Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Henrik Brauer

Entwicklung eines Augmented Reality Frameworks auf Basis von Kamera-basierten Trackingverfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Henrik Brauer

Entwicklung eines Augmented Reality Frameworks auf Basis von Kamera-basierten Trackingverfahren

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik (Master of Science)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 25. Februar 2010

Henrik Brauer

Thema der Masterarbeit

Entwicklung eines Augmented Reality Frameworks auf Basis von Kamera-basierten Trackingverfahren.

Stichworte

Augmented Reality, Erweiterte Realität, SIFT, SURF, KLM, 3D Rekonstruktion, Positionsbestimmung

Kurzzusammenfassung

In der vorliegenden Ausarbeitung wird ein Framework entwickelt, mit dessen Hilfe Augmented Reality-Anwendungen erstellt werden können, ohne dass hierfür spezielle Marker oder Sensoren notwendig sind. Stattdessen können Alltagsgegenstände wie ein Bild oder eine Postkarte als virtuelle Marker genutzt werden. Augmented Reality ist die Bezeichnung für einen direkten oder indirekten Blick auf eine physikalische reale Welt, dessen Elemente erweitert werden mit virtuellen computergenerierten Bildern. Zur Umsetzung wird ein Algorithmus genutzt, der eine Kombination aus Speeded Up Rebust Features (SURF) und Lucas-Kanad-Methode (KLM) ist. Dieser ermöglicht es Augmented Reality Anwendungen in Echtzeit zu betreiben.

Henrik Brauer

Title of the paper

Development of an augmented reality framework, based on camera-based Tracking procedure.

Keywords

Augmented reality,SIFT, SURF, KLM, 3D reconstruction, localization

Abstract

The present thesis describes the development of a framework which enables Augmented Reality applications without the need for specific markers or sensors. Instead, everyday objects such as a picture or a postcard can be used as virtual markers. Augmented reality is a term for a direct or indirect live view of a physical real-world environment whose elements are merged with or augmented by virtual computer-generated imagery. The algorithm used in implementing this is a combination of Speeded Up Rebust Features (SURF) and Lucas-Kanade method (KLM). It allows running the augmented reality applications in real-time.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

In der vorliegenden Arbeit wird ein Framework entwickelt, mit dessen Hilfe Augmented Reality-Anwendungen erstellt werden können, ohne dass hierfür spezielle Marker oder Sensoren notwendig sind. Stattdessen können Alltagsgegenstände wie eine Bild oder eine Postkarte als virtuelle Marker genutzt werden.

Unter Augmented Reality (AR) oder erweiterter Realität versteht man die computergestützte Erweiterung der Realitätswahrnehmung. Diese Erweiterung kann alle menschlichen Sinnesmodalitäten ansprechen, häufig wird jedoch unter erweiterter Realität nur die visuelle Darstellung von Informationen verstanden.

Abgrenzung zur Virtual Reality

Eine mögliche Definition der Virtual Reality (VR) könnte wie folgt aussehen: “Als Virtual Reality wird die Darstellung und gleichzeitige Wahrnehmung der Wirklichkeit und ihrer physikalischen Eigenschaften in einer in Echtzeit computergenerierten, interaktiven virtuellen Umgebung bezeichnet.“ Das bedeutet, dass eine virtuelle Realität eine vollständig vom Computer generierte Umgebung mit einer möglichst guten grafischen Darstellung ist, die im Idealfall vom Betrachter als “real“ empfunden wird.

Im Gegensatz dazu kann er in der Augmented Reality neben den überlagerten virtuellen Objekten, auch die reale Umgebung wahrnehmen. Normalerweise wird computergenerierte Grafik in das Blickfeld des Nutzers projektiert, um extra Informationen über die Umwelt zu liefern oder um eine visuelle Anleitung, für die Erfüllung einer Aufgabe bereitzustellen. Die einfachste Form von Augmented Reality könnte das Anzeigen von einfachen Pfeilen oder Text-Labels sein. Zum Beispiel könnten Pfeile den Nutzer durch eine fremde Stadt führen. In komplexeren Anwendungen könnten z.B. 3D-Modelle angezeiget werden, sodass sie nicht mehr von der Umwelt zu unterscheiden sind.

Augmented Reality befindet sich deutlich näher an der Realität, während die virtuelle Realität wenig bis gar keinen Bezug zur realen Welt besitzt. Augmented Reality-Systeme interagieren stärker mit der Realität, weshalb es von größter Bedeutung ist, dass die überlagerten Informationen an der richtigen Stelle eingeblendet werden, um für den Betrachter die Illusion zu erzeugen, dass reale und virtuelle Welt eins sind.

1.1 Anwendungsgebiete

Augmented Reality-Systeme können in vielen verschiedenen Bereichen füre die unterschiedlichsten Zwecke eingesetzt werden. Die Fülle an Anwendungsbeispielen ist riesig. Im Folgenden werden jeweils Beispiele aus unterschiedlichen Bereichen vorgestellt, um einen besseren Einblick der Nutzungsmöglichkeit der erweiterten Realität zu vermitteln.

Aufbau und Instandhaltung: Mithilfe von Augmented Reality ist es möglich, große komplexe Maschinen leichter aufzubauen, zu warten und zu reparieren, indem einem Monteur die im Schaltplan oder in der Bedienungsanleitung verzeichnete Informationen oder Arbeitsanweisungen in die reale Szene eingeblendet werden. Dem Monteur können so einzelne Handlungsschritte direkt angezeigt werden, was nicht nur Zeit spart, sondern auch die Wahrscheinlichkeit, dass wichtige Schritte vergessen werden, wird verringert. Um Informationen zu Bauteilen abzurufen, muss der Monteur den Arbeitsplatz nicht mehr verlassen, sondern kann sie sich direkt im Blickfeld anzeigen lassen.

Entwicklung und Konstruktion: In der Entwicklung können sich Konstrukteure dreidimensionalen Modelle der Produktes ansehen, ohne ein Modell anfertigen zu müssen. Sie haben die Möglichkeit, sich das Produkt nicht nur von außen, sondern auch von innen anzuschauen. Zudem können mehrere Teilnehmer “live“ mit dem neuen Produkt in Interaktion treten. In Echtzeit können dann die Konstrukteure bestimmte Bereiche des neuen Produktes markieren und ändern und so auf Kundenwünsche direkt eingehen.

Unterhaltung: Im Fernsehen ist Augmented Reality allgegenwärtig. Kaum eine Nachrichtensendung kommt heute ohne aus. Als bekanntestes Beispiel hierfür kann der Wetterbericht genannt werden. Beim Wetterbericht befindet sich der Moderator vor einer leeren Wand, die meist blau oder grün gestrichen ist. Für Fernsehzuschauer ist diese Wand nicht zu sehen. Anstelle der Wand sieht er die Wetterkarte. Aber nicht nur im Film und Fernsehen kann Augmented Reality eingesetzt werden, sondern auch in der Welt der Spieleindustrie. Z.B. ist ein Spiel für die Sony Playstation 3 mit dem Namen Eyepet erschienen. In dem Spiel agiert der Spieler mit einem virtuellen Haustier in der realen Welt. Dies funktioniert über eine Kamera und spezielle Gegenstände die das Spiel erkennt.

Verbraucher-Design und Mode: Heute gibt es schon Anwendungen, die es erlauben, durch das Tragen eines speziellen Markers Kleidung auf den eigenen Körper projizieren zu lassen, um zu sehen, wie sie einem steht. Dieses Konzept, weiter gedacht, könnte sich z.B. ein Kunde beim Friseur direkt anzeigen lassen, wie ihm unterschiedliche Frisuren stehten, um sich dann für die schönste Frisur entscheiden zu können. Oder: Kleidung in einem Geschäft wird nur virtuell anprobiert und dann nach Maß geschneidert.

Militär: Im Kampfeinsatz können sich Soldaten Ziele und Gefahrenzonen im Gelände anzeigen lassen. Kampfpiloten können sich Informationen via Display während des Flugs direkt in das Sichtfeld einblenden lassen. Beispielsweise könnten das Entfernungen zum feindlichen Objekt oder Flugdaten sein. Aber auch zum Training könnte Augmented Reality eingesetzt werden, um kostengünstig realitätsnahe Übungen zu absolvieren.

1.2 Anwendungsszenarie Museum

In vielen Museen gibt es Objekte, deren wirklicher Sinn nur in einem gewissen Kontext zu erkennen ist. In Naturkundemuseen finden sich z.B. oft Teile von Skeletten ausgestorbener Tiere. Um den Besuchern zu erläutern, an welcher Stelle sich die Knochen in dem Tier befunden haben, sind oft Schautafeln aufgebaut oder es werden kleine Filmchen gezeigt. Oft gibt es z.B. auch Nachbauten von echten Tieren wie z.B. Dinosaurierer, zusätzlich wird dann oft der Lebensraum der Tiere angedeutet.

Um den Besucher tiefe Einblicke in die Welt der Tiere zu geben, bietet eine Augmented Reality-Anwendung viele Möglichkeiten. Aus Knochenteilen eines Tieres kann das ganze Skelett entstehen, indem die fehlenden Knochen als virtuelle Elemente hinzugefügt werden. Das ganze Knochenskelett muss nicht mehr starr stehen, sondern mithilfe eines virtuellen Knochenskelettes kann gezeigt werden, wie sich das Tier bewegt hat. Es gibt die Möglichkeit, statt lebloser Nachbildungen in Plastik virtuell die Tiere neu zu erschaffen. So könnte z.B. ein Raum in eine Dinosaurierlandschaft umgewandelt werden, in der nicht nur verschiedene Dinosaurier durch die Landschaft streifen, sondern auch der natürliche Lebensraum nachempfunden wird. Den Besuchern würde das ein Erlebnis wie im Zoo bieten, nur dass all die Tiere, die sie sehen, schon seit Millionen von Jahren nicht mehr leben.

1.3 Schwerpunkte und Projektziel

Die Nutzung bildverarbeitender Prozesse zur Positionsbestimmung ist durch die schneller werden Computer immer einfacher möglich. Leider fehlt es momentan noch an geeigneten offenen Frameworks, die es ermöglichen, Augmented Reality Anwendungen ohne Marker oder externe Sensoren zu implementieren.

Ziel dieser Arbeit ist es, ein Framework zu entwickeln, das es ermöglicht, einfache Augmented Reality-Anwendungen zu entwickeln, ohne Vorkenntnisse in diesem Bereich zu haben und spezielle Sensoren oder Marker nutzen zu müssen. Die Ermittlung der eigenen Position im Raum soll nur über Bilder, die von einer Kamera aufgenommen werden, geschehen. Um die Entwicklung möglichst leicht zu gestalten, sollen zusätzlich Tools entwickelt werden, die den Entwickler bei seiner Arbeit unterstützten.

Abbildung 1.1: Virtueller T-Rex (Vordergrund) auf der Dinosaurier Expo 2009 in Japan. Quelle: NTDTV.com.

1.4 Ideales System

Um eine Einführung zu geben wie ein zukünftiges System das mit dem Framework entwickelt wurde aussehen könnte, wird der Aufbau an dieser Stelle anhand eines idealen Systems erklärt, also eines Systems, bei dem keine Ungenauigkeiten auftreten und das keine Rücksicht auf technische Hürden nimmt. Das System soll aus einem tragbaren Computer sowie einer Kamera bestehen. Auf dem Display des Computers soll das Bild der Kamera angezeigt werden, das um virtuelle Objekte erweitert wurde.

Das System kann in drei Teilschritte aufgeteilt werden:

- Objekterkennung
- Positionsbestimmung
- Erweitern der Szene um virtuelle Objekte

Die Objekterkennung setzt voraus dass das Objekte bekannt sind. Das bedeutet bevor ein Nutzer das System nutzten kann, müssen diese Objekte bestimmt werden. Das kann abhängig von der Technik durch ein einfaches Bild des Objektes geschehen oder durch eine 3D-Repräsentation des Objektes. Durch ein geeignetes Bildverarbeitungsverfahren soll nun das Objekt bestimmt werden. Diese geschieht, indem die bekannten Informationen, sei es nun ein Bild oder eine 3D-Repräsentation, mit dem von der Kamera aufgenommen Bild verglichen werden.

Wurde ein Objekt erkannt, lässt sich aufgrund der Lage des Objektes im Kamerabild im zweiten Schritt die Position bestimmten. Mit Position wird an dieser Stelle die Lage des Objektes im

Raum im Verhältnis zur Kamera bezeichnet, diese kann durch die Rotation und Translation des Objektes zur Kamera eindeutig beschrieben werden.

Im letzten Schritt wird dann aufgrund der ermittelt Position ein virtuelles Objekt eingeblendet. Das heißt ein vorher definiertes virtuelles Objekt wird rotiert und transformiert und dann in das Kamerabild gezeichnet sodass es für den Betrachter den Eindruck erweckt das es sich an der gleichen Stelle befindet wie das erkannte Objekt.

2 Bestehende Ansätze

Dieses Kapitel soll eine Übersicht über die zurzeit genutzten Augmented Reality-Techniken gegeben. Es werden sensorbasierte Techniken und markerbasierte Techniken vorgestellt. Außerdem wird ein Überblick über den Stand der Forschung von markerlosen Techniken gegeben. Die genutzten Verfahren können in zwei Kategorien unterteilt werden. Die erste Kategorie sind spezielle Sensoren, die zur Bestimmung der Position entwickelt wurden. Die zweite Kategorie sind bildbasierte Tracking-Verfahren.

Bei bildbasierten Tracking-Verfahren, die die Anforderungen mobiler Augmented Reality erfüllen, führt der Benutzer die Kamera bei sich. Da die Kamera am Benutzer selbst befestigt ist und die Umgebung beobachtet, spricht man auch von inside-out Tracking. Im Gegensatz zu typischen Tracking-Systemen, bei denen Sensoren in der Umgebung ein Objekt verfolgen, die auch als outside-in Tracking bezeichnet werden. Der Anwender führt die aktiven Komponenten des Tracking-Systems mit sich, dabei handelt es sich um eine oder mehrere Kameras. Bildverarbeitungsalgorithmen extrahieren alle nötigen Informationen. um die Position und Orientierung der Kamera und somit die Position des Nutzers zu schätzen.

In der Praxis wird heutzutage vor allem markerbasiertes Tracking genutzt. Bei diesem Verfahren werden optische Marker im Raum angebracht, deren Position im Weltkoordinatensystem bekannt sind bzw. sie fungieren selbst als Nullpunkt im Weltkoordinatensystem. Durch Bildanalyseverfahren wird dann die Position des Kamerabildes im Verhältnis zum Marker bestimmt. Im Bereich der Sensoren gibt es unterschiedlichste Systeme, die z.B. auf GPS, Ultraschall oder Beschleunigungssensoren basieren. Markerbasiertes Tracking und das Tracking mit Sensoren am Beispiel von GPS und einem digitalen Kompass werden im folgenden Abschnitt erläutert. Außerdem werden featurebasierte Tracking-Verfahren vorgestellt, die die Grundlage dieser Arbeit sind.

2.1 Markerbasiertes Tracking

Beim markerbasierten Tracking wird die Positionsbestimmung von Objekten in der realen Umgebung mit speziellen Markierungen (allgemein Marker genannt) durchgeführt. In der Regel haben die Marker die Form eines Quadrates. In der Mitte befindet sich ein eindeutiges Muster, um verschiedene Marker zu identifizieren und deren Ausrichtung bestimmen zu können. Ein besonderes Merkmal dieser Markierungen ist, dass sie besonders gut durch Bildanalyseverfahren extrahiert werden können und dass sich anhand ihrer speziellen Form Positionierungsinformationen sehr leicht bestimmen lassen. Aufgrund des niedrigen Ressourcenverbrauchs dieses Verfahrens ist es möglich, diese Technik auch auf Smartphones einzusetzen [WS07].

Im weitern Verlauf wird die genaue Funktionsweise eines markerbasierten Ansatzes an Hand der Technik, die dem ARToolKit [ART09b] zu Grunde liegt, erläutert. Das ARToolKit ist ein spezielles Framework für Marker basiertes Tracking.

Die Form der Marker im ARToolKit ist quadratisch. Zur Analyse des Videobildes wird ein Algorithmus durchgeführt, der die Bildpunkte (Pixel) nach allen Regionen durchsucht, deren Kontur durch ein Viereck beschrieben werden kann bzw. die eine Verzerrung eines Vierecks darstellen.

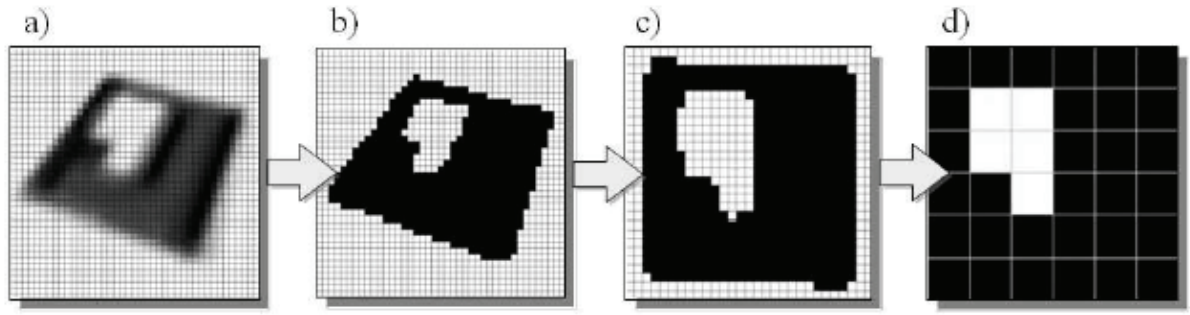


Abbildung 2.1: Analyse des Videobildes: a)Erkennen einer Region. b)Konvertierung. c)Koordinatentransformation. d)Skalierung bestimmen. Quelle: [Teg06].

Eine gefundene Region (Bild 2.1.a) wird bei einem bestimmten Helligkeitsschwellenwert in ein Bild mit einem Bit Farbtiefe (schwarz/weiß) konvertiert (Bild 2.1.b) und anschließend einer perspektivischen Transformation unterzogen. Der Transformationsprozess verschiebt die Eckpunkte der vier Konturlinien, bis sie eine quadratische Form mit möglichst lotrechten Kanten ergeben und berechnet auf dieser Basis neue Pixel-Positionen für das Muster (2.1.c). Das quadratische Bild wird auf die Dimensionen der definierten Markerfelder skaliert (2.1.d), sodass es nun mit dem gesuchten Muster verglichen werden kann.

Wird der Marker erfolgreich erkannt, werden die ursprünglichen Konturlinien der gefundenen Regionen als Vektoren interpretiert und spannen eine x-y-Ebene auf. Ein weiterer, senkrecht zu dieser Ebene stehender Vektor wird mathematisch berechnet und als dritte Dimension zur Erzeugung eines Raumes herangezogen.

Innerhalb dieses (virtuellen) Raumes wird die Länge der Vektoren mit den angegebenen (realen) Abmessungen der Eckpunkte in Beziehung gesetzt, um eine Größeneinheit für das Koordinatensystem im Raum zu erzeugen. Dieses kann nun verwendet werden, um exakte Positionen anzugeben, bspw. um den eindeutigen Standort der Kamera zu ermitteln oder um virtuelle Objekte deckungsgleich zum realen Sichtfeld des Betrachters zu platzieren.

Neben dem ARToolKit implementieren noch mehrere andere Frameworks die gleiche oder eine ähnliche Technik. Zwei Beispiele hierfür sind unter [STT09] und [ATA09] zu finden.

Anwendungsbeispiel: MagicBook

Das MagicBook ist ein Projekt, das es ermöglicht, einen nahtlosen Übergang zwischen realer und virtueller Realität zu erzeugen. Wenn der Nutzer sich die Seiten eines echten Buches mit-

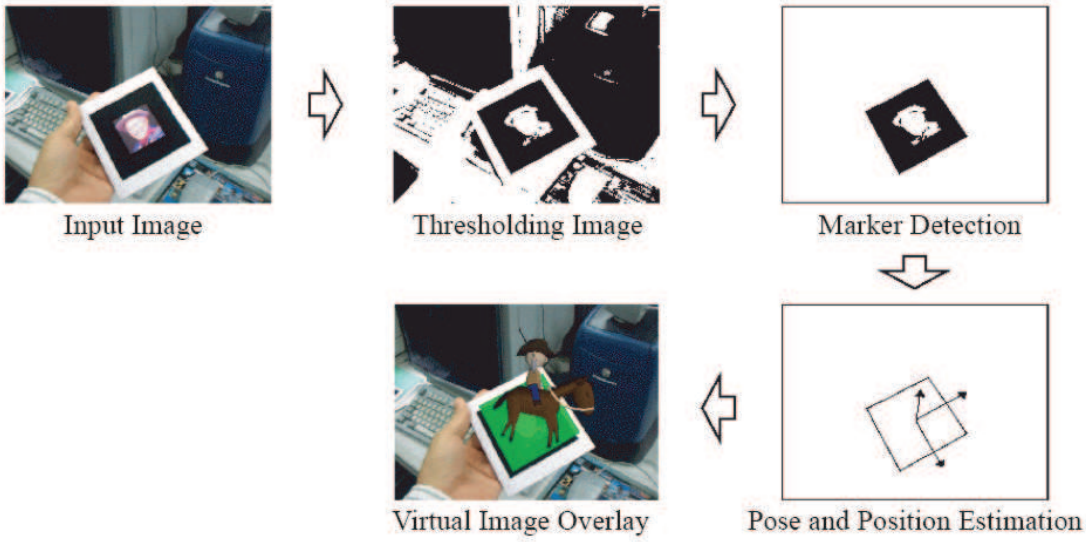


Abbildung 2.2: Bearbeitungsschritte des ARToolkit. Quelle: [LF05].

hilfe eines Handheld-Displays anschaut, sieht er zusätzlich virtuelle Inhalte. Der Nutzer hat die Möglichkeit, sich die Szene von allen Seiten anzuschauen. Diese MagicBooks findet man unter anderem in Museen. Z.B. wurde ein MagicBook entwickelt, um Kindern die Entstehung der Erde zu erklären. Auf jeder Seite des Buches wird ein neuer Schritt erläutert. Die Kinder können sich jedes Stadium der Erde von allen Seiten anschauen und auch Dinge wie Vulkanausbrüche genau beobachten.

Abbildung 2.3: Augmented Reality Beispiel ein Vulkan in einem MagicBook.

2.2 Sensorbasiertes Tracking

Es gibt unterschiedlichste Sensoren, die für das Tracking genutzt werden können. Eine Übersicht ist unter [Teg06] zu finden. Hier wird im Speziellen nur auf GPS in Verbindung mit einem Kompass eingegangen, da die nötige Technik hierfür in vielen mobilen Endgeräten vorhanden und so

ohne zusätzliche Kosten nutzbar ist. Andere Techniken verlangen in der Regel die Anschaffung teurer Sensorentechniken, was in der Praxis nur für spezielle Projekte sinnvoll ist.

GPS

GPS steht für Global Positioning System, 24 Satelliten mit Atomuhren bilden die Basis des weltumspannenden Positionssystems. In gepulsten Abständen senden die Satelliten Signale aus, die die Satellitenposition und einen Zeitstempel enthalten der den genauen Versandzeitpunkt angibt. Am Boden kann der Empfänger auf Basis des Versandzeitpunktes und Empfangszeitpunktes die Laufzeiten ermitteln. Aus der Laufzeit lässt sich nun die Entfernung zum Satelliten ermitteln, woraus sich durch Triangulation mehrerer Satellitensignale die eigene Position bestimmen lässt.

Der größte Nachteil von GPS ist die Verzerrung der Signale durch die amerikanische Regierung. Diese will durch die reduzierte Präzision die Nutzung von GPS zu militärischen Zwecken verhindern. Durch Referenzpunkte am Boden, deren genaue Position bekannt ist, wird dennoch eine hohe Präzision auch für die zivile Nutzung ermöglicht.

Digitaler Kompass

Das Magnetfeld der Erde ist ein komplexes Netz von Nord-Süd-Strahlung. Der Richtungssensor nutzt dieses Feld zur exakten Bestimmung der Richtung. Der Sensor besteht aus zwei rechteckigen Spulen, in denen das Magnetfeld der Erde feinste Spannungen erzeugt. Je nach Position und Richtung sind diese Spannungen unterschiedlich hoch. Ein nichtmagnetischer Widerstand liegt zwischen beiden Spulen und nimmt die feinen Spannungsänderungen auf.

Mit jeder neuen Position ändern sich die Werte, die der Widerstand misst. Ein Mikroprozessor wandelt die Werte um, dadurch lässt sich die Blickrichtung der Kamera bestimmen. Die Beschreibung der Technik des digitalen Kompasses basiert auf der Beschreibung der Technik, die Casio (http://www.casio-europe.com/de/watch/technologie/richtungssensor/) einsetzt. Diese Beschreibung soll exemplarisch die Technik beschreiben. Wie sie konkret in verschiedenen Geräten realisiert wird, ist vom Hersteller abhängig.

Werden beide Sensoren zusammen genutzt, ist es möglich, die Position und die Blickrichtung zu bestimmen und auf dieser Basis Objekte anzuzeigen. Insgesamt ist diese Technik im Verhältnis z.B. zu markerbasierten Ansätzen sehr ungenau. Der Ansatz eignet sich aber besonders für den Einsatz auf mobilen Geräten, da so gut wie keinerlei zusätzlicher Ressourcen verbraucht werden. Zusätzlich bietet er den Vorteil, dass er an jedem Ort der Welt ohne zusätzliche Installationen genutzt werden kann.

Anwendungsbeispiel: WIKITUDE World Browser

WIKITUDE World Browser ist eine Augmented Reality-Anwendung, ursprünglich für Android Handys entwickelt, heute aber auch auf anderen Handys wie z.B. dem IPhone verfügbar. Mit Wikitude ist es möglich, live Informationen über die Umgebung anzeigen zu lassen, indem die

Kamera in Richtung der entsprechenden Objekte gehalten wird. Zurzeit (14.10.2009) kennt Wikitude ca. 350.000 “Points-of-Interest“. Es werden die Namen der entsprechenden Punkte angezeigt und auf Wunsch ist es auch möglich, sich den entsprechenden Wikipedia-Artikel anzeigen zu lassen. Zur Bestimmung der einzelnen Punkte wird die oben beschriebene Technik genutzt. Anhand der GPS-Position werden alle Punkte, die sich in einen gewissen Umkreis befinden, bestimmt und die Blickrichtung wird durch einen digitalen Kompass ermittelt. Der Punkt aus dem Umkreis, der am besten zum Ergebnis passt, wird dem Nutzer angezeigt.

Abbildung 2.4: Augmented Reality-Beispiel “Wikitude AR Travel Guide“ auf einem G1 Google Smartphone (Android). Quelle: http://www.mobilizy.com/.

2.3 Featurebasiertes Tracking

Featurebasierte Tracking-Verfahren verzichten auf die Anbringung von künstlichen Markern in der Umgebung, verursachen aber einen wesentlich höheren Rechenaufwand als markerbasierte Verfahren. Zu Grunde liegen dem featurebasierten Tracking, Algorithmen zum Finden und Beschreiben von Features in Bildern. Feature sind bestimmte Merkmale, die in einem Bild auftauchen können. Das können sowohl markante Punkte, als auch Kanten von Objekten in Form von Linien, Kurven, Kreisen oder Ellipsen sein. Auch Farben können Merkmale als Bildinformation darstellen.

Grundsätzlich lassen sich auch die featurebasierten Tracking-Verfahren in zwei Gruppen unterteilen. Modellbasierte Verfahren, bei denen versucht wird, ein 3D-Modell mit dem Kamerabild zu matchen und hieraus die Kamera-Pose zu schätzen und Verfahren, bei denen Features von Bild zu Bild verfolgt werden. Dabei wird aus der Bewegung der Features im Bild auf die Kamera-Pose geschlossen. Während die modellbasierten Verfahren die Kamera-Pose im Koordinatensystem des zu trackenden Objekts berechnen, bewegt sich die Kamera in den “Bild zu Bild“-Verfahren in einem vollkommen unabhängigen Koordinatensystem. In diesem Fall ist die Transformation zwischen den Tracker- Koordinaten und dem Referenz-Koordinatensystem unbekannt, sodass eine

Initialisierung mit einem anderen Verfahren notwendig ist, auf dessen Ergebnis das “Bild-zu-Bild“ Verfahren dann aufsetzen kann.

Unter Feature Tracking versteht man dann das Verfolgen dieser extrahierten Merkmale innerhalb einer Bildfolge. Dabei müssen die Feature-Informationen logisch miteinander verknüpft werden. Aufeinanderfolgende Merkmale, die von Bild zu Bild übereinstimmen, bilden den so genannten Featurestream. Soweit dem Autor bekannt, gibt es zurzeit noch kein spezielles Framework für featurebasiertes Tracking. Es wurden allerdings mehere „Testsysteme“ entwickelt. Einee Auswahl wird im nächsten Abschnitt vorgestellt.

