Dörner et al., 2013). Der Fokus der Arbeit wird hierbei auf die Microsoft HoloLens gelegt und die wesentlichen technologischen Bestandteile, sowie die Funktionsweise im folgenden Kapitel näher beleuchtet.

3.1 Interpretation der Mixed Reality

In Kapitel 2.1 werden die technischen Aspekte der Teilbereiche der MR erläutert. Generell können die Anforderungen an AR bzw. VR, auf MR angewendet werden (Dörner et al., 2013). Das RV-Continuum lässt unterschiedliche Verhältnisse zwischen der physischen und virtuellen Realität zu, bietet jedoch keine Feinabstufung zwischen den einzelnen Definitionen der AR, AV, sowie VR (Milgram und Kishino, 1994).

Mixed Reality stellt keine atomare Definition dar, sondern ein Spektrum (Milgram et al., 1995). Augmented Virtuality liegt vor, sofern der Anteil der virtuellen Objekte einer Szene überwiegen (Milgram und Kishino, 1994). Im Kontrast dazu steht Augmented Reality, wobei

virtuelle Elemente in eine physische Basis der Realität projiziert werden (Dörner et al., 2013). AR liegt vor, sobald virtuelle Inhalte in die Realität projiziert werden (Dörner et al., 2013). MR stellt sich als komplexer dar und bietet deutlich mehr Interaktionsmöglichkeiten zwischen Virtualität und Realität (Bray, 2021). Konkret ist es bei MR-Anwendungen möglich, eine immersive Interaktion mit virtuellen Elementen zu realisieren, wobei der Nutzer nicht durch die virtuelle Realität eingeschränkt wird (Dörner et al., 2013, Costanza, Kunz und Fjeld, 2009).

Da AR ein Teilbereich der MR darstellt werden beide Begriffe oft synonymisch verwendet (Milgram und Kishino, 1994, Dörner et al., 2013). In dieser Arbeit werden daher reine AR-Anwendungen mit MR-Anwendungen gleichgesetzt und gemeinsam betrachtet, sofern die AR-Systeme in der Lage sind, physische Objekte zu registrieren und virtuelle Objekte in den räumlichen Kontext der Szene zu integrieren. Aktuell existieren im Bereich der Mixed bzw. gemischten Realitäten je nach Ausprägung und Intensität realer, sowie synthetischer Inhalte, unterschiedliche Interpretationen (Bray, 2021). Dies ist durch die Natur der MR als Spektrum begründet (Milgram und Kishino, 1994). Daher wird für diese Arbeit der Fokus auf Microsofts Windows Mixed Reality gelegt.

Windows Mixed Reality (Windows-MR) ist ein aus dem Hause Microsoft stammendes Ökosystem, welches als Plattform für unterschiedliche AR- und VR-Anwendungen dient. Ursprünglich wurde Windows-MR 2015 als „*Windows Holographic*“ vorgestellt und sollte ausschließlich mit dem hauseigenen HMD der HoloLens kompatibel sein. Seit 2017 wurde die Kompatibilität stetig auf weitere HMDs und Geräte erweitert und Windows-MR als integraler Bestandteil von Windows 10 fest in das Betriebssystem integriert (Dehghani, Lee und Mashatan, 2020, Wikipedia, 2021c).

Microsofts Interpretation der MR hält sich an den wissenschaftlichen Standard und gründet auf den Erkenntnissen und Definitionen Milgrams, vermischt die physische mit der virtuellen Realität jedoch weiter (Bray, 2021, Dehghani, Lee und Mashatan, 2020). Demnach werden Umgebungseinflüsse und die menschliche Wahrnehmung stärker mit einbezogen. Heute hat sich die Anwendung von Mixed Reality von ursprünglich reinen displaybasierten Applikationen auf die Verarbeitung von Umgebungsdaten, räumlichen Klang und die Positionsermittlung von virtuellen sowie physischen Objekten weiterentwickelt. Bei

Windows-MR spielt daher der Umgebungseinfluss, sowie die Umgebungswahrnehmung eine zentralisierte Rolle (Dehghani, Lee und Mashatan, 2020, Bray, 2021, Zeller, 2021).

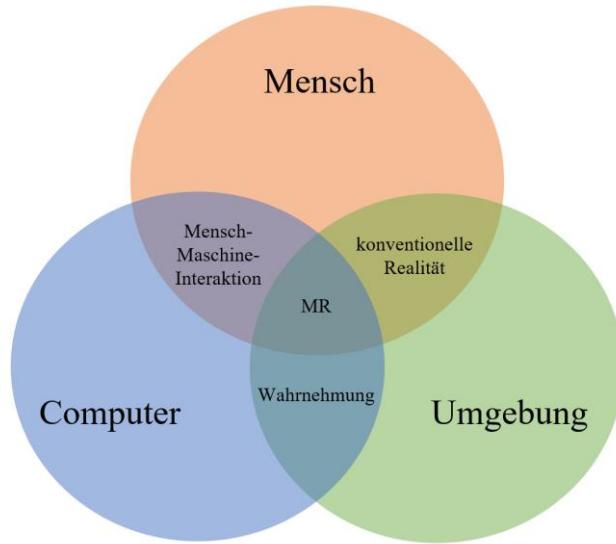


Abbildung 19: Interaktion zwischen Mensch, Computern und der Umgebung in Anlehnung an (Bray, 2021)

Abbildung 19 zeigt das Interaktionsschema zwischen Menschen, Computern und der Umgebung hinsichtlich Windows-MR. Die enge Verbindung zwischen der menschlichen Wahrnehmung in Kombination mit einer physischen Interaktion als Eingabe, ermöglicht den Einbezug der Umgebungsgeometrie durch die in das HMD verbaute Sensorik. Dadurch wird die Umgebung für den Computer interpretierbar. Die Erfassung und Interpretation der Raumgeometrie kann folglich in die Projektion und damit Erweiterung der physischen Realität involviert werden (Bray, 2021, Zeller, 2021).

Die enge Verbindung zwischen dem Menschen als Operator des HMD, erlaubt dabei die sensorische Interpretation und den Transfer der eigenen realen Pose an den Computer. Dadurch können Bewegungen über das HMD und die Sensorik an den Computer übermittelt werden. Die erfassten physischen Bewegungen können dadurch in die digitale Welt übersetzt werden. Der Mensch kann weiterhin die konventionelle Umgebung wahrnehmen und diese durch computergenerierte Inhalte erweitern, sowie beeinflussen (Bray, 2021, Zeller, 2021).

Windows-MR verbindet die menschliche, mit der maschinellen Wahrnehmung bzw. kombiniert die physische und virtuelle Realität, um laut MS eine echte Mixed-Reality-Erfahrung zu ermöglichen (Bray, 2021).

Zusammenfassend betrachtet, hebt sich Microsofts Interpretation durch den hohen Grad der Interoperabilität zwischen physischer und synthetischer Realität von der in Kapitel 2.1 definierten Grundform der MR nach Milgram ab (Bray, 2021, Milgram und Kishino, 1994). Demnach ist die Interaktion zwischen dem Operator und virtuellen Objekten die wesentliche Änderung gegenüber der klassischen MR.

Grundvoraussetzung für eine gemischte Realität und demzufolge Windows-MR-Erfahrung ist – laut MS – der Einbezug von Umgebungseinflüssen als Eingabewerte für die computergesteuerte Informationsverarbeitung (Bray, 2021). Dies erfordert ein sehr präzises Erfassungs- und Lokalisierungssystem, welches nicht nur auf die aktuelle Szene beschränkt konsistent arbeitet, sondern eine global persistente Karte generiert (Zeller, 2021). Virtuelle und physische Objekte können daher auch außerhalb des Field of Views (FOV) des Nutzers existieren und für spätere Observationen oder Interaktionen ihre physisch-konsistente Pose behalten (Liu et al., 2020). Die persistierten Lokationsdaten werden genutzt, um die Trajektorie des Nutzers im Raum nachzuverfolgen und somit die globale Position in einer sich entwickelnden Karte zu optimieren (Zeller, 2021).

Daher muss – um Windows-MR realisieren zu können – das SLAM-Problem gelöst werden, sodass die Posen aller physischen und virtuellen Objekte, insbesondere die Pose des Nutzers inklusive HMD in der Realität bekannt sind (Hübner et al., 2020, Khoshelham, Tran und Acharya, 2019, Evans et al., 2017, Chen et al., 2018a). Die Umgebungserfassung wird auch Spatial Computing (Greenwold, 2003) genannt bzw. die Kartierung auch als Spatial Mapping bezeichnet (Zeller, 2021).

3.2 Microsoft HoloLens

Die HoloLens ist in der Lage virtuelle Objekte in die Realität zu projizieren (Dehghani, Lee und Mashatan, 2020). Dies wird durch die Verwendung unterschiedlicher Sensoren erreicht (Evans et al., 2017). Durch RGB-Kameras in Verbindung mit Tiefenkameras, kann die Umgebung interpretiert werden (Hübner et al., 2020). Intern werden zur Datenverarbeitung ein Intel 32-bit Prozessor und für das Rendering der digitalen Komponenten eine Holographic Processing Unit (HPU) verwendet (Farasin et al., 2020). Gestützt durch das Betriebssystem Windows und das Windows-MR-Ökosystem, ist die HoloLens in der Lage Spatial Mapping zu praktizieren und somit SLAM zu lösen (Liu et al., 2018b, Khoshelham,

Tran und Acharya, 2019, Weinmann et al., 2020). Dabei ist die Rechenleistung jedoch limitiert. Demnach sind ressourcenintensive Algorithmen u.U. nicht in Echtzeit umsetzbar (Farasin et al., 2020).

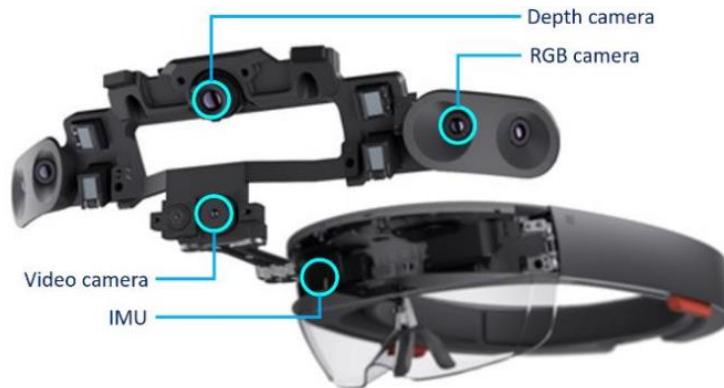


Abbildung 20: Kamerasyteme der HoloLens (Khoshelham, Tran und Acharya, 2019)

Abbildung 20 zeigt die in die HoloLens integrierten Kamerasyteme (Hübner et al., 2020). Darunter finden sich jeweils zwei RGB-Kameras an beiden Seiten (Khoshelham, Tran und Acharya, 2019). Diese werden logisch als zwei Stereopaare angesprochen und analysieren die Umgebung (Liu et al., 2018b). Die Sensorik kann im Kursstecken-Modus Entfernen zwischen 0 m und 0.8 m und im Langstrecken-Modus von 0.8 m bis 3.5 m messen (Hübner et al., 2020).

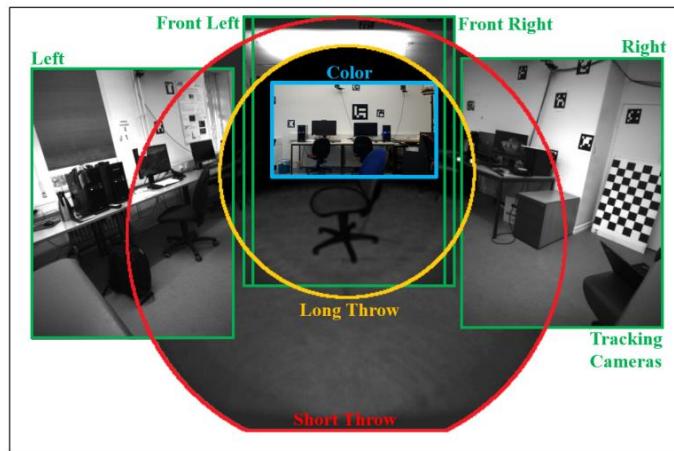


Abbildung 21: Übersicht der Durch die HoloLens generierten Frames (Hübner et al., 2020)

Abbildung 21 zeigt die durch die optischen Sensoren erzeugten Frames (Hübner et al., 2020). Die in grün gekennzeichneten Frames werden durch die Stereopaare erzeugt und zum

Verständnis der physischen Umgebung eingesetzt. Das blau umrandet Color-Frame wird nicht in den SLAM-Prozess involviert und steht für Applikationen zur Verfügung. Long Throw (gelb), zeigt hierbei das Frame der Tiefenkamera im Langstrecken-Modus. Im Gegensatz dazu zeigt Short Throw (rot), den Kurzstrecken-modus der zur Wahrnehmung naheliegender Objekte wie bspw. der Hände verwendet wird (Hübner et al., 2020).

Liu et al. analysieren die HoloLens und stellen fest, dass die Datenverarbeitung und demnach die operativen Abläufe in fünf Hauptkomponenten gegliedert sind (Liu et al., 2018b). Diese bestehen aus dem Head-Tracking, der Umgebungsrekonstruktion, der Datenverarbeitung für die virtuellen Komponenten und der Wahrnehmung bzw. Eingaben durch den Benutzer (Liu et al., 2018b).

Das Head Tracking wird hierbei durch eine Internal Measurement Unit realisiert (Liu et al., 2018b). In Verbindung mit dem ICP-Algorithmus (iterative closest point) (Liu et al., 2018b) wird die Pose konstruiert und optimiert. Durch die umgebungswahrnehmenden Kameras, in Verbindung mit den RGB-D-Kameras, kann die dreidimensionale Struktur der Umgebung rekonstruiert werden (Weinmann et al., 2020). In Kombination mit der Benutzereingabe, lassen sich eine Interaktion zwischen virtuellen und physischen Objekten realisieren (Dehghani, Lee und Mashatan, 2020). Der Nutzer nimmt dadurch die Realität durch die HoloLens – angereichert mit durch die Umgebung beeinflussen interaktiven virtuellen Objekten – war (Bray, 2021, Zeller, 2021). Microsofts Interpretation der MR, wird durch die HoloLens unterstützt (Bray, 2021). Diese ist in der Lage alle Voraussetzungen für Spatial Computing (Kapitel 2.2), respektive die Anforderungen an MR-Systeme (Kapitel 2.1) zu erfüllen.

Die HoloLens löst zwar das SLAM-Problem und realisiert Spatial Computing (Khoshelham, Tran und Acharya, 2019, Farasin et al., 2020, Weinmann et al., 2020), allerdings nicht in Perfektion, was Liu et al. demonstrieren (Liu et al., 2018b). Demnach liefere die IMU der HoloLens die Kopfposition nur zuverlässig, im Zuge langsamer Bewegungen (Liu et al., 2018b). Zudem können planare Oberflächen nur in gut belichteten Szenen erkannt werden. Ebenso wird die Verankerung von AR-Objekten nur in Distanzen zwischen 1.5 m und 2.5 m zuverlässig realisiert (Liu et al., 2018b). Dies impliziert einen weiteren Entwicklungsbedarf der HoloLens, hinsichtlich der erreichbaren Präzision der Hardware, insbesondere der implementierten Algorithmen (Liu et al., 2018b).

Nichtsdestotrotz realisiert die HoloLens in Kombination mit dem MR-Ökosystem Windows Mixed Reality MR-Anwendungen und erfüllt die Voraussetzungen für Spatial Computing (Khoshelham, Tran und Acharya, 2019).

In Kapitel 2.5.3 wird die Kamerakalibrierung als vorbereitenden Schritt zur Anwendung moderner vSLAM-Algorithmen vorgestellt (Ozog und Eustice, 2013). Der im Zuge dieser Arbeit erarbeitete Stand der Technik bzgl. vSLAM-Algorithmen wird in Kapitel 3.4 herausgearbeitet. Basierend auf dem Stand der Technik wird in Kapitel 4 ein Konzept für einen Prototypen erstellt.

3.3 Visual SLAM

Wie in Kapitel 2.4 bereits dargestellt, werden zur Lösung des SLAM-Problems Sensoren zur Informationsextraktion aus der Umgebung verwendet. Wird das SLAM-Problem ausschließlich unter der Verwendung optischer Sensorik gelöst, werden die Verfahren als visual SLAM-Verfahren (vSLAM) klassifiziert (Wang, Wu und Zhang, 2019). Nachfolgend wird vSLAM als eine Lösungsstrategie für SLAM unter Verwendung optischer Sensoren vorgestellt. Zunächst wird eine Abgrenzung zwischen visual und visual-inertial SLAM vorgenommen und anschließend die Taxonomie moderner vSLAM-Systeme vorgestellt. Darauffolgend wird der Prozess zur Erstellung der Umgebungskarte durch den vSLAM-Algorithmus dargelegt und dadurch vSLAM von der reinen visuellen Odometrie abgegrenzt.

3.3.1 Abgrenzung vSLAM von Visual inertial SLAM

Neben vSLAM werden in speziellen Szenarien, z.B. im Kontext Augmented Reality für Smartphones, sog. visual inertial SLAM-Systeme (viSLAM) eingesetzt (Jinyu et al., 2019). Diese kombinieren die Bildinformationen, mit den erhobenen Daten einer Internal Measurement Unit (IMU) (Jinyu et al., 2019). Eine IMU stellt zunächst einen Sensorträger dar, welcher mit unterschiedlichen Sensoren bestückt werden kann. Interessante Sensoren im Kontext vSLAM stellen u.a. Beschleunigungssensoren und Gyroskope dar. Die durch die IMU bereitgestellten Daten, ermöglichen die Ermittlung der Pose des genutzten Endgerätes (Crouse et al., 2015).

Mittels eines SLAM-Algorithmus, können die aus den Bildern gewonnenen Informationen, mit den Daten der IMU kombiniert werden (Ruser und Puente León, 2007).

Anschließend können die Daten durch Filter- und Optimierungsschritte präzisiert werden. Zuzüglich dessen, liefern IMUs auch in texturarmen Szenen weiterhin konstante Daten (Jinyu et al., 2019, Ahmad et al., 2013).

Reine vSLAM Methoden lösen das SLAM Problem unter der Verwendung optischer Sensorik (Huletski, Kartashov und Krinkin, 2015). Hierbei werden – bezogen auf die klassischen vSLAM-Varianten – lediglich Kameras, respektive Tiefenkameras eingesetzt. Daher werden dieser Arbeit vSLAM-Systeme als solche bezeichnet, sofern maximal eine Kombination aus Kameras und Tiefenkameras verwendet wird. Sobald eine IMU eingesetzt wird, werden die Systeme als viSLAM bezeichnet (Jinyu et al., 2019). Das nachfolgende Teilkapitel stellt die Taxonomie der vSLAM-Methoden dar.

3.3.2 Taxonomie moderner vSLAM-Methoden

Visual SLAM-Systeme werden als online-Systeme beschrieben. Online bedeutet, dass die Datenextraktion, sowie Verarbeitung während der Programmlaufzeit – möglichst in Echtzeit – erfolgt (Thrun, Burgard und Fox, 2005). Diese Systeme müssen aufgrund des u.U. massiven Datenflusses (Stachniss, Leonard und Thrun, 2016) und bauartbedingten begrenzten Systemressourcen im Hinblick auf Effizienz konzipiert werden (Klein und Murray, 2007). Oft müssen Kompromisse zwischen der Performanz und der Präzision bzw. der Robustheit der Algorithmen geschlossen werden (Clipp et al., 2010). Die Robustheit gilt im Kontext SLAM als Synonym für die Beschreibung, wie gut ein Algorithmus bspw. mit Störeinflüssen aus der Umgebung oder dynamischen Lichtverhältnissen und sich bewegenden Objekten in der Szene umgeht (Folkesson und Christensen, 2004).

Zur Lösung des SLAM-Problems für mobile Systeme – wie die HoloLens – werden hauptsächlich Kameras eingesetzt (Khoshelham, Tran und Acharya, 2019). Die technische Entwicklung der vergangenen Dekade ermöglichen es, energiesparende, kompakte, sowie rauscharme, sowie kosteneffiziente Geräte aus dem Handel zu beziehen. Kamerasysteme bieten reichhaltige Bildinformationen, welche eine robuste Lokalisierung ermöglichen können (Sualeh und Kim, 2019).

Die Lösungsstrategien für vSLAM lassen sich weiter anhand der eingesetzten Sensortypen und Konfigurationen einstufen. Handelsübliche Kameras bzw. Webcams, werden fortan als RGB-Kameras referenziert und Time of Flight (ToF) oder auch

Tiefenkameras unter der Bezeichnung RGB-Depth (RGB-D) zusammengefasst. RGB-D-Kameras stellen eine separate Kategorie der optischen Sensoren dar (Huang Baichuan, Zhao und Liu, 2019).

RGB-Kameras, darunter auch Webcams, zählen zu den passiven Sensoren und benötigen zur Informationsextraktion aus der Umgebung keine zusätzliche Versorgungsspannung bzw. Hilfsenergie. Hierbei absorbieren diese natürliches Licht, um Informationen aus der Szene zu extrahieren (Discant et al., 2007). Wohingegen aktive Sensoren wie bspw. Laserscanner (LiDAR) oder RGB-D Kameras zur Aktivierung das Anlegen einer Versorgungsspannung benötigen und somit erst anschließend ein Ausgangssignal generieren (Discant et al., 2007). Konkret wird die induzierte Energie in die Umgebung ausgestrahlt und die darauffolgende Reaktion gemessen. Ein weiteres Beispiel hierfür wären ToF-Kameras, welche zunächst Lichtbündel abstrahlen und die Zeit bis zum Eintreffen der Lichtreflektion messen. Mittels der Lichtgeschwindigkeitskonstante und der gemessenen Zeit bis zum Eintreffen der Reflektion im Sensor, kann der Abstand zwischen Sensor und dem reflektierenden Objekt direkt errechnet werden (Yang et al., 2010). Letztere Systeme sind sehr lichtanfällig und daher für dynamische Umgebungen mit wechselnden Lichtverhältnissen weniger gut geeignet (Schuon et al., 2008).

Das vSLAM-Problem lässt sich durch die Verwendung unterschiedlicher Sensorkonfigurationen lösen. Wird nur ein Kamerasensor genutzt, handelt es sich um ein monokulares System (Huang Baichuan, Zhao und Liu, 2019). Monokulare Systeme können durch eine IMU unterstützt arbeiten (Piao und Kim, 2017). Durch viele schnell aufeinanderfolgende Aufnahmen aus unterschiedlichen Perspektiven (Structure from Motion), kann eine dreidimensionale Rekonstruktion erfolgen (Davison et al., 2007). Monokulare Systeme weisen – bezogen auf die 3D-Rekonstruktion – wesentliche Nachteile auf. Zum einen tritt sog. Scale Ambiguity oder auch Skalenmehrdeutigkeit auf. Hierbei kann die Länge der Translationsbewegungen nicht allein aus der Merkmalskorrespondenz ermittelt werden (Kitt et al., 2011, Taketomi, Uchiyama und Ikeda, 2017). Dies führt zudem zum Scale Drift, was im Wesentlichen eine Deformierung der Trajektorie zur Folge hat (Strasdat, M. M. Montiel und Davison, 2011). Um dies auszugleichen werden oft Ansätze aus der Künstlichen Intelligenz oder Stereosysteme verwendet (Li, Wang und Gu, 2021, Mur-Artal und Tardos, 2017).

Angelehnt an die menschliche visuelle Wahrnehmung, werden oft zwei Kameras zeitgleich eingesetzt. Diese Verfahren werden als binokulare Systeme bezeichnet (Banks et al., 2012). Hierbei werden die Kameras in der Horizontalen oder Vertikalen versetzt, fest auf eine Platine montiert. Binokulare Systeme bieten Vorteile gegenüber monokularen Verfahren, da zu jedem Verarbeitungsschritt direkt zwei Perspektiven aus der Szene erstellt werden (Krombach et al., 2018). Dadurch ist es möglich die Disparität zur Tiefenbestimmung zu nutzen. Die Tiefe eines Punktes der Szene gibt den Abstand zwischen der Kamerapose und dem Punkt im Weltkoordinatensystem wieder (Schreer, 2005).

Weiter können vSLAM-Ansätze in zwei Herangehensweisen unterteilt werden. Zum einen werden Filter-basierte Verfahren angewendet, um Sensorrauschen aus den Sensordaten zu aus den Daten herauszufiltern (Smith und Cheeseman, 1986). Hierzu zählen auch die in Kapitel 2.4 vorgestellten Ansätze von Smith und Cheeseman. Die bekanntesten Filter-basierten Ansätze sind EKF SLAM (Smith und Cheeseman, 1986) und die Particle Method (Montemerlo et al., 2003). Optimierungs-basierte Methoden haben heute Filter-basierte Ansätze übertrffen (Sualeh und Kim, 2019).

Des weiteren werden VSLAM-Algorithmen in direkte und indirekte Verfahren unterteilt (Chen et al., 2018b). Direkte Verfahren sind in der Lage Tiefeninformationen direkt – zur Laufzeit des Programms – aus der observierten Szene zu extrahieren. Ein Beispiel hierfür wären ToF-Kameras und Laserscanner. Aktive Sensoren können so eigenständig und ressourcensparend den Abstand zwischen dem Sensor und dem Ursprung der Projektion eines Bildpixels im Weltkoordinatensystem ermitteln. Die bereits angesprochenen Nachteile – wie die geringe Robustheit gegenüber natürlicher, sowie künstlicher Lichteinstrahlung – können je nach Systementwurf durch die Vorteile der ressourcensparenden Informationsextraktion ausgeglichen werden (Huang Baichuan, Zhao und Liu, 2019).

Microsofts HoloLens nutzt bspw. Tiefenkameras zur Ermittlung der Handposition, um die Interaktion zwischen dem Operator und nahen virtuellen Objekten zu realisieren (Evans et al., 2017). Die direkte Extraktion der Bildinformation erfolgt pro Pixel. Demnach wird bspw. im allgemeinen Fall von RGB-D-Sensoren ein Infrarotstrahl für jede Projektion eines Pixels in die Szene abgestrahlt. Dadurch kann für jede Abbildung ein Tiefenwert bestimmt werden. Dies ermöglicht eine dichte Rekonstruktion der gesamten Szene (Whelan et al., 2015). Je nach Anwendungsfall ist dies allerdings nicht immer notwendig. Aufgrund der

hohen Anschaffungskosten und komplexer Kalibrierung, werden nachfolgende keine direkten Verfahren, sowie RGB-D Sensoren berücksichtigt.

Im Kontrast dazu stehen die indirekten Verfahren, die lediglich markante Merkmale in der Szene – abhängig vom Grad der Texturierung der Umgebung – identifizieren (Chen et al., 2018b, Scaramuzza und Fraundorfer, 2011). Hierbei kommt es darauf an, welchen Zweck die Anwendung verfolgt und wie gut texturiert die Umgebung ist. Sind in der Szene ausreichend valide Merkmale identifizierbar, wird eine sparse reconstruction möglich (Chen et al., 2018b, Scaramuzza und Fraundorfer, 2011). Sparse oder spärlich, bezieht sich hierbei auf die Anzahl der Merkmalskorrespondenzen (Chen et al., 2018b, Scaramuzza und Fraundorfer, 2011). Für Lokalisierungsverfahren ist eine spärliche Rekonstruktion ausreichend und zudem performant in Echtzeit umsetzbar (Davison et al., 2007). Demensprechend kann nur die physische Position des identifizierten Merkmals in eine Karte übertragen werden und nicht jeder Pixel der Abbildung aus der Szene (Fraundorfer und Scaramuzza, 2012).

Indirekte Verfahren nutzen RGB-Kameras zur Extraktion der Bildinformationen aus der Szene (Scaramuzza und Fraundorfer, 2011). Die erhobenen Daten werden durch entsprechende Merkmalsdetektoren analysiert und können in nachfolgenden Frames anhand der Deskriptoren wieder identifiziert werden (Scaramuzza und Fraundorfer, 2011). Dies ermöglicht das Tracking der Merkmale und eine Rekonstruktion der Kamerapose, relativ zur Observation (Scaramuzza und Fraundorfer, 2011).

3.3.3 Visual-SLAM-Framework

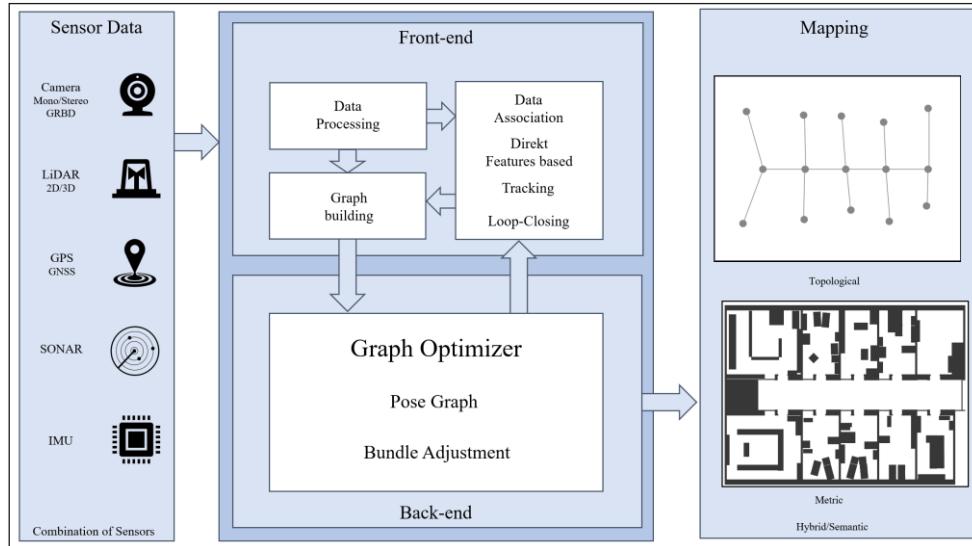


Abbildung 22: Schema vSLAM-Algorithmen in Anlehnung an (Sualeh und Kim, 2019)

Abbildung 22 zeigt den schematischen Ablauf moderner vSLAM-Algorithmen (Sualeh und Kim, 2019). Diese bestehen aus einem Front- und Backend. Das Frontend befasst sich mit der Extraktion und Interpretation der Sensordaten. Hierbei werden oft Ansätze aus der Visuellen Odometrie angewendet, um die sensorischen Messungen mit der lokalen Trajektorie in Verbindung zu bringen. Des Weiteren, praktiziert das Front-end sog. Loop-Closing, welches bekannte Abschnitte der Karte wiedererkennt. Das Backend hingegen, übernimmt die Optimierung der erfassten Daten zur Präzisierung der Lokationsdaten (Sualeh und Kim, 2019).

Die vSLAM-Systeme werden modular aufgebaut und können je nach Anwendungsfall aus unterschiedlichen Modulen bestehen (Sualeh und Kim, 2019). Abbildung 22 zeigt den schematischen Aufbau eines generalisierten vSLAM-Framework. Die Eingabedaten können durch unterschiedliche Sensoren generiert werden. Hierbei kann eine Kapitel 3.3.1 beschriebene IMU eingesetzt werden, welche das vSLAM-System in ein viSLAM-System umwandelt. Dies wird jedoch im weiteren Verlauf der Arbeit nicht weiter berücksichtigt. Ebenso werden keine Sonar-Sensoren, das GPS, sowie LiDAR weiterverfolgt. Demnach reduziert sich die fokussierte Sensorik auf Kameras, welche die Umgebung in Form von Bilddaten erfassen (Sualeh und Kim, 2019).

Das Back-end kann wahlweise graph-based-optimization betreiben oder die Optimierung der Trajektorie und die Aufbereitung der Lokationsdaten für die anschließende Kartierung durch Bundle Adjustment realisiert werden. Die Kartierung kann hierbei als Graph oder semantisch erfolgen (Sualeh und Kim, 2019).

3.3.4 Visuelle Odometrie

Visuelle Odometrie (VO) stellt einen wichtigen Teilbereich von vSLAM dar (Tang und Cao, 2020). VO beschreibt den Prozess der inkrementellen Schätzung der Kamerapose (Scaramuzza und Fraundorfer, 2011). Dazu werden die Langeänderung und Bewegung der identifizierten Bildmerkmale über die Zeit bzw. mehrere Frames hinweg geschätzt (Scaramuzza und Fraundorfer, 2011). Dies bezieht sich – sofern kein vollständiger vSLAM-Algorithmus eingesetzt wird – lediglich auf den lokalen Teil der Karte (Scaramuzza und Fraundorfer, 2011, Fraundorfer und Scaramuzza, 2012). Letzteres bedeutet, dass die Trajektorie der Kamera nicht in einer globalen Karte persistiert wird, sondern nach vordefinierten Zeiträumen verworfen wird (Fraundorfer und Scaramuzza, 2012, Scaramuzza und Fraundorfer, 2011). VO wird daher in Szenarien eingesetzt, in denen eine global konsistente Karte nicht benötigt wird, sondern bspw. VO als Alternative zur wheel-based odometry genutzt werden soll (Fraundorfer und Scaramuzza, 2012).

In der Regel wird eine ausreichende Ausleuchtung der Szene vorausgesetzt, insbesondere wird eine reflektionsarme Umgebung benötigt (Scaramuzza und Fraundorfer, 2011). Weiter funktioniert VO nur in Szenen in denen statische Objekte überwiegen zuverlässig (Scaramuzza und Fraundorfer, 2011). Klassische VO-Methoden – ohne entsprechend trainierte Datasets und KI-Modelle – sind nicht in der Lage volldynamische Szenen zu zuverlässig zu analysieren (Yong-bao et al., 2020, Yang et al., 2020).

