

Bauhaus-Universität Weimar  
Fakultät Medien  
Studiengang Medieninformatik

# Don't SLAM your head

## VR in arbitrary environments

Bachelorarbeit

Joshua Reibert  
geb. am 9. September 1990 in Flensburg

Matrikelnummer 110773

1. Gutachter: Jun.-Prof. Dr. Florian Echtler
2. Gutachter: Prof. Dr. Volker Rodehorst

Datum der Abgabe: 4. April 2016

## **Zusammenfassung**

Virtuelle Realität ist wieder im Trend und leichter zugänglich als jemals zuvor. Genaue mobile VR mit Smartphones ist günstig verfügbar und bereits kabellos. Gleichzeitig sind SLAM-Algorithmen inzwischen effizient genug für den Mobilbereich.

Diese Arbeit implementiert den LSD-SLAM als Android-Plugin für die Spiele-Engine Unity. Während der Nutzer virtuelle Welten erforscht, sammelt der SLAM im Hintergrund Umgebungsinformationen. Damit werden der Nutzer getrackt und eine Punktwolke der Umgebung angezeigt.

Die Auswertung zeigt das Potenzial der Kombination aus SLAM und VR. Diese erlaubt ein flexibles Tracking und erstellt dabei detaillierte Punktwolken. Genauigkeit und Performance werden den Ansprüchen noch nicht gerecht. Vorwissen und die Integration von weiteren Sensoren können das aber in Zukunft ändern.

# Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, 4. April 2016

---

Joshua Reibert

# Inhaltsverzeichnis

1 Einleitung	5
2 Verwandte Arbeiten	9
2.1 Simultaneous Localization and Mapping . . . . .	10
2.2 Virtual Reality . . . . .	14
3 LSD-SLAM	17
3.1 Tracking . . . . .	18
3.2 Depth Map Estimation . . . . .	20
3.3 Map Optimization . . . . .	21
4 Eigene Umsetzung	22
4.1 Desktop-Anwendung . . . . .	22
4.2 Android-Port . . . . .	28
4.3 Unity-Plugin . . . . .	36
5 Auswertung und Ausblick	48
6 Fazit	56
Literatur	59

# 1 Einleitung

Aus Unzufriedenheit mit bisheriger Virtual-Reality-Hardware baute der 18-jährige Palmer Luckey 2011 in der Garage seiner Eltern ein eigenes Head-Mounted Display (HMD) [25]. Im August 2012 wurde sein Crowdfunding-Projekt Oculus Rift innerhalb weniger Stunden erfolgreich finanziert und brachte neuen Schwung in das Thema Virtual Reality.

In der Virtual Reality (VR) geht es um Immersion, also das Eintauchen des Nutzers in die digitale Welt. Dazu werden die Sinne künstlich so realistisch wie möglich angesprochen. Das Bild wird stereoskopisch und der Klang räumlich [21]. Das Sehen übernimmt dabei zunächst die wichtigste Rolle. Während konventionelle Displays ein statisches Fenster in die virtuelle Welt sind, ermöglichen es Head-Mounted Displays dem Nutzer, sich frei in der virtuellen Welt umzuschauen.

Von dieser Immersion profitieren zahlreiche Anwendungen. Im Militär werden Simulatoren schon länger als günstige Möglichkeit verwendet, um z. B. Piloten oder Fallschirmspringer auszubilden. Architektur, CAD und ganz allgemein 3D-Modellierung profitieren vom Plastischen der virtuellen Umgebung [1]. Im Bildungsbereich können historische Stätten, fremde Planeten und atomare Strukturen erkundet werden [37]. Sogar bei medizinischen Behandlungen hilft VR: Durch die Darstellung der fehlenden Gliedmaßen von Amputierten, können Phantomschmerzen gelindert werden [7]. Gerade günstigere Hardware öffnet die VR auch für den Consumer-Bereich. So bietet Youtube beispielsweise seit vergangenem Jahr eine Funktion an, die es Nutzern erlaubt, sich mit HMDs 360°-Videos anzusehen [35].

Die Oculus Rift brachte neben Verbesserungen, die insbesondere das Sichtfeld und die Sensoren betreffen, vor allem erschwingliche VR-Hardware. Das erste Development Kit (DK) der Oculus Rift kostete \$300 [18]. Damit ist high-end VR-Hardware nicht nur Forschern und Unternehmen zugänglich. Auch interessierte Privatpersonen können sich die Hardware leisten und eigene Anwendungen entwickeln.

## 1 Einleitung



(a) Die Frontseite des HMD-Gehäuses.

(b) Die Innenansicht mit den beiden Linsen.

Abbildung 1.1: Das zweite Development Kit der Oculus Rift. Eigene Abbildungen

Google stellte 2014 mit dem Cardboard eine Pappkonstruktion vor, die zusammen mit einem Smartphone als Head-Mounted Display funktioniert [27]. Dabei handelt es sich um eine Anleitung zum Eigenbau: Jeder Besitzer eines Smartphones kann sich mit etwas Geschick, einem Paar Linsen für ca. zwei Euro und Pappe für den Korpus einen eigenen Cardboard-Viewer bauen. Auch Oculus VR, die Firma hinter der Oculus Rift, bietet in Kooperation mit Samsung mit dem Gear VR ein Smartphone-getriebenes HMD an [24].

Da ein Smartphone häufig schon vorhanden ist, ermöglicht es zusammen mit einem Cardboard-Viewer einen leichten und kostengünstigen Einstieg in die VR. Dank zunehmender Leistung eignen sich Smartphones immer besser als HMD, denn das Rendern für HMDs ist anspruchsvoll: Es erfordert in Stereo, also einmal pro Auge, zu rendern und die von der Linse verursachte Verzeichnung durch das Anwenden des inversen Effekts aufzuheben.

Die auch dadurch entstehende Latenz kann erheblich zu Cybersickness beitragen, einem der größten Probleme der VR. Diese äußert sich als Übelkeit, Augenschmerzen und Schwindelgefühl bei längerem Konsum von VR [20]. Sie ist bislang eines der größten Hindernisse für die Akzeptanz von VR. Die genaue Ursache ist bisher nicht nachgewiesen. Nach der gängigsten Theorie, der sensorischen Konflikttheorie,

## 1 Einleitung



(a) „Dive“ von Durovis mit stabilerem Plastikkorpus und verschiebbaren Linsen      (b) „POP! 2.0“ von Mr Cardboard aus Pappe

Abbildung 1.2: Zwei Google Cardboard-Viewer mit eingesetztem Nexus 5. Eigene Abbildungen

wird Cybersickness durch den Konflikt zwischen dem Gleichgewichts- und dem visuellen System ausgelöst [20]. Während ersteres keine Bewegung empfindet, nimmt das Auge die Bewegungen der virtuellen Umgebung wahr. Wenn der Nutzer sich frei bewegen kann und dabei getrackt wird, würde das diesen Sinneskonflikt reduzieren.

Einen Schritt in diese Richtung machte das zweite Development Kit der Oculus Rift mit seinem integrierten Positionstracking. Eine Infrarotkamera muss so angebracht werden, dass sie die Infrarot-LEDs im Gehäuse der Oculus Rift erfasst. Dadurch ist es möglich, auch die Position der Oculus Rift im Raum zu tracken. Allerdings funktioniert das nur in einem kleinen Bereich und so lange der Nutzer in Richtung der Infrarotkamera schaut. Auch die notwendige Kabelverbindung der Oculus zum Rechner schränkt den Nutzer ein und erlaubt nur einen kleinen Bewegungsradius.

Cables are going to be a major obstacle in the VR industry for a long time. Mobile VR will be successful long before PC VR goes wireless.

— Palmer Luckey <sup>1</sup>

---

<sup>1</sup><https://twitter.com/PalmerLuckey/status/660967174597545985>

## 1 Einleitung

Smartphones hingegen sind unabhängig von Kabeln. Doch das allein ermöglicht keine freie Bewegung. Zum einen fehlt ein System zum Positionstracking und zum anderen stellen Hindernisse wie z. B. Möbelstücke in der Umgebung eine Gefahr dar, denn sie sind für den Nutzer in der virtuellen Welt nicht sichtbar.

Eine mögliche Lösung ist in jedem Smartphone bereits integriert: Die nach hinten gerichtete Kamera kann auch während einer laufenden Anwendung Fotos aufnehmen und dadurch Informationen über die Umgebung liefern. Visuelle SLAM-Algorithmen (*simultaneous localization and mapping*) tun genau das: Sie verwenden Bildinformationen, um eine Karte der Umgebung zu erstellen und die eigene Position und Orientierung darin zu ermitteln.

In der vorliegenden Arbeit sollen SLAM und VR in Form eines Plugins für die weit verbreitete Spiel-Engine Unity zusammengebracht werden. Ein SLAM-Algorithmus soll dabei über Kamerabilder des Smartphones Positionstracking und Umgebungsinformationen in Form einer Punktewolke liefern.

## 2 Verwandte Arbeiten

Erst sehr wenige wissenschaftliche Arbeiten verknüpfen SLAM-Algorithmen und VR. Im Bereich der Augmented Reality (AR) gibt es allerdings bereits Ansätze für das Tracking von Smartphones mit SLAM-Techniken. Dort ist die Genauigkeit nicht ganz so entscheidend wie in der VR, weil der virtuelle den realen Inhalt nur ergänzt. In der VR hingegen wird der Nutzer vollständig in die virtuelle Welt hineinversetzt und daher ist es umso kritischer, dass diese Illusion nicht dadurch gestört wird, dass Bewegungen in der realen Welt zu langsam in die virtuelle Umgebung übersetzt werden.

Für die vorliegende Arbeit wird ein SLAM-Algorithmus benötigt, der diese nötige Genauigkeit für VR-Inhalte liefert. Zusätzlich zum Tracking sollen die Bildinformationen des SLAM-Algorithmus auch verwendet werden, um Umgebungsinformationen zu gewinnen. In Abschnitt 2.1 werden einige SLAM-Algorithmen auf ihre Eignung untersucht.

Im Bereich der VR wird schon seit einiger Zeit am Problem der freien Bewegung geforscht. In Abschnitt 2.2 werden einige Methoden und auch Hardware vorgestellt, die dazu dienen, die Umgebung besser auszunutzen beziehungsweise Beschränkungen durch die Umgebung zu umgehen.

Außerdem liefert die Forschung Ansätze, wie gewonnene Informationen in der virtuellen Umgebung genutzt werden können. Der SLAM liefert Informationen über die Umgebung an die VR-Anwendung. Diese nutzt die Informationen einerseits, um den Nutzer in der realen Welt zu orten. Andererseits kann die Anwendung auch auf die Umgebung reagieren und den Nutzer darin steuern. Wenn der beispielsweise in der realen Welt auf eine Wand zugeht, kann die Anwendung dort in der virtuellen Welt einen Abgrund anzeigen.

## 2.1 Simultaneous Localization and Mapping

SLAM-Algorithmen lösen das Problem eines autonomen Roboters, sich in einer gänzlich unbekannten Umgebung zu orientieren. Dazu wird gleichzeitig eine Karte der Umgebung erstellt und die eigene Position darin bestimmt [5]. SLAM-Algorithmen haben insbesondere durch selbstfahrende Autos und Drohnen an Bedeutung gewonnen [22]. Das SLAM-Problem ist ein aktuelles Forschungsfeld. Während die Grundproblematik als gelöst gelten kann, verbessern aktuelle Ansätze ständig Genauigkeit und Performance. Durch einen stetigen Anstieg der Leistung von Mobilgeräten und neue, verbesserte SLAM-Algorithmen sind diese beiden Bereiche erst seit kurzem miteinander vereinbar.

Für den Einsatz in der vorliegenden Arbeit wird ein visueller SLAM-Algorithmus für eine Mono-Kamera benötigt. Der Algorithmus darf kein Vorwissen wie z. B. Marker benötigen, da er in einer beliebigen, unbekannten Umgebung funktionieren soll. Eine weitere Anforderung ist, dass der Quellcode veröffentlicht wurde, denn eine eigene Implementierung wäre viel zu umfangreich. Im Idealfall ist eine mobile Implementierung bekannt oder liegt sogar vor. Zumindest sollten die Hardwareanforderungen aber niedrig genug sein, sodass davon auszugehen ist, dass moderne Smartphones die erforderliche Leistung erbringen.

Die meisten SLAM-Algorithmen sind feature-basiert. Sie verwenden Verfahren, um eindeutige Featurepunkte im Bild zu finden und voneinander zu unterscheiden. Damit können übereinstimmende Bildpunkte in unterschiedlichen Bildern lokalisiert werden. Das wiederum erlaubt dann die Berechnung der Position dieses Punktes im Raum. Da die Featurepunkte eindeutig sein müssen, kommen aber nur wenige Bildpunkte in Frage. Während das die Komplexität des Problems deutlich reduziert, bleibt ein Großteil der Bildinformationen ungenutzt [16].

Das ist bei direkten Methoden anders. Diese verwenden Pixel-Intensitäten, um korrespondierende Punkte in unterschiedlichen Bildern zu finden. Dadurch können alle nicht-homogenen Bildbereiche verwendet werden. Allerdings haben diese Methoden im Allgemeinen auch einen höheren Berechnungsaufwand [16]. Insbesondere für die Erstellung der Punktfolge ist es aber von enormem Vorteil, dass eine deutlich größere Menge von Bildpunkten verwendet wird.

## 2 Verwandte Arbeiten

Die genannten Anforderungen schließen einige Arbeiten aus. Der Ansatz von Pirchheim und Reitmayr z. B. erfordert eine planare Szene, um einige Vereinfachungen anwenden zu können [15]. Das ist für AR-Anwendungen geeignet, die z. B. ein 3D-Objekt auf eine Schreibtischfläche rendern, aber nicht für die Berechnung einer Punktwolke in einer unbekannten Umgebung.

In „Real-Time 3D Tracking and Reconstruction on Mobile Phones“ wird ein Algorithmus speziell für die Rekonstruktion eines Objekts als 3D-Modell mittels eines Smartphones vorgestellt [32]. Durch die spezielle Ausrichtung auf die Rekonstruktion kann dieser Ansatz aber nur die Punktwolke für ein Objekt im Vordergrund erstellen und trackt das Smartphone relativ dazu. Das Tracken des Smartphones in einem ganzen Raum ist damit nicht möglich. Ein interessanter Nebenaspekt der Arbeit ist aber die Verwendung der IMU (*inertial measurement unit*) des Smartphones. Dabei handelt es sich um die Kombination von Accelerometer, Gyroskop und Magnetometer. Diese werden dazu genutzt, Mehrdeutigkeiten bei Rotationen aufzulösen [32]. Das könnte auch als Stabilisierung in Kombination mit einem anderen SLAM-Algorithmus verwendet werden.

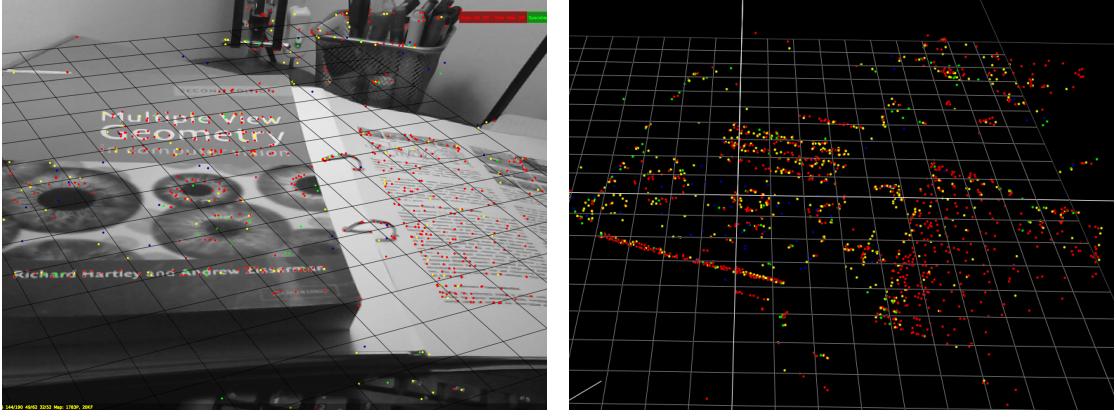
Dagegen ist der ActionSLAM ein SLAM-Algorithmus, der ausschließlich die Sensoren des Smartphones und die Kamera gar nicht verwendet [17]. Diese Technik allein ist zu ungenau für das Positionstracking von VR-Anwendungen. Im Unterschied zu „Real-Time 3D Tracking and Reconstruction on Mobile Phones“ kombiniert der ActionSLAM aber noch mehr Datenquellen wie Wifi-Signalstärken. Damit könnte ein rein visueller SLAM eventuell noch besser stabilisiert werden.

Als erster real-time-fähiger und open-source veröffentlichter<sup>1</sup> SLAM-Algorithmus, ist der PTAM (Parallel Tracking and Mapping, [8]) immer noch sehr beliebt. Er ermöglicht das Tracken einer Mono-Kamera in einem kleinen Arbeitsbereich und verwendet Featurepunkte, die mit dem Interest-Operator FAST-10 ermittelt werden. Durch eine Trennung von Tracking und Mapping kann der PTAM in zwei Threads laufen, was auch für Smartphones mit mehreren Kernen von Vorteil ist. Die Autoren haben ebenfalls eine Implementierung für das iPhone 3G vorgestellt [10], deren Sourcecode allerdings nicht veröffentlicht wurde.

---

<sup>1</sup><https://github.com/Oxford-PTAM/PTAM-GPL>

## 2 Verwandte Arbeiten



(a) Die Anzeige des Kamerabildes mit eingezeichneten Punkten und Koordinatensystem.

(b) Die Punktfolge der gleichen Szene.