2.4 Bestehende Systeme

An dieser Stelle sollen zwei verschiedene Trackingsystem vorgestellt werden, die an verschieden Universitäten entwickelt wurden. Das erste System ist eins der ersten Systeme, bei dem Markerless Tracking genutzt wurde und ist besonders simple. Das zweite System ist an der Eidgenössischen Technischen Hochschule in Lausanne (EPFL) entwickelt worden. Es verwendet ein hybrides Verfahren, bei dem sowohl relative als auch absolute Informationen in die Positionsbestimmung eingehen.

Die Zusammenfassung wurde teilweise von [Esc06] übernommen.

2.4.1 Das System der Universität Oxford

Das Tracking-System der Universität Oxford [SFZ00a] ist ein relativ einfach aufgebauter Online-Tracker, der so konzipiert wurde, dass er ohne Vorkenntnisse über die Szenengeometrie und mit minimalem Aufwand für den Benutzer einsetzbar ist. Der einzige manuelle Eingriff findet zur Initialisierung statt. Das anschließende Tracking der Kameraposition erfolgt vollautomatisch.

Bei der Positionsbestimmung macht sich das System den Umstand zu Nutze, dass sich in vielen Situationen planare Strukturen im Bild finden lassen. Dies könnte z.B. der Fußboden eines Zimmers oder die Fassade eines Gebäudes sein.

Initialisierung

Als ersten Schritt der Initialisierung markiert der Benutzer einen möglichst großen Bildbereich, der vollständig innerhalb einer Ebene liegen sollte. Dies gibt dem nachfolgenden Matchingschritt einen Hinweis darauf, welche Ebene im Bild verfolgt werden soll und kann weggelassen werden, wenn die zu verfolgende Ebene auch die größte Ebene im Bild ist. Im nächsten Schritt ermittelt das System für die ausgewählte Ebene Trackingpunkte.

Eine Positionsbestimmung mittels einer Ebene im Bild ist dann möglich, wenn die Kalibrationsparameter der Kamera sowie die Lage der Ebene im Raum bekannt sind. Da das Weltkoordinatensystem frei gewählt werden kann, kann die Ebene als xy-Ebene festgelegt werden. Der Benutzer markiert nun vier Punkte, auf der im ersten Schritt ausgewählten Fläche, mit der Maus.

Abbildung 2.5: Die Initialisierung des Tracking-Systems der Universität Oxford ist besonders einfach. Zuerst wird ein Bildbereich markiert, der auf der zu verfolgenden Ebene liegen muss(links). Darauf setzt das Tracking ein (Mitte). Der Benutzer muss nun vier Punkte in rechteckiger Anordnung markieren(rechts). Damit ist die Initialisierung abgeschlossen. Quelle: [SFZ00a].

Dabei muss es sich um Punkte handeln, die in der (tatsächlichen, dreidimensionalen) Szene ein Rechteck auf der gewählten Ebene bilden. Es kann sich beispielsweise um die vier Ecken eines Fensters oder eines großen Pflastersteins handeln.

Der erste auf diese Weise markierte Punkt wird zum Ursprung des Weltkoordinatensystems, die darauf folgenden Punkte definieren zusammen mit dem ersten die X- und Y-Koordinatenachsen. Vier Punkte genügen, um die zweidimensionale projektive Abbildung von Punkten aus der XY-Ebene des Weltkoordinatensystems in die Bildebene zu bestimmen. Eine solche Abbildung nennt sich Homographie, diese wird in Abschnitt 4.3 vorgestellt. Diese spezielle Homographie H_w^0 wird „Welthomographie für Bild 0“ genannt. Aus dieser Homographie kann ähnlich zum Verfahren aus dem Abschnitt 4.3 die Pose der Kamera bestimmt werden.

Matching aufeinander folgender Bilder

Für jedes weitere ($i+1$-te) Kamerabild wird nun versucht, Merkmale zu finden, die mit jenen des vorherigen Bildes korrespondieren. Dabei wird die Suche auf ein Suchfenster um den Merkmalspunkt eingeschränkt. Der optische Vergleich zweier Merkmale geschieht, indem im Suchfenster nach Merkmalen mit hoher Korrelation Ausschau gehalten wird. Mithilfe des RANSAC-Algorithmus (siehe 4.6) wird gleichzeitig nach einem guten Matching zwischen den Merkmalen und einer Homographie H_i^{i+1} gesucht, welche die projektive Abbildung zwischen den beiden Merkmalsmengen beschreibt. Durch Verkettung dieser Homographie mit der Welthomographie des Vorgängerbildes kann auch für dieses Bild eine Welthomographie berechnet werden:

$$H_w^{i+1} = H_i^{i+1} H_w^i \tag{2.1}$$

Auf diese Weise steht zu jedem Bild i eine Welthomographie H_w^i zur Verfügung, die die Punkte auf der XY-Ebene des Weltkoordinatensystems auf Punkte in der Bildebene abbildet. Aus H_w^i lässt sich nun die Rotation und Translation bestimmen (siehe 4.3).

2.4.2 Das System der EPFL

Das an der Eidgenössischen Technischen Hochschule in Lausanne entstandene und in mehreren Veröffentlichungen ([VLF04b] [VLF04a] [LPF04]) vorgestellte System ist zurzeit einer der am weitesten fortgeschrittenen Ansätze auf dem Gebiet der Augmented Reality. Es vereint erfolgreich relative und absolute Informationen. Dafür benötigt es allerdings in der Lernphase viel manuelle Unterstützung. Konkret wird gefordert, dass die zu trackende Szene als dreidimensionales Polygonmodell vorliegt und eine Reihe von Beispielansichten (Keyframes) vorhanden sind. Zu diesen Keyframes muss eine gültige Registrierung[1] vorliegen. Weiterhin müssen die Kalibrationsparameter der Kamera bekannt sein.

Damit das Tracking nicht auf die durch die Keyframes grob vorgegebenen Kamerapositionen reduziert wird, sieht das System die Erstellung von Keyframes im laufenden Betrieb vor. Diese so genannten Online-Keyframes ermöglichen ein unterbrechungsfreies Tracking, solange sich ein Objekt im Bild befindet, das als Polygonmodell vorliegt. Das Polygonmodell der Szene (oder Objekten darin) ermöglicht zusätzlich, ein Keyframe aus einer anderen Perspektive zu berechnen. Über diese Neuberechnung kann die Korrelation der Merkmale des aktuellen Bildes mit denen des Keyframes maximiert werden, indem eine Ansicht nahe der aktuellen Kameraposition erzeugt wird. Zusätzlich erlaubt das Polygonmodell auf einfache Weise, Punkte in die Registrierung aufzunehmen, deren 3D-Positionen unbekannt sind.

Die Lernphase

Der Tracker der EPFL enthält Vorwissen über die Szene. Dieses besteht zum einen aus Keyframes, in denen mit dem Harris Corner Detector nach Merkmalspunkten gesucht wird. Mittels des kommerziellen Programms ImageModelerTM von RealVizTM wird die dreidimensionale Lage der Merkmalspunkte bestimmt.

Als weitere Information wird ein Polygonmodell der Szene beziehungsweise einiger darin vorkommender Objekte genutzt. Dieses kann mit einer beliebigen CAD-Anwendung erstellt werden. Die Verwendung eines Polygonmodells erhöht die Anforderungen an den Nutzer beträchtlich. Vacchetti et al. rechtfertigen diesen Aufwand damit, dass für viele Anwendungen aus dem Bereich der Augmented Reality in jedem Fall ein Modell benötigt wird [VLF04a]. Modell und Merkmalspunkte müssen dasselbe Weltkoordinatensystem verwenden.

Das System der EPFL nutzt bereits während der Registrierung der Kamerapose für das aktuelle Bild eine Kombination aus absoluten Informationen aus der Lernphase und relativen Informationen aus dem Vergleich mit dem Vorgängerbild. Wie die meisten anderen Algorithmen verwendet auch dieses System korrespondierende Punktmerkmale, die über einen Korrelationsvergleich bestimmt werden. Eine Besonderheit des Verfahrens besteht darin, dass die so gefundenen Merkmalspaare nicht zu 100% fehlerfrei sein müssen. Durch die Verwendung eines robusten M-Schätzers[2] kann eine gute Bestimmung der Kamerapose auch mit fehlerhaften Korrespondenzen stattfinden.

[1] Als Registrierung wird der Prozess ein Bild mit einem anderen Bild oder einem Modell derselben Szene, bestmöglich in Übereinstimmung zu bringen bezeichnet.

[2] M-Schätzer (von Maximum-Likelihood-Artig) stellen eine Klasse von Schätzfunktionen dar. M-Schätzer sind im Vergleich zu vielen anderen Schätzern robuster gegen Ausreißer.

Es treten hierbei zwei Typen von Korrespondenzen auf: 2D-2D und 2D-3D. Die Erstgenannten stammen aus gefundenen Korrespondenzen zwischen aktuellem Bild und Vorgängerbild. Letztere werden durch Korrespondenzenfindung mit dem aktuellen Keyframe ermittelt und sollen eine Fehlerakkumulation verhindern, die bei reinem “Bild-zu-Bild“-Tracking auftreten würden.

Es ist wünschenswert, eine Kamerapose zu finden, die den Abstand zwischen projizierten 3D und gemessenen 2D-Punkten minimiert. Dieser lässt sich durch die folgende Summe für alle 3D-2D-Korrespondenzen $X \leftrightarrow x$ des Kamerabilds beschreiben:

$$r_q = \sum_i p(||\Phi(q, X_i) - x_i||^2) \tag{2.2}$$

Hierbei sei q der zu optimierende Parametervektor für die Pose der Kamera im aktuellen Bild, p M-Schätzer von Tukey (siehe [Zha95]) und $\Phi(q, X_j)$ die Projektion von Punkt X_j vom Weltkoordinatensystem in die Bildebene unter Kameraparametern q.

Die Verwendung von Polygonmodellen macht den Einsatz von 2D-2D-Korrespondenzen zur Posebestimmung möglich, da sie eine Rückprojektion eines 2D-Punktes auf die Modelloberfläche, also einen Punkt im Weltkoordinatensystem ermöglichen. Diese Rückprojektion sei durch $\Psi(q, x)$ bezeichnet und projiziert einen Punkt x auf der Bildebene der Kamera mit Pose q auf einen Punkt X im Weltkoordinatensystem. Somit lässt sich der folgende Transferfehler als Summe über alle 2D-2D-Korrespondenzen $x' \leftrightarrow x$ zwischen vorherigem und aktuellem Kamerabild definieren:

$$s = \sum_i p(||\Phi(q, \Psi(q', x_i')) - x_i||^2 + ||x_i' - \Phi(q, \Psi(q, x_i))||^2) \tag{2.3}$$

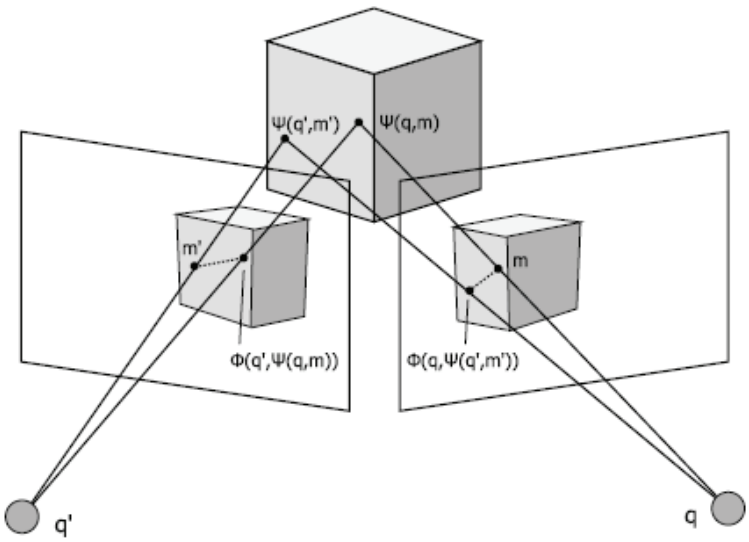


Abbildung 2.6: Transferfehler zwischen Kameras mit Poseparametern q und q' sowie Punkten x und x'. Die gestrichelte Linie stellt die zu minimierende Distanz da. Quelle: [Esc06].

In Abbildung 2.6 wird die Situation anschaulicher dargestellt. Die Terme r_q und s werden nun zusammen minimiert. Da die berechnete Pose q' des Vorgängerbildes fehlerhaft gewesen sein soll und weil sich dieser Fehler auch auf die neue Pose q auswirken könnte, wird q' noch einmal mit in die Minimierung einbezogen:

$$min_{q,q'}(r_q + r'_q + s) \tag{2.4}$$

Als Initialwert für q und q' wird jeweils das bereits berechnete q' verwendet. Für weitere Ausführungen sei auf [LPF04] verwiesen.

Matching mit dem Keyframe

Zum Vergleich der Merkmale des aktuellen Bildes mit denen des Keyframes wird die normalisierte Kreuzkorrelation[3] verwendet. Da es sich bei dieser jedoch um einen einfachen Pixel-für-Pixel-Vergleich in einem kleinen Fenster um den Punkt handelt, reagiert der NCC-Operator äußerst empfindlich auf Änderungen im Betrachtungswinkel. Dies würde sich sehr negativ auf das Matching mit den Keyframes auswirken.

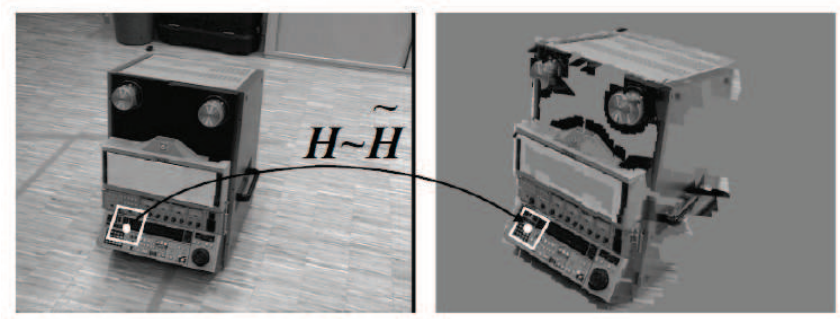


Abbildung 2.7: Neu erzeugte Ansicht eines Keyframes. Für jedes Merkmal wird durch eine Homographie ein kleiner, quadratischer Bildausschnitt transformiert. Auf diese Weise wird die Korrelation maximiert. Quelle: [VLF04a].

Da jedoch ein Polygonmodell der Szene zur Verfügung steht, ist es möglich, eine Version des Keyframes aus einer anderen Perspektive zu erzeugen. Dies geschieht vor dem eigentlichen Matching-Vorgang. Es steht also noch keine aktuelle Kamerapose zur Verfügung. Als Ersatz wird die Pose q' des Vorgängerbildes verwendet, die in den meisten Fällen der aktuellen (jedoch noch unbekannten) Pose q ähnlich ist. Die eigentliche Neuberechnung der Keyframe-Ansicht geschieht, indem zu jedem Merkmalspunkt des Keyframes eine Homographie berechnet wird, die ein kleines rechteckiges Fenster um den Punkt in das neue Bild überträgt. Die Homographie wird dabei durch die jeweilige Fassette des Polygonmodells bestimmt. Abbildung 2.7 zeigt eine solche Neuberechnung.

[3] Normalisierte Kreuzkorrelation ist ein Verfahren mit dem es möglich ist festzustellen, ob zwei Punkte gleich sind, im Gegensatz zu SIFT und SURF ist es nicht rotationinvariant. Siehe auch http://en.wikipedia.org/wiki/Cross-correlation.

3 Objekterkennung und Objektverfolgung

Dieses Kapitel beschäftigt sich mit verschieden Verfahren aus dem Bereich featurebasierter Techniken zur Objekterkennung und Objektverfolgung. Diese sind die Grundlage für die spätere ermittlung der Position von Objekten, die wiederum nötig ist, um virtuelle Objekte in eine Szene einzublenden. Außerdem werden verschiede Verfahren zur Nächsten-Nachbar-Suche vorgestellt die nötwendig ist um gleiche Featurepunkte zu ermitteln.

3.1 Optischer Fluss

Als Optischer Fluss (eng. Optical Flow) wird in der Bildverarbeitung ein Vektorfeld bezeichnet, das die Bewegungsrichtung und Geschwindigkeit für jeden Bildpunkt einer Bildsequenz angibt. Der Optische Fluss kann als die auf die Bildebene projizierten Geschwindigkeitsvektoren von sichtbaren Objekten verstanden werden. Auf diese Weise kann die Position von einzelnen Pixeln über eine Sequenz von Bildern verfolgt werden.

Für jedes Pixel im Bild kann eine Geschwindigkeit angeben werden oder gleichbedeutend die Strecke die sich ein Pixel zwischen dem vorherigen Frame und dem jetzigen Frame bewegt hat. Diese wird üblicherweise als “Dense Optical Flow“ bezeichnet. Die Horn-Schunck-Methode [HS80] versucht so, ein Geschwindigkeitsfeld zu berechnen. Eine scheinbar sehr einfache Methode, bei der nur versucht werden muss, von Frame zu Frame die zusammengehörenden Pixel zu finden. In der Praxis ist die Berechnung des “Dense Optical Flow“ nicht so einfach. Das wird klar, wenn versucht wird, die Bewegung eines weißen Blattes zu verfolgen. Der Unterschied zwischen verschieden weißen Pixeln ist nicht erkennbar und somit auch nicht deren Bewegung. Nur Kanten verändern sich und auch nur die, die senkrecht zur Bewegungsrichtung liegen. Daraus folgt, dass eine Methode notwendig ist, die die Bewegung zwischen einfach zu trackenden Punkten interpoliert. Diese Schwierigkeit sorgt dafür, dass “Dense Optical Flow“-Methoden einen hohen Rechenaufwand erzeugen.

Das führt zu einer alternativen Methode, dem “Sparse Optical Flow“-Verfahren. Diese beruht darauf, dass bestimmte, vorher festgelegte Punkte verfolgt werden, vorausgesetzt, diese Punkte haben leicht wiedererkennbare Eigenschaften wie z.B. Ecken. Dann ist es einfach, diese Punkte zu verfolgen.

3.1.1 Lucas-Kanade-Methode

Der Lucas-Kanade-Methode (KLM) [LK81b], oft auch als Kanade-Lucas-Tracker (KLT) bezeichnet, wurde ursprünglich im Jahr 1982 vorgestellt und war eigentlich als “Dense Optical Flow“-

Methode gedacht. Heutzutage ist sie eine wichtige “Sparse Optical Flow“-Technik, da sie gut für einzelne Punkte funktioniert.

Der Lucas-Kanade-Algorithmus kann für “Sparse Optical Flow“ angewendet werden, weil er nur auf lokalen Informationen beruht, genauer gesagt, auf einem kleinen Fenster um jeden Punkt, der von Interesse ist. Der Nachteil der Benutzung von kleinen Fenstern ist, dass größere Bewegungen Punkte außerhalb des Fensters schieben können und es dadurch dem Algorithmus unmöglich ist, die Punkte wiederzufinden. Das Problem lässt sich lösen, indem eine Pyramide mit unterschiedlichen Auflösungen des Bildes genutzt wird, wobei das Tracking vom höchsten Level startet (geringste Auflösung) und sich herunterarbeitet bis zum untersten Level (größte Auflösung). Auf diese Weise ist es möglich auch große Bewegungen mithilfe von kleinen Fenstern zu erfassen.

Der Lucas-Kanade Algorithmus geht von 3 Grundannahmen aus:

- Konstante Helligkeit: Ein Pixel des Bildes eines Objekte darf sein Aussehen von Frame zu Frame nicht verändern. Für Grauwertbilder bedeutet das, dass die Helligkeit eines Pixels sich von Frame zu Frame nicht verändert.
- Nur geringe Bewegungen: Das bedeutet, dass die Zeit zwischen zwei Frames kurz genug ist, damit sich Objekte von Frame zu Frame nicht zu stark bewegen.
- Räumlicher Zusammenhang: Benachbarte Punkte, die zur selben Oberfläche gehören, bewegen sich auf die gleiche Weise.

Die konstante Helligkeit verlangt, dass ein zu trackendes Pixel über die Zeit gleich aussieht:

$$I(x(t), t) = I(x(t + \Delta t), t + \Delta t) \tag{3.1}$$

Wobei $I(x(t), t)$ die Intensität, also die Helligkeit, vom Pixel x zum Zeitpunkt t ist.

Die zweite Annahme wird zuerst im eindimensionalen Raums erläutert. Ausgehend von der Gleichung zur konstanten Helligkeit 3.1 lässt sich durch Abziehen der Definition der Helligkeit $f(x, t)$ und anschließendes Umstellen mithilfe der Regeln der partiellen Differentialrechnung, folgende Formel herleiten (für Details siehe [BK08]):

$$\underbrace{\frac{\delta I}{\delta x}\bigg|_t}_{I_x} \underbrace{\left(\frac{\delta x}{\delta t}\right)}_{v} + \underbrace{\frac{\delta I}{\delta t}\bigg|_{x(t)}}_{I_t} = 0 \tag{3.2}$$

Wobei I_x die räumliche Ableitung des ersten Bildes ist, I_t ist die Ableitung zwischen den Bildern über die Zeit und v ist die Geschwindigkeit, die gesucht wird. Umgestellt ist dies eine einfache Gleichung für die Geschwindigkeit des Optischen Fluss im eindimensionalen Fall:

$$v = -\frac{I_t}{I_x} \tag{3.3}$$

Ein Beispiel ist in Abbildung 3.1 zu sehen. Es wird eine “Kante“ dargestellt, bestehend aus einem hohen Wert auf der linken Seite und einem niedrigen Wert auf der rechten Seite. Dann

bewegt sich die Kante entlang der x Achse. Das Ziel ist es jetzt festzustellen, wie hoch die Geschwindigkeit v der sich bewegenden Kante ist (dargestellt im oberen Bereich). Im unteren Teil ist zu erkennen, dass das Maß für die Geschwindigkeit die Zeit durch die Steigung ist.

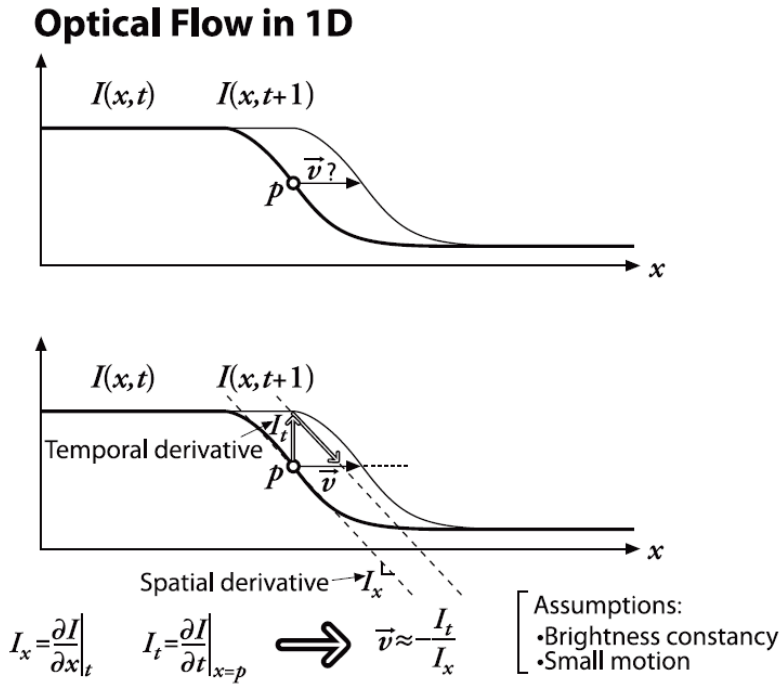


Abbildung 3.1: Lucas-Kanade Optischer Fluss in einer Dimension. Quelle: [BK08].

Ausgehend von der eindimensionalen Lösung kann nun die die zweidimensionale Lösung entwickelt werden, indem die y Komponente der Geschwindigkeit v und die x Komponente der Geschwindigkeit u genannt werden.

$$I_x u + I_y v + I_t = 0 \tag{3.4}$$

Allerdings hat diese Gleichung zwei Unbekannt für jedes gegebene Pixel. Das bedeutet, dass die Gleichung unterbestimmt ist und nicht genutzt werden kann, um eine eindeutige Lösung für zweidimensionale Bewegungen zu ermitteln. Stattdessen kann lediglich die senkrechte oder "normal" zur Kante liegende Komponente des Verschiebungsvektors bestimmt werden. Diese wird auch als Normale Optischer Fluss[1] bezeichnet. Abbildung 3.2 beschreibt die mathematischen und geometrischen Details.

Der Normale Optische Fluss leidet am so genannten Blendenproblem, das auftritt, wenn die Blenden oder Fenster im Verhältnis zur Bewegung zu klein sind. Wenn Bewegungen mit kleinen Fenstern bestimmt werden, sind oft nur eine Kanten sichtbar und keine Ecken. Aber mithilfe von Kanten allein ist es nicht möglich, die exakte Bewegung zu bestimmen. Dies wird in Abbildung 3.3 deutlich. In den oberen 4 Bildern wird die Bewegung durch die Blende gezeigt. Die Kante bewegt sich jeweils nur vorwärts. In den unteren 4 Bildern wird die Bewegung hinter der

[1] Normale von Normalenvektor, in der Geometrie ist ein Normalenvektor ein Vektor, der senkrecht (orthogonal) auf einer Geraden, Kurve, Ebene, (gekrümmten) Fläche oder einer höherdimensionalen Verallgemeinerung eines solchen Objekts steht.

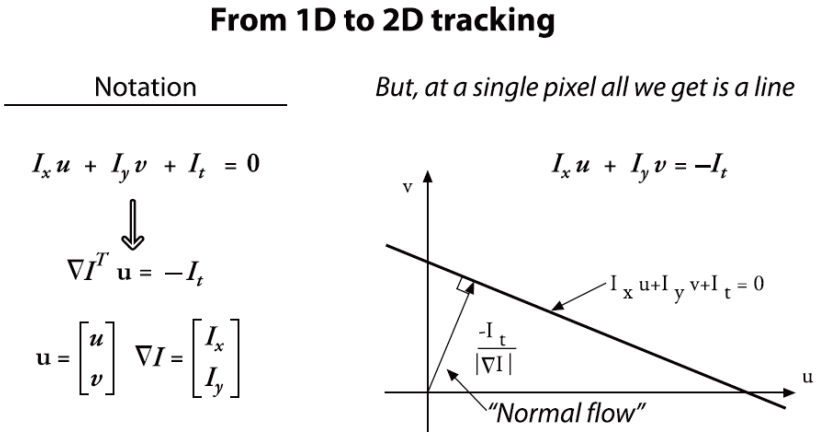


Abbildung 3.2: Zweidimensionaler Optischer Fluss an einem Pixel. Quelle: [BK08].

Blende gezeigt. Es wird klar, dass es nicht nur eine Vorwärtsbewegung gibt, sondern auch eine Abwärtsbewegung. Diese wird aber hinter der Blende nicht erkannt.

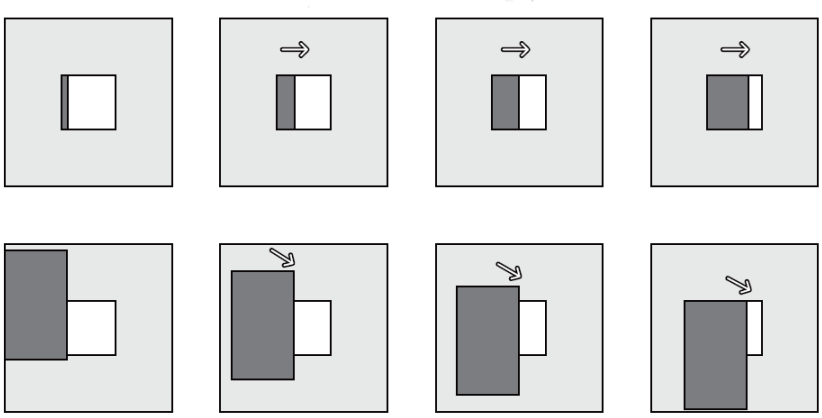


Abbildung 3.3: Blendenproblem. Quelle: [BK08]

Dies kann gelöst werden, indem die Pixel, die das Hauptpixel umgeben, genutzt werden, um ein Gleichungssystem aufzustellen. Zum Beispiel könnte ein 5×5-Fenster von Pixeln genutzt werden, woraus ein Set von 25 Gleichungen aufgestellt werden kann.