Wie unter Kapitel 3.3.2 bereits vorgestellt, sind kamerabasierte Ansätze oftmals mit merkmalsbasierten Verfahren verknüpft (Chen et al., 2018b). Daher wird bei VO angenommen, die Umgebung sei möglichst Texturreich und biete keine repetitiven Muster, um ausreichend robuste Merkmale identifizieren zu können (Scaramuzza und Fraundorfer, 2011). Weiter wird vorausgesetzt, dass aufeinanderfolgende Frames ausreichende Korrespondenzen aufweisen und kein Bruch zwischen der Observation zum Zeitpunkt t und $t+1$ entsteht (Scaramuzza und Fraundorfer, 2011). Daher muss eine entsprechend angepasste

Bildwiederholungsrate realisierbar sein, um Echtzeitfähigkeit herstellen zu können (Scaramuzza und Fraundorfer, 2011).

VO kann – ebenso wie vSLAM – zunächst in direkte und indirekte Methoden differenziert werden (Scaramuzza und Fraundorfer, 2011). Wie in Kapitel 3.3.2 bereits beschrieben, bedingen direkte Verfahren den Einsatz von optischer Sensorik, welche in der Lage ist, Tiefeninformation zur Laufzeit aus der Szene extrahieren zu können (Scaramuzza und Fraundorfer, 2011). Demnach eignen sich RGB-D/ToF-Kameras für diesen Einsatzzweck (Whelan et al., 2015). Diese Arbeit verfolgt jedoch den indirekten Ansatz. Indirekte VO-Algorithmen nutzen merkmalsbasierte Ansätze (Fraundorfer und Scaramuzza, 2012), welche mit handelsüblichen Kameras funktionieren können.

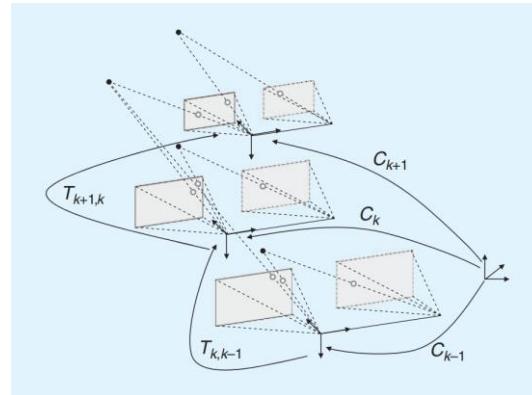


Abbildung 23: Darstellung der Problemstellung der Visuellen Odometrie (Fraundorfer und Scaramuzza, 2012)

Abbildung 23 fasst den Ablauf der Visuellen Odometrie zusammen. Hierbei wird eine Stereokamera verwendet. Im monokularen Fall wird dementsprechend pro Zeiteinheit nur ein Frame erzeugt. Abbildung 23 zeigt so zu jedem diskreten Zeitpunkt k die Trajektorie T bzw. die Pose. Dadurch kann die Trajektorie der Kameraeinheit inkrementell seit der Initialisierung des Systems nachverfolgt werden.

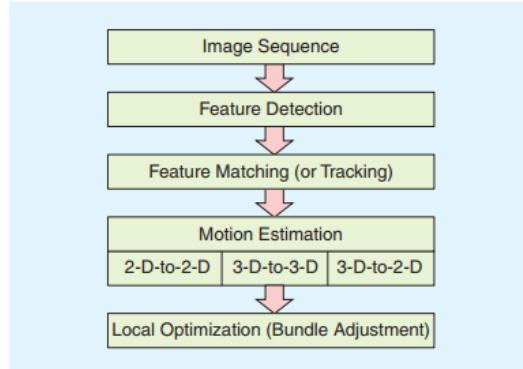


Abbildung 24: Komponenten eines VO-Systems (Scaramuzza und Fraundorfer, 2011)

Abbildung 24 beschreibt den Ablauf eines VO-Algorithmus. Nach jeder erfolgreichen Akquirierung der Frames (Image Sequence) werden Bildmerkmale extrahiert (Feature Detection) (Scaramuzza und Fraundorfer, 2011). Werden Stereopaare verwendet, können direkt korrespondierende Merkmale in beiden Bildern zum gleichen Zeitpunkt k identifiziert werden (Feature Matching). In der monokularen Version können die Frames zum Zeitpunkt $k-1$ zur Korrespondenzanalyse herangezogen werden. Anschließend wird die Trajektorie zwischen dem vorangegangen Frame und dem aktuellen Frame vorgenommen, um die Bewegung der Kamera nachzuverfolgen. Hierbei definieren der Einsatzzweck und die Sensorkonfiguration die Herangehensweise. Im Wesentlichen wird die Herangehensweise von den erhaltenen Daten der Image Sequence bestimmt. Abschließend wird für VO ein lokaler Optimierungsschritt, üblicherweise mittels Bundle Adjustment durchgeführt (Scaramuzza und Fraundorfer, 2011).

In Kapitel 5 wird der für diese Arbeit identifizierte SLAM-Algorithmus vorgestellt und die involvierten Verfahren näher betrachtet.

3.3.5 Kartenkonstruktion

Die in Kapitel 3.3.3 vorgestellte Visuelle Odometrie ermöglicht lediglich eine temporäre und lokale Abbildung der Raumgeometrie (Fraundorfer und Scaramuzza, 2012). Erst wenn die durch die Observation gewonnenen Daten innerhalb einer Karte persistiert werden, können Lokationsdaten virtueller Objekte und die eigene Trajektorie global nachverfolgt werden (Fraundorfer und Scaramuzza, 2012). Nur wenn eine Kartierung erfolgt, handelt es sich um SLAM (Williams und Reid, 2010). Ohne Kartierung, handelt es sich um Visuelle Odometrie, was als mögliche Teilmenge von SLAM gilt (Williams und Reid, 2010). Zur Realisierung

einer MR-Anwendung, inklusive der Erfüllung aller Voraussetzungen für Spatial Computing, respektive Spatial Mapping, muss eine Karte erstellt werden (Weinmann et al., 2020, Greenwold, 2003). Nur dann können stationäre virtuelle Objekte innerhalb einer physisch konsistenten Pose vorgehalten werden (Çöltekin et al., 2020).

Die Kartierung kann in zwei unterschiedlichen Varianten erfolgen. Zum einen kann eine rein topologische Karte erstellt werden. Diese dient im Wesentlichen der Nachverfolgung der eigenen Trajektorie. Zum anderen werden metrische bzw. semantische Karten eingesetzt. Hierbei muss vorab eine Kalibrierung der Kamera erfolgt sein, sodass Pixelinformationen in die metrische Korrespondenz übersetzt werden können. Ohne diese Translation, können keine absoluten, sondern lediglich relative physische Verhältnisse ermittelt werden. (Scaramuzza und Fraundorfer, 2011, Fraundorfer und Scaramuzza, 2012). Für MR-Anwendungen im Kontext Spatial Computing, kommt daher nur eine semantische Representation und Konstruktion der Karte in Frage (Greenwold, 2003, Çöltekin et al., 2020, Weinmann et al., 2020).

3.4 Zusammenfassung

Windows-MR stellt sich als Interpretation basierend auf dem RV-Continuum dar. Microsoft bezieht allerdings Umgebungsdaten stärker in die Simulation mit ein. Dazu wird ein geeignetes Ein-/Ausgabegerät benötigt. Mit der HoloLens schuf MS ein HMD, das den eigenen Anforderungen an Windows-MR gerecht wird (Bray, 2021).

Visual SLAM stellt sich je nach Anwendungsfall als sehr spezifisch dar. Anhang 1 zeigt eine Zusammenfassung der Taxonomie moderner vSLAM-Systeme. Es können zur Informationsextraktion ein oder mehrere optische Sensoren eingesetzt werden. Weiter wird unter Filter-basierten und Optimierungsansätzen unterschieden, wobei letztere bessere Ergebnisse erzeugen. Merkmalsbasierte Verfahren erzeugen eine spärliche Rekonstruktion der Szene, welche für die Ermittlung der Pose ausreichend ist. Zudem können merkmalsbasierte Ansätze performant in Echtzeit operieren. Eine dichte Rekonstruktion, kann durch direkte Verfahren erreicht werden. Diese wird oft für Hand Tracking (vgl. HoloLens) eingesetzt. Die Kartenkonstruktion kann topologisch, sowie metrisch bzw. semantisch erfolgen, wobei letztere Variante für Spatial Computing essentiell ist (Chen et al., 2018b).

4 Lösungskonzept

In diesem Kapitel werden die Anforderungen an den Prototypen analysiert und präsentiert. Zunächst wird die Umgebung und das Einsatzgebiet, als Voraussetzung für die Identifikation einer geeigneten Lösungsstrategie für SLAM analysiert und definiert. Anschließend werden die Anforderungen an AR/VR-Systeme von Dörner et.al. verwendet, um die Anforderungen an MR-Systeme zu definieren (Dörner et al., 2013).

In Kapitel 4.3 wird eine geeignete Lösungsstrategie für das SLAM-Problem, unter Einhaltung der angelegten Anforderungskriterien identifiziert, welche in Kapitel 5 vorgestellt wird. Abschließend zu diesem Kapitel, wird der konzeptionelle Systemaufbau des Prototypen präsentiert.

4.1 Umgebungsdefinition und Einsatzgebiet

Die Identifikation einer geeigneten Lösungsstrategie für das SLAM-Problem, wird in erster Linie durch den Einsatzort und Zweck des Systems definiert. Physische Gegebenheiten lassen sich somit in dynamische, semi-dynamische und statische Umgebungen klassifizieren (Hentschel und Wagner, 2011, Stachniss, Leonard und Thrun, 2016).

Eine dynamische Umgebung liegt z.B. in verkehrsähnlichem Kontext vor (Li et al., 2018). Hierbei ist es nicht immer möglich, die Bewegungen von Passanten vorauszusehen und demnach aus der Datenverarbeitung zu exkludieren. Die dynamische Bewegung, bezieht sich hierbei zudem auf die Merkmale, welche oftmals an eben diesen beweglichen Objekten detektiert werden. Daher gelten dynamische Umgebungen als bisher noch schwer analysierbar, da bspw. die global konsistente Pose eines Merkmals aufgrund dynamischer Bewegung u.U. über reinige Frames hinweg nicht detektiert werden kann (Li et al., 2018, Yong-bao et al., 2020). Ebenso müssen stationäre von nicht-stationären Objekten unterschieden werden, welche jeweils anders behandelt müssen. Zur reinen Lokalisierung werden daher meist nur zeit- und positionskonstante Objekte und Merkmale verwendet (Henein et al., 2020, Hentschel und Wagner, 2011).

Rein statische Umgebungen können nur simuliert werden, da selbst wechselnde Lichtverhältnisse unvorhersehbare Dynamik produzieren können. Demnach beinhaltet die

physischen Realität immer Dynamik (Thrun, Burgard und Fox, 2005). Semi-Statistische Umgebungen bestehen hauptsächlich aus statischen Objekten. Die Dynamik wird durch wechselnde Lichtverhältnisse, wie einstrahlendes natürliches Licht oder Luftbewegungen bzw. minimale Objektbewegungen erzeugt (Borthwick und Durrant-Whyte, 1994, Hentschel und Wagner, 2011).

Außenbereiche sind i.d.R. dynamischer Natur und fordern sehr gut abgestimmte Algorithmen und Methoden – oft aus dem Bereich der Künstlichen Intelligenz (KI). Hierbei werden KI-Methoden genutzt, um ungewünschte Dynamik aus den Bilddaten zu entfernen (Yong-bao et al., 2020). Diese Arbeit befasst sich mit Lösungen für Innenbereiche und vernachlässigt nachfolgend Outdoor-Szenarien.

Vor der Entwicklung ist es essentiell den Zweck der MR-Applikation präzise zu analysieren und zu definieren. Daher werden Innenräume als semi-statische Umgebungen klassifiziert und im weiteren Verlauf ausschließlich für diesen Einsatzzweck geeignete vSLAM-Verfahren bzw. Lösungsstrategien berücksichtigt. Die Lösungsansätze müssen robust genug arbeiten, um bei sporadisch auftretender Dynamik, weiterhin ausreichend präzise Lokationsergebnisse zu produzieren (Hentschel und Wagner, 2011).

4.2 Anforderungserhebung und -analyse

Das System wird anhand der in Kapitel 4.1 definierten Formen der möglichen Einsatzumgebungen für semi-statische Environments ausgelegt und hat daher nicht den Anspruch in Außenbereichen zu funktionieren bzw. extreme Dynamik verarbeiten zu können. Weiter soll die Lösung die Daten in Echtzeit auf dem Prozessor der eingesetzten Hardware verarbeiten können. Nachfolgenden werden in Kapitel 4.2.1 und 4.2.2 die funktionalen und nicht-funktionalen Anforderungen an das System aufgestellt (Sommerville, 2016).

4.2.1 Funktionale Anforderungen

- Auswertung von Bilddaten durch den SLAM-Algorithmus in Echtzeit
- Ausgabe von Positionsinformationen zur Raumgeometrie und Pose der Kamera
- Anbindung an die Unity Engine mit schneller Datenübertragung für Karten- und Positionsdaten

- Rendering virtueller Szene in Unity Engine mit den Daten des SLAM-Algorithmus
- Echtzeit-Tracking natürlicher Merkmale

4.2.2 Nicht-funktionale Anforderungen

- Das System soll in Echtzeit ausführbar sein
- Das System soll nachvollziehbar dokumentiert sein
- Das System soll modulare aufgebaut sein, sodass der SLAM-Algorithmus ggf. ausgetauscht werden kann
- Das System soll hinsichtlich einer Weiterentwicklung anpassbar bleiben
- Das System soll die Entwicklung einer MR-Anwendung ermöglichen
- Das System soll robust genug arbeiten und demensprechend tolerant gegenüber Fehlzuständen reagieren, um das Tracking nicht durch Systemabstürze vorzeitig zu beenden

4.3 Identifikation einer SLAM-Lösungsstrategie

In Kapitel 3.3 wird die Arbeitsweise von vSLAM-Algorithmen vorgestellt. In dieser Arbeit werden die für AR bzw. MR geltenden Anforderungen, auf die fokussierte vSLAM-Lösungsstrategie übertragen. Diesbezüglich muss der eingesetzte Algorithmus die Bildinformationen in Echtzeit verarbeiten, um präzise Lokationsdaten einerseits der observierten Merkmale und andererseits der Kamerapose zu liefern (Dörner et al., 2013). Eine spärliche Rekonstruktion, wird hierbei aus Gründen der Performanz bevorzugt, um die Echtzeitfähigkeit auch ohne Verwendung einer Grafikkarte als Recheneinheit gewährleisten zu können. Da im Zuge dieser Arbeit keine MR-Applikation entwickelt werden soll, sondern lediglich das SLAM-Problem für MR-Anwendungen in Form eines Prototypen, bezogen auf reine Machbarkeit analysiert werden soll, wird die Grafikkarte ausschließlich für das Rendering verwendet.

Um die höchstmögliche Flexibilität zu erhalten, werden Open-Source-Varianten bevorzugt. Dadurch lassen sich Eingriffe in den Source-Code mit der Lizenz vereinbaren. Weiter werden Algorithmen bevorzugt, welche sich mittels der eingesetzten Softwarekomponenten vereinbaren lassen. Diesbezüglich kommen lediglich Verfahren in Frage, welche sich in ROS integrieren lassen und OpenCV unterstützen. Zuzüglich dessen,

wird eine Anbindung des Algorithmus an die Unity Engine angestrebt, um die künftige Entwicklung einer MR-Applikation durch die Erkenntnisse dieser Arbeit zu ermöglichen.

ORB-SLAM2 von Mur-Artal et al. bietet eine Lösungsstrategie, welche sich mittels aller vorgestellten – für vSLAM geeigneten – Sensortypen realisieren lässt (Mur-Artal und Tardos, 2017). In dieser Arbeit wird der monokulare Ansatz verfolgt, da hierbei lediglich die reine Machbarkeit analysiert werden soll. Konkret soll überprüft werden, ob auch ohne die Verwendung einer HoloLens die Lösung des SLAM-Problems und damit verbunden MR möglich ist. ORB-SLAM2 ist für den Einsatz in Echtzeit konzipiert und bietet die Möglichkeit der Integration in ROS bzw. wird zudem von ROS selbst unterstützt (Mur-Artal und Tardos, 2017). Da ORB-SLAM2 zudem unter Open-Source verfügbar ist, darf der Source-Code verändert werden (Mur-Artal und Tardos, 2017).

ORB SLAM2 wird in Forschungsprojekten als Grundlage genutzt (Ouerghi et al., 2020) bzw. wurde mehrfach evaluiert (Mishra, Griffin und Sevil, 2021, Zhao und Vela, 2020). Demnach ist davon auszugehen, dass durch die Verwendung von ORB-SLAM2, valide Lokalisationsdaten erzeugt werden können (Cui und Wen, 2019). Demensprechend kann ORB-SLAM2 als valide Lösungsstrategie für die Entwicklung des Prototypen identifiziert werden und wird daher in Kapitel 5 vorgestellt.

4.4 Systemaufbau

Abbildung 26 zeigt den konzeptionellen Systementwurf. Die Bilddaten sollen via ROS an den ORB-SLAM2-Algorithmus übergeben werden. ORBSLAM soll die entsprechenden Bilddaten analysieren und dem in Kapitel 5 beschriebenen Algorithmus folgen. Anschließend sollen die ermittelten Posen, sowie die Kartierungsdaten, über die TCP-Schnittstelle an die Unity Engine übergeben werden. Danach soll das Datenpaket über den TCP-Endpoint von Unity empfangen werden. Weiter soll das Subscriber-Script die Umwandlung der Daten in ein für die Unity-Engine verwendbares Format konvertieren.

Um Objekte im Koordinatensystem von Unity darstellen zu können, muss zunächst das Koordinatensystem von ROS in das Unity-Koordinatensystem konvertiert werden (Epic Games, 2021, Cheers, 2021). Danach wird die Positionsinformation mit 100 multipliziert, um die Daten an den Maßstab der virtuellen Szene anzupassen. Über entsprechende primitive Körper sollen die Kartierungsdaten in Form einer Point Cloud dargestellt werden

Lösungskonzept

und die Unity-Kamera der Pose der physischen Kamera folgen. Demensprechend sollen die Bewegungen der physischen, an die virtuelle Kamera übertragen werden. Die Point-Cloud-Daten sollen Teile der Raumgeometrie zeigen und in Verbindung mit der übertragenen Pose markerless-tracking bzw. die Lösung des SLAM-Problems nachweisen.

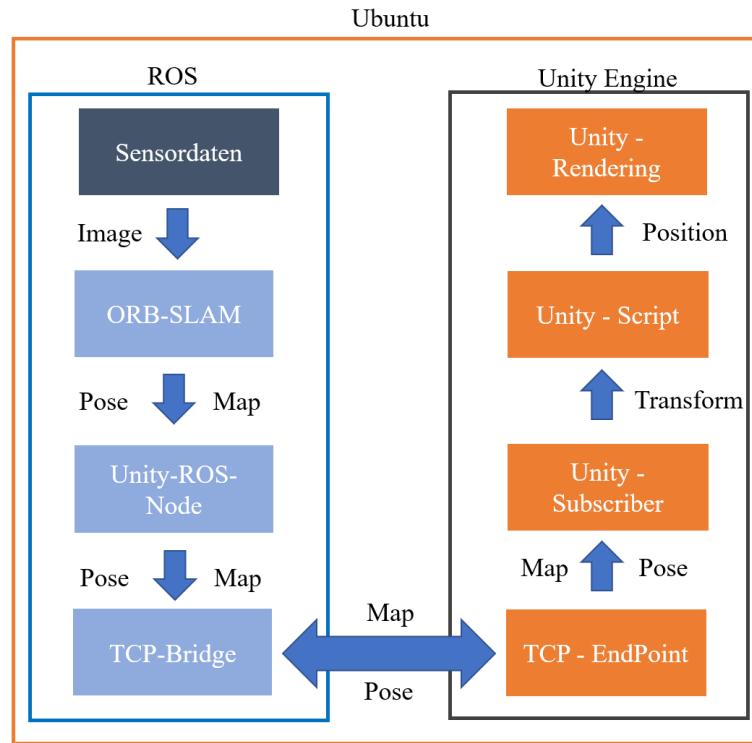


Abbildung 25: Systemaufbau in Anlehnung an (Thaller, 2021, Koubaa, 2016, Hussein, Garcia und Olaverri-Monreal, 2018, Epic Games, 2021)

Anhang 2 zeigt das Zusammenspiel der erforderlichen Komponenten hinsichtlich der Realisierung des Konzepts für den Prototypen und in Anhang 3 wird der Data-Flow visualisiert.

5 ORB-SLAM

Ethan Rublee hat 2011 oriented FAST and rotated BRIEF (ORB) vorgestellt (Rublee et al., 2011). Dies sollte die Nachteile von SIFT und SURF ausgleichen und unter Open-Source verfügbar gemacht werden (Rublee et al., 2011). SIFT und SURF liefern zwar gute Ergebnisse, sind allerdings durch Patente geschützt bzw. können nur in Kombination mit einer Grafikkarte als Recheneinheit in Echtzeit operieren (Chien et al., 2016). ORB-Merkmale daher ein guter Kompromiss zwischen Performance und Robustheit bzw. Präzision dar. Hierbei wird eine verbesserte Variante der FAST-Technik verwendet, um Merkmale zu identifizieren und direction-normalized BRIEF zur Extraktion verwendet. Durch Downsampling der Images - auch Image Pyramid genannt – werden in der Computer Vision und Bildverarbeitung verschiedene Skalierungen der Szene bzw. Objekte in der Szene behandelt (Peters, 2017). ORB nutzt dieses Verfahren in Kombination mit FAST und wendet für jedes Layer der Pyramide FAST erneut an (Rublee et al., 2011). Anschließend wird eine rotations-invariante Version des BRIEF-Algorithmus angewendet. Dadurch können Rotationen der Szene erkannt werden (Rublee et al., 2011).

Das Konzept des BRIEF-Algorithmus besteht aus einer Reihe von Tests auf Pixelebene, die schlussendlich das identifizierte Merkmal beschrieben. Hierbei wird jeder Pixel in einem definierten Areal um das identifizierte Merkmal analysiert und mit einem binären Descriptor beschrieben. Dadurch kann das einzelne Merkmal als Serie von n-Tests in einem Bit-String zusammengefasst und beschrieben werden (Kaehler und Bradski, 2015, Rublee et al., 2011).

Abbildung 29 zeigt Beispiele der ungefilterten Merkmalsextraktion mittels ORB. Die Bilder 1 und 2 zeigen gut texturierte Oberflächen in schlechten Lichtverhältnissen. In Bild 1 können aufgrund der schlechten Lichtverhältnisse nur Schattierungen wahrgenommen werden. Jede Bewegung erzeugt hierbei neue Schattenverläufe, welche die Szene ändern und dadurch ein erneutes Identifizieren der vorherigen Merkmale verhindert. Bild 2 zeigt eine gute Struktur, enthält durch die repetitive Muster jedoch wenige voneinander unterscheidbaren Kanten. Bild 3 zeigt eine ebenmäßige strukturmöglichkeit Oberfläche, welche keine validen Merkmale bietet. Die Bilder 4-6 zeigen Beispiele, in denen mehrere Merkmale identifiziert werden konnten. Hierbei erzeugen die unterschiedlichen Objekte viele Merkmale. Die Szenen ist gut ausgeleuchtet und wird daher weniger durch Schattenverläufe beeinflusst. Dies impliziert, dass zu einer erfolgreichen Merkmalsextraktion mit ORB aus

ORB-SLAM

einer Szene in Innenräumen, die in den Bildern 4-6 gezeigten Zustände erreicht werden sollten. Anhang 4 zeigt das für die Beispiele verwendete Programm zur ORB-Feature-Extraktion.



Abbildung 26: Beispiele Merkmalsextraktion ORB

ORB SLAM basiert auf den ORB-Features und wird unter Open-Source weiterentwickelt (Mur-Artal, Montiel und Tardos, 2015). Aktuell befindet sich ORB-SLAM3 in der Beta-Phase (Campos et al., 2021). In dieser Arbeit wird ORB-SLAM2 verwendet, was auf dem von Mur-Artal 2015 vorgestellten ORB-SLAM (Mur-Artal, Montiel und Tardos, 2015) für monokulare Systeme basiert. ORB-SLAM2 unterstützt bereits RGB-D und Stereosysteme (Mur-Artal und Tardos, 2017). Daher eignet sich der Algorithmus auch für auf diese Arbeit folgende Forschungsbestreben.

ORB-SLAM

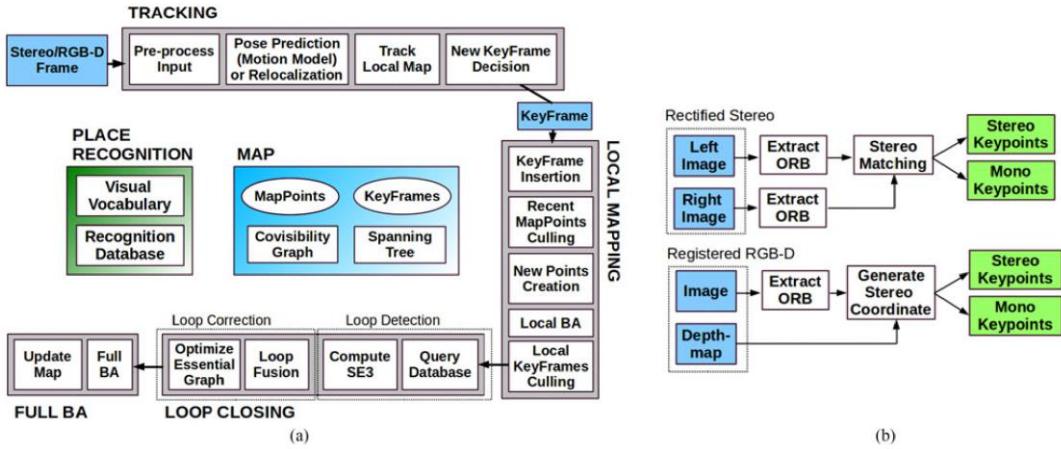


Abbildung 27: Übersicht ORB-SLAM2 (Mur-Artal und Tardos, 2017)

ORB-SLAM2 wurde 2017 von Mur-Artal et al. vorgestellt (Mur-Artal und Tardos, 2017). Abbildung 30 zeigt den Aufbau des Algorithmus. Demnach wird ORB-SLAM2 in drei Hauptthreads unterteilt: tracking, local mapping und loop closing. Die hohe Kompatibilität, bezogen auf die einsetzbare Sensorik, wird durch die Entkopplung des Tracking Moduls erreicht, welches das gesamte Image-Processing übernimmt. Anschließend wird lediglich mit den extrahierten Daten weitergearbeitet. Da in dieser Arbeit der monokulare Ansatz betrachtet wird, werden Verfahrensabläufe den Stereofall betreffend, bzw. Methoden zur Verwendung von RGB-D-Kameras nicht weiter berücksichtigt (Mur-Artal und Tardos, 2017).

Das Tracking-Modul dient der Lokalisierung der Kamera im Raum mittels den Bilddaten aus jedem Frame. Hierbei werden Features extrahiert und diese werden mit der lokalen Karte verglichen. Dadurch können entsprechende Matches identifiziert werden und lokales Bundle Adjustment zur Minimierung des Messfehlers praktiziert werden (Mur-Artal und Tardos, 2017). Durch das Bundle Adjustment werden die Kameraparameter und Positionen der extrahierten Merkmale im Raum, an die aufgenommen Bilder angepasst, um die Messfehler auf alle Aufnahmen zu verteilen. Dies ermöglicht eine gleichmäßige Verteilung der Messfehler, sodass der Fehler pro Messung möglichst optimiert wird. Hierbei stellt das Bundle bzw. Bündel entsprechend zusammengefasste Bildaufnahmen einer Serie dar, welche optimiert werden soll. Dazu werden Strahlen, ausgehend vom Projektionszentrum, der Kamera bis hin zum identifizierten Merkmal analysiert (Engels, Stewénius und Nistér, 2006).

Über korrespondierte Punkte (Feature Matching) werden die Bilder miteinander verknüpft. Dadurch können die Features über eine Bildserie hinweg erneut identifiziert werden. Anschließend werden die Strahlen mittels eines Gleichungssystems berechnet. Durch die Ausgleichungsberechnung wird der Fehler optimiert. Dies wird durch eine hohe Anzahl der einbezogenen Bilder erreicht (Engels, Stewénius und Nistér, 2006).

Der Loop-Closing-Zyklus von ORB-SLAM2, erkennt bereits kartierte Abschnitte der Szene erneut. Dadurch kann der angesammelte Drift (vgl. Scale Drift, Kapitel 3.3.2) reduziert bzw. ausgeglichen werden. Anschließend wird pose-graph-optimization durchgeführt (Mur-Artal und Tardos, 2017, Grisetti et al., 2010). Hierbei werden die Frames mittels dem Bag-of-Word-Ansatz von Galvez-Lopez et al. in eine hierarchische Representation transferiert (Galvez-López und Tardos, 2012). Dazu wird anschließend mit einem Nearest-Neighbour-Ansatz ein visuelles Vokabular der Bilddaten erstellt, was schnell abgefragt werden kann (Galvez-López und Tardos, 2012). Dadurch wird Loop-Closing bzw. das Identifizieren bereits observierter Kartenabschnitte ermöglicht (Galvez-López und Tardos, 2012, Mur-Artal und Tardos, 2017).

6 Realisierung

In diesem Kapitel folgt die Realisierung. Zunächst wird die Systemumgebung bzw. der technische Aufbau des Systems und die damit verbundene vorbereitende Konfigurationsschritte, bezogen auf das verwendete Betriebssystem, vorgestellt. Anschließend werden die für diese Arbeit relevanten Teilespekte zur Implementierung des ORB-SLAM2-Algorithmus dargelegt. Danach wird die Integration des SLAM-Algorithmus in das ROS, insbesondere die Verbindung zur Unity Engine präsentiert. Abschließend folgt die Vorstellung der Entwicklungsschritte zur Nutzbarmachung der durch den ORB-Algorithmus errechneten Lokationsdaten in der Unity Engine.

6.1 Sensoren

Die Zielstellung der Arbeit fordert eine hardwareunabhängig Lösung des SLAM-Problems. Daher wird eine handelsübliche Kamera eingesetzt. Dies erfolgt im Hinblick auf Reproduzierbarkeit und einer vereinfachten Umsetzung der hier vorgestellten Ergebnisse. Die Logitech C270 wird als Sensor identifiziert, da diese im Kontext vSLAM als valider Sensortyp klassifiziert wird. Entsprechende Forschungsprojekte demonstrieren den erfolgreichen Einsatz der Logitech C270 (Oleari, Lodi Rizzini und Caselli, 2013).



Abbildung 28: Logitech C270 (Logitech, 2021)

Abbildung 31 zeigt die Logitech C270, welche eine maximale Bildwiederholungsrate von 1280 x 720 Pixel mit einer Frequenz von 10 Hz erzeugen kann. Zudem wird die vollständige Kompatibilität mit dem UVC-Standard geboten, was anschließend die Anbindung an ROS erleichtert. Die Anbindung erfolgt über USB 2.0. Da die Bilder auf 640 x 480 Pixel komprimiert werden, wird die Schnittstelle nicht durch die Transferrate ausgelastet (Oleari, Lodi Rizzini und Caselli, 2013).

6.2 Hardware

Als Basis des Systems wird ein Laptop aus dem Hause MSI genutzt. Das MSI G72 8RE wurde modifiziert und alle wesentlichen Komponenten in Tabelle 1 präsentiert (msi.com, 2021).

CPU	Intel I7 – 8750H
Arbeitsspeicher	16 GB
Hard Drive	1 TB NVMe SSD
Grafikkarte	NVIDIA GTX 1060 6GB mobile

Tabelle 1: Relevante Hardwarekonfiguration des Testsystems (msi.com, 2021)

6.3 Software

Kapitel 2.7 stellt die zur Realisierung verwendeten Softwareprodukte vor. In diesem Teilkapitel werden die verwendeten Versionen, sowie Anhang 5 die jeweiligen Abhängigkeiten in präsentiert. Die Abhängigkeiten werden nur unter Berücksichtigung des verwendeten Betriebssystems Linux Ubuntu betrachtet und andere Betriebssysteme (OS) vernachlässigt.

6.4 Kamerakalibrierung

Die Kamerakalibrierung wird unter Verwendung von OpenCV durchgeführt (Kaehler und Bradski, 2015). Hierbei wird die Methodik von Zhang et al. verfolgt (Zhang, 2000). Den im Anhang 6 aufgeführten Programm, werden die aufgenommenen Bilder zur Auswertung präsentiert. Es konnte aus der Fachliteratur nicht ermittelt werden, welche Anzahl an Bildern hierbei die besten Ergebnisse liefern würden. Daher werden 100 Aufnahmen durch den Algorithmus auswertet und in Anhang 7 die Ergebnisse der Kalibrierung präsentiert. Abbildung 32 zeigt einen Auszug der Rohdaten für die Kalibrierung mit dem vorgeschlagenen Kalibrierkörper.