Abbildung 2.1: Screenshots des PTAM. Eigene Abbildungen

Eine höhere Robustheit zeichnet den SLAM-Algorithmus DT-SLAM (Deferred Triangulation, [23]) aus. Bei den meisten Algorithmen werden Featurepunkte mit ungenügenden Informationen verworfen, z. B. nach einer reinen Rotationsbewegung. Beim DT-SLAM hingegen werden diese zurückgestellt bis genug Informationen vorliegen. Dadurch erreicht der Algorithmus eine höhere Stabilität und verwendet mehr Bildpunkte. Insbesondere VR-Anwendungen könnten von der Rotationsrobustheit profitieren. Der Sourcecode wurde bei GitHub veröffentlicht<sup>2</sup>. Eine mobile Implementierung gibt es nicht, aber die Autoren vermuten, dass ihr Algorithmus mit Optimierungen auch auf einem Mobilgerät läuft [23].

Der LSD-SLAM (Large-scale Direct Monocular SLAM, [22]) ist der erste direkte, echtzeitfähige SLAM-Algorithmus [16]. Alle Pixel mit ausreichendem Gradienten werden für die Tiefenberechnung verwendet. Dadurch sind die Tiefeninformationen und damit letztlich auch die damit berechnete Punktfolge sehr viel dichter. Der Sourcecode des LSD-SLAM wurde veröffentlicht.<sup>3</sup> Eine mobile Implementierung für Android wird in „Semi-dense visual odometry for AR on a smartphone“ beschrieben. Der dazugehörige Sourcecode ist aber nicht frei zugänglich [26].

<sup>2</sup><https://github.com/plumonito/dtslam>

<sup>3</sup>[https://github.com/tum-vision/lsd\\_slam](https://github.com/tum-vision/lsd_slam)

## 2 Verwandte Arbeiten

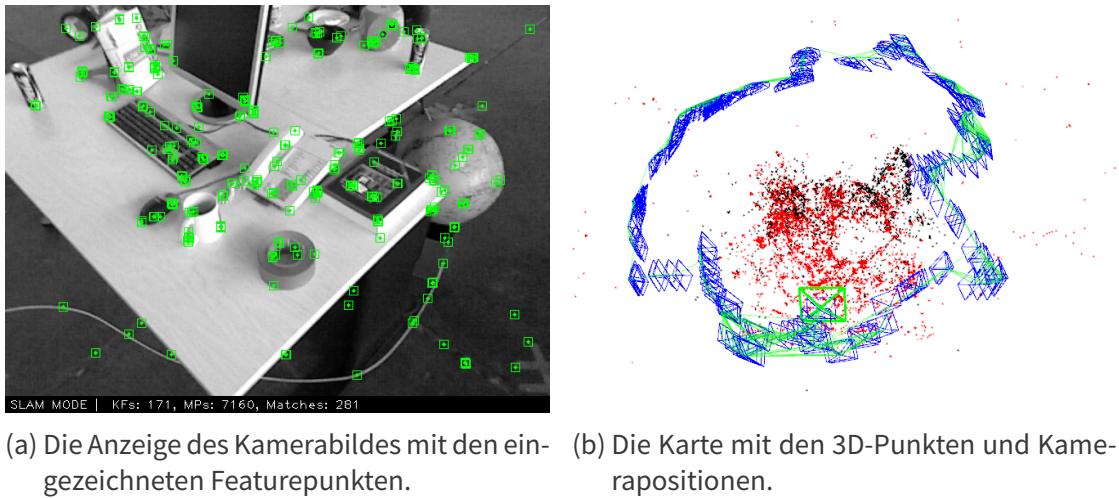


Abbildung 2.2: Screenshots des ORB-SLAM mit dem Datensatz fr2/desk der TU München. Eigene Abbildungen

Ein sehr aktueller SLAM-Algorithmus ist der ORB-SLAM [30]. Er verwendet den Feature-Deskriptor ORB (Oriented FAST and Rotated BRIEF). Dieser Algorithmus sieht sich in der Nachfolge des PTAM und vereint vor allem aktuelle state-of-the-art Techniken zu einer vielseitigen Bibliothek. Der Sourcecode zum Paper wurde bei GitHub veröffentlicht<sup>4</sup>. Am 17. Januar 2016 wurde eine neue, verbesserte Version publiziert<sup>5</sup>, die damit aber zum Zeitpunkt der Recherche noch nicht verfügbar war. Eine mobile Implementierung gibt es nicht und im Paper werden auch keine Hardwareanforderungen genannt.

Die Autoren des ORB-SLAM empfehlen in „Monocular SLAM for User Viewpoint Tracking in Virtual Reality“ ihren Algorithmus für das Tracking eines Head-Mounted Display [31]. Dabei wird die Karte in einem gesonderten Schritt vor der Benutzung erstellt. Dadurch muss der SLAM dann nur noch tracken und kann auf die bereits erstellte Karte zurückgreifen. Abstriche bei der Flexibilität ermöglichen also eine höhere Performance.

<sup>4</sup>[https://github.com/raulmur/ORB\\_SLAM](https://github.com/raulmur/ORB_SLAM)

<sup>5</sup>[https://github.com/raulmur/ORB\\_SLAM2](https://github.com/raulmur/ORB_SLAM2)

Tabelle 2.1: Übersicht geeigneter visueller SLAM-Algorithmen für Mono-Kameras

Name	feature-basiert / direkt	open-source	mobil
PTAM	feature-basiert	✓	(✓)
LSD-SLAM	direkt	✓	(✓)
DT-SLAM	feature-basiert	✓	✗
ORB-SLAM	feature-basiert	✓	✗

Auf Basis der Recherche wurde der LSD-SLAM als Grundlage für diese Arbeit ausgewählt. Die Anforderung nach einer mobilen Implementierung erfüllen der PTAM und der LSD-SLAM. Als direkter SLAM liefert der LSD-SLAM aber nicht nur dichtere Tiefeninformationen, sondern erkennt insbesondere auch Kanten. Feature-basierte SLAM-Algorithmen wie der PTAM sind auf Featurepunkte angewiesen, die eindeutig identifizierbar sein müssen [22]. Entlang einer Kante sehen sich aber die meisten Punkte sehr ähnlich und kommen daher nicht in Frage. Gerade Kanten machen aber den Großteil möglicher Hindernisse in mensch-gemachten Umgebungen wie Gebäuden aus [22]. Der LSD-SLAM wird in Kapitel 3 ausführlicher beschrieben.

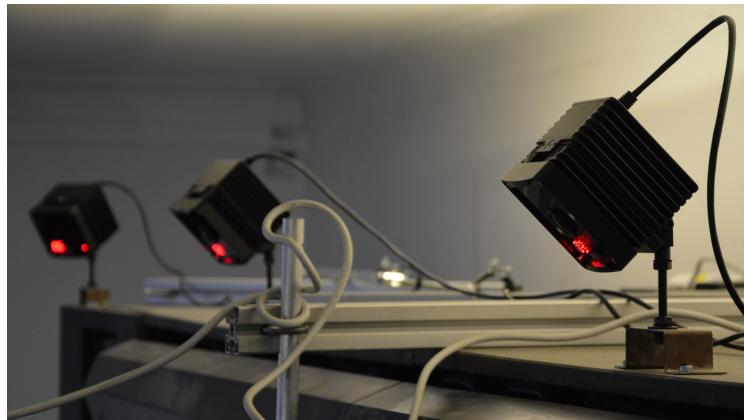
## 2.2 Virtual Reality

Die freie, natürliche Fortbewegung (engl. locomotion) in einer virtuellen Umgebung ist ein großes Problem der VR. Im Forschungsbereich werden häufig Tracking-Systeme verwendet, die mit Infrarotkameras besonders stark reflektierende Marker erfassen (vgl. Abbildung 2.3). Über mehrere Kameras mit unterschiedlichen Positionen und Orientierungen kann die tatsächliche 3D-Position des Markers im Raum trianguliert werden. Doch solche Systeme sind viel zu teuer für den Privatbereich und außerdem nicht flexibel, da die Installation aufwendig ist und nur einen festgelegten Bereich abdeckt.

Locomotion is an important interaction technique in virtual environments. Real walking is better than flying or walking-in-place, in terms of presence, ease of use, and naturalness.

— Razzaque, Kohn und Whitton [3]

## 2 Verwandte Arbeiten



(a) Erst mehrere Infrarotkameras ermöglichen Tracking.



(b) Eine einzelne Kamera.

Abbildung 2.3: Das Tracking-System ARTTrack der Firma ART wird im VR-Labor der Bauhaus-Universität Weimar verwendet. Eigene Abbildungen

Eine unbegrenzt große, begehbarer virtuelle Umgebung können hingegen Laufbänder abdecken, die in alle Richtungen funktionieren. Das Laufband bewegt sich dabei so, dass der Nutzer immer möglichst zentral darauf steht. Üblicherweise wird es mit einem HMD kombiniert, wobei die Kabel über die Decke geführt werden. Sicherheitsgurte gewährleisten, dass der Nutzer nicht vom Laufband fallen kann oder mit den Fingern an das Laufband kommt. Erstmals wurde ein solches System 1997 vorgeschlagen [2].

Inzwischen gibt es mit dem Virtuix Omni sogar ein kommerzielles Produkt. Dabei handelt es sich nicht um ein Laufband, sondern eine Mulde, in der man mittels Spezialschuhen immer wieder in die Mitte rutscht. Während dadurch eine kleinere Fläche benötigt wird, fühlt sich das Laufen so aber auch weniger intuitiv an. Der Omni ist mit \$699 zudem nicht sehr günstig und für das heimische Wohnzimmer ein ziemlich sperriges Konstrukt [38].

Beim Redirected Walking wird die Laufbahn eines Nutzers mit HMD leicht manipuliert, sodass dieser innerhalb eines festen Bereichs bleibt. Während der Nutzer in der virtuellen Welt eine gerade Strecke zurücklegt, läuft er tatsächlich, ohne es zu bemerken, einen leichten Bogen und verlässt so zu keinem Zeitpunkt den getrackten Bereich [3]. Damit der Nutzer in der virtuellen Umgebung kontinuierlich geradeaus

## 2 Verwandte Arbeiten

laufen kann und dies nicht wahrnimmt, wird allerdings ein Raum von  $40\text{ m} \times 40\text{ m}$  benötigt. Der benötigte Raum könnte aber reduziert werden, wenn die Nutzer aktiv von der Umgebung abgelenkt werden oder in Kauf genommen wird, dass Nutzer die Krümmung ab und zu wahrnehmen [12]. In jedem Fall wird für Redirected Walking ein relativ großer, getraktter Bereich benötigt.

Die Technik Substitutional Reality integriert reale Objekte in die virtuelle Umgebung. In einem virtuellen Piratenschiff könnte z. B. ein Couchtisch durch eine Kiste gleicher Größe ersetzt werden. Dadurch werden Hindernisse in der virtuellen Umgebung dargestellt und der Nutzer stößt nicht unerwartet mit diesen zusammen. Damit die virtuelle Umgebung frei gestaltet werden kann, werden Diskrepanzen zwischen den realen Objekten und ihrer virtuellen Repräsentation in Kauf genommen [34].

Allerdings erfordert Substitutional Reality eine umgebungsspezifische und damit nutzerspezifische virtuelle Umgebung. Um eine möglichst exakte Abbildung zu erreichen, muss der Raum manuell vermessen und modelliert werden. Ein Vorschlag für eine automatisierte Lösung sieht vor, die verwendete Oculus Rift mit einer Kinect zu erweitern. Über das Tiefenbild könnten eine Punktwolke berechnet und darin Objekte erkannt werden. Ein Katalog von geeigneten und zum virtuellen Setting passenden Modellen soll dann beim Finden einer geeigneten virtuellen Repräsentation helfen [34]. Dieser Ansatz kann auch mit einem SLAM-Algorithmus umgesetzt werden, da dieser ebenfalls eine Punktwolke der Umgebung erstellt.

Alle präsentierten VR-Techniken haben schwerwiegende Nachteile. Laufbänder sind teure und sperrige zusätzliche Hardware. Redirected Walking benötigt sehr viel freien Platz und Substitutional Reality erfordert ein Vermessen und Modellieren der Umgebung. Beide sind auf ein externes Tracking-System angewiesen.

SLAM-Algorithmen könnten sowohl Tracking- als auch Umgebungsinformationen liefern. Darauf aufsetzend können Ideen aus der Substitutional Reality genutzt werden, um die virtuelle Umgebung dynamisch an die reale Umgebung anzupassen. Ein solches System würde ein dynamisches VR-Erlebnis in einer beliebigen Umgebung ermöglichen.

Die vorliegende Arbeit setzt die ersten Schritte eines solchen Systems um. Der LSD-SLAM wird mobil implementiert und darüber Umgebungsinformationen an die Spiele-Engine Unity weitergegeben und dem Nutzer angezeigt.

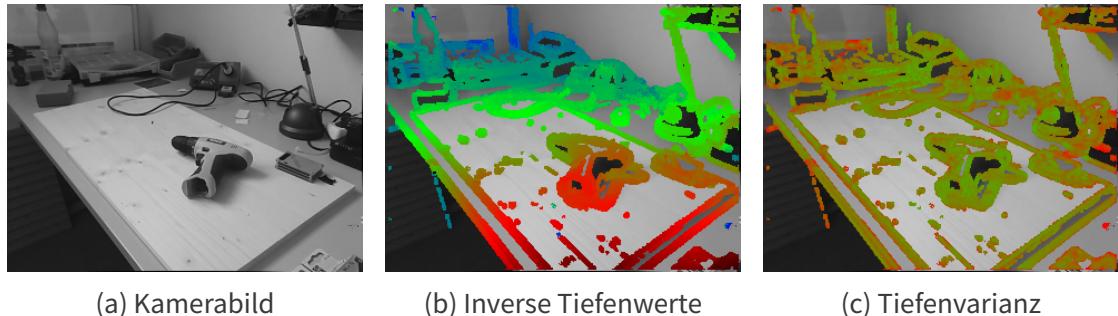
## 3 LSD-SLAM

Im 2013 veröffentlichten Paper „Semi-dense Visual Odometry for a Monocular Camera“ beschreiben Engel, Sturm und Cremers einen Ansatz für das Tracken einer Mono-Kamera über deren Kamerabilder [16]. Eine solche sogenannte visuelle Odometrie erstellt im Unterschied zu SLAM-Algorithmen keine globale Karte der Umgebung, sondern hält nur Informationen zum jeweils aktuellen Bild bereit. Auf der Basis dieses Papers erschien 2014 der LSD-SLAM [22]. Durch das Speichern von Keyframes werden die einzelnen lokalen Tiefeninformationen zu einer globalen Karte der Umgebung zusammengefügt.

Theoretisch wird für jedes Pixel aller Keyframes der inverse Tiefenwert und die Varianz dieses Werts berechnet. In der Praxis geschieht das nur für Pixel mit hinreichend großem Gradienten, also nicht für homogene Bildbereiche. Daher bezeichnen die Autoren ihre Methode als „semi-dense“. Für den ersten Keyframe werden die inversen Tiefenwerte zufällig gesetzt und sehr große Varianzwerte gespeichert. Nach anfänglicher Translationsbewegung initialisiert sich der LSD-SLAM automatisch: „Given sufficient translational camera movement in the first seconds, the algorithm ‚locks‘ to a certain configuration, and after a couple of keyframe propagations converges to a correct depth configuration“ [22]. Die Kameraposition des ersten Keyframes bildet den Ursprung des Tracking-Koordinatensystems.

Monokulare, visuelle SLAM-Algorithmen wie der LSD-SLAM können die absolute Skalierung ihrer Beobachtungen nicht bestimmen. Ihnen fehlt eine bekannte Bezugsgröße in der realen Welt. Für die geplante Anwendung ist das problematisch, denn der Abstand zu den erkannten Objekten ist wichtig. Andererseits hat das den Vorteil, dass es große Skalierungsunterschiede in der Umgebung mit modelliert. Tiefen- oder Stereokameras können zwar die absolute Skalierung berechnen, funktionieren dafür aber nur innerhalb einer bestimmten Reichweite [22]. In Unterabschnitt 4.3.5 wird das Problem der Skalierung in Bezug auf die letztliche App diskutiert.

### 3 LSD-SLAM



Eigene Abbildungen

The scale of the world cannot be observed and drifts over time, being one of the major error sources.

— Engel, Schöps und Cremers [22]

Beim LSD-SLAM ist die Skalierung von der Initialisierung abhangig. Fur jeden Keyframe wird die Skalierung so bestimmt, dass die durchschnittliche inverse Tiefe Eins ist [22]. Beim ersten Keyframe wird zuerst die Skalierung auf Eins festgelegt und dann werden die Tiefenwerte entsprechend gesetzt.

Der LSD-SLAM besteht aus drei Bestandteilen, die parallel in Threads laufen [22]. Abschnitt 3.1 beschreibt, wie jedes neue Kamerabild zunächst relativ zum aktuellen Keyframe getrackt wird. Abschnitt 3.2 erläutert die eigentliche Berechnung der Tiefenwerte. Alle Keyframes zusammen bilden als *pose graph* die globale Karte der Umgebung. Die Optimierung dieser Karte behandelt Abschnitt 3.3.

### 3.1 Tracking

Der LSD-SLAM trackt neue Kamerabilder mit dense image alignment [16]. Dabei wird die relative Pose eines neuen Kamerabildes zum aktuellen Keyframe anhand des gesamten Kamerabildes und der vorhandenen Tiefenwerte abgeschatzt. Dem liegt die Annahme zugrunde, dass ein Punkt in der Welt in beiden Kamerabildern etwa die gleiche Intensitat aufweist. Die Intensitatsunterschiede der beiden Bilder sind so ausschlielich das Ergebnis der relativen Kamerabewegung. Es wird diejenige Bewegung gesucht, die diese Unterschiede unter Bercksichtigung der Tiefenwerte des

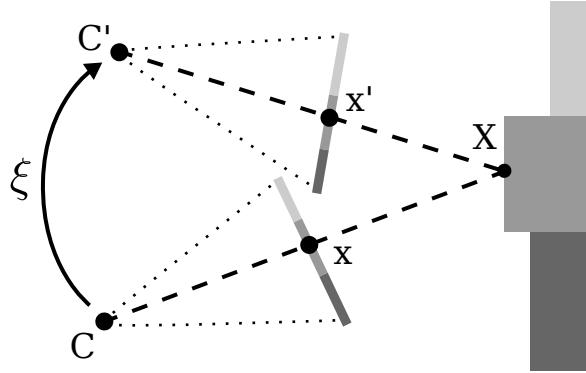


Abbildung 3.2: Die relative Pose  $\xi$  zwischen zwei Kamerabildern berechnet der LSD-SLAM mit dense image alignment. Eigene Abbildung nach [19]

Keyframes am besten erklären kann [19]. Die relative Kamerabewegung ergibt sich schrittweise mit einer Minimierung der gesamten Intensitätsabweichungen durch das Gauß-Newton-Verfahren [16]. Eine zusätzliche Gewichtung anhand der Tiefenvarianz verhindert, dass neue und damit noch ungenaue Tiefenabschätzungen zu stark mit einfließen. Da für das neue Kamerabild noch keine Tiefeninformationen vorliegen, ist die Skalierung zunächst nicht bekannt [22].

