Hochschule für Technik und Wirtschaft Berlin

Internationaler Studiengang Medieninformatik

Masterarbeit

von

Daniel **Schneider**

Photogrammetrie zur Platzierung von standortbezogenen dynamischen Inhalten in AR

Photogrammetry for placement of location-based dynamic content in AR

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Informatik, Kommunikation und Wirtschaft

Studiengang Internationaler Studiengang
Medieninformatik

Masterarbeit

von

Daniel **Schneider**

Photogrammetrie zur Platzierung von standortbezogenen dynamischen Inhalten in AR

Photogrammetry for placement of location-based dynamic content in AR

Bearbeitungszeitraum: von 13.05.2019
bis 16.09.2019

1. Prüfer: Prof. Dr. Tobias Lenz

2. Prüfer: Prof. Dr. Klaus Jung

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Informatik, Kommunikation und Wirtschaft

Eigenständigkeitserklärung

Name und Vorname
der Studentin/des Studenten: **Schneider, Daniel**

Studiengang: **Internationaler Studiengang Medieninformatik**

Ich bestätige, dass ich die Masterarbeit mit dem Titel:

Photogrammetrie zur Platzierung von standortbezogenen dynamischen Inhalten in AR

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 3. September 2019

Unterschrift:

Hochschule für Technik und Wirtschaft Berlin
Fachbereich Informatik, Kommunikation und Wirtschaft

Masterarbeit Zusammenfassung

Studentin/Student (Name, Vorname):	**Schneider, Daniel**
Studiengang:	Internationaler Studiengang Medieninformatik
Aufgabensteller, Professor:	Prof. Dr. Tobias Lenz
Durchgeführt in (Firma/Behörde/Hochschule):	HTW Berlin
Betreuer in Firma/Behörde:	
Ausgabedatum: 13.05.2019	Abgabedatum: 16.09.2019

Titel:

Photogrammetrie zur Platzierung von standortbezogenen dynamischen Inhalten in AR

Zusammenfassung:

Diese Arbeit analysiert die wichtigsten Verfahren aus der Photogrammetrie und der Computer Vision, wie den Bündelblockausgleich, SLAM (Simultaneous Location and Mapping) und SfM (Structure from Motion) im Kontext von Augmented Reality (AR) für Android Smartphones. Die Algorithmen der beiden Disziplinen werden in Zusammenhang gestellt und eine inhaltliche Unterscheidung, trotz der Überschneidung der Algorithmen, formuliert. Die Tauglichkeit der photogrammetrischen Verfahren wird in Bezug zu Augmented Reality geprüft. Im Praxisteil dieser Arbeit wird eine Anwendung erstellt, die ein aktuelles AR Framework verwendet, welches SLAM implementiert.

Schlüsselwörter: Photogrammetrie, Computer Vision (CV), Augmented Reality (AR), standortbezogene Daten, Android, Java, Structure from Motion (SfM), Simultaneous Location and Mapping (SLAM)

Inhaltsverzeichnis

1 Motivation

Photogrammetrie ist „die Wissenschaft und Technologie der Gewinnung von Informationen über die physische Umwelt aus Bildern, mit einem Schwerpunkt auf Vermessung, Kartierung und hochgenauer Messtechnik“. (Heipke, 2017, S.5 [1]) Die Photogrammetrie beschäftigt sich mit der Rekonstruktion von dreidimensionalen Daten aus zweidimensionalen Informationsträgern, wie Bildern oder Laserscan Daten. Dabei gehen diese Daten alle auf das Prinzip der Aufnahme der elektromagnetischen Strahlung zurück. Bei Bildern ist das die Helligkeits und Farbverteilung, bei Laserscans sind es Entfernungsbilder, beziehungsweise Punktwolken. Die Disziplin der Photogrammetrie ist dabei dem Bereich der Fernerkundung zuzuordnen, die sich mit der Auswertung von geometrischen oder semantischen Informationen beschäftigt. Beides sind Fachbereiche, die sich über die Jahrzehnte entwickelt haben und sich dem Gebiet der Geodäsie zuordnen lassen. Die Geodäsie erfasst Geoinformationen über die Erde, die dann beispielsweise mit Kartographie visualisiert werden können. Das große Potenzial der Bereitstellung und Gewinnung von dreidimensionalen Daten war in der Photogrammetrie seit Beginn eine der wichtigsten Stärken der Technologie. Photogrammetrie wird von Raumsonden im All bis hin zur Mikroskopie eingesetzt. Auch wenn die ursprüngliche Kernanwendung die Kartenerstellung war, sind nichttopographische Anwendungen seit der Einführung digitaler Kameras stark expandiert. Dazu zählt zum Beispiel die hochpräzise industrielle Messtechnik, die architektonische Photogrammetrie, die medizinische und forensische Bildgebung und Analyse, sowie die Erstellung von dreidimensionalen Modellen aus Fotoserien oder Laserdaten, welche der Nahbereichsfotogrammetrie zugeordnet werden kann. (vgl. [26] S.1)

Das Gebiet der Computer Vision (CV), das sich in den letzten Jahren teilweise parallel mit der Photogrammetrie entwickelt hat, verfolgt einen ähnlichen Ansatz und ist deshalb inhaltlich sehr stark mit der digitalen Photogrammetrie verwandt. In diesem Bereich haben sich ebenfalls verschiedene Verfahren zur Kartierung der Umwelt und der gleichzeitigen Lokalisierung in dieser entwickelt, welche ebenfalls in dieser Arbeit beschrieben werden, um einen Vergleich von Photogrammetrie und den Verfahren der Computer Vision zu ermöglichen.

Verschiedene Algorithmen die der CV zugeordnet werden können, sind in modernen Augmented Reality Frameworks in Verwendung. Das Forschungsziel der Computer Vision ist es, anhand von zweidimensionalen Bilder oder Videos die geometrischen Informationen von dreidimensionalen Objekten, wie Form, Position, räumlicher Lage, Bewegung oder Daten über den Inhalt der Bilder zu gewinnen, sowie dem Computer ein umfassenes Verständnis der Inhalte der Bilder zu ermöglichen. Unter diesem Gesichtspunkt gibt es sehr viele Gemeinsamkeiten zwischen Computer Vision und digitaler Photogrammetrie. Nachdem sich Photogrammetrie und Computer Vision lange unabhängig voneinander entwickelt haben, ist Photogrammetrie heute als Grundlage von Computer Vision anerkannt (vgl. [1] S. 5-7).

Durch die Entwicklung der Technik der letzten Jahrzehnte und der damit eingehenden Steigerung der Rechenpower im mobilen Bereich, haben sich die Bereiche vergrößert, in denen digitale Photogrammetrie eingesetzt werden kann. Im Rahmen dieser Arbeit soll evaluiert werden, ob photogrammetrische Verfahren für Augmented Reality Anwendungen im Bereich von Smarthphones eingesetzt werden können, um in Echtzeit aus Videodaten die dreidimensionale Beschaffenheit der gefilmten Objekte zu rekonstruieren. Dazu werden, nach einer kurzen Definition und Einführung in Augmented Reality, die Verfahren und Algorithmen aus der Photogrammetrie und der Computer Vision analysiert und anschließend verglichen. Es werden weiterhin Vorteile sowie Nachteile der jeweiligen Disziplinen aufgezeigt und anschließend im Kontext zu Augmented Reality beschrieben.

Im Rahmen dieser Arbeit ist weiterhin eine auf dem Android Betriebsystem basierende Anwendung erstellt worden, welche eine der neuen Technologien im Bereich Augmented Reality implementiert und aktuelle Frameworks und Bibliotheken verwendet. Die Besonderheiten dieser Frameworks werden im Praxistest analysiert und im Bezug zu Photogrammetrie und Computer Vision untersucht.

2 Augmented Reality

Im Gegensatz zu „Virtual Reality“ (VR), welche eine interaktive, dreidimensionale, computergenerierte, immersive Umgebung schafft, in die eine Person versetzt wird, erlaubt „Augmented Reality“ (AR) die Überblendung von digitalen Medieninformationen über die Wahrnehmung der echten Welt. Dadurch fällt AR in die Definition von „Mixed Reality“ (MR).

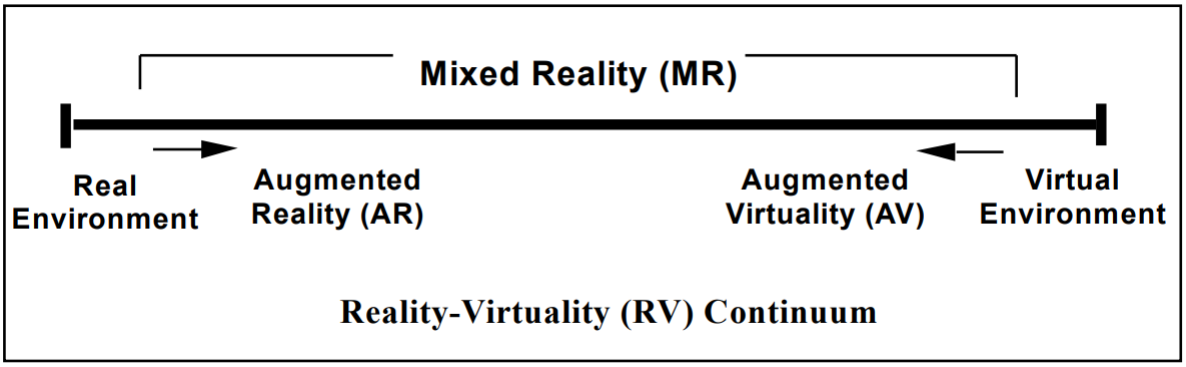


Abbildung 2.1: Das Realität - Virtualität Kontinuum, Bildquelle [11]

Im Rahmen dieser Arbeit wird der Begriff Augmented Reality auf Monitor basierte, nicht immersive Geräte bezogen, da bei der Analyse von Photogrammetrie und Computer Vision, sowie der Implementation einer AR Anwendung, das Smartphone als Medium im Fokus steht. Diese Anzeigesysteme werden auch als „Window-on-the-World“ bezeichnet, da computergenerierte Bilder oder Informationen digital in Echtzeit über das wiedergegebene Kamerabild des Smartphones überlagert werden (vgl. [11] S.284).

2.1 Augmented Reality - SDKs

„Software Development Kits“ (SDK) sind Werkzeuge und Bibliotheken, welche Algorithmen und Basistechnologien liefern, um Programme zu entwickeln.

Im Bereich von Augmented Reality umfassen Frameworks meistens die drei folgenden Hauptkomponenten (vgl. [12] S.3):

- **Recognition**: Erkennung von Bildern, Objekten, Gesichtern oder Räumen, auf welchen die virtuellen Objekte oder Informationen überlagert werden können.
- **Tracking**: Echtzeit-Lokalisierung der erkannten Objekte und Berechnung der lokalen Position des Gerätes.
- **Rendering**: Überlagerung der virtuellen Medieninformationen über das Bild und Anzeige der generieren Mixed Reality.

2.1.1 Marktübersicht - Software Development Kits

Die folgende Tabelle gibt eine Übersicht an aktuellen und weit verbreiteten Software Development Kits, sowie deren Plattformkompatibilität zu Android und iOS:

	Vuforia	Wikitude	Metaio	ARToolKit	Kudan	EasyAR	MaxST	ARCore	ARKit
Android	✓	✓	✓	✓	✓	✓	✓	✓	x
iOS	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	✓	✓	✓	✓	x	✓	✓	x	x

Bis auf das von Apple entwickelte ARKit, sind alle hier genannten Frameworks für Android verwendbar. Weiterhin unterstützen alle das Betriebsystem iOS, sowie Windows, bis auf Kudan, ARCore und ARKit. Im praktischen Teil dieser Arbeit werden mehrere dieser Software Development Kits auf der Android Plattform verwendet, getestet und analysiert.

2.2 Voraussetzungen für Augmented Reality

Augmented Reality Anwendungen haben hohe Anforderungen an die Rechnenpower der Hardware, die Verarbeitungsgeschwindigkeit der Algorithmen und Robustheit der verwendeten Verfahren. Folgende Liste gibt eine Übersicht an wichtigen Aspekten, um AR zu realisieren (vgl. [18] S.1):

- **Hohe räumliche Genauigkeit**: 6 „Degrees of Freedom“ (Freiheitsgrade) in Position und Ausrichtung.

- **Sehr geringer Jitter (Zittern)**: das Rauschen im Tracking System muss minimal gehalten werden.
- **Hohe Aktualisierungsraten**: mindestens 30Hz, besser mehrere 100Hz.
- **Sehr geringer Lag**: die Verzögerung von Messung bis zur Trackerausgabe muss minimal sein.
- **Volle Mobilität**: Bewegungsfreiheit für den Nutzer: keine Kabel, kein eingeschränkter Umfang an Bedienmöglichkeiten.

Erst durch die Entwicklung der Technik in den letzten zwei Jahrzehnten und einer damit einhergehenden Steigerung der Rechenpower von mobilen Geräten, hat sich Augmented Reality auf Smartphones etablieren können.

2.3 Arten des Augmented Reality Trackings

Das Erkennen der Umgebung und die Lokalisierung der Kamera (Camera Pose Estimation) in dieser, siehe Abbildung 2.2, ist der ausschlaggebende Schritt zur Realisierung von Augmented Reality. Erst dieses Verfahren erlaubt die Projektion von digitalen Modellen in der richtigen Position auf den echten Bildern, um die Perspektiven von echter Welt und digitalem Modell zu vereinen.

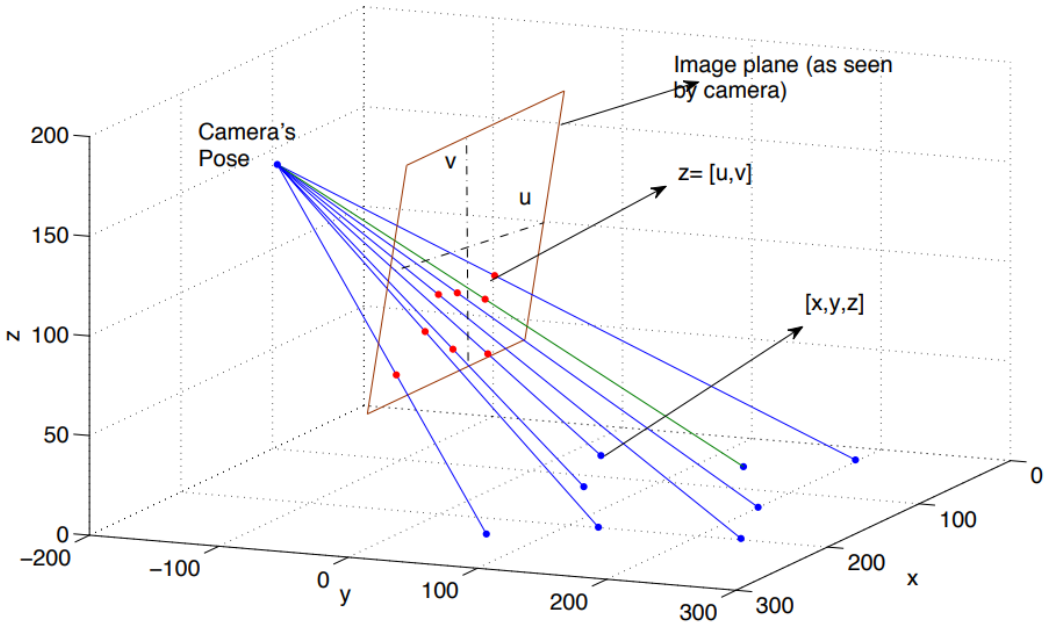


Abbildung 2.2: Kamera Posenschätzung, anhand von Markern. Bildquelle [47]

Bis Anfang der 2000er Jahre basierten fast alle visuell basierten Tracking- und Lokalisierungstechniken auf Markern. Dann sind Technologien, wie modellbasiertes Tracking und Tracking ganz ohne Marker, entwickelt worden. Schließlich sind Keyframe basierte Systeme populär geworden, wie SLAM (Simultaneous Localization and Mapping), das als Weiterentwicklung aus SfM (Structure from Motion) hervorging (vgl. [50] S.1).

Präzise und robuste Kamerapositionsdaten sind eine Grundvoraussetzung für eine Vielzahl von Anwendungen, wie dynamischer Szenenanalyse und Interpretation, 3D-Szenen Strukturerkennung oder Videodatenkompression. Augmented Reality Umgebungen sind ein komplexes Anwendungsgebiet der Kameralokalisierung, da ein eingeschränkter Arbeitsbereich, limitierte Rechenpower und spezifische Anwendungsfälle hohe Anforderungen an die Robustheit und Schnelligkeit der Algorithmen stellen. Es existieren viele verschiedene Ansätze, um die Kameralokalisierung im Raum zu lösen. Das Problem wird als nichtlineares Problem betrachtet und wird meistens durch die „Method of Least Squares“ (Methode der kleinsten Quadrate) oder mit nichtlinearen Optimierungsalgorithmen gelöst, typischerweise durch das Gauß-Newton oder Levenberg-Marquardt Verfahren (vgl. [17] S.1). Im Folgenden werden die vier gängigsten Ansätze zur Lösung des Tracking Problems erläutert.

2.3.1 Referenzmarken-basiertes Tracking

Markerbasiertes Tracking war lange Zeit eine der häufigsten verwendeten Techniken, um Augmented Realtiy zu realisieren. Dies liegt in der einfachen Erkennung der Marker mit hohem Kontrast.

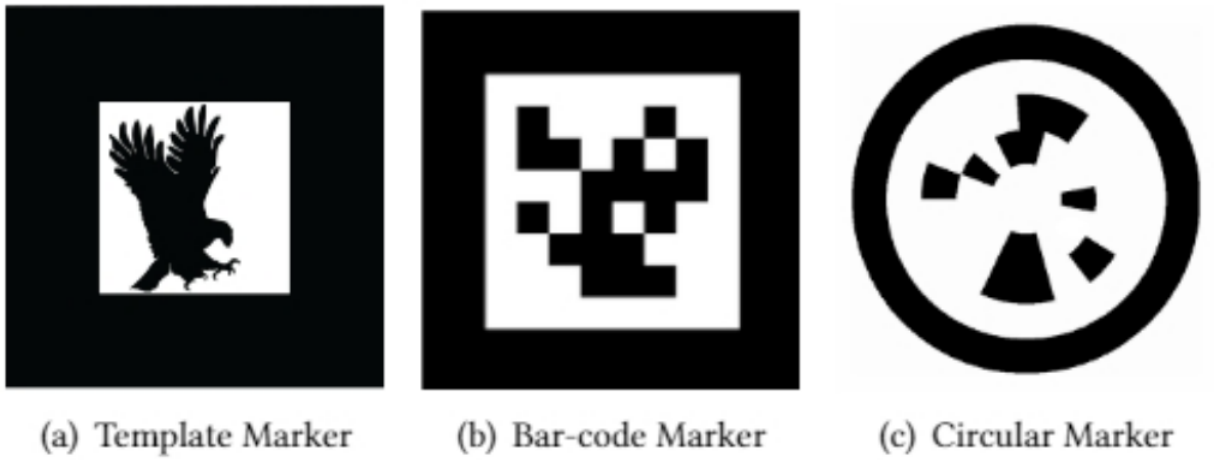


Abbildung 2.3: Moderne Marker. Bildquelle [48]

Für robuste Anwendungen werden meist schwarz-weiße Marker mit dicken Kanten verwendet, siehe Abbildung 2.3. Die Kanten dienen zur Erkennung des Markers, das einzigartige Muster zur genauen Identifizierung, um welchen Marker es sich handelt (vgl. [48] S.2). Dadurch können neben der Relation des Geräts zum Marker auch relativ einfach die Entfernung und der Winkel berechnet werden. Der Nachteil liegt in der Limitierung der Anwendungsgebiete, in denen diese Technik verwendet werden kann, da Marker immer im Sichtfeld der Kamera lokalisiert sein müssen und nicht von anderen Objekten verdeckt werden dürfen. Weiterhin muss immer auf eine externe Ressource zurückgegriffen werden um diese Marker zu erstellen, zu registrieren und zu verwenden, was für den Endnutzer immer mit einem Mehraufwand verbunden ist (vgl. [13] S.13).

2.3.2 Hybrid-basiertes Tracking

Hybrid basiertes Tracking verwendet mehrere Datenquellen, wie das Global Positioning System (GPS), den Kompass oder die Beschleunigungssensoren des Smartphones zur Bestimmung der Orientierung und Lokalisierung des Geräts. Dabei wird per GPS der Standort des Geräts bestimmt, um Objekte in der Nähe zu identifizieren, die augmentiert werden sollen. Mit Hilfe des Kompasses kann dann ein Pfad erstellt und überprüft werden, ob die Orientierung des Geräts auch in diese Richtung zeigt. Der Beschleunigungssensor bestimmt die Ausrichtung des Geräts mithilfe der Gravitation. Durch die Vereinigung all dieser Informationen aus den Sensoren kann berechnet werden, was im Sichtfeld ergänzt werden soll, ohne dass eine Auswertung und Verarbeitung des realen aufgenommen zweidimensionalen Kamerabildes stattzufinden hat. Anschließend werden die Informationen über das Kamerabild gelegt (vgl. [13] S.13, [18] S.4).

2.3.3 Modell-basiertes Tracking

Beim Modell-basiertem Tracking wird ein rekursiver Algorithmus verwendet. Hierbei wird die vorherige Kameraposition als Grundlage für die Berechnung der aktuellen Kameraposition verwendet. Dazu muss zuerst eine Kamerakalibrierung durchgeführt werden, anschließend wird die Pose des zu verfolgenden Objekts initial bestimmt. Dann wird bestimmt, welche Ecken des Objekts sichtbar sind. An diesen Ecken werden dann Kontrollpunkte verteilt, die in späteren rekursiven Aufrufen mit denen älterer Frames verglichen werden, was mit einer Karte der Ecken ermöglicht wird.

Die Map der Ecken wird mithilfe des Canny-Algorithmus zur Eckenerkennung erstellt. Ist die Abweichung zum vorherigen Frame bestimmt worden, wird die Position der Kamera im Bezug zum Objekt aktualisiert. Durch die Rekursivität ist dieses Verfahren nicht sehr rechenintensiv und benötigt eine relativ geringe Prozessorleistung. Weiterhin kann zwischen verschiedenen Merkmalen unterschieden werden, welche für das Tracking verwendet werden. Bei der kantenbasierten Methode wird versucht, ein dreidimensionales Wireframe mit den Kanten des Objekts in der realen Welt zuzuordnen (vgl. [14] S.2-3).