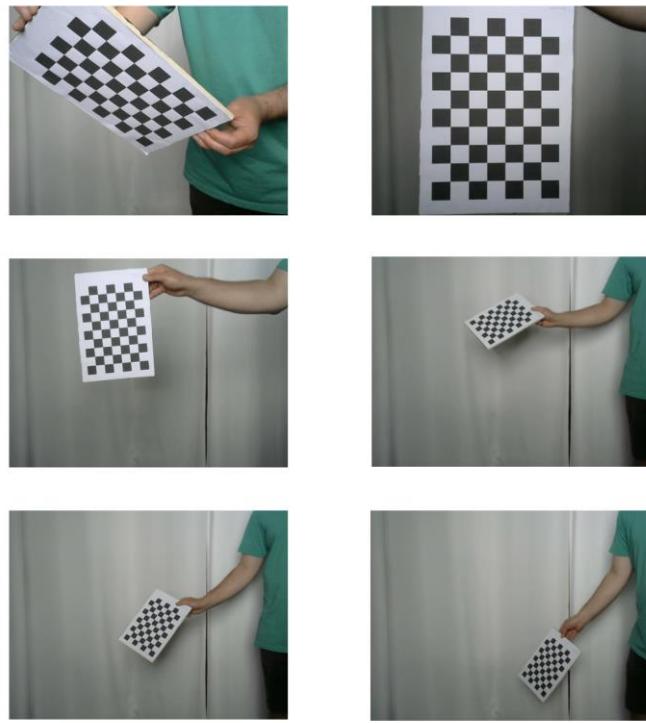


Abbildung 29: Aufnahmen für die Kamerakalibrierung

6.5 Implementierung ORB-SLAM2-Native

Die Implementierung wird unter Verwendungen des Betriebssystems Linux Ubuntu durchgeführt. Die Ubuntu-Versionen 14.04, 16.04, 18.04 und Windows 10 wurden im Zuge der Implementierung von ORB-SLAM2 erfolglos erprobt. Aufgrund unterschiedlichster Hindernisse, wie bspw. Kompatibilitätsprobleme und unlösaren Abhängigkeiten, konnte unter den durch Mur-Artal et al. vorgeschlagenen Versionen keine lauffähige Variante des ORB-SLAM2-Algorithmus implementiert werden (Mur-Artal, Montiel und Tardos, 2015, Mur-Artal und Tardos, 2017).

Allerdings konnten in Verbindung mit OpenCV Version 4 und aktuellen Versionen der benötigen Bibliotheken alle Abhängigkeiten gelöst werden. Hierzu war es erforderlich, in den Quellcode von ORB-SLAM2 einzugreifen und die Kompatibilität zu OpenCV 4 herzustellen. Dazu wurden die Fehlermeldungen bzgl. der Kompilierung analysiert und im Quellcode alle veralteten Bezeichner von OpenCV 3 durch die unter OpenCV 4 verwendeten

ersetzt. Zusätzlich wurden die für die Kompilierung relevanten CMakeFiles, an die neue OpenCV-Version angepasst, um den Algorithmus kompilieren zu können.

6.6 Anpassung des AR-Demo für ORBSLAM2

Das durch die Repository von ORB-SLAM2 durch Mur-Artal et al. bereitgestellte AR-Demo, ist für die Verwendung von ROS als Middleware konzipiert. Um den Algorithmus nativ bzgl. der AR-Fähigkeiten zu testen, müssen Anpassungen vorgenommen werden. In Anhang 8 wird der angepasste Code komplett aufgeführt. Listing 1 zeigt hierbei einen Auszug der Änderungen.

```
cv::VideoCapture cap;
cap.open(camIndex);
cv::Mat img;

if (!cap.isOpened()) {
    cerr << "Failed to open camera!\n";
    return -1;
}
```

Listing 1: Änderungen an ORB-SLAM2 AR-Demo – Image Capturing

Listing 1 zeigt das Image-Capturing durch OpenCV statt durch ROS. Hierbei werden die durch die USB-Camera aufgenommenen Frames direkt an OpenCV übergeben. Sofern keine Kamera gefunden wird, wird eine Fehlermeldung ausgegeben und das Programm vorzeitig beendet.

```

bool done = false;
cv::Mat im_raw;
cv::Mat im;

while (!done) {
    cap >> im_raw;
    cv::cvtColor(im_raw, im, cv::COLOR_RGB2BGR);
    cv::Mat imu;

    cv::Mat Tcw = SLAM.TrackMonocular(im, time(NULL));
    int state = SLAM.GetTrackingState();
    vector<ORB_SLAM2::MapPoint*> vMPs = SLAM.GetTrackedMapPoints();
    vector<cv::KeyPoint> vKeys = SLAM.GetTrackedKeyPointsUn();

    cv::undistort(im, imu, K, DistCoef);

    if (bRGB)
        viewerAR.SetImagePose(imu, Tcw, state, vKeys, vMPs);
    else
    {
        cv::cvtColor(imu, imu, CV_RGB2BGR);
        viewerAR.SetImagePose(imu, Tcw, state, vKeys, vMPs);
    }
}
cap.release();

// Stop all threads
SLAMShutdown();

// Save camera trajectory
SLAM.SaveKeyFrameTrajectoryTUM("KeyFrameTrajectoryAR.txt");

```

Listing 2: Änderungen and ORB_SLAM2 AR-Demo

Listing 2 zeigt die Hauptschleife des Programms. Hier muss zunächst das eingehende Bild in den für ORB-SLAM2 benötigten Farbraum konvertiert werden. Sonst würden alle Ausgabeframes farblich verzerrt dargestellt. Anschließend wird das Tracking-Modul via ORB-SLAM2 gestartet und die Initialisierung des Algorithmus eingeleitet. Die Images werden durch die Kalibrierungsdaten korrigiert und nach Abschluss des SLAM-Verfahrens dem AR-Viewer übergeben. Der Ablauf wiederholt sich, bis das Programm durch die Nutzereingabe beendet wird.

6.7 ORBSLAM2-ROS-Integration

Sobald unter ROS der Arbeitsbereich erstellt und kompiliert wurde, können Zusatzprogramme in Form von sog. Nodes integriert werden. ROS übernimmt hierbei einen

Teil der Abhängigkeitsverwaltung und kann zudem über den eigenen Paketmanager für ROS optimierte Pakete installieren (Koubaa, 2016).

6.8 Anbindung Kamera

ROS biete eine Node zur Verwendung von USB-Cameras an. Diese muss vor der Verwendung installiert und kompiliert werden (Okada, 2021).

```
<launch>
<node name = "camera" pkg = "usb_cam" type = "usb_cam_node" output =
"screen" >
<param name = "video_device" value = "/dev/video0" / >
<param name = "image_width" value = "640" / >
<param name = "image_height" value = "480" / >
<param name = "pixel_format" value = "yuyv" / >
<param name = "camera_frame_id" value = "usb_cam" / >
<param name = "io_method" value = "mmap" / >
<param name = "camera_info_url" value =
"file:///home/major/Dokumente/left.yaml" / >

</node>
<node name = "image_view" pkg = "image_view" type = "image_view" respawn =
"false" output = "screen">
<remap from = "image" to = "/camera/image_raw" / >
<param name = "autosize" value = "true" / >
</node>
</launch>
```

Listing 3: USB-Kamera Launch-File

Listing 3 zeigt das angepasste Launch-File für die USB-Cam-Node. Hier erfolgt die Konfiguration der Kamera. Für ORB-SLAM2 werden hierfür Bilder mit 640 x 480 Pixel angefordert, um die Bildverarbeitung performant zu halten. Die Kamera kann eine maximale Auflösung von 1280 x 720 Pixel produzieren. Dies würde das System u.U. zu sehr belasten, sodass die performante Ausführung von Zusatzprogrammen, nur noch eingeschränkt erfolgen könnte. Daher wird die Bildauflösung reduziert.

Des Weiteren werden die Topics, unter denen die Daten für ROS publiziert werden, definiert. Topics können als Thema interpretiert werden, welche andere Nodes abonnieren können. Konkret wird zwischen Publisher und Subscriber unterschieden. Vergleicht man das Prinzip mit dem Sprechfunk, würde der Publisher einem Sender gleichkommen und der

Subscriber einem Empfänger. Daher kann sich der Subscriber in die Frequenz einwählen und die Daten empfangen. Die Frequenz stellt hierbei das ROS-Topic dar, unter diesem die Daten publiziert werden. Die fertigstellte Node stellt die durch die Kamera erfassten Bilder für weitere Nodes zur Verfügung und wird durch das Launch-File gestartet (Koubaa, 2016).

6.9 Unity Engine

Die Unity Engine kann plattformübergreifend installiert und genutzt werden (Hocking, 2015). Für diesen Prototypen wird die Version 2020.3.15f2 unter Ubuntu 20.04.2 genutzt und installiert.

6.9.1 Unity-Robotics-Hub

Der Unity-Robotics-Hub bietet die Möglichkeit an, ROS mit Unity zu verbinden (Hussein, Garcia und Olaverri-Monreal, 2018). Dadurch kann der Datenaustausch über eine TCP-Bridge erfolgen. Dazu werden die in der Dokumentation angegebenen Konfigurationsschritte verfolgt und die entsprechenden Nodes in ROS integriert (Epic Games, 2021).

6.9.2 TCP-Bridge

Zur Initialisierung der TCP-Bridge, wird die Node in ROS, anhand der Dokumentation integriert (Cheers, 2021). Ebenso ist die Bridge im Unity-Projekt bereits enthalten. Hierbei wird auf beiden Seiten die Konfiguration vorgenommen und die IP-Adressen konfiguriert.

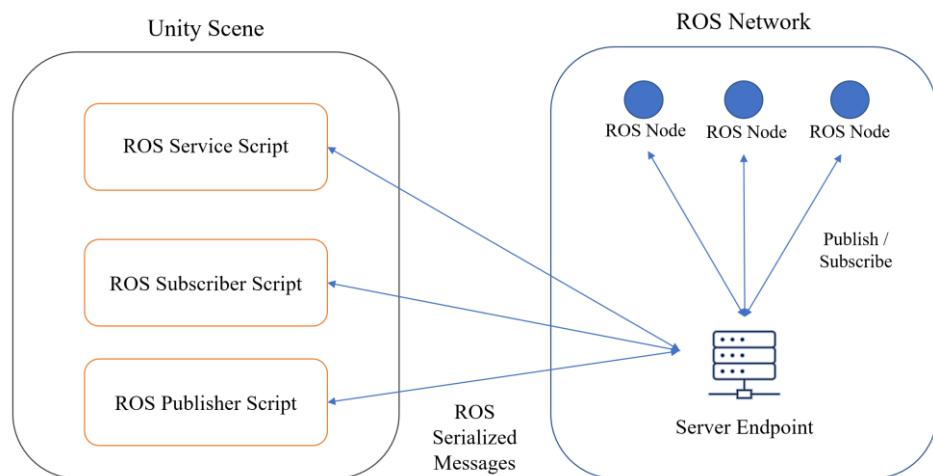


Abbildung 30: ROS-Unity Kommunikation in Anlehnung an (Cheers, 2021)

Abbildung 33 visualisiert die Kommunikation zwischen ROS und der Unity Engine. Das Zentrum stellt der Server-Endpoint dar. An dieser Stelle wird die bidirektionale, verbindungsorientierte und paketbasierte Verbindung zu allen Gegenstellen realisiert. Die durch ROS bereitgestellten Nodes, kommunizieren über den Server-Endpoint mit angeschlossenen Anwendungen. Hierzu werden die ROS-Messages zur Übertragung serialisiert und in der Unity Engine deserialisiert. Die entsprechenden Scripts werden angesteuert und behandeln die empfangenen Daten individuell (Hussein, Garcia und Olaverri-Monreal, 2018, Koubaa, 2016, Miller, 2021).

6.9.3 Unity Projekt

Zur Implementierung des Unity-Projekts, wird auf das Unity-Robotics-Beispielprojekt zurückgriffen und dieses weiterentwickelt. Nachfolgend werden die dazu erstellen Scripts vorgestellt (Miller, 2021).

6.9.4 Subscriber für die Kamerapose

Die Kamerapose wird durch ROS-Messages bereitgestellt. ROS-Messages sind unter ROS genutzte Datentypen, die als Grundlage für die Generierung der C#-Klassen in Unity dienen. Die Erstellung der entsprechenden Rohklassen, erfolgt hierbei über eine graphische Oberfläche in der Unity Engine. Die über ORB-SLAM2 bereitgestellten Topics, müssen analysiert werden, um den verwendeten Datentypen zu ermitteln. Anschließend lassen sich die Rohklassen über die Benutzeroberfläche generieren (Epic Games, 2021).

```

void Start()
{
    ROSConnection.instance.Subscribe<RosTF>("tf", respond);
}

void respond(RosTF msg)
{
    camTranslation = new
Vector3((float)msg.transforms[0].transform.translation.y * -100,
(float)msg.transforms[0].transform.translation.z * 100,
(float)msg.transforms[0].transform.translation.x * 100);
    camRotX = (float)msg.transforms[0].transform.rotation.y;
    camRotY = (float)msg.transforms[0].transform.rotation.z;
    camRotZ = (float)msg.transforms[0].transform.rotation.x;
}

void Update() {
    cam.transform.position = camTranslation;
    // camRotZ - Roll
    cam.transform.rotation = Quaternion.Euler(camRotX * 100, camRotY *
-100, camRotZ * -100);
}

```

Listing 4: Unity Script Subscriber Lokationsdaten

Listing 4 zeigt den Subscriber für die Lokationsdaten. Durch die Start-Methode wird die Verbindung zu ROS hergestellt und das Topic „tf“ abonniert. Hierbei handelt es sich um sog. Pose-Stamped-Messages, welche die Pose der Kamera beinhalten. Die erhaltenen Daten, werden in das Koordinatensystem von Unity übertragen und zur Hauptkamera der Szene – mit dem Faktor 100 multipliziert – transportiert. Die Multiplikation ist notwendig, um den Maßstab der Trajektorie an die Szenendimensionen anzugeleichen. So kann die Translation und Rotation der Kamera in die Unity-Szene übertragen werden. Anhang 9 zeigt das vollständige Script.

6.9.5 Script zur Nachverfolgung der Trajektorie

Zur Verfolgung der Trajektorie werden Bread-Crumbs eingesetzt. In definierbaren Abständen hinterlässt die Kamera rote Meshes. Dadurch lässt sich die Trajektorie der Kamera in der Szene nachverfolgen. Dies ermöglicht den Vergleich zwischen der wahren und der virtuellen Bewegung der Kamera (Anhang 10).

6.9.6 Subscriber zur Integration der Karteninformationen

Um die physische Korrespondenz auch virtuell darstellen zu können und damit die gemischte Realität zu produzieren, werden Teile der physischen Karte in die Unity-Engine übertragen. Die Informationen der Karte werden in Form primitiver Meshes in die Szene projiziert. Dies erfolgt in Intervallen mit langen Zeitabständen, da die Projektion tausender Meshes die Systemperformance enorm beeinflusst. Nach der Initialisierung des Systems, werden die bis zu diesem Zeitpunkt generierten Landmarks aus der Karte in die Unity-Szene projiziert. Das dazu erstellte Script befindet sich in Anhang 11.

6.10 Zusammenfassung

Durch die Umsetzung der in diesem Kapitel beschrieben Schritte, konnte ein System erstellt werden, was zum einen den ORB-SLAM2-Algorithmus nativ implementiert und zum anderen die Integration von ORB-SLAM2 in ROS demonstriert. Durch die TCP-Bridge kann die Verbindung zur Unity Engine hergestellt werden. Durch den Datenaustausch können die durch den ORB-SLAM-Algorithmus gewonnenen Daten in der Unity Engine verwendet werden. Da die Unity Engine die Entwicklung von VR- und AR-Anwendungen unterstützt, kann in weiteren Forschungsvorhaben eine entsprechende Applikation entwickelt werden (Hocking, 2015). In Kapitel 7 werden die in diesem Kapitel realisierten Komponenten und das System getestet.

7 Test

Der in Kapitel 6 erstellte Prototyp wird in diesem Kapitel getestet. Hierfür werden im Verlauf dieses Kapitels Testvorhaben vorgestellt. Jeder Testdurchlauf wird aufgezeichnet und befindet sich auf den dieser Arbeit beiliegenden Datenträgern. Der Inhalt der Datenträger wird im Anhang 18 aufgeführt.

7.1 Testkriterien

Das Dataset der Technischen Universität München bietet 39 Sequenzen, die in zwei verschiedenen Umgebungen im Innenbereich aufgenommen wurden. Hierbei wurde ein kalibrierter Kinect-Sensor verwendet, welcher sowohl Farbbilder als auch Tiefeninformation bietet. Für die Testreihe sind lediglich die Bilddaten relevant. Zudem bietet das Dataset die Ground-Truth, die mit der durch den Algorithmus errechneten Trajektorie verglichen werden kann. Üblicherweise werden in der Fachliteratur SLAM-Verfahren anhand dem absolute translation root mean square error (RMSE) evaluiert (Sturm et al., 2012). Der RMSE gibt Aufschluss über die Genauigkeit der durch den Algorithmus bereitgestellten Lokalisationsdaten (Liu et al., 2020). Hierzu werden die Ergebnisse der Versuche mit den Ergebnissen von Liu et al. verglichen (Liu, Zhang und Bao, 2016, Liu et al., 2020). Liu et al. hat ebenso eine Implementierung von ORB-SLAM2 im Kontext Augmented Reality durchgeführt. Daher stellen die Ergebnisse von Liu et al. valide Vergleichsdaten zur Verfügung, um die in dieser Arbeit präsentierte Implementierung zu evaluieren.

Anschließend wird sowohl die Implementierung als auch das System auf die Merkmalsextraktionsqualität auf mehrere Entfernung hin überprüft. Hierzu wird unter Kapitel 7.2 der Aufbau der Testumgebung vorgestellt. Dadurch wird überprüft, wie gut Merkmale auf Entfernung bis zu 1,5 m identifiziert werden können, um den Vergleich zu den Eigenschaften der HoloLens zu ermöglichen. Die Rekonstruktionseigenschaften, werden für die Implementierung und das System anhand des gleichen Objekts überprüft, um die Kartengenerierung zu demonstrieren.

Für die Implementierung, als auch für das Gesamtsystem, werden Live-Tests durchgeführt. Dadurch wird die Echtzeitfähigkeit, bezogen auf die Lokalisierung der Kamera in Relation zur Raumgeometrie demonstriert. Hierzu wird das in dieser Arbeit angepasste

AR-Demo von ORB-SLAM2 genutzt, um die nativen AR-Fähigkeiten des Algorithmus zu überprüfen. Mur-Artal verwendet den RANSAC-Algorithmus zur Identifikation geeigneter Projektionsflächen für virtuelle Objekte aus den Point-Cloud-Daten (Mur-Artal und Tardos, 2017, Hollies und Fischler, 1981). RANSAC analysiert die Punkte der Point-Cloud, indem bei jedem Durchgang zufällig zwei Punkte aus der Menge identifiziert werden. Zwischen diesen Punkten wird eine Gerade gezogen. Jeder Punkt, der mit dieser Gerade kollidiert oder sich innerhalb eines definierten Bereiches befindet, wird als Score berechnet. Der Algorithmus überprüft alle möglichen Kombinationen und ermittelt so das beste Modell, um eine geeignete Projektionsfläche zu identifizieren (Hollies und Fischler, 1981).

Weiter wird das System, insbesondere die Verbindung zur Unity Engine, in Live-Tests demonstriert, um die Übertragung der Kamerapose, sowie der Kartierungsdaten zu demonstrieren. Dies soll einerseits die Echtzeitfähigkeit des Systems und andererseits die Nutzbarmachung der durch den Algorithmus erzeugten Daten in der Engine demonstrieren.

7.2 Testaufbau

In diesem Teilkapitel wird die Methodik der in dieser Arbeit durchgeführten Tests des erstellen Prototypen, sowie des ORB-SLAM2-Algorithmus vorgestellt.

7.2.1 Test TUM-Dataset

Die Tests mit dem TUM-Dataset werden gemäß des Vorschlags von Sturm et al. durchgeführt (Sturm et al., 2012). Es erfolgt hierbei kein Testaufbau, da das Dataset dem Algorithmus direkt übergeben wird. Diesbezüglich werden die durch Sturm et al. bereitgestellten Kalibrierungsdaten an ORB-SLAM2-Nativ übergeben. Nach erfolgreichem Durchlauf, werden die erhobenen Daten dem Online-Evaluations-Tool (Sturm, 2021) für das entsprechende Dataset übergeben (Sturm et al., 2012). **Anlage 12 und 13** zeigen die an den Algorithmus übergebenen, sowie die durch die Evaluation gewonnenen Daten.

7.2.2 Entfernungstest

Abbildung 34 visualisiert den Aufbau des Entfernungstest. Hierbei wird die Kamera an der grünen Markierung positioniert und der Algorithmus initialisiert. Dazu wird die Kamera entlang der horizontalen Linie jeweils um 15 cm versetzt. Anschließend wird eine Rotation

um 45° nach links und rechts durchgeführt. Der Algorithmus benötigt zur Initialisierung mehrere Perspektiven der Szene und gut strukturierte Objekte für die Identifikation valider Merkmale. Die geeigneten Objekte werden, wie in Abbildung 34 in unterschiedlichen Entfernung positioniert.

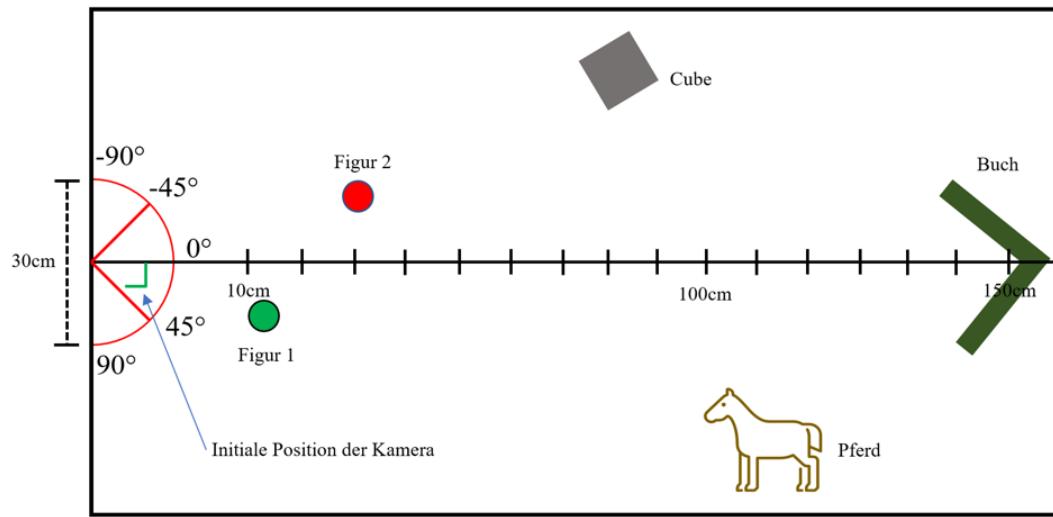


Abbildung 31: Testaufbau Entfernungstest

Anhang 12 zeigt den für die Testfälle verwendeten Aufbau des Entfernungstest in Anlehnung an den in Abbildung 34 konzipierten Aufbau. Der Test gilt als abgeschlossen, wenn aus der Perspektive der Kamera keine neuen Merkmale gefunden werden. Anhang 13 zeigt den Testaufbau aus der Perspektive der Kamera. Anhang 14 zeigt das Debug-Image des initialisierten Algorithmus und die erkannten Merkmale.

7.2.3 Live-Mapping

Die Testumgebung entspricht hierbei einer semi-statischen Umgebung, wie in Kapitel 4.1 vorgestellt. Der Test erfolgt frei und ohne Zeitbegrenzung. Hierbei soll eine möglichst gute Representation der Umgebung erreicht werden, um zu ergründen, wie dicht die Rekonstruktion und damit das Spatial Mapping erfolgt.

7.2.4 Rekonstruktionstest

Zur Rekonstruktion wird das Objekt „Pferd“ verwendet. Das Objekt bietet ausreichende Strukturen und Unterschiede in der Textur, um erfolgreich rekonstruiert zu werden. Dazu wird das Objekt mehrfach abgetastet und die Kamera in rotierenden Bewegungen um das

Objekt geführt. Anhang 15 zeigt hierbei ein Beispiel der Rekonstruktion mit ORB-SLAM2-Native.

7.2.5 Rotationstest

In MR-Szenarien werden HMDs genutzt (Çöltekin et al., 2020, Dörner et al., 2013). Daher ist es naheliegend, dass sich der Operator umsieht, um die Umgebung wahrzunehmen. Dies erfolgt in der Regel in einer rotierenden Bewegung. Im Test soll daher überprüft werden, wie gut der Algorithmus und das System mit puren Rotationen der Kamera umgeht.

7.2.6 Test: RPY-Live

In diesem Testverfahren wird der Neigungswinkel der Kamera und die damit verbundenen Lokalisierungseigenschaften des Systems überprüft. Dies soll die Kopfbewegung simulieren bzw. die Rotationen um die Koordinatenachsen. Konkret wird Roll, Pitch und Yaw in Echtzeit durchgeführt und im Kontrast, zu dem in Kapitel 7.2.1 vorgestellten TUM-RPY-Dataset betrachtet.

7.2.7 Test: XYZ-Live

Hierbei wird ähnlich dem in Kapitel 7.2.6 beschriebenen Testverfahren die Bewegung entlang der Koordinatenachsen getestet. Analog zu dem in Kapitel 7.2.1 vorgestellten TUM-XYZ-Dataset, werden hierbei die natürlichen Translationen in Echtzeit betrachtet.

7.2.8 ROS-Unity-Live-Test

Das System soll gemäß den in Kapitel 4.2 dargestellten Anforderungskriterien und funktionalen Anforderungen, die Lokalisation und Kartierung in Echtzeit ermöglichen. Diesbezüglich wird ein Live-Test durchgeführt, um die Echtzeitfähigkeit zu demonstrieren.

7.3 Ergebnispräsentation und Auswertung

Anhang 16 und 17 zeigen die Ausgaben der jeweiligen Testdurchläufe, insbesondere die Evaluationsergebnisse. Es konnten fünf aufeinanderfolgende Tests zu jeweils einem Dataset durchgeführt werden.

7.3.1 Test TUM-Dataset

Abbildung 39 zeigt die fehlgeschlagenen Initialisierungsversuche pro Test. Die Fehlschläge können den Daten in Anhang 16 und 17 entnommen werden. Sobald die Initialisierung fehlschlägt, muss die Karte neu initialisiert werden. Der Zwischenspeicher wird nach jedem Durchlauf gelehrt. Dies schafft gleiche Ausgangsbedingungen für jeden Durchlauf. Jedoch kann Abbildung 39 entnommen werden, dass der Algorithmus – zumindest bei Bewegungen entlang der Koordinatenachsen – erst bei jedem dritten Versuch eine erfolgreiche Initialisierung erreichen kann. Im Gegensatz dazu, scheint die Initialisierung durch Rotation nicht zuverlässig zu funktionieren. Test 3 mit dem TUM-RPY-Dataset, unterstützt diese These mit 7 vergeblichen Initialisierungsversuchen.

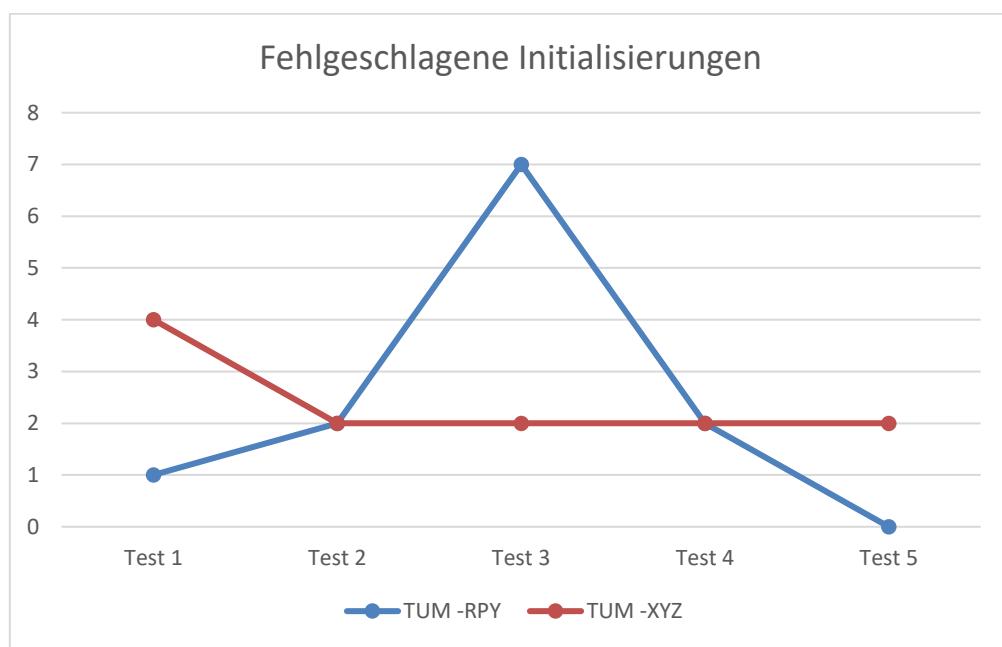


Abbildung 32: Fehlgeschlagene Initialisierungen

7.3.2 Ergebnisse TUM-RPY-Dataset

Die durch das Dataset erzeugten Daten zeigen bis auf Test Nr. 5 in Tabelle 2 ein problematisches Initialisierungsverfahren. Dies ist hinsichtlich MR-Anwendungen, welche auf Echtzeitfähigkeit angewiesen sind als problematisch anzusehen.

Nr.	Failed Initializations	Map Points	Median tracking time	Mean tracking time	Pose Pairs	RMSE (cm)
1	50%	107	0,0153205	0,0159898	36	5,3008
2	66%	105	0,0164173	0,0181199	32	5,5013
3	87%	101	0,016211	0,0195156	22	3,9010
4	87%	115	0,0166927	0,0186322	30	5,1062
5	0%	121	0,0168280	0,0181311	30	3,6771
Mean RMSE						4,69728
Median RMSE						5,1062

Tabelle 2: Ergebnisse TUM-RPY-Dataset

Der Test erzielt einen durchschnittlichen RMSE von 4,68728 cm und im Median 5,1062 cm. Die Tracking time lässt vermuten, dass die Datenverarbeitung schnell genug erfolgt, jedoch konnten zu den verwendeten Datasets aus der Literatur keine Vergleichswerte identifiziert werden. Aus diesem Grund wird lediglich anhand der erhobenen Daten die Echtzeitfähigkeit angenommen. Dies müsste jedoch näher betrachtet werden.

7.3.3 Ergebnisse TUM-XYZ-Dataset

Tabelle 3 zeigt ebenso das unter Kapitel 7.3.2 identifizierte Initialisierungsproblem, allerdings in schwächerer Ausprägung. Hierbei konnte im Vergleich zu den Ergebnissen in Tabelle 2 eine erfolgreichere Initialisierungsphase gezeigt werden. Demnach zeigen die Daten, dass der Algorithmus mit rotierenden Bewegungen der Kamera keine zuverlässige Initialisierung ermöglicht.

Nr.	Failed Initializations	Map Points	Median tracking time	Mean tracking time	Pose Pairs	RMSE (cm)
1	80%	135	0,0188573	0,0211068	32	5,1052
2	66%	104	0,0187696	0,0209025	31	9,4834
3	66%	100	0,0183865	0,0204396	29	1,0138
4	66%	101	0,0200443	0,0219075	30	0,8699
5	66%	100	0,0186890	0,0207981	31	4,7984
Mean RMSE						4,25414
Median RMSE						4,7984

Tabelle 3: Ergebnisse TUM-XYZ-Dataset

Der Test zeigt einen durchschnittlichen RMSE von 4,25414 cm und im Median 4,7984 cm. Im Vergleich der in Tabelle 2 gezeigten Testdaten, fällt die höhere Präzision in Szenarien auf, welche rotationsfreie Bewegungsabläufe simulieren. In folgenden Forschungsbestreben sollte das Verhalten näher untersucht werden, da ORB-Features, wie in Kapitel 5 beschrieben rotationsinvariant sein sollten.

7.3.4 Vergleich der Testergebnisse

Zum Vergleich des RMSE werden die Durchschnittswerte mit den Ergebnissen der Literatur verglichen. Tabelle 4 zeigt die Daten der Literatur im Vergleich zu den im Zuge dieser Arbeit erhobenen Daten.

Autor	Dataset	RMSE (cm)
Liu, Zhang et al. 2016	fr1_xyz	1,05
	fr1_rpy	5,53
Liu, Yulei et al. 2020	fr1_xyz	1,06
	fr1_rpy	-
Mur-Artal et al. 2015	fr1_xyz	0,90
	fr1_rpy	-
Implementierung	fr1_xyz	4,25414
	fr1_rpy	4,69728

Tabelle 4: Vergleichsdaten (Liu, Zhang und Bao, 2016, Liu et al., 2020)

Die Ergebnisse des TUM-Datasets-XYZ zeigen, dass die Implementierung deutlich schlechtere Ergebnisse produziert. Hierbei ist womöglich die Anpassung im Zuge der Implementierung als Ursache zu nennen, da die ausgeführten Autoren mit weniger performanter Hardware deutlich bessere Ergebnisse erzeugen (Liu, Zhang und Bao, 2016, Liu et al., 2020, Mur-Artal und Tardos, 2017).

Unter Betrachtung der Ergebnisse des TUM-Datasets-RPY, konnten bessere Werte beobachtet werden (Liu, Zhang und Bao, 2016). Da jedoch lediglich ein einziger Vergleichswert aus der Literatur entnommen werden konnte, gilt das Ergebnis als wenig aussagekräftig und wir daher nicht weiter berücksichtigt. Daher wird empfohlen hierfür eine eigene Testreihe im Zuge einer separaten Forschungsarbeit durchzuführen.