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A \quad 25x2} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d \ 2x1} = \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b \quad 25x1} \tag{3.5}$$

Mit diesem überbestimmten Gleichungssystem ist es nun möglich, die genaue Bewegung zu bestimmen, vorausgesetzt, es befindet sich nicht nur eine Kante innerhalb des 5×5-Fenster. Zu Lösung des Gleichungssystems kann nun die Methode der kleinsten Quadrate genutzt werden, wobei $min||Ad - b||^2$ wie folgt gelöst werden kann:

$$(A^T A)d = A^T b \tag{3.6}$$

Wenn u und v Komponenten des Verschiebungsvektors sind, kann geschrieben werden:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b} \tag{3.7}$$

Die Lösung dieser Gleichung ist dann:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b \tag{3.8}$$

3.2 Harris Corner Detector [HS88]

Die von Harris und Stevens in [HS88] vorgestellte Methode findet Merkmale in Bildern, indem sie nach Ecken sucht. Wenn ein Bild durch ein kleines “Fenster“ hindurch betrachtet wird, das Bild also bis auf einen kleinen Ausschnitt abdeckt wird, können drei Beobachtungen gemacht werden:

- Befindet sich das Fenster über einer nahezu homogenen Fläche, so wirkt sich eine leichte Verschiebung gar nicht oder nur minimal auf die Intensität aus.
- Befindet sich das Fenster über einer Kante, wirken sich die Verschiebungen entlang der Kante nur in geringem Maße auf die Intensität aus, wohingegen eine Verschiebung entgegen der Kante zu einer großen Veränderung führt.
- Befindet sich das Fenster über einer Ecke oder einem isolierten Punkt, dann ergibt sich bei jeder Verschiebung eine signifikante Veränderung der Intensität.

Dies kann mathematisch wie folgt dargestellt werden:

$$E_{x,y} = \sum_{u,v} w_{u,v} |I_{x+u,y+v} - I_{u,v}|^2 \tag{3.9}$$

E gibt die Stärke der Intensitätsveränderung an. Die Funktion der Intensität wird als I bezeichnet. Diese Funktion bildet einen Punkt im Bild auf seinen Grauwert ab. Das Fenster, durch das der Bildausschnitt betrachtet wird, ist mit w bezeichnet. Die Verschiebungen in x, y werden in Hinblick auf die Richtungen [(1, 0), (1, 1), (0, 1), (-1, 1)] betrachtet.

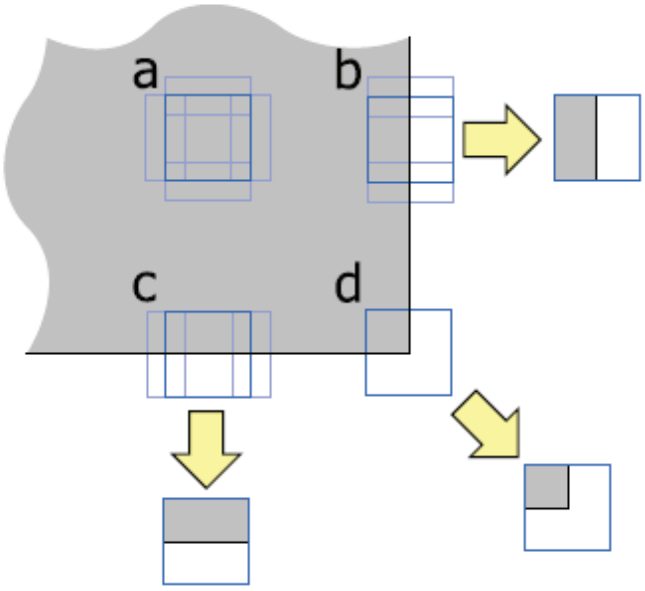


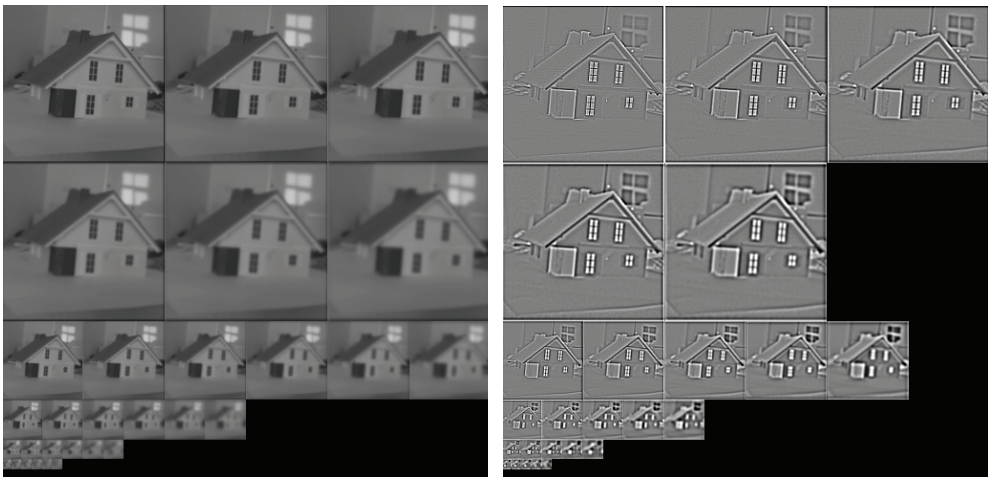
Abbildung 3.4: Das Prinzip der Corner Detection. Das Fenster a liegt weder auf einer Kante, noch auf einer Ecke. Es kann in beide Richtungen verschoben werden, ohne dass sich die Intensität ändert. Die Fenster b und c liegen auf einer Kante. Sie lassen sich nur noch in eine Richtung bewegen, ohne dass sich ihre Intensität ändert. Fenster d liegt auf einer Ecke und lässt sich nicht bewegen, ohne dass sich die Intensität ändert. Quelle: [Esc06].

3.3 SIFT [Low04]

Scale Invariant Feature Transform (SIFT) von Lowe ist ein Verfahren, das Merkmale unabhängig von ihrer Skalierung und Orientierung findet bzw. zueinander zuordnet. Genau genommen beschreibt SIFT dabei einen Difference of Gaussians (DoG) Detektor und einen eigenen Deskriptor. Der SIFT-Deskriptor, der die gefundenen Merkmale beschreibt und Grundlage des Vergleichs ist, wird im nächsten Abschnitt näher erläutert. Interessenpunkte findet der Detektor an Extremstellen im differentiellen Skalenraum eines Bildes. Der Skalenraum wird erstellt, indem das Bild $\mathbf{I}(\mathbf{x}, \mathbf{y})$ mit einem Gaußfilter $\mathbf{G}(\mathbf{x}, \mathbf{y}, \sigma\ )$ gefaltet wird.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.10}$$

Zuerst wird das Bild in verschiedenen Auflösungen erstellt, die, zusammen übereinander gelagert, der Struktur einer Pyramide ähneln. Abbildung 3.5.a zeigt die Ebenen einer Pyramide für ein Kamera- und synthetisches Bild. Auf jeder dieser Ebenen der Pyramide, also in einer bestimmten Bildauflösung, wird nun das Bild mehrmals mit konstantem Abstand k in Richtung σ inkrementell mit einem Gaußfilter gefaltet. Diese Ebenen werden Oktave genannt. Die Differenz zwischen zwei benachbarten Ebenen innerhalb einer Oktaven, daher auch Difference of Gaussians (DoG), bildet den differentiellen Skalenraum. (Abbildung 3.5.b). DoG ist eine Approximation des Laplacian-of-Gaussian-Operators (LoG) $\sigma\Delta^2 G$, der sehr stabile, invariante Merkmale



(a) Gaußpyramiden (b) DoG

Abbildung 3.5: Gaußpyramiden und die dazugehörigen DoG Bilder.

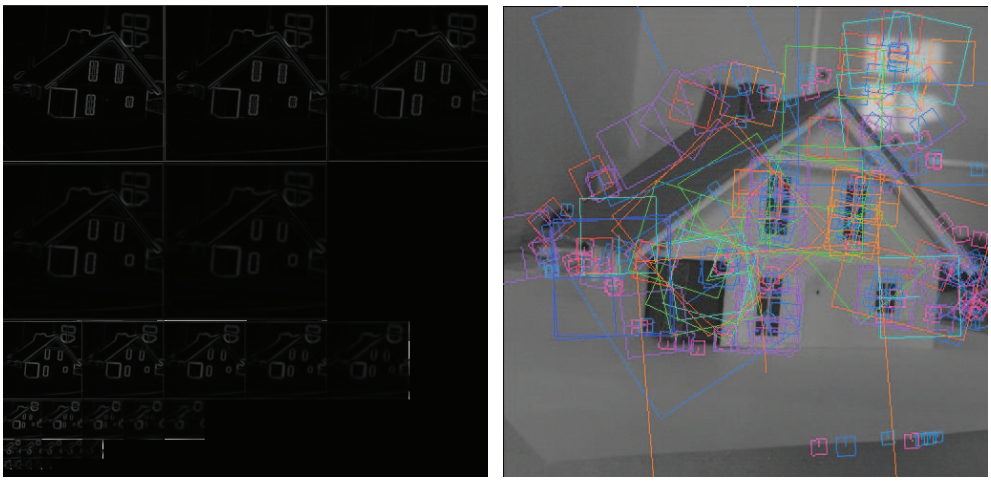
findet, wenn die Normalisierung durch den Skalenparameter σ^2 verwendet wird.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{3.11}$$
$$= L(x, y, k\sigma) - L(x, y, \sigma) \tag{3.12}$$
$$\approx ((k-1)\sigma\Delta^2 G) * I(x, y) \tag{3.13}$$

Die Extremstellen dieses differentiellen Skalenraums, die lokalen Maxima und Minima, werden als Merkmalskandidaten angesehen, wenn die Punkte größer oder kleiner als alle Punkte in ihrere Nachbarschaft sind. Die Nachbarschaft umfasst die umliegenden 8 Punkte auf derselben Ebene innerhalb der Oktave sowie jeweils 9 Punkte auf der nächsten und vorherigen Ebene. Die Merkmalskandidaten werden anschließend näher betrachtet, um ungeeignete Kandidaten auszuschließen.

Es wird die genauere Subpixelposition des Kandidaten bestimmt, außerdem der Kontrast dieses Extremes durch die Differenz mit der Nachbarschaft. Liegt der Kontrast unter einem bestimmten Schwellwert, wäre es sinnvoll, dieses Extrem nicht weiter zu berücksichtigen. Zusätzlich müssen diejenigen Kantenpixel verworfen werden, deren Position auf der Kante durch Rauschen nicht eindeutig bestimmt werden kann. Dazu wird die 2x2 Hesse-Matrix dieses Kandidaten bestimmt, dabei muss das Verhältnis des größeren Eigenwerts zum kleineren unter einem bestimmten Schwellwert liegen.

Die Orientierung eines Merkmals relativ zum Bild wird durch seine Nachbarschaft bestimmt. Gewichtet mit der Steigung (3.6.a) und einem Gauß-gewichteten Fenster werden die Nachbarschaftspunkte in ein Histogramm eingebracht, das auf 36 mögliche Orientierungen in 10 Grad Schritten aufgeteilt ist. Diejenige Orientierung mit dem größten Gewicht bestimmt letztendlich die Orientierung des Merkmals. Abbildung 3.6.b zeigt durch Quadrate markierte Merkmale, wobei die Orientierung der Quadrate der Orientierung der Merkmale entspricht.



(a) Gradient der Pyramide

(b) Quadrate markieren Merkmale und beschreiben Orientierung und Skalierung

Abbildung 3.6: Gradienten sowie erkannte Merkmale.

Die Bilder sowie der überwiegende Teil der Beschreibung von SIFT wurden aus [Hab09] übernommen.

3.4 Speeded Up Rebust Features [BTGL06]

Speeded Up Rebust Features (SURF) ist wie SIFT eine Methode, um Featurepunkte aus Bildern zu berechnen und diese Punkte zu beschreiben. SURF ersetzt die in SIFT verwendeten Gaußfilter durch Mittelwertfilter, welche durch die Verwendung von Integralbildern mit konstantem Zeitaufwand berechnet werden können. Ziel dieses Vorgehens ist es, die Berechnung zu beschleunigen. Nach Aussagen des Autors ist SURF siebenmal schneller als SIFT. Mittlerweile existieren mehrere Implementationen sowohl auf der CPU als auch auf der GPU, die alle auf demselben bzw. einem ähnlichem Algorithmus basieren. Genau wie SIFT bietet SURF eine hohe Wiedererkennungsrate sowie Skalierungs und die Rotationsinvarianz. Der folgende Abschnitt basiert auf der Veröffentlichung [BTGL06].

Die Detektion des SURF Algorithmus basiert auf der Hessematrix H, genauer gesagt auf deren Determinante. Die Merkmalsextraktion durch die Hessematrix ist unempfindlich gegenüber Kanten und bietet einen hohen Grad an Genauigkeit. Die Hessematrix für einen Punkt $x_i = (x, y)$ mit der Skalierung σ ist definiert als:

$$H = \begin{bmatrix} L_{xx}(x_i, \sigma) L_{xy}(x_i, \sigma) \\ L_{xy}(x_i, \sigma) L_{yy}(x_i, \sigma) \end{bmatrix} \tag{3.14}$$

wobei $L_{xx}(x_i, \sigma)$ der Faltung zwischen dem Bildpunkt x_i und der zweiten Ableitung der Gauß-Funktion $\frac{\delta^2}{\delta x^2} g(\sigma)$ entspricht. Dies gilt analog für $L_{xy}(x_i, \sigma)$) und $L_{yy}(x_i, \sigma)$. Die Berechnung der Determinante kann aufgrund der Symmetrie der Matrix durch

$$det(H(x_i, \sigma)) = H_{11}H_{22} - H_{12}^2 \tag{3.15}$$

erfolgen. Da diese Funktion in der Praxis diskret sein muss, wird eine Approximation der Gleichung 3.14 verwendet. Die diskrete Approximation kann als Mittelwertfilter aufgefasst werden.

Hierfür nutzt [BTGL06] Integralbilder. Ein Integralbild dient der schnellen Berechnung von Pixelsummen innerhalb rechteckiger Ausschnitte von Bildern. Der Begriff ist abgeleitet von dem Konzept der diskreten Integration. In jedem Punkt des Integralbildes steht die Summe aller Pixel innerhalb des Rechtecks zwischen dem aktuellen Punkt (x, y) und dem Ursprung $(0, 0)$ des Bildes. In Punkt (x, y) steht also die Summe $I_{\sum}$ der Pixel innerhalb des Rechtecks, das von den Punkten (0,0), (x,0), (0,y) und (x,y) aufgespannt wird.

$$\sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i, j) \tag{3.16}$$

Integralbilder erlauben es, dass die Berechnung der Summe von Intensitätswerten über einem beliebig großen rechteckigen Bereich mit nur vier Operationen vorgenommen werden kann.

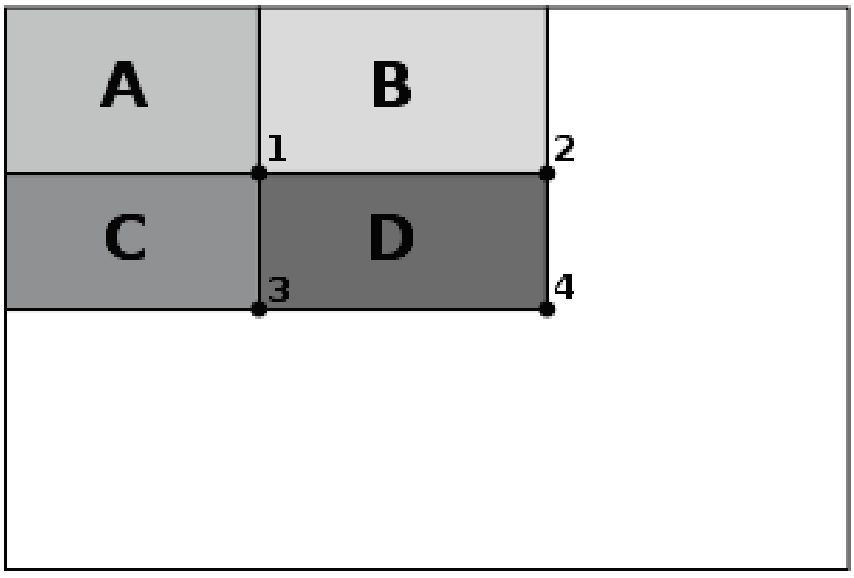


Abbildung 3.7: Beispiel Integralbilder.

Die Pixelsumme der Fläche D (siehe 3.7) berechnet sich wie folgt:

$$I_{\sum FlcheD} = I_{\sum}(P_1) + I_{\sum}(P_4) - (I_{\sum}(P_2) + I_{\sum}(P_3)) \tag{3.17}$$

Diese Eigenschaft macht sich [BTGL06] zu Nutze, indem er vergleichsweise große Mittelwertfilter nutzt.

3.5 Deskriptor

Ein Deskriptor ist ein Vektor, dessen Aufgabe es ist, die Umgebung eines gefunden Featurepunktes eindeutig zu beschreiben, sodass es möglich ist, diesen Punkt in anderen Bildern wieder zu finden. Ein guter Deskriptor ist gegen Rotation und unterschiedliche Beleuchtung invariant. Hier wird die Technik des SIFT-Deskriptors und des SURF-Deskriptors vorgestellt.

3.5.1 SIFT-Deskriptor

Wie in Abschnitt 3.3 beschrieben, wird die Orientierung eines Featurepunktes relativ zum Bild durch seine Nachbarschaft bestimmt. Anhand der berechneten Orientierung wird der Diskriptor ausgerichtet, um invariant gegenüber Rotation zu sein. Als Erstes werden der Gradientenwert und die Gradientenorientierung an jedem Bildpunkt in der Umgebung des Featurepunktes berechnet(siehe Abbildung 3.8). Um den Featurepunkt herum werden die Gradienten mit einer kreisförmigen Gaußmaske gewichtet, um den Einfluss der Gradienten vom Abstand abhängig zu machen und somit weit entfernte Punkte weniger zu gewichten. Die Werte eines 4×4-Teilbereiches werden dann in einem Orientierungshistogramm addiert, wie in 3.8 zu sehen ist.

Das Orientierungshistogramm besteht aus 8 möglichen Richtungen, die auf 360° abgebildet werden. Die Gradientenwerte werden anhand ihrer Orientierung auf diese 8 Richtungen verteilt. Insgesamt setzt sich der Diskriptor aus 16 Teilbereichen zusammen. Die Histogrammwerte dieser 16 Teilbereiche werden in einem Vektor aus 128 Werten gespeichert (16 Teilbereiche mit 8 Werten je Histogramm). Mithilfe des Diskriptors lässt sich jetzt bestimmen, ob es sich bei zwei Punkten aus unterschiedlichen Bildern um die gleichen Punkte handelt, indem der nächste Nachbar[2] gesucht wird, also die beiden Punkte, die sich am ähnlichsten sind (siehe hierzu auch 6.7).

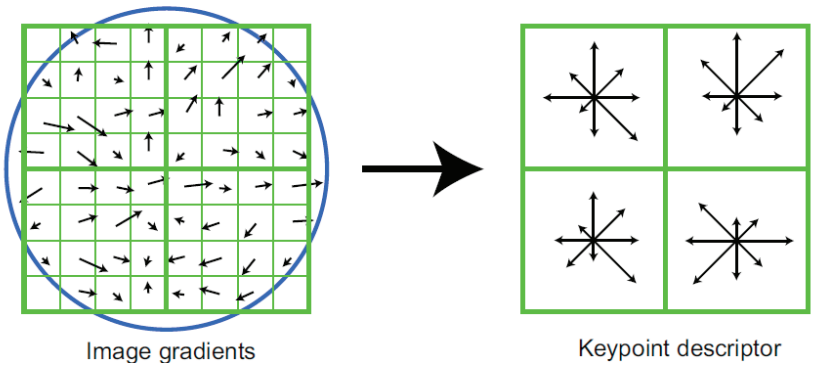


Abbildung 3.8: SIFT-Deskriptor, links sind die Gradientenwerte jedes Pixels zu sehen, auf der rechten Seite sind die addierten Gradientenwerte eines 4×4-Feldes zu sehen. Quelle: [Low04].

[2] Als nächsten Nachbarn (eng. nearest neighbor) des Vektors v wird der Vektor aus der Menge M bezeichnet, der zum Vektor v den geringsten Abstand hat

Entscheidend bei der Festlegung verschiedener Features ist, dass diese Features eindeutig beschreibbar sind, sodass es möglich ist, den gleichen Punkt in zwei oder mehreren Bildern wiederzufinden.

3.5.2 SURF-Deskriptor

Zur Berechnung des SURF-Deskriptors wird im ersten Schritt die dominante Orientierung im Umkreis eines gefunden Featurepunktes bestimmt. Dies geschieht, indem das Haar-Wavelets[3] in x und y Richtung berechnet wird. Diese wird mit der Gauß-Funktion gewichtet. Danach wird die dominante Orientierung geschätzt, indem die x- und y-Antworten des Wavelets entsprechend der Winkel summiert werden. Dabei werden jeweils Winkel in $\frac{3}{\pi}$-Bereichen zusammengefasst. Der Maximalwert wird dann als Orientierung des Deskriptors festgelegt.

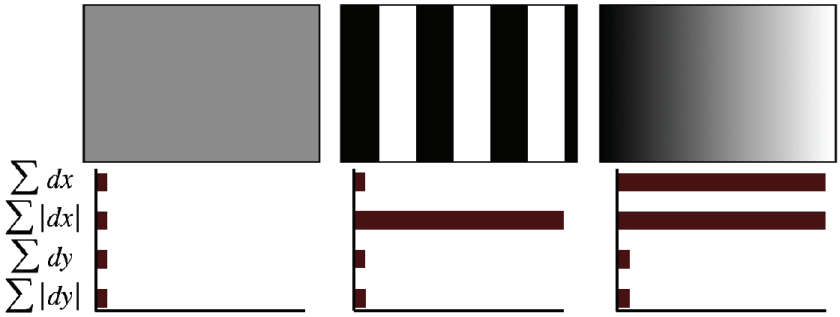


Abbildung 3.9: Auswertung bzw. Interpretation der Haar-Wavelet-Antworten. Von links nach rechts: Links, bei homogenen Flächen ist die Summe verhältnismäßig klein. Mitte, bei einer alternierenden Abbildung in x Richtung, ist die Summe der Beträge in x Richtung hoch ($|dx|$). Rechts, wenn die Intensität langsam ansteigt, sind sowohl Betrag $|dx|$ als auch Summe dx hoch. Quelle: [BTGL06].

In einem weiteren Schritt wird ein Quadrat um das betrachtete Feature definiert, dies hat die im letzten Schritt berechnete Orientierung. Das bestimmte Quadrat wird in 4×4 gleich große Regionen aufgeteilt. In jeder Sub-Region wird das Haar-Wavelet berechnet. Zum Schluss werden die Wavelet Antworten aus jeder Region in horizontal dx und vertikal dy Richtung aufsummiert. Außerdem werden die Absolutwerte jeder Region addiert: $|dx|$ und $|dy|$. Jede Sub-Region wird damit beschrieben durch:

$$v = (\sum_i dx_i \sum_i dy_i \sum_i |dx_i| \sum_i |dy_i|) \tag{3.18}$$

[3] Mit Wavelet-Transformation wird eine bestimmte Familie von linearen Zeit-Frequenz-Transformationen in der Mathematik bezeichnet. Das Haar-Wavelet ist das erste in der Literatur bekannt gewordene Wavelet und wurde 1909 von Alfred Haar vorgeschlagen. Es hat eine Rechteckform.

3.6 Real-time 3D Object Pose Estimation and Tracking for Natural Landmark

Changhyun Choi, Seung-Min Baek und Sukhan Lee schlagen in [CBL08] eine Methode vor, die SIFT mit Optischen Fluss basierten Tracking kombiniert. Changhyun Choi, Seung-Min Baek und Sukhan Lee stellen das Verfahren in einer Mono- und einer Stereovariante vor. Da das in dieser Arbeit entwickelte Framework nur den Monobetrieb vorsieht, wird auch nur diese Variante hier beschieben.

Wie in 3.1.1 schon beschrieben, besteht der Vorteil eines KLM-Trackers darin, dass er wenig Rechenkraft benötigt. Allerdings kann er nicht mit einer signifikanten Änderungen der Szene, starker Änderung der Beleuchtung und Verdeckungen umgehen. Außerdem können sich Fehler über die Zeit so stark akkumulieren, dass es zu einem Abdriften der Punkte kommt.

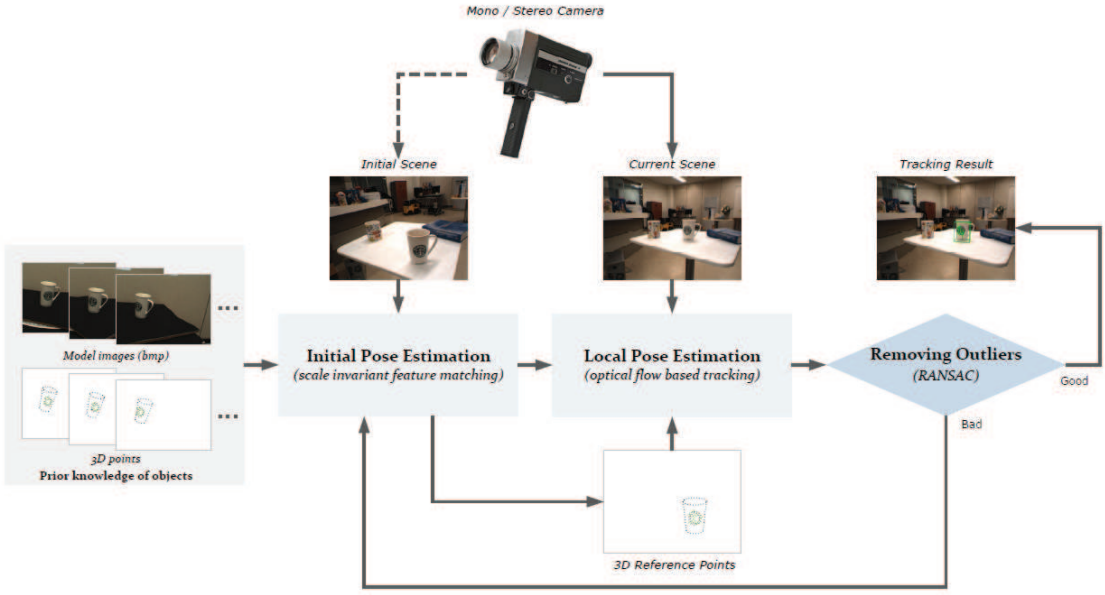


Abbildung 3.10: Systemübersicht. Quelle: [CBL08].

In Abbildung 3.10 wird der Systemaufbau von Changhyun Choi, Seung-Min Baek und Sukhan Lee dargestellt. Die Methode ist in zwei Teile aufgeteilt: “inital pose estimation“ und der “local pose estimation“. Die Offlineinformationen werden in “inital pose estimation“ zusammengefasst. Diese Informationen, also SIFT-Punkte eines Objektes und die zugehörige 3D-Position, stammen von einem Bild des Objektes und von einem mittels Streifenprojektion[4] erzeugten Modell. Sobald die initalie Position bestimmt ist erstellt das System automatisch KLM Tracking-Punkte an den Stellen, an denen SIFT-Punkte gefunden wurden. Zusätzlich werden die dazugehörigen 3D-Referenzpunkte gespeichert. Diese werden spätere genutzt, um die 3D-Position zu bestimmen.

[4] Das Streifenprojektionsverfahren, fasst eine Gruppe optischer Messmethoden zusammen, bei der Bildsequenzen zur dreidimensionalen Erfassung von Oberflächen verwendet werden.

Algorithm 1 MONO 3D POSE TRACKING

Input: model images $I_{1:k}(x, y)$, 3D model points $J^M_{1:k}(x, y)$, a current scene image $S(x, y)$

Initialize:

1) Extract m SIFT feature points $\mathbf{p}^C_{1:m}$ globally on the $S(x, y)$ and get m 3D model points $\mathbf{P}^M_{1:m}$ corresponding to the extracted SIFT feature points by looking up $J^M_{1:k}(x, y)$
2) Determine the initial pose $\mathbf{\Phi}^C_0$ of the object by using POSIT algorithm in which $\mathbf{p}^C_{1:m}$ and $\mathbf{P}^M_{1:m}$ are served as parameters
3) Generate n 2D KLT tracking points $\mathbf{t}^C_{1:n}$ within the convex hull of a set of $\mathbf{p}^C_{1:m}$
4) Calculate n 3D reference points $\mathbf{T}^M_{1:n}$ corresponding to $\mathbf{t}^C_{1:n}$ by using the plane approximation based on $\mathbf{t}^C_{1:n}$, $\mathbf{P}^M_{1:m}$, and $\mathbf{\Phi}^C_0$

Loop:

1) Track $\mathbf{t}^C_{1:n}$ by using KLT tracker
2) Remove outliers by applying RANSAC to produce $\mathbf{t}^C_{1:n'}$ and $\mathbf{T}^M_{1:n'}$
3) If the number of remained tracking points n' is fewer than τ_{mono}, go to **Initialize**
4) Determine the current pose $\mathbf{\Phi}^C$ of the object by using POSIT algorithm in which $\mathbf{t}^C_{1:n'}$ and $\mathbf{T}^M_{1:n'}$ are served as parameters.