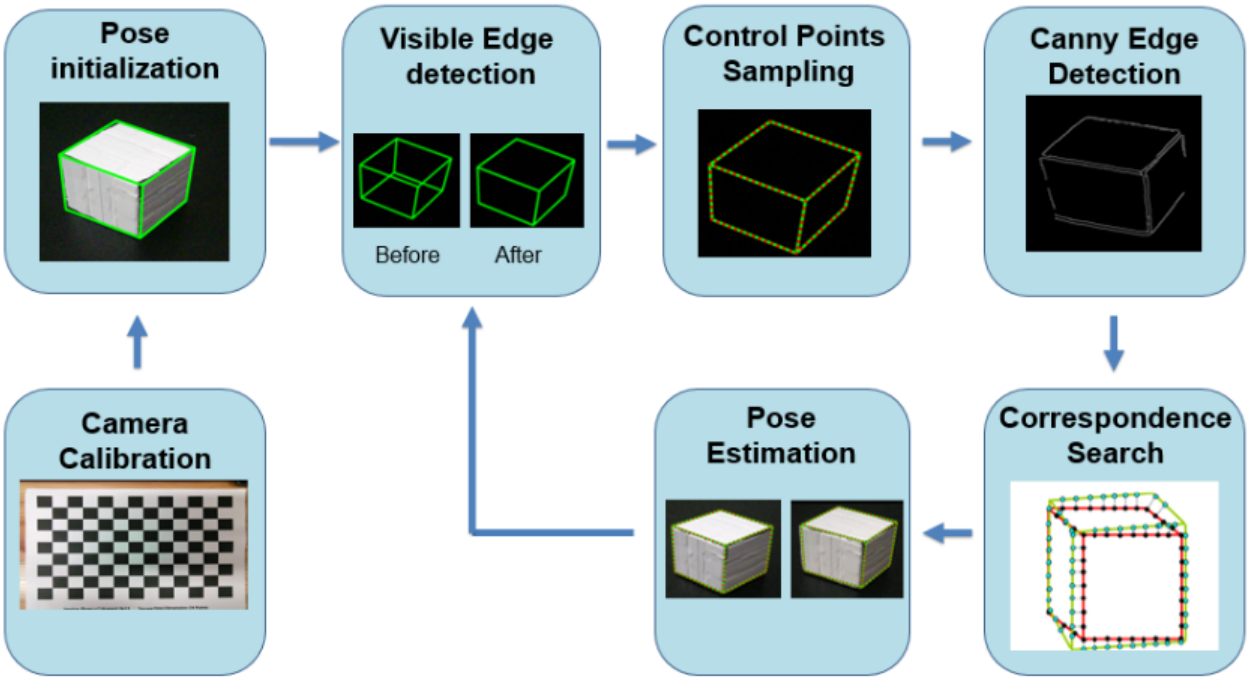


Abbildung 2.4: Kantenbasiertes rekursives Tracking, Pipeline. Bildquelle: [14] S.3

Außerdem sind Ansätze wie „Optical Flow based Tracking“, das zeitliche Informationen, entnommen aus der Bewegung der Projektion des Objekts relativ zur Bildebene verwendet, sowie texturbasierte Ansätze verbreitet (vgl. [14] S.1).

2.3.4 Natürliches Feature Tracking

Natürliches Feature Tracking ist ein bildbasiertes Verfahren und kann die Position des Gerätes zur Umgebung bestimmen, ohne über Wissen über einen initialen vorherigen Zustand zu verfügen. Diese Methode ist in der Regel sehr rechenintensiv und benötigt hohe Prozessorleistung (vgl. [14] S.1-2). Die Technik verwendet die Merkmale von Objekten in der echten Welt und erkennt ihre natürlichen Eigenschaften.

Diese Merkmale werden „Features“ genannt, sind typischerweise, basierend auf einem mathematischem Algorithmus, sehr gut unterscheidbar und äußern sich in der Form von Ecken, Kanten oder starke Kontrasten. Die mathematischen Beschreibungen dieser Features (Deskriptoren) eines Bildes werden zur späteren Erkennung gespeichert. Anhand des gespeicherten Datensets aus Merkmalen kann dann erkannt werden, ob ein Bild im Vergleich zu einem anderen den gleichen Inhalt zeigt, unabhängig von Entfernung, Orientierung, Beleuchtungsintensität, Rauschen oder Verdeckung (vgl. [13] S.13). Beim natürlichen Feature Tracking wird, meistens basierend auf dem Verfahren der kleinsten Quadrate und mit dem Gauß-Newton oder Levenberg-Marquardt Verfahren, versucht, eine Reduzierung des „Re-Projection Errors“ (Reprojektionsfehlers) zu erreichen, um alle internen und externen Kameraparameter zu bestimmen und zu optimieren. Dies sind nicht-lineare Methoden zur Lösung des Problems der kleinsten Quadrate. Typischerweise sind zwei bis vier Wiederholungen genug (vgl. [15] S.28-29). Die genaue Funktionsweise des natürlichen Feature Trackings wird in Kapitel 3.5 und 3.6 weiter ausgeführt.

2.4 Photogrammetrie oder Computer Vision für AR

In Kapitel 2 wurde eine kurze Einführung in Augmented Reality gegeben und deren Voraussetzungen sowie aktuellen Umsetzungen der Kameralokalisierung vorgestellt. Der folgende Teil dieser Arbeit beschreibt Photogrammetrie, sowie Verfahren aus der Computer Vision, wie etwa SLAM (Simultaneous Localisation and Mapping), welches von den meisten Augmented Reality Software Development Kits heutzutage verwendet wird. Ein Vergleich der beiden Verfahren zeigt Gemeinsamkeiten, Unterschiede, sowie Möglichkeiten und Schwächen auf und bringt sie in Kontext. Anschließend wird evaluiert, ob photogrammetrische Verfahren in Kontext von Augmented Reality eingesetzt werden können.

3 Photogrammetrie

3.1 Einführung in die Photogrammetrie

Das Grundprinzip der Messung mit Kameras ist auf die Messung von Licht zurückzuführen. Licht breitet sich mit einer bestimmten Wellenlänge in annähernd geraden Strahlen aus. Diese Strahlen werden vom Sensor der Kamera aufgenommen, sodass dieser die Richtungen im dreidimensionalen Raum misst, aus welchen die Lichtwellen kommen. Der grundlegende geometrische Zusammenhang der Photogrammetrie ist somit die Zentralprojektion, die sich mathematisch durch die Kollinearitätsgleichung beschreiben lässt. Ein dreidimensionaler Punkt in der echten Welt, sein Bild in der Kamera und das Projektionszentrum müssen alle auf einer geraden Linie liegen (vgl. [20] S.1). Das fundamentale photogrammetrische Problem besteht in der Bestimmung von internen und externen Ausrichtungsparametern der Kamera und der Messung von Objekt und Raumkoordinaten der aufgenommenen Fotografien.

- **Interne Orientierung**: Bei der internen Orientierung werden Kameraparameter gemessen und ausgewertet. Dazu werden die „Principle Distance“ (Brennweite) und der „Principle Point“ (Optisches Zentrum) betrachtet (vgl. Abbildung 3.1).

  Weiterhin müssen Parameter, welche die Verzeichnung, also die nicht maßstabsgetreue Abbildung von Objekten beschreiben, betrachtet werden. Diese Parameter, die beispielsweise in der Objektivkorrektur verwendet werden, müssen mit in die Berechnung einfließen, um die interne Orientierung der Kamera genau abzubilden.

- **Externe Orientierung**: Bei der externen Orientierung wird versucht, die genaue räumliche dreidimensionale Lage der Kamera zum Zeitpunkt der Belichtung des Bildes zu rekonstruieren. Für die Bestimmung der Orientierung von ein oder mehreren Fotos, gibt es verschiedene Methoden. Dies kann in Teilschritten (relative und absolute Orientierung) oder gleichzeitig (Bündelblockausgleich) durchgeführt werden.

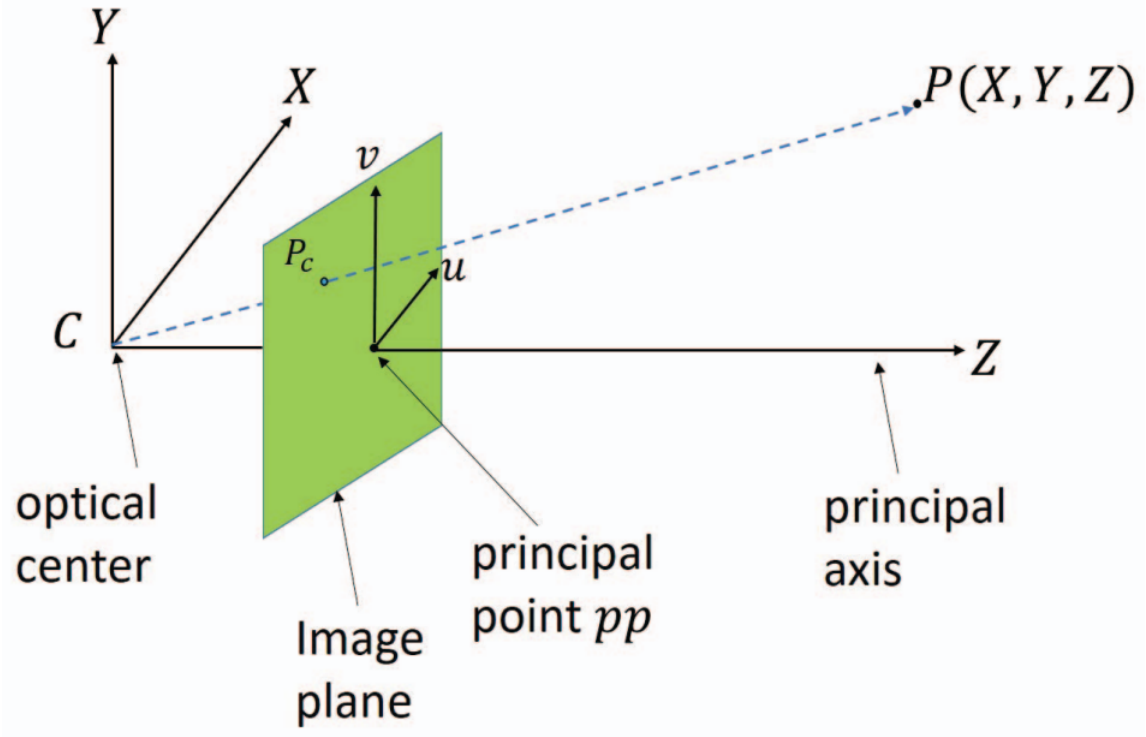


Abbildung 3.1: Kamera Kalibierungsmodell, Bildquelle [21]

In der Photogrammetrie werden häufig drei grundlegende Bedingungen verwendet, um die Parameter der externen Orientierung zu bestimmen. Dies sind die Bedingungen der Kollinearität, der Koplanarität und der Koangularität. Diese Verfahren verwenden alle Punktkoordinaten als Eingabedaten, aber in vielen Fällen sind auch räumliche oder kamerabedingte Einschränkungen möglich. In Bereich der Computer Vision wird die Bestimmung der externen Orientierung als „Pose Estimation Problem“ (Posenschätzungsproblem) bezeichnet. Im Vergleich zur Photogrammetrie ist dieser Bereich darauf fokussiert, die Lösung der Posenschätzung unter der Verwendung von minimalen Objektinformationen zu erreichen. Direkte lineare Lösungen basieren hier hauptsächlich auf Konzepten der projektiven algebraischen Geometrie. Homogene Koordinaten werden verwendet, um Kameraparameter abzuleiten, entsprechend der direkten linearen Transformation, häufig eingesetzt in der Photogrammetrie und in der Fernerkundung (vgl. [22] S.616).

Der grundsätzliche Ablauf, um anhand von zweidimensionalen Bildern die internen und externen Kameraparameter, sowie die Lage der Bilder zueinander im dreidimensionalen Raum zu rekonstruieren, kann wie in Abbildung 2.3 dargestellt, beschrieben werden.

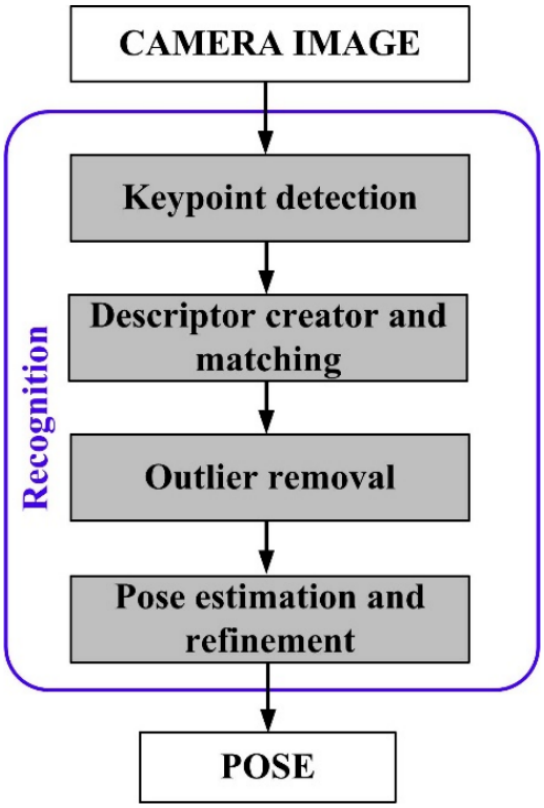


Abbildung 3.2: Tracking Pipeline Bildquelle [15]

Die Schritte „Keypoint Detection“, „Descriptor Creator and Matching“ sowie „Outlier Removal“ werden im folgenden Kapitel 3.2, die „Pose Estimation and Refinement“ in Kapitel 3.3 beschrieben.

3.2 Feature & Image Matching

Um verschiedene Bilder miteinander in Beziehung setzen zu können, müssen die Features einer dreidimensionalen Szene in den verschiedenen Bildern abgeglichen werden. Seit einigen Jahren sind „Image Feature Detectors and Descriptors“ die am weitesten verbreiteten Techniken für diese Anwendungen. Zum Anwendungsbereich dieser Algorithmen zählen beispielsweise 3D-Rekonstruktion, Panoramaerstellung, Bildklassifizierung, Objekterkennung oder Roboterlokalisierung. Heute gibt es eine Vielzahl an verschiedenen Algorithmen, welche die Erkennung und Beschreibung von Features implementieren, um „Regions of Interest“, Ecken oder Kanten zu erhalten (vgl. [35] S.3).

Dazu sollte zwischen „Feature Detektors“ und „Feature Descriptors“ unterschieden werden. Detektoren sind Operationen, welche zweidimensionale Positionen im Bild suchen (einen Bildpunkt oder eine Region), die unter verschiedenen Transformationen stabil sind und einen hohen Informationsgehalt enthalten. Die Ergebnisse werden als „Interest Points“, affine Regionen, invariante Regionen, oder Ecken bezeichnet. Deskriptoren hingegen analysieren das Bild und liefern für einen bestimmten „Point of Interest“ im Bild einen 2D-Vektor mit Pixelinformationen. Diese Informationen sind dann Features, um diesen bestimmten Punkt im Bild zu klassifizieren und ihn mit anderen Features aus anderen Bildern zu vergleichen (vgl. [36] S.1).

Die Eigenschaften eines guten Feature Matching Systems sind:

- **Invarianz**: unabhängig von geometrischen oder radiometrischen Verzerrungen (Drehung, Skalierung, Rotation)
- **Stabilität**: robust gegen Bildrauschen
- **Unterscheidbarkeit**: klare Unterscheidbarkeit zum Hintergrund
- **Einzigartigkeit**: Jedes Feature ist (im Vergleich zu den anderen) einzigartig

Die Merkmalserkennung und Merkmalsanpassung wird in drei Schritte unterteilt [35]:

(1) **Detektion**: Finden der Keypoints in jedem Bild.

(2) **Beschreibung**: Im Idealfall sollte die lokale Umgebung des Features immer invariant gegenüber Skalierung, Drehung, Rauschen, Änderung der Beleuchtung oder affinen Transformationen sein. Die Merkmalsbeschreibungen für jeden Keypoint werden anhand seiner Nachbarschaft bestimmt. Zu jedem Keypoint wird ein „Descriptor Vector“ (Beschreibungsvektor) berechnet.

(3) **Matching**: Um ähnliche Merkmale zwischen verschiedenen Bildern zu identifizieren, werden die Deskriptoren verglichen. Ist ein Feature in zwei Bildern enthalten, zeigen die Bilder das gleiche Objekt.

Die häufigsten verwendeten Verfahren sind „Scale Invariant Feature Transform“ (SIFT) (Lowe, 2004), „Features from Accelerated Segment Test“ (FAST) (Rosten and Drummond, 2005), „Speeded Up Robust Features“ (SURF) (Bay et al., 2006), „Binary Robust Independent Elementary Features“ (BRIEF) (Calonder et al. 2010) oder „Oriented FAST and Rotated BRIEF“ (ORB) (Rublee et al. 2011). Diese unterscheiden sich hauptsächlich durch die erkannten Features im Bild, die getrackt werden sollen, sowie dem erzeugtem Modell der Umgebung (vgl. [35] S.3, [15] S.28-29). Im folgenden Abschnitt werden SIFT, FAST und BRIEF erläutert.

3.2.1 Scale Invariant Feature Transform

SIFT ist 1999 von D. Lowe vorgestellt worden (vgl. [49]). Bei SIFT schätzt der Algorithmus die dominante Orientierung des Keypoints mit Hilfe von Gradienten und beschreibt anschließend die Keypoints in Bezug zu den Gradienten der Umgebung. Die Features werden in einem „Difference of Gaussian“ (DoG) Skalenraum erfasst, der die Differenz der „Gaussian convolutions“ (Gaußschen Faltung) des Originalbilds repräsentiert (Abbildung 3.3-a).

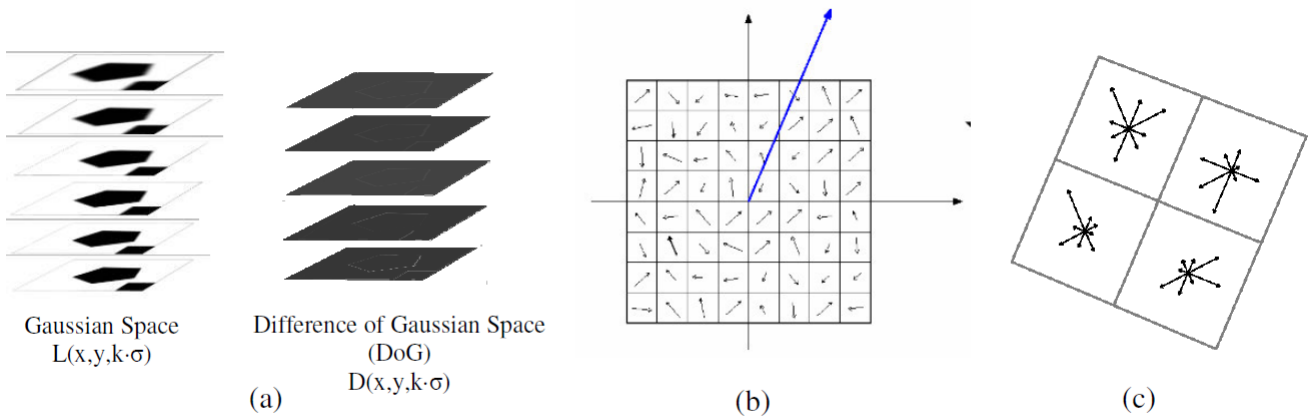


Abbildung 3.3: (a) Unterschied des Gaußschen (DoG) Skalenraums. (b) Überwiegende Ausrichtung des radiometrischen Gradienten. (c) SIFT Deskriptor. Bildquelle: [37]

Jedem lokalem Maximum der DoG Funktion ist eine dominante Orientierung des radiometrischen Gradienten zugewiesen, um die Invarianz gegenüber Rotation zu gewährleisten (Abbildung 3.3-b). Schließlich wird jedem Keypoint ein Deskriptor zugeordnet, der die extrahierten Features, welche die Helligkeitsinvarianz in der Umgebung zum Keypoint beschreiben, als 128-dimensionalen Vektor zusammenfasst (Abbildung 3.3-c). Den Deskriptor kann man als 3D-Histogramm der Lage und Orientierung des Gradienten verstehen. Die erkannten Deskriptoren werden in eine Datenbank gespeichert, in welcher dann auf Gemeinsamkeiten mit anderen Bildern geprüft werden kann. Dies geschieht über die Auswertung des Mindestabstands zwischen zwei Deskriptoren. Dazu erfolgt die Berechnung der Entferungen mithilfe des Euklidischen Abstands oder des Mahalanobis-Abstands (vgl. [15] S.28-29, [36] S.3, [37] S.3).

3.2.2 Features from Accelerated Segment Test

FAST, das 2006 von E. Rosten und T. Drummond vorgestellt wurde, hat eine ähnliche Vorgehensweise, ist jedoch um ein Vielfaches schneller als SIFT (vgl. [19] S.1). Wie in Abbildung 3.4 zu sehen ist, arbeitet FAST mit einem Segementtest, der einem Kreis von 16 Pixeln um einen Eckkandidaten p nutzt.