7.3.5 Entfernungstest

Abbildung 40 zeigt die Ergebnisse des Entfernungstest, welche durch den ORB-SLAM2-Algorithmus erzeugt wurden. Dies zeigt die gefundenen Merkmale bis 1,5m. Im linken Bereich fallen einige Ausreißer auf, die aufgrund des Testaufbaus nicht existieren können, da die Testumgebung ab 1,6m durch ein Objekt blockiert wird. Hierbei ist die Translation der Kamera nach hinten zu sehen. Diese Translation wird durch die pure Rotation der Kamera hervorgerufen. Vermutliche wirkt sich hierbei der Scale Drift aus, der in Kapitel 3.3.2 vorgestellt wird und bei monokularen Systemen zur Deformierung der Trajektorie führt.

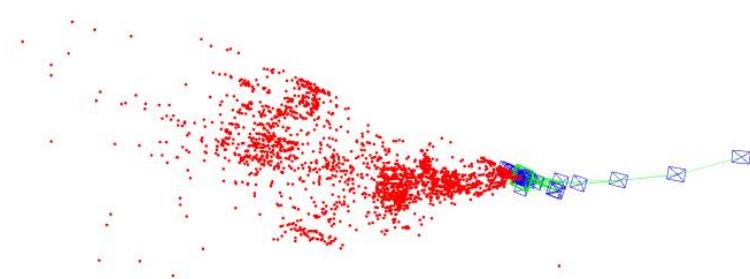


Abbildung 33: ORB-SLAM2-Nativ Entfernungstest

Abbildung 41 zeigt die durch den Entfernungstest erzeugte Point-Cloud von ORB-SLAM2, in Verbindung mit ROS. Die Rekonstruktion weist im Vergleich zu ORB-SLAM2-Nativ klare Strukturen auf. ORB-SLAM2 nutzt Pangolin zur Visualisierung, wohingegen ROS eine eigene Lösung bietet. Hierbei können die Landmarks in unterschiedlichen Farben

visualisiert werden. Dies unterstützt die Darstellung enorm. Allerdings weist der Test nach, dass beide Implementierungen in der Lage sind, Merkmale bis zu einer Entfernung von 1,5m nachzuweisen und demnach zu identifizieren.

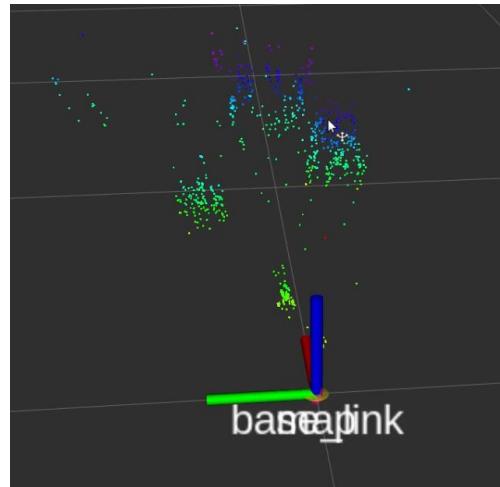


Abbildung 34: ORB-SLAM-ROS Entfernungstest

7.3.6 ORB-SLAM2 – Live Mapping

Abbildung 42 zeigt das Ergebnis des Live-Mappings mit ORB-SLAM2-Native. Die Strukturen der originalen Szene können gut erkannt werden. Die Trajektorie wird hierbei nicht analysiert.

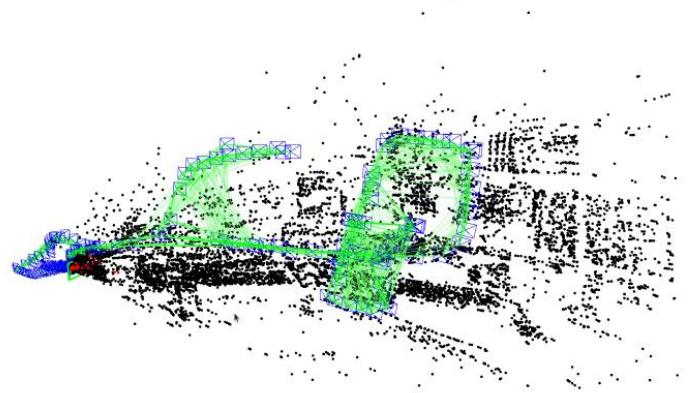


Abbildung 35: Ergebnis ORB-SLAM2-Nativ Desk-Mapping

Abbildung 43 zeigt das Ergebnis von ORB-SLAM-ROS. Hierbei können klare Strukturen nachgewiesen werden, sowie Objekte visuell identifiziert werden

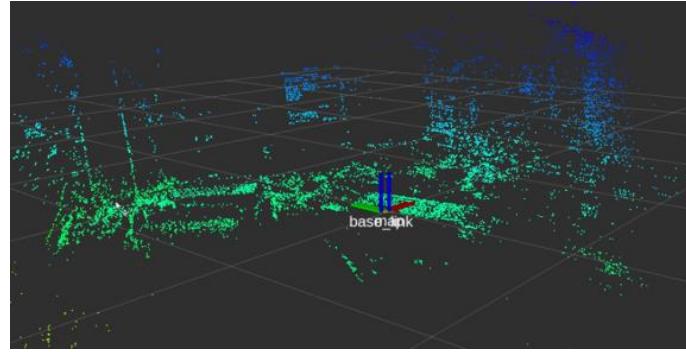


Abbildung 36: ORB-SLAM-Nativ Desk-Mapping

7.3.7 ORB-SLAM2 - Rekonstruktion

Zur Rekonstruktion wurde dem Algorithmus ein geeignetes Objekt präsentiert. Beide Varianten rekonstruieren das Objekte. Abbildung 44 zeigt ORB-SLAM2-Nativ. Hierbei fallen wiederum die bereits erkannten „Ausreißer“ auf.

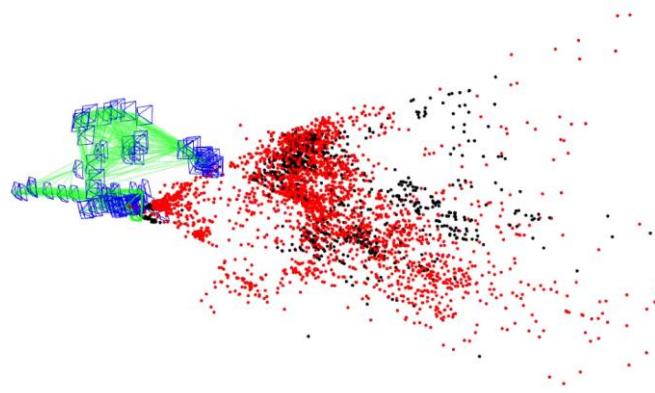


Abbildung 37: ORB-SLAM-Nativ Rekonstruktion

Abbildung 45 zeigt ORB-SLAM in Verbindung mit ROS. Hierbei werden keine „Ausreißer“ visualisiert und die Struktur des Objekts besser nachgebildet. Im Zuge dieser Arbeit konnte der Ursprung der „Ausreißer“ nicht geklärt werden. Es wird empfohlen den Ursprung zu untersuchen und zu überprüfen ob hierbei ein Fehler bzgl. der Visualisierung oder Implementierung vorliegt.

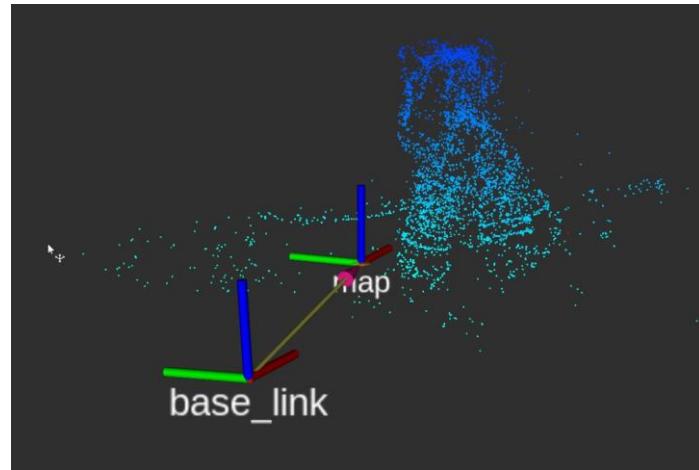


Abbildung 38: ORB-SLAM-ROS Rekonstruktion

7.3.8 ORB-SLAM2 - Rotationstest

Die pure Rotation zeigt in Abbildung 46 ähnliche Auswirkungen wie unter Kapitel 7.3.4 durch den Test der Entfernungswahrnehmung festgestellt wurde. Hierbei ist die Deformation der gesamten Rekonstruktion, sowie der Trajektorie deutlich zu erkennen. Demnach eignen sich die Daten nicht mehr für eine Rekonstruktion und eine Lokalisation sollte ebenso nicht mehr valide durchgeführt werden können.

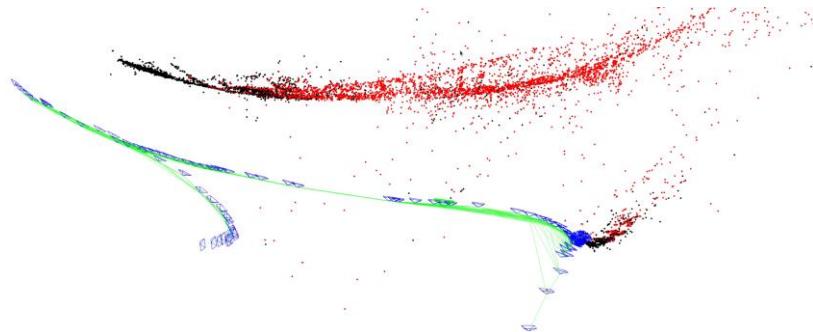


Abbildung 39: ORB-SLAM2-Nativ Rotation

Abbildung 47 zeigt die Visualisierung der Ergebnisse des Rotationstest mit der Implementierung von ORB-SLAM2 unter ROS. Hierbei funktioniert die pure Rotation und wird durch den Algorithmus korrekt interpretiert. Ebenso kann die Rekonstruktion einer Rotation zugeordnet werden. Dies impliziert, dass ORB-SLAM2-Nativ u.U. nicht korrekt arbeitet. Diesbezüglich müsste in einer separaten Forschungsleistung untersucht werden, wie

sich die für die Implementierung durchführten Änderungen an dem Algorithmus auf die Arbeitsweise von ORB-SLAM2-Nativ auswirken.

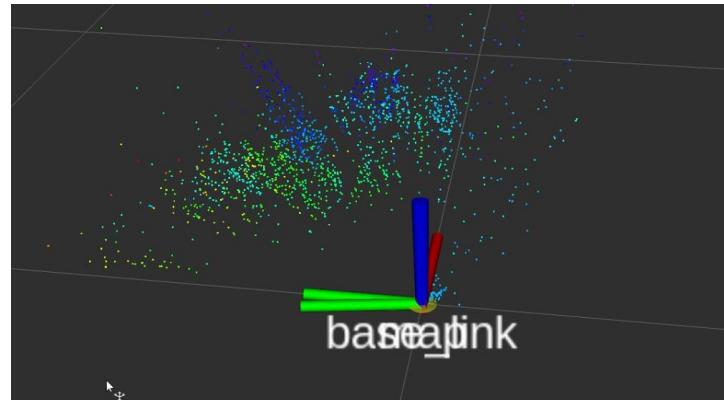


Abbildung 40: ORB-SLAM2-ROS Rotation

7.3.9 ORB-SLAM2 - RPY-Live

Der Test zeigt, dass ORB-SLAM2-Nativ die Rotationen um die Koordinatenachsen nicht korrekt interpretiert werden. Hierbei kann festgestellt werden, dass der Algorithmus Rotationen häufig als Translationen interpretiert (Abbildung 48). Unklar ist, weshalb dies geschieht. Da ORB-SLAM für die Robotik entwickelt wird (Mur-Artal und Tardos, 2017), könnten diese Bewegungsmuster nicht von Relevanz für den eigentlichen Zweck des Algorithmus sein. Dementsprechend wäre eine eingehende Analyse des Verhaltens und eine Weiterentwicklung zweckmäßig.

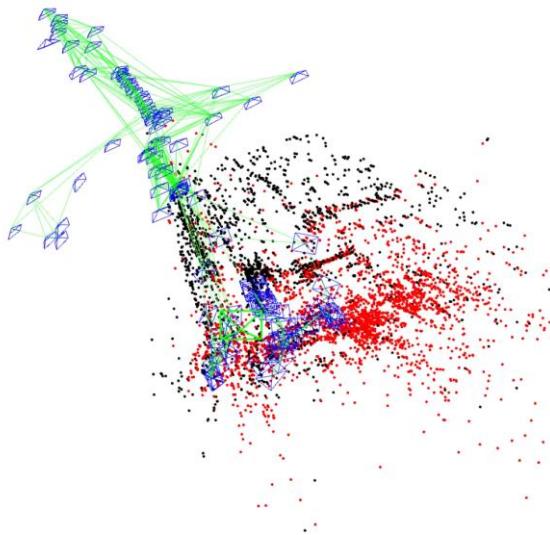


Abbildung 41: ORB-SLAM-Nativ RPY-Live

7.3.10 ORB-SLAM2 - XYZ-Live

Abbildung 49 zeigt die Ergebnisse von ORB-SLAM2-Nativ in der Live-Simulation der Bewegungsmuster entlang der Koordinatenachsen. Hierbei kann durch den Verlauf der Trajektorie der Scale Drift nachgewiesen werden.

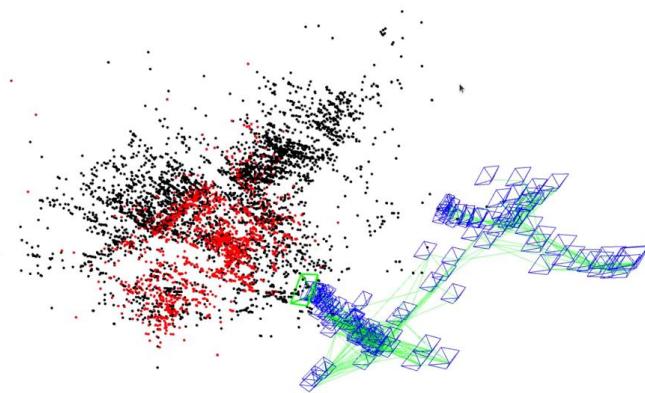


Abbildung 42: ORB-SLAM-Native XYZ-Live

7.3.11 Zusammenfassung der Ergebnisse von ORB-SLAM-Nativ

Der Test des Algorithmus durch die TUM-Datasets zeigen die schwierige Initialisierungsphase des Algorithmus. Rotationen reichen dem Algorithmus nicht um eine

zuverlässige Initialisierung der Karte zu erreichen. Weiter wurde demonstriert, dass der Algorithmus Merkmale bis zu Entfernungen bis 1,5m identifizieren kann. Im Zuge der Rekonstruktionstest wurde gezeigt, dass der Algorithmus zwar ausreichende Merkmale identifizieren kann, jedoch die Visualisierung nur schwer eine Identifikation der konkreten Objekte ermöglicht. Da die spärliche Rekonstruktion lediglich zur Lokalisation verwendet werden soll, wird dies als ausreichend eingestuft. Die Lokalisationseigenschaften der Kamera funktionieren entlang der Koordinatenachsen zuverlässig, jedoch werden Rotationen nicht präzise registriert.

7.3.12 Zusammenfassung der Ergebnisse von ORB-SLAM-ROS

Die Testergebnisse von ORB-SLAM in Verbindung mit ROS zeigen bessere Ergebnisse als ORB-SLAM2-Nativ. Dies kann durch den manuellen Eingriff in den Source-Code des Algorithmus für die Native-Version begründet werden. ORB-SLAM2 zeigt in Kombination mit ROS eine bessere Rekonstruktion, was u.U. durch die in ROS integrierte Visualisierungslösung begründet werden kann. Es konnte zudem eine bessere Registrierung der Kamerarotation festgestellt werden. In Kapitel 7.4 wird die in ROS integrierte Lösung in Kombination mit der Unity Engine getestet.

7.3.13 AR-DEMO

Das AR-Demo lässt sich nur schwer initialisieren. Abbildung 50 zeigt hierbei die erfolgreiche Verankerung eines virtuellen Objekts in der physischen Realität. Die Oberflächenerkennung durch RANSAC funktioniert hierbei jedoch nicht wie von Mur-Artal vorgestellt. Demnach konnte kein Objekt absolut planar platziert werden.

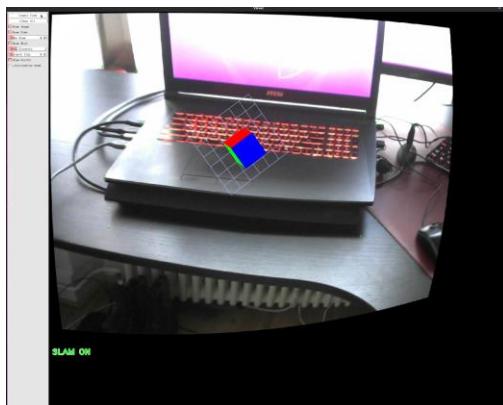


Abbildung 43: Verankerung des virtuellen Objekts



Abbildung 44: Änderung der Perspektive

Abbildung 51 zeigt jedoch, dass die Position des Objekt bei der Bewegung der Kamera relativ konstant bleibt.

7.3.14 ROS-Unity

Zunächst wurde das Objekt „Pferd“ durch den Algorithmus rekonstruiert, um einerseits die Initialisierung des Algorithmus durchzuführen und andererseits die Karte zu initialisierten. Abbildung 52 zeigt hierbei die an Unity zu übertragende Karte. Das Ziel ist es die geschätzte Kamerapose in Echtzeit an Unity zu übergeben, sowie die bis zum Start des Unity-Projekts initialisierten Map-Points der Karte als Meshes in die Szene zu projizieren.

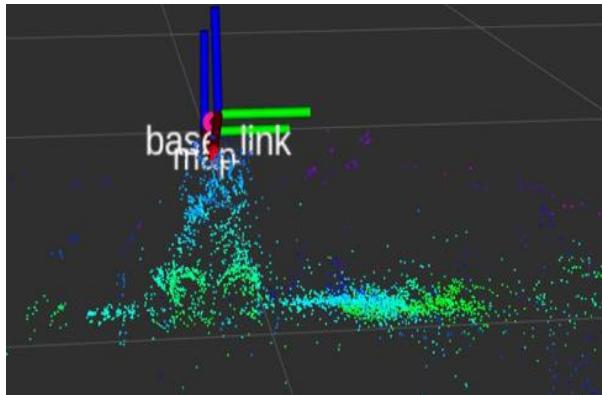


Abbildung 45: Map-Points der initialisierten Karte

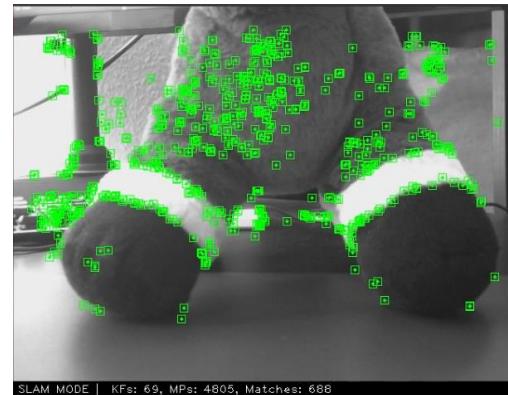


Abbildung 46: Objektscann

Dazu wird das Demonstrationsobjekt abgescannt, wie in Abbildung 53 zeigt wird. Sobald der Algorithmus erfolgreich initialisiert und die Karte erstellt wurde, wird das Unity-Projekt gestartet. Hierbei konnten die Kartendaten erfolgreich übertragen werden. Abbildung 54 zeigt die an die Unity Engine übertragenen Map-Points. In Gelb wird hierbei ein Orientierungspunkt gezeigt, um im Applikationsbetrieb die Skalierung und Entfernung abschätzen zu können.

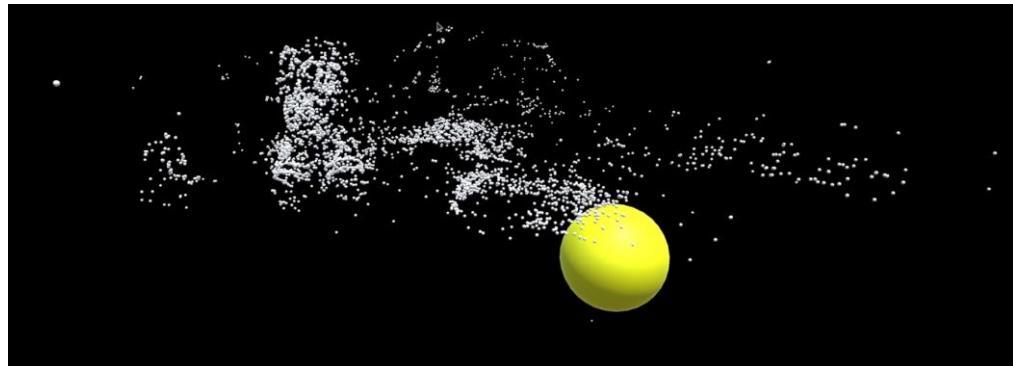


Abbildung 47: Map-points in der Unity Engine

Abbildung 55 zeigt die Kameraperspektive nach erfolgreicher Translation und Rotation der Kamera, insbesondere die Verlagerung der Perspektive. Hierbei kann am gelben Objekt die regelrechte Skalierung und Translation beobachtet werden.

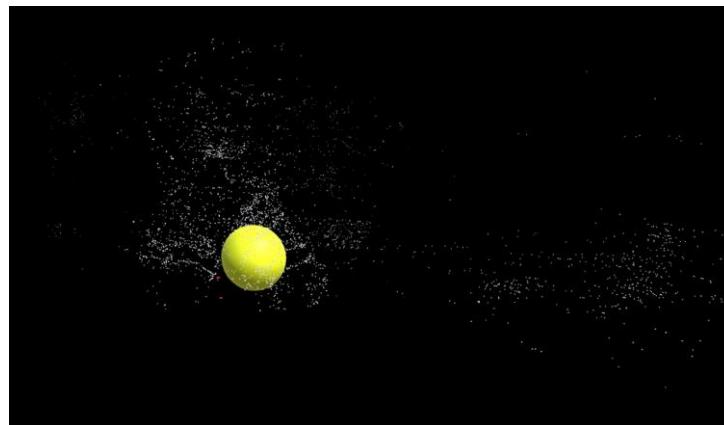


Abbildung 48: Szene nach Translation und Rotation der Kamera

7.4 Zusammenfassung

In diesem Kapitel wurden die im Zuge dieser Forschungsarbeit entwickelten Komponenten getestet. Es wurde der Algorithmus vom System isoliert und mit den für SLAM-Verfahren üblichen Datasets getestet. Weiter wurden die Eigenschaften des Algorithmus, respektive des Gesamtsystems zur Merkmalsextraktion in Entfernung bis zu 1,5 m überprüft. Zudem wurden die Kartierungs- und Rekonstruktionseigenschaften beider Varianten untersucht und verschiedene Tests in Echtzeit durchgeführt. Zudem wurden die AR-Eigenschaften in simplen AR-Demo getestet. In Kapitel 8 werden die Forschungsleistung evaluiert.

8 Evaluation

Die initial zu dieser Forschungsarbeit aufgestellte Forschungsfrage, konnte beantwortet werden, indem der hierfür entwickelte Prototyp durch die Testreihen in Kapitel 7 validiert wurde. Das SLAM-Problem, lässt sich demnach – auch ohne die HoloLens und dem damit verbundenen Ökosystem Windows-MR – lösen. Dazu konnte ORB-SLAM 2 als valide Lösungsstrategie für das SLAM-Problem identifiziert werden. Zudem konnte die Integration des ROS in Unity demonstriert werden. In Kapitel 7 wurde gezeigt, dass die durch den SLAM-Algorithmus erzeugten Daten, in der Unity Engine verwendet werden können. Dadurch kann das gesamte Leistungsspektrum der Unity Engine, mittels den durch den Algorithmus generierten Daten verwendet werden, um MR-Applikationen auch ohne das Windows-Ökosystem zu realisieren. Somit wurde demonstriert, dass nicht nur ohne die Hardwarekomponente HoloLens, sondern auch ohne die softwareseitige Unterstützung durch das Betriebssystem Windows und Windows Mixed Reality, die Lösung des SLAM-Problems für MR möglich ist.

Zusammenfassend demonstrieren die Testergebnisse in Kapitel 7, dass ORB-SLAM 2 zwar eine Lösung für das SLAM-Problem für MR darstellt, diese jedoch nicht optimal ist. Es konnten erhebliche Schwierigkeiten während der Initialisierungsphase gezeigt werden. Dies ist für MR-Szenarien nicht akzeptabel, da sich die Initialisierungsphase zeitlich nicht bestimmen lässt. Dies würde sich negativ auf die Immersion auswirken, da während der Laufzeit des Algorithmus das Tracking bei schnellen Bewegungen – vor allem bei Rotationen – fehlschlägt. Kann der Nutzer den letzten erfolgreich observierten Szenenabschnitt wieder finden, fährt der Algorithmus an dieser Stelle fort. Sofern das nicht der Fall ist, müsste der Algorithmus neu initialisiert werden. Für MR-Anwendungen wäre er Algorithmus in aktueller Version noch nicht praktikabel und müsste weiterentwickelt werden. Hierbei müsste das Tracking unabhängig von der generierten Karte an jeder Stelle erneut initialisiert werden können. Konkret müssten beim Verlust des Trackings, vorhandene lokale Merkmale zur Initialisierung eines global-unabhängigen Kartenabschnitts zwischengespeichert werden. Anschließend könnte mittels Loop-Closing versucht werden, die Kartenfragmente zusammenzufügen. Dies könnte vermeiden, bereits bekannte Teilabschnitte unbedingt wieder finden zu müssen, um das Tracking fortzusetzen.

Nichtsdestotrotz ist die Lösungsstrategie in aktueller Fassung für Forschungszwecke verwertbar.

Bezüglich der Sensorkonfiguration, kann durch die Verwendung eines Stereosystems der in Kapitel 3 vorgestellte und in Kapitel 7 nachgewiesene Scale-Drift minimiert werden (Mur-Artal und Tardos, 2017). Die Tests zeigen, dass ORB-SLAM 2 in der monokularen Ausführung viel Drift ansammelt. Durch Loop-Closing, soll der angesammelte Drift jedoch reduziert werden können (Mur-Artal und Tardos, 2017). Dies konnte in des Tests jedoch nicht gezeigt werden, da die Strukturen des Innenraums der Testumgebung kein Loop-Closing ermöglichen. In einer weiteren Forschungsleistung wird ein detailliertes Testverfahren für ORB-SLAM 2 in monokularer Ausführung vorgeschlagen. Hierbei soll explizit überprüft werden, in welchem Umfang sich der Scale-Drift durch Loop-Closing reduzieren lässt. Im Zuge dessen wäre es sinnvoll, den Sensor durch einen qualitativ hochwertigeren auszutauschen und diesbezüglich weitere Recherchen anzustellen bzw. Tests durchzuführen.

Die für diese Arbeit durchgeführte Kamerakalibrierung wurde durch OpenCV anhand Zhangs Methode durchgeführt (Zhang, 2000, Kaehler und Bradski, 2015). Hierbei müsste überprüft werden, inwiefern die dadurch erzeugten Kalibrierungsdaten präzisiert werden können. Dies müsste sich positiv auf den in Kapitel 7 erzeugten RMSE auswirken. Vermutlich würde sich bereits ein besseres Druckerzeugnis des Schachbretts mit schärferen Kanten positiv auf die Kalibrierungsdaten auswirken (Kaehler und Bradski, 2015). Ebenso konnte keine eindeutige Anzahl der benötigten Bilder ermittelt werden. Daher wäre es sinnvoll, dies im Zuge der Kalibrierungstestes zu überprüfen.

In Kapitel 5 wurden Alternativen zu ORB vorgestellt (Ruble et al., 2011). Hierbei werden SIFT und SURF genannt. Beide fordern viel Rechenleistung, sodass eine Ausführung in Echtzeit nur auf der Grafikkarte realisierbar sei (Chien et al., 2016). Der Vergleich beider Methoden mit ORB – auf der vorgestellten Hardware – könnte von Interesse sein. Dies könnte das festgestellte Rotationsproblem von ORB abmildern, indem die präziseren Daten von SIFT bzw. SURF verwendet werden.

Das in Kapitel 7 festgestellte Rotationsproblem, konnte in der Testreihe ORB-SLAM-ROS und ORB-SLAM in Unity nur noch abgeschwächt nachgewiesen werden. Nichtsdestotrotz sollte das Problemverhalten weiter analysiert werden. Diesbezüglich eignet

sich die in Kapitel 3.3.1 vorgestellte IMU (Ahmad et al., 2013). Hierfür müsste die Kombination aus Kameradaten und IMU-Daten robustere Ergebnisse produzieren.

Kapitel 7 zeigt zudem, dass die Oberflächenerkennung mittels RANSAC und die Identifikation geeigneter Projektionsflächen nicht orientiert funktioniert. Es können zwar aus den Point-Cloud-Daten entsprechende Ankerflächen identifiziert werden, jedoch nicht präzise orientiert. Hierbei können sich andere Herangehensweisen mittel RANSAC eignen wie Förster et al. und Kim et al. demonstrieren (Ying Yang und Förstner, 2010, Kim et al., 2017). Hierbei könnten beide Ansätze näher untersucht und miteinander verglichen werden, um planare Projektionsflächen lagerichtig zu erfassen.

Zur Plane-Detection werden vermehrt Ansätze aus der Künstlichen Intelligenz eingesetzt. Hierbei werden aktuell Echtzeitansätze realisiert und diskutiert. Dazu könnten entsprechende Ansätze recherchiert und ausgewertet werden, um die Echtzeiteigenschaften zu untersuchen. Lui et. al. demonstrieren, dass selbst ein einziges RGB-Bild ausreiche, um durch die Verwendung eines entsprechend trainierte Neuronalen Netzes valide planare Oberflächen zu identifizieren (Liu et al., 2018a). Im Kontext dieser Arbeit, könnte das Verfahren pro deklariertem Keyframe eingesetzt werden, um in parallel in einem separaten Thread zu arbeiten. Dadurch lassen sich selbst anspruchsvolle Anwendungen parallelisieren. Aufgrund der in Kapitel 4 vorgestellten Hardware, könnte die performante Abarbeitung Mehrer Threads demonstriert werden. Dadurch wäre die Anwendung von Chens Herangehensweise u.U. mit dem Anspruch der Echtzeitfähigkeit zu vereinbaren, insbesondere für die Projektion geeignete Oberflächen identifizierbar.

Die Implementierung kostet durch die Build-Prozesse viel Zeit. Hierbei würde sich für anschließende Forschung die Verwendung von Python als interpretierte Skriptsprache aufgrund der schnelleren Ausführung der Skripts eignen. Sowohl OpenCV als auch ROS können mit Python umgehen (Kaehler und Bradski, 2015, Koubaa, 2016). Hierbei stellt sich nur die Frage, inwiefern ORB-SLAM 2 zu Python portiert werden kann.

Die HoloLens bringt als autarkes HMD eigene Hardware und ein Windows-Betriebssystem mit (Evans et al., 2017). Diesbezüglich macht es Sinn, ORB-SLAM auf Einplatinenrechnern zu erproben, um zu überprüfen, wie gut der Algorithmus sich für den autarken betrieb eignet.

9 Fazit und Ausblick

Diese Arbeit behandelt die Frage, ob Mixed Reality auch ohne die Microsoft HoloLens realisierbar wäre. Konkret soll das SLAM-Problem mit einem dem Stand der Technik entsprechenden Algorithmus gelöst werden. Dafür konnte ORB-SLAM-2 identifiziert werden (Mur-Artal und Tardos, 2017).

Weiter wird die Integration des Algorithmus in die Unity Engine umgesetzt, um auf dieser Basis die Entwicklung von MR-Anwendungen zu ermöglichen. Die in Kapitel 7 durchgeführten Testverfahren weisen die Lösung des SLAM-Problems durch den ORB-SLAM-2-Algorithmus nach. Zudem wird in einer Demonstration die Anwendung von Augmented Reality mittels des RANSAC-Algorithmus demonstriert. Außerdem wird die Integration des ORB-SLAM2-Algorithmus, unter Verwendung von ROS als Middleware, demonstriert. Im gleichen Zuge wird so der Transfer der Ergebnisse aus der Lösung des SLAM-Problems an Unity realisiert. Dadurch lässt sich nachweisen, dass sowohl AR als auch VR, sowie MR in Verbindung mit dem Algorithmus realisiert werden können. Abschließen lässt sich die Forschungsfrage beantworten und demonstrieren, dass das SLAM-Problem auch ohne HoloLens für MR lösbar ist.

Im Zuge der Evaluations- und Testverfahren konnte eine schwierige Initialisierungsphase des ORB-SLAM-2-Algorithmus nachgewiesen werden. Die Evaluation ergab, dass sich hierbei vermutlich der Einsatz von Stereosensoren positiv auf die Initialisierung auswirken könnte, da sich dadurch die mit monokularen Verfahren eingehoregenden Nachteile ausgeglichen werden können. Die Ursache der unsauberen Initialisierung konnte in dieser Arbeit nicht geklärt werden. Um eine Initialisierung beim Systemstart gewährleisten zu können, ist mehr Forschungsbedarf erforderlich.