Zur Beschleunigung der Berechnung kommt außerdem eine Bildpyramide zum Einsatz. Die relative Pose wird zunächst mit einer geringeren Auflösung der Kamerabilder berechnet. Das Ergebnis ist der Startwert für die nächsthöhere Auflösung, sodass für die höheren Auflösungen weniger Iterationen benötigt werden. Insgesamt kommen vier Pyramidenlevel zum Einsatz [16].

Überschreitet die Distanz oder der Winkel der Kamera zum aktuellen Keyframe einen Schwellenwert, wird das aktuelle Kamerabild zu einem neuen Keyframe. Punkte in der Nähe liegender Keyframes werden in den neuen Keyframe projiziert, um die Tiefenmap zu initialisieren. Danach wird die Skalierung des neuen Keyframes so bestimmt, dass die durchschnittliche inverse Tiefe Eins ist [22].

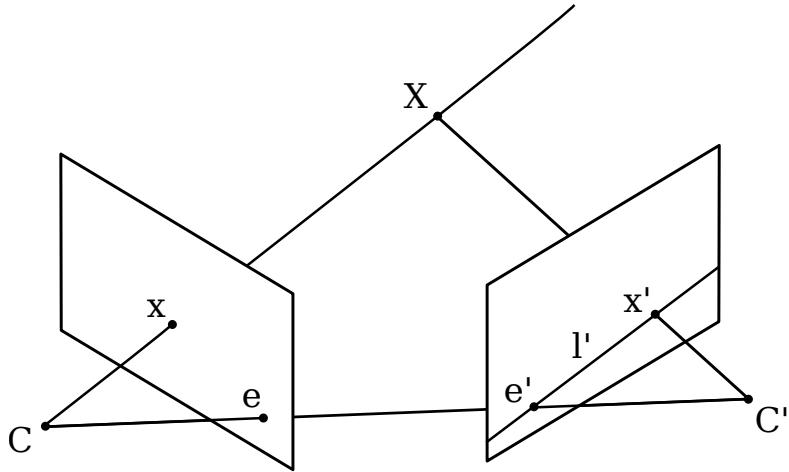


Abbildung 3.3: Veranschaulichung der Epipolargeometrie. Eigene Abbildung nach [4]

### 3.2 Depth Map Estimation

Kamerabilder, die keine neuen Keyframes bilden, werden dazu genutzt, Tiefeninformationen für den aktuellen Keyframe zu gewinnen. Der Algorithmus sucht dabei für sehr viele Pixel die korrespondierenden Pixel aus vorherigen Kamerabildern [22]. Zwischen den Bildern hat die Kamera sich nicht viel bewegt. Die *baseline* ist also sehr klein und daher auch der Abstand zwischen den Pixeln meist nicht groß. Dadurch ist das *stereo matching* recht leicht und der LSD-SLAM kann so viele korrespondierende Punkte finden. Der Abstand der Bildkoordinaten dieser Pixel heißt Disparität. Bei geringer Kamerarotation ist er proportional zur inversen Tiefe des Punktes. Daher kann mit der Disparität auf die inverse Tiefe geschlossen werden.

Das *stereo matching* wird durch die Berücksichtigung der Epipolargeometrie vereinfacht. Der aktuelle Keyframe definiert für jedes Pixel einen Strahl zwischen dem Kamerazentrum  $C$  und der Pixelposition  $x$  (vgl. Abbildung 3.3). Auf diesem Strahl liegt der tatsächliche 3D-Punkt  $X$  mit der Tiefe als Abstand zum Kamerazentrum. Für das neue Kamerabild wurde im Tracking-Schritt bereits das Kamerazentrum  $C'$

berechnet. Dieses definiert zusammen mit dem Strahl von  $C$  über  $X$  die Epipolare Ebene  $\pi$ . Der Schnitt dieser Ebene mit der Bildebene des neuen Kamerabildes ergibt die Epipolarlinie  $l'$ . Auf dieser Linie muss das gesuchte Pixel im neuen Bild liegen [4]. Der LSD-SLAM sucht daher nur entlang dieser Epipolarlinie nach dem Intensitätswert des Pixels  $x$ . Bereits vorhandenes Wissen über den Tiefenwert kann dazu genutzt werden, den Suchbereich noch weiter einzuschränken [16].

### 3.3 Map Optimization

Immer wenn ein neuer Keyframe den aktuellen ersetzt, wird der alte Keyframe zur globalen Karte hinzugefügt, dem sogenannten *pose graph*. Dieser wird als Graph von Keyframes mit den relativen Posen als Kanten dazwischen gespeichert. Beim Hinzufügen eines neuen Keyframes wird überprüft, ob dieser einen *loop* schließt und gegebenenfalls die relativen Posen der anderen Keyframes im Graph angepasst werden müssen [22].

Bei SLAM-Algorithmen entsteht ein nicht vermeidbarer *drift*, eine Abweichung zwischen tatsächlicher und gemessener Position. Das fällt insbesondere dann auf, wenn der Nutzer über einen unbekannten Weg an einen bereits besuchten Ort zurückkehrt. In der Karte des SLAM kann dieser Ort wegen des *drift* dann zwei Mal auftauchen. *Loop closure* bezeichnet das Schließen solcher Lücken, indem identische Orte erkannt und die Pfade dazwischen entsprechend korrigiert werden [6].

Der *pose graph* wird im Hintergrund kontinuierlich vom *map optimization thread* optimiert [22]. Für die Optimierung wird das Framework g<sup>2</sup>o verwendet. Dabei handelt es sich um ein „open-source C++ framework for optimizing graph-based nonlinear error functions“ [14]. g<sup>2</sup>o erlaubt es, das Optimierungsproblem mit einer Fehlerfunktion und der Wahl eines Lösungsverfahrens zu formulieren. Die eigentliche Optimierung wird dann ausschließlich von g<sup>2</sup>o durchgeführt.

# 4 Eigene Umsetzung

Die eigene Umsetzung basiert auf dem Sourcecode des LSD-SLAM, der am 13. September 2014 unter der GNU General Public License auf GitHub<sup>1</sup> veröffentlicht wurde. In einem ersten Schritt soll der LSD-SLAM in einer kompakten, eigenständigen Desktop-Anwendung funktionieren. Das dient als *proof of concept* dafür, dass der LSD-SLAM prinzipiell lauffähig ist. Darauf aufbauend wird der Algorithmus für das mobile Betriebssystem Android portiert. Aus dieser eigenständigen mobilen Implementierung soll schließlich ein Android-Plugin für die Spiel-Engine Unity hervorgehen.

## 4.1 Desktop-Anwendung

Der veröffentlichte Code des LSD-SLAM soll zunächst auf einem Desktoprechner mit Linux als Betriebssystem und einer angeschlossenen Webcam funktionieren. Dazu wurde die Webcam C270 von Logitech<sup>2</sup> eingesetzt, die mit einer Auflösung von bis zu  $1280 \times 720$  Pixeln aufnimmt und über USB betrieben wird.

### 4.1.1 ROS-Abhängigkeit entfernen

Der veröffentlichte LSD-SLAM-Code verwendet ROS<sup>3</sup>. Dabei handelt es sich um ein Software-Framework für Robotik-Anwendungen, das eine einfache Kommunikation zwischen modularen Komponenten ermöglicht. ROS stellt eine einfache Möglichkeit dar, den LSD-SLAM in eine bestehende ROS-basierte Anwendung zu integrieren.

---

<sup>1</sup>[https://github.com/tum-vision/lsd\\_slam](https://github.com/tum-vision/lsd_slam)

<sup>2</sup><http://www.logitech.com/de-de/product/hd-webcam-c270>

<sup>3</sup><http://www.ros.org/>

Note that building without ROS is not supported, however ROS is only used for input and output, facilitating easy portability to other platforms.

— README des LSD-SLAM <sup>4</sup>

Diese Flexibilität wird aber für die geplante Anwendung nicht benötigt, denn der gesamte Datenfluss findet nur innerhalb der Anwendung selbst statt. ROS bringt daher nur unnötige Komplexität mit sich.

Die im Zitat genannte „easy portability“ wird durch eine Trennung des Input-Output-Codes vom eigentlichen SLAM-Code ermöglicht. Die IO-Komponenten verfügen allesamt über Basisklassen, die Schnittstellen zum LSD-SLAM definieren. Sämtliche ROS-abhängige Klassen liegen nochmals in einem extra Ordner und erben von diesen Basisklassen. Das erlaubt es, ROS gezielt zu ersetzen, ohne in den Basis-Code des LSD-SLAM eingreifen zu müssen.

### 4.1.2 CMake Build System

Der LSD-SLAM verwendet für seinen Build-Prozess *rosbuild*, das auf CMake<sup>5</sup> aufsetzt und es um einige ROS-spezifische Makros erweitert. Es liegt daher nahe, ebenfalls CMake zu benutzen und die vorhandenen CMake-Dateien anzupassen.

CMake abstrahiert den Build-Prozess durch Skripte, mit denen Makefiles beziehungsweise Projektdateien für unterschiedliche Systeme und Anwendungen generiert werden können. Dabei kann CMake über Kommandozeilenparameter oder eine grafische Oberfläche konfiguriert werden. Das erlaubt es, für nicht automatisch lokalisierte Abhängigkeiten manuell einen Pfad anzugeben. Wenn CMake konfiguriert wurde und alle benötigten Bibliotheken und Header gefunden hat, generiert es die Makefiles. Das Build-Management-Tool *make* kompiliert dann den LSD-SLAM als *shared library*. Die eigentliche Desktop-Anwendung wird als separate Anwendung mit den gleichen Tools kompiliert und gegen die *shared library* des LSD-SLAM gelinkt.

---

<sup>4</sup>[https://github.com/tum-vision/lsd\\_slam/blob/master/README.md#2-installation](https://github.com/tum-vision/lsd_slam/blob/master/README.md#2-installation)

<sup>5</sup><https://cmake.org/>

## 4 Eigene Umsetzung

### 4.1.3 Abhangigkeiten

Der LSD-SLAM bringt einige Abhangigkeiten mit sich. CMake stellt Makros zum Finden und Einbinden von Bibliotheken und Headern mit sich. Das macht es einfacher, diese in einem CMake-Projekt zu verwenden. Die in der CMake-Syntax geschriebenen Makros versuchen, die benotigten Bibliotheken und Header zu lokalisieren. Im einfachsten Fall wird der Pfad ermittelt, der eine Datei eines bestimmten Namens enthalt. So sucht das g<sup>2</sup>o-Makro zum Beispiel nach dem Ordner, der die Datei g2o/core/base\_vertex.h enthalt.

**g<sup>2</sup>o** Das Framework g<sup>2</sup>o optimiert Graphen mittels einer Fehlerfunktion [14]. Es wird vom LSD-SLAM verwendet, um die zu den Keyframes gehorenden Posen zu optimieren (vgl. Abschnitt 3.3). g<sup>2</sup>o ist uber GitHub<sup>6</sup> verfugbar. Es wird ebenfalls mit CMake fur den *build* konfiguriert und unter Linux mit *make* kompiliert und installiert.

**Eigen3** „Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.“<sup>7</sup> Eigen3 wird sowohl vom LSD-SLAM selbst als auch in g<sup>2</sup>o verwendet.

**SuiteSparse** Bei SuiteSparse<sup>8</sup> handelt es sich um eine Sammlung von Algorithmen fur dunn besetzte Matrizen (*sparse matrices*). SuiteSparse ist eine optionale Abhangigkeit von g<sup>2</sup>o, wird aber vom LSD-SLAM verwendet und muss daher installiert sein.

**Boost** Die Boost-Bibliothek wird fur die Parallelisierung und Verwaltung der einzelnen Threads verwendet.

**OpenCV** Die *Open Source Computer Vision Library* (OpenCV) wird fur den Kamerazugriff (siehe Unterabschnitt 4.1.4) und die Rektifizierung des Bildes genutzt. Außerdem wird das Matrix-Format von OpenCV als internes Format fur die Bilddaten verwendet.

**openFABMAP** Die optionale Abhangigkeit openFABMAP wird fur „large-scale loop closures“ [22] verwendet.

---

<sup>6</sup><https://github.com/RainerKuemmerle/g2o>

<sup>7</sup><http://eigen.tuxfamily.org>

<sup>8</sup><http://faculty.cse.tamu.edu/davis/suitesparse.html>

## 4 Eigene Umsetzung

Für die vorliegende Anwendung wird openFABMAP zunächst nicht genutzt. Da der SLAM-Algorithmus für VR-Anwendungen genutzt wird, ist davon auszugehen, dass Nutzer sich ausschließlich innerhalb relativ kompakter Umgebungen bewegen. Der LSD-SLAM schließt *loops* bereits selbst [22] und openFABMAP wird nur für große *loops* benötigt.

Alle Abhängigkeiten außer  $g^2o$  sind unter den verwendeten Linux-Distributionen Ubuntu und Arch Linux als Softwarepakete verfügbar. Das erlaubt eine einfache Installation über den Paketmanager.

### 4.1.4 Bildinput

Die Kamerabilder bezieht der LSD-SLAM standardmäßig mittels ROS, das dazu wiederum OpenCV wrappt. Durch das Wegfallen von ROS wird daher eigener Code benötigt, der einen kontinuierlichen Stream von Kamerabildern liefert. OpenCV ist dafür die naheliegende Wahl. Es ermöglicht einen einfachen Zugriff auf Kameras und ist ohnehin eine Abhängigkeit des LSD-SLAM.

Im LSD-SLAM existiert bereits eine Basisklasse für den Bildinput. Die neue Klasse erbt von dieser und greift mittels OpenCV auf die Kamera zu. Deren Hauptschleife läuft in einem eigenen Thread und liest kontinuierlich neue Kamerabilder aus. Diese werden in einem Buffer gespeichert und von dort vom eigentlichen SLAM-Algorithmus ausgelesen und verwendet.

### 4.1.5 Kalibrierung

Kameras wie die verwendete Webcam sind inzwischen äußerst günstig. Allerdings weichen die aufgenommen Bilder vom theoretischen Lochkameramodell ab. Verzeichnungen führen insbesondere am Bildrand zu Verfälschungen der Pixelpositionen [4]. Für viele Anwendungen der Kamera spielt das keine Rolle. Algorithmen aus der Computer Vision gehen aber normalerweise vom Lochkameramodell aus. Dadurch ist zunächst eine Korrektur dieser Verzeichnungen nötig.

## 4 Eigene Umsetzung

```
1 fx fy cx cy k1 k2 p1 p2  
2 inputWidth inputHeight  
3 "crop" / "full" / "none" / "e1 e2 e3 e4 0"  
4 outputWidth outputHeight
```

Listing 1: Die vorgegebene Struktur der Konfigurationsdatei des LSD-SLAM mit den Kalibrierungsparametern für das OpenCV-Kameramodell.



Abbildung 4.1: Das Schachbrettmuster, das für die Kalibrierung verwendet wurde.  
Eigene Abbildung

Die OpenCV-Dokumentation enthält einen Artikel<sup>9</sup> über die Kamerakalibrierung mit OpenCV. Darin wird ebenfalls der Sourcecode für ein Kalibrierungsprogramm<sup>10</sup> verlinkt. Das Programm wird über xml-Dateien konfiguriert und kann auf einem Livestream der Kamera oder zuvor aufgenommenen und abgespeicherten Kamerabildern arbeiten.

Der LSD-SLAM liest die Kalibrierungsparameter aus einer Textdatei aus.<sup>11</sup> Daher wurde das Kalibrierungsprogramm um eine Exportfunktion erweitert. Diese speichert die Parameter in der erwarteten Form ab (vgl. Listing 1).

<sup>9</sup>[http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera\\_calibration/camera\\_calibration.html](http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html)

<sup>10</sup>[http://docs.opencv.org/2.4/\\_downloads/camera\\_calibration.cpp](http://docs.opencv.org/2.4/_downloads/camera_calibration.cpp)

<sup>11</sup>[https://github.com/tum-vision/lsd\\_slam#calibration-file-for-opencv-camera-model](https://github.com/tum-vision/lsd_slam#calibration-file-for-opencv-camera-model)

## 4 Eigene Umsetzung



Abbildung 4.2: Die Struktur der Desktop-Anwendung. Eigene Abbildung

$f_x$  und  $f_y$  sind dabei die Brennweiten.  $c_x$  und  $c_y$  bilden den Bildmittelpunkt. Die Parameter  $k_1$  und  $k_2$  sind die radialen,  $p_1$  und  $p_2$  die tangentialen Verzeichnungsparameter.

Das Kalibrierungsprogramm wird zusammen mit dem LSD-SLAM kompiliert und für die Kalibrierung der Webcam verwendet. Dazu werden Bilder von einem Schachbrettmuster (vgl. Abbildung 4.1) aufgenommen. Mit diesen als Input kann das Kalibrierungsprogramm die Kalibrierungsparameter bestimmen. Im LSD-SLAM werden die Kalibrierungsparameter von der `Undistorter`-Klasse ausgelesen und abgespeichert. Der `Undistorter` verwendet Funktionen von OpenCV, um eingehende Kamerabilder zu rektifizieren und dabei gleichzeitig auf die angegebene Zielgröße zu skalieren.