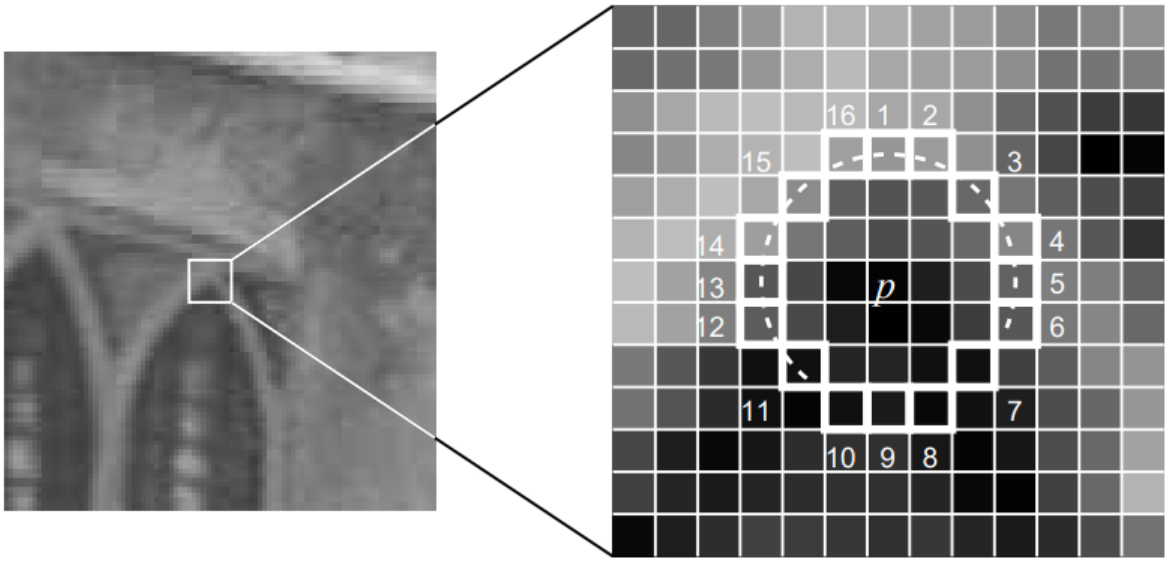


Abbildung 3.4: 12 Punkt Segment Test für die Eckenerkennung [19]

Der Detektor klassifiziert p als Ecke, wenn es ein Set von n zusammenhängenden Pixeln im Kreis gibt, die alle heller als der Kandidat I_p, addiert mit einem Grenzwert t oder alle dunkler als $I_p - t$ sind. n ist zwölf, da dies einen High Speed Test ermöglicht, mit dem eine große Anzahl an Nicht-Ecken schnell ausgeschlossen werden kann.

Der Test untersucht nur die vier Pixel 1, 5, 9 und 13. Wenn p eine Ecke ist, dann müssen mindestens drei der vier Pixel heller als $I_p + t$ oder dunkler als $I_p - t$ sein. Wenn dies nicht zutrifft, ist p keine Ecke (vgl. [19] S.4-5).

3.2.3 Binary Robust Independent Elementary Features

BRIEF (Binary Robust Independent Elementary Features) wurde 2010 von M. Calonder et al. vorgestellt [42]. Die Verwendung von BRIEF Deskriptoren hat sich aus den steigenden Anforderungen an eine stärker eingeschränkte Arbeitsumgebung für rechenintensive Computer Vision basierte Systeme entwickelt.

Dazu zählt beispielsweise die autonome Navigation in unbekanntem Terrain. Ältere Methoden wie etwa SIFT oder SURF neigen dazu, bei einem großen Set an Features viel Speicher zu benötigen. Der 128-dimensionale Vektor, mit dem beispielsweise SIFT die Features beschreibt, ist für Echtzeitanwendungen zu rechenintensiv. SURF etwa speichert die Histogramme der lokalen Gradienten als Gleitkommazahlen in einem 64-dimensionalen Vektor. Für Echtzeitanwendungen ist SURF jedoch ebenfalls zu teuer, da die Anzahl an zu speichernden Deskriptoren nicht gut skalierbar ist. BRIEF verwendet einen Ansatz, bei dem Bitvektoren aus Ausschnitten (Patches) des Bilds berechnet werden, nachdem diese geglättet wurden. Die Intensitätswerte der Pixel werden paarweise ausgewertet, um einzigartige oder gleiche Bits in jedem Bitvektor zu finden.

Für jeden geglätteten Bildpunkt wird ein Test τ für den Patch p durchgeführt (vgl. [40] S.13-14 , [42] S.3):

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & otherwise \end{cases} \tag{3.1}$$

Wobei $p(x)$ die Intensität des Pixels bei $x = (u, v)^T$ im zu untersuchenden geglätteten Bild Patch ist. Die Auswahl eines Sets an $n_d(x, y)$-Bildpatches definiert eindeutig ein Set an binären Tests. Der BRIEF Deskriptor wird als n_d dimensionaler Bitstring definiert:

$$f_{n_d}(p) := \sum_{1 \geq i \geq n_d} 2^{i-1} \tau(p; x_i, y_i) \tag{3.2}$$

Die Metrik, um die Deskriptoren zu vergleichen, ist die Hamming-Distanz, welche die Unterschiedlichkeit zwischen zwei Zeichenketten anhand der Auswertung der Anzahl verschiedener Zeichen für die gleichen Positionen in zwei Strings der selben Länge beschreibt (vgl. [40] S.13-14).

BRIEF ist nicht nur schneller als andere vergleichbare Algorithmen, sondern erzielt auch höhere Erkennungsraten, solange die Invarianz gegenüber größeren Rotationen in der Ebene nicht erforderlich ist. Aus praktischer Sicht hilft die Verwendung von BRIEF dabei, die benötigte Rechenleistung für das Matching von Deskriptoren zu reduzieren, so dass auch Geräte mit sehr begrenzter Rechenleistung in Echtzeit Feature und Image Matching verwenden können (vgl. [42] S.13-14).

3.2.4 Outlier Removal

„Outlier Removal“ oder auch „Ausreißerbeseitigung“ genannt, besteht aus einer Reihe von Techniken zur Entfernung von unerwünschten, falsch erkannten oder fehlerhaften Keypoints, beginnend mit günstigen Methoden (einfache geometrische Tests) und abschließend mit teuren, Homographie-basierten Tests (vgl. [15] S.28-29).

Die planare Homographie bezieht sich auf die dreidimensionale Lage zwischen zwei Bildebenen zueinander im Raum. Betrachtet man ein Set an korrespondierenden Punkten (x, y) im ersten Bild und ein Set (x', y') im zweiten Bild, dann bildet die Homographie H die Lage dieser beiden Ebenen zueinander wie folgt ab:

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.3}$$

Die Homographiematrix ist eine 3x3 Matrix mit 8 DoF (Degrees of Freedom). Sie wird standardmäßig normalisiert mit:

$$h_33 = 1 \tag{3.4}$$

oder

$$h^2{}_{11} + h^2{}_{12} + h^2{}_{13} + h^2{}_{21} + h^2{}_{22} + h^2{}_{23} + h^2{}_{31} + h^2{}_{32} + h^2{}_{33} = 1 \tag{3.5}$$

Anhand dieser planaren Homographie kann überprüft werden, ob zwei Keypoints, die in zwei Bildern erkannt worden sind und durch die Deskriptoren beschrieben wurden, wirklich aus geometrischer Sicht übereinstimmen. Sind Abweichungen zwischen den zwei Keypoints zu groß, werden diese als Ausreißer deklariert und anschließend entfernt oder korrigiert.

Homographie wird in vielen Bereichen, wie Panoramaerstellung, Bildausrichtung, perspektivischer Entzerrung oder in der Schätzung der Kameraposition in Augmented Reality angewandt. Das Resultat der Ausreißerbeseitigung ist dann eine Reihe von korrekten Keypoints, die sowohl aus Sicht der Deskriptoren als auch aus geometrischer Sicht übereinstimmen und welche dann als Ausgangspunkt für die Pose Estimation der Kamera verwendet werden können (vgl. [16]).

3.2.5 Pose Estimation

Der letzte Schritt in der Pipeline der Bestimmung der internen und externen Kameraparameter ist die Pose Estimation. Eric Marchand et al. (vgl. [50] S.2) beschreiben die Positionsschätzung als Problem, welches ursprünglich als „Space Resection“ bekannt ist und seinen Ursprung in der Photogrammetrie fand. Sie definieren sie folgendermaßen: „given a set of correspondences between 3D features and their projections in the images plane, pose estimation consists in computing the position and orientation of the camera“. Es gibt eine Vielzahl an Verfahren zur Lösung dieses inversen Problems. In Kapitel 4 werden Lösungen aus der Computer Vision vorgestellt. Bei photogrammetrischen Verfahren wird die Positionsschätzung heute hauptsächlich mit dem Bündelblockausgleich durchgeführt, der alle Parameter gleichzeitig in mehreren Iterationsschritten anpasst.

3.3 Der Bündelblockausgleich

Das Verfahren des Bündelblockausgleichs verwendet die Methoden der „Collinearity Condition “ (Kollinearitätsbedingung), der „Coplanarity Condition“ (Koplanaritätsbedingung) oder die Methode der direkten linearen Transformation (vgl. [23] S.66, [24] S.1).

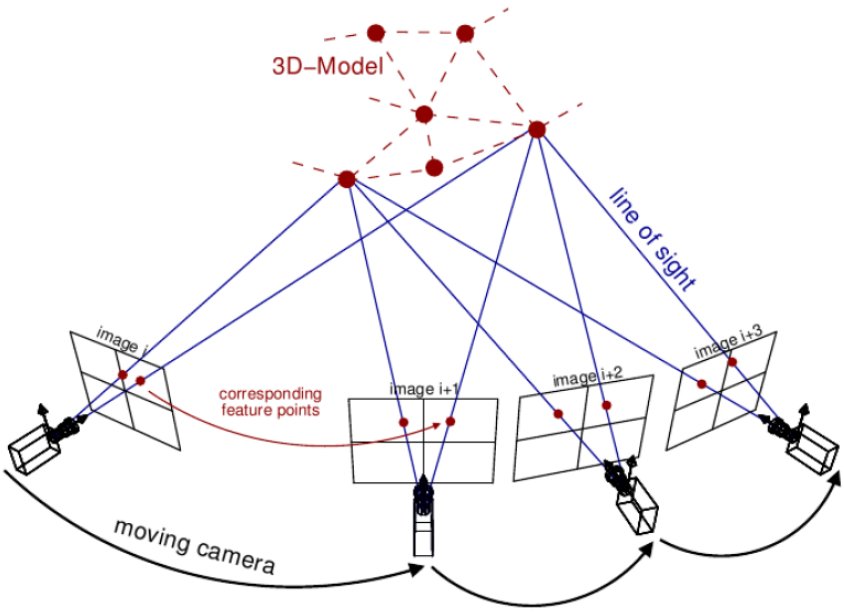


Abbildung 3.5: Visualisierung des Bündelblockausgleichs. Bildquelle [38]

Die gewünschten Parameter aller Fotos werden gleichzeitig durch eine iterative Wiederholung der „Least Squares“ Methode (Methode der kleinsten Quadrate) angepasst und korrigiert. Die Iterationen sind durch die nicht-Linearität der Konditionsgleichungen notwendig. Die Resulate des Bündelblockausgleichs aller Fotos sind dann die Ergebnisse der externen Orientierung der Kamera für jedes einzelne Foto (Siehe Abbildung 3.5). Weiterhin ergibt sich eine Auflistung der Objektraumkoordinaten der gemessenen Punkte aller Fotos, sowie deren gemessene statische Genauigkeit.

El-AShmawy [23] beschreibt die Verwendung von Strahlenbündeln, die durch Fotos generiert werden, zweifelsfrei als den flexibelsten Ansatz zur Blockbildung, Blockanpassung und für Photogrammetrie im Allgemeinen. In der Nahbereichsphotogrammetrie, bei der mehrstufige und konvergente Konfigurationen möglich sind, ist der Bündelansatz in seiner stärksten Form vertreten. Die Nahbereichsphotogrammetrie befasst sich beispielsweise mit der Erstellung von 3D-Modellen aus Bildersets oder Laserscandaten. Der Ausgleich der Strahlenbündel in einem Set an Fotos beinhaltet die Rotation und Translation von jedem Bündel im Raum in eine Position, in der sich alle Strahlen an der korrekten Position im Objektraum schneiden (vgl. [23] S.66-67). Im Allgemeinen besteht ein Bündelblockausgleichsproblem aus den folgenden neun Elementen ([34] S.1-2):

(1) Ein Projektionsmodell, welches die Projektion von 3D-Objektpunkte in 2D-Bildpunkte beschreibt. Dazu gehören innere und äußere Orientierung.

(2) Ein unbekannter Parametersatz, der bestimmt werden soll. Dies können entweder innere oder äußere Parameter sowie Objektpunkte sein.

(3) Eine Reihe an Beobachtungen, typischerweise Messungen von 2D-Bildpunkten oder Kamerapositionen.

(4) Ein Reihe bekannter Parameter, wie innere Orientierung oder Koordinaten von Kontrollpunkten.

(5) Optional eine Reihe von Einschränkungen zwischen Parametern oder Beobachtungen.

(6) Ein Verfahren, um die initialen Werte der unbekannten Parameter zu bestimmen.

(7) Ein Anpassungsalgorithmus, um die optimalen Parameterwerte anhand der initialen Werte zu finden.

(8) Ein Verfahren zur Erkennung von Ausreißern in den Beobachtungen.

(9) Eine Methode zur automatischen Erkennung von instabilen Parametern.

Die folgenden Abschnitten beziehen sich hauptsächlich auf Schritt (7), dem Finden der optimalen Parameterwerte anhand der initialen Daten. Anschließend werden die Verfahren der Kollinearitätsbedingung, der Koplanaritätsbedingung und die Methode der direkten linearen Transformation vorgestellt.

3.4 Nicht-lineare Methode der kleinsten Quadrate

Die „Method of Least Squares“ (Methode der kleinsten Quadrate) ist eine sehr leistungsfähige und flexible Technik, um alle Arten von Datenabgleichsproblemen zu lösen. Das Verfahren wird eingesetzt, um durch Minimierung der Summe des quadratischen Fehlers zwischen Funktion und Datenpunkten eine parametrisierbare Funktion an ein Datenset von Messwerten anzupassen. Die verschiedenen Werkzeuge dieser Methode können auch eingesetzt werden, um beispielsweise die Korrelationsqualität der Messdaten zu bewerten. Gleichzeitig ermöglicht das System durch die Berücksichtigung geometrischer Randbedingungen die Stabilisierung und Verbesserung der Korrelation. Wenn die anzupassende Funktion nicht linear ist, ist das Problem der kleinsten Quadrate ebenfalls nicht linear. Nichtlineare Lösungen der Methode der kleinsten Quadrate reduzieren iterativ die Summe der quadratischen Fehler zwischen Funktion und Messwerten durch eine Abfolge von Aktualisierungen der Parameterwerte (vgl. [29] S.1, [32] S.1).

In den folgenden Abschnitten werden verschiedene Methoden zur Bestimmung der externen Kameraparameter während des Bündelblockausgleichs vorgestellt. Die Methode der kleinsten Quadrate ist dabei essentiell für die Anwendbarkeit dieser Algorithmen, weswegen diese sowie das Gauss-Newton und das Levenberg-Marquardt Verfahren vorgestellt wird.

Die nicht lineare Methode der kleinsten Quadrate ist ein Standardoptimierungsproblem und wird definiert als ([28] S.2) :

$$minimiere : \sum_{i=1}^{m} f_i(x)^2 = ||f(x)||^2 \tag{3.6}$$

Wobei:

$f_1(x), ..., f_m(x)$ differenzierbare Funktionen einer Vektorvariablen x sind.

f eine Funktion von $\mathbf{R}^n$ zu $\mathbf{R}^m$ mit den Komponenten $f_i(x)$ ist:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \cdots \\ f_m(x) \end{bmatrix} \tag{3.7}$$

Das **Kameramodell** wird beschrieben durch die Parameter ([28] S.5-6): $A \in \mathbf{R}^{2\times3}, b \in \mathbf{R}^2, c \in \mathbf{R}^3, d \in \mathbf{R}$ welche die Position und Orientierung charakterisieren. Ein Objekt an Position $x \in \mathbf{R}^3$ erzeugt ein Bild an der Position $x' \in \mathbf{R}^2$ auf der Bildebene.

$$x\prime = \frac{1}{c^T x + d}(Ax + b) \tag{3.8}$$

$c^T x + d > 0$, wenn das Objekt vor der Kamera ist. Angenommen ein Objekt an Position x_{ex} wird von l Kameras betrachtet. (Beschrieben durch A_i, b_i, c_i, d_i) Das Bild des Objekts in der Bildebene von Kamera i ist an Position:

$$y_i = \frac{1}{c_i^T x_{ex} + d_i}(A_i x_{ex} + b_i) + v_i \tag{3.9}$$

Wobei v_i der Mess- oder Quantisierungsfehler ist. Das Ziel ist es nun, die dreidimensionale Position e_{ex} von den l Beobachtungen $y_1, ..., y_l$ zu schätzen. Dies kann mit der nicht-linearen Methode der kleinsten Quadrate berechnet werden. $\hat{x}$ wird durch die Minimierung von (3.9) berechnet (vgl. [28] S.5-6).

$$\sum_{i=1}^{l} ||y_i = \frac{1}{c_i^T x_{ex} + d_i}(A_i x_{ex} + b_i) + v_i||^2 \tag{3.10}$$

3.4.1 Gauß-Newton-Verfahren

Das Gauß-Newton Verfahren ist eine Technik zur Lösung der nicht linearen Methode der kleinsten Quadrate. Das Verfahren besteht aus einer Folge von linearen Annäherungen der kleinsten Quadrate an das nicht-lineare Problem, bei dem jedes einzelne durch einen direkten oder iterativen Prozess gelöst wird (vgl. [31] S.1). Gegeben ist die Definition der Problemstellung der kleinsten Quadrate, siehe (3.6). Beginnend mit einer anfänglichen Schätzung $x^{(1)}$, werden weitere Näherungslösungen $k = 1, 2, ...$ berechnet (vgl. [28] S.16-17):

Dazu wird f um $x^{(k)}$ linearisiert:

$$\overline{f}(x; x^{(k)}) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)}) \tag{3.11}$$

Danach wird die affine Annäherung $\overline{f}(x; x^{(k)})$ für f im Problem der kleinsten Quadrate ersetzt:

$$minimiere : ||\overline{f}(x; x^{(k)})||^2 \tag{3.12}$$

Die Lösung dieses linearen Problems wird nun als $x^{(k+1)}$ verwendet.

Das Problem der kleinsten Quadrate ist in Wiederholung k des Gauß-Newton Verfahrens gelöst:

$$minimiere : ||f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})||^2 \tag{3.13}$$

Wenn $Df(x^{(k)})$ linear unabhängige Spalten hat, wird die Lösung gegeben durch:

$$x^{k+1} = x^{(k)} - \Big(Df(x^{(k)})^T Df(x^{(k)})\Big)^{-1} Df(x^{(k)})^T f(x^{(k)}) \tag{3.14}$$

Der Gauß-Newton Schritt $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ ist:

$$\begin{aligned} \Delta x^{(k)} &= -\Big(Df(x^{(k)})^T Df(x^{(k)})\Big)^{-1} Df(x^{(k)})^T f(x^{(k)}) \\ &= -\frac{1}{2}\Big(Df(x^{(k)})^T Df(x^{(k)})\Big)^{-1} \nabla g(x^{(k)}) \end{aligned} \tag{3.15}$$

Wobei $\nabla g(x)$ der Gradient der Kosten für nichtlineare kleinste Quadrate ist.

Das Gauß-Newton Verfahren eignet sich besonders gut für die Verarbeitung von großen Datenmengen mit hoher Varianz. Im Vergleich zum normalen Newton-Verfahren ist der Algorithmus attraktiver, da er keine Auswertung der zweiten Ableitungen in der Hesse-Matrix der Zielfunktion benötigt (vgl. [31] S.1, [28] S.16-17).

3.4.2 Levenberg-Marquardt Verfahren

Das Levenberg-Marquardt Verfahren wurde 1944 von Kenneth Levenberg [33] veröffentlicht, in den 1960er Jahren von Donald Marquardt wiederentdeckt und entwickelt um nicht lineare Probleme der kleinsten Quadrate zu lösen. Der Algorithmus kombiniert zwei Minimierungsmethoden: „Gradient Descent“ (Gradientenverfahren) und das Gauß-Newton Verfahren. Beim Gradientenverfahren wird die Summe der quadratischen Fehler durch die Aktualisierung der Parameter in Richtung der steilsten Richtung zum Minimum hin reduziert. Das Levenberg-Marquardt Verfahren verhält sich wie das Gradientenverfahren, wenn die Parameter weit von ihrem optimalen Wert entfernt sind und wie das Gauß-Newton Verfahren, wenn die Parameter nahe an ihrem optimalen Wert liegen. Es wechselt also adaptiv die Parameterupdates zwischen den beiden Verfahren (vgl. [32] S.1).

Der Algorithmus befasst sich mit zwei Problembereichen des Gauß-Newton Verfahrens (vgl. [28] S.19):

- Wie aktualisiert man $x^{(k)}$, wenn die Spalten von $Df(x^{(k)})$ linear abhängig sind.
- Wie verfährt man, wenn das Gauß-Newton Update $||f(x)||^2$ nicht reduziert.

Das Verfahren wird mathematisch wie folgt beschrieben: Berechnung von $x^{(k+1)}$ durch Lösung eines normalisierten Problems der kleinsten Quadrate:

$$minimiere : ||\overline{f}(x; x^{(k)})||^2 + \lambda^{(k)}||x - x^{(k)}||^2 \tag{3.16}$$

$\overline{f}(x; x^{(k)})$ ist dabei wie beim Gauß-Newton Verfahren in Gleichung (3.11) gegeben. Mit $\lambda^{(k)} > 0$ gibt es immer eine eindeutige Lösung.