Output: the pose $\mathbf{\Phi}^C$ of the object

Abbildung 3.11: Der von Changhyun Choi, Seung-Min Baek und Sukhan Lee vorgestellt Algorithmus zur Positions bestimmung. Quelle: [CBL08]

Sobald eine Initialisierung im “inital pose estimation“ stattgefunden hat, wird der “local pose estimation“-Prozess ausgeführt. Innerhalb des “local pose estimation“-Prozesses, verfolgt das System die 3D-Position des Zielobjektes mithilfe der 3D-Referenzpunkte und der KLM Tracking-Punkte. Ausreißer werden mittels RANSAC-Algorithmus (siehe 4.6) aussortiert. Das wird wiederholt, bis ein vorher festgelegter Schwellwert für die Anzahl der Trackingpunkte nicht unterschritten ist. Sobald der Schwellwert unterschritten ist, wird das ganze System neu gestartet, indem “inital pose estimation“ erneut ausgeführt wird. In Abbildung 3.11 wird der Algorithmus in Pseudocode dargestellt.

Die Ergebnisse von Changhyun Choi, Seung-Min Baek und Sukhan Lee zeigen deutlich, dass das System nach der Initialisierung mit sehr geringen Systemressourcen auskommt. Die durchschnittliche Rechenzeit pro Bild liegt bei 1 ms auf einem Pentium 4, 2.4 GHz-Rechner. Allerdings braucht die Initialisierung mit knapp 300 ms deutlich länger. Das kann zu starkem Ruckeln führen, wenn öfter neu initialisiert werden muss. Um die Initialisierungszeit zu verkürzen, wird vorgeschlagen, anstelle des klassischen SIFT ein anderes Verfahren zu nutzen z.B. wär es möglich SURF zu nutzten das eine deutlich höhere Performance hat.

3.7 Punkt Matsching

Unter Matching wird der Vergleich von Featurepunkten verstanden mit dem Ziel, Paare von gleichen Punkten zu finden. Der klassische Ansatz hierfür ist die Berechnung des geometrischen Abstandes der Deskriptoren[5]. Je geringer die Distanz zwischen zwei Deskriptoren ist umso ähnlicher sind sie sich. Das wohl bekannteste Distanzmaß für Vektoren ist der Euklidische Abstand $d(x, x')$, auch Euklidische Norm $||x - x'||_2$ des Differenzvektors genannt. Für den Euklidischen Abstand zwischen zwei n-dimensionalen Vektoren x und x' gilt:

$$d(x_1, x_2) = ||x - x'||_2 = \sqrt{x_1 - x'_1 + ... + x_n - x'_n} \tag{3.19}$$

Mithilfe des Euklidischen Abstandes lässt sich der nächster Nachbar zwischen einem Deskriptor d_1 und einer Menge von Deskriptoren $D' = (d'_1, ..., d'_n)$ bestimmen.

Der klassische Ansatz hierfür ist die lineare Suche. Dabei wird der Abstand von allen Punkten aus der Menge D' zum Deskriptor d_1 berechnet und anschließend verglichen. Der Deskriptor mit dem geringsten Abstand ist der nächster Nachbar. Um die Wahrscheinlichkeit zu erhöhen dass es sich beim nächsten Nachbarn wirklich um den gleiche Punkt handelt, wird der nächste Nachbar mit dem Deskriptor verglichen, der den zweitgeringsten Abstand hat. Ist der Abstandsunterschied zwischen den beiden Deskriptoren nicht groß genug, wird das Ergebnis als nicht eindeutig angesehen und verworfen.

Problematisch an dem Verfahren der linearen Suche ist, dass es eine Komplexität von $O(n^2)$ aufweist. In der Praxis bedeutet das, dass bei zwei Bildern, auf denen jeweils 1000-SIFT Punkte gefunden wurden, $1000 * 1000 * 128$ (Größe des SIFT Deskriptor) $= 12.800.000$ Rechenschritte durchgeführt werden müssen. Zur Optimierung des Verfahrens können Abbruchbedingungen eingeführt werden, wenn z.B. der Abstand größer ist als ein zuvor gefundener Abstand, kann der Vergleich gestoppt werden. Des Weiteren kann der Vergleich abgebrochen werden, wenn identische Merkmale vorliegen, der Abstand der beiden Deskriptoren also null ist.

3.7.1 Nächster Nachbar-Suche mit kd-Bäumen

Die lineare Suche nach dem nächsten Nachbarn, wie sie im letzten Abschnitt vorgestellt wurde, hat durch ihre hohe Komplexität von $O(n^2)$ eine sehr schlechte Performance. Um diese zu optimieren, können kd-Bäume genutzt werden. K-d-Bäume sind eine Erweiterung von binären Bäumen. Sie dienen zur strukturierten Speicherung von k-dimensionalen Elementen. Jeder Knoten des Baumes beschreibt einen Teil des k-d-Baumes und hat zwei Verweise auf Nachfolgeknoten. In der homogenen Variante enthält jeder Baumknoten einen Vektor und zwei Zeiger auf den linken und rechten Sohn. In der inhomogenen Variante enthält jeder Baumknoten nur einen Schlüssel. In beiden Fällen werden die Werte der einzelnen Attribute abwechselnd auf jeder Ebene des Baumes zur Aufteilung verwendet. Der Wert, der entscheidet, ob ein Element zum linken oder rechten Nachfolgeknoten gehört, entspricht dem Median der einzelnen Attribute.

[5] Nicht zu verwechseln mit dem Abstand der Position der zum Diskriptor gehörenden Punkte(siehe auch 3.5)

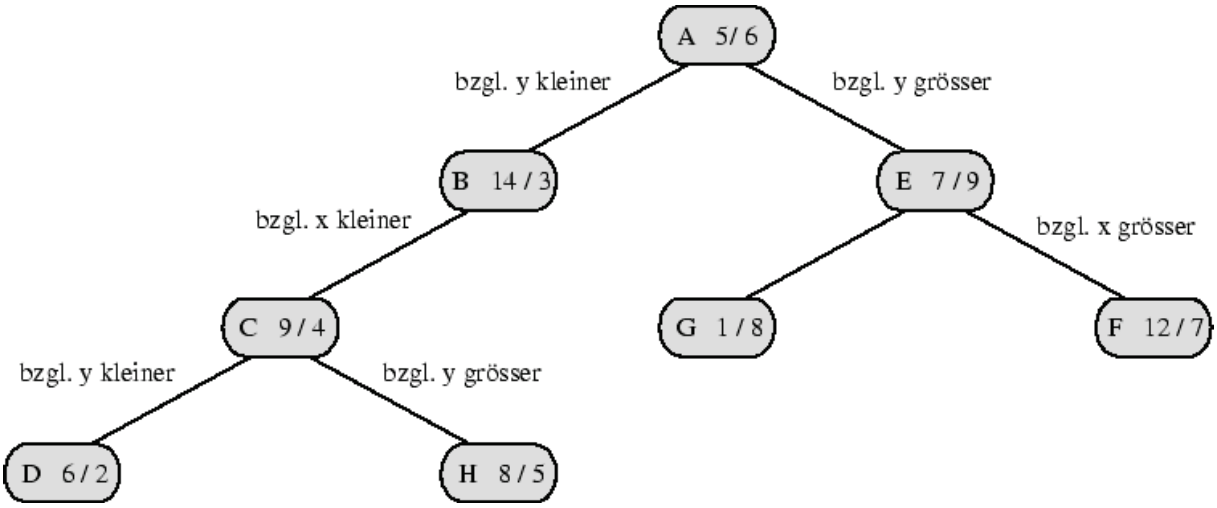


Abbildung 3.12: 2-d-Baum (homogen) zu den Datenpunkten A,B,C,D,E,F,G,H. Quelle: [Bau10].

In Abbildung 3.12 wird ein homogener kd-Baum dargestellt. Jeder Baumknoten ist ein Vektor mit zwei Zeigern auf den linken und rechten Sohn. Auf der linken Seite eines Knotens befinden sich alle Vektoren, die kleiner in Bezug auf ein Attribut sind, auf der rechten alle, die größer sind. Das Attribut sind jeweils Zeilen der Vektoren, in diesem Fall immer der x- oder y-Wert der Vektoren. Der homogene kd-Baum hat den Nachteil, dass es kein Balancierungskriterium gibt und somit im extrem Fall alle Punkte nur auf einer Seite abgebildet werden und somit im Wesentlichen eine lineare Struktur bilden.

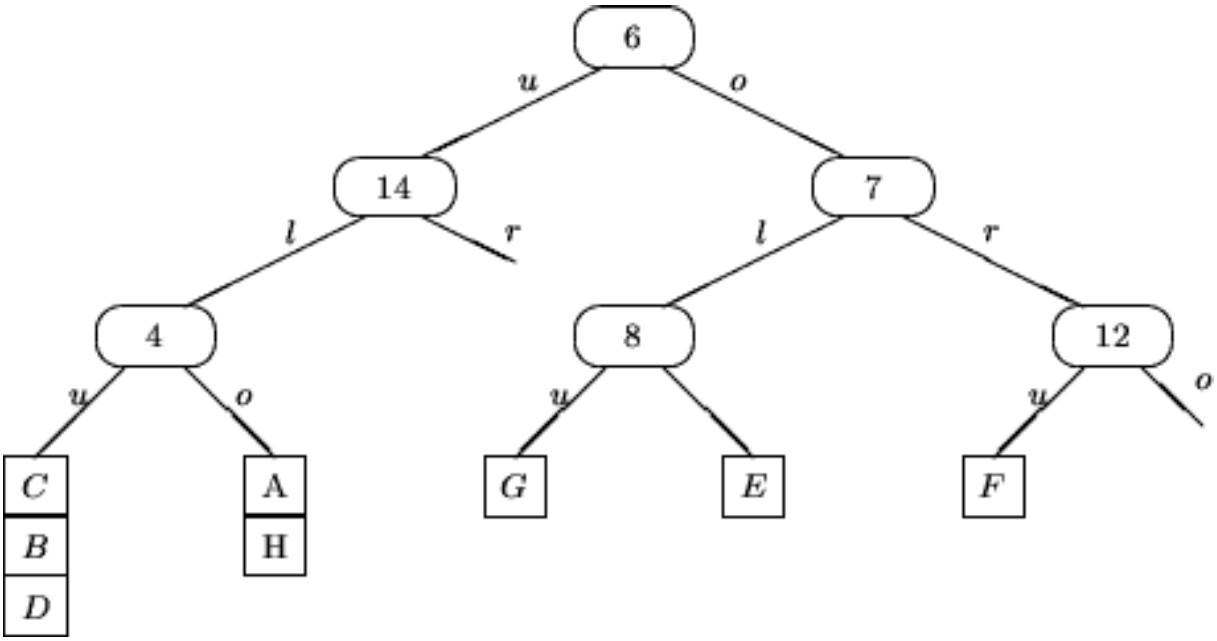


Abbildung 3.13: 2-d-Baum (inhomogen) zu den Datenpunkten A,B,C,D,E,F,G,H. Quelle: [Bau10].

In der Abbildung 3.13 ist ein inhomogener kd-Baum zu sehen. Die Knotenwerte sind Schlüssel, die durch Berechnung des Median[6] bestimmt werden. Der Median wird gewählt, damit die

[6]Der Median bezeichnet eine Grenze zwischen zwei Hälften. Er teile eine Menge M in zwei gleich große Mengen

beiden Nachfolgeknoten immer ungefähr die gleiche Anzahl an Elemente enthalten. Der Median bezieht sich hierbei auf ein Vektorattribut (in der Abbildung die x- oder y-Zeile) von allen Vektoren, die sich unterhalb dieses Knotens befinden. Welches Attribut jeweils ausgewählt wird, um die Elemente aufzuteilen, hängt davon ab, in welcher Dimension die Elemente den größten Wertebereich aufspannen (die größte Varianz haben).

Die Vektoren werden in jedem Knoten jeweils gesplittet. Auf der linken Seite befinden sich alle Vektoren, deren Attribut kleiner ist als das Median und auf der rechten Seite deren Attribut größer ist. Dieses Splitten wird so lange weiter durchgeführt, bis nur noch maximal 2 Vektoren unterhalb eines Knotens vorhanden sind. Diese beiden Vektoren werden dann in den Blättern gespeichert. Es gibt auch noch eine Abwandlung des homogenen Baumes, bei der der Baum nur eine vorher festgelegte Tiefe hat und sich somit mehr als 2 Blätter unterhalb der untersten Konten befindet.

Die klassische Suche nach dem nächsten Nachbarn mithilfe von inhomogenen kd-Bäumen funktioniert wie folgt: Die Suche nach dem nächsten Nachbarn eines Punkts p beginnt in der Wurzel. Zu Beginn befindet sich der Abstand zum nächsten Nachbarn im Unendlichen. Für jeden Knoten wird zunächst unterschieden, ob er ein Blattknoten ist oder nicht. Ist ein Knoten kein Blattknoten, so wird überprüft, zu welchem Nachfolgeknoten p er gehört. Es wird also ermittelt ob das Attribut kleiner oder größer ist als der Wert, der im aktuellen Knoten zum Aufteilen verwendet wurde. Ist der Wert kleiner, erfolgt die Suche rekursiv im linken Nachfolgeknoten, ansonsten im rechten. Falls die Suche einen Blattknoten überprüft, so wird in diesem nach dem nächsten Nachbarn gesucht, d.h., es wird überprüft, ob ein Element näher an p liegt als das nächste bisher gefundene.

Sobald ein Rekursionsschritt beendet wurde, ist es noch notwendig zu überprüfen, ob der nächste Nachbar nicht doch im anderen Nachfolgeknoten liegen kann, dem so genannten “backtracking“. Dazu wird eine k-dimensionale Kugel mit dem Radius der Entfernung von p zum aktuell nächsten Nachbarn und Punkt p betrachtet. Schneidet diese Kugel die Region des anderen Nachfolgeknotens nicht, so kann dieser kein Element enthalten, das näher an p liegt und muss folglich nicht weiter betrachtet werden. Ansonsten erfolgt die Suche rekursiv in diesem Ast.

Silpa-Anan und Hartley empfehlen in [SAH08] für das backtracking die “priority search“ zu nutzen. Hierbei wird beim Herunterlaufen des Baumes im ersten Schritt der jeweilige Abstand des Punktes p zum Knoten direkt berechnet und in einer “priority queue“ gespeichert. Bei der späteren Überprüfung, ob ein nächster Nachbar nicht doch im anderen Nachfolgeknoten liegen kann, werden die Knoten mit dem geringsten Abstand zuerst überprüft.

3.7.2 Multiple randomized kd-Bäume

Multiple randomized kd-Bäume werden von Silpa-Anan and Hartley in [SAH08] vorgestellt. Im Gegensatz zum Original-kd-Baum-Algorithmus, der die Daten in zwei Hälften teilt, anhand der Dimension mit der größten Varianz, werden “randomized“ kd-Bäume erstellt in dem die splitte Dimension zufällig gewählt wird aus einer Menge von x Dimensionen welche die größte Varianz aufweisen. Für jedes Datenset werden mehrere Bäume erzeugt. Während der Suche wird die “priority queue“ über alle “randomized“ kd-Bäume geführt.

M_1 und M_2 für die gilt das alle Elemente aus M_1 kleiner sind als der Median und alle Elementen aus M_2 größer sind.

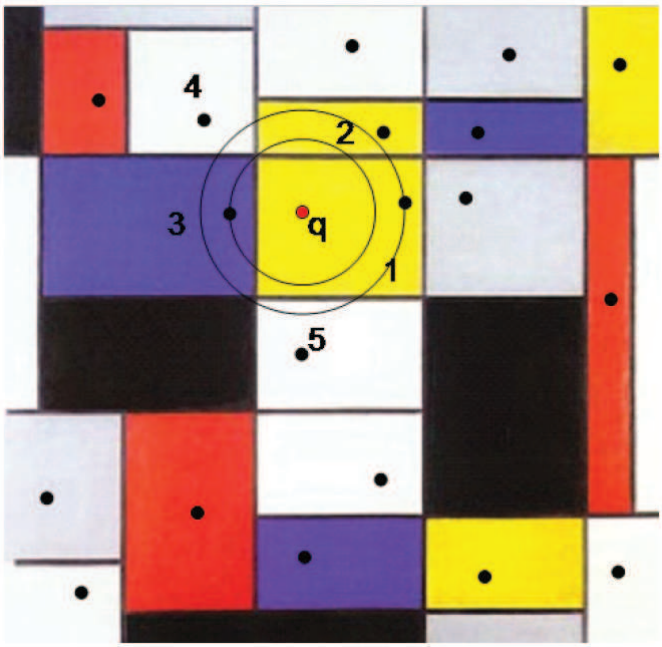


Abbildung 3.14: Die Abbildung zeigt eine “priority search“ in einem kd-Baum. Der Suchpunkt p wird durch den roten Punkt repräsentiert, dessen nächster Nachbar in Zelle 3 liegt. Im ersten Schritt wird der Baum herunter gelaufen und es wird die Zelle gefunden, in der sich der erste nächste Nachbar Kanditat befindet (Zelle 1). Alle Zellen, die sich im Radius r der Entfernung von p zum aktuell nächsten Nachbarn befinden, werden überprüft, das sind hier die Zellen 2 bis 5. Der Radius r wird angepasst wenn sich ein besserer Kandidat findet. Gibt es keine Zelle mehr im Radius r, ist die Suche beendet. Quelle: [SAH08].

3.7.3 Hierarchical k-means-Bäume

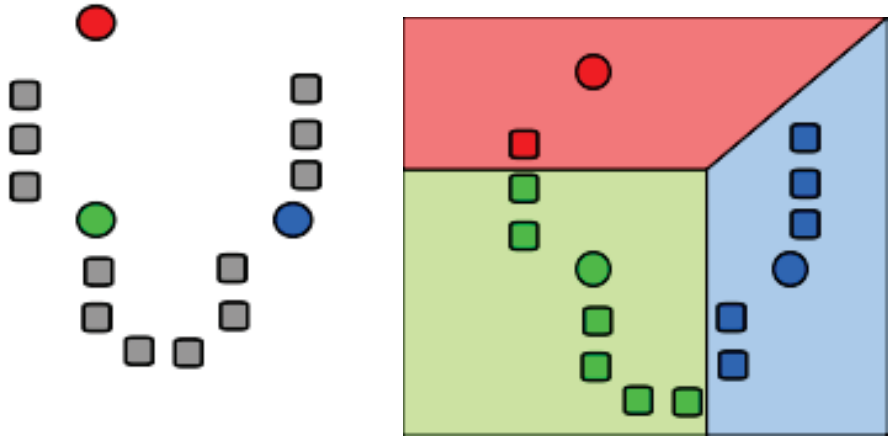
Ein hierarchical k-means-Baum wird erstellt, indem die Menge aller Punkte auf der momentanen Ebene in k Mengen mithilfe des k-Means-Algorithmus unterteilt wird. Diese wird auf jeder Ebene rekursiv solange wiederholt bis die Anzahl der Punkte kleiner als k ist.

K-Means-Algorithmus

Der k-Means-Algorithmus bildet aus einer Menge von ähnlichen Objekten eine vorher bekannte Anzahl von k Gruppen. Der Algorithmus geht dabei wie folgt vor:

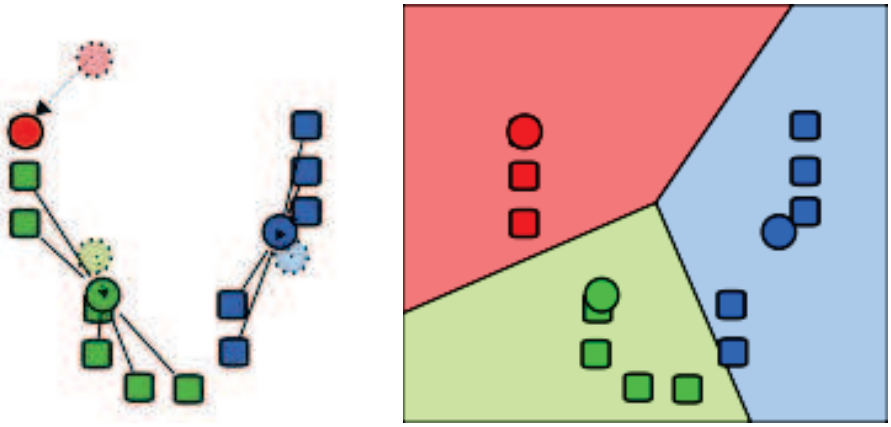
1. Die k Cluster-Schwerpunkte werden zufällig verteilt.
2. Jedes Objekt wird demjenigen Cluster zugeordnet, dessen Schwerpunkt ihm am nächsten liegt. Dazu muss eine Distanzfunktion, zum Beispiel die Euklidische Norm, verwendet werden.
3. Für jeden Cluster wird der Schwerpunkt neu berechnet, sodass er in der Mitte des Clusters liegt.

4. Basierend auf den neu berechneten Zentren werden die Objekte wieder wie in Schritt 2 auf die Cluster verteilt, bis sich die Schwerpunkte nicht mehr bewegen oder eine maximale Iterationstiefe erreicht wurde.



(a) Drei Clusterzentren wurden zufällig gewählt.

(b) Die durch Rechtecke repräsentierten Objekte (Datenpunkte) werden jeweils dem Cluster mit dem nächsten Clusterzentrum zugeordnet.



(c) Die Zentren (jeweilige Schwerpunkte) der Cluster werden neu berechnet.

(d) Die Objekte werden neu verteilt und erneut dem Cluster zugewiesen, dessen Zentrum am nächsten ist.

Abbildung 3.14: K-Means-Algorithmus. Quelle: http://de.wikipedia.org/wiki/K-Means-Algorithmus.

4 Positionsbestimmung

In diesem Abschnitt werden die Grundlagen der Projektive Geometrie und Epipolargeometrie erläutert. Es wird erläutert, wie mithilfe bekannter Punkte die Position bestimmt werden kann und wie die Position von Punkten im Raum bestimmt werden kann. Außerdem werden Verfahren wie der RANSAC-Algorithmus vorgestellt, der notwendig ist, um Ausreißer auszusortieren und so eine korrekte Position zu bestimmen.

4.1 Homogene Koordinaten

In der Geometrie erlauben homogene Koordinaten die Darstellung von Transformationen durch eine Multiplikation der Koordinaten mit Matrizen in einfacher Weise. Im Gegensatz zu den üblichen kartesischen Koordinaten, die im $\mathbb{R}^3$ aus x-, y- und z-Komponente bestehen haben die homogene Koordinaten eine zusätzliche Komponente w. Die Transformation von kartesischen Koordinaten zu homogenen Koordinaten erfolgt durch:

$$(x, y, z)^T \rightarrow (x, y, z, 1)^T \tag{4.1}$$

Zurück transformiert werden kann durch:

$$(x, y, z, w)^T \rightarrow (x/w, y/w, z/w)^T \tag{4.2}$$

Nähere Informationen zu homogenen Koordinaten sind zu finden unter [HZ04] S.3ff.

4.2 Projektive Geometrie

Die projektive Geometrie hat nach [Sch05] ihren Ursprung im frühen Mittelalter. Man beschäftigte sich zu dieser Zeit damit, wie man räumliche Szenen auf einer zweidimensionalen Bildebene abbildet und mathematisch korrekt beschreiben kann. Die klassische Geometrie der Euklidischen Räume $\mathbb{R}^n$ stieß dabei an ihre Grenzen. Insbesondere projektive Effekte im Unendlichen wie beispielsweise der Schnitt zweier parallel verlaufender Geraden im imaginären Fluchtpunkt erforderten neue Formalismen, was zur Einführung von projektiven Räumen $\mathbb{P}^n$ führte.

Diese stellen im Wesentlichen eine Erweiterung der Euklidischen Räume $\mathbb{R}^n$ um ideale Punkte, Geraden und Ebenen dar, die im Unendlichen liegen und damit den Ausgangspunkt zur mathematischen Beschreibung entsprechender Effekte bilden.

Die Grundlagen der projektiven Geometrie bildet das Lochkameramodell. Das Lochkameramodell geht davon aus, dass Lichtstrahlen durch ein unendlich kleines Loch, den Brennpunkt, auf die Rückwand der Lochkamera abgebildet werden. Durch die Punktspiegelung erfährt die Abbildung eine Spiegelung am Brennpunkt. Im folgenden Abschnitt werden nun die mathematischen Grundlagen dieses Abbildungsprozesses erläutert.

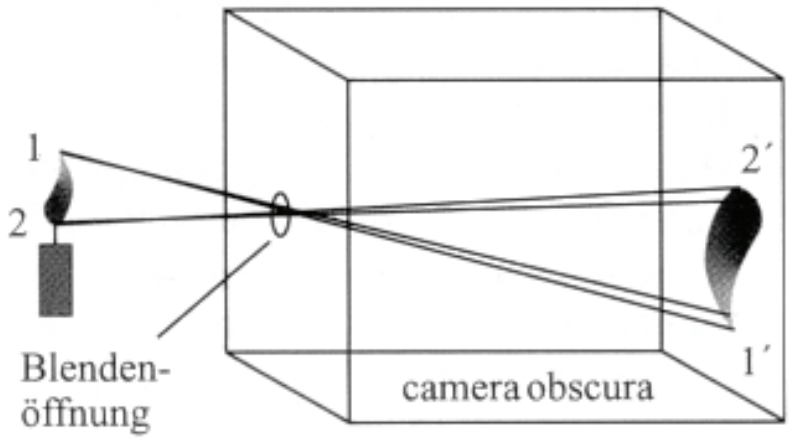


Abbildung 4.1: Modell der Lochkamera.

In Abbildung 4.2 ist der geometrische Zusammenhang zwischen einem 3D-Punkt $X = (x, y, z)$ und seiner Abbildung $x = (u, v)$ dargestellt. Im Gegensatz zum Modell der Lochkamera bzw. einer realen Kamera liegt in der Abbildung 4.2 das Projektionszentrum hinter der Bildebene. Diese hat keine Konsequenzen für die mathematische Herleitung mit der Ausnahme der Spiegelung der x- und y-Achsen der Koordinatensysteme. Diese Darstellungsform ermöglichet es, Zusammenhänge einfacher zu erläutern und wird in allen weiteren Abbildungen so verwendet.

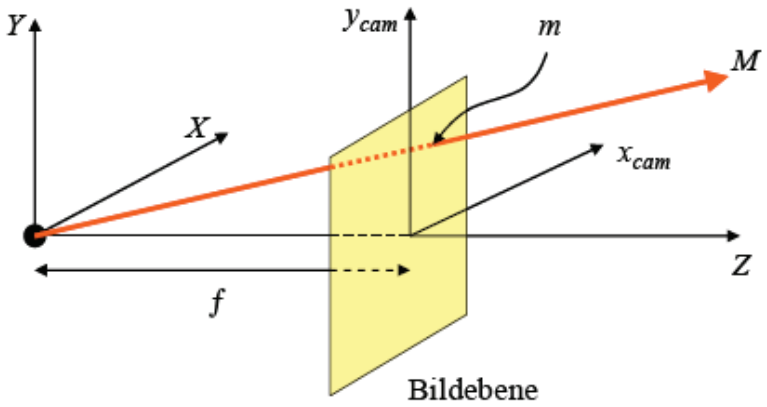


Abbildung 4.2: Projektionsschema beim Lochkameramodell. Quelle: [Mar04b].

4.2.1 Projektionsgesetze

Ein Punkt X im $\mathbb{R}^3$ mit den Koordinaten (X_w, Y_w, Z_w) hat einen korrespondierenden Punkt x auf der Bildebene $\mathbb{R}^2$ mit den Bildkoordinaten (u, v). Die Beziehung zwischen Weltpunkt und Bildpunkt ist wie folgt definiert:

$$\frac{u}{X_w} = \frac{v}{Y_w} = \frac{f}{Z_w} \tag{4.3}$$

Mithilfe der projektiven Geometrie unter Verwendung homogener Koordinaten kann die Gleichung in Matrixform geschrieben werden. Durch Umstellen der Gleichung 4.3 erhält man:

$$u = \frac{f}{Z_w} * X_w, v = \frac{f}{Z_w} * Y_w \tag{4.4}$$

Damit kann man dann (u, v) in homogene Koordinaten umformen:

$$\lambda * \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \text{ mit } \lambda = \frac{Z_w}{f} \tag{4.5}$$

$(X_w, Y_w, Z_w)^T$ kann durch Erweiterung um 1, als homogene Koordinaten dargestellt werden $(X_w, Y_w, Z_w, 1)^T$. Die homogen Koordinaten lassen sich jetzt in Matrixform aufschreiben und durch Multiplikation mit der so genannten Projektionsmatrix P können die Punkte aus dem Raum $\mathbb{R}^3$ in den $\mathbb{R}^2$ überführt werden.

$$\begin{bmatrix} u \\ v \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4.6}$$

Die Elemente der Projektionsmatrix hängen dabei von den Orientierungsparametern der Kamera ab. Diese sind im Einzelnen der innere Aufbau der Kamera (“innere Orientierung“) und die Lage der Kamera im Raum sowie die Blickrichtung der Kamera (“äußere Orientierung“).