### 4.1.6 Anwendung

Damit sind alle Voraussetzungen für die Desktop-Anwendung erfüllt. Diese verwaltet und verknüpft die bereits vorliegenden Komponenten. Als Input werden, wie in Unterabschnitt 4.1.4 beschrieben, Bilder der Webcam über OpenCV verwendet. Eine Output-Klasse verfügt über Callbacks, die z. B. beim Erstellen eines neuen Keyframes oder nach dem Tracking eines normalen Frames aufgerufen werden. Diese geben die aktuelle Kamerapose auf der Konsole aus. Dabei werden der Positionsvektor und die Orientierung als Quaternion angezeigt.

Die Anwendung kann beim Aufruf im Terminal mit Parametern konfiguriert werden. So können der Pfad zur Datei mit den Kalibrierungsparametern angegeben und die Kamera-ID gesetzt werden. Dadurch lässt sich eine von mehreren angeschlossenen Kameras auswählen. Mit der Program-Options-Bibliothek von Boost werden die Kommandozeilenparameter konfiguriert und ausgelesen.

## 4 Eigene Umsetzung

Die Desktop-Anwendung ist als Release im GitHub-Repository<sup>12</sup> verfügbar.

### 4.2 Android-Port

Ziel dieser Arbeit ist eine mobile Implementierung des LSD-SLAM. Die Desktop-Anwendung dient lediglich als *proof of concept*. Sie zeigt, dass der open-source Code des LSD-SLAM funktioniert und gibt Aufschluss über die Abhängigkeiten. Der in Abschnitt 4.1 entstandene Code dient als Basis der im Folgenden beschriebenen mobilen Portierung.

Für diese wurde Googles mobiles Betriebssystem Android als Plattform gewählt. Dafür spricht zum einen, dass die einzige bekannte mobile Implementierung der visuellen Odometrie von Engel, Sturm und Cremers ebenfalls auf Android setzt [26]. Zum anderen unterstützt das vom LSD-SLAM verwendete Framework g<sup>2</sup>o offiziell nur Android als mobile Zielplattform. Für Android sind außerdem die notwendige Hardware sowie Entwicklungserfahrung vorhanden.

Das Paper „Semi-dense visual odometry for AR on a smartphone“ von Schöps, Engel und Cremers beschreibt eine mobile AR-Anwendung [26]. Es setzt auf der visuellen Odometrie auf, die auch dem LSD-SLAM zugrunde liegt [16]. Die semi-dichten Tiefenmaps werden dazu genutzt, ein Kollisionsmesh zu erstellen. Gerenderte Objekte können darüber mit der echten Geometrie physikalisch korrekt interagieren. In der Beispielanwendung können virtuelle Objekte platziert und bewegt werden. Ein virtuelles Automodell kann beispielsweise über eine Rampe aus realen Objekten gefahren werden [26].

Android-specific optimizations and AR integration are not part  
of the open-source release.

— Webseite des LSD-SLAM 

---

<sup>12</sup><https://github.com/mmbuw/dont-slam-your-head/releases/tag/desktop>

<sup>13</sup><http://vision.in.tum.de/research/vslam/lسدslam>

## 4 Eigene Umsetzung

Obwohl der Sourcecode der mobilen Anwendung nicht veröffentlicht wurde, liefert das Paper wichtige Erkenntnisse für eine mobile Implementierung des LSD-SLAM. So zeigt es, dass die Technik des LSD-SLAM prinzipiell unter Android genutzt werden kann. Außerdem geben die Autoren Hinweise auf einige Optimierungen speziell für Smartphones.

Als Hardware kam ein Nexus 5<sup>14</sup> von LG zum Einsatz. Es verfügt über einen 2,26 GHz Quadcore-Prozessor von Qualcomm. Bei einer Bildschirmdiagonalen von fünf Zoll zeigt das Display eine Full-HD Auflösung von  $1920 \times 1080$  Pixeln an. Die Kamera nimmt Bilder mit bis zu 8 Megapixeln auf.

Für die Entwicklung wurde die IDE Android Studio von Google verwendet. Diese ist speziell auf die Entwicklung von Android-Apps zugeschnitten. Die Verwendung von C++-Bibliotheken unter Android erlaubt das Android NDK (Android Native Development Kit).

### 4.2.1 CMake

Mit android-cmake<sup>15</sup> gibt es eine Sammlung von CMake-Skripten, die es ermöglicht, mit CMake nativen Code für Android zu kompilieren. Für den Code des LSD-SLAM kommt bereits CMake als Build-Tool zum Einsatz (vgl. Unterabschnitt 4.1.2). Damit ist android-cmake eine einfache Möglichkeit, Android als Zielplattform zu unterstützen. Durch das Setzen einiger Android-spezifischer Parameter, wie z. B. der Zielarchitektur, kann der Build konfiguriert werden.

Der LSD-SLAM wird zunächst als *shared library* für Android kompiliert. `make install` verschiebt die resultierenden Bibliotheken dann in den dafür vorgesehenen Ordner der Android-App. Ein selbst geschriebenes Bash-Skript vereinfacht den Buildprozess. Es führt diese Schritte nacheinander aus und kann über Parameter konfiguriert werden.

---

<sup>14</sup><http://www.lg.com/de/handy/lg-Nexus-5>

<sup>15</sup><https://github.com/taka-no-me/android-cmake>

## 4 Eigene Umsetzung

### 4.2.2 Java Native Interface

Das Java Native Interface (JNI) ist Javas API für die Verwendung von nativem Code in Java-Anwendungen [33]. Der JNI-Code stellt die Schnittstelle zwischeninem C++-Code und der eigentlichen Java-Anwendung dar.

JNI [...] is a two-way bridge between the Java and native side; the only way to inject the power of C/C++ into your Java application.

— Ratabouil [33]

Es handelt sich dabei selbst um nativen Code. Dieser wird zu einer *shared library* kompiliert und von der Java-Anwendung geladen. Innerhalb des Java-Codes können JNI-Funktionen aufgerufen werden und umgekehrt. Das JNI stellt außerdem Funktionen zum Lesen und Schreiben primitiver Datentypen bereit. So kann beispielsweise das Array mit den Tiefenwerten von Java aus übergeben und im nativen Code mit Werten gefüllt werden. Listing 2 zeigt, wie der JNI-Code Java-Funktionen der Android-Activity aufruft.

```
1 Eigen::Vector3f trans = pose.translation().cast<float>();
2 Sophus::Quaternionf quat = pose.unit_quaternion().cast<float>();
3
4 jclass activity = env->FindClass("de/joshuareibert/dontslamyyourhead/MainActivity");
5 jmethodID trans_setter = env->GetMethodID(activity, "setTranslation", "(FFF)V");
6 jmethodID rot_setter = env->GetMethodID(activity, "setRotation", "(FFFF)V");
7
8 env->CallObjectMethod(thiz, trans_setter, trans[0], trans[1], trans[2]);
9 env->CallObjectMethod(thiz, rot_setter, quat.x(), quat.y(), quat.z(), quat.w());
```

Listing 2: Aus dem JNI-Code werden öffentliche Funktionen der Android-Activity zum Setzen der Trackingwerte aufgerufen.

## 4 Eigene Umsetzung

### 4.2.3 Android NDK und Android Studio

Seit Juli 2015 verfügt Android Studio über eine experimentelle, integrierte Unterstützung des Android NDK. Zum Zeitpunkt der Verwendung war diese Unterstützung allerdings noch nicht vollständig. So war es nicht möglich, Abhängigkeiten zwischen den vorkompilierten Bibliotheken zu definieren [29]. Dies ist aber für den LSD-SLAM notwendig, da dieser von der ebenfalls vorkompilierten g<sup>2</sup>o-Bibliothek abhängt. Außerdem führt der experimentelle Status dazu, dass sich Syntax und Funktionsumfang kurzfristig ändern können.

Daher wurde zunächst auf den integrierten NDK-Support verzichtet. Als Ersatz diente das Tool *ndk-build* des Android NDK. Der native Code wird dafür über *mk*-Dateien konfiguriert. Dabei handelt es sich um Makefile-Fragmente mit zusätzlichen Makros speziell für das NDK. *ndk-build* ruft wiederum *make* mit den entsprechenden *mk*-Dateien auf. Die daraus resultierenden Bibliotheken können dann von der Android-App geladen und verwendet werden.

Das hat zur Folge, dass der vollständige Build-Prozess komplex ist (vgl. Unterabschnitt 4.2.6). Die integrierte NDK-Unterstützung würde diesen vereinfachen und die einzelnen Tools automatisch zusammenbringen. Anstatt diese einzeln ausführen zu müssen, könnte man in Android Studio alle Schritte mit nur einem Knopfdruck ausführen.

### 4.2.4 Abhängigkeiten

Weil Android auf einer anderen Prozessorarchitektur basiert, müssen auch alle Abhängigkeiten für diese kompiliert werden. Erst dann kann der native Code der Android-App auf sie zurückgreifen.

Das Framework g<sup>2</sup>o verwendet android-cmake für die Kross-Komplilierung für Android.<sup>16</sup> Dadurch kann es mit CMake für Android kompiliert werden. Eigen3 besteht nur aus Headern und keiner Bibliothek. Damit kann es ohne Weiteres unter Android verwendet werden.

---

<sup>16</sup><https://github.com/RainerKuemmerle/g2o#cross-compiling-for-android>

## 4 Eigene Umsetzung

Boost hingegen unterstützt Android offiziell nicht. Es kann dennoch mit dem mobilen Betriebssystem verwendet werden. Boost-for-Android<sup>17</sup> ist eine Bash-Skript, das Boost für Android konfiguriert und kompiliert. Allerdings unterstützt Boost-for-Android nicht die aktuelle Version 10e des Android NDK. Daher muss für den gesamten Build-Prozess der Android-App die Version 10d des Android-NDK vom Dezember 2014 verwendet werden.

Der LSD-SLAM greift auf OpenCV für die Matrixklasse zum Speichern der Kamerabilder zurück. Außerdem werden dessen Funktionen für die Rektifizierung und Debugging verwendet wie z. B. das Plotten der Tiefenwerte oder das Abspeichern von Bildern. OpenCV ist mit dem OpenCV4Android-SDK<sup>18</sup> für Android verfügbar. Neben nativen Headern und Bibliotheken ist eine Java-API enthalten. Dadurch kann OpenCV sowohl im nativen Code als auch im Java-Code der Android-App genutzt werden.

Alle benötigten Header und Bibliotheken der Abhängigkeiten werden im JNI-Ordner der Android-App platziert. Von dort aus kann der native Code der App auf diese zugreifen.

### 4.2.5 NEON-Optimierungen

Smartphones verfügen über weniger Prozessorleistung als Desktoprechner. Das ist insbesondere für komplexe Echtzeitanwendungen wie SLAM-Algorithmen ein Problem. Rechenintensive Schritte und Operationen mit großen Datenmengen bremsen das System aus. Die Autoren des LSD-SLAM haben daher architekturspezifische Optimierungen integriert. Diese können aktiviert werden, sofern der SLAM für die entsprechende Architektur kompiliert wird.

Das Nexus 5 hat einen Prozessor der ARMv7-Architektur. ARM bietet mit NEON<sup>19</sup> für diese Prozessoren eine Optimierungstechnologie an. Dabei handelt es sich um eine SIMD-Architekturerweiterung (*single instruction, multiple data*). Das heißt, die gleiche Befehlsfolge wird unabhängig voneinander für mehrere Elemente ausgeführt.

---

<sup>17</sup><https://github.com/MysticTreeGames/Boost-for-Android>

<sup>18</sup>[http://docs.opencv.org/2.4/doc/tutorials/introduction/android\\_binary\\_package/dev\\_with\\_OCV\\_on\\_Android.html](http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html)

<sup>19</sup><http://www.arm.com/products/processors/technologies/neon.php>

## 4 Eigene Umsetzung

Im Quelltext des LSD-SLAM sind NEON-Optimierungen bereits enthalten. Durch das Setzen einer Präprozessor-Direktive werden diese aktiviert. Überall, wo mit NEON optimierte Funktionen vorhanden sind, ersetzen diese dann die ursprünglichen. Für die Kompilierung muss mit CMake zusätzlich noch ein Compilerflag gesetzt werden.

NEON-Optimierungen können im LSD-SLAM an einigen Stellen eingesetzt werden. So beschleunigen sie Matrixberechnungen für das Tracking. Insbesondere aber Operationen, die pro Pixel ausgeführt werden, profitieren von der Parallelität. Die dadurch erreichte Steigerung der Performance ist wichtig, damit der LSD-SLAM auch mobil Echtzeit erreicht [26].

### 4.2.6 Build-Prozess

Der gesamte Build-Prozess der Android-App setzt sich schließlich aus drei Schritten zusammen.

1. Die benötigten Bibliotheken und deren Header müssen gegebenenfalls für Android kompiliert werden. Danach werden sie in den JNI-Ordner kopiert.
2. Der LSD-SLAM wird mit android-cmake als *shared library* für Android kompiliert. Die Installation mit `make install` kopiert die resultierenden Bibliotheken ebenfalls in den JNI-Ordner der Android-App.
3. Android Studio kompiliert die App mit Gradle. Das Gradle-Skript wurde um eine Funktion erweitert, die vor dem Kompilieren das Tool *ndk-build* des Android NDK auruft. Dadurch wird der JNI-Code neu kompiliert, wenn Änderungen am nativen Code oder den Bibliotheken vorliegen.

## 4 Eigene Umsetzung

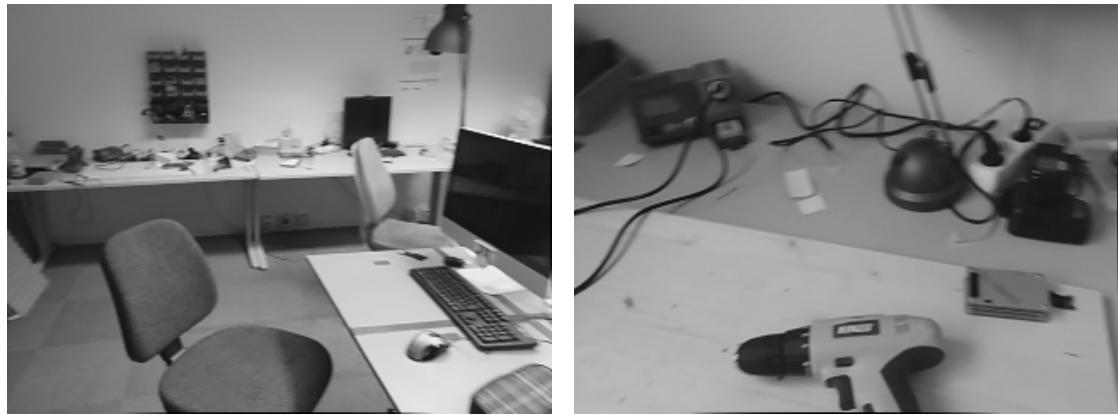


Abbildung 4.3: Zwei Keyframes mit motion blur durch zu schnelle Bewegung des Smartphones. Eigene Abbildungen

### 4.2.7 Kamera

In Smartphones werden üblicherweise CMOS-Bildsensoren verbaut. Diese zeichnen sich durch einen niedrigen Preis und geringen Stromverbrauch aus. Ein großer Nachteil ist aber der Rolling-Shutter-Effekt. Dieser entsteht dadurch, dass der Sensor das Bild zeilenweise ausliest. Jede Zeile wird leicht zeitverschoben belichtet und ausgelesen. Das führt zu Verzeichnungen, wenn sich das Motiv oder die Kamera selbst während der Aufnahme schnell bewegen [9]. 3D-Rekonstruktionstechniken nehmen aber einen globalen Shutter an, also eine gleichzeitige Belichtung aller Pixel. Die Verzeichnungen führen daher zu Fehlern beim *stereo matching*.

[...] we found that ignoring the rolling shutter still gives very good results in practice [...]

— Schöps, Engel und Cremers [26]

Es existieren Methoden, um den Rolling-Shutter-Effekt in Echtzeit zu korrigieren. Dennoch verzichten Schöps, Engel und Cremers zugunsten der Performance darauf, weil die Ergebnisse in der Praxis gut genug seien [26]. Schnelle Bewegungen des Smartphones führen auch zu *motion blur* (vgl. Abbildung 4.3). Beide Effekte können dazu führen, dass das Tracking verloren geht. Allzu schnelle Bewegungen sollten daher generell vermieden werden.

## 4 Eigene Umsetzung

Für den Zugriff auf die Kamera des Smartphones wird OpenCV verwendet. Es ist als Abhängigkeit des LSD-SLAM ohnehin bereits eingebunden. Das Kamerabild kann direkt als OpenCV-Matrix empfangen und in diesem Format an den JNI-Code weitergegeben werden. Allerdings verwendet OpenCV Androids Camera-Klasse, die inzwischen veraltet ist. Der Zugriff auf die Kamerabilder erfolgt über die Vorschaufunktion. Diese ist auf 7 bis 30 Bilder pro Sekunde beschränkt. Die tatsächlich genutzte Bildfrequenz schwankt zwischen diesen Werten.

### 4.2.8 Kalibrierung und Rektifizierung

Auf die gleiche Weise wie in Unterabschnitt 4.1.5 wird auch die Kamera des Nexus 5 kalibriert. Die dabei ermittelten Kalibrierungsparameter werden als Konstanten im JNI-Code gespeichert. Dort wird die `Undistorter`-Klasse des LSD-SLAM angelegt. Diese rektifiziert die eingehenden Kamerabilder, bevor sie an den SLAM weitergegeben werden.

Durch die Rektifizierung entstehen pixelfreie Ränder. Der `Undistorter` schneidet die Bilder so zu, dass diese Bereiche reduziert werden. Das Bild müsste größer skaliert werden, um wieder die Eingangsauflösung zu erhalten. Daher werden die Bilder in einer Auflösung von  $640 \times 480$  Pixeln aufgenommen. Der `Undistorter` gibt dann rektifizierte,  $320 \times 240$  Pixel große Bilder zurück. Diese Auflösung wird in „*Semi-dense visual odometry for AR on a smartphone*“ für die Verwendung des LSD-SLAM auf einem Smartphone empfohlen [26].