Das normalisierte Problem der kleinsten Quadrate wird in Iteration k gelöst:

$$minimiere : ||f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})||^2 + \lambda^{(k)}||x - x^{(k)}|^2 \tag{3.17}$$

Die Lösung wird dann gegeben durch:

$$x^{(k+1)} = x^{(k)} - \left(Df(x^{(k)})^T Df(x^{(k)}) + \lambda^{(k)} I\right)^{-1} Df(x^{(k)})^T f(x^{(k)}) \tag{3.18}$$

Der Levenberg-Marquardt Schritt $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ ist:

$$\begin{aligned}\Delta x^{(k)} &= -\Big(Df(x^{(k)})^T Df(x^{(k)}) + \lambda^{(k)} I\Big)^{-1} Df(x^{(k)})^T f(x^{(k)}) \\ &= -\frac{1}{2}\Big(Df(x^{(k)})^T Df(x^{(k)}) + \lambda^{(k)} I\Big)^{-1} \nabla g(x^{(k)})\end{aligned} \tag{3.19}$$

Für $\lambda^{(k)} = 0$ ist das der Gauß-Newton Schritt; für große $\lambda(k)$:

$$\Delta x^{(k)} \approx -\frac{1}{2}\lambda^{(k)} \nabla g(x^{(k)}) \tag{3.20}$$

Es gibt verschiedene Strategien um $\lambda^{(k)}$ anzupassen (vgl. [28] S.19-21):

- Nach Iteration k berechne die Lösung $\hat{x}$ von Gleichung (3.16).
- Wenn $||f(\hat{x})||^2 < ||f(x^{(k)})||^2$, dann verwende $x^{(k+1)} = \hat{x}$ und verringere λ
- Wenn $||f(\hat{x})||^2 > ||f(x^{(k)})||^2$, lasse x gleich (verwende $x^{(k+1)} = x^{(k)}$) und erhöhe λ

Das Levenberg-Marquardt Verfahren ist aufgrund seiner sehr einfachen Implementation und der Verwendung einer effektiven Dämpfungsstrategie, welche es ermöglicht, aus einer Vielzahl an initialen Situationen schnell an das Optimum zu konvergieren, sehr beliebt. Es hat sich zu einer Standardtechnik für nicht-lineare Probleme der kleinsten Quadrate entwickelt, die in verschiedensten Disziplinen zur Lösung von Datenanpassungsproblemen weit verbreitet ist (vgl. [25] S.1).

3.4.3 Die Kollinearitätsbedingung

Die Bestimmung der externen Kameraparameter mit Hilfe der Kollinearitätsbedingung, um die gleichzeitige Anpassung der Parameter im Bündelblockausgleich zu berechnen, erfolgt durch die Aufstellung von zwei Gleichungen für jeden gemessenen Bildpunkt. Die Lösung all dieser Gleichungen erfolgt dann nach der Methode der kleinsten Quadrate. Die Bedingung der Kollinearität sagt aus, dass ein Objektpunkt P, sein Bildpunkt p und das perspektivische Zentrum O alle auf der gleichen Geraden liegen müssen. Mathematisch wird die Bedingung wie folgt ausgedrückt (vgl. [23] S.67):

$$
\begin{aligned}
x_p &= -f\frac{(X_p - X_O)m_{11} + (Y_p - Y_O)m_{12} + (Z_p - Z_O)m_{13}}{(X_p - X_O)m_{31} + (Y_p - Y_O)m_{32} + (Z_p - Z_O)m_{33}} \\
y_p &= -f\frac{(X_p - X_O)m_{21} + (Y_p - Y_O)m_{22} + (Z_p - Z_O)m_{23}}{(X_p - X_O)m_{31} + (Y_p - Y_O)m_{32} + (Z_p - Z_O)m_{33}}
\end{aligned}
\tag{3.21}
$$

Dabei sind x_p und y_p die korrigierten Fotokoordinaten, X_p, Y_p, Z_p die Objektpunktkoordinaten von P, X_O, Y_O, Z_O, die Koordinaten des perspektivischen Zentrums O, f die kalibrierte Brennweite der Kamera und m_{ij} die Elemente der Orientierungsmatrix M des Fotos. Die linearisierte Form der Gleichung für die Lösung der Methode der kleinsten Quadrate wird gegeben als (vgl. [23] S.67):

$$
V + B \cdot \triangle = \varepsilon \tag{3.22}
$$

Wobei:

- $\triangle$ der Korrekturvektor zum aktuellen Werteset für die unbekannten Werte (innere und äußere Orientierung, Objektkoordinaten der Punkte) der iterativen Lösung ist.
- B die Matrix der partiellen Ableitungen von Gleichung (3.21) in Bezug auf die Unbekannten ist.
- V der Korrekturvektor zu den Beobachtungen ist.
- ε der Abweichungsvektor ist.

El-Ashmawy (vgl. [23] S.68) schlägt weitere Beschränkungen vor, indem ergänzende Beobachtungsgleichungen berücksichtigt werden, die sich aus den a priori Kenntnissen der Raumkoordinaten der Objekte der Kontrollpunkte in Gleichung (3.22) ergeben. Solche zusätzlichen Gleichungen können wie folgt beschrieben werden:

$$
V^c - \triangle^c = \varepsilon^c \tag{3.23}
$$

Wobei:

- $\triangle^c$ der Vektor der beobachtbaren Korrekturen zu den Objektkoordinaten der Kontrollpunkte ist.
- ε^c der Abweichungsvektor zwischen Beobachtungswerten und aktuellen (in iterativer Lösung) Werten der Objektkoordinaten der Kontrollpunkte ist.

Beobachtungsgleichungen kann man durch das Zusammenführen von Gleichung (3.22) und (3.23) erhalten.

Die grundlegenden Voraussetzungen an den Bündelblockausgleich sind die Schätzungen der Parameter für innere und äußere Orientierung der Kamera. Weiterhin können die Schätzungen für Objekt und Raumkoordinaten, je nach Ansatz, aller Kontrollpunkte nützlich sein. Deshalb sollte ein Bündelverfahren immer eine praktikable Methode enthalten, mit der die ungefähr geschätzten initialen Werte ermittelt werden können. Dieses Vorwissen sorgt nicht nur für eine reduzierte Anzahl an Iterationen, sondern resultiert auch in schnelleren und genaueren Ergebnissen (vgl. [23] S.67).

3.4.4 Die Koplanaritätsbedingung

Die Koplanaritätsbedingung ist ein weiteres Verfahren zur Lösung des Bündelblockausgleichs und impliziert, dass die beiden perspektivischen Zentren zweier Aufnahmen, ein beliebiger Objektpunkt und die entsprechenden Bildpunkte auf den beiden Fotos alle in einer gemeinsamen Ebene liegen müssen (vgl. Abbildung 3.6).

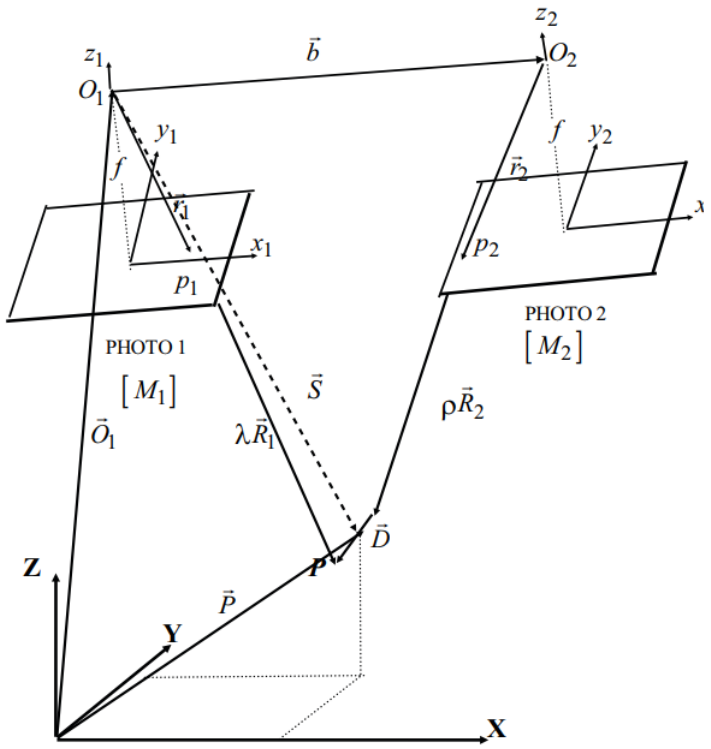


Abbildung 3.6: Koplanaritätsbedinung, Bildquelle [23]

Die Koplanaritätsbedingung kann wie folgt dargestellt werden (vgl. [23] S.68):

$$F_i = \begin{vmatrix} b_X & b_Y & b_Z \\ X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \end{vmatrix} = 0 \tag{3.24}$$

Dabei sind b_X, b_Y, b_Z die Komponenten des Basisvektors b und X_1, Y_1, Z_1, sowie X_2, Y_2, Z_2 die Komponenten des Vektors $\vec{R_1}$ (von O_1 zu P) beziehungsweise $\vec{R_2}$ (von O_2 zu P). Das mathematische Modell besteht aus vier skalaren Gleichungen:

$$\begin{aligned} X_p - (X_{O_1} + 0.5(b_X + \lambda \cdot X_1 + p \cdot X_2)) &= 0.0 \\ Y_p - (Y_{O_1} + 0.5(b_Y + \lambda \cdot Y_1 + p \cdot Y_2)) &= 0.0 \\ Z_p - (Z_{O_1} + 0.5(b_Z + \lambda \cdot Z_1 + p \cdot Z_2)) &= 0.0 \\ D_Y = \lambda \cdot X_1 - p \cdot X_2 - b_Y &= 0.0 \end{aligned} \tag{3.25}$$

Wobei $X_{O_1}, Y_{O_1}, Z_{O_1}$ die Objektkoordinaten der ersten Kameraposition während der Belichtung und λ und p die Skalierungsfaktoren der entsprechenden Positionsvektoren $\vec{r_1}$ und $\vec{r_2}$ im Kameraraum sind. Die linearisierte Form der Gleichungen (3.25) mit ebenfalls von El-Ashmawy (vgl. [23] S.68) vorgeschlagenen zusätzlichen Beschränkungen für das Verfahren der kleinsten Quadrate wird gegeben als:

$$\begin{aligned} A \cdot V + B \cdot \triangle &= \varepsilon \\ V^c - \triangle^c &= \varepsilon^c \end{aligned} \tag{3.26}$$

Wobei:

- $\triangle$ der Korrekturvektor zu dem aktuellen Werteset für die unbekannten Werte (innere und äußere Orientierung, Objektkoordinaten der Punkte) der iterativen Lösung ist.
- A die Matrix der partiellen Ableitungen von Gleichung (3.25) in Bezug auf die Beobachtungen (korrigierte Foto-Koordinaten auf den linken und rechten Fotos des gleichen Objektpunkts) ist.
- B die Matrix der partiellen Ableitungen von Gleichung (3.25) in Bezug auf die Unbekannten ist.
- V der Korrekturvektor zu den Beobachtungen ist.
- ε der Abweichungsvektor ist.

Zur Verwendung der iterativen Lösung der kleinsten Quadrate ist die Berechnung der Ausgangswerte von Unbekannten, wie beim Verfahren mit der Kollinearitätsbedinung, notwendig (vgl. [23] S.68).

3.4.5 Direkte lineare Transformation

Die Methode der „Direct Linear Transformation“ modelliert die Transformation zwischen Bildkoordinatensystem und Objektkoordinatensystem als lineare Funktion. Die direkte lineare Transformation kann aus den Kollinearitätsgleichungen abgeleitet werden und lässt sich mathematisch wie folgt ausdrücken (vgl. [27] S.72):

$$\begin{aligned} x &= \frac{L_1X + L_2Y + L_3Z + L4}{L_9X + L_{10}Y + L_{11}Z + 1} \\ y &= \frac{L_5X + L_6Y + L_7Z + L_8}{L_9X + L_{10}Y + L_{11}Z + 1} \end{aligned} \tag{3.27}$$

Wobei x, y die Bildkoordinaten, $L_1, ..., L_{11}$ die Transformationskoeffizienten und X, Y, Z die Objektkoordinaten des Punktes sind. Die Werte der inneren und externen Kameraparameter werden dann berechnet durch:

$$\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} = - \begin{bmatrix} L_1 & L_2 & L_3 \\ L_5 & L_6 & L_7 \\ L_9 & L_{10} & L_{11} \end{bmatrix}^{-1} \begin{bmatrix} L_4 \\ L_8 \\ 1.0 \end{bmatrix} \tag{3.28}$$

$$\begin{aligned} x_0 &= (L_1L_9 + L_2L_{10} + L_3L_{11})/(L^2{}_9 + L^2{}_{10} + L^2{}_{11}); \\ y_0 &= (L_5L_9 + L_6L_{10} + L_7L_{11})/(L^2{}_9 + L^2{}_{10} + L^2{}_{11}); \\ \omega &= tan^{-1}(-L_{10}/L_{11}); \\ \phi &= sin^{-1}(-L_9\sqrt{(L^2{}_9 + L^2{}_{10} + L^2{}_{11})} \\ \kappa &= cos^{-1}((x_0L_9 - L_1)/(cos\phi\sqrt{(x_0L_9 - L_1)^2 + (x_0L_{10} - L_2)^2 + (x_0L_{11} - L_3)^2})); \\ f &= (x_0L_9)/(cos\kappa \cdot \phi\sqrt{L^2{}_9 + L^2{}_{10} + L^2{}_{11}}) \end{aligned} \tag{3.29}$$

Wobei $X_0, Y_0, Z_0, \omega, \phi, \kappa$ die externen Kameraparameter, x_0, y_0 die Bildkoordinaten des optischen Zentrums und f die Brennweite ist.

Das direkte lineare Transformationsverfahren hat lange Zeit in den Bereichen Photogrammetrie, Computer Vision, Robotik und Biomechanik Verwendung gefunden. Dies liegt an der linearen Formulierung der Beziehung zwischen Objekt- und Bildkoordinaten. Weiterhin können Bildkoordinaten in einem nicht-orthogonalen System mit unterschiedlichen Skalen ausgedrückt werden, die Position des Koordinatensystems kann unbekannt, sowie die Brennweite beliebig sein und von Bild zu Bild variieren (vgl. [27] S.72).

4 Simultaneous Localization and Mapping

Um das Ziel des exakten Matchings von Realität und generierter virtueller Realität zu erreichen, ist „Camera Localization“ die Schlüsseltechnologie für alle Augmented Reality Anwendungen (vgl. [3] S.1). Weiterhin ist es wichtig, die Umgebung zu kartographieren, um Projektionsflächen für virtuelle dreidimensionale Objekte zu finden. Eine mögliche Lösung für dieses Problem ist die Verwendung von SLAM. In den letzten 20 Jahren hat sich bei SLAM ein Trend gezeigt, bei dem Kameras als einzige exterozeptive Sensoren verwendet werden. Der Hauptgrund dafür ist die Fähigkeit eines Kamera-basierten Systems, Tiefeninformationen, Farben, Texturen und Helligkeiten zu erkennen, was beispielsweise Objekt- oder Gesichtserkennung ermöglicht. Darüber hinaus sind Kameras preiswert, leicht und haben einen geringen Stromverbrauch (vgl. [9] S.57).

Erst in den letzten Jahren hat sich SLAM in Alltagsanwendungen profilieren können, hauptsächlich durch das Aufkommen von leistungsstarken Smartphones. Smartphones sind mobil und haben die nötige Rechenleistung um SLAM Algorithmen in Echtzeit auszuführen. Dies hat zu einer Erweiterung der Forschungsmöglichkeiten von SLAM basierten Ansätzen geführt und das Verfahren zu einer robusteren Technologie gemacht. Weiterhin verfügen Smartphones über eine Vielzahl an Sensoren, wie Beschleunigungssensor, Gyroskop, Magentometer oder GPS, mit denen die visuellen Daten ergänzt werden können, um die Map genauer und weniger anfällig für innere Drifteffekte zu machen, was jedoch wieder eigene komplexe Probleme bei der Verbindung von verschiedenen Sensordaten mit sich bringt (vgl. [10] S.4). Folgende Tabelle gibt eine Übersicht an Software Development Kits, die SLAM implementiert haben, sowie die freie Verfügbarkeit deren Quelltext:

	Vuforia	Wikitude	Metaio	ARToolKit	Kudan	EasyAR	MaxST	ARCore	ARKit
SLAM	✓	✓	✓	x	✓	✓	✓	✓	✓
Open Source	x	x	x	✓	x	x	x	✓	x

Die Verwendung von SLAM im mobilen Bereich hat jedoch einige Probleme zu meistern. Durch die Ungenauigkeiten der absoluten Lokalisierung ist es schwierig, kontextsensitive Augmentation zu erzeugen. Auch wenn es möglich ist, bekannte Objekte im Raum während des Trackings zu bestimmen und die Information um diese herum zu zeigen, ist es fast unmöglich, den kompletten Raum einer Szenerie zu bestimmen. Das zweite Problem liegt in der Ergonomie und Benutzbarkeit der Anwendung durch den Endbenutzer. Typischerweise soll ein Endverbraucher keinem komplexen Protokoll folgen müssen, um sich selbst in der Szene zu lokalisieren. Das System benötigt also ein Verfahren zur schnellen und robusten Lokalisierung im Raum (vgl. [3] S.1).

SLAM als Problemstellung ist die Auswertung einer unbekannten Umgebung und Erstellung einer Karte, während gleichzeitig die lokale Position innerhalb dieser Karte bestimmt wird. Die Lösung dieses SLAM Problems war vor allem in der Robotik eine fundamentale Aufgabe der letzten zwei Jahrzehnte. Dabei ist SLAM ein Alltagsproblem: das Problem der räumlichen Erkundung. Jeder Mensch und jedes Tier hat dies gemeistert und benutzt es unterbewusst zur Navigation in unserer räumlichen Umgebung. Wichtige Anwendungsbereiche von SLAM sind die automatische Fahrzeugsteuerung auf unbekanntem Terrain, Rettungseinsätze in Umgebungen mit hohem Risiko, planetarische, terrestrische oder ozeanische Exploration, Augmented Reality Anwendungen, visuelle Überwachungssysteme oder Anwendungen in der Medizin. Die Lösung dieses Problems ist sehr komplex, wenn es für einen Roboter oder für Smartphones automatisiert ausgeführt werden soll. Durch das beherrschen dieser Technik können beispielsweise Roboter oder Fahrzeuge wirklich autonom gesteuert werden. Bei SLAM wird die Bewegung des Objekts an sich durch den Raum und die Position aller zur Positionsbestimmung notwendigen Merkmale berechnet, ohne vorheriges Wissen über Position oder Lage im Raum (vgl. [2] S. 1-2, [9] S.56).

Dabei benötigt man mindestens einen exterozeptiven Sensor um Informationen der Umgebung zu sammeln. SLAM besteht aus drei grundlegenden Operationen, die iterativ pro Zeitintervall ausgeführt werden (vgl. [4] S.3):

Die Kamera bewegt sich und erreicht eine neue Position in der Umwelt. Diese Bewegung erzeugt durch unvermeidbares Rauschen und Fehler Ungewissheit über die wirkliche absolute Position. Eine automatisierte Lösung benötigt ein mathematisches Modell für diese Bewegung. Dies ist das „*Motion Model*“.

Neue Features werden in der Umgebung entdeckt, welche in die Umgebungskarte aufgenommen werden müssen. Diese Features heißen „Landmarks“. Da die Position der Landmarks genauso wie die aktuelle Position der Kamera, durch Fehler in den

exterozeptiven Sensoren ungewiss ist, müssen diese beiden Faktoren passend arrangiert werden. Eine automatisierte Lösung benötigt ein mathematisches Modell, das die Position der Landmarks anhand der Sensordaten bestimmt. Dies ist das „*Inverse Oberservation Model*“.

Schon gemappte Landmarks werden entdeckt, welche dann verwendet werden, um die eigene Position, sowie die aller anderen Landmarks, zu korrigieren. Diese Operation reduziert die Unsicherheit über den aktuellen Standort, sowie die der Landmarks. Die automatisierte Lösung erfordert ein mathemathisches Modell, um die Werte der Messungen aus den prognostizierten Positionen der Landmarks und der aktuellen Position der Kamera zu berechnen. Dies ist das „*Direct Observation Model*“

Mit den drei Modellen ist es möglich, eine automatisierte Lösung für SLAM zu entwerfen. Die Lösung muss diese drei Modelle verbinden und alle Daten korrekt, organisiert und aktuell halten, sowie die korrekten Entscheidungen bei jedem Iterationsschritt machen (vgl. [4] S.2-3).

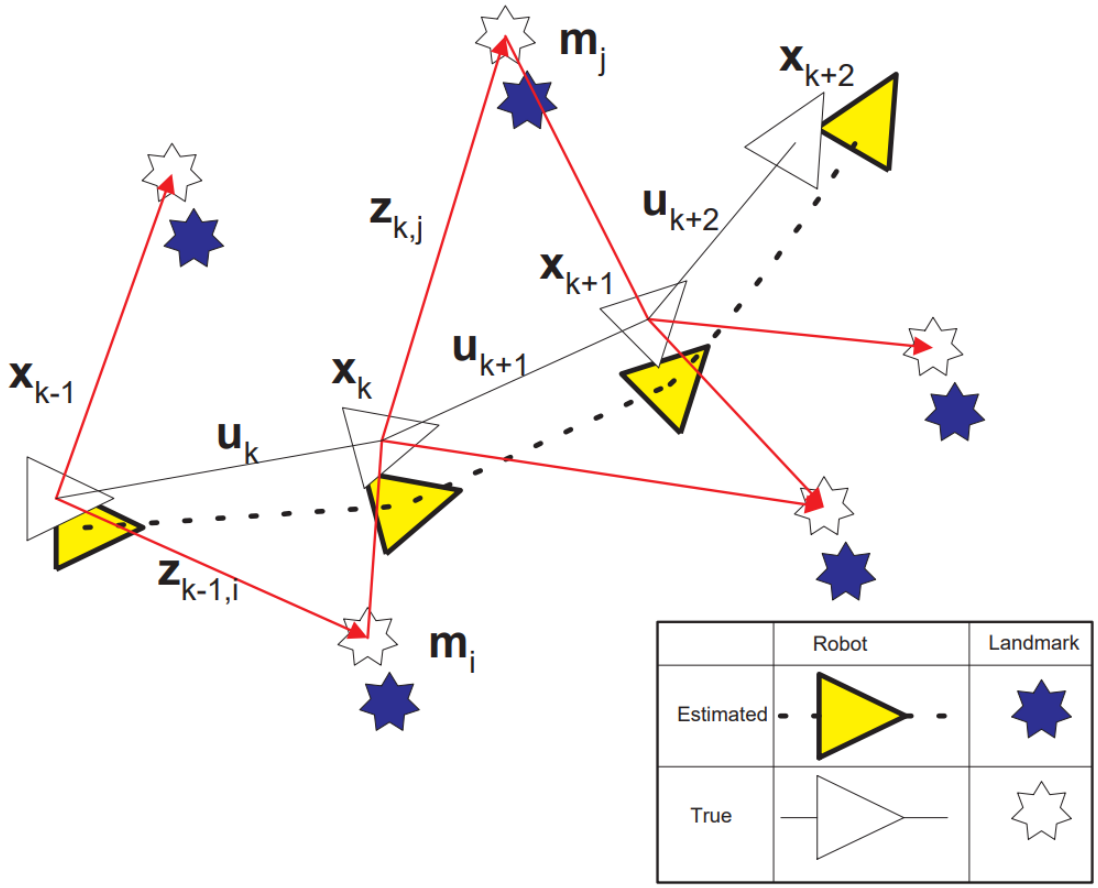


Abbildung 4.1: Das SLAM Problem: Die wahren absoluten Positionen der extrahierten Features sind nie wirklich bekannt. Bildquelle [2]

Eine erfolgreiche Lösung des SLAM Problems setzt weiterhin die Lösung des „Loop Closure Detection“ Problems voraus. Dabei müssen bereits besuchte Orte und Landmarks in einer beliebig großen Karte erkannt werden. Eins der größten Hindernisse, wenn es um die Skalierbarkeit der Lösung des SLAM Problems geht, ist die mögliche Komplexität von großen Karten. Wichtig ist, dass die Loop Closure Detection keine Falsch-Positiven Ergebnisse liefert, da dies die Integrität und Korrektheit der kompletten Karte beeinflussen würde (vgl. [10] S.4, siehe Kapitel 4.1).