Weiter konnte nachgewiesen werden, dass ORB-SLAM-2 nicht sehr gut mit puren Rotationen umgehen kann. Diese produzieren entweder unbrauchbare Daten oder werden als Translationen interpretiert. Im Zuge dieser Arbeit konnte die Ursache nicht geklärt werden. Jedoch ließen sich Rotationen in Verbindung mit ROS regelrecht verarbeiten. Im Zuge dieser Arbeit konnte jedoch hierfür keine Begründung ermittelt werden, was weiteren Forschungsbedarf impliziert.

Fazit und Ausblick

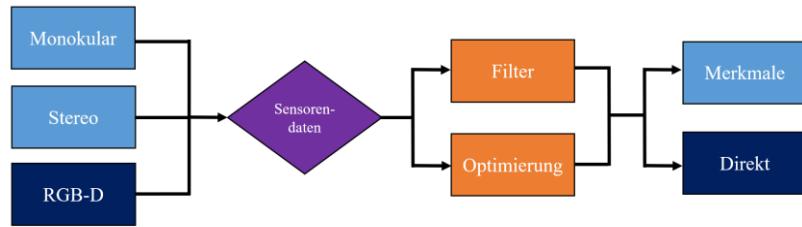
Im Zuge dieser Arbeit konnte keine mit der Unity Engine kompatible AR/MR-Rendering-Pipeline identifiziert werden, welche eine direkte Integration eines externen SLAM-Verfahrens bietet. Daher konnte das erstellte System weder bzgl. AR noch MR getestet werden. Daher ist beabsichtigt, für die nächste Forschungsarbeit bzw. im Zuge des geplanten Master-Studiums, diese Thematik weiter zu verfolgen. Die Integration eines SLAM-Algorithmus soll direkt in der Unity Engine als Native-Plugin erfolgen und die hierfür benötigte Rendering- Pipeline soll entwickelt werden.

Die Unity Engine bietet die Möglichkeit sog. Native Plugins zu integrieren. Hierbei handelt es sich um statische oder dynamische Bibliotheken in C++. Diese können in der Unity Engine als native Plugins genutzt werden (Hocking, 2015). Für eine anschließende Forschungsarbeit würde es sich daher anbieten einen modernen für MR-Szenarien geeigneten Lösungsstrategie für SLAM als statische oder dynamische Bibliothek in Unity zu integrieren. Auf dieser Grundlage könnte eine MR-Applikation realisiert werden. Diesbezügliche könnten mehrere Varianten einer Lösungsstrategie integriert und verglichen werden. Dazu müsste eine eigene Rendering-Pipeline auf Grundlage der Universal-Rendering-Pipeline für Unity entwickelt werden (Hocking, 2015), da sonst entsprechenden Limitierungen durch die Verwendete Lösung auftreten können. In dieser Arbeit wurde auf die Verwendung zusätzlicher AR/MR-Tools verzichtet, um die Abhängigkeit zur HoloLens und Microsoft nicht durch eine neue Abhängigkeit zu ersetzen. Da eine externe SLAM-Lösung verwendet werden soll, wäre es zweckmäßig die passende Rendering-Pipeline für den Algorithmus zu entwickeln.

Im Zuge der Literaturrecherche zu dieser Arbeit ist aufgefallen, dass der Großteil der Lösungsstrategien entweder aus dem Bereich der Robotik kommt oder explizit für Anwendungen der Robotik konzipiert werden. Für Authoring-Tools, wie die Unity Engine, stehen Softwarelösungen zur Verfügung, diese legen die internen Abläufe und konkrete Algorithmen nur sehr selten frei. Dies impliziert weiteren Forschungsbedarf, um das SLAM-Problem gezielt für MR-Anwendungen lösen zu können und für diesen Anwendungsfall geeignete Lösungsstrategien zu entwickeln. Daher werden in naher Zukunft weitere Forschungsvorhaben erwartet, die das SLAM-Problem konkret für MR lösen. Abschließend konnte jedoch ein System präsentiert werden, was die modulare Implementierung ROS-kompatibler SLAM-Lösungen bietet, die über die Unity Engine für MR-Anwendungen verwendet werden kann.

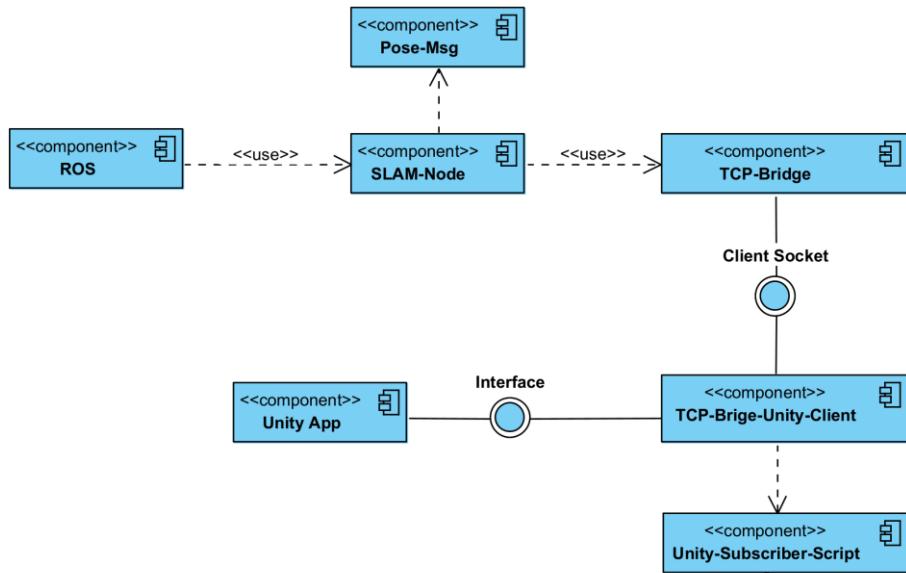
Anhang

Anhang 1 - Taxonomie moderner vSLAM-Verfahren

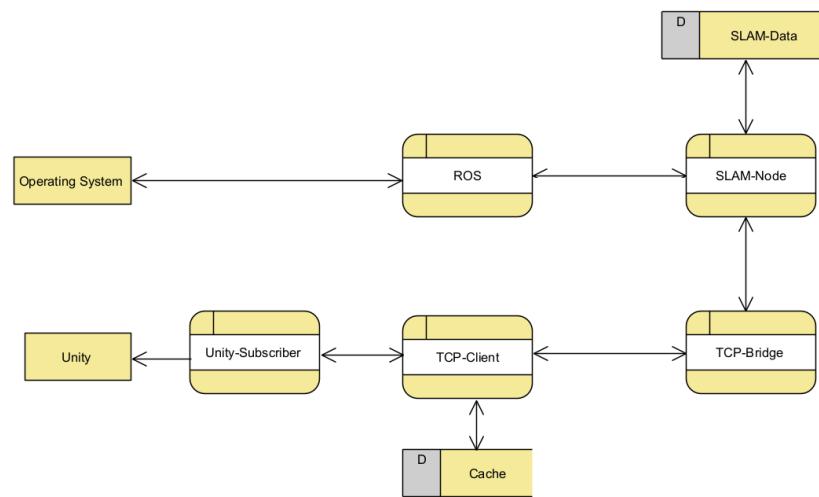


Taxonomie moderner vSLAM-Verfahren in Anlehnung an (Huang Baichuan, Zhao und Liu, 2019, Sualeh und Kim, 2019, Chen et al., 2018b).

Anhang 2 - Komponentendiagramm



Anhang 3 - Data-Flow-Diagramm



Anhang 4 - ORB-Feature Extraktor

```
#include <opencv2/opencv.hpp>
#include <iostream>

int main() {
    cv::Mat frame;
    cv::namedWindow("ORB-Features");
    cv::VideoCapture cap(0);
    std::vector<cv::KeyPoint> keypoints;
    cv::Ptr<cv::FeatureDetector> detector;
    detector = cv::ORB::create();
    cv::Mat features;

    if (!cap.isOpened()) {
        std::cout << " No data from video" << std::endl;
        return -1;
    }

    while (true) {
        cap >> frame;
        detector->detect(frame, keypoints);

        if (frame.empty() || keypoints.empty()) {
            continue;
        }

        cv::drawKeypoints(frame, keypoints, features);
        cv::imshow("raw", frame);
        cv::imshow("ORB-Features", features);
        char key = (char)cv::waitKey(25); // Waitkey == ESC | 25
millsec image proc time
        if (key == 27) {
            break;
        }
    }
    cap.release();
    return 0;
}
```

Anhang 5 – Verwendete Software-Versionen

Softwareprodukt	Version	Abhängigkeit	Version
Unity Engine	2020.3.15f2	Ubuntu	20.04.2
		.Net Core SDK	5.0, 3.1, 2.1
ROS	Noetic Ninjemys	Ubuntu	20.04
ORB-SLAM2	Commit: f2e6f51	Ubuntu	14.04-18.04
		C++	2011 Standard
		Pangolin	0.6
		OpenCV	2.4.3 oder 3
		Eigen 3	3.1.0
		DBow2	Inkludiert
		g2o	Inkludiert
		Linux Build Essentials	In OS inkludiert
ROS-ORBSLAM2	Commit: e933256	ROS	Kinetic, Noetic
		Ubuntu	16.04, 18.04
		Eigen 3	3.1.0
Unity-Robotics-Hub: TCP Bridge	Commit: 431C454	python-pip	aktuell
		ros-noetic-robot-state-publisher	aktuell
		ros-noetic-moveit	aktuell
		ros-noetic-rosbridge-suite	aktuell
		ros-noetic-joy	aktuell
		ros-noetic-ros-control	aktuell
		ros-noetic-ros-controllers	aktuell
		pip3 rospkg	aktuell
		pip3 jsonpickle	aktuell

(Lovegorve, 2021, Epic Games, 2021, Hocking, 2015, Hussein, Garcia und Olaverri-Monreal, 2018, Koubaa, 2016, Open Robotics, 3. September 2021)

Anhang 6 - Kamerakalibrierung

```
#include <opencv2/opencv.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <stdio.h>
#include <iostream>

// Defining the dimensions of checkerboard
int CHECKERBOARD[2]{ 6,9 };
// Quelle: https://learnopencv.com/camera-calibration-using-opencv/
// Kompatibilität zu OpenCV 4 hergestellt
// Pfade bearbeitet
int main()
{
    // Creating vector to store vectors of 3D points for each checkerboard
    // image
    std::vector<std::vector<cv::Point3f> > objpoints;

    // Creating vector to store vectors of 2D points for each checkerboard
    // image
    std::vector<std::vector<cv::Point2f> > imgpoints;

    // Defining the world coordinates for 3D points
    std::vector<cv::Point3f> objp;
    for (int i{ 0 }; i < CHECKERBOARD[1]; i++)
    {
        for (int j{ 0 }; j < CHECKERBOARD[0]; j++)
            objp.push_back(cv::Point3f(j, i, 0));
    }

    // Extracting path of individual image stored in a given directory
    std::vector<cv::String> images;
    // Path of the folder containing checkerboard images
    std::string path = "./Resources/*.png";

    cv::glob(path, images);

    cv::Mat frame, gray;
    // vector to store the pixel coordinates of detected checker board
    // corners
    std::vector<cv::Point2f> corner_pts;
    bool success;
```

```
// Looping over all the images in the directory
for (int i{ 0 }; i < images.size(); i++)
{
    frame = cv::imread(images[i]);
    cv::cvtColor(frame, gray, cv::COLOR_BGR2GRAY);

    // Finding checker board corners
    // If desired number of corners are found in the image then success
= true
    //success = cv::findChessboardCorners(gray,
cv::Size(CHECKERBOARD[0], CHECKERBOARD[1]), corner_pts,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FAST_CHECK |
CV_CALIB_CB_NORMALIZE_IMAGE);
    success = cv::findChessboardCorners(gray, cv::Size(CHECKERBOARD[0],
CHECKERBOARD[1]), corner_pts, cv::CALIB_CB_ADAPTIVE_THRESH |
cv::CALIB_CB_FAST_CHECK | cv::CALIB_CB_NORMALIZE_IMAGE);

    /*
     * If desired number of corner are detected,
     * we refine the pixel coordinates and display
     * them on the images of checker board
    */
    if (success)
    {
        //cv::TermCriteria criteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER,
30, 0.001);
        cv::TermCriteria criteria(cv::TermCriteria::EPS |
cv::TermCriteria::MAX_ITER, 30, 0.001);

        // refining pixel coordinates for given 2d points.
        cv::cornerSubPix(gray, corner_pts, cv::Size(11, 11), cv::Size(-
1, -1), criteria);

        // Displaying the detected corner points on the checker board
        cv::drawChessboardCorners(frame, cv::Size(CHECKERBOARD[0],
CHECKERBOARD[1]), corner_pts, success);

        objpoints.push_back(objp);
        imgpoints.push_back(corner_pts);
    }

    cv::imshow("Image", frame);
    cv::waitKey(2);
}
```

Anhang

```
cv::destroyAllWindows();
std::cout << "Calibrating . . ." << std::endl;
std::cout << "Please wait a few minutes" << std::endl;
cv::Mat cameraMatrix, distCoeffs, R, T;

/*
 * Performing camera calibration by
 * passing the value of known 3D points (objpoints)
 * and corresponding pixel coordinates of the
 * detected corners (imgpoints)
 */
cv::calibrateCamera(objpoints, imgpoints, cv::Size(gray.rows,
gray.cols), cameraMatrix, distCoeffs, R, T);
std::cout << "Beste" << std::endl;

std::cout << "cameraMatrix : " << cameraMatrix << std::endl;
std::cout << "distCoeffs : " << distCoeffs << std::endl;
std::cout << "Rotation vector : " << R << std::endl;
std::cout << "Translation vector : " << T << std::endl;

return 0;
}
```

Anhang 7 – Ergebnisse Kamerakalibrierung

```
% YAML:1.0

#-----
#-----#
# Camera Parameters.Adjust them!
#-----
#-----#


# Camera calibration and distortion parameters(OpenCV)
Camera.fx: 835.75757
Camera.fy : 831.985
Camera.cx : 325.31354
Camera.cy : 224.24859

Camera.k1 : 0.086811
Camera.k2 : 0.147352
Camera.p1 : -0.008621
Camera.p2 : 0.000306
Camera.k3 : 0.000000

# Camera frames per second
Camera.fps: 30.0

# Color order of the images(0: BGR, 1 : RGB.It is ignored if images are
# grayscale)
Camera.RGB: 1

#-----
#-----#
# ORB Parameters
#-----
#-----#


# ORB Extractor : Number of features per image
ORBextractor.nFeatures : 2000

# ORB Extractor : Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor : 1.2

# ORB Extractor : Number of levels in the scale pyramid
ORBextractor.nLevels : 8
```

Anhang

```
# ORB Extractor : Fast threshold
# Image is divided in a grid. At each cell FAST are extracted imposing a
minimum response.
# Firstly we impose initThFAST. If no corners are detected we impose a lower
value minThFAST
# You can lower these values if your images have low contrast
ORBextractor.initThFAST: 20
ORBextractor.minThFAST : 7

#-----
#-----#
# Viewer Parameters
#-----
#-----
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth : 1
Viewer.GraphLineWidth : 0.9
Viewer.PointSize : 5
Viewer.CameraSize : 0.08
Viewer.CameraLineWidth : 3
Viewer.ViewpointX : 0
Viewer.ViewpointY : -0.7
Viewer.ViewpointZ : -1.8
Viewer.ViewpointF : 500
```

Anhang 8 - ORB-SLAM2 – MonoAR.cc

```
/**  
* This file is part of ORB-SLAM2.  
*  
* Copyright (C) 2014-2016 Raúl Mur-Artal <raulmur at unizar dot es>  
(University of Zaragoza)  
* For more information see <https://github.com/raulmur/ORB_SLAM2>  
*  
* ORB-SLAM2 is free software: you can redistribute it and/or modify  
* it under the terms of the GNU General Public License as published by  
* the Free Software Foundation, either version 3 of the License, or  
* (at your option) any later version.  
*  
* ORB-SLAM2 is distributed in the hope that it will be useful,  
* but WITHOUT ANY WARRANTY; without even the implied warranty of  
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
* GNU General Public License for more details.  
*  
* You should have received a copy of the GNU General Public License  
* along with ORB-SLAM2. If not, see <http://www.gnu.org/licenses/>.  
*/
```

```
* This file is part of ORB-SLAM2.
#include <iostream>
#include <algorithm>
#include <fstream>
#include <chrono>

// #include<ros/ros.h>
// #include <cv_bridge/cv_bridge.h>

#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include "../../include/System.h"

#include "ViewerAR.h"

using namespace std;

ORB_SLAM2::ViewerAR viewerAR;
bool bRGB = true;

// ***** LAKUS
const string pathVocabularyFile = "../../Vocabulary/ORBvoc.txt";
const string pathSettingsFile = "left1.yaml";
int camIndex = 0;
// ++++++ LAKUS

cv::Mat K;
cv::Mat DistCoef;

int main(int argc, char* argv[])
{
    ORB_SLAM2::System* mpSLAM;
    char* a = argv[1];
    std::cout << "argv[1]" << argv[1] << std::endl;
    camIndex = atoi(a);
    std::cout << "Cam Index: " << camIndex << std::endl;
    if (argc != 2)
    {
        cerr << endl << "No Camera ID was specified please Start AR-Demo
like this: ./MonoAR <DeviceID>" << endl;
        return 1;
    }
    // Create SLAM system. It initializes all system threads and gets ready
    to process frames.
```

```
// ***** LAKUS
ORB_SLAM2::System SLAM(pathVocabularyFile, pathSettingsFile,
ORB_SLAM2::System::MONOCULAR, false);
// // ++++++ LAKUS

cout << endl << endl;
cout << "-----" << endl;
cout << "Augmented Reality Demo - standalone" << endl;
cout << "<a href='https://github.com/raulmur/ORB_SLAM2'>" << endl;
cout << "Changed for Bachelor Thesis of Andreas Lakus " << endl;
cout << "Now AR-Demo runs without ROS with native ORB-SLAM2" << endl;
cout << "1) Translate the camera to initialize SLAM." << endl;
cout << "2) Look at a planar region and translate the camera." << endl;
cout << "3) Press Insert Cube to place a virtual cube in the plane. "
<< endl;
cout << endl;
cout << "You can place several cubes in different planes." << endl;
cout << "-----" << endl;
cout << endl;

viewerAR.SetSLAM(&SLAM);

// ***** LAKUS
cv::VideoCapture cap;
cap.open(camIndex);
cv::Mat img;

if (!cap.isOpened()) {
    cerr << "Failed to open camera!\n";
    return -1;
}
// // ++++++ LAKUS

cv::FileStorage fSettings(pathSettingsFile, cv::FileStorage::READ);
bRGB = static_cast<bool>((int)fSettings["Camera.RGB"]);
float fps = fSettings["Camera.fps"];
viewerAR.SetFPS(fps);

float fx = fSettings["Camera.fx"];
float fy = fSettings["Camera.fy"];
float cx = fSettings["Camera.cx"];
float cy = fSettings["Camera.cy"];

viewerAR.SetCameraCalibration(fx, fy, cx, cy);
```

Anhang

```
K = cv::Mat::eye(3, 3, CV_32F);
K.at<float>(0, 0) = fx;
K.at<float>(1, 1) = fy;
K.at<float>(0, 2) = cx;
K.at<float>(1, 2) = cy;

DistCoef = cv::Mat::zeros(4, 1, CV_32F);
DistCoef.at<float>(0) = fSettings["Camera.k1"];
DistCoef.at<float>(1) = fSettings["Camera.k2"];
DistCoef.at<float>(2) = fSettings["Camera.p1"];
DistCoef.at<float>(3) = fSettings["Camera.p2"];
const float k3 = fSettings["Camera.k3"];
if (k3 != 0)
{
    DistCoef.resize(5);
    DistCoef.at<float>(4) = k3;
}

thread tViewer = thread(&ORB_SLAM2::ViewerAR::Run, &viewerAR);

// ***** LAKUS
bool done = false;
cv::Mat im_raw;
cv::Mat im;
cv::Mat imu;

while (!done) {
    cap >> im_raw;
    cv::cvtColor(im_raw, im, cv::COLOR_RGB2BGR);
    cv::Mat imu;

    cv::Mat Tcw = SLAM.TrackMonocular(im, time(NULL));
    int state = SLAM.GetTrackingState();
    vector<ORB_SLAM2::MapPoint*> vMPs = SLAM.GetTrackedMapPoints();
    vector<cv::KeyPoint> vKeys = SLAM.GetTrackedKeyPointsUn();

    cv::undistort(im, imu, K, DistCoef);

    if (bRGB)
        viewerAR.SetImagePose(imu, Tcw, state, vKeys, vMPs);
    else
    {
        cv::cvtColor(imu, imu, CV_RGB2BGR);
        viewerAR.SetImagePose(imu, Tcw, state, vKeys, vMPs);
    }
}
cap.release();

// Stop all threads
SLAMShutdown();

// Save camera trajectory
SLAM.SaveKeyFrameTrajectoryTUM("KeyFrameTrajectoryAR.txt");

// ++++++ LAKUS
return 0;
}
```

Anhang 9 - Unity-ROS – Subscriber Lokationsdaten

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Unity.Robotics.ROSTCPConnector;
using RosTF = RosMessageTypes.Tf.TfMessageMsg;
using System;

public class RosSubscriberExample : MonoBehaviour
{
    public GameObject cam;
    Vector3 posCam;
    public Vector3 camTranslation;
    public float camRotX = 0;
    public float camRotY = 0;
    public float camRotZ = 0;
    public Vector3 camRotation;

    void Start()
    {
        ROSConnection.instance.Subscribe<RosTF>("tf", respond);
    }

    void respond(RosTF msg)
    {
        camTranslation = new
Vector3((float)msg.transforms[0].transform.translation.y * -100,
(float)msg.transforms[0].transform.translation.z * 100,
(float)msg.transforms[0].transform.translation.x * 100);
        camRotX = (float)msg.transforms[0].transform.rotation.y;
        camRotY = (float)msg.transforms[0].transform.rotation.z;
        camRotZ = (float)msg.transforms[0].transform.rotation.x;
    }

    void Update() {
        cam.transform.position = camTranslation;
        // camRotZ - Roll
        cam.transform.rotation = Quaternion.Euler(camRotX * 100, camRotY *
-100, camRotZ * -100);
    }
}
```

Anhang 10 - Unity-ROS – Bread Crumbs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BreadCrumbs : MonoBehaviour
{
    Camera mainCamera;
    Vector3 pos;
    public GameObject breadCrumb;
    public float spawnTime = 1f;
    public float spawnDelay = 1f;

    // Start is called before the first frame update
    void Start()
    {
        mainCamera = Camera.main;
        pos = new Vector3(0, 0, 0);

        InvokeRepeating("SpawnBreadcrumb", spawnTime, spawnDelay);
    }

    // Update is called once per frame
    void Update()
    {

        //pos = mainCamera.transform.position;
        //Debug.Log(pos);

    }

    public void SpawnBreadcrumb()
    {
        pos = new Vector3(mainCamera.transform.position.x,
mainCamera.transform.position.y, mainCamera.transform.position.z);
        breadCrumb = Instantiate(breadCrumb, pos, transform.rotation);
        Debug.Log("Spawn");
    }
}
```

Anhang 11 - Unity-ROS – Subscriber Point Cloud

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.Robotics.ROSTCPConnector;
using RosPC = RosMessageTypes.Sensor.PointCloud2Msg;

public class RosSubscriberPointCloud : MonoBehaviour
{
    // PCL Data
    public uint xOffset = 0;
    public uint yOffset = 4;
    public uint zOffset = 8;
    public uint pclHeigth = 0;
    public uint pclWidth = 0;
    public uint pointStep = 12;
    public long rowStep = 0;
    public bool parameterSet = false;
    public bool isDense = false;
    public bool isBigEndian = false;
    public int sizeData = 0;
    private byte[] tempData;
    public Vector3[] pcl;

    // Unity Part
    public float xPos, yPos, zPos;
    public GameObject mapPoint;
    public bool rendering = true;
    private int renderingTimer = 10000;
    private int time = 0;

    void Start()
    {
        ROSConnection.instance.Subscribe<RosPC>("orb_slam2_mono/map_points",
        respond);
    }

    void respond(RosPC msg)
    {
        if (!parameterSet) {
```

```

// xOffset | yOffset | zOffset - Index start of PCL-Point in 1D-Array
(Datastream from ROS)
    xOffset = msg.fields[0].offset;
    yOffset = msg.fields[1].offset;
    zOffset = msg.fields[2].offset;
    // Begin of next Datasegment and Legth of Point Data in Bytes
    pointStep = msg.point_step;
    //True if there are no invalid points
    isDense = msg.is_dense;
    // Big endian? - sortet or not (Byteorder)
    isBigEndian = msg.is_bigendian;

    parameterSet = true;
}
// Dimensions of Point Cloud -> growing
pclHeight = msg.height;
pclWidth = msg.width;
// Length of a row in bytes
rowStep = msg.row_step;

// Data size
sizeData = msg.data.GetLength(0);
tempData = new byte[sizeData];
int i = 0;
// Parse Data
foreach(byte temp in msg.data) {
    tempData[i] = temp;
    i++;
}
sizeData = sizeData / (int)pointStep;
}
void GetPCLCoords() {
    pcl = new Vector3[sizeData];
    // Conversion from byte in float
    for (int n = 0; n < sizeData; n++)
    {
        // Debug.Log("Point NR:" + n);
        int x = n * (int)pointStep + (int)xOffset;
        int y = n * (int)pointStep + (int)yOffset;
        int z = n * (int)pointStep + (int)zOffset;

        xPos = BitConverter.ToSingle(tempData, x);
        yPos = BitConverter.ToSingle(tempData, y);
        zPos = BitConverter.ToSingle(tempData, z);

        pcl[n] = new Vector3(yPos * 100, zPos * 100, xPos * 100 + 1);
        Instantiate(mapPoint, pcl[n], transform.rotation);
        // Debug.Log("pclCoordinates:x=" + pcl[n].x + ",y=" + pcl[n].y
        + ",z=" + pcl[n].y);

        rendering = false;
    }
}

```

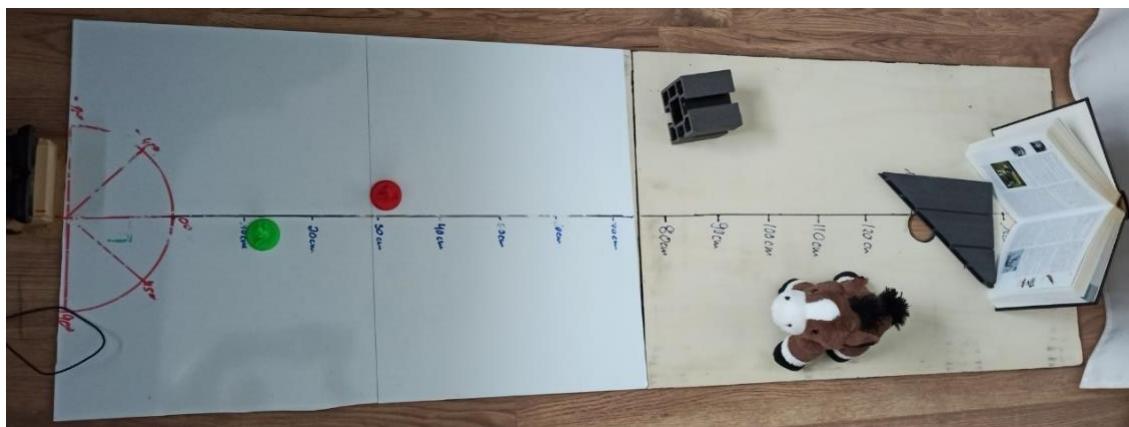
Anhang

```
void Update() {
    Debug.Log(time);
    Debug.Log(renderingTimer);

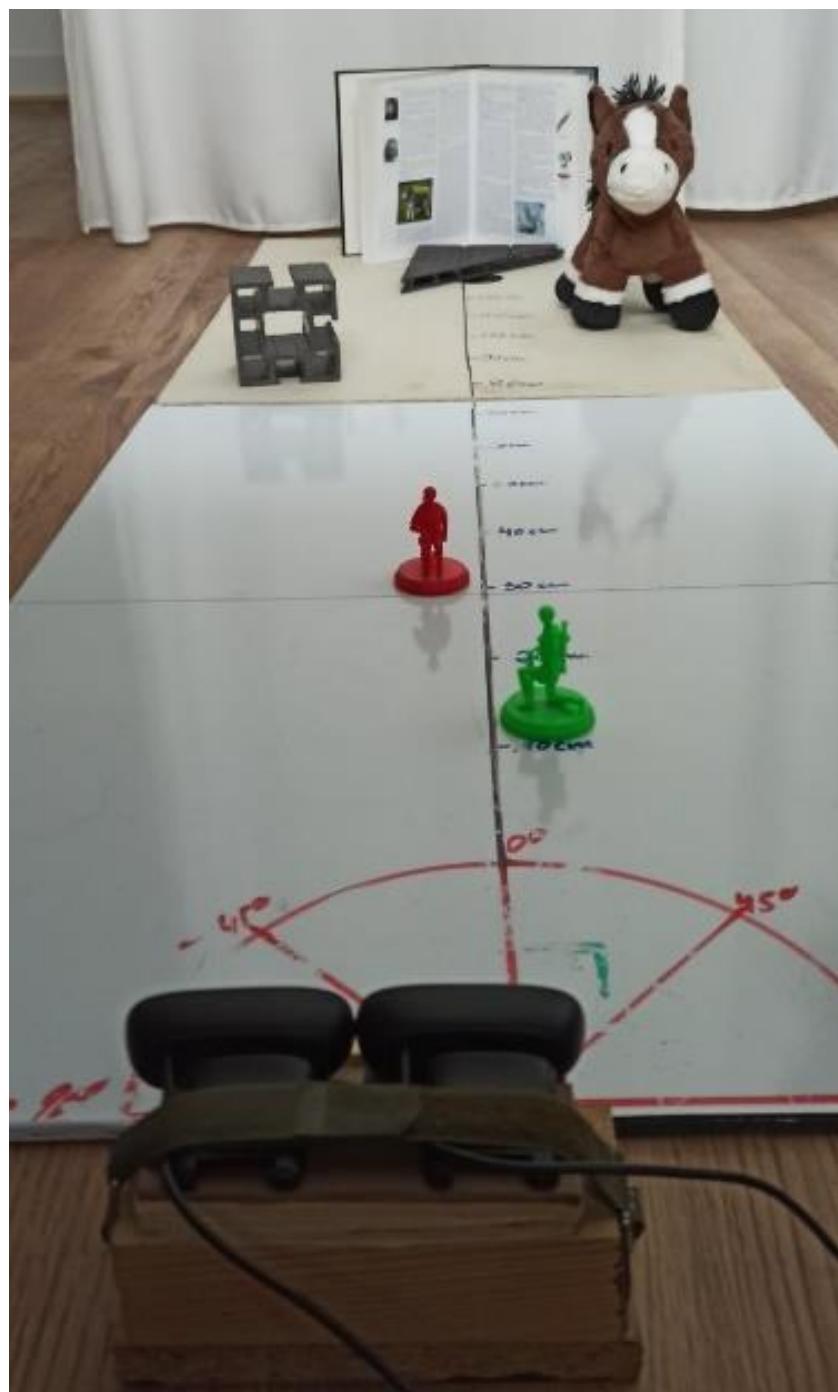
    if (rendering) {
        Debug.Log("RENDERING");
        GetPCLCoords();
    }

    if (time >= renderingTimer) {
        rendering = true;
        time = 0;
    }
    time++;
}
```

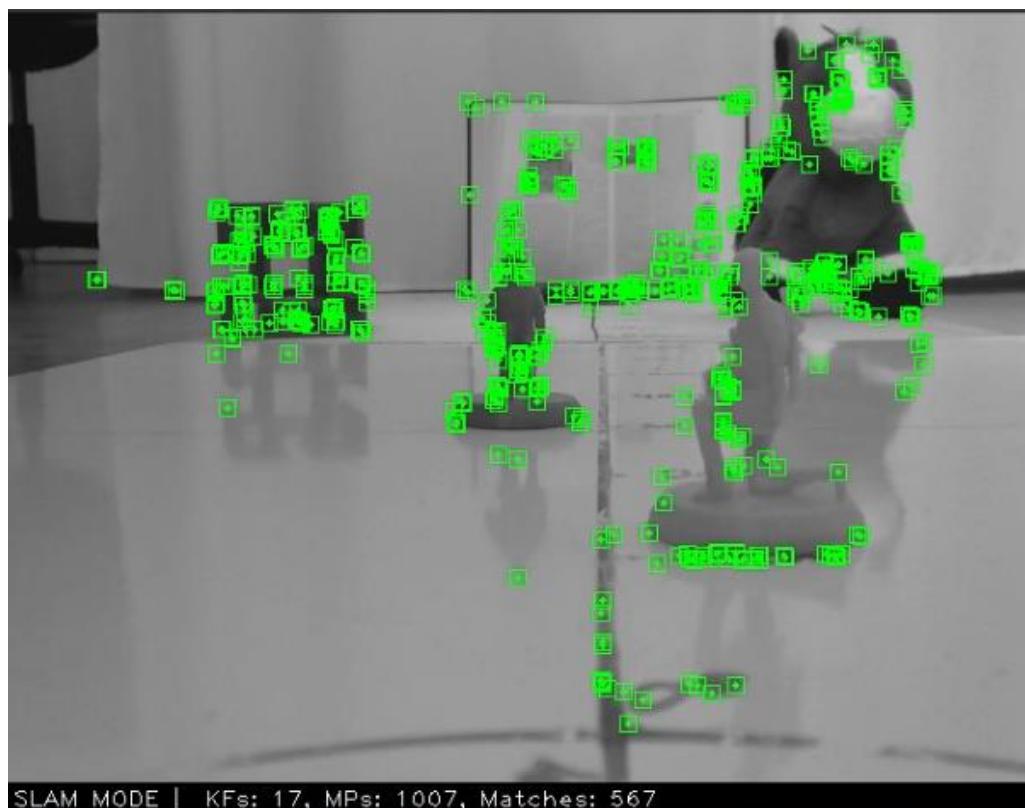
Anhang 12 - Testaufbau Entfernungstest



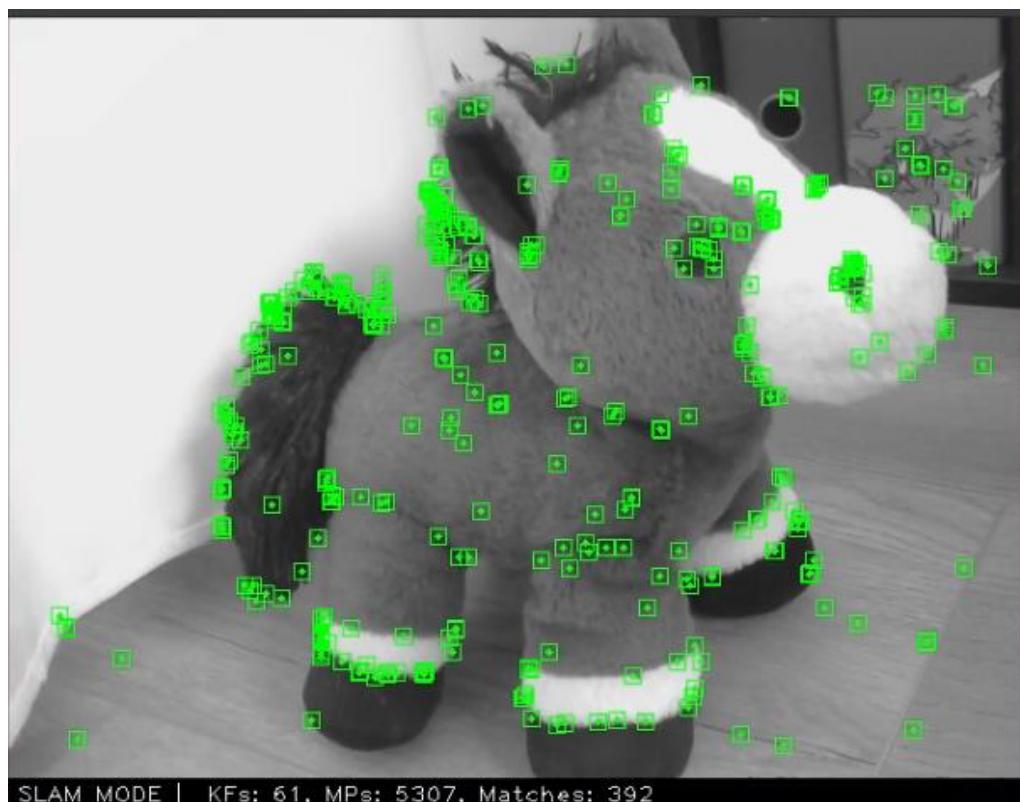
Anhang 13 - Testaufbau Entfernungstest - Kameraperspektive



Anhang 14 - ORB-SLAM-Native – Entfernungstest



Anhang 15 – ORB-SLAM-Native - Rekonstruktion



Anhang 16 - Test – ORBSLAM2 Nativ – TUM rpy

Test Nr.1

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_rpy/  
  
Ausgabe des Tests mit Dataset :  
  
ORB - SLAM2 Copyright(C) 2014 - 2016 Raul Mur - Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions.See LICENSE.txt.  
  
Input sensor was set to : Monocular  
  
Loading ORB Vocabulary.This could take a while...  
Vocabulary loaded!  
  
Camera Parameters :  
-fx : 517.306  
- fy : 516.469  
- cx : 318.643  
- cy : 255.314  
- k1 : 0.262383  
- k2 : -0.953104  
- k3 : 1.16331  
- p1 : -0.005358  
- p2 : 0.002628  
- fps : 30  
- color order : RGB(ignored if grayscale)  
ORB Extractor Parameters :  
-Number of Features : 1000  
- Scale Levels : 8  
- Scale Factor : 1.2  
- Initial Fast Threshold : 20  
- Minimum Fast Threshold : 7  
-----
```

```
Start processing sequence ...
Images in the sequence : 723

New Map created with 91 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 107 points
-----
median tracking time : 0.0153205
mean tracking time : 0.0159898
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/rpy"/>
Estimated trajectory	KeyFrameTrajectory.txt (3239 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
Start evaluation Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.	

Anhang

Keyframe Trajektorie