### 4.2.9 Android-App

Die Android-App ist als Release<sup>20</sup> im GitHub-Repository verfügbar. Sie erlaubt eine grundlegende mobile Nutzung des LSD-SLAM auf einem Smartphone. Dieser wird als native Bibliothek genutzt. Der JNI-Code ist die Schnittstelle zwischen der App und nativem Code. Er nimmt die Kamerabilder von der App entgegen. Nach der Rektifizierung durch den `Undistorter` werden diese an den LSD-SLAM weitergegeben.

---

<sup>20</sup><https://github.com/mmbuw/dont-slam-your-head/releases/tag/android>

## 4 Eigene Umsetzung

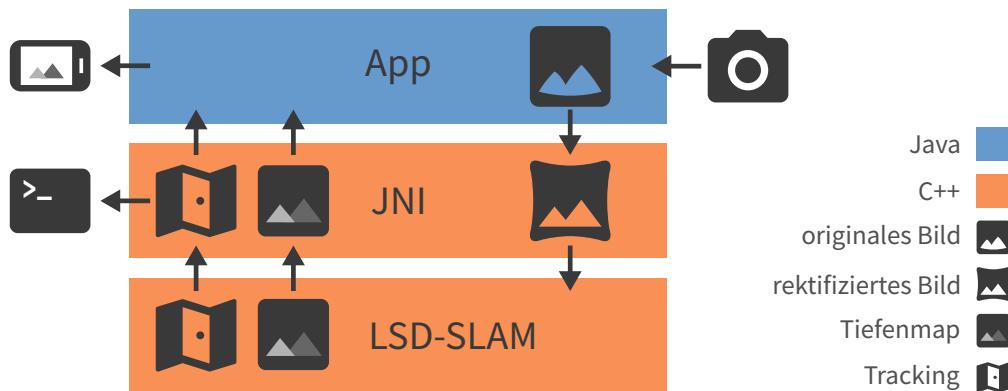


Abbildung 4.4: Der Aufbau und Datenfluss der Android-App. Eigene Abbildung

Dort werden die Bilder verarbeitet und die Ergebnisse zurückgegeben. Dabei handelt es sich um die getrackte Kamerapose sowie die Tiefendaten für den aktuellen Keyframe. Diese bestehen aus dem inversen Tiefenwert und der zugehörigen Varianz für jedes Pixel mit ausreichendem Gradienten.

Die Kamerapose wird zunächst nur über Androids Logging-System ausgegeben. Das Tiefenbild hingegen wird an die App weitergereicht und dort mit OpenCV angezeigt (vgl. Abbildung 4.5).

## 4.3 Unity-Plugin

Mit dem Android-Port ist der LSD-SLAM auch mobil verfügbar. Aber das Programmieren von VR-Anwendungen direkt in nativem Android-Code ist mühsam und komplex. Spiele-Engines erfreuen sich deshalb großer Beliebtheit unter Hobby-Entwicklern. Sie bieten eine einfache Möglichkeit, innerhalb kürzester Zeit VR-Apps zu entwickeln. Daher soll der mobile LSD-SLAM für die Spiele-Engine Unity<sup>21</sup> als Android-Plugin zur Verfügung gestellt werden.

<sup>21</sup><http://unity3d.com/>

#### 4 Eigene Umsetzung

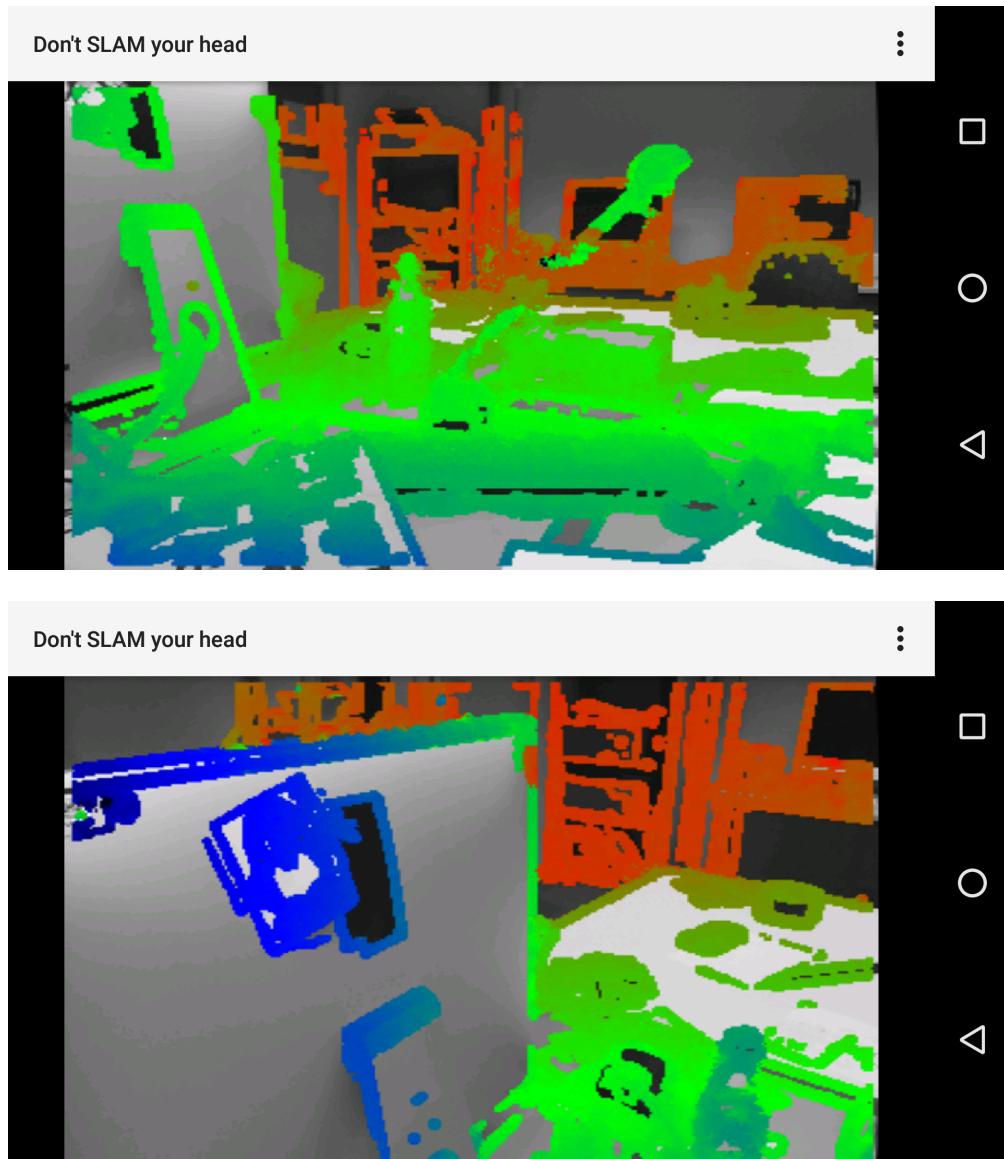


Abbildung 4.5: Screenshots der Android-App, die den aktuellen Keyframe mit den ermittelten inversen Tiefenwerten als rainbow plot anzeigt. Eigene Abbildungen

## 4 Eigene Umsetzung

Unity ist eine äußerst beliebte Spiele-Engine, die von 4,5 Millionen Entwicklern genutzt wird [28]. Das liegt zum einen daran, dass die Nutzung prinzipiell kostenlos ist. Erst wenn erweiterte Funktionalität benötigt oder die Einkommensgrenze von \$100 000 überschritten wird, muss eine monatlich zu bezahlende, professionelle Lizenz erworben werden.<sup>22</sup> Zum anderen bedient Unity eine sehr große Anzahl von Plattformen. Spiele können mit dem Editor der Engine ohne großen Mehraufwand für zahlreiche Plattformen zeitgleich entwickelt werden.

### 4.3.1 Android Plugin

Unity unterstützt bereits standardmäßig Android als Zielplattform. Mit dem Editor erstellte Spiele können direkt für Android exportiert werden. Auf tiefergehende, plattformspezifische Funktionen kann über Plugins zugegriffen werden. So zeigen z. B. viele Spiele Buttons zum Teilen in sozialen Netzwerken an, die per Android-Plugin umgesetzt werden.

Ein Android-Plugin setzt auf einer Android-Activity auf. Diese erbt von einer Basisklasse von Unity. Innerhalb der Activity kann die gewünschte Funktionalität mit den üblichen Mitteln der Android-Entwicklung implementiert werden. Öffentliche Funktionen sind später auch von Unity-Skripten aus erreichbar. Die Activity wird nicht als Android-App kompiliert, sondern als Java-Archiv (.jar).<sup>23</sup> Dieses wird zusammen mit dem Android-Manifest im Android-Ordner von Unity abgelegt. Beim Kompilieren der App mit Unity wird dann die Activity des Plugins als Basis für die App verwendet.

Auch die Integration des LSD-SLAM geschieht in Form eines Android-Plugins. Das Gradle Build-Skript wird so modifiziert, dass Android Studio die Activity als Java-Archiv kompiliert. Die Activity stellt Funktionen zum Zugriff auf die Daten des SLAM bereit. Darüber greifen Unity-Skripte auf die getrackte Position, die Tiefenmap und die Punktwolke zu.

---

<sup>22</sup><http://unity3d.com/get-unity>

<sup>23</sup><http://docs.unity3d.com/Manual/PluginsForAndroid.html>

## 4 Eigene Umsetzung

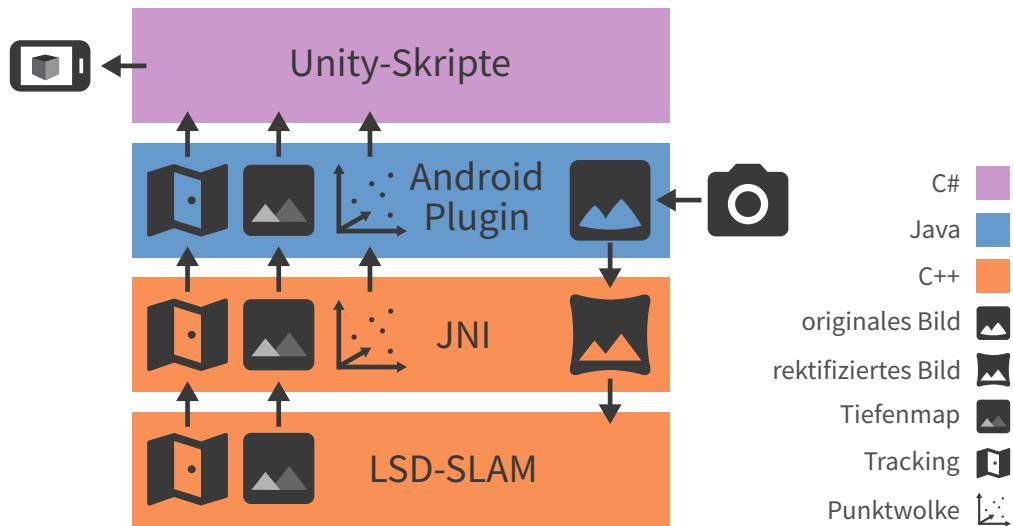


Abbildung 4.6: Der Aufbau und Datenfluss der Unity-App. Eigene Abbildung

### 4.3.2 Cardboard SDK

Cardboard ist Googles VR-Plattform. Seit dem Prototypen für eine Pappkonstruktion, die das Smartphone in ein HMD verwandelt, hat sich Cardboard weiterentwickelt. Inzwischen gibt es mehrere Generationen von Viewern und eine große Anzahl von Herstellern. Google stellt außerdem Tools für die Entwicklung von VR-Apps zur Verfügung. Darauf wurde auch für diese Arbeit zurückgegriffen.

Mit dem Cardboard SDK<sup>24</sup> unterstützt Google auch Unity. Das SDK kann als Unity-package heruntergeladen und zu einem vorhandenen Unity-Projekt hinzugefügt werden. Ein enthaltenes Beispielprojekt kann dabei als Basis eigener Projekte dienen. Dieses kam auch als Grundlage der hier entwickelten App zum Einsatz.

Mit dem Cardboard SDK kann man aus Unity direkt mit der Entwicklung von VR-Apps beginnen. Das SDK liefert C#-Skripte, Shader und plattformspezifischen Code mit. Diese ermöglichen Unity das Rendern für Cardboard Viewer (vgl. Abbildung 4.7). Dafür wird die Szene als Stereobildpaar gerendert. Die typische tonnenförmige Verzeichnungskorrektur hebt die Verzeichnung durch die Linsen auf. Für das Rotationstracking werden die Daten der IMU des Smartphones per *sensor fusion* kombiniert.

<sup>24</sup><https://developers.google.com/cardboard/unity/>

## 4 Eigene Umsetzung

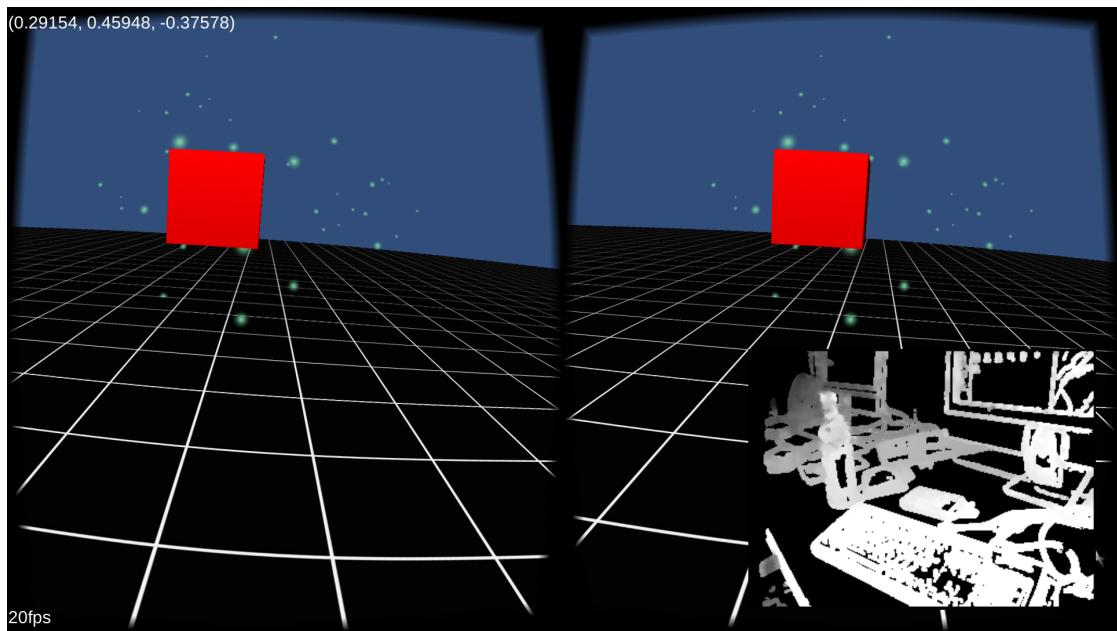


Abbildung 4.7: Die minimalistische Beispielszene des Cardboard SDK. Die Tiefenmap des LSD-SLAM wird als überlagerte Textur angezeigt. Eigene Abbildung

### 4.3.3 Tracking

Das Tracking des Cardboard SDK erfasst nur die Kopfrotation. Die Position im Raum kann über die IMU-Daten des Smartphones nicht genau genug bestimmt werden [11]. Aufgrund fehlender Fixpunkte summiert sich der Fehler der Sensoren auf. Die getrackte Position wird dadurch innerhalb kurzer Zeit unbrauchbar. Beim visuellen SLAM hingegen werden Bildinformationen für das Tracking genutzt. Damit gibt es Fixpunkte in der realen Welt.

Daher wird für die App auf das Tracking des LSD-SLAM zurückgegriffen. Dieser gibt die getrackte Pose für jedes Kamerabild zurück. Die Pose wird in Position und Orientierung getrennt und an die Activity weitergegeben. Dort werden die Werte gespeichert und über öffentliche Funktion nach außen verfügbar gemacht.

Über diese Funktionen können die C#-Skripte von Unity auf die Werte zugreifen. Das Skript `SLAM.cs` wird im Unity-Projekt an die Hauptkamera angehängt. Über Flags können Positions- und Rotationstracking einzeln aktiviert werden. Wenn diese gesetzt sind, wird in der `Update`-Funktion des Skripts die Position beziehungsweise Orientierung der Hauptkamera entsprechend aktualisiert.

## 4 Eigene Umsetzung

Das Tracking des LSD-SLAM hat den Nachteil, dass sich die Werte gelegentlich abrupt ändern. Insbesondere für die Kopfrotation ist das sehr unangenehm. Deshalb werden in der App letztlich das Positionstracking des LSD-SLAM und das Rotationstracking des Cardboard SDK kombiniert. Langfristig entstehen dabei Abweichungen zwischen Kopfrotation und -position, da beide unterschiedlich stark driften. Daher ist es wünschenswert, zukünftig das Rotationstracking des LSD-SLAM zu integrieren. Stabilisierungen mit den IMU-Daten könnten die sprunghaften Änderungen verhindern.

Ein weiteres, optionales C#-Skript zeigt die getrackte Position zum Debuggen an (vgl. Abbildung 4.7 links oben). Es wird einem GUI-Textlabel hinzugefügt und setzt den Text entsprechend.

### 4.3.4 Tiefenmap

Die Tiefenmap des LSD-SLAM ist eigentlich nur ein Zwischenprodukt auf dem Weg zu den tatsächlichen Punkten im Raum. Sie wird vom LSD-SLAM bei jedem Kamerabild als Array mit einem Wert pro Pixel zurückgegeben. Pixel ohne gültigen Tiefenwert – wie z. B. in homogenen Bereichen – haben einen negativen Tiefenwert. Der JNI-Code gibt die Tiefenwerte an die Android-Activity weiter.