Wie in Abbildung 4.1. erkennbar ist, bewegt sich in diesem Beispiel ein Roboter durch eine unbekannte Umgebung und nimmt mit seinem Sensor Features der näheren Objekte (Landmarks) auf. Wobei x_k der Vektor des Roboters, u_k der Bewegungsvektor, m_i der Vektor des Landmarks und $z_i k$ die Oberservation eines Landmarks durch den Roboter zur Zeit k sind. Wie man sehen kann, ist der Fehler zwischen echten und geschätzten Landmarks bei allen geschätzten Landmarks ähnlich, was an der initialen Betrachtung der Umgebung liegt. Zu diesem Zeitpunk wird nur das erste Feature erkannt. Daraus kann man schließen, dass die Fehler in der Schätzung der Landmarkpositionen korrelieren. Praktisch bedeutet dies, dass die relative Position zweier Landmarks, $m_i - m_j$ zueinander sehr genau sein kann, auch wenn die absolute Position sehr ungenau ist (vgl. [2] S.2).

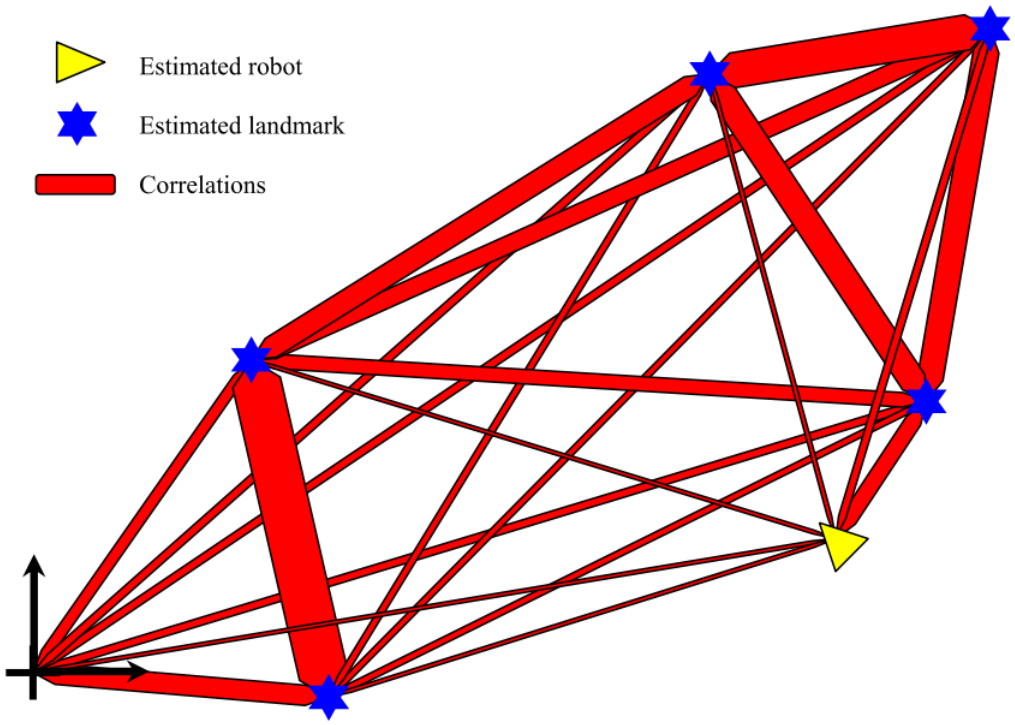


Abbildung 4.2: Die Landmarks sind durch „Federn“ verbunden, welche die Korrelation zwischen ihnen darstellen. Bildquelle [2]

Je mehr Landmarks in das Modell aufgenommen werden, desto besser wird das Modell der relativen Positionen, egal wie sich der Roboter bewegt. Dieser Prozess wird in Abbildung 4.2. veranschaulicht.

Während sich der Roboter durch die Umgebung bewegt, werden die Korrelationen stetig aktualisiert. Je mehr Beobachtungen über die Umwelt gemacht werden, desto steifer werden die Federn in diesem Modell. Im Nachhinein werden neue Beobachtungen von Landmarks durch das ganze Netzwerk propagiert und je nach Input, kleinere oder größere Anpassungen vorgenommen.

4.1 Loop Closure Detection

Die „Loop Closure Detection“ (Schleifenerkennung) besteht in der Erkennung von bereits besuchten Orten nach der zyklischen Erkundung der Umwelt und nach beliebiger Länge des Weges. Dies ist eins der größten Hindernisse, um SLAM im großen Maßstab zu verwenden, da kritische Fehler vermieden werden müssen, die schon durch ein falsch-positives Ergebnis entstehen.

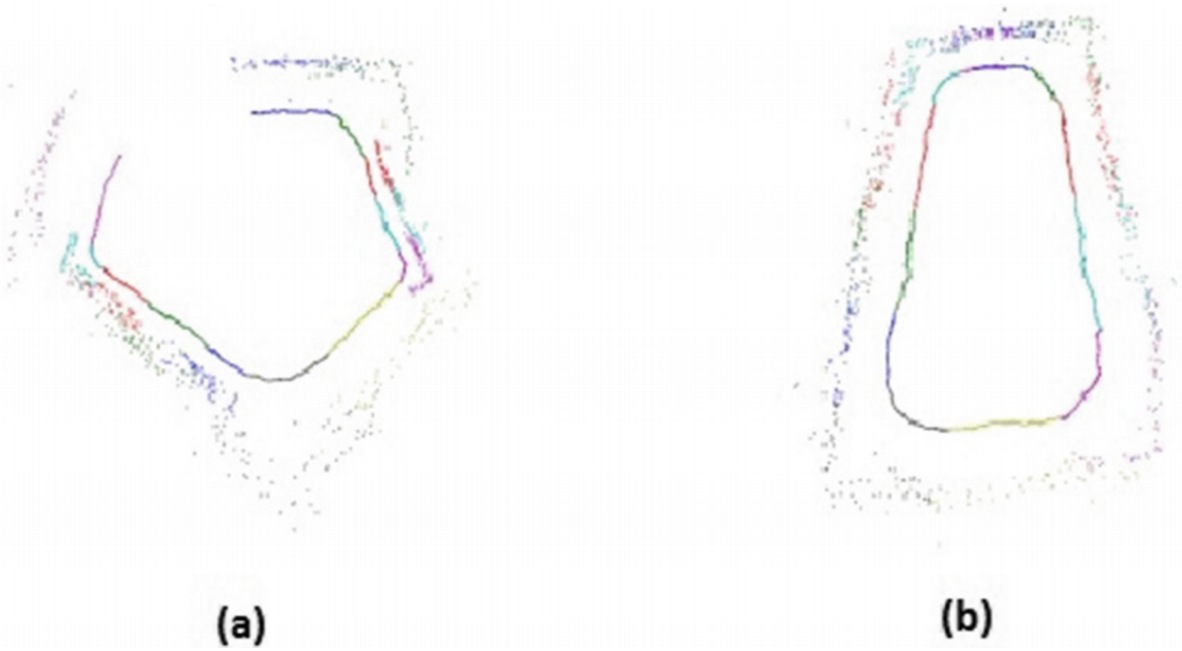


Abbildung 4.3: Lösung des SLAM Problems ohne (a) und mit (b) einem Loop Closure Algorithmus. Die innere Kurve ist der Pfad des mobilen Systems, die äußeren Punkte stellen die Features in der Karte dar. Bildquelle [39]

Es ergibt sich ein weiteres Problem, das „Perceptual Aliasing“ genannt wird und bei dem zwei verschiedene Orte aus der Umgebung als ein gleicher Ort erkannt werden. Dies stellt vor allem bei sich wiederholenden Features in der Umgebung, wie Fluren, ähnlichen architektonischen Elementen oder Zonen mit vielen Büschen oder Bäumen, ein großes Problem dar. Eine gute Methode zur Schleifenerkennung darf keine falsch-positiven Ergebnisse und nur ein Minimum an falsch-negativen Ergebnissen liefern. Deshalb müssen alle Loop Closure Systeme auf eine Präzision von 100 % abzielen. Dies wird benötigt, da ein einzelnes falsch-positives Ergebnis zu irreparablen Fehlern in der Erstellung der Karte führt. Falsch-negative hingegen reduzieren nur den Anteil der Rückrufquote, haben aber keinen Einfluss auf die Präzision der Genauigkeit der Karte (vgl. [9] S.64-65).

Algorithmen für die Schleifenerkennung lassen sich in drei Gruppen unterteilen (vgl. [39] S.212):

- **map-to-map**: Die Schleifenerkennung erfolgt durch die Aufteilung der globalen Karte in Teilkarten und dem Finden von Übereinstimmungen zwischen diesen Teilkarten.
- **image-to-map**: Die Suche wird anhand von Übereinstimmungen zwischen Bild und Karte ermöglicht und die Position des Systems wird dann in Bezug zur Karte wiederhergestellt.
- **image-to-image**: Die Schleifenerkennung wird anhand von Bildfeatures zwischen verschiedenen Bildern durchgeführt.

Nach der Erkennung einer Schleife wird das globale Kartenmodell angepasst, um die gewonnenen Erkenntnisse in die Karte zu übernehmen (vgl. [39] S.213).

4.2 Visual SLAM

Wenn Kameras als einziger exterozeptiver Sensor verwendet werden, spricht man von „Visual SLAM“ oder „Vision-Based SLAM“. Viele visuelle SLAM Ansätze haben jedoch Probleme wenn sie unter folgenden Bedingungen eingesetzt werden:

- In externen, dynamischen Umgebungen
- In Umgebungen mit sehr wenigen oder sehr vielen markanten Merkmalen
- In sehr großflächigen Umgebungen

- Bei sehr unregelmäßigen Bewegungen der Kamera
- Bei partieller oder großflächiger Verdeckung des Sensors

Ein Schlüsselelement von erfolgreichen SLAM Systemen ist die Fähigkeit, trotz dieser Schwierigkeiten korrekt zu arbeiten (vgl. [9] S.56).

Lösungen für das visuelle SLAM Problem benötigen eine angemessene Repräsentation für die Observierungen der Landmarks, welche eine konsistente und schnelle Berechnung ermöglichen. Die geläufigste Repräsentation besteht in der Form einer Zustandsraumdarstellung mit gaußschem Rauschen, was zur Verwendung des „Extended Kalman Filter“ (EKF) führt (vgl. [2] S. 2-4). Weitere gängige Lösungen für das SLAM Problem sind „Maximum Likelihood Techniques“, „Sparse Extended Information Filters“ (SEIFs), „Rao Blackwellized Particle Filters“ (RBPFs), „FAST-SLAM“ und „ORB-SLAM“ (vgl. [7] S. 2). Im folgenden wird der Extended Kalman Filter vorgestellt, der in vielen SLAM Systemen zum Einsatz gekommen ist, sowie seine Weiterentwicklungen FAST-SLAM und ORB-SLAM.

4.3 Extended Kalman Filter - SLAM

Eine der ersten Methoden zur Lösung des SLAM Problems war die Verwendung von Extendet Kalman Filter SLAM, welches um 1990 entwickelt wurde ([40] S.1). Der Kalman Filter ist eine Schätzfunktion für das „Linear-Quadratic-Problem“, welches das Problem der Schätzung des augenblicklichen Zustands eines linearen dynamischen Systems, gestört durch weißes Rauschen, darstellt. Der Kalman Filter wird auch dazu verwendet, um die mögliche Zukunft von dynamischen Systemen vorherzusagen, die von Menschen nicht kontrolliert werden können, wie zum Beispiel die Flugbahn von Himmelskörpern oder der Kurs von gehandelten Rohstoffen (vgl. [5] S.1).

Der Kalman Filter besteht aus drei Schritten. Zuerst wird eine Messung vorhergesagt, welche dann mit der realen Messung verglichen wird. Die resultierende Differenz wird mit der Varianz der Messung gewichtet, um daraus eine neue Schätzung des Zustands zu erhalten (vgl. [8] S.13). Der Kalman Filter lässt sich jedoch nur auf lineare Systeme anwenden. Der Extended Kalman Filter verwendet für die Vorhersage der Messungen und der Zustände hingegen nicht-lineare Funktionen (vgl. [8] S.16-17). Beim Extended Kalman Filter - SLAM ist die Map ein großer Stapel an Vektor und Sensordaten, sowie Zuständen von Landmarks.

$$x = \begin{bmatrix} R \\ M \end{bmatrix} = \begin{bmatrix} R \\ L_1 \\ ... \\ L_n \end{bmatrix} \tag{4.1}$$

R ist der Zustand des Sensors und $M = (L_1, ..., L_n)$ ist ein Set an Zuständen der Landmarks. Beim Extended Kalman Filter wird die Karte durch eine gaußsche Variable modelliert, die den Mittelwert und die Kovarianzmatrix des Zustandsvektors verwendet, die jeweils durch $\overline{x}$ und P beschrieben werden. Das Ziel ist es, die Map $\{\overline{x}, P\}$ zu allen Zeiten auf dem aktuellen Stand zu halten.

$$\overline{x} = \begin{bmatrix} \overline{R} \\ \overline{M} \end{bmatrix} = \begin{bmatrix} \overline{R} \\ \overline{L_1} \\ ... \\ \overline{L_n} \end{bmatrix} \qquad P = \begin{bmatrix} P_{RR} & P_{RM} \\ P_{MR} & P_{MM} \end{bmatrix} = \begin{bmatrix} P_{RR} & P_{RL1} & ... & P_{RLn} \\ P_{L1R} & P_{L1L1} & ... & P_{L1Ln} \\ ... & ... & ... & ... \\ P_{LnR} & P_{LnL1} & ... & P_{LnLn} \end{bmatrix} \tag{4.2}$$

Diese stochastische Karte wird durch die Vorhersage- und Korrekturprozesse des EKF in Stand gehalten. Um eine echte Erkundung der Umgebung zu erreichen, wird der EKF Algorithmus mit einem zusätzlichen Schritt der Landmark Erkennung und Initialisierung gestartet, bei dem neue Landmarks zu der Karte hinzugefügt werden. Die Landmark Initialisierung erfolgt durch eine Umkehrung der Bewertungsfunktion und der Verwendung dieser und der Ableitungsmatrix, um die beobachteten Landmarks und die benötigten Co- und Crossvarianzen für den Rest der Karte zu berechnen. Diese Beziehungen werden dann an den Zustandsvektor und die Kovarianzmatrix angehängt (vgl. [4] S.6-7).

Eine zentrale Einschränkung des EKF basierten SLAM Ansatzes ist die Komplexität der Berechnung. Sensor-Updates benötigen Zeit, quadratisch zur Anzahl der Landmarks K, die zu berechnen sind. Diese Komplexität ergibt sich aus der Tatsache, dass die vom Kalman-Filter verwaltete Kovarianzmatrix $O(K^2)$ Elemente enthält, die alle aktualisiert werden müssen, auch wenn nur ein einzelnes Landmark beobachtet wurde. Diese Komplexität limitiert die Anzahl an Landmarks, die durch diesen Ansatz verarbeitet werden konnen, auf ein paar Hundert, während natürliche Umgebungsmodelle häufig Millionen von Features enthalten (vgl. [6] S.1).

4.4 FAST-SLAM

Ein SLAM Verfahren, welches sich aus EKF-SLAM entwicklelt hat, ist FAST-SLAM (vgl. [40] S.1). FAST-SLAM (Features from Accelerated Segment Test SLAM 1.0) zerlegt das SLAM-Problem in ein Lokalisierungsproblem des Roboters und eine Reihe von Landmark Schätzungsproblemen, die auf der Schätzung der Roboterposition beruhen. Fast-SLAM verwendet einen modifizierten Partikelfilter zur Schätzung der „A posteriori“ Wahrscheinlichkeit für die Roboterposition. Partikelfilter sind vom Prinzip her ähnlich wie Kalman Filter. Diese werden auch zur Schätzung von Zuständen verwendet, können aber viele verschiedene mögliche Zustände betrachten. Diese Anzahl an Zuständen werden Partikel genannt. Jedes Partikel besitzt wiederum Kalman-Filter, welche die Positionen der Landmarks abhängig von der Pfadschätzung beurteilen.

Eine naive Implementation dieser Idee führt zu einem Algorithmus, der $O(MK)$ Zeit benötigt, wobei M die Anzahl an Partikeln im Partikel Filter und K die Anzahl an Landmarks ist. Mit der Verwendung einer Baumstruktur kann die Laufzeit von FAST-SLAM auf $O(MlogK)$ reduziert werden, was diesen Algorithmus deutlich schneller als EKF basierte SLAM Algortihmen macht (vgl. [6] S.1-2, [8] S.18-19).

Bei Fast-SLAM werden nicht nur die unterschiedlichen geschätzten Positionen verwendet, sondern auch verschiedene Maps der Umgebung betrachtet. Da verschiedenste Maps mit verschieden Warscheinlichkeiten der geschätzten Positionen betrachtet werden, können sich Fehler in der Map, die sich durch falsch gemessene oder geschätzte Positionen ergeben, durch die Anzahl an verschieden Maps relativieren. Es können sich im Laufe der Zeit Maps, die vorher weniger wahrscheinliche Positionen hatten, als richtig herausstellen (vgl. [8] S.23).

Fast-SLAM 2.0 ist eine weiterentwickelte Variante von Fast-SLAM. Hierbei wird eine Vorauswahl getroffen, berechnet durch einen weiteren Kalman Filter, in wie weit und mit welcher Varianz die Partikel um den Mittelpunkt verteilt werden. Es ergibt sich eine bessere Auswahl des Mittelpunktes und des Streuradius für den Partikelfilter als in Fast-SLAM 1.0. Diese Methode reduziert die Anzahl an Partikel und generierten Maps und rechnet daher schneller (vgl. [8] S.29-30).

4.5 ORB-SLAM

ORB-SLAM (Oriented FAST and Rotated BRIEF - SLAM) ist eine Weiterentwicklung und Kombination von den Feature Erkennungsalgorithmen FAST und BRIEF. ORB Deskriptoren sind schnell berechnet und können sehr schnell verglichen werden, während sie gleichzeitig einen gute perspektivische Invarianz besitzen (vgl. [41] S.3). Der erste Schritt von ORB-SLAM besteht darin, ORB-Features aus dem Bild zu extrahieren. Dies geschieht durch das Finden von Ecken mit FAST und der anschließenden Erstellung eines Deskriptors mit BRIEF.

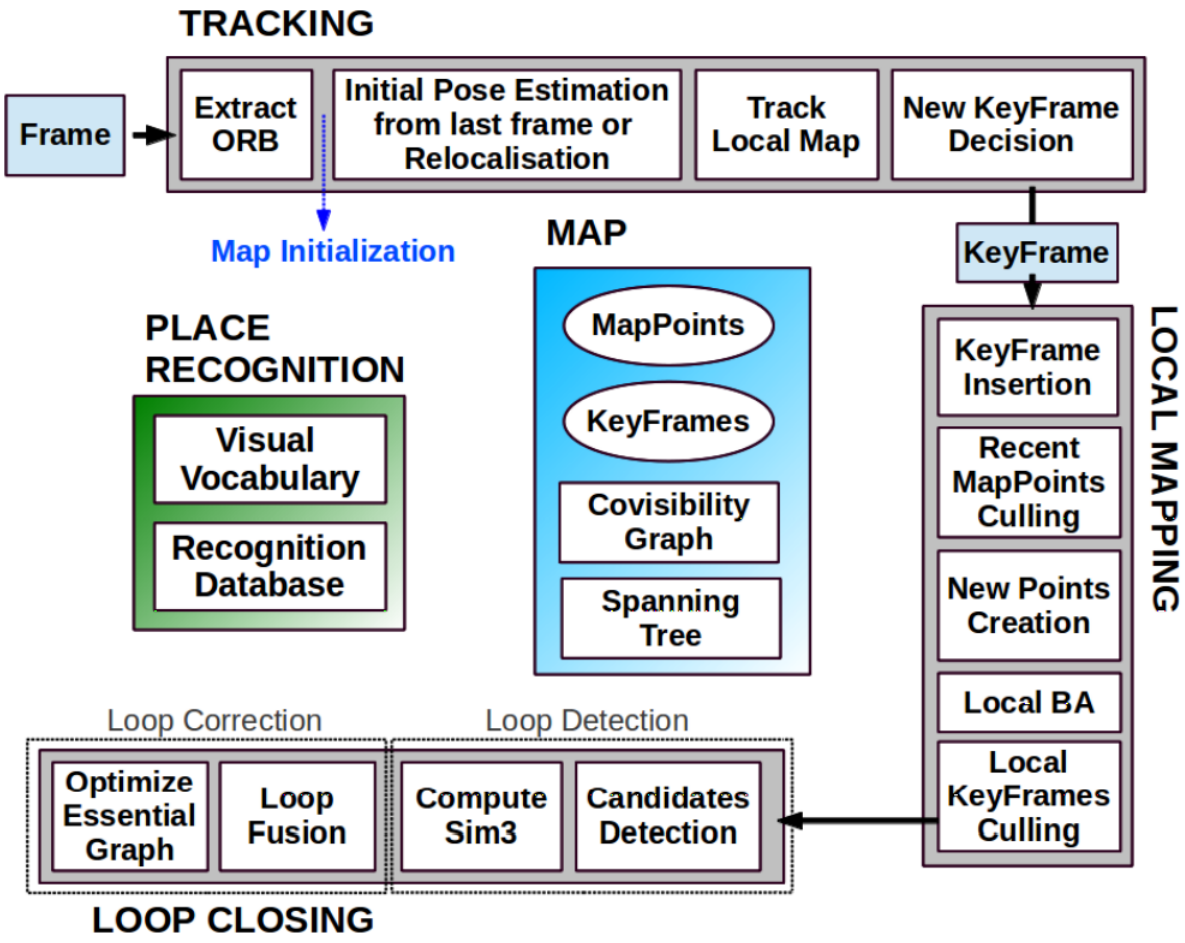


Abbildung 4.4: ORB-SLAM System Übersicht: Tracking, Mapping und Loop Closing. Bildquelle [41].