Die innere Orientierung der Kamera setzt sich aus folgenden Elementen zusammen:

1. Bei k_x und k_y handelt es sich um Scale (Maßstabs) Faktoren.
2. Der Position des Bildhauptpunktes $h_0 = (u_0, v_0)$ als der Durchstoßpunkt der optischen Achse durch die Bildebene.
3. Dem Scherungswinkel θ zwischen den Bildachsen.

Zusammengefasst wird das in der Kalibrierungsmatrix K:

$$K = \begin{bmatrix} k_x & \theta & u_0 \\ 0 & k_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

Neben der Kalibrierungsmatrix enthält die Projektionsmatrix auch den Rotations-, und den Translationsanteil der Transformation zum Weltursprung. Im Weiteren wird der Transitionsvektor (X, Y, Z) bezüglich des Weltkoordinatensystems mit t, die Aufnahmerichtung mit R bezeichnet. Letzteres ist eine 3×3-Rotationsmatrix. Für P ergibt sich damit:

$$P = KR[I|-C] \tag{4.8}$$

$[I|-C]$ ist eine 3×4-Matrix zusammengesetzt aus der 3×3-Einheitsmatrix I und dem negierten Kameracenter C. Es ist oft praktisch anstelle der Transformation der Welt zum Kameracenter die Transformation der Welt zum Bildmittelpunkt zu nutzen $X_{cam} = RX + t$. In dem Fall ist die Projektionsmatrix:

$$P = K[R|t] \tag{4.9}$$

mit $t = -RC$. Mithilfe der Projektionsmatrix lässt sich nun folgender Zusammenhang zwischen Objekt- und Bildpunkt definieren:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ W_w \end{bmatrix} \tag{4.10}$$

4.2.2 Bestimmung der Kameraposition mithilfe der Projektionsmatrix

Die Projektionsmatrix lässt sich, wie im Abschnitt 4.2.1 gezeigt, direkt aus den Orientierungsparametern der Kamera berechnen. Da die Berechnung der Projektionsmatrix meist vor einer Bestimmung der Kameraparameter durchgeführt wird, tritt dieser Fall selten auf. Im Folgenden wird erläutert, wie P mithilfe von bekannten 3D-Punkten und deren Abbildungen berechnet werden kann. Danach wird erläutert, wie die Projektionsmatrix zerlegt wird und hieraus die Rotation und Translation zum Weltursprung bestimmt werden können.

Sind eine Menge Punktkorrespondenzen $X_i \leftrightarrow x_i$ gegeben, lässt sich P aus diesen Punktepaaren berechnen. Ziel ist es, eine Matrix P zu bestimmen, sodass $x_i = PX_i$. Dazu wird die Formel mithilfe des Kreuzproduktes nach $x_i \times PX_i = 0$ umgestellt. Wenn $x_i = [x_i \quad y_i \quad w_i]$, ergibt sich nach Umstellung der Gleichung folgender Zusammenhang:

$$\begin{bmatrix} 0^T & -w_iX_i^T & v_iX_i^T \\ w_iX_i^T & 0^T & -u_iX_i^T \\ -v_iX_i^T & u_iX_i^T & 0^T \end{bmatrix} \begin{pmatrix} p^1 \\ p^2 \\ p^3 \end{pmatrix} = 0 \tag{4.11}$$

mit p^i der i-ten Zeile von P. Alternativ reicht es auch, die ersten beiden Gleichungen zu nutzen, da nur 2 von 3 Gleichungen linearunabhängig sind.

$$\begin{bmatrix} 0^T & -w_i X_i^T & v_i X_i^T \\ w_i X_i^T & 0^T & -u_i X_i^T \end{bmatrix} \begin{pmatrix} p^1 \\ p^2 \\ p^3 \end{pmatrix} = 0 \tag{4.12}$$

Eine Punktkorrespondenz liefert somit 2 Gleichungen. Da die Matrix P 12 Einträge hat und 11 Freiheitsgrade (der Skalierungsfaktor kann ignoriert werden), sind 11 Gleichungen notwendig, um P zu lösen. Da jede Punktkorrespondenz zwei Gleichungen liefert sind mindestens 5 1/2 Korrespondenzen notwendig um P zu lösen, wobei sich die Punkte nicht auf einer Ebene befinden dürfen.

Zerlegung der Projektionsmatrix

Es ist möglich, aus P wiederum die einzelnen Orientierungsparameter der Kamera zu berechnen. Für das Projektionszentrum t gilt $Pt = 0$. Diese Eigenschaft kann als lineares Gleichungssystem aufgefasst und mittels Singulärwertzerlegung (siehe [HZ04] S.585) gelöst werden. Dabei ist zu beachten, dass die Rechteckmatrix P um eine Zeile mit Nullen ergänzt werden muss.

Die Rotationsmatrix R und die Kalibrierungsmatrix K extrahiert eine RQ-Zerlegung (siehe [HZ04] S.579) aus der ersten Matrix 3×3 Teilmatrix M von P:

$$M = KR = \begin{bmatrix} k_{11} & k_{21} & k_{31} \\ 0 & k_{22} & k_{32} \\ 0 & 0 & k_{33} \end{bmatrix} \begin{bmatrix} r_{11} & r_{21} & r_{31} \\ r_{21} & r_{22} & r_{32} \\ r_{31} & r_{23} & r_{33} \end{bmatrix} \tag{4.13}$$

4.3 Pose Estimation from a 3D Plane

Wie in Abschnitt 4.2.2 beschrieben, muss zur Bestimmung der Projektionsmatrix die Relation zwischen 3D-↔ 2D-Punkten bekannt sein. Die Bestimmung dieser 3D-Positionen ist allerdings recht aufwändig (siehe 4.5). Um diesen Aufwand zu vermeiden, werden oft Flächen (z.B. Bilder) genutzt, da deren 3D-Position ihrer 2D-Position entspricht (aufgrund der nicht vorhanden Tiefe). Allerdings ist es nicht möglich, das in 4.2.2 vorgestellte Verfahren zu nutzen, um die Projektionsmatrix zu bestimmen, da die fehlende z-Komponente zu einer singulären Lösungsmatrix führt. Um diese zu umgehen, kann, falls die Kalibrierungsmatrix K vorhanden ist, die Projektionsmatrix bzw. die Rotation und Translation mithilfe der Homographie bestimmt werden.

Die Homograhie H beschreibt die Abbildung zwischen zwei Flächen:

$$\begin{bmatrix} x\prime_1 \\ x\prime_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \tag{4.14}$$

Ein Beispiel für die Gleichung 4.16 ist in Abbildung 4.3 zu sehen. Ein Punkt wird mithilfe eines Strahls, der durch das Projektionszentrum O geht, von Fläche π auf Fläche π' abgebildet.

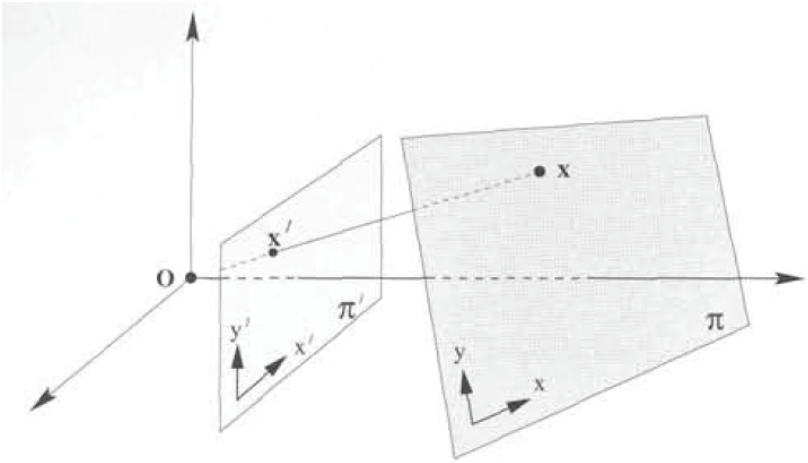


Abbildung 4.3: Projektion eines Punktes zwischen zwei Flächen. Quelle: [HZ04].

Die Beziehung zwischen einer 3D-Fläche und ihrer Abbildung lassen sich jetzt mithilfe einer Homograhie beschreiben. Es wird angenommen, dass für die Fläche gilt $Z = 0$. Daraus folgt, dass jeder Punkt auf der Fläche in homogener Schreibweise, wie folgt dargestellt werden kann $X = (X, Y, 0, 1)^T$. Wie in Abschnitt 4.2.1 beschrieben, ist P wie folgt definiert $P = K[R|t]$ und die Abbildung zwischen 3D und 2D als $x = PX$ definiert. Dies ist das Gleich wie

$$x = PX = K[r_1 r_2 r_3 t] \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \tag{4.15}$$

mit r_1,r_2 und r_3 als erste, zweite und dritte Spalte von R. Da $Z = 0$ ist, gilt $r_3 * Z = r_3 * 0 = (0, 0, 0)^T$ und somit kann r_3 weggelassen werden:

$$= K[r_1 r_2 t] \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = H \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{4.16}$$

Wobei gilt $K[r1r2t] \sim H$.

Bestimmung von H

Sobald H und K bekannt sind, kann die Kameraposition zurückgewonnen werden. Wenn die $X \leftrightarrow x$ Zuordnungen bekannt sind, kann H mithilfe von DLT ähnlich wie in 4.2.2 berechnet werden (für genauere Erläuterungen siehe [HZ04] S. 88ff.) :

$$\begin{bmatrix} 0^T & -w_i x_i^T & y_i x_i^T \\ w_i x_i^T & 0^T & -x_i x_i^T \end{bmatrix} \begin{pmatrix} p^1 \\ p^2 \\ p^3 \end{pmatrix} = 0 \tag{4.17}$$

Bestimmung von R und t aus H

Zur Bestimmung von R und t muss H zerlegt werden. Bis auf ein λ ist $H = K[r_1 r_2 t]$. Deshalb gilt $H = \lambda\, K[r_1 r_2 t]$ und $K^{-1}H = \lambda\, [r_1 r_2 t]$.

Eine Eigenschaft der Rotationsmatrix besteht darin, dass alle Zeilen und Spalten Einheitsvektoren sind, also die Länge eins haben. Daraus kann man folgern, dass die Abweichung von r_1 und r_2 zum Einheitsvektor das gesuchte λ ist. Somit lässt sich λ bestimmen, indem die Norm[1] für r_1 oder r_2 bestimmt wird. Da R orthonormal ist gilt $r_3 = r_1 \times r_2$ und somit gilt:

$$P = K[r_1 r_2 (r_1 \times r_2) t] \tag{4.18}$$

4.4 Epipolargeometrie

Die Epipolargeometrie ist ein mathematisches Modell aus der Geometrie, das die geometrischen Beziehungen zwischen verschiedenen Kamerabildern des gleichen Objekts beschreibt. Mithilfe der Epipolargeometrie kann eine einfache Beziehung zwischen korrespondierenden Punkten ohne Kenntnis der Kamerapositionen hergestellt werden.

Diese Beziehung kann mit einer 3×3-Matrix beschrieben werden. Ist die innere Orientierung bekannt, können die Korrespondenzen in Sensorkoordinaten beschrieben werden und die Matrix wird als Essentialmatrix bezeichnet. Sind diese Parameter nicht bekannt, werden die Beziehungen in Bildkoordinaten angegeben und die Matrix wird als Fundamentalmatrix bezeichnet.

4.4.1 Aufbau der Epipolargeometrie

Bei der allgemeinen Epipolargeometrie wird davon ausgegangen, dass eine Szene aus zwei verschiedenen Perspektiven aufgenommen wird. Dabei werden die Kameras verschoben und zueinander verdreht. Abbildung 4.4 zeigt den geometrischen Aufbau der Epipolargeometrie und deren wichtigsten Parameter, die im Folgenden erläutert werden.

$Kamera1$ und $Kamera2$ bezeichnen die optischen Zentren der Kameras. Die Gerade, die sie verbindet, nennt man Basislinie B. Die Schnittpunkte dieser Linie mit den Bildebenen sind die so genannten Epipole. Diese werden mit e_1 und e_2 gekennzeichnet. Der Punkt M ist ein dreidimensionaler Punkt mit seinen Abbildungen auf der zweidimensionalen Ebene m_1 und m_2. Der

[1] Der mathematische Begriff der Norm ist die Verallgemeinerung des geometrischen Begriffs der Länge eines Vektors.

Punkt M und die beiden Punkte der optischen Zentren spannen eine Ebene auf, die Epipolarebene π genannt wird. Diese Ebene schneidet die beiden Bildebenen mit den Geraden l_1 und l_2. Diese Schnittgeraden werden Epipolarlinien genannt. Die Verschiebung und Drehung der zwei Kameras, wird mit einer Rotationsmatrix R und einem Verschiebungsvektor t beschrieben.

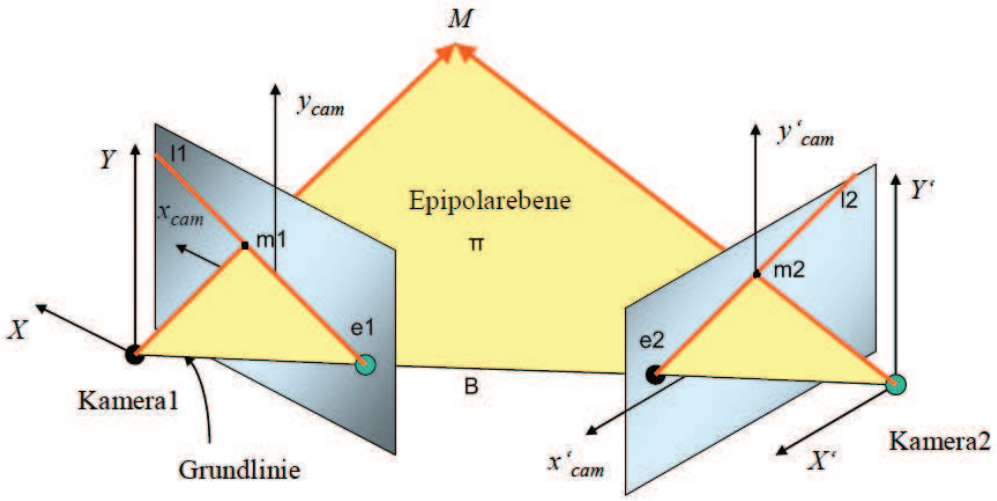


Abbildung 4.4: Epipolare Geometrie bei Stereobildern. Quelle: [Mar04b].

Die Fundamentalmatrix F enthält die gesamte Information über die Epipolargeometrie. Sie kann auch ohne Kenntnis der Orientierung der Kameras (das heißt ohne Kenntnis der Projektionsmatrizen P_1 und P_2 sowie des Projektionszentrums O_1) aus den Bildkoordinaten korrespondierender Punkte bestimmt werden. Sie beschreibt sowohl die Beziehung zwischen den Bildpunkt und Epipolarlinie als auch die Beziehung zwischen den Bildpunkten.

$$l_1 = F^T m_1 \tag{4.19}$$

$$l_2 = F m_1 \tag{4.20}$$

$$m_2^T F m_1 = 0 \tag{4.21}$$

Aus der Fundamentalmatrix lässt sich nun die Essentialmatrix E bestimmen, welche die Beziehung von $Kamera2$ zu $Kamera1$ ausschließlich über die äußeren Parameter, also Rotation und Translation, beschreibt.

$$E = [t]_x R \tag{4.22}$$

Um das Kreuzprodukt zweier dreidimensionaler Vektoren durch eine Matrixmultiplikation ausdrücken zu können, wird der 3D-Translationsvektor t wie folgt als schiefsymmetrische 3×3-Matrix definiert.

$$[t]_x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \equiv \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{4.23}$$

Der Zusammenhang zwischen Essential- und Fundamentalmatrix ist über die inneren Parameter der beiden Kameras K_1 und K_2 gegeben.

$$E = K_2^T F K_1 \tag{4.24}$$

4.4.2 Berechnung der Fundamentalmatrix

Um aus einer Menge korrespondierender Bildpunkte die Fundamentalmatrix zu bestimmen, wird die Epipolargleichung 4.21 ausmultipliziert:

$$x_2x_1f_{11} + x_2y_1f_{12} + x_2f_{13} + y_2x_1f_{21} + y_2y_1f_{22} + y_2f_{23} + x_1f_{31} + y_1f_{32} + f_{33} = 0 \tag{4.25}$$

oder in vektorieller Schreibweise:

$$\begin{pmatrix} x_2x_1 & x_2y_1 & x_2 & y_2x_1 & y_2y_1 & y_2 & x_1 & y_1 & 1 \end{pmatrix} \cdot f = 0 \tag{4.26}$$

mit

$$f = \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{21} & f_{22} & f_{23} & f_{31} & f_{32} & f_{33} \end{pmatrix}^T. \tag{4.27}$$

Aus n Punktkorrespondenzen kann das folgende homogene lineare Gleichungssystem aufgestellt werden (der obere Index gibt die Punktindex an):

$$\begin{pmatrix} x_2^1x_1^1 & x_2^1y_1^1 & x_2^1 & y_2^1x_1^1 & y_2^1y_1^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_2^nx_1^n & x_2^ny_1^n & x_2^n & y_2^nx_1^n & y_2^ny_1^n & y_2^n & x_1^n & y_1^n & 1 \end{pmatrix} \cdot f = A \cdot f = 0 \tag{4.28}$$

4.4.3 Bestimmung der Projektionsmatrizen aus der Essentialmatrix

Die Projektionsmatrix lässt sich aus der Essentialmatrix bestimmen, hierfür muss im ersten Schritt die Essentialmatrix durch Singulärwertzerlegung[2] zerlegt werden ($E = UDV^T$).

Nun kann mithilfe der Matrixen

$$Z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.29}$$

und

[2] Die Singulärwertzerlegung (Abk.: SVD für Singular Value Decomposition) einer Matrix bezeichnet deren Darstellung als Produkt dreier spezieller Matrizen. Daraus kann man die Singulärwerte der Matrix ablesen. Siehe auch [HZ04] S. 585.

$$W = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.30}$$

die Rotation und Translation bestimmt werden.

$$R = UWV^T \text{oder } UW^TV^T \tag{4.31}$$

und

$$t = U(001)^T = u_3 \tag{4.32}$$

mit $U = [u_1 u_2 u_3]$.

Für die Projektionsmatrix von Bild 1 wird die Projektionsmatrix als $P = [I|0]$ festgelegt. Damit gibt es für die Projektionsmatrix des zweiten Bildes vier Möglichkeiten:

$$P' = [UWV^T| + u_3] \text{ oder } [UWV^T| - u_3] \text{ oder } [UW^TV^T| + u_3] \text{ oder } [UW^TV^T| - u_3] \tag{4.33}$$

Um festzustellen, welche der Matrizen die korrekte ist, muss die 3D-Position eines beliebigen Punktes bestimmt werden (wie die Position bestimmt wird, wird im nächsten Abschnitt erläutert). Bei der richtigen Projektionsmatrix befindet sich der bestimmte Punkt bei beiden Kameras vor der Bildfläche (siehe Abbildung 4.5).

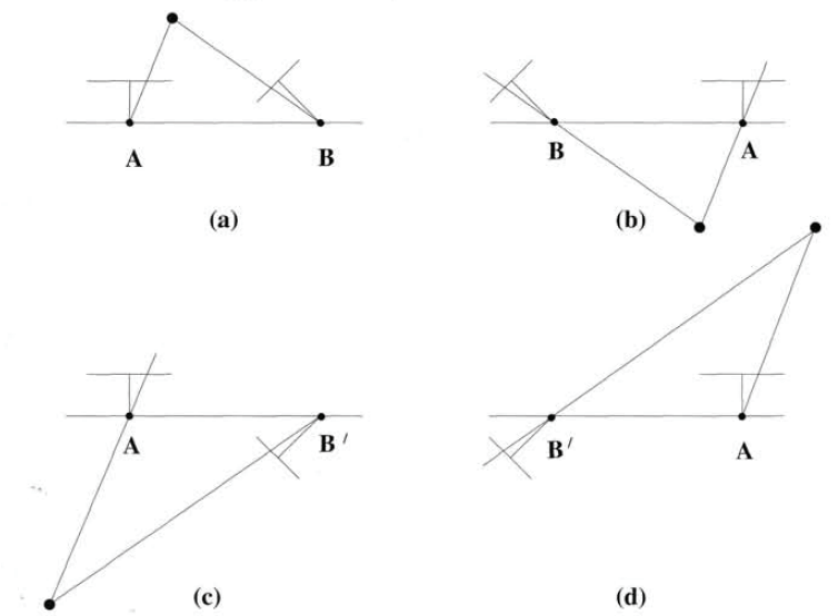


Abbildung 4.5: Die vier möglichen Lösungen für eine Rekonstruktion von E. Quelle: [HZ04].

4.5 Triangulation

Im Bereich Computer Vision beschreibt Triangulation einen Prozess zur Bestimmung eines 3D-Punktes aus seiner Abbildung auf zwei oder mehr Bildern. Um dieses Problem zu lösen, ist es notwendig, dass man die Parameter der Kamera kennt, die zur Projektion von 3D zu 2D verwendet wurde. Das Triangulationsproblem ist theoretisch trivial, jeder Punkt in einem Bild korrespondiert mit einer Linie im 3D Raum, sodass alle Punkte auf dieser Linie auf dem gleichen Punkt abgebildet werden. Wenn ein Punkt in zwei oder mehr Bildern gefunden wird, muss es sich um eine Abbildung eines 3D-Punktes X handeln. Die Menge aller Linien, die durch diesen Punkt erzeugt werden, müssen sich alle am Punkt X treffen.

Das Problem in der Praxis ist, dass es immer Abweisungen durch z.B. Verformungen der Linsen der Kameras gibt. Als Konsequenz daraus treffen sich die Linien nicht immer genau in einem Punkt. Das Problem ist dann, einen 3D-Punkt zu finden, der zu den gemessen Bildpunkten passt. Eine gute Approximation für die Lage des tatsächlichen 3D-Punktes ist der Mittelpunkt der kürzesten Strecke zwischen den beiden optischen Strahlen.

Eine Methode zur Lösung der Triangulation ist z.B. das homogene Verfahren (siehe auch [HZ04] S. 312). Der gleiche Punkt X wird in zwei Bildern auf x und x' abgebildet. Hierfür gilt $x = PX$ und $x' = P'X$. In den folgenden Gleichungen bezeichnet p_{jk} das (j,k)-Element der Projektionsmatrix P. Die ersten drei Elemente jeder Zeile von P werden zu einem Vektor $p_j = (p_{j1}p_{j2}p_{j3})$ zusammengefasst. Nun kann durch Anwendung des Vektorproduktes der homogene Skalierungsfaktor eliminiert werden. Man erhält z.B. für das erste Bild $x \times PX = 0$ oder:

$$\begin{array}{rcl} v(p_3^T X) - (p_2^T X) & = & 0 \\ u(p_3^T X) - (p_1^T X) & = & 0 \\ u(p_2^T X) - v(p_1^T X) & = & 0 \end{array} \tag{4.34}$$

Aus den ersten beiden linear unabhängigen Gleichungen für die beiden Komponenten u und v kann dann entsprechend für zwei Punktkorrespondenzen eine Gleichung folgender Form $AX = 0$ aufgestellt werden. Die Matrix A lautet:

$$A = \begin{bmatrix} up_3^T - p_1^T \\ vp_3^T - p_2^T \\ u'p_3'^T - p_1'^T \\ v'p_3'^T - p_2'^T \end{bmatrix} \tag{4.35}$$

Diese Gleichung kann nun mithilfe der direkten linearen Transformation (DLT) gelöst werden (siehe hierzu [HZ04] S. 592).

4.6 RANSAC-Algorithmus

Der Random Sample Consensus, deutsch etwa “Übereinstimmung mit einer zufälligen Stichprobe“), kurz RANSAC Algorithmus, ist speziell für die automatisierte Bildanalyse konzipiert und

wurde erstmals 1981 von Fischler und Bolles vorgestellt [FB81]. Er zeichnet sich dadurch aus, dass er eine große Datenmenge mit einer verhältnismäßig hohen Anzahl an Ausreißern an ein vorgegebenes Modell anpassen kann.

Problemstellung

In der Bildverarbeitung soll oft aus einer Menge von Datenpunkten ein Modell bestimmt werden. Dabei liegen mehr Datenpunkte vor, als zur Ermittlung des Modells notwendig sind, das Modell ist also überbestimmt. Die Datenpunkte können Ausreißer enthalten, also Werte, die nicht in die erwartete Messreihe passen. Traditionelle Verfahren wie die Ausgleichungsrechnung gehen von der Normalverteilung der Datenpunkte aus. Dies ist in Abbildung 4.6 dargestellt. Es soll eine Gerade (das Modell) an die Punkte angepasst werden. Der einzelne Ausreißer unter den 20 Datenpunkten kann einerseits durch traditionelle Verfahren vor Bestimmung der Geraden nicht ausgeschlossen werden. Andererseits beeinflusst er aufgrund seiner Lage die Ausgleichsgerade unverhältnismäßig stark (so genannter Hebelpunkt).

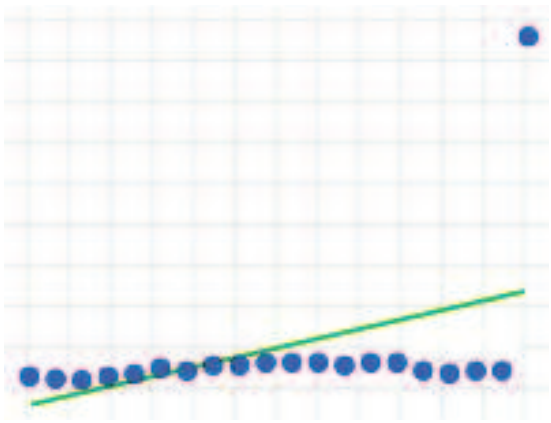


Abbildung 4.6: Der einzelne Ausreißer zieht die Ausgleichsgerade nach oben.

Der RANSAC-Algorithmus verfolgt einen iterativen Ansatz. Statt alle Datenpunkte gemeinsam auszugleichen, werden lediglich so viele zufällig ausgewählte Datenpunkte benutzt wie nötig sind, um die Modellparameter zu berechnen (im Fall einer Geraden wären das zwei Punkte). Dabei wird zunächst angenommen, dass die ausgewählten Datenpunkte frei von Ausreißern sind. Diese Annahme wird überprüft, indem zuerst das Modell aus den zufällig gewählten Werten berechnet und danach die Distanz aller Messwerte (also nicht nur der ursprünglich ausgewählten) zu diesem Modell bestimmt wird. Ist die Distanz eines Datenpunktes zum Modell kleiner als ein vorher festgelegter Schwellwert, dann ist dieser Datenpunkt in Bezug auf das berechnete Modell kein grober Fehler. Er unterstützt es somit. Je mehr Datenpunkte das Modell unterstützen, desto wahrscheinlicher ist es, dass die zufällig zur Modellberechnung ausgewählten Datenpunkte keine Ausreißer enthielten. Der Algorithmus geht dabei wie folgt vor:

for $i = 1$ to N **do**
  Zufällige Auswahl von Messwerten
  Berechnung der Modellparameter
  Bestimmung der Unterstützung

end for

In jeder Iteration wird gespeichert, welche Messwerte das jeweilige Modell unterstützen. Diese Menge wird „Consensus set“ genannt. Aus dem größten Consensus Set, der im Idealfall keine Ausreißer mehr enthält, wird abschließend mit einem der traditionellen Ausgleichsverfahren die Lösung ermittelt.

Ein Nachteil des Random Sample Consensus-Algorithmus ist, dass dieser drei unspezifizierte Parameter besitzt:

1. Fehlertoleranz: Welche Distanz zum Modell gilt noch als korrekt?
2. Laufzeit: Wie viele Teilmengen müssen überprüft werden, um eine gewünschte Stabilität zu erreichen?
3. Grenzwert: Wie hoch muss die Anzahl der Punkte im „Consensus Set“ sein.

Als Lösung für die Wahl eines Parameters in Hinblick auf die Fehlertoleranz, schlagen Fischler und Bolles eine experimentelle Bestimmung vor. Das bedeutet, dass für unterschiedliche Experimente auch unterschiedliche Parameter resultieren können.

4.7 Kamerakalibrierung

Unter einer vollständigen Kamerakalibrierung versteht man die Bestimmung der Abbildungseigenschaften einer Kamera (innere Orientierung) sowie der Orientierung des Weltkoordinatensystems relativ zum Kamerastandpunkt (äußere Orientierung).