Das C#-Skript `DepthTexture.cs` greift auf diese Tiefenwerte zu. Es mappt die Werte die Tiefenwerte im Bereich von 0 bis 1 auf die Intensitätswerte einer Graustufentextur. Diese wird dann als überlagertes GUI-Element in der App angezeigt (vgl. Abbildung 4.7 rechts unten). Das Skript lässt sich leicht deaktivieren und dient nur als Informationsquelle für die Entwicklung.

### 4.3.5 Punktwolke

Die Berechnung der Punktwolke erfolgt aus Performancegründen im JNI-Code. Da später ohnehin nicht alle Punkte angezeigt werden können, wird nur eine Teilmenge tatsächlich berechnet. Es wird eine gleichmäßig verteilte Teilmenge der Pixel ausgewählt und nach ihrer Tiefenvarianz sortiert. Davon werden die 3D-Positionen für die 500 Punkte mit der niedrigsten Varianz berechnet. Die Anzahl der Punkte ist als Konstante angelegt. Sie kann also bei Bedarf angepasst werden.

## 4 Eigene Umsetzung

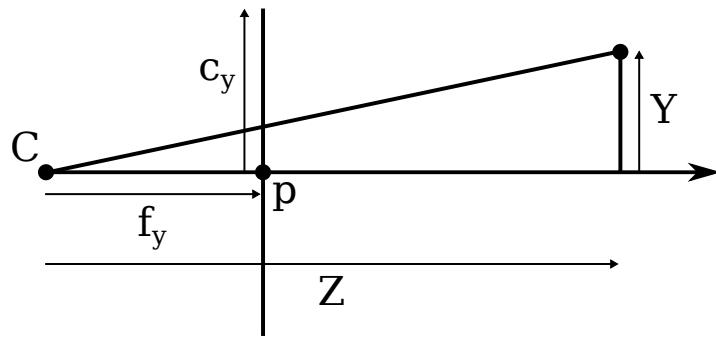


Abbildung 4.8: Seitenansicht der Geometrie des Lochkameramodells. Eigene Abbildung nach [4]

Die Geometrie des Lochkameramodells dient als Grundlage für die Berechnung (vgl. Abbildung 4.8). Aus der vom LSD-SLAM ermittelten inversen Tiefe  $d$  ergibt sich die Tiefe als  $Z = 1/d$ . Da es sich bei den beiden Dreiecke in Abbildung 4.8 um ähnliche Dreiecke handelt, sind die Seitenverhältnisse gleich. Gleichung 4.1 setzt das am Beispiel der Y-Komponente um.

$$\frac{Y}{Z} = \frac{y - c_y}{f_y} \quad (4.1)$$

$y$  ist dabei die vertikale Pixelkoordinate des Bildpunktes, also für die verwendete Auflösung ein Wert zwischen 0 und 240. Die Subtraktion von  $c_y$  zentriert diese Koordinate um den Bildmittelpunkt. Durch Umstellen der Gleichung erhält man  $Y = (y - c_y) \cdot f_y^{-1} \cdot Z$ . Die X-Komponente der 3D-Position wird analog berechnet. Damit können alle drei Komponenten der Position des Punktes relativ zum Kamerazentrum  $C$  berechnet werden.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} (x - c_x) \cdot f_x^{-1} \\ (y - c_y) \cdot f_y^{-1} \\ 1 \end{pmatrix} \cdot Z \quad (4.2)$$

## 4 Eigene Umsetzung

```
1 typedef std::pair<float, int> P;
2 std::priority_queue<P, std::vector<P>, std::greater<P>> q;
3 for (int i = 0; i < out_width * out_height; i += 13) {
4     q.push(std::pair<float, int>(idepthVar[i], i));
5 }
6 for (int i = 0; i < numPoints; ++i) {
7     int idx = q.top().second;
8     int x = idx % out_width;
9     int y = idx / out_height;
10    float depth = 1.0 / idepthMap[idx];
11    Sophus::Vector3f pos((x * fxi + cxi) * depth, (y * fyi + cyi) * depth, depth);
12    Sophus::Vector3f world_pos = camToWorld.cast<float>() * pos;
13    points[i * 3] = world_pos[0];
14    points[i * 3 + 1] = world_pos[1];
15    points[i * 3 + 2] = world_pos[2];
16
17    q.pop();
18 }
```

Listing 3: Die Funktion für die Berechnung der 3D-Positionen der stabilsten Punkte.

Diese Position ist allerdings noch im Koordinatensystem der Kamera. Für den jeweils aktuellen Keyframe wird im JNI-Code die Matrix gespeichert, die das Kamerazentrum in den Ursprung des Weltkoordinatensystems des LSD-SLAM verschiebt. Die Multiplikation der berechneten Position mit dieser Matrix ergibt die absolute Position des Punktes innerhalb der Punktwolke. Die resultierenden drei Komponenten aller 500 Punkte werden als Array an die Android-Activity weitergegeben.

Das Koordinatensystem des LSD-SLAM weicht aber auch noch vom Koordinatensystem von Unity ab. Dadurch muss die Punktposition im Unity-Skript nochmals transformiert werden. Der LSD-SLAM setzt die Pose des ersten Kamerabildes als Ursprung seines Koordinatensystems. Die Koordinaten hängen also davon ab, wie das Smartphone beim Start der App gehalten wird. Als rechtshändiges Koordinatensystem zeigt die X-Achse nach rechts, die Y-Achse nach unten und die Z-Achse nach hinten.

Das Koordinatensystem von Unity hingegen ist linkshändig. Die Y-Achse zeigt dort nach oben. Für die Positionen aus dem LSD-SLAM muss also die Y-Komponente invertiert werden. Das Cardboard SDK setzt die initiale Kopfrotation, so dass X- und Z-Achse die Grundfläche aufspannen. Die Y-Achse steht dazu orthogonal und umgekehrt zur Erdanziehungskraft. Dadurch ist nur die Rotation um die Y-Achse frei und wird auf Null gesetzt.

## 4 Eigene Umsetzung

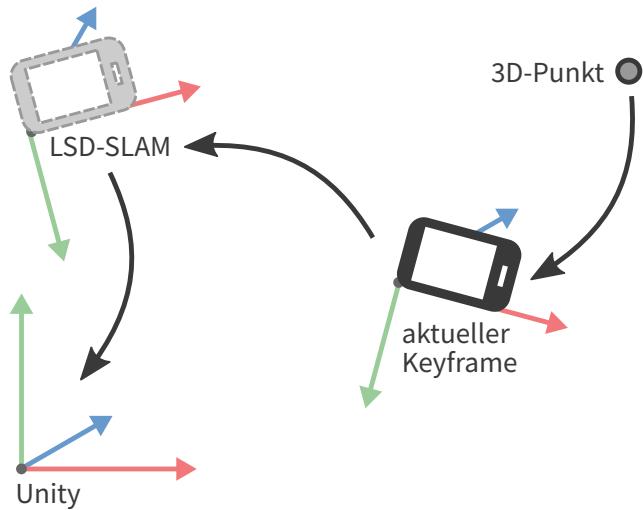


Abbildung 4.9: Die verschiedenen Koordinatensysteme innerhalb der Unity-App.

Eigene Abbildung

Das Unity-Skript `BoxCloud.cs` visualisiert die Punktwolke, indem es Boxen an den Punktposition rendert (vgl. Abbildung 4.10). Dazu speichert es zunächst die initiale Kopfrotation vom Cardboard SDK. In der Update-Funktion greift es auf die Punktdaten der Android-Activity zu. Es invertiert die Y-Komponente, um das rechtshändige in ein linkshändiges Koordinatensystem zu transformieren. Mit der gespeicherten initialen Rotation wird die Neigung des Smartphones beim Start der App kompensiert.

```
1 Vector3 pos = initialRot * (new Vector3(x, -y, z) * cloudScale);
2 cloud[i].GetComponent<Transform>().localPosition = pos;
3 float distance = Mathf.Clamp01(
4     (cloud[i].GetComponent<Transform>().position - head.position).magnitude
5 );
6 cloud[i].GetComponent<Renderer>().enabled = true;
7 cloud[i].GetComponent<Renderer>().material.color =
8     Color.Lerp(Color.red, Color.green, distance);
```

Listing 4: Die Berechnung von Position und Farbe eines einzelnen Punktes für die Anzeige in Unity.

## 4 Eigene Umsetzung

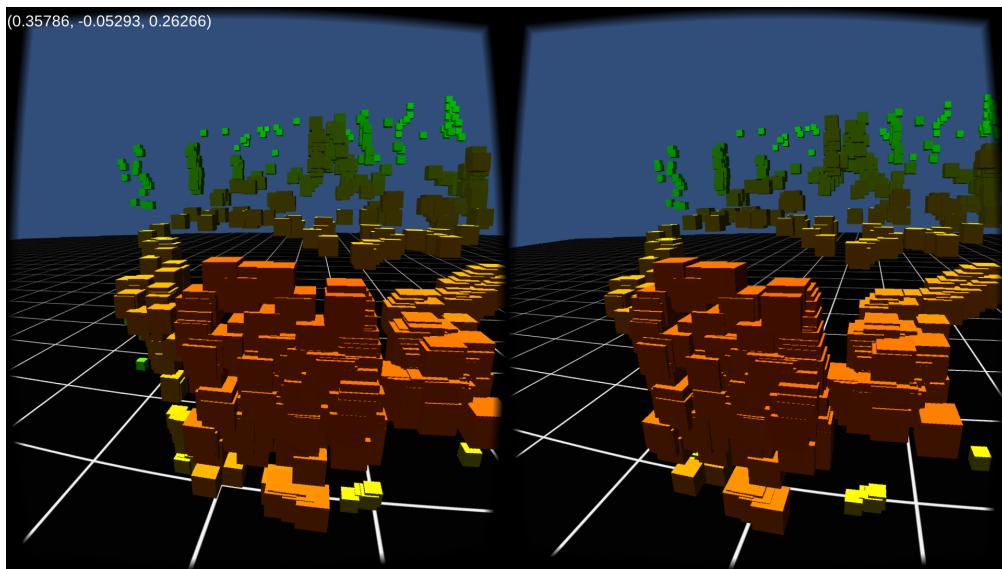


Abbildung 4.10: Die von Unity angezeigte Visualisierung der 3D-Punkte als Boxen. Der Abstand zur Kamera ist farbkodiert. Nahe Punkte sind rot, weit entfernte grün. Eigene Abbildung

Das Rendern der Boxen ist eine naive Methode, um die Punktwolke zu visualisieren. Doch die hat den Vorteil, dass sie mit Unity sehr einfach zu skripten ist. Die Unity-App büßt dadurch aber deutlich Performance ein. Unity selbst bietet keine Möglichkeit, Punktwolken zu rendern. Mit existierenden Plugins oder eigenen Shadern kann die Punktwolke eleganter und performanter dargestellt werden. Für die vorliegende Arbeit lag der Fokus aber zunächst darauf, die Punktdaten zu berechnen und von Unity aus auf sie zuzugreifen.

Wie in Kapitel 3 bereits erwähnt, kann die Skalierung der Punktwolke des LSD-SLAM nicht bestimmt werden. Das BoxCloud-Skript erlaubt daher das Setzen eines Skalierungsfaktors. Dieser wird auf die Position der Punkte im Koordinatensystem des LSD-SLAM multipliziert. Als Standardwert ist der Skalierungsfaktor Eins und damit ohne Effekt.

## 4 Eigene Umsetzung

Die Skalierung der Koordinaten des LSD-SLAM hängt von der Tiefe des ersten Keyframes ab. Für diesen wird eine Skalierung vom Faktor Eins festgelegt. Die inversen Tiefenwerte werden dann so bestimmt, dass die durchschnittliche inverse Tiefe ebenfalls Eins ist. Das erlaubt es, die absolute Skalierung des LSD-SLAM über die Initialisierung festzulegen. Wenn der erste Keyframe durchschnittlich eine Tiefe von einem Meter hat, entspricht eine Einheit des Koordinatensystems des LSD-SLAM ebenfalls einem Meter. In Tests hat das ausreichende Ergebnisse hervorgebracht. Ein zusätzliches Initialisierungsverfahren könnte dieses Problem zukünftig eleganter lösen.

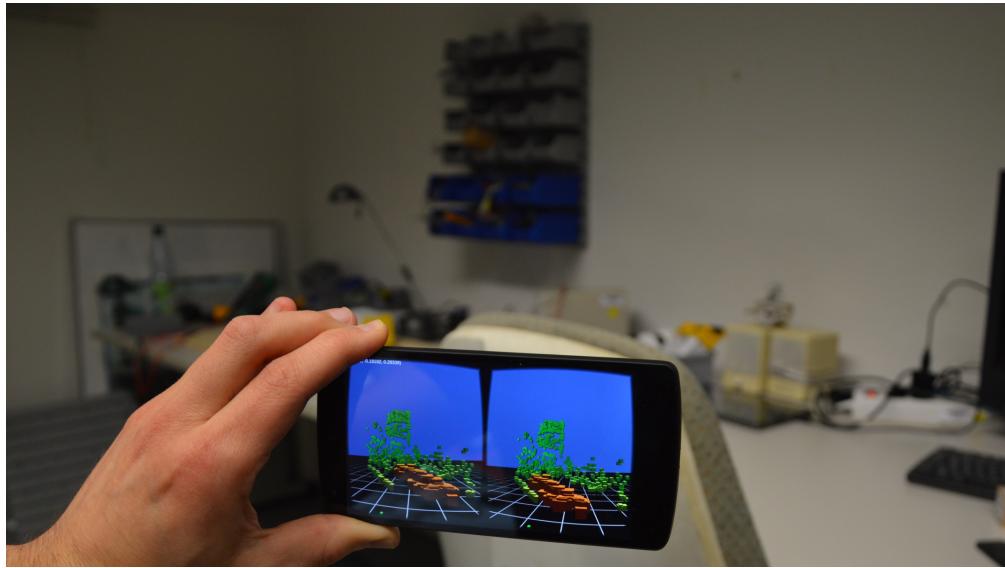
### 4.3.6 Unity-Projekt

Das Unity-Projekt inklusive der verwendeten C#-Skripte und Dateien des Android-Plugins ist ebenfalls im GitHub-Repository<sup>25</sup> enthalten. Es setzt auf dem vorher umgesetzten Android-Port des LSD-SLAM auf. Das Android-Plugin erlaubt einen Zugriff auf die Tracking- und Punktdaten des SLAM. Damit können in Unity leicht VR-Anwendungen erstellt werden, die Umgebungsinformationen nutzen. Eine einfache Visualisierung der Punktfolge erfolgt über ein C#-Skript. Das erlaubt es, Strukturen der realen Umgebung wiederzuerkennen.

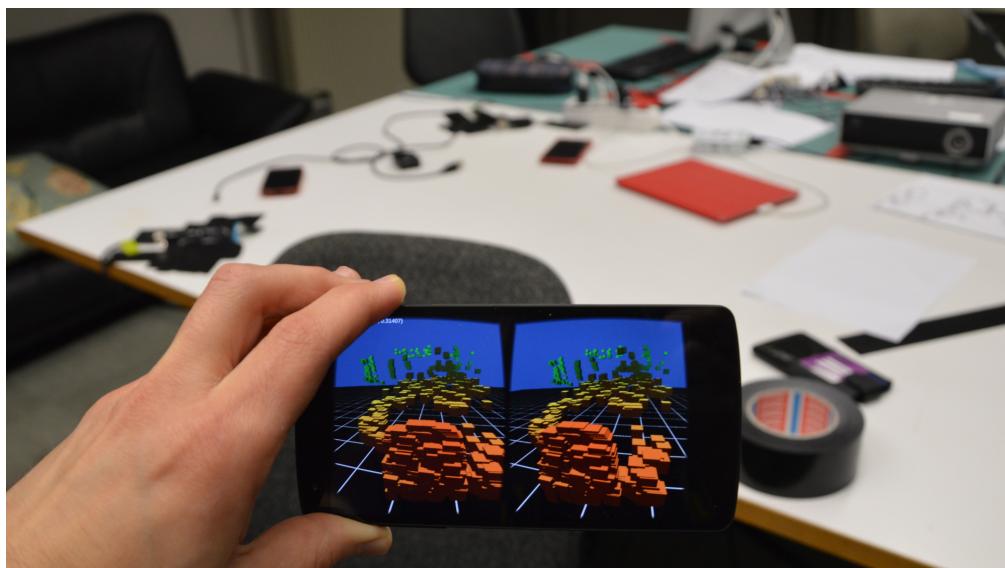
---

<sup>25</sup><https://github.com/mmbuw/dont-slam-your-head/tree/master/unity>

#### 4 Eigene Umsetzung



(a) Auf dem Display ist im Vordergrund in Orange die Stuhllehne zu sehen. Im Hintergrund zeichnet sich klar das Wandregal ab.



(b) Wieder ist im Vordergrund in Orange eine Stuhllehne zu sehen. Die freien Flächen des Tisches erzeugen keine Punkte. Nur die Gegenstände auf dem Tisch sind in Gelb zu sehen. Die weit entfernten grünen Objekte und Kanten sind im Foto nicht mehr sichtbar.

Abbildung 4.11: Die Visualisierung eines Ausschnitts der Punktfolge mit der Unity-App und die dazugehörige reale Umgebung. Eigene Abbildungen

## 5 Auswertung und Ausblick

Die umgesetzte mobile Implementierung des LSD-SLAM soll im Kontext von VR-Anwendungen genutzt werden. Sie zielt daher darauf ab, in Echtzeit Umgebungsinformationen zur Verfügung zu stellen. Das Einlesen und Verarbeiten von Videodateien ist nicht möglich. Damit kann auch keine quantitative Auswertung mit bekannten Datensätzen für die Evaluierung von SLAM-Algorithmen verwendet werden. Die Ergebnisse des auf Android portierten LSD-SLAM wurden deshalb rein qualitativ ausgewertet.

Der größte Nachteil bei der Verwendung eines monokularen SLAM-Algorithmus ist die unbekannte absolute Skalierung. Diese kann bislang nur mit Wissen über die Umgebung beim ersten Keyframe grob beeinflusst werden. Es fehlen Referenzgrößen, die eine tatsächlich Berechnung des Skalierungsfaktors ermöglichen. Solange das nicht gegeben ist, sind die Tiefenwerte des LSD-SLAM immer nur eine Annäherung an die tatsächlichen Werte.