ORB-SLAM arbeitet mit verschiedenen Threads, um das Tracking, Mapping und Loop Closing parallel auszuführen.

Der **Tracking** Thread ist für die Lokalisierung der Kamera zuständig und entscheidet auch, welche Keyframes ausgewählt werden. Keyframes sind die Menge der Frames, die für die Erstellung der Map verwendet werden. Dabei wird mit einem „Survival of the Fittest“ Ansatz gearbeitet, der die nützlichsten Keyframes findet. Die initiale Positionsschätzung wird durch das Feature Matching zwischen ausgewähltem und vorherigem Keyframe erreicht. Danach wird die aktuelle Position mithilfe eines Bewegungsmodells optimiert, um die korrespondierenden Map Points aus dem Keyframe vorherzusagen und die 3D-Rekonstruktion zu optimieren. Wenn das Tracking eines vorherigen Frames verloren geht, wird ein globaler Lokalisierungsalgorithmus verwendet, um ein neues Keyframe zu finden, das dann wieder mit dem aktuellen Frame verglichen werden kann. Ein erfolgreicher Tracking Schritt resultiert in dem Finden einer Kamerapose und einem initialen Set an Features, aus denen dann eine lokale Karte berechnet werden kann.

Die **Map** wird aus den ermittelten Keyframes und den darin enthaltenen Map Points im zweiten Thread erstellt. Wenn ein Keyframe erfasst wird, wird es in einem Graphen abgelegt, der dieses als Node und die gemeinsam beobachteten Map Points in den verschiedenen Keyframes als Verbindungen zwischen den Nodes modelliert. Die Gewichtung der Kanten ist die Anzahl der Map Points, die auf beiden Frames observiert wurden. Um ein unbeschränktes Wachstum des Graphen zu vermeiden, werden Keyframes aussortiert, die im Vergleich zu anderen zu ähnlich sind. Map Points werden ebenfalls aussortiert, wenn sie von zu wenigen Keyframes gleichzeitig erfasst werden. Auf diese Weise können Keyframes sehr großzügig in den Graphen eingefügt und bei Redundanz wieder entfernt werden.

Das **Loop Closing** wird vom dritten Thread ausgeführt. Wenn erkannt wird, dass ein bestimmter Ort schon einmal besucht wurde, kann der Drift, der sich beim Durchlaufen der Schleife angesammelt hat, korrigiert werden. Bei ORB-SLAM wird die Schleifenerkennung beim Einfügen jedes Keyframes in den Graphen vorgenommen. Dazu wird die Ähnlichkeit zwischen den Keyframes betrachtet, indem ein Ähnlichkeitswert mithilfe der Anwendung des „Bag-of-Words“ Modells berechnet wird. Wenn ein Keyframe mehr Ähnlichkeiten mit einem neu eingefügten Keyframe aufweist als seine Nachbarn im Graphen, ist es ein Kandidat für die Schleifenschließung. Wenn drei miteinander verbundene Kandidaten gefunden wurden, gilt der Loop als akzeptiert und die Schleife wird geschlossen. Abschließend wird der Graph aktualisiert (vgl. [41] S.6-8, [40] S.4-5).

4.6 Structure from Motion

Structure from Motion (SfM) ist ein Oberbegriff unter dem verschiedene Verfahren zusammengefasst werden können. Er steht für die gleichzeitige Schätzung der Struktur einer dreidimensionalen Szene und der Bewegung des Sensors in dieser Szene, erstellt aus mehreren überlappenden Bildern einer Szene (siehe Abbildung 4.5). Die Rekonstruktion der 3D-Struktur der Originalobjekte aus diesen 2D-Bildsequenzen wird dann als SfM bezeichnet.

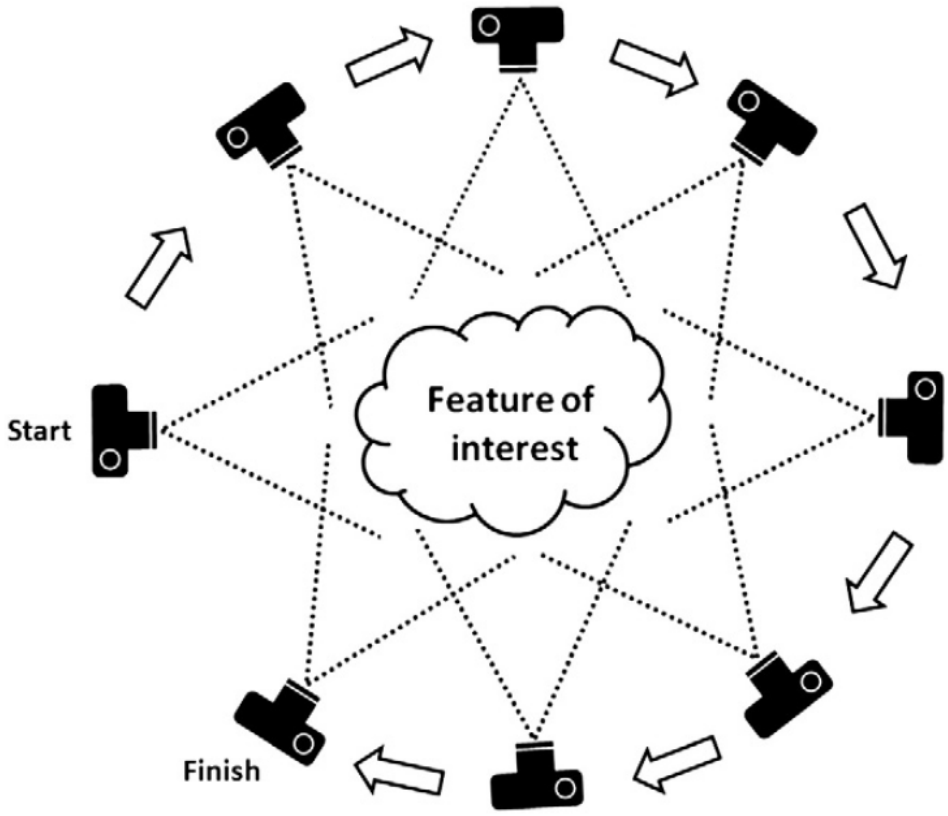


Abbildung 4.5: Structure from Motion: Überlappende Fotos zur Rekonstruktion der 3D Szene. Bildquelle [45]

Structure from Motion hat in der Computer Vision Community große Beachtung gefunden. Kürzlich vorgestellte Systeme haben erhebliche Fortschritte bei der bildbasierten Modellierung und beim Rendering gemacht. Die meisten Structure from Motion Systeme arbeiten sequentiell, beginnend mit einer kleinen Rekonstruktion aus beispielsweise zwei Bildern und fügen dann schrittweise neue Aufnahmen durch die Pose Estimation der Kamera hinzu. Die Punkte im 3D-Modell werden durch Triangulation oder mit hierarchischen Verfahren eingefügt.

Diese SfM Ansätze erfordern je Sequenz die Durchführung des Bündelblockausgleichs, sowie mehrere Iterationen der Ausreißerbeseitigung, um die Ausbreitung von Fehler zu minimieren, während die Rekonstruktion wächst. Für große Datensets ist dies sehr rechenintensiv (vgl. [44] S.1).

Structure from Motion ist nicht nur in der Computer Vision vertreten, sondern hat auch in der Photogrammetrie bereits Anwendungsbereiche gefunden. In den letzten Jahrzehnten sind verschiedene Ansätze zur Lösung des SfM Problems entstanden, wie die „Maximum likehood estimation“, „Kalman und Extended Kalman Filter“, „Particle Filter“ oder das „Hidden Markov Model“ (vgl. [38] S.5).

SfM unterscheidet sich von der herkömmlichen Photogrammetrie, indem die Geometrie der Szene, die Kameraposition und die Ausrichtung automatisch ohne die Kenntnis von a priori definierten 3D-Zielen berechnet werden kann (vgl. [45] S.301).

4.7 Weitere Verfahren - SLAM mit Stereo Kameras

Visual SLAM kann mit nur einer monokularen Kamera durchgeführt werden. Dies ist die günstigste und kleinste mögliche Sensoranordnung. Das Problem dabei ist jedoch, dass die Tiefeninformationen nicht von einer Kamera feststellbar ist. Dadurch sind der Maßstab der erstellten Map, sowie die geschätzte Trajektorie unbekannt. Weiterhin benötigt das System Mehrbildansichten oder Filtertechniken um die Generierung der initialen Map zu ermöglichen, da diese nicht vom ersten Frame an trianguliert werden kann. Weiterhin ist monokulares SLAM einem Drift ausgesetzt, der regelmäßig korrigiert werden muss und kann bei reiner Rotation in der Erkundung der Umwelt versagen. Durch den Einsatz von Stereo- oder RGB-D-Kameras können diese Probleme gelöst werden, was eine zuverlässige Lösung für SLAM bietet (vgl. [68] S.1).

4.7.1 Stereo SLAM

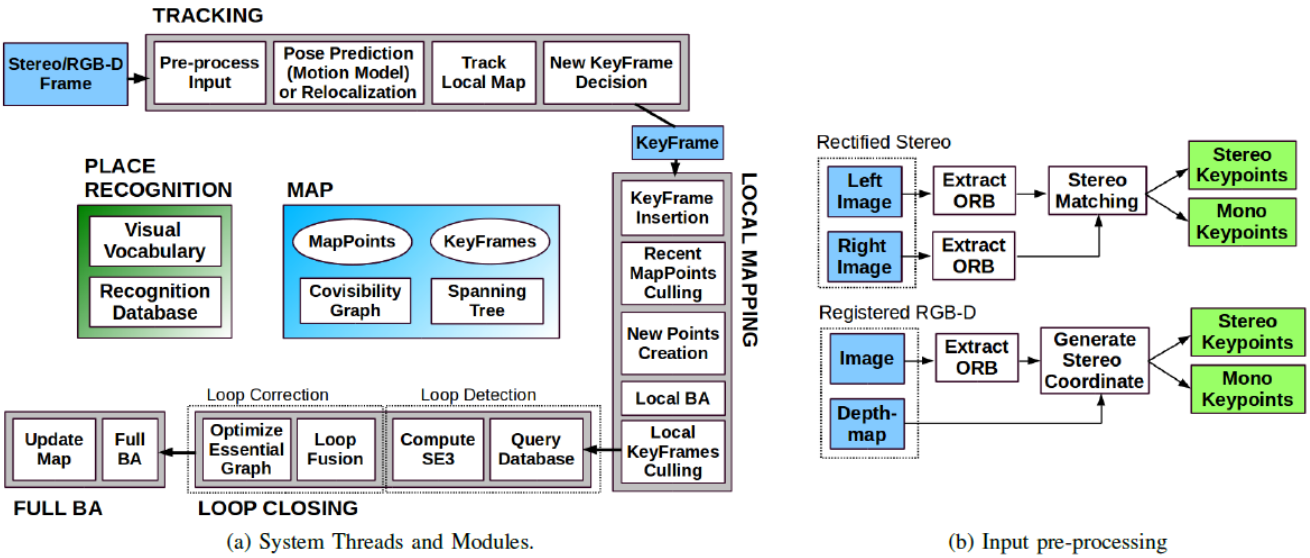
Die meisten modernen Stereo SLAM Systeme sind Keyframe basiert und führen die Optimierung in einem lokalen Bereich mithilfe des Bündelblockausgleichs durch, um Stabilität zu erreichen. Beispiele für Stereo SLAM sind etwa „Conditionally Independet Divide and Conquer EKF-SLAM“ (Paz et al. 2008), „RSLAM“ (Mei et al. 2011), „Stereo parallel tracking and mapping (S-PTAM)“ (Pire et al. 2015) oder „Large-scale direct SLAM (LSD-SLAM)“ (Engel et al. 2015) (vgl. [68] S.2).

4.7.2 RGB-D SLAM

Mithilfe von RGB-D Kameras, die neben den Farbinformationen noch einen extra Kanal für die Tiefeninformationen des Bild bereitstellen, haben sich verschiedenste SLAM Ansätze entwickelt. Dies hat sich jedoch erst nach dem Aufkommen von RGB-D Kamerasystemen ergeben. Die ersten Ansätze von RGB-D SLAM, basierten auf der Entwicklung der Kinect von Microsoft. Das System hieß KinectFusion und wurde von Newcombe et al. (2011) eingeführt. Bei diesem Verfahren wurden alle Tiefeninformationen zu einem volumetrisch ditchten Modell verschmolzen, mit dem dann die Kameraposition mittels ICP (Iterative Closest Point) verfolgt wird. Dieses System war aufgrund seiner volumetrischen Repräsentation auf kleine Arbeitsbereiche beschränkt, was auch durch das Fehlen von Loop Closing bedingt wurde. Weitere Beispiele für RGB-D SLAM Verfahren sind etwa „Kintinous“ (Whelan et al. 2015), das Open Source RGB-D SLAM (Endres et al. 2014), „DVO-SLAM“ (Kerl et al. 2013) oder „ElasticFusion“ (Whelan et al. 2016) (vgl. [68] S.2-3).

4.7.3 ORB-SLAM2

ORB-SLAM2 ist die Erweiterung von ORB-SLAM für Stereo oder RGB-D Kameras und verarbeitet die merkmalsbasierten Eingaben vor, um Merkmale an bestimmten Keypoints zu finden (siehe Abbildung 4.6. (b)).



(a) System Threads and Modules. (b) Input pre-processing

Abbildung 4.6: (a) Gesamtaufbau der drei Verarbeitungsthreads in ORB-SLAM2 (b) Vorverarbeitung des Stereo oder RGB-D Kamerabildes. Bildquelle [68]

Die Eingabebilder werden nach der Vorverarbeitung verworfen und alle Operationen des Systems basieren nur noch auf den extrahierten Features, sodass das System unabhängig von Stereo oder RGB-D Sensor arbeitet. ORB-SLAM2 kann aber genauso monokularen Input verarbeiten. Anders als die in Kapitel 4.7.2. genannten Methoden arbeitet ORB-SLAM2 mit dem Bündelblockausgleich und baut eine globale Rekonstruktion auf. Das Ziel ist es eine stabile, schnelle Methode zu liefern, die auf den meisten Standart CPUs arbeiten kann und die statt einer detailreichen dichten Rekonstruktion eher auf langfristige und konsistente Lokalisierung abzielt (vgl. [68] S.2-3).

5 Vergleich von Photogrammetrie und SLAM

Es ist inzwischen allgemein bekannt, dass Photogrammetrie und geometrische Computer Vision, in dessen Bereich SLAM einzuordnen ist, zwei eng zusammenhängende Disziplinen sind. Sie haben viele ähnliche Aufgabenstellungen und Ziele, wie Kalibrierung, Orientierung und Rekonstruktion. Viele Arbeiten und Forschungen beziehen sich auf beide Gebiete, wie relative Orientierung (Philip, 1996; Nistèr, 2004), die räumliche Analyse von Einzelbildern (Masry, 1981; Lepetit et al., 2009), Feature Erkennung (Förstner & Gülch, 1986; Lowe, 2004) oder etwa der Bündelblockausgleich (Triggs et al., 2000). Dabei sollte beachtet werden, dass viele dieser Probleme erst in der Photogrammetrie untersucht und beschrieben worden sind und erst später in der Computer Vision signifikant weiter entwickelt wurden. Dies hat die Kommunikation zwischen beiden Fachbereichen gefördert (vgl. [43] S.93).

5.1 Ähnlichkeiten und Unterschiede zwischen Visual SLAM, Photogrammetrie und SfM

In den Kapiteln 3 und 4 wurde ein Überblick über die Funktionsweise von Photogrammetrie und Computer Vision, mit Fokus auf SLAM, gegeben. Um einen Gesamtüberblick über diese Themenbereiche geben zu können, sowie die Abgrenzung der Bereiche zu ermöglichen, werden nun die Ähnlichkeiten und Unterschiede der Technologien beschrieben. Der Zusammenhang der Inhalte von Photogrammetrie und SLAM, bezieht sich hauptsächlich auf die Theorie und Anwendung der Zentralprojektion. Die Gemeinsamkeiten der beiden Felder sind Kamera Kalibrierung, Positionsbestimmung der Kamera, Feature Erkennung und Matching, sowie Modellerstellung der Umwelt. Doch warum ist SLAM ein Teil von Computer Vision und nicht nur ein Teilbereich der Photogrammetrie?

Obwohl bemerkenswerte Entwicklungen in beiden Bereichen gemacht wurden, ist das Verhältnis zwischen den beiden Disziplinen noch sehr distinkt. Die Unterscheidung kann auf die unterschiedlichen Traditionen, Philosophien und Anwendungsbereiche zurückgeführt werden. In der Photogrammetrie, die ihren Ursprung in der Vermessung und Kartierung hat, ist Genauigkeit und Präzision das Hauptaugenmerk und die meisten photogrammetrischen Arbeiten sind im Bereich des Post-Processing anzusiedeln. Für Computer Vision hingegen ist die Genauigkeit eher zweitrangig und wird meist nur durch die Anwendung definiert. Die Verarbeitungsgeschwindigkeit stellt in diesem Bereich jedoch den kritischen Faktor dar, da viele Anwendungen in Echtzeit ablaufen müssen, wie beispielsweise Objekterkennung, Roboternavigation oder Positionsbestimmung im Raum. Auch wenn photogrammetrische Verfahren immer mehr in Echtzeitanwendungen beteiligt sind, sind die beiden Ansätze grundverschieden. Weiterhin ist die Geometrie nach wie vor ein Hauptanliegen der Photogrammetrie, während in der Computer Vision maschinelles Lernen und Erkennung im Vordergrund steht. Aus mathematischer Sicht sind Photogrammetrie und Computer Vision zwei verschiedene Repräsentationen und Lösungen der Kamerageometrie (vgl. [43] S.93-94).

Doch wie kann man die Gebiete nun voneinander abgrenzen? Alle diese Verfahren beschreiben eigentlich unterschiedliche Ansätze für das gleiche Problem: die gleichzeitige Lokalisierung eines Sensors in Bezug auf seine Umgebung bei gleichzeitiger Erstellung einer 3D-Karte dieser Umgebung. Das einzigartige Merkmal von monokularem **visuellem SLAM** ist nicht, dass die Kameraposition und Szenenstruktur wiederhergestellt werden, sondern dass der Prozess simultan, rekursiv und in Echtzeit durchgeführt wird. Dies impliziert dynamisches Abtasten, wobei die Karte so aufgebaut sein muss, dass sie im laufenden Prozess die Positionsbestimmung von neu aufgenommenen Bildern integrieren kann, sowie die Triangulation neuer Punkte unterstützt. SLAM Konzepte sind deswegen aus den Bereichen wie der Selbsterkundung von Robotern, der automatischen Fahrzeugnavigation oder im nicht fotografischen geometrischen Kontext, wie etwa dem Handlaser Scanning entstanden und weiterentwickelt worden. Wenn der Echtzeitaspekt von monokularem visuellem SLAM, was auch als visuelle Odometrie bezeichnet werden kann, wegfällt, bleiben SfM und Photogrammerie übrig. **Structure from Motion** wird als eine der herausragendsten Leistungen der Computer Vision bezeichnet. Es muss jedoch bedacht werden, dass die Entwicklung dieser Feature basierten Matching- und Lokaliserungstechnik von 3D Punkten im Raum wenig mit der Bildbasierten 3D-Messung, das heißt der Photogrammetrie zu tun hatte, sondern mehr mit der räumlichen Archivierung von ungeordneten Sammlungen von Bildern, oft von unbekannten Kameras.

Bei SfM wird hauptsächlich die Kameralokalisierung und nicht die Erzeugung von Punktwolken betont und es wurde die Notwendigkeit der Kamerakalibrierung erkannt. Die technischen Prinzipien, welche die Grundlagen der **Photogrammetrie** bilden, sind bei der Entwicklung von Structure from Motion nicht berücksichtigt. Dazu zählen zum Beispiel geometrisches Netzwerkdesign, metrische Kamerabetrachtung, Genauigkeitsoptimierung und Varianzausbreitung, systematische Fehlerkompensation und grobe Fehlererkennung, ortsunabhängige Kamerakalibrierung und die Einführung von Beobachtungsredundanz zur Erhöhung der Zuverlässigkeit und Qualitätskontrollverfahren in allen Phasen der photogrammetrischen Datenverarbeitungspipeline, die alle durch strenge, unveränderlich nichtlineare mathematische Modelle gestützt werden (vgl. [46] S.1-2).

C. Fraser beschreibt den Unterschied zwischen Structure from Motion und Photogrammetrie folgendermaßen [46]:

„It could be well argued that a data processing pipeline that is initiated with SfM-based camera localisation, but then follows the principles of photogrammetry, is no-longer SfM per se, but indeed standard photogrammetry. To call a photogrammetry measurement an SfM solution not only implies that the solution is less 'metric' (i.e. accurate and reliable) than may be the case, but also attributes more to the SfM process than what is actually there.“

Structure from Motion ist also ein leistungsstarker Ansatz zur Lösung des Lokalisationsproblems, ausgehend von einer unbekannten Umgebung. Die neu gewonnenen Erkenntnisse hatten dann in der Photogrammetrie wenig Einfluss, da diese schon seit langem ihre eigenen präzisen Algorithmen zur Sensorausrichtung hat. Die Bezeichnungen SLAM, SfM und Photogrammetrie haben also großteils unterschiedliche Bedeutungen, auch wenn es große konzeptionelle und algorithmische Überschneidungen gibt. Weder SLAM noch SfM umfassen den gesamten Prozess der metrischen bildbasierten 3D-Messung, Objektrekonstruktion und Kartierung, welche heute die automatisierte Photogrammetrie auszeichnet (vgl. [46] S.2).