Das Lochkameramodell für die Abbildung einer dreidimensionalen Szene auf eine zweidimensionale Bildebene enthält 11 unbekannte Parameter in der Projektionsmatrix P. Die extrinsische Transformation hat drei Freiheitsgrade für die Translation und drei für die Rotation. Die intrinsische Transformation hat zusätzlich zwei Freiheitsgrade für die horizontale und vertikale Skalierung, zwei Freiheitsgrade für die Verschiebung des Bildkoordinatensystems und einen für die Scherung zwischen den beiden Bildachsen. Die Ermittlung dieser Unbekannten nennt sich Kalibrierung. Die Kamerakalibrierung im Kontext dreidimensionaler maschineller Bildverarbeitung ist also die Bestimmung der intrinsischen und/oder extrinsischen Kameraparameter. Die Kamerakalibrierung wird in dieser Arbeit nur zur Bestimmung der intrinsischen Kameraparameter genutzt.

Einige Ansätze zur Ermittlung der Parameter gehen von einer bekannten dreidimensionalen Struktur der Szene aus, wie etwa von einen planaren Kalibrationsaufbau. Andere Autoren versuchen, durch einschränkendes Wissen über die Kamera - wie etwa konstante intrinsische Parameter oder zeitlich variierende und unbekannte intrinsische Parameter - eine Kalibrierung durchzuführen.

In dieser Arbeit wird zur Kalibrierung die Kamerakalibrierungsfunktion von OpenCV genutzt, die im Wesentlichen auf der Kamerakalibrierung von Zhang [Zha99] basiert. Der Ansatz von Zhang hat als Grundlage die gleich Idee wie das schon in Abschnitt 4.3 vorgestellt Verfahren zur Berechnung der Rotation und der Translation aus zwei Bildern mittels Homographie.

Bei diesem Verfahren wird ein planares Schachbrettmuster (siehe Abbildung 4.7) als Kalibrierungsobjekt genutzt. Die Eckpunkte der einzelnen Schachfelder dienen als Kalibrierungsmarken. Zur Bestimmung der intrinsischen Parameter sind mindestens zwei Bilder des Musters mit unterschiedlichen Orientierungen nötig, wobei diese Orientierungen nicht bekannt sein müssen. Um ein möglichst genaues Ergebnis zu erhalten, werden allerdings in der Praxis ca. 20 Bilder mit jeweils unterschiedlichen Orientierungen genutzt.

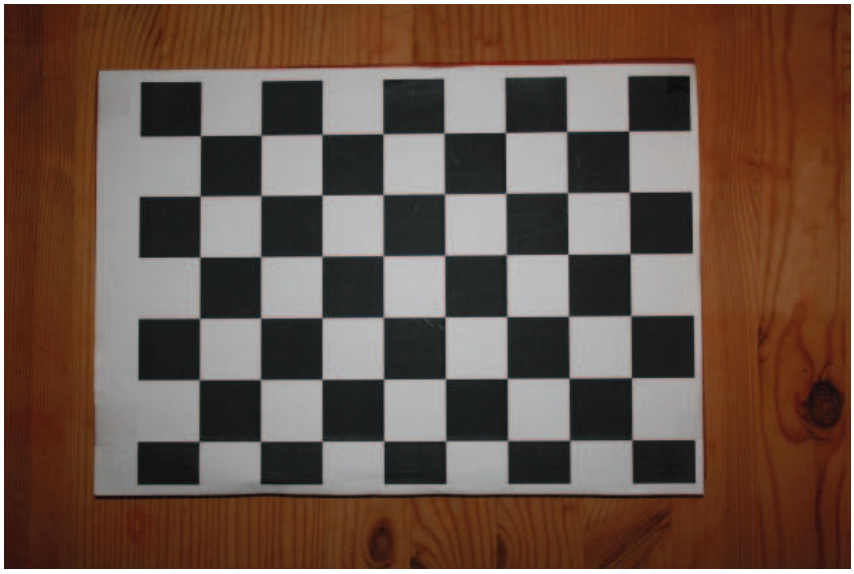


Abbildung 4.7: Schachbrettmuster das zur Kamerakalibrierung genutzt wurde.

Zhang berechnet im ersten Schritt die Homographiematrix was z.B. mit dem in Abschnitt 4.3 vorgestellten Verfahren möglich ist. Daraus lassen sich nun die Kameraparameter bestimmen. Die genaue mathematisch Vorgehensweise ist ausführlich in [Zha99] beschrieben.

4.8 Berechnung einer 3D-Karte aus 2 oder mehr Bildern

An dieser Stelle soll erläutert werden, wie mit den Verfahren die in diesem Kapitel vorgestellt wurden, eine 3D-Rekonstruktion einer Szene möglich ist. Mit 3D-Rekonstruktion wird an dieser Stelle nur die Rekonstruktion der Position von einer Menge von Punkten gemeint, nicht die Rekonstruktion von Oberflächen oder Formen.

Um die 3D-Position einzelner Punkte aus zwei und mehr Bildern zu Rekonstruktion kann wie folgt vorgegangen werden. Vor Beginn der Rekonstruktion muss eine Kamerakalibrierung stattfinden. Ist die Kalibrierungsmatrix bekannt, wird im nächsten Schritt die Fundamentalmatrix aus zwei Bildern bestimmt (siehe 4.4.2). Daraus lässt sich nun mit der bestimmten Kalibrierungsmatrix, die Essentialmatrix bestimmen (siehe 4.24). Aus der Essentialmatrix lassen sich nun die beiden Projektionsmatrizen bestimmen (siehe 4.4.3). Mithilfe der Projektionsmatrizen ist es nun wiederum möglich, durch Triangulation die 3D-Position aller Punkte zu bestimmen (siehe 4.5).

Sobald die 3D-Position einzelner Punkte bekannt ist, lässt sich die Projektionsmatrix aus jedem weiteren Bild direkt bestimmen (siehe 4.2.2). So ist es möglich, eine 3D-Rekonstruktion mit einer

beliebig großen Menge von Bildern, die einen beliebig großen Bereich abdecken, durchzuführen. Voraussetzung dafür ist nur, dass jedes neue Bild mit einem alten Bild überlappt und so von einer gewissen Anzahl an Punkten die 3D-Position bekannt ist, damit die Projektionsmatrix bestimmt werden kann.

5 Entwicklung eines Lösungsansatzes

Das Ziel dieser Masterarbeit ist die Entwicklung und anschließende Umsetzung eines Lösungsansatzes zur markerlosen Positionsbestimmung für Augmented Reality-Anwendungen. Diese soll in einem Framework zusammengefasst und durch zusätzliche Tools erweitert werden. In den vorangehenden Kapiteln wurden grundlegende mathematische Konzepte erläutert und verschiedene Techniken vorgestellt, die bei der Umsetzung hilfreich sein können. In diesem Abschnitt wird ein Lösungsansatz erläutert, der auf den in den vorangehenden Kapitelen vorgestellten Techniken basiert. Hierfür werden als Erstes die nötigen Anforderungen definiert und dann wird anhand dieser Anforderungen nach Lösungsmöglichkeiten gesucht.

5.1 Problemdefinition

Zur richtigen Platzierung der virtuellen Objekte in der Szene muss der Anwendung die Blickrichtung des Nutzers bekannt sein. Diese muss alleine aus den Bildinformationen der Kamera ermittelt werden. Es soll keine besondere Hardware genutzt werden, um Nutzer die Möglichkeit zu geben, die Anwendungen auf üblichen Computern zu nutzen. Zusätzliche Informationen wie die Beschaffenheit der Umgebung können nur genutzt werden, wenn diese vorher bekannt sind. Ein besonderes Anforderung soll es sein eine Genauigkeit zu erreichen, die eine Wirkung erzeugt so als ob sich die virtuellen Objekte tatsächlich im Bild befinden. Das ist eine Herausforderung, da selbst kleine Ungenauigkeiten dazu führen können, dass virtuelle Objekte in der Szene springen. Zusätzlich ist es wichtig, dass das Framework in Echtzeit arbeiten kann. Das bedeutet, dass die Performance gut genug sein soll, so das Augmented Reality Anwendungen die mit dem Framework entwickelt wurden, auf heute üblichen Computern mit 24 Bildern pro Sekunde laufen. Es soll sowohl auf künstliche Marker als auch zusätzlichen Sensoren außer der Kamera verzichtet werden.

Anforderungen

Konkret soll das Framework folgende Anforderungen erfüllen:

1. Echtzeitfähig: Eine Anwendung läuft in Echtzeit, wenn mindestens 24 FPS (Frames-per-second) erreicht werden und dem Nutzer die Verzögerung nicht störend auffällt, die zwischen dem Aufnahmezeitpunkt des Bildes und der Anzeige auf dem Display entsteht. Das wäre der Fall, wenn z.B. die Kamera nach links geschwenkt wird und dieser Schwenk erst stark verzögert auf dem Display zu sehen ist.

2. Robustheit: Das Framework kann als robust angesehen werden, wenn es möglich ist, die Position auch zu bestimmen, wenn die Verhältnisse nicht optimal sind. Nicht optimale Verhältnisse sind z.B. starke Beleuchtungsveränderungen, starke Bewegungen und Verdeckungen. Hierbei wird nicht gefordert, dass in Extremsituationen ein Einsatz möglich ist, vielmehr soll vermieden werden dass, sollten im normalen Betrieb Situationen auftreten, die nicht optimal sind, es sofort zu Störungen kommt.
3. Genauigkeit: Da es in der Praxis unmöglich sein wird, die Position ganz exakt zu bestimmen, wird hier genau mit: „Für den Nutzer nicht merklich zu sehen“ definiert. Das heißt, dass keine offensichtliches Wackeln oder Schwanken der virtuellen Objekte auftreten soll.
4. Entwicklerfreundlich: Es sollen Tools erstellt werden, die es dem Entwickler möglich machen, ohne großen Zeitaufwand Augmented Reality-Anwendungen zu entwickeln.

Diese Anforderungen können als Grundvoraussetzungen angesehen werden, die erfüllt werden müssen, damit das Framework in der Praxis nutzbar ist. Zusätzlich Anforderungen an bestimmte Teilbereiche des Frameworks werden in den entsprechenden Abschnitten definiert.

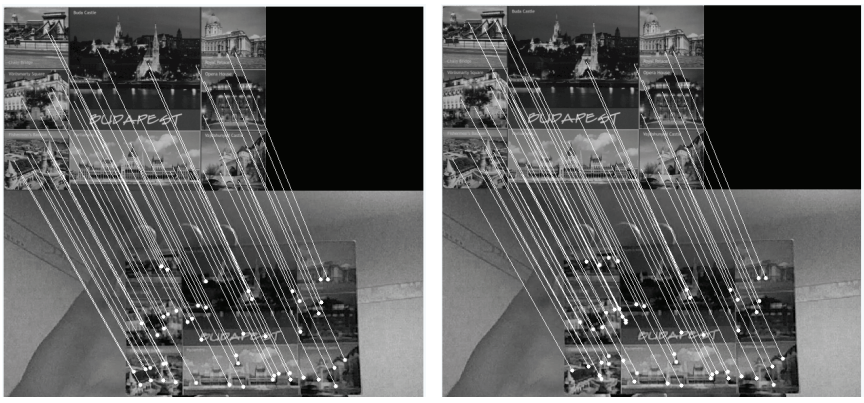
5.2 Tracking

Zum Tracking soll das in 3.6 vorgestellte Verfahren “Real-time 3D Object Pose Estimation and Tracking for Natural Landmark“’ genutzt werden, da es die Vorteile eines featuresbasierten Trackingverfahrens und die Geschwindigkeit eines Optischer Fluss Verfahren kombiniert. Zusätzlich bietet dieses Verfahren die Möglichkeit, Punkte auch weiter zu verfolgen, wenn eine sehr starke Rotation auftritt (siehe Abbildung 5.-1).

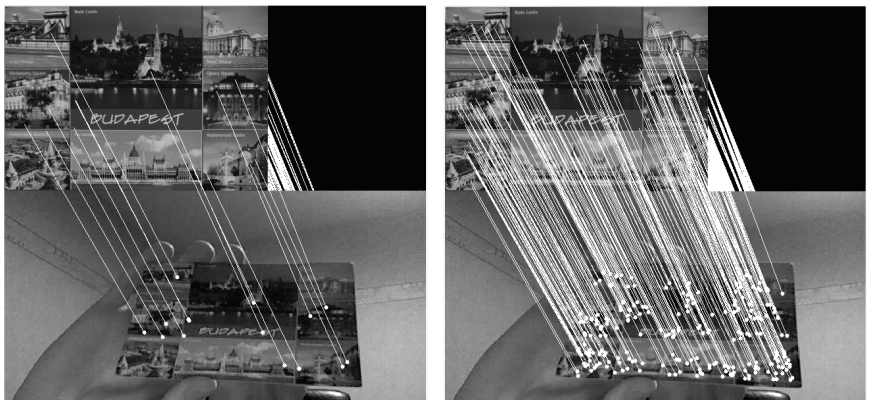
Bei starken Rotation wie in Abbildung 5.-1 hat sich gezeigt, dass Techniken wie SIFT oder SURF nur noch sehr wenige Punkte finden - trotz versprochener Rotationsinvarianz. Diese Probleme sind bei einem KLM-Tracker nicht vorhanden, da der Optische Fluss unabhängig von der Umgebung ist. Das ist darauf zurückzuführen, dass er nur den Fluss der Pixel verfolgt und nicht einzelne Objekte. Da Optischer Fluss Verfahren aber eine Initialisierungsphase voraussetzen und hierfür wieder Techniken wie SIFT oder SURF genutzt werden müssen, sind diese Vorteile nur dann vorhanden, wenn die Initialisierung stattfindet, bevor das Objekt gedreht wird. Das Verfahren kann allerdings nicht direkt übernommen werden, weil es die geforderten Ansprüche nicht erfüllt.

Problemdefinition

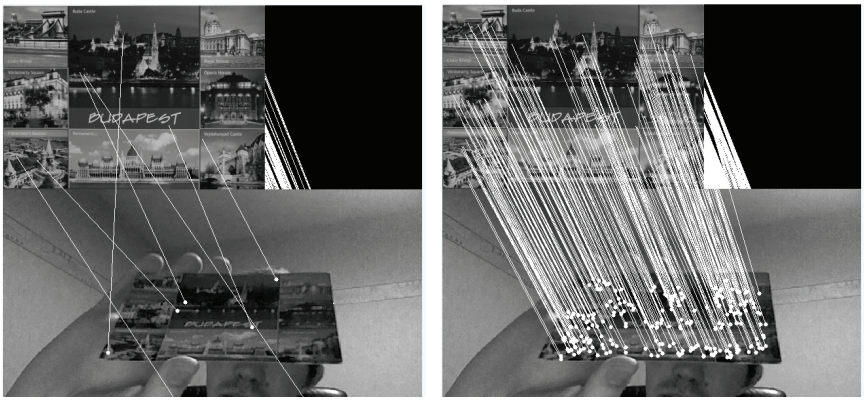
Das in 3.6 vorgestellte Verfahren muss bei jedem Verlust der zur trackenden Punkte neu initialisiert werden. Das führt dazu, dass es nur in einem sehr eingeschränkten Spezialfall genutzt werden kann und zwar dann, wenn aus einer Richtung auf ein zu trackendes Objekt geschaut wird. Ist das nicht der Fall, muss abhängig von der genauen Situation immer wieder neu initialisiert werden, was dazu führt, dass die in 5.1 definierte Echtzeitanforderungen nicht erfüllt werden können, da das Initialisieren nicht in Echtzeit möglich ist.



(a) SURF (b) SURF + KLT



(c) SURF (d) SURF + KLT



(e) SURF (f) SURF + KLT

Abbildung 5.-1: Vergleich: gefundene Punkte SURF (linke Seite) und SURF + KLT(rechte Seite) bei der Drehung eines Objektes. Beim ersten Bildpaar ist die Karte noch nicht gedreht, beim zweiten um ca. 25° und beim dritten um 50°. Deutlich ist zu erkennen, dass sich die Anzahl der Punkte beim reinen SURF stetig verringert, bei der Kombination SURF + KLT ist dies nicht der Fall.

Möglicher Lösungsansatz

Um trotz dieser Defizite das Verfahrens nutzten zu können, soll das Verfahren weiterentwickelt werden. Hierbei sollen im ersten Schritt die Vorteile von folgenden Maßnahmen evaluiert werden:

- **Nutzung eines anderen Verfahrens als SIFT zur Initialisierung:** Changhyun Choi, Seung-Min Baek und Sukhan Lee haben in ihrer Implementierung SIFT als Grundlage genutzt. Da SIFT eine deutlich schlechtere Performance hat als andere Verfahren wie z.B. SURF, soll das Verfahren ausgetauscht werden.
- **Automatisches Update alle x Frames:** Der Grundgedanke hierbei ist, dass der Initialisierungsprozess neben dem KLM-Tracker parallel abläuft. Der Initialisierungsprozess soll aber nicht für jeden Frame ausgeführt werden, sondern alle x Frames, z.B. alle 5 Frames. Die Berechnung der Featurepunkte soll also auch dementsprechend länger dauern. Das heißt, die Berechnung wird auf mehrere Frames aufgeteilt, sodass die zusätzliche Belastung so gering ist, dass es möglich ist, den Tracker in Echtzeit zu betreiben. Dabei soll der Bearbeitungszeitraum abhängig von den Ressourcen, die zu Verfügung stehen, festgelegt werden. Dadurch kann das System an die Hardware direkt angepasst werden, um auf unterschiedlichen Rechnern alle verfügbaren Ressourcen ausnutzen zu können und trotzdem immer Echtzeit zu garantieren. Dies könnte auch im laufenden Betrieb geschehen, indem die Zeit für eine Initialisierung bestimmt wird und durchschnittliche Rechenzeit für einen Frame. Abhängig von dem Ergebnis werden dann die Frames festgelegt, in denen eine neue Initialisierung stattfinden soll. Diese wird nochmal in Abbildung 3.10 verdeutlicht.

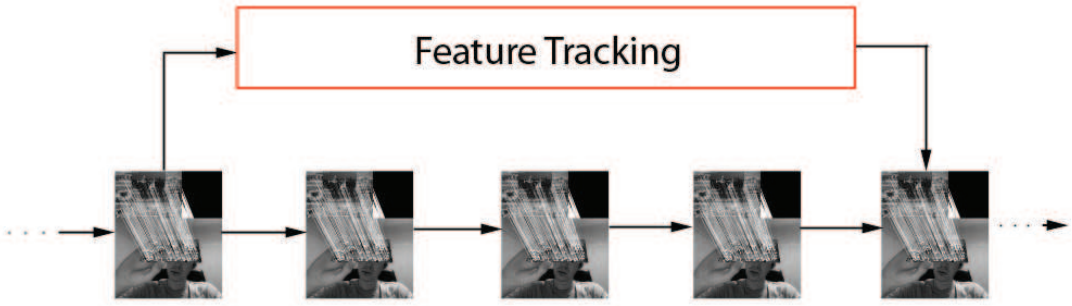


Abbildung 5.0: KLT mit parallelem Feature Tracking.

Dieser Ansatz geht von der Annahme aus, dass die Veränderung zwischen wenigen Frames sehr gering ist und nach Ablauf der Initialisierung immer noch ein Großteil der gleichen Punkte im Bild ist. Nachdem die neuen Featurepunkte berechnet wurden, muss deren Position im aktuellen Frame bestimmt werden. Diese würde im Idealfall funktionieren indem der KLM-Tracker die Veränderung der Position zwischen Frame 1 und Frame x berechnet. Sollte das nicht direkt möglich sein, müssen alle Zwischenschritte nochmal durchgeführt werden, indem der KLM-Tracker auf Frame 1 bis x nacheinander angewendet wird.

Es soll evaluiert werden, ob dieser Ansatz in der Praxis die Wahrscheinlichkeit, dass es notwendig, ist eine komplette Neuinitialisierung durchzuführen, deutlich verringert wird und damit die Echtzeitanforderungen erfüllt werden können. Es soll überprüft werden, ob es möglich ist, sich mit diesem Verfahren frei durch einen Raum zu bewegen, ohne dass eine regelmäßige, komplette neue Initialisierungen notwendig ist. Diese gilt besonders in Bezug auf Situationen, in denen sich schnell viel ändert wie z.B. der Gang um eine Ecke. Zusätzlich soll überprüft werden, ob es sinnvoll ist, Punkte zu „sammeln“, also getrackte Punkte aus älteren Initialisierungsphasen zu speichern und somit eine größere Menge an Punkten zu Verfügung zu haben oder ob das System aufgrund des Driftens der KLM-Punkte ungenauer wird.

Sind die Veränderungen des ursprünglichen Verfahrens ausreichend, um die gestellten Echtzeitanforderungen zu erfüllen, muss überprüft werden, ob das hier vorgestellte Verfahren genau genug ist, um in der Praxis die Position exakt genug bestimmen zu können.

5.3 Generierung von 3D-Umgebungsmodellen

Um die Möglichkeit zu haben, virtuelle Objekte in Bezug zu echten Objekten zu setzen, muss eine 3D-Rekonstruktion der Umgebung stattfinden, falls in der ganzen Umgebung navigiert werden soll. Falls nur die Position im Verhältnis zu einzelnen Objekten bestimmt werden soll, muss eine 3D-Rekonstruktion dieser Objekte stattfinden. Um das Platzieren von virtuellen Objekten hinterher zu erleichtern, soll eine Art Karte der Objekte bzw. der Umgebung angefertigt werden.

Hierbei sollen zwei verschiedene Ansätze verfolgt werden. Das erste Ziel ist es, flache Objekte wie Bilder Postkarten oder Wände zu nutzen. Dies hat den Vorteil, dass es weitaus einfach ist, eine 3D-Rekonstruktion eines Objektes zu erzeugen, das nur aus einer 2D-Oberfläche besteht. Hierfür muss nur eine Aufnahme in 2D erzeugt werden, woraus die x- und y-Koordinaten übernommen werden können, der z-Koordinate kann, aufgrund der nicht vorhanden Tiefe, der Wert 0 zugewiesen werden.

Falls am Ende des Bearbeitungszeitraumes noch zusätzliche Zeit vorhanden ist, sollen im zweiten Schritt auch andere Objekte genutzt werden. Dabei soll wie folgt vorgegangen werden. Als Erstes sollen verschiedene überlappende Aufnahmen der Umgebung gemacht werden. Auf jeder Aufnahme lassen sich dann, mit einem geeigneten featurebasierten Tracking-Verfahren eindeutige Punkte bestimmen. Die Punkte einer Aufnahme werden mit den Punkten der anderen Aufnahmen verglichen und gleiche Punkte bestimmt. Durch Triangulation (siehe 4.5) wird aus der 2D-Position der Punkte auf verschieden Bildern die 3D-Position bestimmt. Diese 3D-Position soll dann in einer geeigneten Datenbank gespeichert werden. Die 3D-Position der einzelnen Punkte muss hierbei in dem gleichen Weltkoordinatensystem platziert sein, um eindeutige Positionen zu erhalten. Da die Punkte eindeutig sind, können sie später wieder bestimmt werden. Durch die bekannte 3D-Position lässt sich dann die eigene Position berechnen.

5.4 Platzierung der 3D-Objekte

Anhand des 3D-Umgebungsmodelles soll die Möglichkeit bestehen, die virtuellen Objekte auszurichten und der Entwickler so die Möglichkeit erhalten, die Objekte einfach und schnell zu platzieren. Die Objekte sollen Positionen entsprechend des Weltkoordinatensystems zugeordnet werden, anhand derer im späteren Verlauf festgestellt werden kann, ob ein virtuelles Objekt im Bild ist. Um das möglichst einfach zu gestalten, soll das 3D-Umgebungsmodell in einen Editor geladen werden können, um es zu bearbeiten. Hierfür empfiehlt es sich, einen Opensource Editor zu erweitern und an die speziellen Anforderungen anzupassen.

5.5 Positionsbestimmung im Raum

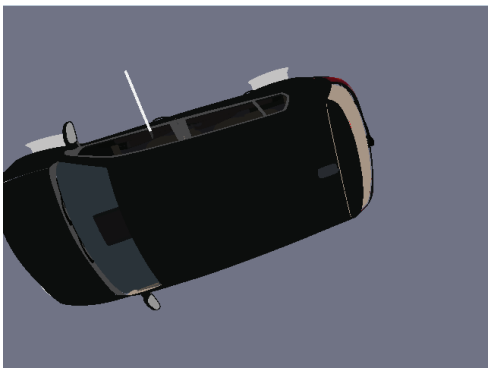
Bei der Positionsbestimmung soll wie folgt vorgegangen werden. Im ersten Schritt wird das zum Tracking genutzt Verfahren ausgeführt (siehe 5.2), aus den zugeordneten Punkten soll dann die Position berechnet werden. Für die Positionsberechnung sollen für die flachen Objekte und die tatsächlichen 3D-Objekte zwei verschieden Verfahren genutzt werden. Diese hat den Grund, dass es, wie in Abschnitt 4.3 beschrieben, wenn das Verfahren 4.2.2 für die Positionsberechnung von flachen Objekten genutzt wird, es zu einer singulären Lösungsmatrix kommt. Für die flachen Objekte soll das Verfahren, das in 4.3 vorgestellt wurde, genutzt werden. Hierbei wurde, die Homograhie zwischen zwei Flächen berechnet und daraus die Rotation und Translation rekonstruiert.

Für die tatsächlichen 3D Objekte soll mit dem Abschnitt 4.2.2 vorgestellten Verfahren die Projektionsmatrix bestimmt werden. Aus der Projektionsmatix lässt sich dann mit dem in im Abschnitt 4.2.2 beschriebenen Verfahren die Translation zum Ursprung des Weltkoordinatensystem bestimmen und die entsprechende Rotation.

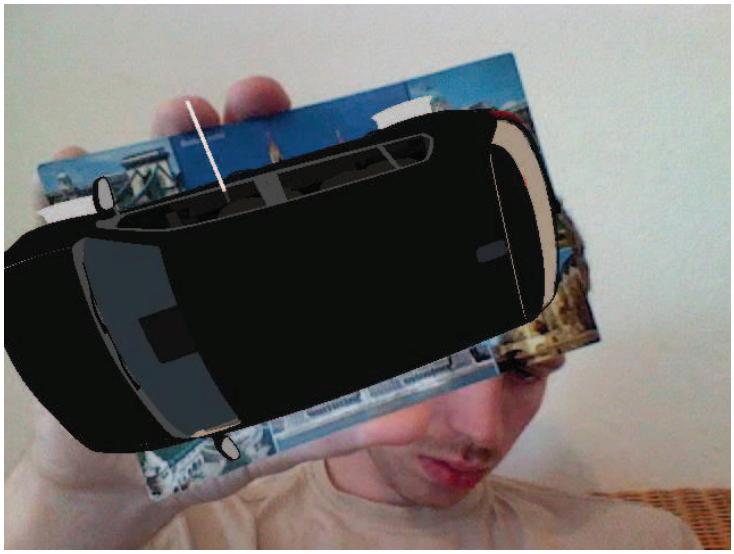
5.6 Darstellung

Ziel ist es, das Framework so offen zu gestalten, dass es möglich ist, beliebige 3D-Bibliotheken einzubinden. Mithilfe dieser Bibliotheken sollen dann eine virtuelle Welt erzeugt werden, deren Hintergrund das von der Kamera aufgenommene Bild ist. Das Weltkoordinatensystem in der virtuellen Welt entspricht dem in der echten Welt. Sobald sich die echte Kamera bewegt und die neue Position der Kamera bestimmt wurde, kann der virtuellen Kamera diese Position zugewiesen werden (als virtuelle Kamera wird der Blickwinkel des Users bezeichnet, dieser ist vergleichbar mit dem Bild einer Kamera die in die 3D-Welt schaut). Da nun im Hintergrund das Bild aus der realen Welt zu sehen ist und im Vordergrund sich die Objekte der virtuellen Welt befinden, wirkt es so, als sei die virtuelle Welt Teil der echten Welt.

(a) Reale Welt



(b) Virtuelle Welt



(c) Reale und virtuelle Welt zusammen

Abbildung 5.1: Aufbau der realen und virtuellen Welt.

6 Umsetzung

Dieses Kapitel beschreibt die Umsetzung des in Kapitel 5 vorgestellten Lösungsweges. Als Erstes werden genutzte Tools und Bibliotheken vorgestellt. Daraufhin wird der Aufbau der einzelnen Module des Frameworks beschrieben. Weiterhin werden ihre Funktionsweiße und Test zu den einzelnen Modulen erläutert.

6.1 Verwendete Tools und Bibliotheken

In diesem Abschnitt werden kurz die verwendeten Tools und Bibliotheken vorgestellt.

6.1.1 Irrlicht

Die Irrlicht Engine ist eine Open-Source Echtzeit 3D-Engine, die eine sehr hohe Performance bietet. Geschrieben und verwendbar ist sie in C++. Für .NET Sprachen, wie z.B. C# ist sie auch verfügbar. Irrlicht ist plattformunabhängig und läuft unter anderem unter Linux, Mac OS, Sun Solaris und diversen Windows-Versionen.