Eine mögliche Lösung ist ein Initialisierungsverfahren beim Start der App. Der SLAM-Algorithmus PTAM erfordert z. B. eine anfängliche Translationsbewegung von 10 cm zwischen zwei Nutzereingaben [8]. Weil dadurch der Abstand zwischen den beiden Bildern bekannt ist, kann der Skalierungsfaktor berechnet werden. Das setzt aber voraus, dass die Kamera wirklich um genau diese Distanz bewegt wurde. Für den Nutzer einfacher wäre eine Initialisierung mit Markern bekannter Größe wie etwa einem QR-Code. Beim Start der App müsste dieser sichtbar sein und von der App erkannt werden. Darüber ist dann ebenfalls eine Berechnung der Skalierung möglich.

## 5 Auswertung und Ausblick

Die Unity-App kann auf die Trackingwerte des LSD-SLAM über das Android-Plugin zugreifen. Das ermöglicht das Tracken der Position des Smartphones im Raum. Mit den IMU-Daten ist es dagegen nur möglich, die Orientierung des Smartphones zu bestimmen. Allerdings kann sich die vom SLAM getrackte Orientierung sehr sprunghaft ändern. Das passiert insbesondere dann, wenn die Kamera hauptsächlich rotiert wird. Dann ist es nicht möglich, neue Tiefeninformationen zu berechnen.

Letztlich wurde daher das Positionstracking vom SLAM mit dem Rotationstracking des Cardboard SDK kombiniert. Die Kombination des Trackings birgt die Gefahr, dass die Werte unterschiedlich stark drifteten. Das führt dann zu fehlerhaftem Tracking. Insbesondere bei längerer Verwendung der App werden diese Abweichungen im Tracking deutlich. Die dargestellte Punktwolke ist in solchen Fällen nicht mehr ganz vor der Kamera, sondern seitlich verschoben.

Deshalb ist es wünschenswert, das gesamte Tracking vom LSD-SLAM zu beziehen. Die IMU des Smartphones könnte nach dem Vorbild anderer mobiler SLAM-Algorithmen zur Stabilisierung des Trackings verwendet werden [32, 17]. So könnte bei ungewöhnlich großen Sprüngen im Tracking oder zu viel Kamerarotation kurzfristig auf die Sensordaten zurückgegriffen werden. Im Idealfall werden so korrigierte Trackingdaten auch wieder an den SLAM zurückgereicht. Das würde Folgefehler im Mapping verhindern.

Der LSD-SLAM erzeugt sehr dichte Punktwolken der Umgebung. Jeder Pixel mit einem Tiefenwert in jedem Keyframe entspricht einem erkannten 3D-Punkt. Nur für homogene Bildbereiche kann die Tiefe nicht berechnet werden. Die resultierende Punktwolke ist zu groß, um komplett von einem Smartphone genutzt werden zu können. Darum wird nur eine sehr kleine Teilmenge der Punkte des aktuellen Keyframes berechnet und in der App angezeigt (vgl. Abbildung 4.11). Dennoch ist es möglich, einige Strukturen der Umgebung deutlich zu identifizieren.

In der aktuellen Umsetzung werden diese Punkte für jedes Kamerabild neu berechnet. Dafür werden sie zuvor nach ihrer Tiefenvarianz sortiert, um die stabilsten Punkte auszuwählen. Als Optimierung könnten die Tiefenwerte bereits im LSD-SLAM in einer sortierten Struktur gespeichert werden. Dadurch ist es immer noch möglich, alle Tiefeninformationen abzurufen. Anwendungen, die diese Daten in Echtzeit verarbeiten, könnten aber gleichzeitig leicht auf eine Teilmenge zugreifen.

## 5 Auswertung und Ausblick

Im nativen Teil der Android-App sind alle berechneten Informationen des SLAM verfügbar. Der LSD-SLAM enthält bereits eine Funktion, um Informationen zum *pose graph*, Tiefen- und Bilddaten zu exportieren. Diese wurde um das Exportieren der gesamten Punktwolke als csv-Datei erweitert. Dazu wird über alle Keyframes iteriert und dessen ID und Pose sowie alle Bildpunkte mit validem Tiefenwert in eine Datei geschrieben. Für jeden Punkt werden der Index in der Tiefenmap, der inverse Tiefenwert, der zugehörige Varianzwert und der Intensitätswert des Pixels aus dem Kamerabild gespeichert.

Zur Visualisierung dieser Daten wurde eine OpenGL-Applikation<sup>1</sup> zum Betrachten von Punktwolken geschrieben. Diese liest die csv-Dateien mit den Tiefeninformationen ein und berechnet daraus die Punktwolke (vgl. Unterabschnitt 4.3.5). Mit der Anwendung kann die Qualität vom Smartphone aufgenommener Punktwolken bewertet werden. Insbesondere werden auch die Keyframe-Posen angezeigt. Das erlaubt es, den Kamerapfad und eventuelle Fehler im Tracking nachzuvollziehen.

Abbildung 5.1 zeigt ein Kamerabild und zwei Screenshots der gerenderten Punktwolke eines Arbeitstisches. Die Szene ist relativ klein, dadurch ist die Punktwolke sehr detailliert. Hier zeigt sich, dass die freie Skalierung monokularer SLAM-Algorithmen auch ein Vorteil ist. Die Kinect<sup>2</sup> als Beispiel einer Tiefenkamera hat eine Reichweite von etwa 80 cm bis 4 m. Feine Strukturen wie z. B. die Zahlen auf dem Akkubohrer könnte diese daher nicht erfassen. Der LSD-SLAM hingegen bestimmt für jeden Keyframe eine Skalierung. Bei nahen Objekten ist der Skalierungsfaktor klein. In der gleichen Szene kann der SLAM aber auch ganze Gebäude erfassen. Für diese Keyframes ist dann der Skalierungsfaktor deutlich größer.

Die Seitenansicht der Punktwolke in Abbildung 5.1c zeigt die Streuung der Tiefenwerte. Fehler in der Berechnung der Tiefe führen dazu, dass einige der Werte stark abweichen. Solche Werte weisen aber auch eine größere Varianz auf und können darüber herausgefiltert werden. Dadurch wird die Punktwolke korrekter, aber auch weniger dicht.

Punktwolken, die über einen längeren Pfad entstanden sind, zeigen häufig einen deutlichen *drift*. Abbildung 5.2 zeigt eine solche fehlerhafte Punktwolke. Bei der Aufnahme wurde der Tisch in zügiger Schrittgeschwindigkeit umrundet. Eigentlich müsste der Kamerapfad also ein geschlossener Kreis sein. Wie in der Abbildung zu erkennen, verläuft der Pfad aber spiralförmig.

---

<sup>1</sup><https://github.com/el-josho/point-cloud-viewer>

<sup>2</sup><https://msdn.microsoft.com/en-us/library/hh438998.aspx>

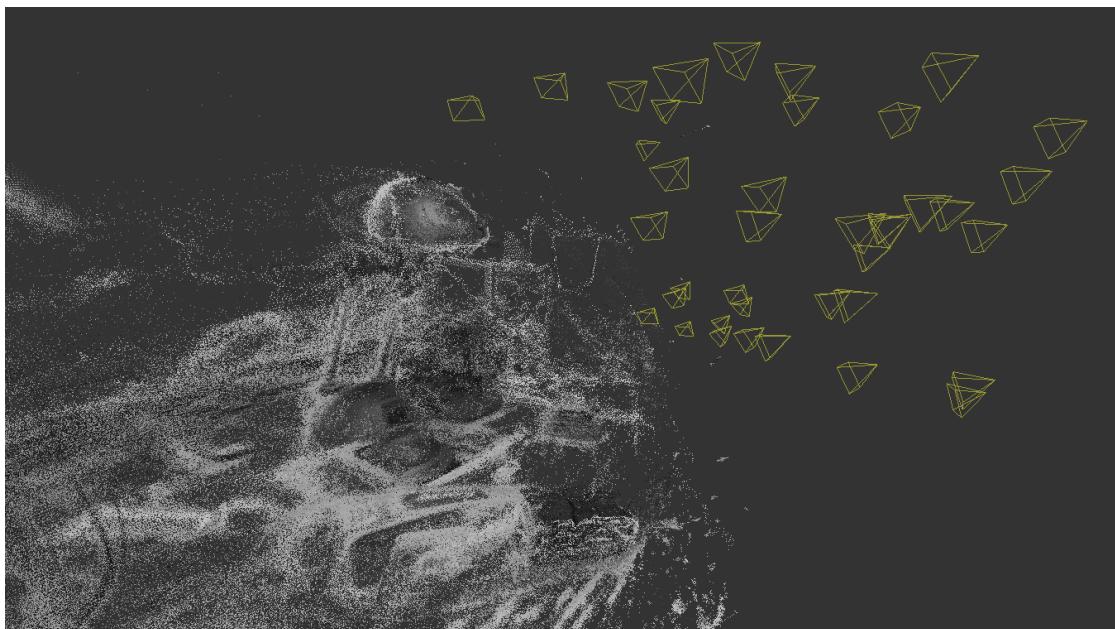
## 5 Auswertung und Ausblick



(a) Einer der verwendeten Keyframes.



(b) Die gerenderte Punktwolke aus ähnlicher Perspektive.



(c) Die Seitenansicht der Punktwolke mit eingezeichneten Keyframe-Posen.

Abbildung 5.1: Ein Kameraframe und zwei Screenshots des Punktwolken-Viewers. Die Punktwolke besteht aus insgesamt 38 Keyframes und 986 353 Punkten. Eigene Abbildungen

## 5 Auswertung und Ausblick

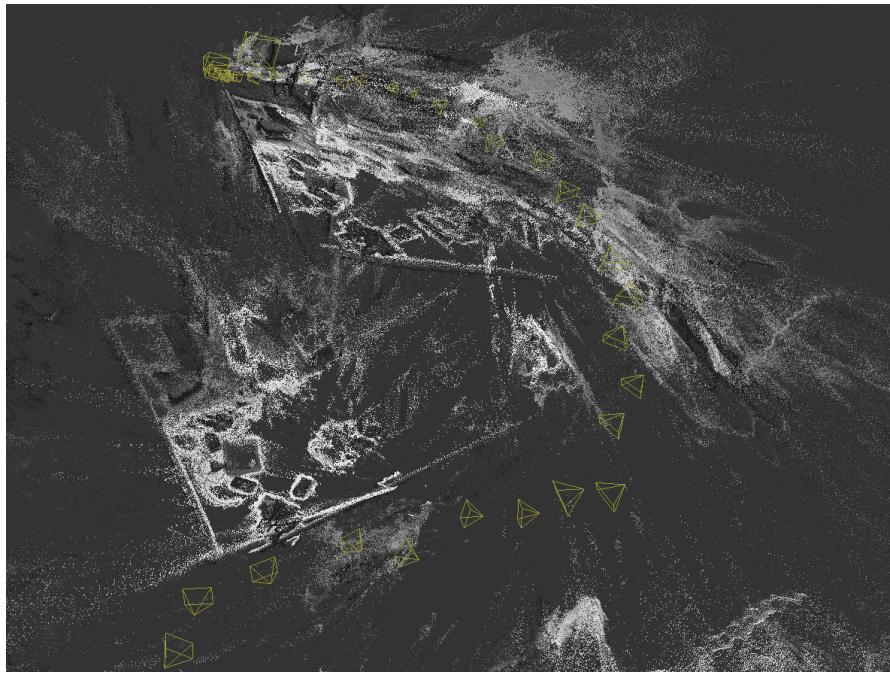
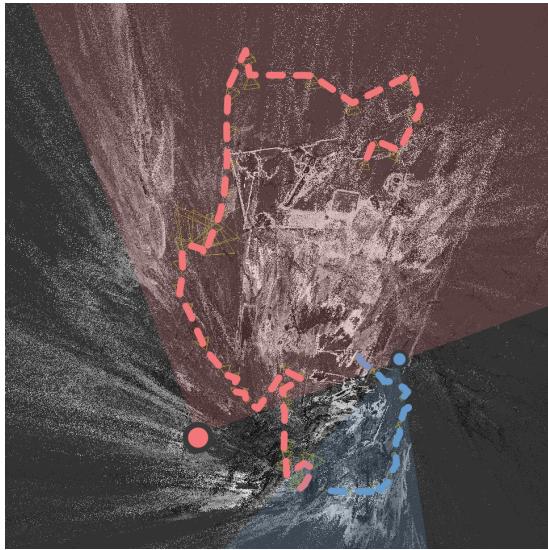


Abbildung 5.2: Eine Punkt Wolke mit starkem *drift*. Dadurch entsteht ein nicht geschlossener *loop*. Eigene Abbildung

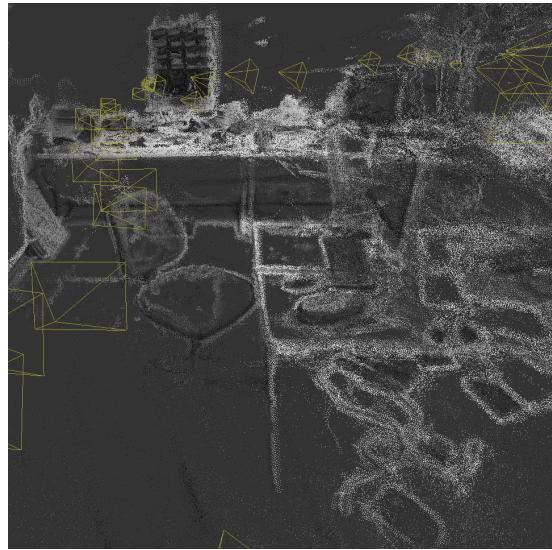
Auch Abbildung 5.3 zeigt Screenshots einer fehlerhaften Punkt Wolke. Darin hat *scale drift* dazu geführt, dass die Punkt Wolke zwei unterschiedlich skalierte Versionen der gleichen Szene enthält. In Abbildung 5.3a ist der erste Teil des Kamerapfades blau eingezeichnet. An der Stelle zwischen blauem und rotem Kamerapfad verändert sich die Skalierung zwischen zwei Keyframes abrupt. Vermutlich ist dies auf eine zu große Rotation der Kamera dort zurückzuführen. Alle Keyframes des roten Pfades sind deutlich größer skaliert als die vorherigen. Dadurch entsteht eine zusätzliche, größer skalierte Version des Tisches in der Mitte des Raums.

Punkt Wolken liefern nur unstrukturierte Informationen über die Umgebung. Zukünftig könnten im nativen Code bereits Cluster von Punkten berechnet werden. Das ermöglicht es, der App kompaktere und strukturierte Information zu übergeben. Dann könnten auch die in Abschnitt 2.2 beschriebenen Konzepte aus der Substitutional Reality zum Einsatz kommen. Cluster von Punkten werden in Unity-Skripten durch

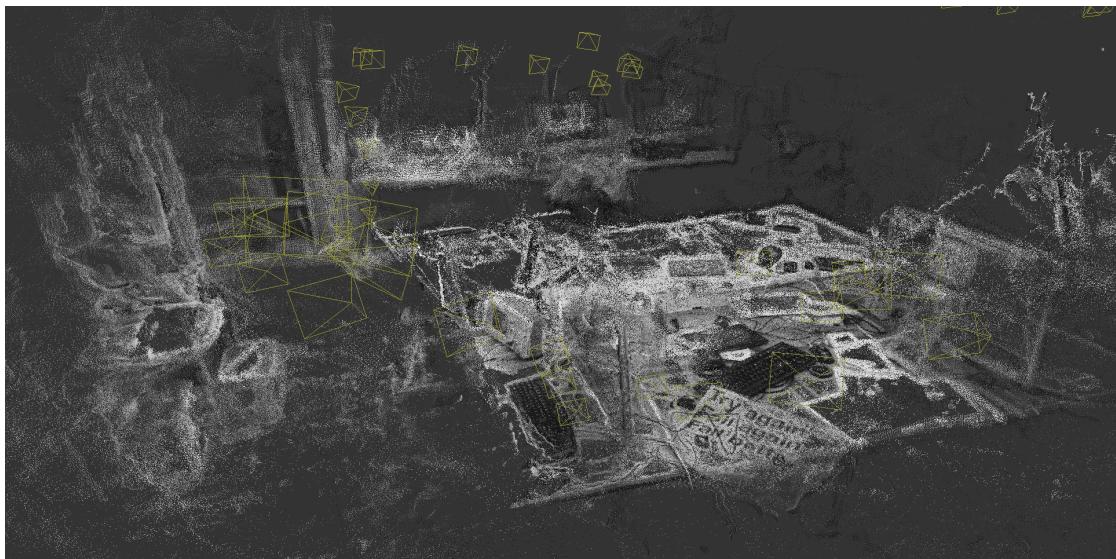
## 5 Auswertung und Ausblick



(a) Die Draufsicht auf die Punktfolge mit eingezeichnetem Kamerapfad und den Positionen, aus denen die anderen beiden Screenshots gemacht wurden.



(b) Die gerenderte Punktfolge des ersten blauen Abschnitts.



(c) Screenshot der gerenderten Punktfolge des zweiten roten Teils.

Abbildung 5.3: Eine aufgenommene Punktfolge des Mobile-Media-Labs bestehend aus 77 Keyframes und 2 220 564 Punkten. Durch *scale drift* enthält die Punktfolge zwei unterschiedlich skalierte Versionen der gleichen Szene. Eigene Abbildungen

## 5 Auswertung und Ausblick

passende geometrische Objekte ersetzt. Eine Datenbank thematisch passender 3D-Modelle könnte zuvor erstellt werden [34]. Dadurch würden Nutzer auf eine natürliche Weise davon abgehalten werden, mit Hindernissen im Raum zusammenzustoßen. Gleichzeitig stört das die Immersion nicht. Passende Objekte tragen sogar zu einer authentischen Atmosphäre bei.