```
1305031230.497228 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031230.526720 -0.0028024 -0.0001099 0.0002817 -0.0027895 -0.0273884 0.0051163 0.9996079
1305031230.566084 -0.0030275 0.0001087 0.0000525 -0.0050504 -0.0563886 0.0078653 0.9983652
1305031230.598825 -0.0013552 0.0001642 0.0018290 -0.0085184 -0.0872965 0.0101971 0.9960938
1305031230.865523 -0.0014514 0.0027519 0.0092709 -0.0194176 -0.2665499 0.0296596 0.9631689
1305031230.897379 0.0001614 0.0024378 0.0095077 -0.0185724 -0.2912802 0.0304616 0.9559723
1305031230.925860 0.0047742 0.0015841 0.0103747 -0.0200153 -0.3152667 0.0326442 0.9482303
1305031230.997080 0.0063773 0.0012680 0.0118437 -0.0193068 -0.3362800 0.0317070 0.9410301
1305031231.225711 0.0068383 0.00111867 0.0118311 -0.0179598 -0.3097365 0.0305945 0.9501604
1305031231.297846 0.0098116 0.0005421 0.0120918 -0.0184982 -0.2633518 0.0293228 0.9640766
1305031231.365066 0.0187676 0.0000429 0.0103141 -0.0187299 -0.2009050 0.0220542 0.9791833
1305031231.493669 0.0117823 -0.0003754 0.0078074 -0.0221392 -0.1047468 0.0150813 0.9941381
1305031232.025790 0.0223151 0.0017665 0.0014588 -0.0220284 0.2448815 -0.0468190 0.9681714
1305031232.125662 0.0213316 0.0024036 0.0013515 -0.0176945 0.3064376 -0.0558100 0.9500885
1305031232.193716 0.0224157 0.0022754 -0.0007573 -0.0152307 0.3371940 -0.0596637 0.9394192
1305031233.393685 -0.0029296 0.0118848 0.0162247 -0.0050800 0.3706700 -0.0311369 0.9282287
1305031233.461836 0.0069671 0.0042091 0.0050423 -0.0182836 0.2888581 -0.0208258 0.9569708
1305031235.861603 0.0199116 0.0108200 0.0168527 -0.1213864 0.1810440 0.0319721 0.9754313
1305031236.061630 0.0193270 0.0139477 0.0183058 -0.2026629 0.1842539 0.0577464 0.9600227
1305031236.161670 0.0202696 0.0132251 0.0190025 -0.2264138 0.1849815 0.0632555 0.9542103
1305031236.529695 0.0226601 0.0056348 0.0180628 -0.1536208 0.1873730 0.0486257 0.9689828
1305031236.693667 0.0203210 0.0011564 0.0138771 -0.0785600 0.1866642 0.0220881 0.9790285
1305031237.193642 0.0086689 -0.0030565 -0.0065342 0.1833781 0.1756904 -0.0628250 0.9651727
1305031239.193706 0.0122159 0.0051112 0.0054159 -0.1703400 0.1645122 -0.0073221 0.9715279
1305031244.961694 0.0154830 0.0176378 0.0047566 0.2277048 0.1467725 0.4144760 0.8688027
1305031245.061797 0.0153858 0.0173373 0.0052164 0.2309655 0.1407638 0.4173782 0.8675459
1305031249.062337 0.0013091 0.0254108 -0.0163635 0.2983270 0.2136079 0.1905614 0.9105268
1305031249.797667 0.0016583 0.0193200 -0.0156199 0.1683469 0.2672492 -0.1024335 0.9432627
1305031250.061924 -0.0010392 0.0124318 -0.0144190 0.0889020 0.3643069 -0.2757761 0.8850561
1305031251.462288 -0.0039550 0.0087628 -0.0048371 0.0692252 0.2277975 -0.3350601 0.9116199
1305031251.761755 -0.0045358 0.0099481 -0.0072811 0.1650146 0.1661035 -0.0628490 0.9701700
1305031252.029784 -0.0053481 0.0116921 -0.0111924 0.2342181 0.0737553 0.1185323 0.9621082
1305031252.761680 0.0073246 0.0202772 0.0022335 0.2548199 0.1092207 0.2870373 0.9169227
1305031252.929687 0.0042371 0.0169524 -0.0046481 0.2198731 0.1695952 0.1670682 0.9460346
1305031253.197610 -0.0023862 0.0149577 -0.0039181 0.1631914 0.1925389 -0.0462747 0.9665174
1305031253.497697 -0.0009966 0.0119538 -0.0027718 0.1737212 0.1587000 -0.0390424 0.9711390
```

Ausgabe Evaluation

```
compared_pose_pairs 36 pairs
absolute_translational_error.rmse 0.053008 m
absolute_translational_error.mean 0.049856 m
absolute_translational_error.median 0.045634 m
absolute_translational_error.std 0.018005 m
absolute_translational_error.min 0.024838 m
absolute_translational_error.max 0.096774 m
```

Test Nr. 2

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_rpy/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

```

Start processing sequence ...
Images in the sequence: 723

New Map created with 87 points
Wrong initialization, resetting...
System Resetting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 98 points
Wrong initialization, resetting...
System Resetting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 105 points
-----
median tracking time: 0.0164173
mean tracking time: 0.0181199

```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/rpy"/> <input type="button" value="▼"/>
Estimated trajectory	KeyFrameTrajectory.txt (2901 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
<input type="button" value="Start evaluation"/> <p>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.</p>	

Keyframe Trajektorie

```
1305031235.893698 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031235.961704 0.0008413 0.0041137 0.0034599 -0.0280413 0.0001267 0.0023103 0.9996041
1305031235.993673 0.0020281 0.0069007 0.0059185 -0.0393112 -0.0000776 0.0018896 0.9992252
1305031236.329663 0.0075340 0.0224022 0.0169475 -0.0686566 0.0100541 0.0000616 0.9975897
1305031236.493658 0.0063008 0.0205700 0.0064543 -0.0006093 0.0079774 -0.00009091 0.9999676
1305031238.629751 0.0051612 0.0018411 -0.0469369 0.1038297 -0.0254772 -0.0516496 0.9929263
1305031239.993691 0.0087549 0.0096607 -0.0455473 0.3236238 -0.0316301 -0.0425873 0.9446976
1305031240.193676 0.0079122 0.0104979 -0.0455410 0.4249152 -0.0370892 -0.0364538 0.9037381
1305031240.293646 0.0081149 0.0102729 -0.0462092 0.4764240 -0.0394678 -0.0322775 0.8777361
1305031240.393695 0.0077968 0.0095557 -0.0477277 0.5136125 -0.0348732 -0.0297381 0.8567973
1305031240.493713 0.0078366 0.0095098 -0.0483098 0.5330680 -0.0345320 -0.0282964 0.8448937
1305031240.529672 0.0081330 0.0094530 -0.0486173 0.5379646 -0.0347545 -0.0290755 0.8417487
1305031240.631058 0.0082467 0.0090699 -0.0498326 0.5234207 -0.0381462 -0.0301722 0.8506852
1305031240.761607 0.0079418 0.0090023 -0.0496160 0.4921413 -0.0340832 -0.0332846 0.8692108
1305031240.829620 0.0077576 0.0083206 -0.0510160 0.4662878 -0.0313529 -0.0379842 0.8832609
1305031240.929673 0.0074924 0.0069395 -0.0521067 0.4182223 -0.0230767 -0.0448807 0.9069417
1305031241.029623 0.0074457 0.0066237 -0.0519992 0.3563578 -0.0154238 -0.0487261 0.9329507
1305031241.161725 0.0065202 0.0026264 -0.0524488 0.2672545 -0.0207164 -0.0376234 0.9626684
1305031245.529685 0.0064021 -0.0062785 -0.0564748 0.2754411 -0.0491872 0.1286613 0.9513985
1305031248.497597 0.0179327 -0.0018353 -0.0593210 0.3274257 -0.0640753 -0.0422575 0.9417543
1305031248.629727 0.0177945 -0.0007071 -0.0603001 0.3381931 -0.0547507 0.0321496 0.9389324
1305031249.161749 0.0128092 0.0003355 -0.0682800 0.3377954 -0.0110283 0.2739751 0.9003945
1305031249.761822 0.0126249 0.0017241 -0.0655797 0.3077695 0.0575877 -0.0274778 0.9493190
1305031250.061924 0.0152079 -0.0009544 -0.0618385 0.2636199 0.1963809 -0.2328802 0.9152628
1305031250.329709 0.0175764 -0.0051757 -0.0581217 0.2179676 0.1529076 -0.4337697 0.8607864
1305031250.797838 0.0167157 -0.0066942 -0.0552635 0.1367746 0.1772277 -0.6341686 0.7400765
1305031251.661814 0.0170943 -0.0009142 -0.0519859 0.2952726 0.0034159 -0.1260017 0.9470618
1305031251.862143 0.0174125 0.0010152 -0.0520622 0.3151829 -0.1156960 0.0435136 0.9409468
1305031252.661594 0.0097700 -0.0037791 -0.0629494 0.2610309 -0.1787489 0.4203107 0.8504415
1305031252.831021 0.0116025 -0.0044170 -0.0620165 0.2871754 -0.0825085 0.2754105 0.9137132
1305031252.997652 0.0132378 -0.0053445 -0.0603906 0.2828847 -0.0398263 0.1470812 0.9469727
1305031253.165977 0.0173369 -0.0046191 -0.0566997 0.2869513 -0.0254303 0.0010914 0.9576069
```

Ausgabe Evaluation

```
compared_pose_pairs 32 pairs
absolute_translational_error.rmse 0.055013 m
absolute_translational_error.mean 0.052755 m
absolute_translational_error.median 0.052458 m
absolute_translational_error.std 0.015601 m
absolute_translational_error.min 0.028673 m
absolute_translational_error.max 0.081025 m
```

Test Nr. 3

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_rpy/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

Anhang

```
Start processing sequence ...
Images in the sequence: 723

New Map created with 89 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 90 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 83 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 94 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 83 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 97 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 81 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 101 points
-----
median tracking time: 0.016211
mean tracking time: 0.0195156
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/rpy"/> <input type="button" value="▼"/>
Estimated trajectory	<input type="text" value="KeyFrameTrajectory.txt (2076 bytes)"/> Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s) ▼"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
<input type="button" value="Start evaluation"/> <i>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.</i> <i>Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM ↗ as well as instructions how to reproduce ↗ these trajectories.</i>	

Keyframe Trajektorie

```

1305031241.261662 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031241.329910 0.0012660 -0.0005152 -0.0002130 -0.0439785 0.0027830 0.0089302 0.9989887
1305031241.393718 0.0014906 0.0041944 0.0004481 -0.0762880 -0.0026103 0.0308995 0.9966035
1305031241.893622 -0.0005617 0.0034372 0.0019237 -0.0776902 -0.0186982 -0.0107904 0.9967438
1305031242.061654 -0.0004222 0.0032633 0.0018481 -0.0876931 -0.0253665 -0.0853836 0.9921573
1305031244.997688 0.0249871 -0.0054328 -0.0344452 0.0647872 0.0252107 0.4788205 0.8751560
1305031245.261749 0.0186113 0.0091459 -0.0256511 0.0950807 -0.0142313 0.3927207 0.9146188
1305031247.461694 -0.0083542 0.0259797 0.0055926 -0.0548379 -0.0988725 -0.7181006 0.6866940
1305031247.729684 -0.0102020 0.0256813 0.0056394 -0.0263104 -0.0941731 -0.5218550 0.8474117
1305031248.029684 -0.0092792 0.0275472 0.0033962 0.0752231 -0.0534012 -0.2902037 0.9525081
1305031248.661660 -0.0085841 0.0306323 -0.0046678 0.1800655 0.0045573 0.1311205 0.9748657
1305031248.761715 -0.0084968 0.0310627 -0.0058615 0.1943985 0.0081420 0.1939643 0.9615200
1305031249.261609 -0.0031766 0.0299725 -0.0208280 0.1795984 0.0524778 0.3452242 0.9196797
1305031249.662284 -0.0027654 0.0284000 -0.0194151 0.1686880 0.0630121 0.1180203 0.9765475
1305031251.529827 -0.0123154 0.0273387 0.0142003 0.0971578 0.0516652 -0.1856139 0.9764418
1305031251.661814 -0.0122356 0.0278158 0.0129827 0.1176561 0.0339323 -0.0542277 0.9909920
1305031251.761755 -0.0124182 0.0278816 0.0130769 0.1329250 -0.0044506 0.0429513 0.9901850
1305031251.897873 -0.0129204 0.0288687 0.0108128 0.1608455 -0.0641044 0.1560667 0.9724518
1305031252.397494 0.0047297 0.0202976 -0.0213632 0.1226436 -0.0509133 0.4629585 0.8763765
1305031252.729811 0.0066039 0.0217337 -0.0194236 0.1278180 -0.0554664 0.4295849 0.8922123
1305031252.997652 0.0047550 0.0241687 -0.0148309 0.1172720 0.0195410 0.2094503 0.9705648
1305031253.129728 -0.0041442 0.0311124 0.0032916 0.1205541 0.0289814 0.1025891 0.9869662
1305031253.265536 -0.0047213 0.0306888 0.0038972 0.1098665 -0.0038578 0.0595498 0.9921533

```

Ausgabe Evaluation

```
compared_pose_pairs 22 pairs
absolute_translational_error.rmse 0.039010 m
absolute_translational_error.mean 0.034567 m
absolute_translational_error.median 0.036915 m
absolute_translational_error.std 0.018081 m
absolute_translational_error.min 0.007522 m
absolute_translational_error.max 0.072914 m
```

Test Nr. 4

Programmausgabe

```
Befehl:
~/Downloads/ORB_SLAM2/Examples/Monocular$ ./mono_tum ../../Vocabulary/ORBvoc.txt
TUM1.yaml ~/Downloads/Datasets/rbgd_dataset_freiburg1_rpy/

Ausgabe des Tests mit Dataset:

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

Input sensor was set to: Monocular

Loading ORB Vocabulary. This could take a while...
Vocabulary loaded!

Camera Parameters:
- fx: 517.306
- fy: 516.469
- cx: 318.643
- cy: 255.314
- k1: 0.262383
- k2: -0.953104
- k3: 1.16331
- p1: -0.005358
- p2: 0.002628
- fps: 30
- color order: RGB (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 1000
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7

-----
```

```
Start processing sequence ...
Images in the sequence: 723

New Map created with 99 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 83 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 115 points
-----
median tracking time: 0.0166927
mean tracking time: 0.0186322
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/rpy"/> <input type="button" value="▼"/>
Estimated trajectory	KeyFrameTrajectory.txt (2723 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)

Start evaluation

Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.
Alternatively, you can also download the [evaluation script](#) and perform the evaluation offline. Additional information about the [evaluation options](#) and the [file formats](#) is available. We also provide an [example trajectory](#) for freiburg1/xyz by RGBD-SLAM  as well as instructions how to [reproduce](#)  these trajectories.

Anhang

Keyframe Trajektorie

```
1305031237.329672 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031237.461646 -0.0000495 -0.0000945 -0.0005897 0.0494739 0.0080042 -0.0012490 0.9987426
1305031237.493702 -0.0000921 0.0001834 -0.0011135 0.0606213 0.0076693 -0.0012869 0.9981306
1305031239.361715 -0.0002455 0.0070025 0.0011479 -0.3598524 -0.0358635 0.0075223 0.9322894
1305031239.529662 -0.0004872 0.0069084 0.0003997 -0.2980657 -0.0363211 0.0136526 0.9537564
1305031240.193676 -0.0001745 0.0053460 -0.0003519 0.0395254 -0.0248037 0.0162829 0.9987779
1305031240.293646 -0.0000210 0.0036573 -0.0008116 0.0918506 -0.0232185 0.0190656 0.9953195
1305031240.393695 -0.0000477 0.0031307 -0.0012280 0.1364821 -0.0172078 0.0181043 0.9903276
1305031240.493713 0.0000002 0.0025404 -0.0015068 0.1579523 -0.0153980 0.0177471 0.9871672
1305031240.593679 0.0001658 0.0017988 -0.0019037 0.1553650 -0.0173663 0.0174299 0.9875507
1305031240.729633 0.0002804 0.0010386 -0.0020526 0.1174876 -0.0214127 0.0149564 0.9927308
1305031240.829620 0.0003146 0.0001036 -0.0022428 0.0796891 -0.0200685 0.0086914 0.9965798
1305031240.961709 0.0003589 -0.0002752 -0.0020845 0.0062809 -0.0146232 -0.0012170 0.9998726
1305031241.093615 0.0004296 0.0000794 -0.0014803 -0.0800678 -0.0155194 0.0040288 0.9966605
1305031241.429669 0.0008281 0.0016562 0.0017457 -0.2824444 -0.0233655 0.0379649 0.9582473
1305031244.061755 0.0016876 0.0048843 0.0009147 -0.1110352 -0.0405763 -0.0228746 0.9927243
1305031244.197672 0.0017231 0.0048633 0.0008559 -0.1093805 -0.0300007 0.0612219 0.9916591
1305031244.429740 -0.0016264 0.0006803 0.0020601 -0.1136980 0.0552694 0.2264789 0.9657771
1305031245.329736 0.0012097 -0.0006191 0.0009126 -0.0886261 0.0430789 0.3249933 0.9405684
1305031245.597588 0.0034889 0.0018063 -0.0002947 -0.1084719 0.0004703 0.1207978 0.9867327
1305031245.729697 0.0036651 0.0014600 -0.0007181 -0.1343668 -0.0172953 0.0359407 0.9901286
1305031248.701134 0.0084429 -0.0011196 -0.0088905 -0.0349111 -0.0151486 0.1290855 0.9909030
1305031249.229768 0.0063854 -0.0034578 -0.0120450 -0.0313422 0.0946284 0.2999654 0.9487275
1305031249.561863 0.0070839 -0.0020968 -0.0111315 -0.0248608 0.0571424 0.1747721 0.9826350
1305031249.761822 0.0075099 -0.0013282 -0.0102652 -0.0759158 0.0508973 -0.0107862 0.9957560
1305031251.061854 0.0119186 0.0012810 -0.0050681 -0.1391043 -0.1029780 -0.6077729 0.7750210
1305031252.297671 0.0048470 -0.0033321 -0.0072268 -0.0611247 0.0040860 0.4153917 0.9075775
1305031253.165977 0.0102559 0.0006062 -0.0064901 -0.0917071 -0.0167423 0.0422669 0.9947477
1305031253.329972 0.0084000 0.0020842 -0.0051290 -0.0984481 -0.0510728 0.0408049 0.9929927
1305031253.565670 0.0095379 0.0013772 -0.0053647 -0.0745741 -0.0541390 0.0487339 0.9945515
```

Ausgabe Evaluation

```
compared_pose_pairs 30 pairs
absolute_translational_error.rmse 0.051062 m
absolute_translational_error.mean 0.049622 m
absolute_translational_error.median 0.046111 m
absolute_translational_error.std 0.012044 m
absolute_translational_error.min 0.030498 m
absolute_translational_error.max 0.073367 m
```

Test Nr. 5

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_rpy/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----  
Start processing sequence ...  
Images in the sequence: 723  
  
New Map created with 121 points  
-----  
median tracking time: 0.016828  
mean tracking time: 0.0181311
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/rpy"/> <input type="button" value="..."/>
Estimated trajectory	KeyFrameTrajectory.txt (2745 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
Start evaluation <i>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.</i> <i>Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.</i>	

Keyframe Trajektorie

```

1305031236.693667 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031236.793675 -0.0049561 -0.0120762 -0.0196228 0.0485182 -0.0029936 -0.0063292 0.9987978
1305031236.893505 -0.0089726 -0.0269879 -0.0400691 0.0970239 -0.0033100 -0.0146565 0.9951686
1305031239.261721 -0.0466248 0.0114159 -0.0711491 -0.0828060 -0.0084774 -0.0416272 0.9956598
1305031240.229661 -0.0261333 -0.0120568 -0.0936122 0.3623809 -0.0259178 -0.0339336 0.9310515
1305031240.329670 -0.0260870 -0.0146864 -0.0966225 0.4064459 -0.0274112 -0.0316764 0.9127141
1305031240.429707 -0.0259660 -0.0157784 -0.0980503 0.4402357 -0.0230334 -0.0313185 0.8970402
1305031240.529672 -0.0259497 -0.0159912 -0.0985945 0.4579725 -0.0218130 -0.0312441 0.8881493
1305031240.631058 -0.0260825 -0.0170687 -0.1005244 0.4435460 -0.0251109 -0.0305608 0.8953784
1305031240.729633 -0.0265731 -0.0172605 -0.1009745 0.4178288 -0.0242055 -0.0329222 0.9076064
1305031240.829620 -0.0268454 -0.0183732 -0.1021668 0.3849100 -0.0199354 -0.0371227 0.9219917
1305031240.961799 -0.0278750 -0.0196781 -0.1021491 0.3174594 -0.0090844 -0.0414038 0.9473240
1305031246.561679 -0.1454261 0.0262856 -0.1358292 0.0833514 0.1139385 -0.6034060 0.7848387
1305031246.961581 -0.1390706 0.0609559 -0.1524905 0.0252695 0.0624484 -0.8021271 0.5933412
1305031247.661681 -0.1832974 0.0875007 -0.1525783 0.0405098 0.0122935 -0.6493785 0.7592861
1305031247.797650 -0.1914950 0.0724659 -0.1474902 0.0721565 0.0059922 -0.5260689 0.8473542
1305031248.729769 -0.0914876 0.0749813 -0.2041997 0.2982347 -0.0125044 0.0983970 0.9493249
1305031249.062337 -0.0573803 0.1072026 -0.1813224 0.3234960 0.0310417 0.2443042 0.9136204
1305031249.229768 -0.0589799 0.1052821 -0.1786644 0.3074029 0.0171666 0.2724732 0.9115740
1305031249.662284 -0.0575415 0.0758432 -0.1939678 0.2914727 0.0458670 0.0570731 0.9537728
1305031249.997687 -0.0635687 0.0438432 -0.1944885 0.2021924 0.1958219 -0.1891683 0.9407377
1305031250.129780 -0.0959698 0.0391547 -0.1622046 0.1848791 0.2276124 -0.2897687 0.9110688
1305031250.361596 -0.1216730 0.0175584 -0.1741203 0.1299576 0.1746580 -0.4671549 0.8569551
1305031250.498138 -0.1285964 0.0223418 -0.1647623 0.1034840 0.1911475 -0.5444168 0.8101630
1305031250.829785 -0.1445567 0.0476135 -0.1172819 0.0529101 0.1923698 -0.6476524 0.7353508
1305031251.730175 -0.1044879 0.0056679 -0.1804234 0.2213310 0.0134426 -0.0650540 0.9729337
1305031252.098735 -0.0717967 0.0416257 -0.1672571 0.2706730 -0.1196327 0.2169147 0.9302538
1305031252.761680 -0.0589328 0.0713760 -0.1278253 0.2591112 -0.0809118 0.3297427 0.9042037
1305031252.965771 -0.0685643 0.0685712 -0.1309978 0.2525471 -0.0055844 0.1751833 0.9515774
1305031253.129728 -0.0969909 0.0384428 -0.1533688 0.2306372 0.0297296 0.0332205 0.9720180

```

Ausgabe Evaluation

```
compared_pose_pairs 30 pairs
absolute_translational_error.rmse 0.036771 m
absolute_translational_error.mean 0.033442 m
absolute_translational_error.median 0.033553 m
absolute_translational_error.std 0.015288 m
absolute_translational_error.min 0.009214 m
absolute_translational_error.max 0.072186 m
```

Anhang 17 - Test – ORBSLAM2 Nativ – TUM xyz

Test Nr. 1

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_xyz/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

Anhang

```
Start processing sequence ...
Images in the sequence: 798

New Map created with 86 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 98 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 99 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 75 points
Wrong initialization, reseting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 135 points
-----
median tracking time: 0.0188573
mean tracking time: 0.0211068
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/xyz"/> <input type="button" value="▼"/>
Estimated trajectory	<input type="button" value="Durchsuchen..."/> KeyFrameTrajectory.txt
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s) ▾"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
Start evaluation <i>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.</i>	

Keyframe Trajektorie

```

1305031110.611307 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031110.979345 -0.0197039 0.0064662 0.0076493 -0.0025788 -0.0161709 -0.0160618 0.9997369
1305031111.079283 -0.0037935 0.0094170 0.0067255 -0.0049073 -0.0040457 -0.0094651 0.9999350
1305031112.043270 0.3164802 0.0255731 -0.0486724 0.0481293 0.1098041 0.0765397 0.9898325
1305031112.144342 0.3466890 0.0319965 -0.0552745 0.0516198 0.1347991 0.0676576 0.9872118
1305031112.343323 0.3992836 0.0287792 -0.0590834 0.0672714 0.1780446 0.0518834 0.9803483
1305031112.679952 0.4828822 0.0453769 -0.0795051 0.0892607 0.1636739 0.1086937 0.9764369
1305031112.743245 0.4845305 0.0562460 -0.0788977 0.0862255 0.1709334 0.1147667 0.9747695
1305031112.843286 0.4713915 0.0506353 -0.0698042 0.0730279 0.1729326 0.1137114 0.9756182
1305031112.943321 0.4580658 0.0446245 -0.0652860 0.0645222 0.1624297 0.1172567 0.9776013
1305031114.811303 -0.0533400 -0.0315032 0.0343218 0.0542017 0.00233418 -0.0123511 0.9984509
1305031114.979280 -0.0348615 -0.0200454 0.0337648 0.0502041 0.0021089 -0.0164143 0.9986019
1305031116.279385 0.4361310 0.0499128 0.0076039 0.0091391 0.1703712 0.1765243 0.9693964
1305031116.411333 0.4410149 0.0500870 0.0046575 0.0138337 0.1954404 0.1874541 0.9625345
1305031116.643355 0.4024226 0.0383126 -0.009833 0.0131628 0.1883605 0.1732948 0.9666002
1305031116.779298 0.3647523 0.0345429 -0.0303940 0.0084130 0.1670954 0.1326391 0.9769418
1305031116.943296 0.2756158 0.0356464 0.0140373 0.0163238 0.1353244 0.0948721 0.9861137
1305031121.647183 0.2150871 -0.1633315 0.0544423 0.0792068 0.0447165 0.0540596 0.9943864
1305031122.114672 0.2122052 -0.1715594 0.0562947 0.0687369 0.0481148 0.0567488 0.9948567
1305031123.652411 0.1564170 0.2119537 0.0187790 -0.0302739 0.0645090 0.0790418 0.9943211
1305031123.811608 0.1574717 0.2325736 0.0190051 -0.0412225 0.0698255 0.0884926 0.9927710
1305031124.011302 0.1575038 0.2268190 0.0218680 -0.0298615 0.0611055 0.0920196 0.9934318
1305031124.182694 0.1609613 0.1922150 0.0269086 -0.0163421 0.0529618 0.0855365 0.9947922
1305031124.382420 0.1722009 0.1373102 0.0277252 0.0008851 0.0367600 0.0781795 0.9962609
1305031125.250632 0.2100997 -0.1115464 0.0619467 0.0586684 0.0261013 0.0518191 0.9965900
1305031125.551020 0.2162523 -0.1527175 0.0738401 0.0598167 0.0542001 0.0401188 0.9959291
1305031126.079356 0.2007754 -0.0563357 0.0509645 0.0619714 0.0266293 0.0441311 0.9967461
1305031126.247298 0.1976053 -0.0014468 0.0523230 0.0451489 0.0266534 0.0484125 0.9974505
1305031127.047297 0.1886415 0.1306062 0.0372830 -0.0062555 0.0513527 0.0788796 0.9955409
1305031127.379262 0.1888066 0.1104375 0.0420827 0.0044056 0.0393180 0.0741998 0.9964583
1305031127.579345 0.1887356 0.0996291 0.0480584 0.0074802 0.0345842 0.0697609 0.9969360
1305031128.279336 0.1931650 0.0813169 0.0596144 0.0041596 0.0272959 0.0742401 0.9968581

```

Ausgabe Evaluation

```

compared_pose_pairs 32 pairs
absolute_translational_error.rmse 0.051052 m
absolute_translational_error.mean 0.047235 m
absolute_translational_error.median 0.050089 m
absolute_translational_error.std 0.019369 m
absolute_translational_error.min 0.005429 m
absolute_translational_error.max 0.078074 m

```

Test Nr. 2

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_xyz/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
-----
```

```
Start processing sequence ...
Images in the sequence: 798

New Map created with 88 points
Wrong initialization, resetting...
System Reseting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 98 points
Wrong initialization, resetting...
System Reseting
Resetting Local Mapper... done
Resetting Loop Closing... done
Resetting Database... done
New Map created with 104 points
-----
median tracking time: 0.0187696
mean tracking time: 0.0209025
```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/xyz"/>
Estimated trajectory	KeyFrameTrajectory.txt (2800 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)

Start evaluation

Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.
Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.

Ausgabe Evaluation