Für die grafische Darstellung werden OpenGL, DirectX oder eines von zwei Programmen zum Rendern genutzt (Software Renderer). Irrlicht unterstützt den Entwickler von Vertex- oder Pixelshader durch ihre einfache Integration. Es existieren Klassen und Funktionen für Matrix- und Vektorrechnung und einfache Kollisionsabfragen. Der Quellcode ist offen und in C++ geschrieben. Er steht unter der zlib-Lizenz, die auch eine Verwendung in kommerziellen Produkten ohne Herausgabe des Quelltextes ermöglicht. Viele bekannte Dateiformate werden von Irrlicht unterstützt und können direkt von Irrlicht eingelesen werden. Dies gibt dem Entwickler die Möglichkeit verschiedene Dateitypen ohne Konvertierung, zu nutzen. Irrlicht wird im Projekt genutzt um die Bildschirmausgabe zu generieren, also das Bild, das der Nutzer sieht. Dazu gehört die Darstellung des Kamerabildes sowie für die Darstellung und die Transformation der 3D-Objekte.

6.1.2 Blender

Blender ist eine freie 3D-Grafik-Software, die unter der GNU General Public License steht. Blender enthält Funktionen, um dreidimensionale Körper zu modellieren, sie zu texturieren, zu animieren und zu rendern. Blender ist für mehrere Betriebssysteme verfügbar, unter anderem Linux, Mac OS X und Microsoft Windows. Blender wurde in der Arbeit genutzt, um 3D-Modelle anzufertigen und vorhandene 3D-Modelle anzupassen.

6.1.3 OpenCV

OpenCV ist eine quelloffene Programmbibliothek unter BSD-Lizenz. Die Bibliothek ist plattformunabhängig und läuft unter anderem unter Linux, Mac OS, Sun Solaris und Microsoft Windows. Sie ist für die Programmiersprachen C und C++ geschrieben und enthält Algorithmen für die Bildverarbeitung und maschinelles Sehen. Die Entwicklung der Bibliothek wurde von Intel initiiert. Im September 2006 wurde die Version 1.0 herausgegeben. Die Stärken von OpenCV liegen in ihrer Geschwindigkeit und in der großen Menge der Algorithmen aus neuesten Forschungsergebnissen. Mit OpenCV wurden schon verschiedene kleine Augmented Reality-Projekte realisiert. OpenCV wird genutzt, um die Kameras zu Kalibieren, für Matrix-Operationen, außerdem werden verschiedene Bildverarbeitungsalgorithmen genutzt.

6.2 Testumgebung

Als Testumgebung wurde ein Thinkpad W500 mit 2,8 GHz genutzt. Für das Nachladen der aktuellen Webcam-Bilder wurde, wenn nicht anders angegeben, OpenCV genutzt. Zum Testen wurde jeweils ein Referenzbild genutzt, für das zu Beginn die entsprechenden Featurepunkte bestimmt wurden. Diese Punkte sollten dann jeweils im aktuellen Bild der Webcam wieder gefunden werden.

6.3 Klassenaufbau

Das Framework besteht im Wesentlichen aus 7 Klassen, die in Abbildung 6.1 dargestellt sind. Die ARManager-Klasse ist die Hauptklasse, über die die 3 Teilbereichte Punktbestimmung, Positionsschätzung und grafische Darstellung komunizieren. Die KLMPointFinder hat die Aufgabe, eindeutige Punkte zu bestimmen, die später zur Positionsschätzung genutzt werden. Es gibt mehrere Hilfsklassen, SURFHelper, PointManager und Homograhy, die vom KLMPointFinder genutzt werden.

Für die Positionsbestimmung ist die Klasse PositionManager zuständig, die auf Basis der von der Klasse KLMPointFinder gefunden Punkte versucht, die exakte Position zu bestimmen. Die bestimmte Position wird dann an die Klasse GrafikManager weitergegeben, die virtuelle Objekte abhängig von der Position in die Umwelt zeichnet. Die einzelnen Klassen werden im nun folgenden Teil ausführlich erläutert.

6.4 ARManager

Die ARManager Klasse übernimmt die Steuerung des Gesamtsystems. Ihre Aufgabe besteht darin, zu Beginn die Initialisierungs-Daten einzulesen und die Initialisierung der anderen Klassen zu veranlassen. Im laufenden Betrieb besteht ihre Aufgabe darin, Daten zwischen den anderen Klassen zu verteilen. Das heißt, sie leitet ermittelte Punkte der KLMPointFinder-Klasse an die

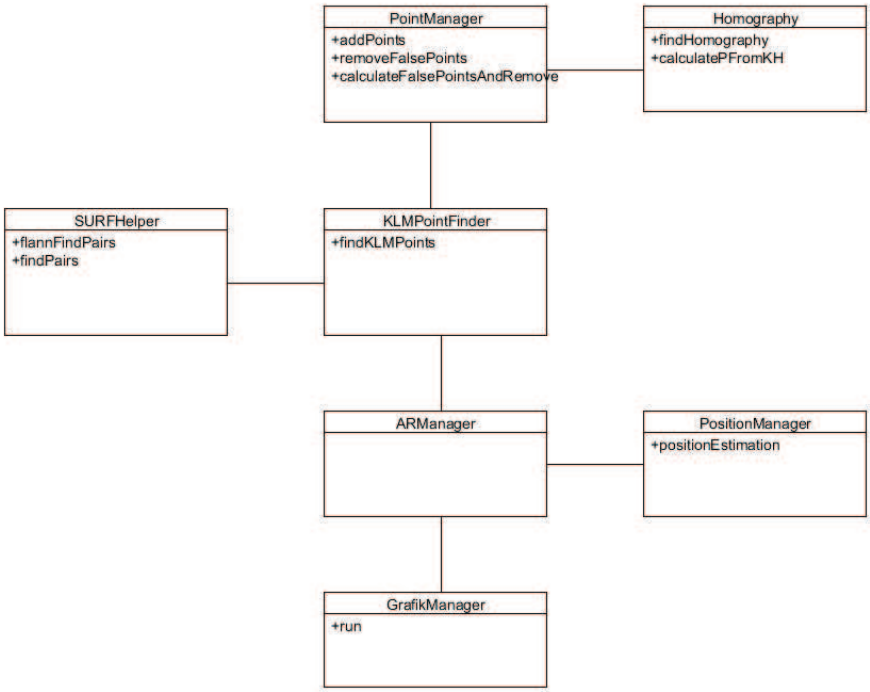


Abbildung 6.1: Klassendiagramm.

PositionManager-Klasse weiter und ihre berechneten Positionsdaten wiederum weiter an die GrafikManager-Klasse.

6.5 KLMPointFinder

In diesem Abschnitt wird die Klasse KLMPointFinder entwickelt. Diese Klasse hat die Aufgabe Featurepunkte zu finden und diese vorher bekannten Punkten zuzuordnen. Die vorher bekannten Punkte können z.B. aus einer Datenbank stammen. Im ersten Schritt wird dafür ein passendes, featurebasiertes Tracking-Verfahren gesucht. Im darauf folgenden Schritt wird der im Abschnitt 5.2 vorgestellte Lösungsansatz implementiert.

6.5.1 Auswahl des Verfahres zur Initialisierung

In Abschnitt 5.2 wurde vorgeschlagen, zur Optimierung des von Changhyun Choi, Seung-Min Baek und Sukhan Lee vorgestellten Verfahrens eine andere featurebasierte Tracking-Technologie zu nutzen, die eine bessere Performance hat als die Standard SIFT-Technik. Changhyun Choi, Seung-Min Baek und Sukhan Lee schlagen hierbei vor, SiftGPU zu verwenden. SiftGPU lagert Teile des SIFT-Prozesses auf die Grafikkarte aus und erreicht dadurch eine deutlich höhere Performance. Diese setzt allerdings voraus, dass eine entsprechende leistungsstarke

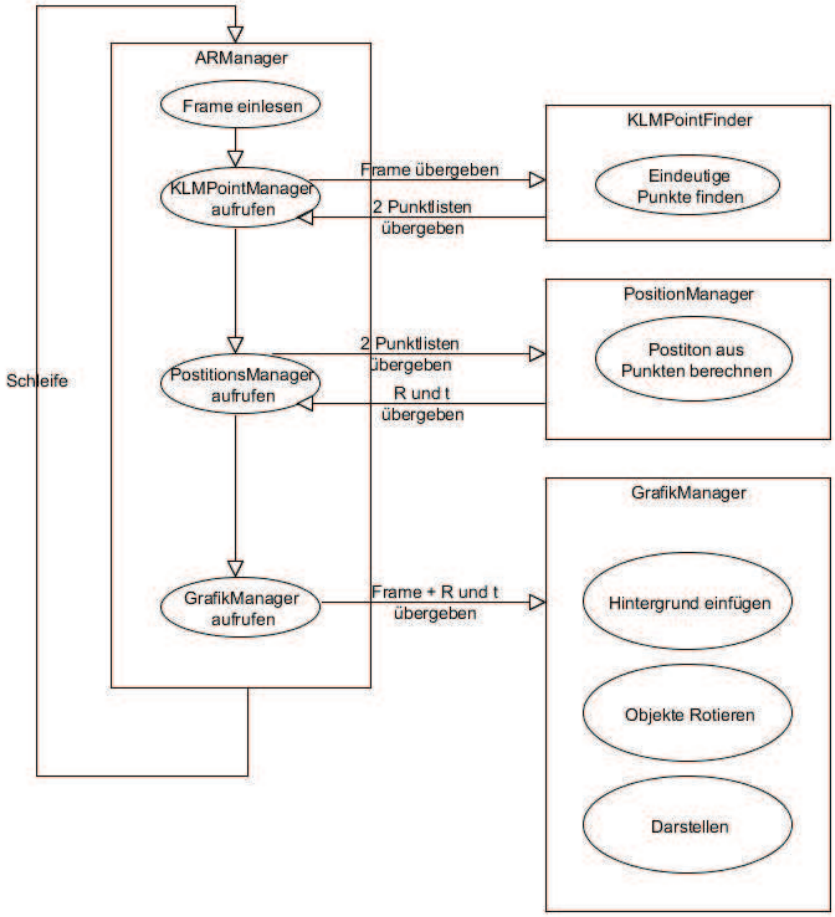


Abbildung 6.2: Ablaufdiagramm.

Grafikkarte vorhanden ist. Entsprechende Grafikarten sind aber speziell auf mobilen Geräten oft nicht vorhanden. Deshalb wurde das in Abschnitt 3.4 vorgestellte Verfahren SURF als Alternative getestet. Zum Testen wurden verschiedene kurze Video-Sequenzen genutzt. Wie in Abbildung 6.3 zu sehen, ist SURF deutlich schneller. Im Schnitt brauchte SURF für ein 640×480-Bild nur 155 ms, wohingegen SIFT meist deutlich mehr als 500ms benötigt.

6.5.2 Implementierung

Im ersten Schritt wurde das Verfahren, das von Changhyun Choi, Seung-Min Baek und Sukhan Lee vorgestellt wurde, umgesetzt. Hierfür wurden in der Klasse KLMPointFinder alle Algorithmen, die für das Finden und Verfolgen von Trackingpunkten notwendig sind, gekapselt. Dazu

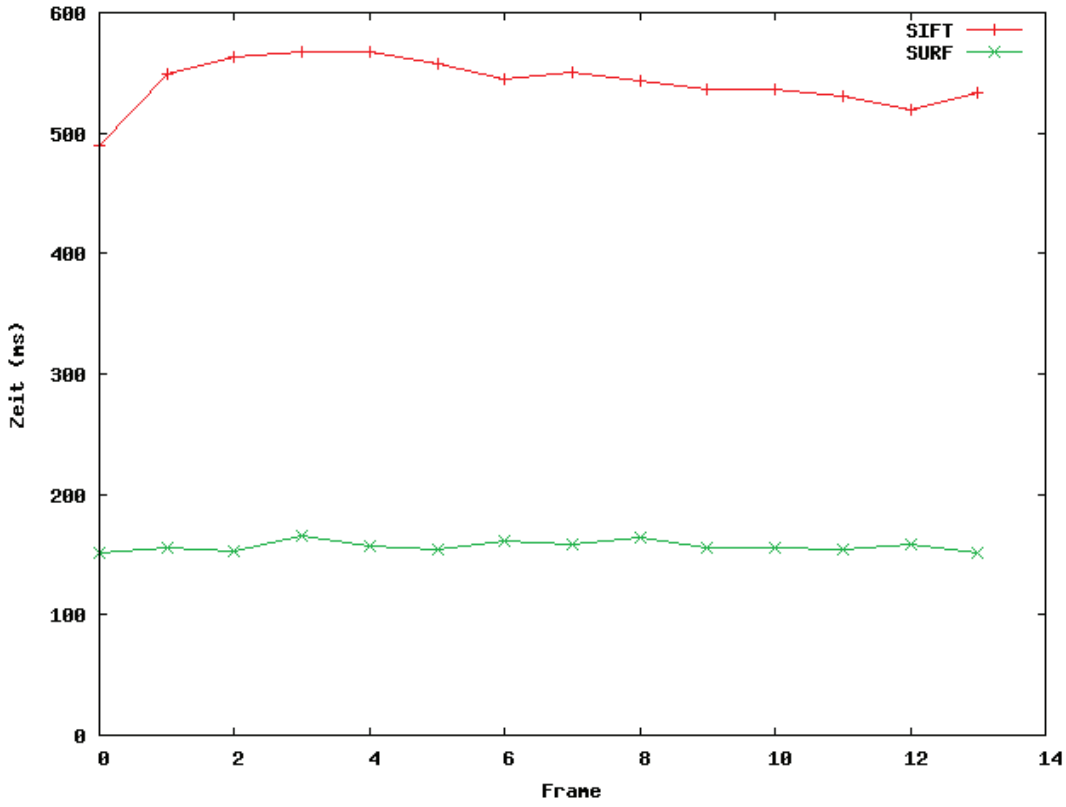


Abbildung 6.3: Performance vergleich von SIFT und SURF.

gehören das Ausführen des KLM Trackings und die Initialisierung mittels SURF sowie das Aussortieren von Ausreißern (siehe auch 6.6). KLM und SURF wurden nicht selbst implementiert, es wurde auf die Implementierung aus der OpenCV Bibliothek zurückgeriffen.

Im nächsten Schritt wurden die in Abschnitt 5.2 vorgeschlagen Verbesserungen getestet. Diese wurden implementiert, indem ein zweiter Thread eingefügt wurde, der in einer Periode von vier Frames[1] einen Initialisierungsprozess durchgeführt hat, also die Suche nach SURF-Punkten auf dem ersten Frame durchführt und anschließend die Position der Punkte im momentanen Bild mittels Optischer Fluss ermittelt.

Durch das kontinuierliche Hinzufügen von neuen Trackingpunkten waren deutlich mehr aktuelle Punkte vorhanden, was zu einer Verbesserung der Robustheit führte. Um die Möglichkeit zu haben, diese Prozesse auf mehrere Prozessorkerne zu verteilen, wurde eine weitere Verbesserung eingeführt. Statt ein Bild alle x Frames zu berechnen wird jeden Frame $\frac{1}{x}$ des Bildes berechnet. Es werden also nur Teilbereiche eines Bildes nach Freature Punkten durchsucht und sofort hinzugefügt. Abbildung 6.4 zeigt eine mögliche Aufteilung mit vier Teilbereichen.

Im ersten Frame werden nur die Featurepunkte für den linken oberen Bereich berechnet, der mit 1 markiert ist, im darauf folgenden Frame dann der mit 2 markierte rechte obere Bereich. Dies wird weitergeführt, bis alle Teilbereiche nach Featurepunkten durchsucht wurden und dann wird erneut gestartet.

[1] An dieser Stelle steht Frame für einen kompletten Berechnungsschritt, also Bild laden, Featurepunkte berechnen, Position bestimmen, Anzeigen des Augmented Reality-Bildes.

Es wird grundsätzlich das momentan aktuelle Bild zur Suche von Featurepunkten genutzt. So wird z.B. bei vier Teilbereichen wie in Abbildung 6.4 in vier Bildern jeweils ein Teil durchsucht. Das bedeutet auch, dass zwangläufig nicht immer alle Teile der Umgebung geupdatet werden, da durch Veränderung des Kamerabildes neue Elemente in Teilbereichen auftauchen können, die schon vorher durchsucht wurden. Da die Veränderungen zwischen einzelnen Bildern aber in der Regel sehr gering sind, hat das keine größeren Auswirkungen auf den Gesamtprozess.

Abbildung 6.4: Das Bild wird in vier Teilbereiche aufgeteilt. In jedem Frame wird ein Teilbereich bearbeitet.

Durch diese Veränderung wird es deutlich einfacher zu skalieren als im ersten Ansatz. Es ist möglich, ohne große Änderungen auf Prozessoren mit mehren Kernen zu arbeiten, indem anstelle eines Threads mehrere Threads gleichzeitig ausgeführt werden, die jeweils einen Teilbereich bearbeiten. Umgekehrt ist es auch möglich, durch Ändern der Größe der Teilbereiche, auf langsameren Prozessoren zu arbeiten. Es könnten z.B. anstelle von vier Bereichen auch acht Bereiche mit der halben Größe verwendet werden. Die Reaktionszeit des Systems wird außerdem erhöht, da immer mit dem aktuellen Bild gearbeitet wird und die Ergebnisse im aktuellen Frame direkt zur Berechnung der Position genutzt werden können.

Zusätzlich ist die Komplexität des Verfahrens geringer geworden, was darauf zurückzuführen ist, dass weder die Zwischenschritte gespeichert werden müssen noch die aktuelle Position der gefunden Punkte rekonstruiert werden muss. Dadurch verringert sich der Programmieraufwand und gleichzeitig steigt die Performance.

Zusatz

Eine zusätzliche Erweiterung wurde eingefügt, die sinnvoll nutzbar ist im Spezialfall wenn das Framework auf sehr langsamen Rechnern betrieben werden soll, die nicht in der Lage sind ein komplettes Bild in akzeptabler Zeit komplett upzudaten. Vorausetzung hierfür ist, dass ein klares Zielobjekt bekannt ist, das sich auch im Bild befindet. Zu Beginn muss eine einmalige Initialisierung stattfinden. So bald dies geschehen ist, kann die Position des Objektes berechnet werden. Unter der Annahme, dass sich zwischen zwei Frames die Position eines Objektes nur geringfügig ändert, kann im nächsten Frame dann an der Position des Objektes im Bild nach neuen Punkten gesucht werden.

Der Bereich, an dem sich das Objekt befindet, muss nicht komplett durchsucht werden, sondern es können auch nur Teilbereiche untersucht werden. Dadurch wird eine Optimierung bezügliche der Hardwareleistung des benutzten Rechners möglich. Auch dieser Ansatz bietet eine deutliche Verbesserung des ursprünglichen Ansatzes, auch wenn bei Verlust der Position eines Objektes eine Neuinitialisierung vonnöten ist. Er ist besonders geeignet, um auf mobilen Geräten genutzt zu werden, mit denen nur einzelne Objekte augmentiert werden sollen. Er kann z.B. auf Smartphones in Museen eingesetzt werden bei denen Bilder mit Hintergrundinformationen erweitert werden sollen.

6.5.3 Tests

In dem folgenden Abschnitt werden verschiedene Test beschrieben, bei denen Videos mit den unterschiedlichen Verfahren getestet und gegenübergestellt werden. Dabei wird das reine SURF mit dem von Changhyun Choi, Seung-Min Baek und Sukhan Lee vorgestellten Verfahren (SURF init + KLT) und den in dieser Arbeit vorgestellten Verbesserungen (SURF + KLT) verglichen. Als Testobjekt dient eine Postkarte. Um mögliche Ausreißer aussortieren zu können, müssen mindestens vier Punkte zur Berechnung der Homographie bekannt sein. Werden weniger als vier Punkte im Bild gefunden, wird der Wert zurück auf null gesetzt (siehe auch 6.6).

Video: Kippen

(a) Frame 1 (b) Frame 116

(c) Frame 200 (d) Frame 300

Abbildung 6.4: Auschnitte aus Video: Kippen einer Postkarte.

Im ersten Teil des Videos wird die Postkarte direkt in die Kamera gehalten (Abbildung 6.4.a). Ab Frame 100 wird die Postkarte langsam gedreht. Da die Postkarte mit Klarsichtfolie überzogen ist, kann es bei ungünstigen Lichtverhältnissen zu einer Spiegelung kommen. Dies geschieht in Frame 116 (Abbildung 6.4.b). Die Postkarte wird langsam weiter gedreht (Abbildung 6.4.c), bis sie schließlich fast horizontal zur Kamera liegt (Abbildung 6.4.d).

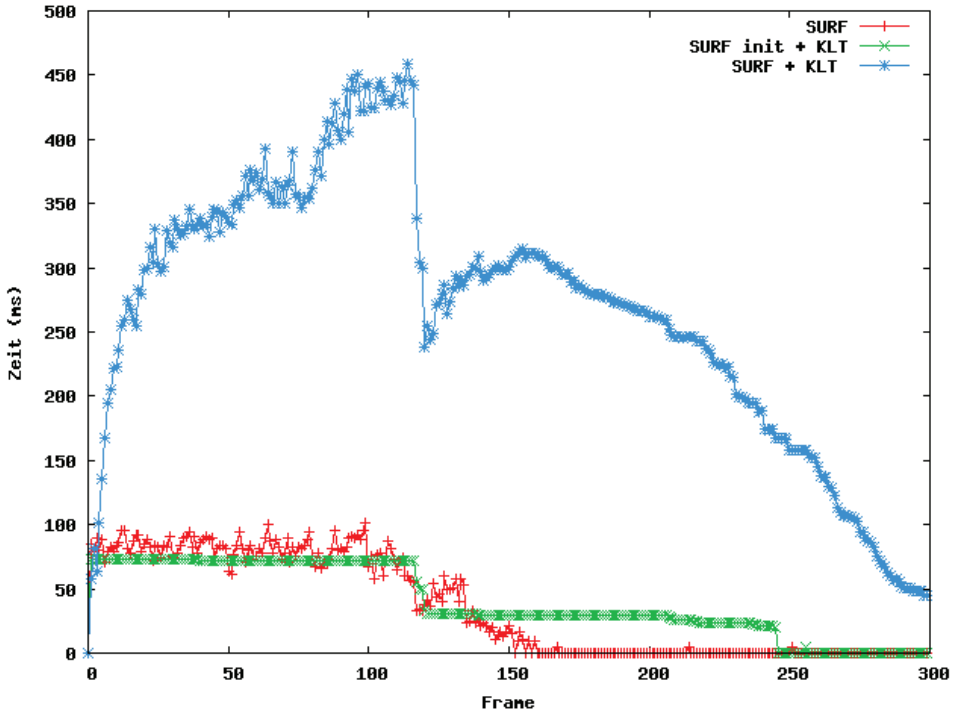
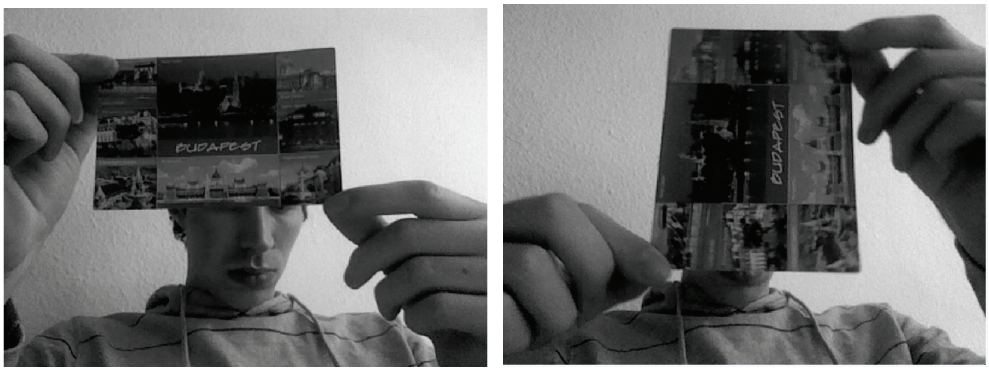


Abbildung 6.5: Gefundene Punkte in Video: Kippen.

In Abbildung 6.5 ist zu sehen, dass SURF + KLM durchgehend deutlich mehr Punkte gefunden hat als SURF init + KLT und SURF. Der Grund dafür ist, dass SURF + KLM kontinuierlich neue Punkte findet und diese hinzufügt - im Gegensatz zu SURF init + KLT, das nur zu Beginn initiale Punkte sucht und diese in den weiteren Frames verfolgt. SURF findet zwar in jedem Frame neue Punkte, verfolgt diese aber nicht und muss somit jeden Frame von vorne beginnen.

Beim Frame 116 ist deutlich zu sehen, dass durch das Spiegeln auf der Postkarte Punkte verloren gehen. Sowohl SURF init + KLT als auch SURF + KLM halbieren an dieser Stelle ca. ihre Punkte. SURF init + KLT verfolgt nur die noch vorhanden Punkte, wohingegen SURF + KLM neue Punkte ermittelt und diese weiterverfolgt. Umso weiter die Postkarte gedreht wird, umso weniger Punkte findet SURF, bis ab Frame 160 gar keine Punkte mehr gefunden werden. Dadurch wird deutlich, dass Algorithmen wie SURF nur bis zu einem bestimmten Grad rotationsinvariant sind. Der SURF + KLT-Algorithmus zeigt seine Stärke besonders deutlich zum Ende. Selbst in Frame 300, wo die Postkarte fast horizontal zur Kamera liegt hat der Algorithmus ca. genau so viel eindeutige Punkte wie die anderen Algorithmen zu Beginn.

Video: Drehung



(a) Frame 1 (b) Frame 115

(c) Frame 220

Abbildung 6.5: Ausschnitte aus dem Video: 180° Drehung einer Postkarte.

In Video 2 wird die Postkarte um 180° gedreht. Es ist deutlich zu sehen, dass sowohl SURF als auch SURF init + KLT über das ganze Video nicht die Grenze von 40 Punkten überschreitet, wohingegen SURF + KLT stetig bis zu einem Spitzenwert von 170 Punkten ansteigt und so im Schnitt ca. die 5- bis 6-fache Anzahl an eindeutigen Punkten vorhanden ist.

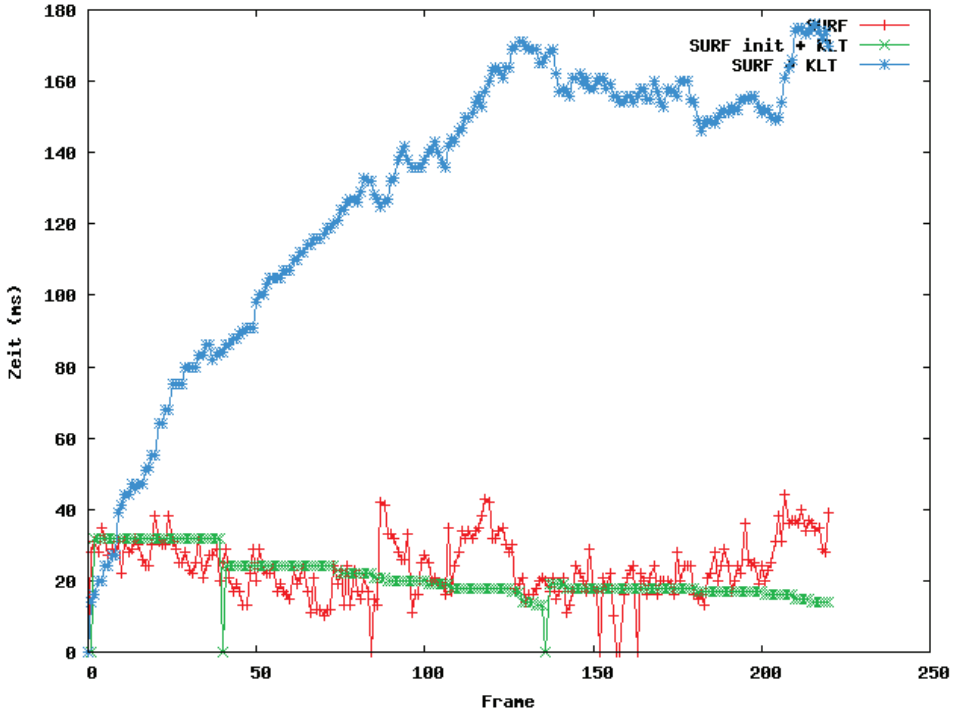


Abbildung 6.6: Gefundene Punkte in Video: Drehung.

(a) Frame 1 (b) Frame 35

Video: Verdeckung

In Video 3 wird, wie in Abbildung 6.6.b zu sehen ist, ein Teil der Postkarte im Verlaufe des Videos durch eine Hand verdeckt, die sich durch das Bild bewegt (ca. bei Frame 35). Für den reinen SURF-Algorithmus ist das kein Problem. Weil in jedem Frame neue Punkte bestimmt werden, die unabhängig von den im Frame zuvor bestimmten Punkten sind, führt die Hand nur dazu, dass im verdeckten Bereich keine Punkte mehr entdeckt werden. Bei den anderen beiden Algorithmen führt die Verdeckung zu einem deutlichen Verlust von Punkten. Beim SURF init + KLT-Algorithmus muss deshalb eine Neuinitialisierung durchgeführt werden. Der SURF + KLT-Algorithmus verliert ca. die Hälfte der vorhanden Punkte, hat allerdings mit ca. 35 Punkten immer noch mehr Punkte als die anderen beiden Algorithmen.