Die Auswertung der Punktwolken zeigt, dass nicht geschlossene *loops* ein großes Problem sind. openFABMAP wird in der README<sup>3</sup> des LSD-SLAM als optional und „for large loop-closure detection“ bezeichnet. Alle Beispiele wurden innerhalb eines Raumes aufgenommen. Dennoch gelingt es dem LSD-SLAM nicht, die *loops* selbst zu schließen. Es sollte daher in Betracht gezogen werden, openFABMAP trotz eventueller Leistungseinbußen einzusetzen.

Es ist nicht auszuschließen, dass die genannten Probleme auf die Verwendung eines Smartphones zurückzuführen sind. Denn dessen Prozessorleistung ist deutlich geringer als die eines Desktoprechners. Der LSD-SLAM reduziert die Anzahl der *stereo comparisons*, um eine konstante *framerate* zu erreichen [16]. Auch die Kamera des Smartphones entspricht nicht den Empfehlungen<sup>4</sup> der Autoren des LSD-SLAM. Diese legen die Benutzung eines globalen Shutters, eines Weitwinkel-Objektivs sowie eine hohe Bildfrequenz nahe.

Von diesen Empfehlungen lässt nur die Bildfrequenz Spielraum für Optimierungen. OpenCV wird für den Kamerazugriff der Android-App verwendet. Das wiederum greift auf die inzwischen nicht mehr aktuelle Camera-Klasse von Android zurück. Deren Bildfrequenz schwankt zwischen 7 und 30 FPS. Das neuere camera2-Paket verspricht „fine-grain control of image capture and post-processing at high frame rates“<sup>5</sup>. Auf dem Nexus 5 kann es Bilder in voller Auflösung mit konstanten 30 FPS aufnehmen [13]. Es empfiehlt sich daher, zukünftig dieses Paket für den Kamerazugriff zu verwenden.

---

<sup>3</sup>[https://github.com/tum-vision/lsd\\_slam/blob/master/README.md](https://github.com/tum-vision/lsd_slam/blob/master/README.md)

<sup>4</sup>[https://github.com/tum-vision/lsd\\_slam#316-general-notes-for-good-results](https://github.com/tum-vision/lsd_slam#316-general-notes-for-good-results)

<sup>5</sup><http://developer.android.com/reference/android/hardware/camera2/CameraDevice.html>

## 5 Auswertung und Ausblick

Darüber hinaus kann die Benutzerfreundlichkeit des entstandenen Codes verbessert werden. Dieser ist bisher nur über das GitHub-Repository verfügbar. Unity erlaubt eine Bündelung von C#-Skripten und weiteren Dateien in sogenannten Asset Packages<sup>6</sup>. Als solches wird beispielsweise auch das Cardboard SDK verteilt. In dieser Form aufbereitet wäre der portierte LSD-SLAM Unity-Entwicklern viel leichter zugänglich.

---

<sup>6</sup><http://docs.unity3d.com/Manual/AssetPackages.html>

# 6 Fazit

Immersion ist der Schlüsselbegriff der VR. Das langweilige Wohnzimmer wird zu einer fantastischen, virtuellen Welt, in die der Nutzer eintaucht. Häufig folgt dann aber die erste Ernüchterung, wenn man losziehen und die virtuelle Welt erkunden möchte. Dafür wird ein Controller benötigt und erinnert so daran, dass alles nur eine Illusion ist. Aber auch wenn man getrackt wird, stößt man früher oder später an die Grenzen des Systems und sich möglicherweise den Kopf.

SLAM-Algorithmen sind theoretisch eine Lösung für beide Probleme. In der Praxis sind sie aber leider noch nicht so weit. Doch obwohl die Ergebnisse noch nicht ganz überzeugen können, wird das Potenzial der Kombination von SLAM und VR deutlich.

Das Tracking des Nutzers ist die Schlüsseltechnologie der VR. Es lässt die virtuelle Welt direkt auf Bewegungen des Nutzers reagieren. Das erlaubt eine natürliche Interaktion ohne den Umweg über Eingabegeräte. Eine hohe Genauigkeit beim Tracking verbessert nicht nur die Immersion, sondern hilft auch gegen Cybersickness [20].

[...] VR-grade inside-out tracking is not currently workable on mobile devices.

— Palmer Luckey  <sup>1</sup>

Beim Tracking von SLAM-Algorithmen führt jedoch *drift* zu Abweichungen. Der Fehler wächst mit der Zeit und ist deshalb schon bald nicht mehr mit den Ansprüchen aus der VR vereinbar. Ein weiterer Nachteil ist, dass für die Tiefenberechnung immer eine ausreichende Translation der Kamera benötigt wird. Gerade in VR-Anwendungen schaut sich der Nutzer aber viel um. Diese reinen Rotationsbewegungen führen dazu, dass der SLAM das Tracking verliert.

---

<sup>1</sup>[https://www.reddit.com/r/oculus/comments/3lyt0k/oculus\\_connect\\_prediction/cval9n8](https://www.reddit.com/r/oculus/comments/3lyt0k/oculus_connect_prediction/cval9n8)

## 6 Fazit

Der LSD-SLAM enthält die Umgebungsinformationen als semi-dichte Tiefenmaps. Jeder Tiefenwert ist ein Punkt im Raum; alle Punkte zusammen ergeben eine Punktwolke. In Echtzeit kann davon auf Smartphones nur ein Bruchteil angezeigt werden. Speichert man die Ergebnisse aber ab und betrachtet sie am Desktop, wird deutlich, wie dicht und detailliert die Punktwolken sind. Die Tiefenvarianz erlaubt dabei eine feine Einstellung des Trade-offs zwischen Detailgrad und fehlerhaften Werten. Allerdings führen die Fehler im Tracking häufig dazu, dass die Punktwolken die selben Strukturen mehrfach enthalten.

Weiterentwickelte SLAM-Algorithmen und die steigende Leistung von Smartphones werden die genannten Probleme langfristig lösen. Bis es so weit ist, kann nur die Komplexität des Problems reduziert werden. Dazu benötigt der SLAM zusätzliche Informationen. Die können beispielsweise weitere Sensoren liefern. Eine IMU ist in jedem Smartphone integriert und stellt Rotations- und Bewegungsdaten zur Verfügung. Inzwischen gibt es auch Mobilgeräte mit integrierten Tiefensensoren. Mit diesen Daten können das SLAM-Problem einfacher gelöst und Ergebnisse stabilisiert werden.

Eine weitere Möglichkeit ist, wie von Mur-Artal und Tardós vorgeschlagen, die Karte bereits vorab zu erstellen [31]. Dazu müsste der Nutzer den gewünschten Bereich zuerst mit der Kamera aufnehmen. Diese Daten kann der SLAM-Algorithmus dann verarbeiten. Dabei ist er nicht an die Performance-Anforderungen einer Echtzeit-Anwendung gebunden. Die daraus resultierende Karte kann immer wieder und auch auf unterschiedlichen Geräten benutzt werden. Im Echtzeitbetrieb muss der SLAM-Algorithmus dann nur noch tracken und nicht mehr gleichzeitig die Karte erstellen.

Project Tango<sup>2</sup> setzt sich ebenfalls mit SLAM und diesen Problemen auseinander. Dabei handelt es sich um eine Technologie zur Umgebungserkennung von Google. Kompatible Mobilgeräte verfügen neben den üblichen Sensoren noch über einen Tiefensensor und eine besonders weitwinklige Kamera. Die zugehörige Software kann mit *visual-inertial odometry*, also einer Kombination von visuellen und IMU-Daten, das Gerät zuverlässig tracken. Das Mapping nutzt Featurepunkte, um bereits besuchte Orte wiederzuerkennen und *drift* zu korrigieren. Ein Artikel über Tangos VR-Potenzial vermutet in dem Projekt „Google’s secret VR weapon“ [36].

---

<sup>2</sup><https://www.google.com/atap/project-tango/>

## 6 Fazit

Tango zeigt, dass auch große Tech-Unternehmen wie Google an mobilem SLAM interessiert sind. Das macht Hoffnung, dass es bald schon noch größere Fortschritte geben wird. Die Ergebnisse dieser Arbeit geben einen Einblick in das Potenzial der Kombination von SLAM und VR. Als günstiges mobiles Positionstracking löst diese ein fundamentales Problem der VR. Bisher wird SLAM den hohen Erwartungen noch nicht gerecht. Aber vieles deutet darauf hin, dass sich das schon bald ändert.

# Literatur

- [1] Tomasz Mazuryk und Michael Gervautz. Virtual Reality – History, Applications, Technology and Future. Techn. Ber. Institut für Computergraphik und Algorithmen, Technische Universität Wien, 1996 (siehe S. 5).
- [2] Rudolph P. Darken, William R. Cockayne und David Carmein. „The Omni-Directional Treadmill: A Locomotion Device for Virtual Worlds“. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST 1997, Banff, Alberta, Canada, October 14-17, 1997. Hrsg. von George G. Robertson und Chris Schmandt. ACM, 1997, S. 213–221 (siehe S. 15).
- [3] Sharif Razzaque, Zachariah Kohn und Mary C. Whitton. „Redirected Walking“. In: Eurographics 2001 - Short Presentations. Eurographics Association, 2001 (siehe S. 14, 15).
- [4] Richard Hartley und Andrew Zisserman. Multiple View Geometry in Computer Vision. 2. Aufl. Cambridge University Press, 2003 (siehe S. 20, 21, 25, 42).
- [5] Hugh F. Durrant-Whyte und Tim Bailey. „Simultaneous localization and mapping: part I“. In: IEEE Robotics & Automation Magazine 13.2 (2006), S. 99–110 (siehe S. 10).
- [6] Hugh F. Durrant-Whyte und Tim Bailey. „Simultaneous localization and mapping: part II“. In: IEEE Robotics & Automation Magazine 13.3 (2006), S. 108–117 (siehe S. 21).
- [7] Craig D. Murray u. a. „Immersive Virtual Reality as a Rehabilitative Technology for Phantom Limb Experience: A Protocol“. In: Cyberpsychology, Behavior, and Social Networking 9.2 (2006), S. 167–170 (siehe S. 5).

## Literatur

- [8] Georg Klein und David W. Murray. „Parallel Tracking and Mapping for Small AR Workspaces“. In: Sixth IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR 2007, 13-16 November 2007, Nara, Japan. IEEE Computer Society, 2007, S. 225–234 (siehe S. 11, 48).
- [9] Chia-Kai Liang, Li-Wen Chang und Homer H. Chen. „Analysis and Compensation of Rolling Shutter Effect“. In: IEEE Trans. Image Processing 17.8 (2008), S. 1323–1330 (siehe S. 34).
- [10] Georg Klein und David W. Murray. „Parallel Tracking and Mapping on a camera phone“. In: Science & Technology Proceedings, 8th IEEE International Symposium on Mixed and Augmented Reality 2009, ISMAR 2009, Orlando, Florida, USA, October 19-22, 2009. Hrsg. von Gudrun Klinker, Hideo Saito und Tobias Höllerer. IEEE Computer Society, 2009, S. 83–86 (siehe S. 11).
- [11] David Sachs. Sensor Fusion on Android Devices: A Revolution in Motion Processing. Google TechTalks. 2010. URL: <https://youtu.be/C7JQ7Rpwn2k?t=23m22s> (besucht am 31.03.2016) (siehe S. 40).
- [12] Frank Steinicke u. a. „Estimation of Detection Thresholds for Redirected Walking Techniques“. In: IEEE Transactions on Visualization and Computer Graphics 16.1 (2010), S. 17–27 (siehe S. 16).
- [13] Eddy Talvala. Google I/O 2014 - Building great multi-media experiences on Android. Google. 2010. URL: <https://youtu.be/92fgcUNCHic?t=29m53s> (besucht am 01.04.2016) (siehe S. 54).
- [14] Rainer Kümmel u. a. „ $g^2o$ : A general framework for graph optimization“. In: IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011. IEEE, 2011, S. 3607–3613 (siehe S. 21, 24).
- [15] Christian Pirchheim und Gerhard Reitmayr. „Homography-based planar mapping and tracking for mobile phones“. In: 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011, Basel, Switzerland, October 26-29, 2011. IEEE Computer Society, 2011, S. 27–36 (siehe S. 11).
- [16] Jakob Engel, Jürgen Sturm und Daniel Cremers. „Semi-dense Visual Odometry for a Monocular Camera“. In: IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013. IEEE, 2013, S. 1449–1456 (siehe S. 10, 12, 17–19, 21, 28, 54).

## Literatur

- [17] Michael Hardegger u. a. „ActionSLAM on a smartphone: At-home tracking with a fully wearable system“. In: International Conference on Indoor Positioning and Indoor Navigation, IPIN 2013, Montbeliard, France, October 28-31, 2013. IEEE, 2013, S. 1–8 (siehe S. 11, 49).
- [18] Hartmut Gieselmann und Jan-Keno Janssen. Endlich mittendrin! Erster Test der VR-Brille Oculus Rift. 2013. URL: <http://heise.de/-2323126> (besucht am 21.02.2016) (siehe S. 5).
- [19] Christian Kerl, Jürgen Sturm und Daniel Cremers. „Robust odometry estimation for RGB-D cameras“. In: 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013. IEEE, 2013, S. 3748–3754 (siehe S. 19).
- [20] Simon Davis, Keith Nesbitt und Eugene Nalivaiko. „A Systematic Review of Cybersickness“. In: Proceedings of the 2014 Conference on Interactive Entertainment, IE 2014, Newcastle, NSW, Australia, December 2-3, 2014. Hrsg. von Keith V. Nesbitt, Shamus P. Smith und Karen L. Blackmore. ACM, 2014, 8:1–8:9 (siehe S. 6, 7, 56).
- [21] Ralf Dörner u. a. „Virtual Reality und Augmented Reality (VR/AR)“. In: Informatik Spektrum 39.1 (2014), S. 30–37 (siehe S. 5).
- [22] Jakob Engel, Thomas Schöps und Daniel Cremers. „LSD-SLAM: Large-Scale Direct Monocular SLAM“. In: Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II. Hrsg. von David J. Fleet u. a. Bd. 8690. Lecture Notes in Computer Science. Springer, 2014, S. 834–849 (siehe S. 10, 12, 14, 17–21, 24, 25).
- [23] Daniel Herrera u. a. „DT-SLAM: Deferred Triangulation for Robust SLAM“. In: 2nd International Conference on 3D Vision, 3DV 2014, Tokyo, Japan, December 8-11, 2014, Volume 1. IEEE Computer Society, 2014, S. 609–616 (siehe S. 12).
- [24] Eike Kühl. Virtuelle Realität zum Mitnehmen. 2014. URL: <http://www.zeit.de/digital/mobil/2014-09/samsung-gear-vr-brille> (besucht am 02.03.2016) (siehe S. 6).
- [25] Peter Rubin. The Inside Story of Oculus Rift and How Virtual Reality Became Reality. 2014. URL: <http://www.wired.com/2014/05/oculus-rift-4/> (besucht am 12.02.2016) (siehe S. 5).

## Literatur

- [26] Thomas Schöps, Jakob Engel und Daniel Cremers. „Semi-dense visual odometry for AR on a smartphone“. In: IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2014, Munich, Germany, September 10-12, 2014. IEEE Computer Society, 2014, S. 145–150 (siehe S. 12, 28, 33–35).
- [27] Daniel AJ Sokolov. Google Cardboard: Die Virtual-Reality-Brille aus Pappe. 2014. URL: <http://heise.de/-2238744> (besucht am 02.03.2016) (siehe S. 6).
- [28] Laura Kate Dale. Unity - does indie gaming's biggest engine have an image problem? 2015. URL: <http://www.theguardian.com/p/4aayb/sbl> (besucht am 08.03.2016) (siehe S. 38).
- [29] Xavier Hallade. The new NDK support in Android Studio. 2015. URL: <http://ph0b.com/new-android-studio-ndk-support/> (besucht am 17.03.2016) (siehe S. 31).
- [30] Raúl Mur-Artal, J. M. M. Montiel und Juan D. Tardós. „ORB-SLAM: A Versatile and Accurate Monocular SLAM System“. In: IEEE Transactions on Robotics 31.5 (2015), S. 1147–1163 (siehe S. 13).
- [31] Raúl Mur-Artal und Juan D. Tardós. „Monocular SLAM for User Viewpoint Tracking in Virtual Reality“. In: Workshop on Challenges in Virtual Reality, ICRA 2015. 2015 (siehe S. 13, 57).
- [32] Victor Adrian Prisacariu u. a. „Real-Time 3D Tracking and Reconstruction on Mobile Phones“. In: IEEE Trans. Vis. Comput. Graph. 21.5 (2015), S. 557–570 (siehe S. 11, 49).
- [33] Sylvain Ratabouil. Android NDK: Beginner’s Guide. 2. Aufl. Packt Publishing Ltd, 2015 (siehe S. 30).
- [34] Adalberto Lafcadio Simeone, Eduardo Velloso und Hans Gellersen. „Substitutional Reality: Using the Physical Environment to Design Virtual Reality Experiences“. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18–23, 2015. Hrsg. von Bo Begole u. a. ACM, 2015, S. 3307–3316 (siehe S. 16, 54).
- [35] YouTube Creator Blog. 360-degree videos now on Google Cardboard and iOS. 2015. URL: <http://youtubecreator.blogspot.de/2015/05/360-degree-videos-now-on-google.html> (besucht am 21.02.2016) (siehe S. 5).

## Literatur

- [36] Dieter Bohn. Google's secret VR weapon is Project Tango. 2016. URL: <http://www.theverge.com/2016/2/25/11112544/google-vr-project-tango-io-2016> (besucht am 12.03.2016) (siehe S. 57).
- [37] Elizabeth Reede und Larissa Bailiff. When Virtual Reality Meets Education. 2016. URL: <http://techcrunch.com/2016/01/23/when-virtual-reality-meets-education/> (besucht am 13.03.2016) (siehe S. 5).
- [38] Will Shanklin. Virtuix Omni: VR treadmills not yet living up to the promise (hands-on). 2016. URL: <http://www.gizmag.com/virtuix-omni-vr-treadmill-review-hands-on/41438/> (besucht am 04.03.2016) (siehe S. 15).