6 Implementation einer AR Anwendung für Android

Im Rahmen des Praxisteils dieser Arbeit ist eine Applikation für Android basierte Smartphones erstellt worden. In diesem Abschnitt werden die verwendeten Tools und Frameworks, um die Augmented Reality Anwendung zu erstellen, beschrieben. Das im folgenden vorgestellte Framework ARCore, implementiert den Simultaneous Location and Mapping Ansatz und ist als einzige AR-Programmierschnittstelle Open Source (Siehe Tabelle S.29).

6.1 ARCore

ARCore, das von Google entwickelt wird und das seit März 2018 veröffentlicht wurde, ist der Nachfolger des ebenfalls von Google ins Leben gerufene Projekt Tango, welches jedoch spezielle „Time-of-flight“ (TOF) Kameras benötigt, um Distanzen im Raum mithilfe des Lichtlaufzeitverfahrens zu messen. Dieser Ansatz hat sich wegen der fehlenden technischen Voraussetzungen der meisten Android Geräte nicht durchgesetzt. Da Google mit dem von Apple veröffentlichen Framework ARKit gleichziehen wollte, wurde Projekt Tango beendet und durch ARCore ersetzt. ARCore verwendet drei Schlüsselfunktionen um virtuelle Objekte in die reale Welt zu integrieren (vgl. [51]):

- **Motion Tracking** ermöglicht es dem Smartphone seine Position relativ zur Umgebung zu verfolgen und zu verstehen.
- **Environmental understanding** ermöglicht es dem Smartphone die Größe und Lage aller Arten von Oberflächen zu erfassen, dazu zählen horizontale, vertikale oder schräge Oberflächen.
- **Light estimation** ermöglicht die Abschätzung der aktuellen Lichtverhältnisse in der Umgebung, zur korrekten Beleuchtung und zur Erstellung von Schatten.

6.1.1 Unterstütze Geräte

Um eine gute und flüssige Erfahrung für den Endnutzer zu gewährleisten, müssen unterstützte Smartphones bestimmte Kriterien erfüllen. Dazu zählen die Qualität der verbauten Kamera, der Bewegungssensoren und der Designarchitektur. Weiterhin muss das Gerät über eine leistungsstarke CPU verfügen, die sich in das Hardware-Design integriert, um gute Leistung und effektive Echtzeitberechnung zu gewährleisten. Die Voraussetzungen um ARCore zu verwenden, werden wie folgt definiert:

- **Android 7.0** oder höher (Einige Modelle benötigen neuere Versionen).
- Ein Gerät, das ursprünglich mit dem **Google Play Store** ausgeliefert wurde.
- **Internetzugang**, um Google Play Services für AR zu installieren oder zu aktualisieren.

Aktuell werden 144 Android Smartphones unterstützt, die mit dem Google Play Store ausgeliefert werden. Weiterhin werden 29 chinesische Modelle unterstützt, bei welchen die Google Play Services manuell über die App Stores der Hersteller installiert werden müssen. Auch 20 iOS Smartphone Modelle können ARCore verwenden, dabei benötigt ARCore jedoch ARKit kompatible Geräte mit iOS 11.0 oder höher (Stand 22.08.2019, vgl. [52]).

Die aktuell unterstützten Geräte sind die Flaggschiffe der größen Smartphonehersteller seit ungefähr 2017. Google zielt mit ARCore also auf Leistungsstarke Hardware ab, um eine stabile Performance zu ermöglichen, versucht aber gleichzeitig so viele Endnutzer wie möglich zu erreichen.

6.1.2 Grundlegende Konzepte

In diesem Abschnitt werden grundlegende Schlüsselfunktionen von ARCore, wie in Kapitel 6.1 bereits eingeführt, genauer erläutert, sowie ein Einblick in die Pipeline hinter ARCore gegeben.

Motion Tracking wird in ARCore mit einem Prozess namens „Concurrent Odometry and Mapping“ (COM) ermöglicht. Dabei werden Features im Kamerabild erkannt und zur Bestimmung der Änderung der Position des Smartphones verwendet. Diese visuellen Informationen werden mit Trägheitsmessungen der internen Messeinheiten (IMU) des Smartphones kombiniert, um die Position und Ausrichtung der Kamera relativ zur Umwelt zu bestimmen.

Google hat sich das System und die Methodik hinter „Concurrent Odometry and Mapping“ patentieren lassen. Die Beschreibung des Patents und des Systems hinter COM zeigt viele Zusammenhänge des implementierten Systems in ARCore mit SLAM und photogrammetrischen Verfahren (vgl. [56]):

„An electronic device tracks its motion in an environment while building a three-dimensional visual representation of the environment that is used to correct drift in the tracked motion. A motion tracking module estimates poses of the electronic device based on feature descriptors corresponding to the visual appearance of spatial features of objects in the environment. A mapping module builds a three-dimensional visual representation of the environment based on a stored plurality of maps, and feature descriptors and estimated device poses received from the motion tracking module. The mapping module provides the three-dimensional visual representation of the environment to a localization module, which identifies correspondences between stored and observed feature descriptors. The localization module performs a loop closure by minimizing the discrepancies between matching feature descriptors to compute a localized pose. The localized pose corrects drift in the estimated pose generated by the motion tracking module.“

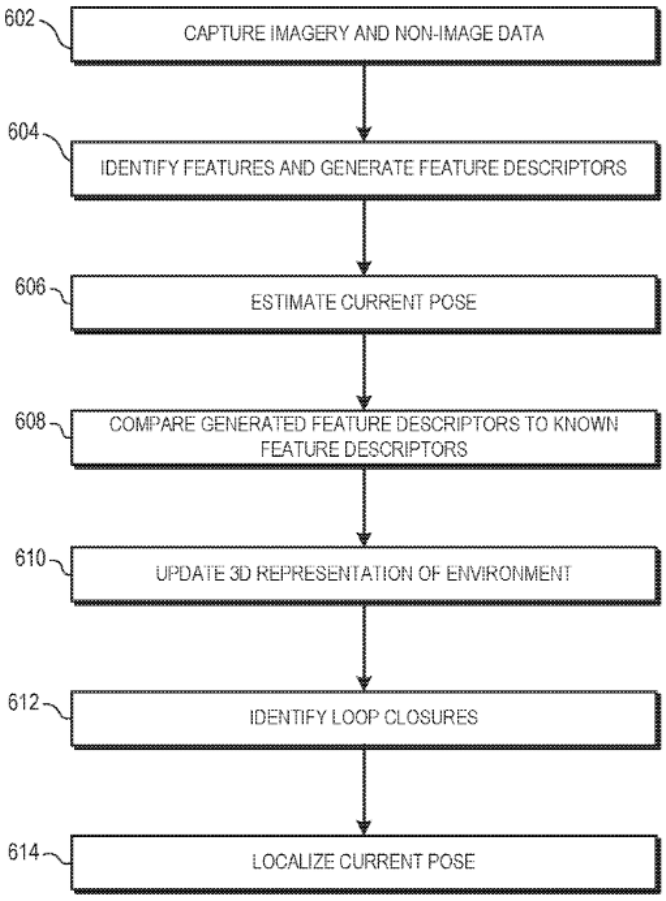


Abbildung 6.1: Pipeline von COM. Bildquelle [56]

Auch die Abbildungen über die Pipeline von COM im Patent zeigen eine hohe Übereinstimmung mit der photogrammetrischen Pipeline, sowie gegenüber SLAM (siehe Abbildung 6.1.).

Concurrent Odometry and Mapping kann als Implementation des SLAM Verfahrens von Google interpretiert werden. Der Begriff Odometrie, der aus der Messung der realtiven Positionierung von Fahrzeugen anhand deren Eigenbewegung kommt (vgl. [61] S.7-9), wurde für Smartphones übernommen. Dies liegt wohl hauptsächlich in der dadurch möglichen Abgrenzung zu anderen Verfahren, für die einfachere Patentierung der Methode von Google.

Environmental Understanding wird in ARCore durch die Erkennung von Clustern von Feature Punkten auf üblichen horizontalen oder vertikalen Flächen ermöglicht. Weiterhin können die Grenzen dieser Ebenen bestimmt werden. Da ARCore zur Erkennung der Ebenen Features verwendet, werden flache oder texturlose Oberflächen, wie beispielsweise eine weiße Wand möglicherweise nicht richtig erkannt.

Light Estimation wird in ARCore verwendet, um Informationen über die Beleuchtung der Umgebung zu erhalten. Dazu werden Daten wie durchschnittliche Intensität der Helligkeit oder Farbtemperatur erfasst. Anhand dieser Informationen kann mit einer Helligkeitsanpassung oder Farbkorrektur das Gefühl von Realismus in der Szene verstärkt werden, indem virtuelle Objekte unter den gleichen Bedingungen wie die echte Welt beleuchtet werden. Weiterhin kann ARCore bestimmen aus welcher Richtung natürliches oder künstliches Licht kommt. Anhand dieser Informationen werden von ARCore Schatten berechnet und in der Szene gerendert.

User Interaction wird in ARCore mit Hilfe von „Hit-Testing“ ermöglicht. Eine (x, y) Koordinate auf dem Bildschirm des Smartphones wird mithilfe eines Strahls (Ray) in das Kamerabild projiziert. Alle Schnittpunkte mit Ebenen oder Merkmalspunkten des Strahls werden zusammen mit der Pose dieses Schnittpunktes zurückgegeben. Dadurch können Nutzer mit virtuellen Objekten interagieren. Es über die Touchsteuerung möglich die virtuellen Objekte zu verschieben und in ihrer Größe zu transformieren.

Anchors und Trackables (Anker und trackbare Objekte) werden verwendet, da sich die Positionen von Flächen verändern kann, wenn ARCore im Lokalisierungsprozess das Verständnis für die eigene Position und das Umfeld verbessert. Um ein virtuelles Objekt zu platzieren, muss ein Anker erstellt werden, um sicherzustellen, dass ARCore die Position dieses Objekts über die Zeit verfolgt. Ebenen und Punkte sind hier eine spezielle Art von Objekt, das als „Trackable“ bezeichnet wird.

Virtuelle Ojekte können an bestimmten Trackables verankert werden, um sicherzustellen, dass die Beziehung zwischen virtuellen Objekt und dem zu verfolgenden Punkt oder der zu verfolgenden Ebene stabil bleibt, auch wenn sich das Gerät bewegt. Das heißt, dass virtuelle Objekte, die beispielsweise auf einem gewissen Punkt am Boden platziert werden, immer noch auf exakt der gleichen Position bleiben, auch wenn die Ebene, die den Boden repräsentiert, durch ARCore angepasst und verschoben wird.

Augmented Images ist eine Funktion, mit der Augmented Reality Anwendungen erstellt werden können, die auf bestimmte 2D-Bilder, wie beispielsweise Produktverpackungen oder Filmposter reagieren können. Diese registrierten Bilder funktionieren nach dem Prinzip von Markern. Deshalb ist es wichtig, dass genügend Merkmale im Bild vorhanden sind.

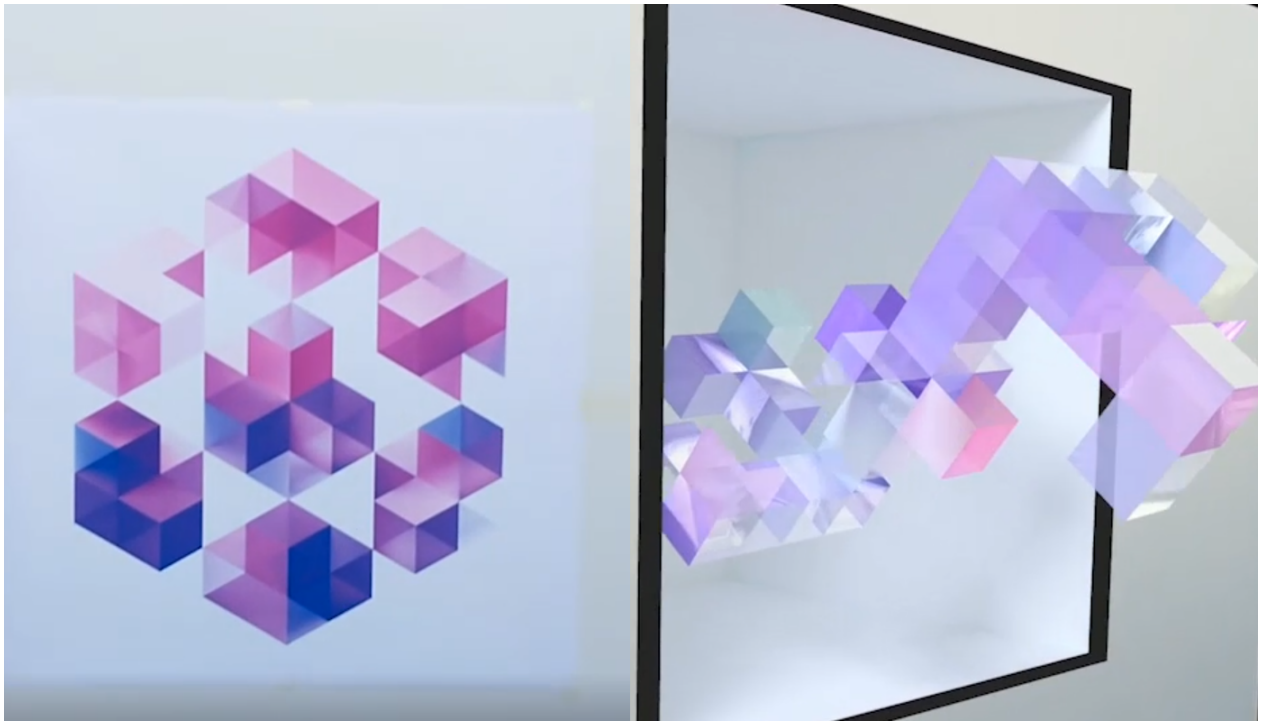


Abbildung 6.2: Augmented Image mit ARCore, Bildquelle [53]

ARCore bietet auch eine Funktion um sich zu bewegende Bilder zu verfolgen, wie beispielsweise eine Plakatwand, auf der Seite eines fahrenden Busses. Die Bilder können offline zusammengestellt werden, um eine Bilddatenbank zu erstellen, es ist jedoch auch möglich einzelne Bilder in Echtzeit vom Gerät hinzuzufügen. Nach der Registrierung erkannt ARCore diese Bilder, sowie die Grenzen dieser und gibt eine entsprechende Pose eines virtuellen Objekts zurück. Das Bild wird von einer 2D Ansicht auf eine virtuelle 3D-Ansicht augmentiert.

Mithilfe der **ARCore Cloud Anchor API** können kollaborative AR Multiplayer Anwendungen erstellt werden. Dazu wird von einem Gerät der Anker eines virtuellen Objekts, sowie die Feature Points der näheren Umgebung, um dieses Objekt, in der Cloud gespeichert. Diese Anker können dann mit anderen Benutzern auf Android- oder iOS-Geräten in der selben Umgebung geteilt werden. Mit dieser Methode können zwei Endgeräte die gleiche virtuellen 3D-Objekte an exakt der gleichen Stelle im Raum rendern, so dass beide Benutzer das gleiche virtuelle Erlebnis teilen (vgl. [54]).

6.1.3 Entwicklungsumgebungen

ARCore kann mit vielen gängigen Entwicklungsumgebungen verwendet werden. Dazu zählen Android Studio, Android Native Development Kit, Unity für Android, Unity für iOS, Unreal Engine und iOS (vgl. [55]). Im Rahmen dieser Arbeit wurde Android Studio in der Version 3.4.1, sowie ARCore in der Version 1.11.0 verwendet.

6.2 Sceneform

Sceneform ist eine Framework, das von Google entwickelt wurde und welches nahtlos in ARCore integriert werden kann. Es ermöglicht das einfache Rendern von realistischen 3D-Szenen in AR- und Nicht-AR-Anwendungen, ohne bei der Programmierung auf OpenGL zurückgreifen zu müssen. Sceneform bietet folgende Features (vgl. [57]):

- Einen „high-level“ **Szenengraph API**.
- Einen realistischen **physikalischen Renderer** namens **Filament**.
- Ein **Android Studio-Plugin** zum Importieren, Anzeigen und Erstellen von **3D-Assets**.

Sceneform benötigt Android Studio Version 3.1 oder höher und wird als Plugin installiert. Es liefert ein sogenanntes „ArFragment“ welches automatisch das ARCore Session Management übernimmt, sowie die notwendigen ARCore Laufzeitprüfungen durchführt. Dazu gehört die automatische Überprüfung von kompatiblen Versionen der Google Play Services für Augmented Reality und die Überprüfung der Berechtigung für Kamera und anderer Sensoren. Sind alle Überprüfungen erfolgreich, erstellt das ArFragment eine „ArSceneView“ und eine ARCore Session. Mithilfe der ArSceneView kann nun das Kamerabild gerendert werden, sowie mithilfe eines „PlaneRenderers“ die erkannten Ebenen zur Platzierung von virtuellen Objekten visualisert werden.

Der ArSceneView ist eine Szene zugeordnet, welche eine baumartige Datenstruktur beinhaltet. Die Nodes dieser Datenstruktur sind die zu rendernden virtuellen Objekte. Jedes Node enthält alle Informationen, um es zu rendern und damit zu interagieren. Dazu gehören Position, Ausrichtung, Modell, Kollisionsform und Event Listener. Nodes können mit anderen Nodes Eltern-Kind-Beziehungen formen, welche sich in einer baumartigen Struktur kristallisieren. Diese Struktur wird dann als Szenengraph bezeichnet. Bei jedem Einzelbild rendert Sceneform den Szenengraph aus Sicht der Kamera, welche durch das ARCore Tracking geführt wird (vgl. [58]).

Filament ist eine physikalisch basierte Rendering (PBR) Engine für Android, die ihren Fokus auf Echtzeit Performance bei mobilen Geräten legt. Das Hauptziel von Filament sind GPUs der Klasse OpelGL ES 3.x. Die Hauptgesichtspunkte der Engine liegen in den Bereichen Qualität, Benutzerfreundlichkeit, Vertrautheit, Flexibilität und in der Größe der bereitgestellten Renderbibliothek. Physikalisch basiertes Rendering ist ein Verfahren, das eine genauere Darstellung von Materialien und deren Wechselwirkungen mit Licht im Vergleich zu herkömmlichen Echtzeitmodellen ermöglicht. Die Trennung von Materialien und Beleuchtung im Kern der PBR Methode macht es einfacher, realistische Objekte zu erstellen, die unter allen Lichtverhältnissen präsize aussehen (vgl [59]).

In der Implementierung der App wird Sceneform in der Version 1.11.0 verwendet.

6.3 Google Places API

Im Rahmen dieser Arbeit wird die Google Places API verwendet. Diese ermöglicht die Abfrage von Informationen mithilfe des aktuellen Standort des Smartphones. Die Places API ist ein Dienst, der Informationen über bestimmte Orte als Antwort auf eine HTTP-Anfrage zurückgibt. Places sind innerhalb der API als Einrichtungen, geografische Standorte oder berühmte Points of Interest definiert. Die Informationen die mit der API abgefragt werden können, sind etwa einfache Listen der näheren Orte, Details über die Orte in der Nähe oder Fotos der näheren Orte, bezogen aus der Datenbank von Google. Die API ermöglicht auch die Autovervollständigung von Suchanfragen bestimmter Orte oder Querys.

Jeder dieser Dienste wird als HTTP-Request aufgerufen und gibt entweder XML- oder JSON-Dateien als Antwort zurück. Alle Anfragen müssen das HTTPS Protkoll verwenden und benötigen einen eigenen API Schlüssel (vgl. [60]).

Bei der Implementation der App wurde die „Place Search“ Funktion der API verwendet. Ein Aufruf an die API schaut beispielsweise wie folgt aus:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=
 + LAT_LNG + &rankby=distance&keyword= + KEY_WORD + &key= + API_KEY;
```

Das Ergebnis dieser Anfrage erhält eine Vielzahl an Informationen, die anschließend weiter verarbeitet werden können. In dieser Implementierung wurden die Parameter **location**, **name** und **rating** verwendet, die aus der JSON Antwort der API entnommen werden.