```
compared_pose_pairs 31 pairs
absolute_translational_error.rmse 0.094834 m
absolute_translational_error.mean 0.089324 m
absolute_translational_error.median 0.092443 m
absolute_translational_error.std 0.031853 m
absolute_translational_error.min 0.038948 m
absolute_translational_error.max 0.147424 m
```

Keyframe Trajektorie

```
1305031108.043418 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1305031108.111378 0.0104035 -0.0005746 0.0064229 0.0032282 0.0154148 0.0056802 0.9998598
1305031108.143334 0.0182431 0.0013411 0.0078504 0.0055944 0.0180743 0.0066434 0.9997989
1305031108.575414 0.1175179 0.0133725 0.0344436 -0.0192999 0.0305813 0.0365097 0.9986788
1305031108.775493 0.1598682 0.0204861 0.0417372 -0.0283763 0.0598874 0.0500463 0.9965459
1305031110.743249 -0.2345575 0.0283879 0.0173089 -0.0906984 -0.1276118 -0.0493887 0.9864329
1305031110.843218 -0.2412448 0.0295768 0.0166022 -0.0923851 -0.1308045 -0.0537492 0.9856299
1305031112.679952 0.1653181 0.0227016 0.0463708 -0.0106313 0.0487575 0.0636029 0.9967268
1305031112.811310 0.1570296 0.0245330 0.0506357 -0.0262834 0.0650749 0.0682388 0.9951974
1305031112.943321 0.1437886 0.0265083 0.0517654 -0.0354475 0.0496486 0.0696428 0.9957049
1305031113.243227 0.0817816 0.0262405 0.0404063 -0.0416851 0.0187217 0.0568924 0.9973340
1305031114.679251 -0.2679306 0.0078349 0.0355001 -0.0393423 -0.1092446 -0.0411023 0.9923852
1305031114.779289 -0.2673324 0.0086974 0.0384712 -0.0353057 -0.1198362 -0.0473816 0.9910337
1305031115.111230 -0.2198846 0.0234771 0.0483163 -0.0488667 -0.1165216 -0.0518655 0.9906285
1305031116.179572 0.0920144 0.0410163 0.0951522 -0.0882100 0.0475323 0.1116589 0.9886819
1305031116.443369 0.1244278 0.0399940 0.0969606 -0.0962199 0.0912613 0.1407857 0.9811180
1305031116.611261 0.0987844 0.0335905 0.0921557 -0.0934001 0.0908757 0.1253467 0.9835173
1305031116.743291 0.0658283 0.0404219 0.0845468 -0.0840044 0.0658949 0.0884072 0.9983461
1305031116.912044 0.0056632 0.0409792 0.0807824 -0.0804784 0.0348274 0.0516126 0.9948097
1305031119.911401 -0.0877886 0.1412570 0.0340941 -0.0805832 -0.0602781 0.0197364 0.9947278
1305031120.115232 -0.0902930 0.1568348 0.0290355 -0.1018528 -0.0558415 0.0257910 0.9928960
1305031121.683200 -0.0808317 -0.1064922 0.1187263 -0.0198154 -0.0685275 0.0175864 0.9972973
1305031122.214959 -0.0812493 -0.0805902 0.1124280 -0.0248835 -0.0676830 0.0053250 0.9973823
1305031123.783858 -0.0835662 0.1918966 0.0319154 -0.1416341 -0.0410501 0.0364217 0.9883968
1305031124.011302 -0.0849417 0.1886919 0.0343787 -0.1322760 -0.0496714 0.0429255 0.9890365
1305031124.211396 -0.0862170 0.1629630 0.0468019 -0.1070766 -0.0601795 0.0365683 0.9917539
1305031125.250632 -0.0793222 -0.0603414 0.1172289 -0.0380806 -0.0917323 0.0157151 0.9949312
1305031125.451195 -0.0798319 -0.0828744 0.1289975 -0.0341060 -0.0753879 0.0057215 0.9965544
1305031127.079299 -0.0726137 0.1210355 0.0640159 -0.1044429 -0.0617238 0.0335230 0.9920474
1305031127.379262 -0.0741370 0.1073837 0.0700499 -0.0932609 -0.0746974 0.0310217 0.9923509
1305031128.611395 -0.0782686 0.0832727 0.0857137 -0.0910093 -0.0850566 0.0247463 0.9919024
```

Test Nr. 3

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_xyz/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

```

Start processing sequence ...
Images in the sequence: 798

New Map created with 83 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 77 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 100 points
-----
median tracking time: 0.0183865
mean tracking time: 0.0204396

```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/xyz"/>
Estimated trajectory	<input type="text" value="KeyFrameTrajectory.txt (2637 bytes)"/> Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> second(s) ▾ (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
Start evaluation <small>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.</small> <small>Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM ▾ as well as instructions how to reproduce ▾ these trajectories.</small>	

Anhang

Keyframe Trajektorie

```
1305031109.843296 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1305031109.943299 -0.0406738 0.0073174 0.0067079 -0.0052266 -0.0101374 -0.0113745 0.9998702
1305031110.979345 -0.2929302 0.0344765 0.0308844 -0.0448681 -0.0963059 -0.0985569 0.9894435
1305031112.379360 0.2398029 -0.0352522 0.0309527 0.0352006 0.0819679 -0.0251564 0.9956954
1305031112.479418 0.2710281 -0.0353406 0.0303934 0.0435504 0.0762798 -0.0049477 0.9961226
1305031112.643246 0.3233901 -0.0332010 0.0252586 0.0482218 0.0676257 0.0271300 0.9961753
1305031112.743245 0.3282316 -0.0231671 0.0306283 0.0427365 0.0827215 0.0354563 0.9950244
1305031112.979278 0.2930605 -0.0271161 0.0383242 0.0205762 0.0679296 0.0335941 0.9969121
1305031113.211259 0.2187610 -0.0266652 0.0302448 0.0130695 0.0493420 0.0207477 0.9984809
1305031114.543236 -0.3237538 0.0032859 0.0567474 0.0227831 -0.0623661 -0.0897145 0.9937518
1305031114.779289 -0.3472238 -0.0014555 0.0613446 0.0126066 -0.0813456 -0.0909160 0.9924507
1305031114.979280 -0.3216693 0.0100070 0.0603918 0.0092827 -0.0820078 -0.0962662 0.9919281
1305031116.379384 0.2713169 -0.0233307 0.1136078 -0.0425382 0.1094624 0.0976209 0.9882705
1305031116.579313 0.2522506 -0.0128632 0.1024861 -0.0326734 0.1141434 0.1039675 0.9874687
1305031116.711634 0.2023511 -0.0113580 0.0971097 -0.0291627 0.0925514 0.0678041 0.9929684
1305031116.880165 0.1186732 -0.0013171 0.0931170 -0.0283770 0.0661931 0.0235240 0.9971258
1305031120.347787 -0.0533498 0.1718617 0.0597396 -0.0312846 -0.0204135 -0.0198426 0.9991050
1305031121.847335 -0.0635430 -0.2478566 0.1564518 0.0197743 -0.0354846 -0.0183336 0.9990063
1305031122.114672 -0.0634864 -0.2194129 0.1510139 0.0208249 -0.0399650 -0.0240366 0.9986948
1305031123.579583 -0.0451782 0.2372203 0.0663771 -0.0632983 -0.0172112 -0.0095447 0.9978006
1305031123.751723 -0.0373834 0.2702124 0.0655444 -0.0769677 -0.0065310 -0.0028679 0.9970081
1305031124.051585 -0.0405685 0.2563638 0.0673143 -0.0692410 -0.0190428 0.0046081 0.9974076
1305031125.551020 -0.0608002 -0.1945334 0.1696661 0.0148660 -0.0329649 -0.0424759 0.9984428
1305031125.711443 -0.0651591 -0.1975413 0.1595253 0.0226569 -0.0348871 -0.0388473 0.9983789
1305031126.079356 -0.0550281 -0.0753846 0.1328819 0.0181667 -0.0598851 -0.0375525 0.9973332
1305031126.947437 -0.0250020 0.1499947 0.0999786 -0.0486837 -0.0272519 -0.0089653 0.9984021
1305031127.811829 -0.0341674 0.1058786 0.1158184 -0.0338198 -0.0506216 -0.0137739 0.9980501
1305031128.179350 -0.0320250 0.0909238 0.1285428 -0.0421291 -0.0543923 -0.0099619 0.9975808
1305031128.679282 -0.0371684 0.0829827 0.1256949 -0.0368614 -0.0555002 -0.0177978 0.9976193
```

Ausgabe Evaluation

```
compared_pose_pairs 29 pairs
absolute_translational_error.rmse 0.010138 m
absolute_translational_error.mean 0.008877 m
absolute_translational_error.median 0.008725 m
absolute_translational_error.std 0.004896 m
absolute_translational_error.min 0.001512 m
absolute_translational_error.max 0.019347 m
```

Test Nr. 4

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_xyz/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

```

Start processing sequence ...
Images in the sequence: 798

New Map created with 83 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 90 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 98 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 101 points
-----
median tracking time: 0.0200443
mean tracking time: 0.0219075

```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/xyz"/>
Estimated trajectory	<input type="text" value="KeyFrameTrajectory.txt (2921 bytes)"/> Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="text" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
Start evaluation <small>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers.</small> <small>Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.</small>	

Keyframe Trajektorie

```
1305031109.843296 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000
1305031109.975258 -0.0568093 0.0065647 0.0119087 -0.0098514 -0.0112852 -0.0162111 0.9997564
1305031111.111508 -0.2664771 0.0324976 0.0394326 -0.0482796 -0.0864626 -0.0889461 0.9911014
1305031112.379360 0.2447624 -0.0346026 0.0371790 0.0356316 0.0831993 -0.0253848 0.9955721
1305031112.479418 0.2774099 -0.0348117 0.0375048 0.0438120 0.0771743 -0.0058371 0.9960374
1305031112.711251 0.3370245 -0.0335666 0.0327119 0.0443589 0.0790095 0.0313021 0.9953943
1305031112.811310 0.3209248 -0.0340446 0.0436470 0.0300970 0.0919377 0.0347978 0.9947013
1305031113.143306 0.2550549 -0.0219870 0.0368204 0.0152774 0.0547016 0.0191341 0.9982025
1305031114.443345 -0.3096550 0.0029246 0.0550712 0.0273237 -0.0449537 -0.0860445 0.9949015
1305031114.711306 -0.3566478 -0.0033647 0.0641643 0.0090212 -0.0784312 -0.0834580 0.9933791
1305031115.011300 -0.3222927 0.0157731 0.0641473 0.0068074 -0.0826195 -0.0979177 0.9917358
1305031116.211299 0.2425232 -0.0061466 0.1216551 -0.0348144 0.0759829 0.0832980 0.9930136
1305031116.343292 0.2752623 -0.0192765 0.1258534 -0.0459965 0.1064442 0.0978836 0.9884193
1305031116.611261 0.2410391 -0.0170028 0.1131719 -0.0333549 0.1134479 0.0962573 0.9883074
1305031116.743291 0.1938852 -0.0016820 0.1042086 -0.0258852 0.0883966 0.0573356 0.9940969
1305031116.912044 0.0980123 0.0025574 0.0991785 -0.0266526 0.0588229 0.0178272 0.9977533
1305031121.847335 -0.0672616 -0.2515851 0.1615120 0.0226375 -0.0349426 -0.0184742 0.9989621
1305031122.214959 -0.0636901 -0.1789270 0.1488879 0.0263285 -0.0371179 -0.0390945 0.9981987
1305031123.182155 -0.0601260 0.1337754 0.0690417 -0.0225664 -0.0495050 -0.0354478 0.9978895
1305031123.511360 -0.0505697 0.2256241 0.0702531 -0.0621954 -0.0162634 -0.0157547 0.9978071
1305031123.683753 -0.0439892 0.2597665 0.0673010 -0.0752662 -0.0096633 -0.0078164 0.9970860
1305031123.851444 -0.0390807 0.2800801 0.0653348 -0.0827439 -0.0068782 0.0014555 0.9965460
1305031124.011302 -0.0413563 0.2730278 0.0697608 -0.0713477 -0.0160098 0.0038040 0.9973158
1305031125.551020 -0.0620135 -0.1960692 0.1758625 0.0173920 -0.0332734 -0.0417743 0.9984214
1305031125.911305 -0.0675354 -0.1385069 0.1446254 0.0287260 -0.0383112 -0.0430907 0.9979230
1305031126.079356 -0.0547335 -0.0785527 0.1380327 0.0178556 -0.0609271 -0.0368166 0.9973031
1305031127.079299 -0.0272953 0.1514109 0.1029293 -0.0485722 -0.0289840 -0.0073649 0.9983719
1305031127.847366 -0.0356361 0.1060390 0.1213330 -0.0355514 -0.0507369 -0.0147350 0.9979703
1305031128.111391 -0.0357487 0.0984723 0.1306687 -0.0411223 -0.0531950 -0.0132537 0.9976490
1305031128.679282 -0.0392813 0.0848625 0.1314621 -0.0373083 -0.0553482 -0.0181522 0.9976047
```

Ausgabe Evaluation

```
compared_pose_pairs 30 pairs
absolute_translational_error.rmse 0.008699 m
absolute_translational_error.mean 0.007522 m
absolute_translational_error.median 0.006731 m
absolute_translational_error.std 0.004371 m
absolute_translational_error.min 0.001536 m
absolute_translational_error.max 0.016976 m
```

Test Nr. 5

Programmausgabe

```
Befehl:  
~/Downloads/ORB_SLAM2/Examples/Monocular$./mono_tum ../../Vocabulary/ORBvoc.txt  
TUM1.yaml ~/Downloads/Datasets/rgbd_dataset_freiburg1_xyz/  
  
Ausgabe des Tests mit Dataset:  
  
ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.  
This program comes with ABSOLUTELY NO WARRANTY;  
This is free software, and you are welcome to redistribute it  
under certain conditions. See LICENSE.txt.  
  
Input sensor was set to: Monocular  
  
Loading ORB Vocabulary. This could take a while...  
Vocabulary loaded!  
  
Camera Parameters:  
- fx: 517.306  
- fy: 516.469  
- cx: 318.643  
- cy: 255.314  
- k1: 0.262383  
- k2: -0.953104  
- k3: 1.16331  
- p1: -0.005358  
- p2: 0.002628  
- fps: 30  
- color order: RGB (ignored if grayscale)  
  
ORB Extractor Parameters:  
- Number of Features: 1000  
- Scale Levels: 8  
- Scale Factor: 1.2  
- Initial Fast Threshold: 20  
- Minimum Fast Threshold: 7  
  
-----
```

```

Start processing sequence ...
Images in the sequence: 798

New Map created with 83 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 82 points
Wrong initialization, resetting...
System Reseting
Reseting Local Mapper... done
Reseting Loop Closing... done
Reseting Database... done
New Map created with 104 points
-----
median tracking time: 0.0186869
mean tracking time: 0.0207981

```

Evaluationskonfiguration

Submission form for automatic evaluation of RGB-D SLAM results

Groundtruth trajectory	<input type="text" value="freiburg1/xyz"/>
Estimated trajectory	KeyFrameTrajectory.txt (2816 bytes) Or upload new file: <input type="button" value="Durchsuchen..."/> Keine Datei ausgewählt.
Evaluation options	Offset: <input type="text" value="0.00"/> seconds (add to stamps of estimated traj.) Scale: <input type="text" value="1.00"/> (scale estimated traj. by this factor)
Evaluation mode	<input checked="" type="radio"/> absolute trajectory error (recommended for the evaluation of visual SLAM methods) <input type="radio"/> relative pose error for pose pairs with a distance of <input type="text" value="1"/> <input type="button" value="second(s)"/> (recommended for the evaluation of visual odometry methods) <input type="radio"/> relative pose error for all pairs (downsampled to <input type="text" value="10000"/> pairs, alternative method for the evaluation of visual SLAM methods)
<input type="button" value="Start evaluation"/> <p>Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.</p>	

Keyframe Trajektorie

```

1305031109.575362 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1305031109.675263 -0.0328574 0.0069937 -0.0047115 -0.0014673 -0.0020994 -0.0012009 0.9999960
1305031111.043327 -0.3184755 0.0448273 0.0185782 -0.0607099 -0.1052367 -0.0988785 0.9876551
1305031111.743386 -0.0831495 0.0093686 0.0359799 -0.0273846 -0.0475003 -0.0433954 0.9975523
1305031112.379360 0.1043817 -0.0143216 0.0329464 0.0207969 0.0760404 -0.0326396 0.9963533
1305031112.479418 0.1301705 -0.0147146 0.0329869 0.0289494 0.0708765 -0.0125984 0.9969853
1305031112.611261 0.1662362 -0.0128252 0.0297225 0.0364405 0.0578601 0.0114105 0.9975942
1305031112.711251 0.1792338 -0.0130058 0.0302585 0.0298849 0.0737326 0.0246265 0.9965259
1305031112.879421 0.1646814 -0.0064981 0.0395972 0.0116373 0.0770423 0.0282177 0.9965605
1305031112.979278 0.1467529 -0.0071935 0.0414229 0.0058855 0.0646052 0.0254512 0.9975689
1305031114.111263 -0.2431648 0.0108775 0.0281476 0.0189788 -0.0240107 -0.0626250 0.9976879
1305031114.611391 -0.3686797 0.0208070 0.0382313 0.0035251 -0.0808118 -0.0874259 0.9928815
1305031115.111230 -0.3154979 0.0289912 0.0516587 -0.0168123 -0.0946331 -0.0986399 0.9904706
1305031116.311336 0.1253431 0.0106266 0.1051081 -0.0558070 0.0945262 0.0862671 0.9902062
1305031116.443369 0.1315669 0.0080151 -0.0509275 0.1104107 0.1027530 0.9872475
1305031116.579313 0.1110907 0.0093896 0.0964430 -0.0454763 0.1125684 0.0948318 0.9880623
1305031116.711634 0.0735980 0.0123702 0.0894822 -0.0412825 0.0872830 0.0599519 0.9935206
1305031116.880165 0.0080529 0.0140761 0.0803947 -0.0447433 0.0574818 0.0167346 0.9972030
1305031120.079418 -0.1331440 0.1932950 0.0360540 -0.0606831 -0.0309250 -0.0224567 0.9974251
1305031121.782835 -0.1483991 -0.1814000 0.1313869 0.0105281 -0.0495162 -0.0264187 0.9983686
1305031121.914931 -0.1505393 -0.1899315 0.1347830 0.0033705 -0.0477947 -0.0133994 0.9987616
1305031123.711323 -0.1194458 0.2365638 0.0421097 -0.0952649 -0.0188698 -0.0123692 0.9951962
1305031123.883786 -0.1208081 0.2480383 0.0416008 -0.0977996 -0.0182491 -0.0036174 0.9950322
1305031124.051505 -0.1214802 0.2304200 0.0447638 -0.0888639 -0.0298159 -0.0018932 0.9955956
1305031124.382420 -0.1219830 0.1479255 0.0626841 -0.0608586 -0.0563169 -0.0121552 0.9964823
1305031125.650575 -0.1491179 -0.1528979 0.1412519 0.0031970 -0.0370503 -0.0447938 0.9983038
1305031126.011333 -0.1414819 -0.0702978 0.1131208 0.0067401 -0.0723731 -0.0407565 0.9965218
1305031126.879357 -0.1120986 0.1395946 0.0788214 -0.0662053 -0.0406814 -0.0168984 0.9968331
1305031127.211700 -0.1112913 0.1344911 0.0780599 -0.0626591 -0.0505708 -0.0122873 0.9966772
1305031127.847366 -0.1187885 0.1047175 0.0920513 -0.0524654 -0.0620939 -0.0198332 0.9964930
1305031128.479296 -0.1208053 0.0925510 0.1022731 -0.0557476 -0.0656480 -0.0212918 0.9960568

```

Ausgabe Evaluation