```
{
   "geometry" : {
      "location" : {
         "lat" : 52.5333868,
         "lng" : 13.5175705
      },
      "viewport" : {
         "northeast" : {
            "lat" : 52.53474927989272,
            "lng" : 13.51911097989272
         },
         "southwest" : {
            "lat" : 52.53204962010727,
            "lng" : 13.51641132010728
         }
      }
   },
   "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/lodging-71.png",
   "id" : "c84d8cdaa6eb89b025b27bb472ec5d65a3c5eded",
   "name" : "Comfort Hotel Lichtenberg",
   "opening_hours" : {
      "open_now" : true
   },
   "photos" : [
      {
         "height" : 2724,
         "html_attributions" : [
            "\u003ca href=\"https://maps.google.com/maps/contrib/101958906858002
         ],
         "photo_reference" : "CmRaAAAAk0u_P0IdUs6bW0g781_ZwIRprywWchaq65Yaf56kZK
         "width" : 3936
      }
   ],
   "place_id" : "ChIJc_MmF69OqEcR_e4eA0_y5oU",
   "plus_code" : {
      "compound_code" : "GGM9+92 Berlin",
      "global_code" : "9F4MGGM9+92"
   },
   "rating" : 3.8,
   "reference" : "ChIJc_MmF69OqEcR_e4eA0_y5oU",
   "scope" : "GOOGLE",
   "types" : [ "lodging", "point_of_interest", "establishment" ],
   "user_ratings_total" : 918,
   "vicinity" : "Rhinstraße 159, Berlin"
}
```


Abbildung 6.3: Ein Ergebnis des Resultats einer Places Search mit der Google Places API

6.4 Dexter

Dexter ist eine Open Source Android-Bibliothek der Firma Karumi, die den Prozess der Anfrage von Berechtigungen während der Laufzeit einer App vereinfacht. Seit Android Version Marshmallow (6.0), welche im Oktober 2015 veröffentlicht wurde, werden App-Berechtigungen nicht mehr bei der Installation, sondern beim Zeitpunkt der Nutzung abgefragt. Weiterhin fragt die App nur nach einer bestimmten Erlaubnis für ein Feature, wenn dieses verwendet wird (vgl. [63]). Dieser neue Ansatz gibt dem Endnutzer mehr Kontrolle über die Berechtigungen seiner Anwendungen, erfordert jedoch auch die Implementation dieser Features durch den Entwickler.

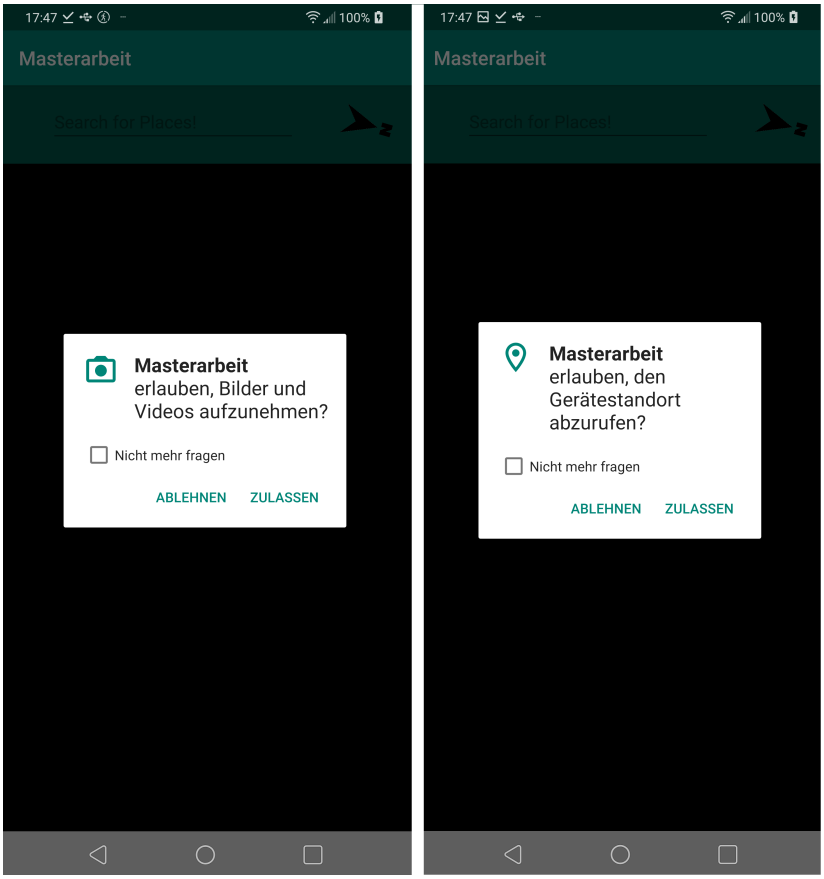


Abbildung 6.4: Abfrage der Berechtigungen mit Dexter

Die offizielle API zum Abfragen der Berechtigungen ist tief mit der Android „Activity“ Klasse verbunden. Mit Dexter kann man die App Berechtigungen auch ohne Activities und zu einem beliebigen Zeitpunkt abfragen. Weiterhin kann die Logik überall implementiert werden (vgl. [62]). Mit Dexter werden in der hier implementierten Applikation die Berechtigungen für Kamera und Standort, beim ersten Starten der Anwendung, abgefragt.

6.5 Konzept

Die grundlegende Idee hinter der Anwendung ist es, den Nutzer nach beliebigen Informationen - in Bezug auf seinen aktuellen Standort - suchen zu lassen und die Resultate dann in AR anzuzeigen. So kann sich der Nutzer beispielsweise Informationen über die näheren Hotels in der Umgebung anzeigen lassen. Dann werden ihm, nachdem er seine Suche durchgeführt und die virtuellen Modelle platziert hat, die Entfernung und die Himmelsrichtung der drei nähsten Hotels angezeigt.

6.6 Anwendungsbeispiel

Step by Step Tutorial

6.7 Benchmarks

Zeit bis die Flächen gefunden wurden in Abhängigkeit der Lichtverhältnisse

Performance (vertices to fps) (kleinen Test bauen mit großem model)

Anfragegeschwindigkeit für HTTP Requests an Google

6.8 Probleme

Die ursprüngliche Idee, die virtuellen Infoboxen richtig zur Himmelsrichtung ausgerichtet in der Augmented Reality zu platzieren, ist wegen mehrerer Probleme nicht realisierbar.

Problem 1: Die Orientierung des von ARCore benutzen globalen Koordinatensystems ist bei jeder Session verschieden. Nur die Y-Achse ist immer nach oben orientiert, was an der Erkennung der horizontalen Flächen liegt. Die globale Orientierung der X- und Z-Achse ist jedoch immer unterschiedlich (vgl. [67]).

Problem 2: Versuche das globale Koordinatensystem anhand der internen Sensoren, wie Magnetometer und Lagesensor auszurichten, hat wegen starken Ungenauigkeit dieser Sensoren zu oft zu falschen Ergebnissen geführt. Dies hat zur Folge, dass bei mehreren Initialisierungen des Koordinatenraums bei gleichen Startbedingungen teils

sehr verschiedene Resultate und damit Drehungen des Koordinatensystems in die falsche Richtung zur Folge hatte. Auch nach der Kalibrierung der internen Sensoren war das Ergebnis eher ungenau.

https://stackoverflow.com/questions/54356589/what-sensors-does-arcore-use

7 Zusammenfassung und Ausblick

Fortschritte in der Photogrammetrie sind viel zu sehr mit den Fortschritten der Computer Vision verflochten, als dass sich die Konvergenz der beiden Disziplinen umkehren wird. Photogrammetrie und Computer Vision haben die Extraktion und Rekonstruktion von Daten aus Bildmaterial zu unterschiedlichen Zeiten und mit unterschiedlichen Zielen begonnen. Als sich dann 3D-Modelle als Referenzziel herausstellten, wurde der Austausch von Ansätzen und Techniken zwischen den Disziplinen vorangetrieben (vgl. [26] S.9). Dies hat dazu geführt, dass Photogrammetrie und Computer Vision verfahrenstechnisch kaum mehr unterscheidbar sind.

Literaturverzeichnis

[1] Heipke, C. (2017), „Photogrammetrie und Fernerkundung “, 1.Auflage, Berlin: Springer Verlag, S. 5-7.

[2] Hugh Durrant-Whyte, Tim Bailey (2006), Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. URL: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf (Zuletzt abgerufen am 09.07.2019)

[3] P. Martin, E. Marchand, P. Houlier, I. Marchal. Mapping and re-localization for mobile augmented reality. IEEE Int. Conf. on Image Processing, Oct 2014, Paris, France. URL: https://hal.inria.fr/hal-00994756/document (Zuletzt abgerufen am 09.07.2019)

[4] Joan Sol'a, (2014), Simulataneous localization and mapping with the extended Kalman filter. URL: http://www.iri.upc.edu/people/jsola/JoanSola/objectes/curs_SLAM/SLAM2D/SLAM%20course.pdf (Zuletzt aufgerufen am 10.07.2019)

[5] Mohinder S. Grewal, Angus P. Andrews (2001), Kalman Filtering: Theory and Practice with MATLAB. URL: http://staff.ulsu.ru/semoushin/_index/_pilocus/_gist/docs/mycourseware/13-stochmod/2-reading/grewal.pdf (Zuletzt aufgerufen am 10.07.2019)

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. URL: http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf (Zuletzt aufgerufen am 11.07.2019)

[7] G. Grisetti, G.D. Tipaldi, C. Stachniss, W. Burgard, D. Nardi (2007). Fast and accurate SLAM with Rao-Blackwellized particle filters. URL: http://srl.informatik.uni-freiburg.de/publicationsdir/grisettiRAS07.pdf (Zuletzt aufgerufen am 11.07.2019)

[8] M. Mengelkoch (2007). Implementieren des FastSLAM Algorithmus zur Kar-

tenerstellung in Echtzeit. URL: https://kola.opus.hbz-nrw.de/opus45-kola/frontdoor/deliver/index/docId/183/file/sa-00.pdf (Zuletzt aufgerufen am 11.07.2019)

[9] J. Fuentes-Pacheco, J. Ruiz-Ascencio, J.M. Rendón-Mancha (2012). Visual simultaneous localization and mapping: a survey. In: J.M. Artif Intell Rev (2015) 43: 55. Springer Netherlands. DOI: https://doi.org/10.1007/s10462-012-9365-8

[10] J. Halvarsson, (2018), Using SLAM-based technology to improve directional navigation in an Augmented Reality game. URL: http://umu.diva-portal.org/smash/get/diva2:1245293/FULLTEXT01.pdf (Zuletzt aufgerufen am 11.07.2019)

[11] P. Milgram, H. Takemura, A. Utsumi, F. Kishino (1994), Augmented Reality: A class of displays on the reality-virtuality continuum. URL: http://etclab.mie.utoronto.ca/publication/1994/Milgram_Takemura_SPIE1994.pdf (Zuletzt aufgerufen am 16.07.2019)

[12] A. Hanafi, L. Elaachak, M. Bouhorma (2019), A comparative Study of Augmented Reality SDKs to Develop an Educational Application in Chemical Field. DOI: 10.1145/3320326.3320386

[13] D. Amin, S. Govilkar (2015), Comparative Study of Augmented Reality SDK's. International Journal on Computational Sciences & Applications (IJCSA) Vol.5, No.1, February 2015 URL: https://pdfs.semanticscholar.org/e752/17e8897cb46b466d6ba83e909cca4ecff8f2.pdf (Zuletzt aufgerufen am 16.07.2019)

[14] M. Lowney, A. S. Raj (2016), Model Based Tracking for Augmented Reality on Mobile Devices. URL: https://web.stanford.edu/class/ee368/Project_Autumn_1617/Reports/report_lowney_raj.pdf (Zuletzt aufgerufen am 16.07.2019)

[15] S. Ćuković, M. Gattullo, F. Pankratz, G. Devedzic, E. Carrabba, K. Baizid (2015), Marker Based vs. Natural Feature Tracking Augmented Reality Visualization of the 3D Foot Phantom. URL: https://www.researchgate.net/publication/278668320_Marker_Based_vs_Natural_Feature_Tracking_Augmented_Reality_Visualization_of_the_3D_Foot_Phantom (Zuletzt aufgerufen am 17.07.2019)

[16] Basic concepts of the homography explained with code. URL: https://docs.opencv.org/3.4.1/d9/dab/tutorial_homography.html (Zuletzt aufgerufen am 17.07.2019)

[17] M. Maidi, J.Y. Didier, F. Ababsa, M. Mallem (2008), A performance study for camera pose estimation using visual marker based tracking. URL:

https://www.academia.edu/13152554/A_performance_study_for_camera_pose_estimation_using_visual_marker_based_tracking (Zuletzt aufgerufen am 18.07.2019)

[18] A. Pinz, M. Brandner, H. Ganster, A. Kusej, P. Lang, M. Ribo (2002) Hybrid Tracking for Augmented Reality. URL: https://www.researchgate.net/publication/229025765_Hybrid_tracking_for_augmented_reality (Zuletzt aufgerufen am 18.07.2019)

[19] E. Rosten, T. Drummond (2006) Machine learning for high-speed corner detection. URL: https://www.edwardrosten.com/work/rosten_2006_machine.pdf (Zuletzt aufgerufen am 18.07.2019)

[20] K. Schindler (2014) Mathematical Foundations of Photogrammetry. URL: https://ethz.ch/content/dam/ethz/special-interest/baug/igp/photogrammetry-remote-sensing-dam/documents/pdf/math-of-photogrammetry.pdf (Zuletzt aufgerufen am 19.07.2019)

[21] A.S. Alturki, J.S. Loomias (2016) Camera Principal Point Estimation from Vanishing Points. DOI: 10.1109/NAECON.2016.7856820 (Zuletzt aufgerufen am 19.07.2019)

[22] P. Grussenmeyer, O. Al Khalil (2002) Solutions for Exterior Orientation in Photogrammetry: A Review. Photogrammetric Record, 17(100):615-634. URL: https://hal.archives-ouvertes.fr/hal-00276983/document (Zuletzt aufgerufen am 19.07.2019)

[23] K.L.A. El-Ashmawy (2015). A comparison study between collinearity condition, coplanarity condition, and direct linear transformation (DLT) method for camera exterior orientation parameters determination. Geodesy and Cartography, 41(2), 66–73. URL:https://journals.vgtu.lt/index.php/GAC/article/view/2837/2334 (Zuletzt aufgerufen am 19.07.2019)

[24] E.E. Elnima (2015) A solution for exterior and relative orientation in photogrammetry, a genetic evolution approach. Journal of King Saud University – Engineering Sciences. URL: https://core.ac.uk/download/pdf/82822280.pdf (Zuletzt aufgerufen am 22.07.2019)

[25] M. Lourakis, A. Argyros (2005) Is Levenberg-Marquardt the Most Efficient Optimization Algorithm for Implementing Bundle Adjustment? URL: https://www.ics.

forth.gr/_publications/0201-P0401-lourakis-levenberg.pdf (Zuletzt aufgerufen am 20.08.2019)

[26] G. Forlani, R. Roncella, C. Nardinocchi (2015) Where is photogrammetry heading to? State of the art and trends. Rendiconti Lincei, 26(S1), 85–96. DOI:10.1007/s12210-015-0381-x (Zuletzt aufgerufen am 24.07.2019)

[27] K. L. El-Ashmawy (2018) Using direct linear Transformation (DLT) Method for aerial Photogrammetry Applications. Geodesy and Cartography 2018 Volume 44 Issue 3: 71–79. URL: https://journals.vgtu.lt/index.php/GAC/article/download/1629/5048 (Zuletzt aufgerufen am 24.07.2019)

[28] L. Vandenberge (2018) 13. Nonlinear least squares. URL: http://www.seas.ucla.edu/~vandenbe/133A/lectures/nlls.pdf (Zuletzt aufgerufen am 20.08.2019)

[29] A. W. Gruen (1985) Adaptive least Squares Correlations: A powerful Image Matching Technique. URL: https://www.researchgate.net/publication/265292615_Adaptive_Least_Squares_Correlation_A_powerful_image_matching_technique (Zuletzt aufgerufen am 24.07.2019)

[30] S. Boyd (2016) Nonlinear Least Squares. URL: https://stanford.edu/class/ee103/lectures/nlls_slides.pdf (Zuletzt aufgerufen am 24.07.2019)

[31] S. Gratton, A. S. Lawless, N. K. Nichols (2007) Approximate Gauss–Newton Methods for Nonlinear Least Squares Problems. URL: https://www.researchgate.net/publication/220133629_Approximate_Gauss-Newton_Methods_for_Nonlinear_Least_Squares_Problems (Zuletzt aufgerufen am 30.07.2019)

[32] H. P. Gavin (2019) The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems. URL: http://people.duke.edu/~hpgavin/ce281/lm.pdf (Zuletzt aufgerufen am 30.07.2019)

[33] K. Levenberg (1944) A method for the solution of certain non-linear problems in least squares. URL: https://www.ams.org/journals/qam/1944-02-02/S0033-569X-1944-10666-0/ (Zuletzt aufgerufen am 30.07.2019)

[34] N. Börlin, P. Grussenmeyer (2013) Bundle Adjustment With and Without Damping. The Photogrammetric Record, 28(144), 396–415. DOI: 10.1111/phor.12037 (Zuletzt aufgerufen am 07.08.2019)

[35] A. Abbas, S. Ghuffar (2018) Robust Feature Matching in terrestrial Image Sequences. The International Archives of the Photogrammetry,

Remote Sensing and Spatial Information Sciences, Volume XLII-3, URL: https://www.researchgate.net/publication/324855190_ROBUST_FEATURE_MATCHING_IN_TERRESTRIAL_IMAGE_SEQUENCES (Zuletzt aufgerufen am 07.08.2019)

[36] F. Remondino (2006) Detectors and descriptors for photogrammetric applications. URL: http://3dom.fbk.eu/sites/3dom.fbk.eu/files/pdf/remondino_ISPRS_III_06.pdf (Zuletzt aufgerufen am 07.08.2019)

[37] A. Lingua, D. Marenchio, F. Nex (2009) A comparison between "old and new" feature extraction and matching techniques in Photogrammetry. URL: https://pdfs.semanticscholar.org/b9c3/067b6fc111e1cc02f42357d5fb459a642152.pdf (Zuletzt aufgerufen am 07.08.2019)

[38] Z. Qu (2018) Efficient Optimization for Robust Bundle Adjustment. URL: https://vision.in.tum.de/_media/members/demmeln/qu2018msc.pdf (Zuletzt aufgerufen am 08.08.2019)

[39] A. Bokovoy, K. Yakovlev (2017) Original Loop-Closure Detection Algorithm for Monocular vSLAM. Analysis of Images, Social Networks and Texts, 210–220. URL: https://arxiv.org/pdf/1707.04771.pdf (Zuletzt aufgerufen am 08.08.2019)

[40] M. Andersson, M. Baerveldt (2018) Simultaneous localization and mapping for cars using ORB2-SLAM. URL: http://publications.lib.chalmers.se/records/fulltext/256291/256291.pdf (Zuletzt aufgerufen am 08.08.2019)

[41] R. Mur-Artal, J. M. M, Montiel (2015) ORB-SLAM: a Versatile and Accurate Monocular SLAM System. URL: https://arxiv.org/pdf/1502.00956.pdf(Zuletzt aufgerufen am 08.08.2019)

[42] M. Calonder, V. Lepetit, C. Strecha, P. Fua (2010) BRIEF: Binary Robust Independent Elementary Features. URL: https://www.researchgate.net/publication/221304115_BRIEF_Binary_Robust_Independent_Elementary_Features (Zuletzt aufgerufen am 08.08.2019)

[43] R. Tang (2013) Mathematical Methods for Camera Self-Calibration in Photogrammetry and Computer Vision. URL: https://elib.uni-stuttgart.de/bitstream/11682/3934/1/Tang_Uni.pdf (Zuletzt aufgerufen am 12.08.2019)

[44] S. N. Sinha1, D. Steedly, R. Szeliski1 (2016) A multi-stage linear approach to structure from motion. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/sinhaRMLE10_linearSfm.pdf (Zuletzt aufgerufen am 13.08.2019)

[45] M. J. Westobya, J. Brasington, N. F. Glasser, M. J. Hambrey, J. M. Reynolds (2012) 'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications. URL: https://www.sciencedirect.com/science/article/pii/S0169555X12004217 (Zuletzt aufgerufen am 13.08.2019)

[46] C. Fraser (2018) SLAM, SFM AND PHOTOGRAMMETRY: WHAT'S IN A NAME? URL: https://www.isprs.org/tc2-symposium2018/images/ISPRS-Interview-Fraser.pdf (Zuletzt aufgerufen am 13.08.2019)

[47] F. Herranz, K. Muthukrishnan, K. Langendoen (2011) Camera pose estimation using particle filters. URL: https://www.researchgate.net/publication/229033775_Camera_pose_estimation_using_particle_filters (Zuletzt aufgerufen am 19.08.2019)

[48] H. Le, M. Nguyen, H. Tran, W. Yeap (2017) Pictorial AR Tag with Hidden Multi-Level Bar-Code and Its Potential Applications. URL: https://www.mdpi.com/2414-4088/1/3/20 (Zuletzt aufgerufen am 19.08.2019)

[49] D. G. Lowe (1999) Object Recognition from Local Scale-Invariant Features. URL: https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf (Zuletzt aufgerufen am 19.08.2019)

[50] E. Marchand, H. Uchiyama, F. Spindler (2016) Pose Estimation for Augmented Reality: A Hands-On Survey. IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers, 2016, 22 (12), pp.2633 - 2651. URL: https://hal.inria.fr/hal-01246370/document (Zuletzt aufgerufen am 19.08.2019)

[51] ARCore overview. URL: https://developers.google.com/ar/discover (Zuletzt aufgerufen am 29.08.2019)

[52] ARCore Supported Devices. URL: https://developers.google.com/ar/discover/supported-devices (Zuletzt aufgerufen am 29.08.2019)

[53] ARCore Augmented Images URL: https://developers.google.com/ar/develop/java/augmented-images/ (Zuletzt aufgerufen am 29.08.2019)

[54] ARCore Fundamental Concepts. URL: https://developers.google.com/ar/discover/concepts (Zuletzt aufgerufen am 29.08.2019)

[55] ARCore Development Environment. URL: https://developers.google.com/ar/develop (Zuletzt aufgerufen am 29.08.2019)

[56] Patent: System and method for concurrent odometry and mapping. URL: https://patents.google.com/patent/US20170336511A1/en (Zuletzt aufgerufen am 30.08.2019)

[57] Sceneform SDK for Android. URL: https://github.com/google-ar/sceneform-android-sdk (Zuletzt aufgerufen am 30.08.2019)

[58] Sceneform Overview. URL: https://developers.google.com/ar/develop/java/sceneform (Zuletzt aufgerufen am 30.08.2019)

[59] Physically Based Rendering in Filament. URL: https://google.github.io/filament/Filament.html#overview/principles (Zuletzt aufgerufen am 30.08.2019)

[60] Google Places API. URL: https://developers.google.com/places/web-service/intro (Zuletzt aufgerufen am 30.08.2019)

[61] Navigation mobiler Systeme. URL: http://ots.fh-brandenburg.de/downloads/scripte/ams/2014-Navigation-Teil%201.pdf (Zuletzt aufgerufen am 02.09.2019)

[62] Github, Dexter: Android library that simplifies the process of requesting permissions at runtime. URL: https://github.com/Karumi/Dexter (Zuletzt aufgerufen am 02.09.2019)

[63] Android 6.0 Marshmallow. URL: https://android.googleblog.com/2015/10/get-ready-for-sweet-taste-of-android-60.html (Zuletzt aufgerufen am 02.09.2019)

[64] Github, Volley. URL: https://github.com/google/volley (Zuletzt aufgerufen am 03.09.2019)

[65] Volley overview. URL: https://developer.android.com/training/volley/index.html (Zuletzt aufgerufen am 03.09.2019)

[67] Feature Request: Provide a geo-oriented world coordinate space. URL: https://github.com/google-ar/arcore-android-sdk/issues/119 (Zuletzt aufgerufen am 03.09.2019)

[68] R. Mur-Artal, J. D. Tardós (2017) ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. URL: https://arxiv.org/pdf/1610.06475.pdf (Zuletzt aufgerufen am 03.09.2019)

Abbildungsverzeichnis