```

compared_pose_pairs 31 pairs
absolute_translational_error.rmse 0.047984 m
absolute_translational_error.mean 0.045123 m
absolute_translational_error.median 0.042725 m
absolute_translational_error.std 0.016321 m
absolute_translational_error.min 0.017778 m
absolute_translational_error.max 0.076577 m

```

Anhang 18 - Inhalt beiliegender DVDs

- DVD 1 – Thesis, Implementierung, Literatur
 - Citavi – Thesis
 - Literatur
 - Thesis
 - Implementierung
 - Camera Calibration
 - ORB-Extractor
 - ORB-SLAM2-Native
 - ORB-SLAM2-ROS
 - Unity-Projekt
- DVD 2 - Testergebnisse
 - Datasets
 - Kamera Kalibrierung
 - ORB-SLAM2-Native
 - ORB-SLAM2-ROS
 - ROS-ORBSLAM2-Unity

Literatur

- ACM-DL, 2021a. *Peter C. Cheeseman* [online]. Association for computing machinery - Digital Library. 25. Juni 2021 [Zugriff am: 25. Juni 2021]. Verfügbar unter:
<https://dl.acm.org/profile/81100139683>
- ACM-DL, 2021b. *Randall C. Smith* [online]. Association for computing machinery - Digital Library. 25. Juni 2021 [Zugriff am: 25. Juni 2021]. Verfügbar unter:
<https://dl.acm.org/profile/81100078935>
- AHMAD, N., R.A.R. GHAZILLA, N.M. KHAIRI und V. KASI, 2013. Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications [online]. *International Journal of Signal Processing Systems*, 256-262. ISSN 2315-4535. Verfügbar unter:
doi:10.12720/ijspcs.1.2.256-262
- AITHAL, P.S. und S. AITHAL, 2015. *A Review on Anticipated Breakthrough Technologies of 21st Century*.
- AL-SADA, M., K. JIANG, S. RANADE, M. KALKATTAWI und T. NAKAJIMA, 2020. HapticSnakes: multi-haptic feedback wearable robots for immersive virtual reality [online]. *Virtual Reality*, 24(2), 191-209. ISSN 1359-4338. Verfügbar unter: doi:10.1007/s10055-019-00404-x
- AZUMA, R., 2021. *Ronald Azuma: Home* [online]. 15. März 2021 [Zugriff am: 4. Juli 2021]. Verfügbar unter: <https://ronaldazuma.com/>
- AZUMA, R.T., 1997. A Survey of Augmented Reality [online]. *Presence: Teleoperators and Virtual Environments*, 6(4), 355-385. ISSN 1054-7460. Verfügbar unter:
doi:10.1162/pres.1997.6.4.355
- BAILEY, T. und H. DURRANT-WHYTE, 2006. Simultaneous localization and mapping (SLAM): part II [online]. *IEEE Robotics & Automation Magazine*, 13(3), 108-117. ISSN 1070-9932. Verfügbar unter: doi:10.1109/MRA.2006.1678144
- BANKS, M.S., J.C.A. READ, R.S. ALLISON und S.J. WATT, 2012. Stereoscopy and the Human Visual System [online]. *SMPTE motion imaging journal*, 121(4), 24-43. ISSN 1545-0279. Verfügbar unter: doi:10.5594/j18173

- BEYERER, J., F. PUENTE LEÓN und C. FRESE, 2016. *Automatische Sichtprüfung. Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung* [online]. 2., erweiterte und verbesserte Auflage. Berlin: Springer Vieweg. Verfügbar unter: <http://www.springer.com/de/book/9783662477854>
- BORTHWICK, S. und H. DURRANT-WHYTE, 1994. Dynamic localisation of autonomous guided vehicles. In: *Proceedings of 1994 IEEE International Conference on MFI '94. Multisensor Fusion and Integration for Intelligent Systems*: IEEE.
- BOYD, D.E. und B. KOLES, 2019. An introduction to the special issue “Virtual Reality in Marketing”: Definition, Theory and Practice [online]. *Journal of Business Research*, **100**, 441-444. ISSN 01482963. Verfügbar unter: doi:10.1016/j.jbusres.2019.04.023
- BRAY, B., 2021. *Was ist Mixed Reality? - Mixed Reality* [online]. 21. Januar 2021 [Zugriff am: 21. Januar 2021]. Verfügbar unter: <https://docs.microsoft.com/de-de/windows/mixed-reality/discover/mixed-reality>
- BRUDER, G., F. STEINICKE, K. ROTHAUS und K. HINRICHES, 2009. Enhancing presence in Head-Mounted Display environments by Visual Body Feedback using Head-Mounted cameras. In: *2009 International Conference on CyberWorlds*: IEEE, S. 43-50.
- BRYSON, M. und S. SUKKARIEH, 2005. *Bearing-only SLAM for an airborne vehicle* [online]. Verfügbar unter: https://www.researchgate.net/profile/salah-sukkarieh/publication/228754606_bearing-only_slam_for_an_airborne_vehicle
- BRYSON, S., 1993. Virtual reality in scientific visualization [online]. *Computers & Graphics*, **17**(6), 679-685. ISSN 00978493. Verfügbar unter: doi:10.1016/0097-8493(93)90117-R
- CADENA, C., L. CARLONE, H. CARRILLO, Y. LATIF, D. SCARAMUZZA, J. NEIRA, I. REID und J.J. LEONARD, 2016. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age [online]. *IEEE Transactions on Robotics*, **32**(6), 1309-1332. ISSN 1552-3098. Verfügbar unter: doi:10.1109/TRO.2016.2624754
- CAMPOS, C., R. ELVIRA, J.J.G. RODRIGUEZ, J.M. M. MONTIEL und J. D. TARDOS, 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM [online]. *IEEE Transactions on Robotics*, 1-17. ISSN 1552-3098. Verfügbar unter: doi:10.1109/TRO.2021.3075644

- CHEERS, L., 2021. *Unity-Robotics-Hub/2_ros_tcp.md at main · Unity-Technologies/Unity-Robotics-Hub* [online]. 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/pick_and_place/2_ros_tcp.md
- CHEN, L., W. TANG, N. JOHN, T.R. WAN und J.J. ZHANG, 2018a. *Context-Aware Mixed Reality: A Framework for Ubiquitous Interaction* [online]. Verfügbar unter: <http://arxiv.org/pdf/1803.05541v1>
- CHEN, Y., Y. ZHOU, Q. LV und K.K. DEVEERASETTY, 2018b. A Review of V-SLAM *. In: *2018 IEEE International Conference on Information and Automation (ICIA)*: IEEE, S. 603-608.
- CHIEN, H.-J., C.-C. CHUANG, C.-Y. CHEN und R. KLETTE, 2016. When to use what feature? SIFT, SURF, ORB, or A-KAZE features for monocular visual odometry. In: *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*: IEEE.
- CLIPP, B., J. LIM, J.-M. FRAHM und M. POLLEFEYS, 2010. Parallel, real-time visual SLAM. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*: IEEE, S. 3961-3968.
- ÇÖLTEKIN, A., I. LOCHHEAD, M. MADDEN, S. CHRISTOPHE, A. DEVAUX, C. PETTIT, O. LOCK, S. SHUKLA, L. HERMAN, Z. STACHOŇ, P. KUBÍČEK, D. SNOPKOVÁ, S. BERNARDES und N. HEDLEY, 2020. Extended Reality in Spatial Sciences: A Review of Research Challenges and Future Directions [online]. *ISPRS International Journal of Geo-Information*, 9(7), 439. ISPRS International Journal of Geo-Information. Verfügbar unter: doi:10.3390/ijgi9070439
- COSTANZA, E., A. KUNZ und M. FJELD, 2009. Mixed Reality: A Survey. In: D. LALANNE und J. KOHLAS, Hg. *Human Machine Interaction*. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 47-68.
- COUTRIX, C. und L. NIGAY, 2006. Mixed Reality. In: A. CELENTANO, Hg. *Proceedings of the working conference on advanced visual interfaces - AVI '06*. New York, New York, USA: ACM Press.

- CROUSE, D., H. HAN, D. CHANDRA, B. BARBELLLO und A.K. JAIN, 2015. Continuous authentication of mobile user: Fusion of face image and inertial Measurement Unit data. In: *2015 International Conference on Biometrics (ICB)*: IEEE, S. 135-142.
- CUI, L. und F. WEN, 2019. A monocular ORB-SLAM in dynamic environments [online]. *Journal of Physics: Conference Series*, **1168**, 52037. ISSN 1742-6588. Verfügbar unter: doi:10.1088/1742-6596/1168/5/052037
- DAVISON, A.J., I.D. REID, N.D. MOLTON und O. STASSE, 2007. MonoSLAM: real-time single camera SLAM [online]. *IEEE transactions on pattern analysis and machine intelligence*, **29**(6), 1052-1067. ISSN 0162-8828. Verfügbar unter: doi:10.1109/TPAMI.2007.1049
- DEHGHANI, M., S.H. LEE und A. MASHATAN, 2020. Touching holograms with windows mixed reality: Renovating the consumer retailing services [online]. *Technology in Society*, **63**, 101394. ISSN 0160791X. Verfügbar unter: doi:10.1016/j.techsoc.2020.101394
- DISCANT, A., A. ROGOZAN, C. RUSU und A. BENSRAIR, 2007. Sensors for Obstacle Detection - A Survey. In: *2007 30th International Spring Seminar on Electronics Technology (ISSE)*: IEEE, S. 100-105.
- DÖRNER, R., W. BROLL, P. GRIMM und B. JUNG, 2013. *Virtual und Augmented Reality (VR / AR)*. Berlin, Heidelberg: Springer Berlin Heidelberg.
- DURRANT-WHYTE, H. und T. BAILEY, 2006. Simultaneous localization and mapping: part I [online]. *IEEE Robotics & Automation Magazine*, **13**(2), 99-110. ISSN 1070-9932. Verfügbar unter: doi:10.1109/MRA.2006.1638022
- DURRANT-WHYTE, H., D. RYE und E. NEBOT, 2000. Localization of Autonomous Guided Vehicles. In: G. GIRALT und G. HIRZINGER, Hg. *Robotics Research*. London: Springer London, S. 613-625.
- ENGELS, C., H. STEWÉNIUS und D. NISTÉR, 2006. *Bundle adjustment rules* [online]. Verfügbar unter: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.6901&rep=rep1&type=pdf>
- EPIC GAMES, 2021. *Unity-Robotics-Hub/0_ros_setup.md at main · Unity-Techologies/Unity-Robotics-Hub* [online]. 6. September 2021 [Zugriff am: 6. September]

- 2021]. Verfügbar unter: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/pick_and_place/0_ros_setup.md
- EVANS, G., J. MILLER, M. IGLESIAS PENA, A. MACALLISTER und E. WINER, 2017. Evaluating the Microsoft HoloLens through an augmented reality assembly application. In: J.N. SANDERS-REED und J.J. ARTHUR, Hg. *Degraded Environments: Sensing, Processing, and Display 2017*: SPIE, 101970V.
- FARASIN, A., F. PECIAROLO, M. GRANGETTO, E. GIANARIA und P. GARZA, 2020. Real-time Object Detection and Tracking in Mixed Reality using Microsoft HoloLens. In: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications: SCITEPRESS - Science and Technology Publications*, S. 165-172.
- FILIPENKO, M. und I. AFANASYEV, 2018. Comparison of various SLAM systems for mobile robot in an indoor environment. In: *2018 International Conference on Intelligent Systems (IS)*: IEEE, S. 400-407.
- FOLKESSON, J.B. und H.I. CHRISTENSEN, 2004. Robust SLAM [online]. *IFAC Proceedings Volumes*, **37**(8), 722-727. ISSN 14746670. Verfügbar unter: doi:10.1016/S1474-6670(17)32064-5
- FRAUNDORFER, F. und D. SCARAMUZZA, 2012. Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications [online]. *IEEE Robotics & Automation Magazine*, **19**(2), 78-90. ISSN 1070-9932. Verfügbar unter: doi:10.1109/MRA.2012.2182810
- FRESE, U., 2010. Interview: Is SLAM Solved? [online]. *KI - Künstliche Intelligenz*, **24**(3), 255-257. ISSN 0933-1875. Verfügbar unter: doi:10.1007/s13218-010-0047-x
- GALVEZ-LÓPEZ, D. und J.D. TARDOS, 2012. Bags of Binary Words for Fast Place Recognition in Image Sequences [online]. *IEEE Transactions on Robotics*, **28**(5), 1188-1197. ISSN 1552-3098. Verfügbar unter: doi:10.1109/tro.2012.2197158
- GAUTIER, Q.K., T.G. GARRISON, F. RUSHTON, N. BOUCK, E. LO, P. TUELLER, C. SCHURGERS und R. KASTNER, 2020. Low-cost 3D scanning systems for cultural heritage documentation [online]. *Journal of Cultural Heritage Management and Sustainable*

Anhang

Development, **10**(4), 437-455. ISSN 2044-1266. Verfügbar unter: doi:10.1108/JCHMSD-03-2020-0032

GONZALEZ, R.C. und R.E. WOODS, 2007. *Digital image processing*. 3rd ed. Reading, Massachusetts: Addison-Wesley Publishing Company; Pearson Prentice Hall.

GRAESSLER, I. und P. TAPLICK, 2019. Supporting Creativity with Virtual Reality Technology [online]. *Proceedings of the Design Society: International Conference on Engineering Design*, **1**(1), 2011-2020. Proceedings of the Design Society: International Conference on Engineering Design. Verfügbar unter: doi:10.1017/dsi.2019.207

GREENWOLD, S., 2003. *Spatial computing* [online]. Verfügbar unter: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.4352&rep=rep1&type=pdf>

GREENWOLD, S., 2021. *Simon Greenwold* [online]. 12. Dezember 2003 [Zugriff am: 15. Juli 2021]. Verfügbar unter: <https://acg.media.mit.edu/people/simong/resume.html>

GRISETTI, G., R. KUMMERLE, C. STACHNISS und W. BURGARD, 2010. A Tutorial on Graph-Based SLAM [online]. *IEEE Intelligent Transportation Systems Magazine*, **2**(4), 31-43. ISSN 1939-1390. Verfügbar unter: doi:10.1109/MITS.2010.939925

HARTLEY, R. und A. ZISSELMAN, 2015. *Multiple view geometry in computer vision*. 2. ed., 13.pr. Cambridge: Cambridge Univ. Press.

HENEIN, M., J. ZHANG, R. MAHONY und V. ILA, 2020. Dynamic SLAM: The Need For Speed. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*: IEEE, S. 2123-2129.

HENTSCHEL, M. und B. WAGNER, 2011. An Adaptive Memory Model for Long-Term Navigation of Autonomous Mobile Robots [online]. *Journal of Robotics*, **2011**, 1-9. ISSN 1687-9600. Verfügbar unter: doi:10.1155/2011/506245

HOCHSCHULE RHEINMAIN, 2021. *Dörner, Ralf - Hochschule RheinMain* [online]. 27. Juni 2021 [Zugriff am: 27. Juni 2021]. Verfügbar unter: <https://www.hs-rm.de/de/hochschule/personen/doerner-ralf/>

HOCKING, J., 2015. *Unity in action. Multiplatform game development in C# with Unity 5* [online]. Verfügbar unter: <http://proquest.tech.safaribooksonline.de/9781617292323>

Anhang

- HOLLIES, R.C. und M.A. FISCHLER, 1981. *A RANSAC-based approach to model fitting and its application to finding cylinders in range data* [online]. Verfügbar unter: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.7836&rep=rep1&type=pdf>
- HUANG BAICHUAN, H., J. ZHAO und J. LIU, 2019. *A Survey of Simultaneous Localization and Mapping with an Envision in 6G Wireless Networks* [online]. Verfügbar unter: https://www.researchgate.net/profile/jun-zhao-36/publication/335754983_a_survey_of_simultaneous_localization_and_mapping_with_an_envision_in_6g_wireless_networks
- HUBEL, D.H. und T.N. WIESEL, 2005. *Brain and visual perception. The story of a 25-year collaboration* [online]. Oxford: Oxford Univ. Press. Verfügbar unter: <http://www.loc.gov/catdir/enhancements/fy0618/2004049553-d.html>
- HÜBNER, P., K. CLINTWORTH, Q. LIU, M. WEINMANN und S. WURSTHORN, 2020. Evaluation of HoloLens Tracking and Depth Sensing for Indoor Mapping Applications [online]. *Sensors (Basel, Switzerland)*, **20**(4). Sensors (Basel, Switzerland). Verfügbar unter: doi:10.3390/s20041021
- HULETSKI, A., D. KARTASHOV und K. KRINKIN, 2015. Evaluation of the modern visual SLAM methods. In: *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*: IEEE, S. 19-25.
- HUSSEIN, A., F. GARCIA und C. OLAVERRI-MONREAL, 2018. ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation. In: *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*: IEEE, S. 1-6.
- IEEE EXPLORE, 2021. *Fumio Kishino* [online]. 25. Juni 2021 [Zugriff am: 25. Juni 2021]. Verfügbar unter: https://ieeexplore.ieee.org/author/37284006700?history=no&sortType=newest&highlight=true&returnType=SEARCH&matchPubs=true&searchWithin=%22Author%20Ids%22:37284006700&ranges=1994_1994_Year&returnFacets=ALL
- JÄHNE, B., 2005. *Digitale Bildverarbeitung. Mit 155 Übungsaufgaben*. 6., überarb. und erw. Aufl. Berlin: Springer.

- JINYU, L., Y. BANGBANG, C. DANPENG, W. NAN, Z. GUOFENG und B. HUJUN, 2019. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality [online]. *Virtual Reality & Intelligent Hardware*, **1**(4), 386-410. ISSN 20965796. Verfügbar unter: doi:10.1016/j.vrih.2019.07.002
- KAEHLER, A. und G. BRADSKI, 2015. *Learning OpenCV 3. Computer Vision in C++ With the OpenCV Library*. s.l.: O'reilly & Associates Inc.
- KHOSHELHAM, K., H. TRAN und D. ACHARYA, 2019. INDOOR MAPPING EYEWEAR: GEOMETRIC EVALUATION OF SPATIAL MAPPING CAPABILITY OF HOLOLENS [online]. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **XLII-2/W13**, 805-810. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Verfügbar unter: doi:10.5194/isprs-archives-XLII-2-W13-805-2019
- KIM, D., S. CHAE, J. SEO, Y. YANG und T.-D. HAN, 2017. Realtime plane detection for projection Augmented Reality in an unknown environment. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*: IEEE, S. 5985-5989.
- KITT, B.M., JOERN REHDER, A.D. CHAMBERS, M. SCHONBEIN, H. LATEGAHN und S. SINGH, 2011. *Monocular Visual Odometry using a Planar Road Model to Solve Scale Ambiguity*.
- KLEIN, G. und D. MURRAY, 2007. Parallel Tracking and Mapping for Small AR Workspaces. In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*: IEEE, S. 1-10.
- KOUBAA, A., Hg., 2016. *Robot Operating System (ROS). The Complete Reference (Volume 1)*. Cham: Springer International Publishing. Studies in Computational Intelligence. 625.
- KRESS, B.C., 2020. *Optical architectures for augmented-, virtual-, and mixed-reality headsets*. Bellingham Washington: SPIE.
- KROMBACH, N., D. DROESCHEL, S. Houben und S. BEHNKE, 2018. Feature-based visual odometry prior for real-time semi-dense stereo SLAM [online]. *Robotics and Autonomous Systems*, **109**, 38-58. ISSN 09218890. Verfügbar unter: doi:10.1016/j.robot.2018.08.002

- LA VIOLA, J.J., B.M. WILLIAMSON, R. SOTTILARE und P. GARRITY, 2017. *Analyzing slam algorithm performance for tracking in augmented reality systems* [online]. Verfügbar unter: <https://www.eecs.ucf.edu/~jjl/pubs/17161.pdf>
- LALANNE, D. und J. KOHLAS, Hg., 2009. *Human Machine Interaction*. Berlin, Heidelberg: Springer Berlin Heidelberg. Lecture Notes in Computer Science.
- LI, L., Z. LIU, U. OZGINER, J. LIAN, Y. ZHOU und Y. ZHAO, 2018. Dense 3D Semantic SLAM of traffic environment based on stereo vision. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*: IEEE, S. 965-970.
- LI, R., S. WANG und D. GU, 2021. DeepSLAM: A Robust Monocular SLAM System With Unsupervised Deep Learning [online]. *IEEE Transactions on Industrial Electronics*, **68**(4), 3577-3587. ISSN 0278-0046. Verfügbar unter: doi:10.1109/TIE.2020.2982096
- LIU, C., K. KIM, J. GU, Y. FURUKAWA und J. KAUTZ, 2018a. *PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image* [online]. Verfügbar unter: https://www.researchgate.net/profile/chen-liu-135/publication/329588296_planercnn_3d_plane_detection_and_reconstruction_from_a_single_image
- LIU, H., G. ZHANG und H. BAO, 2016. Robust Keyframe-based Monocular SLAM for Augmented Reality. In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*: IEEE, S. 1-10.
- LIU, J., Y. XIE, S. GU und X. CHEN, 2020. A SLAM-Based Mobile Augmented Reality Tracking Registration Algorithm [online]. *International Journal of Pattern Recognition and Artificial Intelligence*, **34**(01), 2054005. ISSN 0218-0014. Verfügbar unter: doi:10.1142/S0218001420540051
- LIU, Y., H. DONG, L. ZHANG und A.E. SADDIK, 2018b. Technical Evaluation of HoloLens for Multimedia: A First Look [online]. *IEEE MultiMedia*, **25**(4), 8-18. ISSN 1070-986X. Verfügbar unter: doi:10.1109/MMUL.2018.2873473
- LOGITECH, 2021. *Logitech C270 HD Webcam, 720p-Videogespräche mit Geräuschunterdrückung* [online]. 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: <https://www.logitech.com/de-de/products/webcams/c270-hd-webcam.960-001063.html?gclid=EAIaIQobChMIPpz9u4Hq8gIV->

v3VCh2CPQ67EAAYAiAAEgLeifD_BwE&utm_source=google&utm_medium=paid_sear
h

LOVEGORVE, S., 2021. *Pangolin* [online]. *Pangolin is a lightweight portable rapid development library for managing OpenGL display / interaction and abstracting video input.* 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: <https://github.com/stevenlovegrove/Pangolin>

MILGRAM, P. und F. KISHINO, 1994. A Taxonomy of Mixed Reality Visual Displays [online]. *IEICE TRANSACTIONS on Information and Systems*, **E77-D**(12), 1321-1329. ISSN 0916-8532. Verfügbar unter: https://search.ieice.org/bin/summary.php?id=e77-d_12_1321

MILGRAM, P., H. TAKEMURA, A. UTSUMI und F. KISHINO, 1995. Augmented reality: a class of displays on the reality-virtuality continuum. In: H. DAS, Hg. *Telemanipulator and Telepresence Technologies*: SPIE, S. 282-292.

MILLER, D., 2021. *Unity-Robotics-Hub/1_urdf.md at main · Unity-Technologies/Unity-Robotics-Hub* [online]. 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/pick_and_place/1_urdf.md

MILLER, D., S. GIBSON und A. NAVARRO, 2021. Advance your robot autonomy with ROS 2 and Unity [online]. *Unity Blog* [Zugriff am: 2. September 2021]. Verfügbar unter: <https://blog.unity.com/manufacturing/advance-your-robot-autonomy-with-ros-2-and-unity>

MISHRA, B., R. GRIFFIN und H.E. SEVIL, 2021. Modelling Software Architecture for Visual Simultaneous Localization and Mapping [online]. *Automation*, **2**(2), 48-61. Automation. Verfügbar unter: doi:10.3390/automation2020003

MOEZzi, R., D. KRCMARIK, J. HLAVA und J. CÝRUS, 2020. Hybrid SLAM modelling of autonomous robot with augmented reality device [online]. *Materials Today: Proceedings*, **32**, 103-107. ISSN 22147853. Verfügbar unter: doi:10.1016/j.matpr.2020.03.036

MONTEMERLO, M., S. THRUN, D. KOLLER und B. WEGBREIT, 2003. FastSLAM 2.0 : An improved particle filtering algorithm for simultaneous localization and mapping that provably converges [online]. *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI-03)*. Proc. Int. Joint Conf. Artificial Intelligence (IJCAI-03). Verfügbar unter: <https://ci.nii.ac.jp/naid/10013042851/>

MSI.COM, 2021. *GV72 8RE* [online]. 6. September 2021 [Zugriff am: 6. September 2021].

Verfügbar unter: <https://de.msi.com/Laptop/GV72-8RE>

MUR-ARTAL, R. und J.D. TARDOS, 2017. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras [online]. *IEEE Transactions on Robotics*, **33**(5), 1255-1262. ISSN 1552-3098. Verfügbar unter:
doi:10.1109/TRO.2017.2705103

MUR-ARTAL, R., J.M.M. MONTIEL und J.D. TARDOS, 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System [online]. *IEEE Transactions on Robotics*, **31**(5), 1147-1163. ISSN 1552-3098. Verfügbar unter: doi:10.1109/TRO.2015.2463671

NAYAR, S.K., 2021. *First Principles of Computer Vision* [online]. 5. August 2021 [Zugriff am: 5. August 2021]. Verfügbar unter: <https://fpcv.cs.columbia.edu/>

NOOR, A.K., 2016. The Hololens Revolution [online]. *Mechanical Engineering*, **138**(10), 30-35. ISSN 0025-6501. Verfügbar unter: doi:10.1115/1.2016-Oct-1

OKADA, K., 2021. *ros-drivers/usb_cam: A ROS Driver for V4L USB Cameras* [online]. 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: https://github.com/ros-drivers/usb_cam

OLEARI, F., D. LODI RIZZINI und S. CASELLI, 2013. A low-cost stereo system for 3D object recognition. In: *2013 IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP)*: IEEE, S. 127-132.

OPEN ROBOTICS, 3 September 2021, 12:00. *noetic/Installation/Ubuntu - ROS Wiki* [online] [Zugriff am: 3. September 2021]. Verfügbar unter:
<http://wiki.ros.org/noetic/Installation/Ubuntu>

OUERGHI, S., N. RAGOT, R. BOUTTEAU und X. SAVATIER, 2020. Comparative Study of a commercial tracking camera and ORB-SLAM2 for person localization [online]. In: SCITEPRESS - Science and Technology Publications, S. 357-364 [Zugriff am: 8. Mai 2020]. Verfügbar unter: <https://hal.archives-ouvertes.fr/hal-02567816/>

OZOG, P. und R.M. EUSTICE, 2013. On the importance of modeling camera calibration uncertainty in visual SLAM. In: *2013 IEEE International Conference on Robotics and Automation*: IEEE, S. 3777-3784.

- PARK, K.-B., S.H. CHOI, M. KIM und J.Y. LEE, 2020. Deep learning-based mobile augmented reality for task assistance using 3D spatial mapping and snapshot-based RGB-D data [online]. *Computers & Industrial Engineering*, **146**, 106585. ISSN 03608352.
Verfügbar unter: doi:10.1016/j.cie.2020.106585
- PETERS, J.F., 2017. *Foundations of computer vision. Computational geometry, visual image structures and object shape detection* [online]. Cham: Springer. Intelligent systems reference library. 124. Verfügbar unter: <http://www.springer.com/>
- PIAO, J.-C. und S.-D. KIM, 2017. Adaptive Monocular Visual-Inertial SLAM for Real-Time Augmented Reality Applications in Mobile Devices [online]. *Sensors (Basel, Switzerland)*, **17**(11). Sensors (Basel, Switzerland). Verfügbar unter: doi:10.3390/s17112567
- PIERZCHAŁA, M., P. GIGUÈRE und R. ASTRUP, 2018. Mapping forests using an unmanned ground vehicle with 3D LiDAR and graph-SLAM [online]. *Computers and Electronics in Agriculture*, **145**, 217-225. ISSN 01681699. Verfügbar unter:
doi:10.1016/j.compag.2017.12.034
- REITMAYR, G., T. LANGLOTZ, D. WAGNER, A. MULLONI, G. SCHALL, D. SCHMALSTIEG und Q. PAN, 2010. Simultaneous Localization and Mapping for Augmented Reality. In: *2010 International Symposium on Ubiquitous Virtual Reality*: IEEE, S. 5-8.
- ROKHSARITALEMI, S., A. SADEGHI-NIARAKI und S.-M. CHOI, 2020. A Review on Mixed Reality: Current Trends, Challenges and Prospects [online]. *Applied Sciences*, **10**(2), 636. Applied Sciences. Verfügbar unter: doi:10.3390/app10020636
- RUBLEE, E., V. RABAUD, K. KONOLIGE und G. BRADSKI, 2011. ORB: An efficient alternative to SIFT or SURF. In: *2011 International Conference on Computer Vision*: IEEE, S. 2564-2571.
- RUSER, H. und F. PUENTE LEÓN, 2007. Informationsfusion – Eine Übersicht (Information Fusion – An Overview) [online]. *tm – Technisches Messen*, **74**(3). ISSN 0171-8096. Verfügbar unter: doi:10.1524/teme.2007.74.3.93
- SATTLER, A., 2021. *Virtual-Reality-Labor* [online]. 6. Juli 2021 [Zugriff am: 6. Juli 2021].
Verfügbar unter: <https://www.mpib-berlin.mpg.de/institut/labore/vr-labor>

- SCARAMUZZA, D. und F. FRAUNDORFER, 2011. Visual Odometry [Tutorial] [online]. *IEEE Robotics & Automation Magazine*, **18**(4), 80-92. ISSN 1070-9932. Verfügbar unter: doi:10.1109/MRA.2011.943233
- SCHMIDT, U., 2013. *Professionelle Videotechnik. Grundlagen, Filmtechnik, Fernsehtechnik, Geräte- und Studiotechnik in SD, HD, DI, 3D.* 6. Aufl. Berlin: Springer Vieweg.
- SCHREER, O., 2005. *Stereoanalyse und Bildsynthese. Mit 6 Tabellen.* Berlin: Springer.
- SCHUON, S., C. THEOBALT, J. DAVIS und S. THRUN, 2008. High-quality scanning using time-of-flight depth superresolution. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*: IEEE, S. 1-7.
- SERAFIN, S., M. GERONAZZO, C. ERKUT, N.C. NILSSON und R. NORDAHL, 2018. Sonic Interactions in Virtual Reality: State of the Art, Current Challenges, and Future Directions [online]. *IEEE computer graphics and applications*, **38**(2), 31-43. IEEE computer graphics and applications. Verfügbar unter: doi:10.1109/MCG.2018.193142628
- SMITH, R.C. und P. CHEESEMAN, 1986. On the Representation and Estimation of Spatial Uncertainty [online]. *The International Journal of Robotics Research*, **5**(4), 56-68. ISSN 0278-3649. Verfügbar unter: doi:10.1177/027836498600500404
- SNOWDEN, R.J., P. THOMPSON und T. TROSCIANKO, 2012. *Basic vision. An introduction to visual perception.* Revised edition. Oxford: Oxford University Press.
- SOMMERVILLE, I., 2016. *Software engineering.* Tenth edition, global edition. Boston: Pearson. Always learning.
- STACHNISS, C., 2009. *Robotic mapping and exploration.* Berlin: Springer. Springer tracts in advanced robotics. Vol. 55.
- STACHNISS, C., J.J. LEONARD und S. THRUN, 2016. Simultaneous Localization and Mapping. In: B. SICILIANO und O. KHATIB, Hg. *Springer Handbook of Robotics.* Cham: Springer International Publishing, S. 1153-1176.
- STRASDAT, H., J. M. M. MONTIEL und A. DAVISON, 2011. Scale Drift-Aware Large Scale Monocular SLAM. In: Y. MATSUOKA, H.F. DURRANT-WHYTE und J. NEIRA, Hg. *Robotics. Science and systems VI.* Cambridge, Mass: MIT Press.

- STURM, J., 2021. *Computer Vision Group - Submission form for automatic evaluation of RGB-D SLAM results* [online]. 9. September 2021 [Zugriff am: 9. September 2021]. Verfügbar unter: https://vision.in.tum.de/data/datasets/rbgd-dataset/online_evaluation
- STURM, J., N. ENGELHARD, F. ENDRES, W. BURGARD und D. CREMERS, 2012. A benchmark for the evaluation of RGB-D SLAM systems. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*: IEEE, S. 573-580.
- SUALEH, M. und G.-W. KIM, 2019. Simultaneous Localization and Mapping in the Epoch of Semantics: A Survey [online]. *International Journal of Control, Automation and Systems*, **17**(3), 729-742. ISSN 1598-6446. Verfügbar unter: doi:10.1007/s12555-018-0130-x
- SUTHERLAND, I.E., 1965. *The ultimate display* [online]. Verfügbar unter: <http://papers.cumincad.org/cgi-bin/works/browsedtreefield=seriesorder=az/show?c58e>
- SUTHERLAND, I.E., 1968. A head-mounted three dimensional display. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*. New York, New York, USA: ACM Press, S. 757.
- TAHERI, H. und Z.C. XIA, 2021. SLAM; definition and evolution [online]. *Engineering Applications of Artificial Intelligence*, **97**, 104032. ISSN 09521976. Verfügbar unter: doi:10.1016/j.engappai.2020.104032
- TAKETOMI, T., H. UCHIYAMA und S. IKEDA, 2017. Visual SLAM algorithms: a survey from 2010 to 2016 [online]. *IPSJ Transactions on Computer Vision and Applications*, **9**(1). IPSJ Transactions on Computer Vision and Applications. Verfügbar unter: doi:10.1186/s41074-017-0027-2
- TANG, B. und S. CAO, 2020. A review of VSLAM technology applied in Augmented Reality [online]. *IOP Conference Series: Materials Science and Engineering*, 42014. IOP Conference Series: Materials Science and Engineering. Verfügbar unter: doi:10.1088/1757-899X/782/4/042014
- THALLER, L., 2021. *A ROS implementation of ORB_SLAM2* [online]. 6. September 2021 [Zugriff am: 6. September 2021]. Verfügbar unter: https://github.com/appliedAI-Initiative/orb_slam_2_ros
- THRUN, S., W. BURGARD und D. FOX, 2005. *Probabilistic robotics*. Cambridge, Mass.: MIT Press. Intelligent robotics and autonomous agents.

- TIAN, Y., L. LI, A. FUMAGALLI, Y. TADESSE und B. PRABHAKARAN, 2021. *Haptic-enabled Mixed Reality System for Mixed-initiative Remote Robot Control* [online]. Verfügbar unter: <http://arxiv.org/pdf/2102.03521v2>
- UNIV. TORONTO, 2021. *Paul Milgram - Department of Mechanical & Industrial Engineering* [online]. 14. Juni 2021 [Zugriff am: 2. Juli 2021]. Verfügbar unter: https://www.mie.utoronto.ca/faculty_staff/milgram/
- VLAMINCK, M., H. LUONG und W. PHILIPS, 2017. A markerless 3D tracking approach for augmented reality applications. In: *2017 International Conference on 3D Immersion (IC3D). Proceedings : 11-12 December 2017, Brussels, Belgium*. Piscataway, NJ: IEEE.
- WANG, S., Z. WU und W. ZHANG, 2019. An Overview of SLAM. In: Y. JIA, J. DU und W. ZHANG, Hg. *Proceedings of 2018 Chinese Intelligent Systems Conference*. Singapore: Springer Singapore, S. 673-681.
- WEINMANN, M., M.A. JÄGER, S. WURSTHORN, B. JUTZI und P. HÜBNER, 2020. 3D INDOOR MAPPING WITH THE MICROSOFT HOLOLENS: QUALITATIVE AND QUANTITATIVE EVALUATION BY MEANS OF GEOMETRIC FEATURES [online]. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **V-1-2020**, 165-172. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Verfügbar unter: doi:10.5194/isprs-annals-V-1-2020-165-2020
- WHELAN, T., M. KAESZ, H. JOHANSSON, M. FALLON, J.J. LEONARD und J. McDONALD, 2015. Real-time large-scale dense RGB-D SLAM with volumetric fusion [online]. *The International Journal of Robotics Research*, **34**(4-5), 598-626. ISSN 0278-3649. Verfügbar unter: doi:10.1177/0278364914551008
- WIKIPEDIA, 2021a. *Hugh F. Durrant-Whyte* [online]. 3. April 2021 [Zugriff am: 25. Juni 2021]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Hugh_F._Durrant-Whyte&oldid=1015738613
- WIKIPEDIA, 2021b. *Ivan Sutherland – Wikipedia* [online]. 27. Juni 2021 [Zugriff am: 6. Juli 2021]. Verfügbar unter: https://de.wikipedia.org/wiki/Ivan_Sutherland
- WIKIPEDIA, 2021c. *Windows Mixed Reality* [online]. 16. Mai 2021 [Zugriff am: 23. Juli 2021]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Windows_Mixed_Reality&oldid=212053542

- WILHELM BURGER, 2016. *Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation* [online]. Verfügbar unter: https://www.researchgate.net/profile/wilhelm-burger/publication/303233579_zhang's_camera_calibration_algorithm_in-depth_tutorial_and_implementation
- WILLIAMS, B. und I. REID, 2010. On combining visual SLAM and visual odometry. In: *2010 IEEE International Conference on Robotics and Automation*: IEEE, S. 3494-3500.
- WOJO, M., 2021. *Windows Mixed Reality VR-Dokumentation - Enthusiast Guide* [online]. 12. Mai 2021 [Zugriff am: 12. Mai 2021]. Verfügbar unter: https://docs.microsoft.com/de-de/windows/mixed-reality/enthusiast-guide/?WT.mc_id=mixedreality_product
- WULF, O., A. NUCHTER, J. HERTZBERG und B. WAGNER, 2007. Ground truth evaluation of large urban 6D SLAM. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*: IEEE, S. 650-657.
- YANG, G., Y. WANG, J. ZHI, W. LIU, Y. SHAO und P. PENG, 2020. A Review of Visual Odometry in SLAM Techniques. In: *2020 International Conference on Artificial Intelligence and Electromechanical Automation (AIEA)*: IEEE.
- YANG, Q., K.-H. TAN, B. CULBERTSON und J. APOSTOLOPOULOS, 2010. Fusion of active and passive sensors for fast 3D capture. In: *2010 IEEE International Workshop on Multimedia Signal Processing*: IEEE, S. 69-74.
- YING YANG, M. und W. FÖRSTNER, 2010. *Plane detection in point cloud data* [online]. Verfügbar unter: <https://ris.utwente.nl/ws/files/103953896/yang2010plane.pdf>
- YONG-BAO, A., R. TING, Y. XIAO-QIANG, H. JIA-LIN, F. LEI, L. JIAN-BIN und L. MING, 2020. Visual SLAM in dynamic environments based on object detection [online]. *Defence Technology*. ISSN 22149147. Verfügbar unter: doi:10.1016/j.dt.2020.09.012
- ZELLER, M., 2021. *Räumliche Abbildung - Mixed Reality* [online]. 15. Juli 2021 [Zugriff am: 15. Juli 2021]. Verfügbar unter: <https://docs.microsoft.com/de-de/windows/mixed-reality/design/spatial-mapping>
- ZHANG, Z., 2000. A flexible new technique for camera calibration [online]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(11), 1330-1334. ISSN 0162-8828. Verfügbar unter: doi:10.1109/34.888718

Anhang

ZHAO, Y. und P.A. VELA, 2020. Good Feature Matching: Toward Accurate, Robust VO/VSLAM With Low Latency [online]. *IEEE Transactions on Robotics*, **36**(3), 657-675. ISSN 1552-3098. Verfügbar unter: doi:10.1109/TRO.2020.2964138

Abbildungsverzeichnis

Abbildung 1: Reality-Virtuality Continuum in Anlehnung an (Milgram et al., 1995)	16
Abbildung 2: Beispiel einer VR-Anwendung (Sattler, 2021).....	17
Abbildung 3: Haptic Snake (Al-Sada et al., 2020).....	19
Abbildung 4: Beispiel Augmented Virtuality (Bruder et al., 2009).....	21
Abbildung 5: Beispiel Augmented Reality (Dörner et al., 2013).....	21
Abbildung 6: "Sword of Damocles" (<i>Sutherland, 1968</i>).....	24
Abbildung 7: Video see-through Display in Anlehnung an (Dörner et al., 2013)	25
Abbildung 8: Optical see-through Display in Anlehnung an (Dörner et al., 2013)	26
Abbildung 9: Das SLAM-Problem in Anlehnung an (Durrant-Whyte und Bailey, 2006)	27
Abbildung 10: CMOS-Sensor in Anlehnung an (Gonzalez und Woods, 2007, Schreer, 2005)	30
Abbildung 11: Beispielhafte Darstellung eines Bildsensors in Anlehnung an (Hartley und Zisserman, 2015, Gonzalez und Woods, 2007)	30
Abbildung 12: Beispiel einer Kamera in Anlehnung an (Schreer, 2005)	31
Abbildung 13: Dreidimensionale Projektion auf die Bildebene in Anlehnung an (Schreer, 2005, Hartley und Zisserman, 2015).....	32
Abbildung 14: Schematische Darstellung Lochkamera (Schreer, 2005)	33
Abbildung 15: Darstellung der Kamerakalibrierung (Kaehler und Bradski, 2015)	35
Abbildung 16: Lineares Kameramodell in Anlehnung an (Hartley und Zisserman, 2015, Nayar, 2021).....	36
Abbildung 17: Projektion von Objektkoordinaten in Anlehnung an (Nayar, 2021, Hartley und Zisserman, 2015).....	36
Abbildung 18: Maker für visuelles Tracking (Dörner et al., 2013)	40
Abbildung 19: Interaktion zwischen Mensch, Computern und der Umgebung in Anlehnung an (Bray, 2021)	46
Abbildung 20: Kamerasysteme der HoloLens (Khoshelham, Tran und Acharya, 2019)	48
Abbildung 21: Übersicht der Durch die HoloLens generierten Frames (Hübner et al., 2020)	48
Abbildung 22: Schema vSLAM-Algorithmen in Anlehnung an (Sualeh und Kim, 2019)	55
Abbildung 23: Darstellung der Problemstellung der Visuellen Odometrie (Fraundorfer und Scaramuzza, 2012).....	57

Abbildungsverzeichnis

Abbildung 24: Komponenten eines VO-Systems (Scaramuzza und Fraundorfer, 2011)	58
Abbildung 26: Systemaufbau in Anlehnung an (Thaller, 2021, Koubaa, 2016, Hussein, Garcia und Olaverri-Monreal, 2018, Epic Games, 2021)	64
Abbildung 29: Beispiele Merkmalsextraktion ORB	66
Abbildung 30: Übersicht ORB-SLAM2 (Mur-Artal und Tardos, 2017)	67
Abbildung 31: Logitech C270 (Logitech, 2021).....	69
Abbildung 32: Aufnahmen für die Kamerakalibrierung	71
Abbildung 33: ROS-Unity Kommunikation in Anlehnung an (Cheers, 2021)	75
Abbildung 34: Testaufbau Entfernungstest.....	81
Abbildung 39: Fehlgeschlagene Initialisierungen.....	83
Abbildung 40: ORB-SLAM2-Nativ Entfernungstest.....	86
Abbildung 41: ORB-SLAM-ROS Entfernungstest.....	87
Abbildung 42: Ergebnis ORB-SLAM2-Nativ Desk-Mapping.....	87
Abbildung 43: ORB-SLAM-Nativ Desk-Mapping.....	88
Abbildung 44: ORB-SLAM-Nativ Rekonstruktion	88
Abbildung 45: ORB-SLAM-ROS Rekonstruktion	89
Abbildung 46: ORB-SLAM2-Nativ Rotation	89
Abbildung 47: ORB-SLAM2-ROS Rotation	90
Abbildung 48: ORB-SLAM-Nativ RPY-Live	91
Abbildung 49: ORB-SLAM-Native XYZ-Live	91
Abbildung 50: Verankerung des virtuellen Objekts.....	92
Abbildung 51: Änderung der Perspektive	92
Abbildung 52: Map-Points der initialisierten Karte	93
Abbildung 53: Objektscann.....	93
Abbildung 54: Map-points in der Unity Engine	94
Abbildung 55: Szene nach Translation und Rotation der Kamera.....	94

Tabellenverzeichnis

Tabelle 1: Relevante Hardwarekonfiguration des Testsystems (msi.com, 2021)	70
Tabelle 2: Ergebnisse TUM-RPY-Dataset.....	84
Tabelle 3: Ergebnisse TUM-XYZ-Dataset	85
Tabelle 4: Vergleichsdaten (Liu, Zhang und Bao, 2016, Liu et al., 2020).....	85

Listingverzeichnis

Listing 1: Änderungen an ORB-SLAM2 AR-Demo – Image Capturing.....	72
Listing 2: Änderungen and ORB_SLAM2 AR-Demo	73
Listing 3: USB-Kamera Launch-File	74
Listing 4: Unity Script Subscriber Lokationsdaten.....	77

Formelverzeichnis

Formel 1: Zentralprojektion (Hartley und Zisserman, 2015, Nayar, 2021, Schreer, 2005) ...	33
Formel 2: Relation Welt- zu Kamerakoordinaten (Hartley und Zisserman, 2015, Nayar, 2021, Schreer, 2005)	34
Formel 3: Umrechnung Pixel in mm für u (Hartley und Zisserman, 2015).....	37
Formel 4: Umrechnung Pixel in mm für v (Hartley und Zisserman, 2015).....	37
Formel 5: Vereinfachung von Formel 3 (Hartley und Zisserman, 2015).....	37
Formel 6: Vereinfachung von Formel 4 (Hartley und Zisserman, 2015).....	37
Formel 7: Formel 3 und 4 mit homogenisierten Koordinaten.....	37
Formel 8: Kalibrierungsmatrix (Hartley und Zisserman, 2015).....	38
Formel 9: Intrinsische Matrix (Hartley und Zisserman, 2015)	38
Formel 10: Extrinsische Matrix (Hartley und Zisserman, 2015)	38
Formel 11: Formel 10 vereinfacht (Hartley und Zisserman, 2015)	38
Formel 12: Zusammenfassung von intrinsischer und extrinsischer Matrix (Hartley und Zisserman, 2015).....	39
Formel 13: Projektionsmatrix (Hartley und Zisserman, 2015)	39

Abkürzungsverzeichnis

AR	-	Augmented Reality
AV	-	Augmented Virtuality
bspw.	-	beispielsweise
bzw.	-	beziehungsweise
CMOS	-	complementary metal oxide semiconductor
d.h.	-	das heißt
et al.	-	und andere
FOV	-	Field of View
HMD	-	Head-mounted Display
i.d.R.	-	in der Regel
IMU	-	Internal Measurement Unit
LIDAR	-	light detection and ranging
MR	-	Mixed Reality
MS	-	Microsoft
OpenCV	-	Open Computer Vision Library
OS	-	Betriebssystem
RGB	-	Red Green Blue
RGB-D	-	Red Green Blue Depth
RMSE	-	absolute translation root-mean-square error
RV-Continuum	-	Reality Virtuality Continuum
SLAM	-	Simultaneous Localization and Mapping
sog.	-	sogenannt
ToF	-	Time of Flight Camera
u.U.	-	unter Umständen
vgl.	-	Vergleich
viSLAM	-	visual inertial Simultaneous Localization and Mapping
VO	-	Visual Odometry
VR	-	Virtual Reality
vSLAM	-	visual Simultaneous Localization and Mapping
Windows-MR	-	Windows Mixed Reality