(c) Frame 65

Abbildung 6.6: Auschnitte aus Video: Teilweise Verdeckung einer Postkarte.

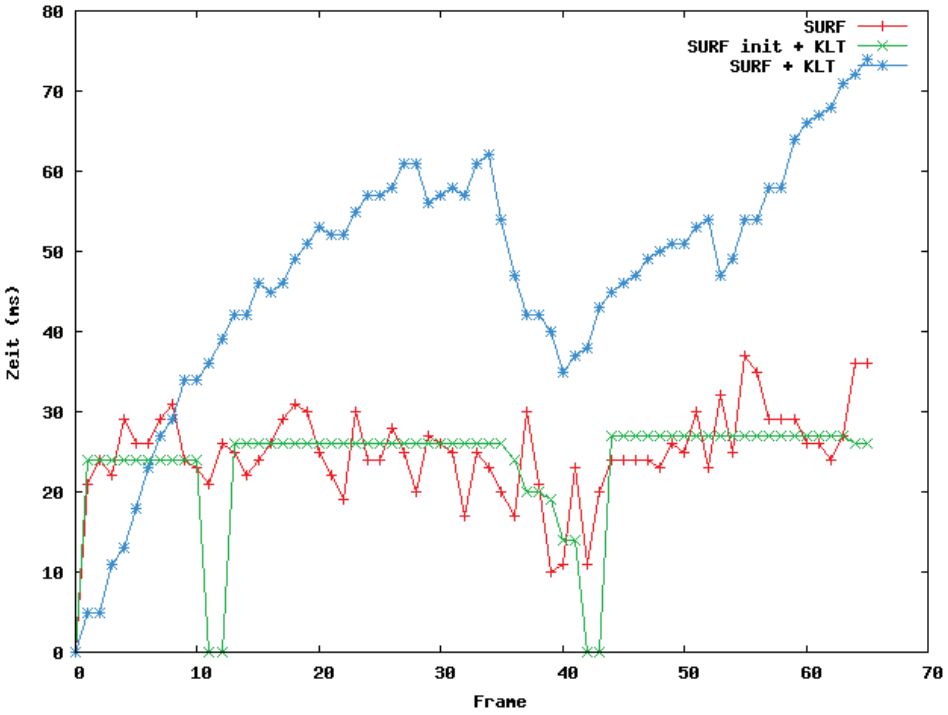


Abbildung 6.7: Gefundene Punkte in Video: Verdeckung.

Zusammenfassung

In den Tests wurde deutlich, dass der hier entwickelte Algorithmus, der eine Kombination aus SURF und KLT darstellt, den anderen beiden Varianten in jeder Situation deutlich überlegen ist. Auch in schwierigeren Situationen ist es mithilfe von SURF + KLT möglich, genug Punkte zur Bestimmung der Position festzulegen. Indem durchgängig neue Punkte mittels SURF hinzugefügt werden, die dann mit KLT verfolgt werden können, werden die Vorteile von beiden Verfahren kombiniert. Die Nachteile der ursprünglichen Algorithmen, wie das beim KLM-Tracker, bei Verlust von Punkten diese nicht mehr wiedergefunden werde oder das SURF nicht performant genug ist für den Echtzeit Einsatz, treten nicht mehr auf oder sind deutlich geringer.

Mögliche zukünftige Weiterentwicklung:

Trotz der deutlich verbesserten Performance kann davon ausgegangen werden, dass das Verfahren auf Smartphones nur sehr begrenzt nutzbar ist. Um an dieser Stelle bessere Ergebnisse zu erreichen, könnte statt SURF das von Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond und Dieter Schmalstieg in [WRM+08] vorgestellt Verfahren PhonySIFT genutzt werden. Dieses ist ein für Smartphones optimiertes Verfahren. Es nutzt statt dem Verfahren das SIFT nutzt zum finden von Featurepunkten (siehe 3.3, einen Corner Detector wie der in 3.2 vorgestellte Harris Corner Detector. Dieser Corner Detector wird mit dem in Abschnitt 3.5 vorgestellten SIFT-Deskriptor kombiniert. Um nicht wie in SIFT verschiedene Auflösungen durchsuchen zu müssen, werden bei PhonySIFT Featurepunkte aus möglichst vielen verschiedenen Auflösungen gespeichert. Durch diese Optimierung kann das Verfahren auch auf Smartphones genutzt werden. Wird dieser Algorithmus statt SURF genutzt ist eine zusätzliche Performanceverbesserung zu erwarten.

Eine weitere Verbesserung könnte stattfinden wenn mit 3D-Modellen der zu trackenden Objekte gearbeitet wird. Unter der Annahme, dass die Position des Objektes bestimmt wurde und das Modell zu diesem Objekt bekannt ist könnten zusätzliche KLM-Punkte hinzugefügt werden indem anhand der Position des Objektes und seines Modells bestimmt wird, welche Bereiche des Bildes zum Objekt gehören. In diesen Bereichen können dann Punkte hinzugefügt werden.

6.6 PointManager

Um alle Operationen zu kapseln, die direkt dafür verantwortlich sind, die Integrität der Punkte sicherzustellen, wurde die Klasse PointManger erstellt. Sie enthält eine Liste aller Punkte, die im momentanen Bild vorhanden sind und eine Liste der zugehörigen Punkte aus der Datenbank. Um die Integrität zu gewährleisten, enthält die Klasse mehrere Sicherheitsfunktionen. Dazu gehört, dass überprüft wird, ob ein neu hinzugefügter Punkt sich schon in den Listen befindet. Ist dies der Fall, wird der Punkt geupdatet.

Die Klasse wurde aber bewusst nicht so ausgelegt, dass nur Zugriffe über getter und setter möglich sind. Dies führt zwar zu dem Risiko, dass bei falscher Benutzung die Integrität der Punkte nicht sichergestellt werden kann. Es wird aber verhindert, dass durch unnötiges Kopieren Performance-verluste auftreten. Zusätzlich entstehen durch Anfertigung von Kopien verschiedene Versionen der Liste, was ohne zusätzliche Sicherheitsfunktionen zwangsläufig dazu führt, dass neue Werte von alten überschrieben werden.

Beispiel

Zwei Kopien werden erstellt: Liste A und Liste B. Liste A wird in Thread 1 und Liste B in Thread 2 verarbeitet. Thread 1 hat die neue Position von Punkt X ermittelt, sobald Thread 1 fertig ist, wird Liste A wieder der Originalliste hinzugefügt und Punkt X wird geupdatet. Wenn danach Thread 2 fertig ist, wird die Liste B der Originalliste hinzugefügt. Punkt X hat aber in der Liste B noch den alten Wert. Dass es sich dabei um einen alten Wert handelt, weiß die Originalliste nicht und fügt den Wert von Punkt X als neuen Wert hinzu.

Es bestände die Möglichkeit, dieses Problem zu verhindern, indem jedem Punkt eine Sequenznummer zugeordnet wird, die bei jedem Update um 1 erhöht wird und nur Updates stattfinden wenn die neue Sequenznummer höher ist als die alte. Dies führt aber zu zusätzlichem Rechenaufwand, der an dieser Stelle nicht erwünscht ist. Um trotzdem die Integrität sicherzustellen wenn mehrere Threads neue Punkte finden, wurde durch Semaphoren der Schreibzugriff auf einen Thread beschränkt.

Die Klasse bietet zusätzlich noch die Möglichkeit, falsch zugeordnete Punkte zu eliminieren. Das geschieht, indem die Homographie unter Zuhilfenahme des RANSAC-Algorithmus berechnet wird. Hierbei werden, wie in 4.6 schon erläutert, Punkte, die nicht in das Modell passen, aussortiert. Diese Punkte werden sofort gelöscht. Problematisch bei diesem Ansatz ist, dass eine Mindestanzahl von vier Punkten vorhanden sein muss, um die Homographie zu berechnen und um mehr Ausreißer sicher zu finden. Dies führt zu dem Konflikt, dass entschieden werden muss, wenn keine richtige Homographie gefunden wurde, ob dann alle Punkt gelöscht werden oder alle gespeichert werden.

Dies ist eine entscheidende Frage. Es hat sich in Tests gezeigt, dass gerade dann, wenn das zu trackende Objekt nicht im Bild ist, trotzdem ein paar (falsche) Punkte gefunden werden. Werden diese Punkte nicht direkt gelöscht, führt das dazu, dass im späteren Verlauf des Programmes davon ausgegangen wird, dass diese Punkte richtig sind und entsprechend die Position bestimmt wird, was wiederum bedeutet, dass ein virtuelles Objekt irgendwo in den Raum gezeichnet wird. Dies führt dazu, dass wenn der User durch eine Umgebung läuft, für die es keine virtuellen Objekte gibt, regelmäßig irgendwelche Objekte auftauchen. Dies könnte dann vom User als störend empfunden werden.

Werden allerdings die Punkte sofort gelöscht, könnte das dazu führen, dass wenn nur mit kleinen Teilbereichen gearbeitet (siehe 6.5) wird, die Anzahl der gefunden Punkte zu gering ist, um die Homographie zu bestimmen. Somit müssen jedesmal alle Punkte wieder gelöscht werden, obwohl es korrekte Punkte sind. Das führt dazu, dass nie genug Punkte gesammelt werden, um die Homographie zu berechnen. In der Praxis mit vier Teilbereichen eines 640×480-Bildes hat sich gezeigt, dass dies nicht zu Problemen führt. Sollten in einem anderen Projekt kleine Teilbereiche gewählt werden, könnte das Problem umgangen werden, indem frühere Ergebnisse angesehen werden, indem z.B. nur alle vier Frames dahingehend überprüft werden ob die gefunden Punkte korrekt sind.

6.6.1 Problem: Outliner finden

Wie weiter oben schon beschrieben, werden fehlerhafte Punkte mithilfe des RANSAC-Algorithmus aussortiert. Beim RANSAC-Algorithmus wird versucht, ein Modell zu berechnen, bei dem möglichst viele Punkte Inliner sind. Das funktioniert wie in 4.6 erläutert, indem eine Untermenge der vorhandenen Punkte ausgewählt und daraus ein Modell berechnet wird. Für das Modell wird dann die Anzahl der Inliner bestimmt. Als Inliner gelten Punkte, deren Distanz zum Modell kleiner als ein vorher festgelegter Schwellwert ist.

Problematisch an diesem Verfahren ist, dass sich eine Rotation auf Punkte die weiter entfernt vom Ursprung sind, stärker auswirkt als auf Punkte, die nicht so weit entfernt sind. Dies wird in Abbildung 6.8 dargestellt. Die schwarze Linie repräsentiert das „richtige“ Modell, die rote Linie

ein Modell, das um den Winkel α verschoben ist. Wie deutlich zu sehen ist, ist der Abstand des Punktes p_1, der näher am Ursprung liegt, zum rotierten Modell deutlich kleiner als der Abstand von Punkt p_2 zum rotierten Modell.

Da ein Modell aufgrund von Messungenauigkeiten nie ganz genau berechnet werden kann, führt dieses Verhalten dazu, dass weiter entfernte Punkte eher als Outliner gelten und somit eher aussortiert werden, selbst wenn ihre Position exakt stimmt. Um diesem Problem aus dem Weg zu gehen, wurde ein sehr hoher Schwellwert angesetzt. Dies führt nicht zu Problemen, da an dieser Stelle im Programm nicht Punkte aussortiert werden, die ungenau sind, sondern falsch zugeordnete Punkte, die in der Regel sehr viel weiter vom Modell entfernt sind als der festgelegte Schwellwert.

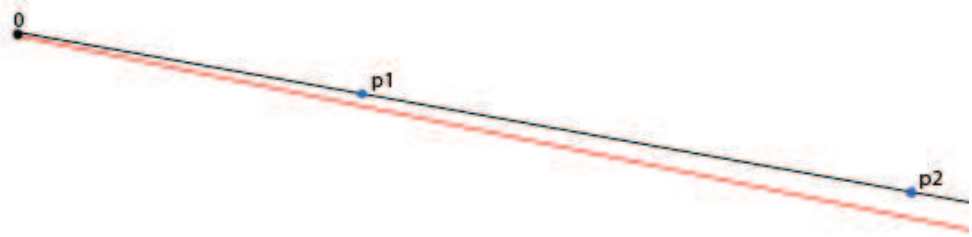


Abbildung 6.8: Auswirkungen der Rotation eines Modelles.

Um die Möglichkeit zu haben, diesen Algorithmus zu nutzen und nicht nur falsch zugeordnete Punkte auszusortieren, sondern auch ungenaue Punkte, kann das Verfahren optimiert werden, indem anstatt der echten Position die normierte Position genutzt wird. Dadurch sind alle Punkte gleich weit vom Ursprung entfernt und somit kann ein Schwellwert definiert werden, der für alle Punkte die gleiche Aussage hat.

6.7 SURFHelper

Nachdem Featurepunkte gefunden wurden, besteht das Problem darin festzustellen, ob diese Punkte bekannt sind und somit zu einem bekannten Objekt bzw. zur Karte der Umgebung gehören. In diesem Abschnitt wird die übliche Vorgehensweise hierfür erläutert, die auftretenden Probleme werden dabei besprochen und der Lösungsweg aufgezeigt, der in der Klasse SURFHelper implementiert wurde.

6.7.1 Fast approximate nearest neighbors [ML09]

Umgesetzt wird die Suche des nächsten Nachbarn durch kd-Bäume mit einer von Marius Muja und David G. Lowe entwickelten Bibliothek FLANN (Fast approximate nearest neighbors). Die FLANN-Bibliothek basiert auf den Auswertungen, die in [ML09] gemacht wurden, hier wurden verschiedene Algorithmen für die Verwendung von kd-Bäumen verglichen, wobei Muja und Lowe zu folgendem Ergebnis kamen:

> In our experiments, one of two algorithms obtained the best performance, depending on the dataset and desired precision. These algorithms used either the hierarchical k-means tree or multiple randomized kd-trees.

6.7.2 Tests

In diesem Abschnitt wird die Suche nach dem nächsten Nachbarn mit lineare, multiple randomized kd-Baum und hierarchical k-means Baum Suche verglichen. Hierfür wurde jeweils eine Videosequenz nach Featurepunkten durchsucht und anschließend die nächste Nachbar Suche mit den unterschiedlichen Algorithmen durchgeführt. In der Datenbank waren 1450 Featurepunkte. In der Videosequenz wurden jeweils ca. 2500 - 3000 Featurepunkte gefunden.

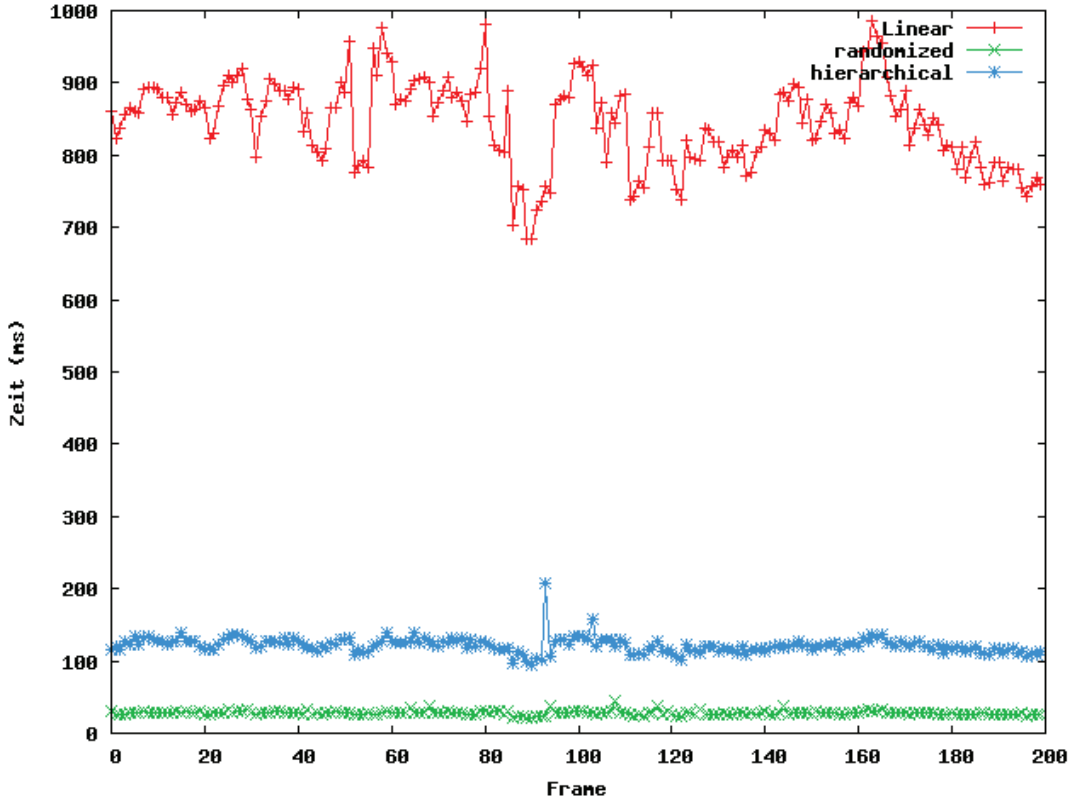


Abbildung 6.9: Linare Suche,multiple randomized kd-Bäume und hierarchical k-means Bäume Vergleich.

Abbildung 6.9 zeigt die Zeit in ms, die pro Frame zur Bestimmung des nächsten Nachbarn mit den unterschiedlichen Algorithmen gebraucht wurde. Es ist zu sehen, dass die lineare Suche deutlich länger braucht als die Suche mit kd-Bäumen. Wobei die Suche mit multiple randomized kd-Bäumen im Schnitt nur $\frac{1}{3}$ der Zeit beansprucht, die die hierarchical k-means Bäume benötigen. In Abbildung 6.10 wird die Anzahl der gefunden Punkte verglichen, es ist zu sehen, dass die drei Algorithmen im Wesentlichen das gleiche Ergebnis liefern.

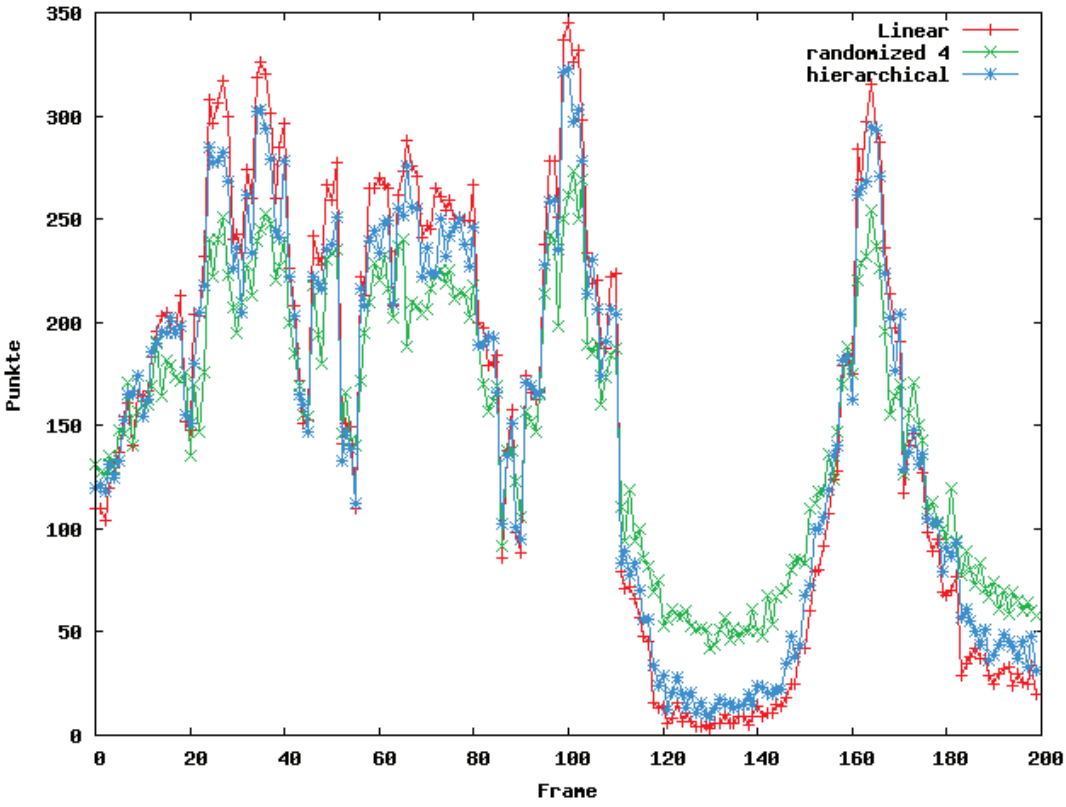


Abbildung 6.10: Gefundene Featurepunkte Vergleich.

6.7.3 Umsetzung

Wie im vorherigen Abschnitt festgestellt, haben die multiple randomized kd-Bäumen die beste Performance gezeigt. Darum wurde die Nächster-Nachbar-Suche mithilfe von multiple randomized kd-Bäume umgesetzt. Hierfür werden zum Start der Anwendung mehrere Sets von multiple randomized kd-Bäume erzeugt. Die Anzahl hängt davon ab, wie viele Threads auf die kd-Bäume zugreifen. Im späteren Betrieb müssen dann nur die gefunden Featurepunkte an die entsprechende FLANN Funktion übergeben werden, die dann auf Basis der kd-Bäume den nächsten Nachbarn ermitteln.

6.8 GrafikManager

Wie in Abschnitt 5.6 beschrieben, soll es möglich sein, dass Framework mit unterschiedlichen Grafikbibliotheken zu nutzen. Dafür wurde das GrafikManger-Interface entwickelt. Dieses Interface besteht im Wesentlichen aus einer Funktion, der ein Bild, die Rotation und die Translation der Kamera übergeben werden.

```
int run(IplImage* frame, const float R[3], const float t[3]);
```

Eine Anwendung kann nun direkt in einer Implementierung des GrafikManger Interfaces stattfinden. Diese Kapselung der Anwendungsdaten hat den Vorteil, dass die eigentliche Anwendung

komplett unabhängig von der Positionsbestimmung entwickelt werden kann. In der Entwicklungsphase können die Bewegungen ganz normal über die Tastatur gesteuert werden und erst später werden die Positionsdaten dafür genutzt. Das führt zu einer weniger aufwändigen Entwicklungsphase und ermöglicht es auch, Teile getrennt zu entwickeln.

6.8.1 IrrGrafikManager

Der IrrGrafikManager implementiert das Interface GrafikManager und nutzt hierfür die Grafikbibliothek Irrlich. Zur Darstellung der realen Welt wird das übergebene Bild auf eine Textur übertragen und als Hintergrund festgelegt. Die virtuellen Elemente werden durch entsprechende Irrlich-Funktionen erzeugt. Die Positionensbestimmung der virtuellen Objekte geschieht, indem zur tatsächlichen Position des Objektes die übergebene Kameraposition hinzugerechnet wird. Dafür wird zuerst der Positionsvektor des Objektes rotiert und dann der Translationsvektor hinzu addiert. Damit nicht immer zwischen tatsächlicher Position und Position im Kamerabild unterschieden werden muss, wurde diese Funktionalität in einer Klasse gekapselt.

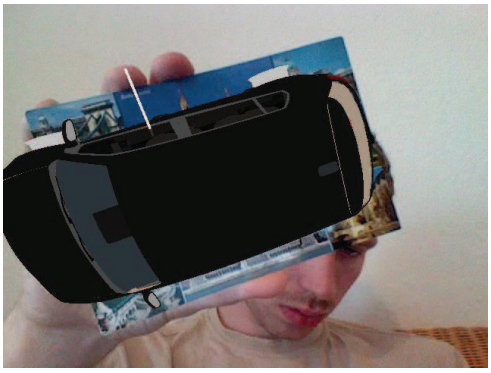
6.9 Positionsbestimmung

In diesem Abschnitt wird erläutert, wie die Position berechnet wird. Der ursprüngliche Ansatz bestand darin, das in Abschnitt 4.3 vorgestellte Verfahren zu nutzen, bei dem aus 2D-Punkt-Koordinaten die Homographie bestimmt wird. Aus dieser kann dann mit dem auch in 4.3 vorgestellten Verfahren die genaue Translation und Rotation zwischen den beiden Bildern bestimmt werden. Allerdings hat sich sehr schnell gezeigt, dass dieses Verfahren nicht stabil genug ist, was sich in der Praxis durch leichtes Wackeln der virtuellen Objekte bemerkbar macht. Wie unstabil das Verfahren ist, wird deutlich, wenn die Translation und Rotation zweimal aus dem gleichen Bild, aber mit unterschiedlichen Punkten berechnet wird. Für Bild 6.11.a wurde zweimal die Rotation bestimmt, diese ist bei der ersten Berechnung $(-0.722369 - 14.7725 - 18.7725)$ und der zweiten $(0.3 - 14.8892 - 18.1377)$ (Angaben in Grad). Somit gibt es eine absolut Differenz zwischen den beiden Berechnungen von $1.7739°$. Diese Ungenauigkeit fällt besonders störend auf, wenn ein virtuelles Objekt aus sehr geringer Entfernung betrachtet wird, wie in Abbildung 6.11.b zu sehen ist. Der Unterschied zwischen den beiden Berechnungen wird deutlich, wenn der „optical flow“ betrachtet wird (6.11.c). Dieser zeigt, an welcher Stelle im Bild eine Bewegung stattgefunden hat.

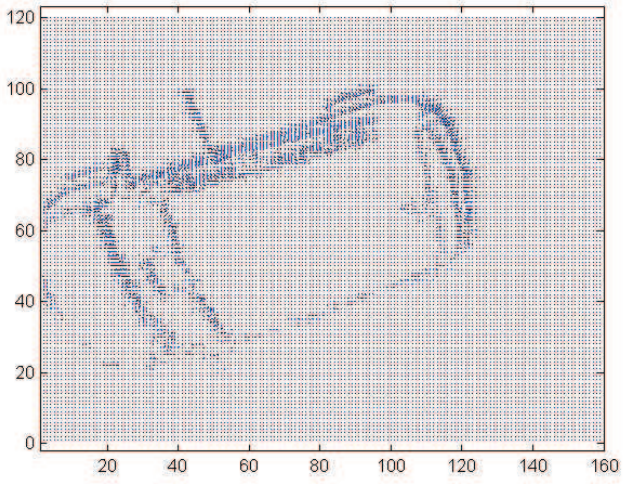
6.9.1 Robust Pose Estimation from a Planar Target

Als Alternative zu dem Verfahren Homographie wurde ein von Gerald Schweighofer und Axel Pinz entwickeltes Verfahren genutzt [Sch06b]. Dieses Verfahren ist speziell für ebene Objekte entwickelt worden und ist auch Teil des ARToolKit+ [ART09c], das zu den markerbasierten Augmented Reality Frameworks gehört. Das Verfahren besteht im Wesentlichen aus zwei Teilen. Im ersten Schritt wir mit einem beliebigen anderen Verfahren eine erste Schätzung der Position durchgeführt. Ein anderes Verfahren können z.B. die Berechnung mithilfe der Homograhie

(a) Bild das zur berechnung von Rotation und Translation genutzt wird.



(b) Bild mit virtuellen Auto.



(c) Optical flow.

Abbildung 6.11: Ungenauigkeit der Positionsbestimmung mittels Homograhie.

sein oder es kann auch die Position des letzten Frames genutzt werden. Im zweiten Schritt wird nun versucht, den Fehler zu minimieren, also eine möglichst genau Schätzung zu erreichen. Diese geschieht mithilfe eines eigens entwickelten Algorithmus und ermöglicht so sehr genaue Ergebnisse.

6.9.2 Vergleich

Um die beiden Verfahren zu vergleichen, wurde ein mathematischer Weg gewählt, es wurde also nicht mit echten Kameraaufnahmen gearbeitet. Stattdessen wurde ein Modell festgelegt. Dieses Modell wurde um zufällige Werte im Raum transformiert und dann mithilfe von zwei Projektionsmatrizen abgebildet. Um Messungenauigkeiten vorzutäuschen, wurden die so entstanden Bildkoordinaten um einen zufällig gewählten Wert verschoben. Insgesamt wurden 10 Tests durchgeführt in denen die maximale Verschiebung jeweils um 1 erhöht wurde. Also beim 10 Durchlauf wurden die tatsächlichen Bildkoordinaten maximal um 10 Pixel verändert. Zusätz-

lich wurde jeder Test insgesamt 100 ausgeführt und dann wurde das Mittel gebildet. Das hat sichergestellt, dass nicht aufgrund von unvorteilhaften Werten eine Abweichung der Ergebnisse entstanden ist. Es wurden jeweils die berechnete Translation mit der original Translation verglichen und der absolute Abstand ermittelt.

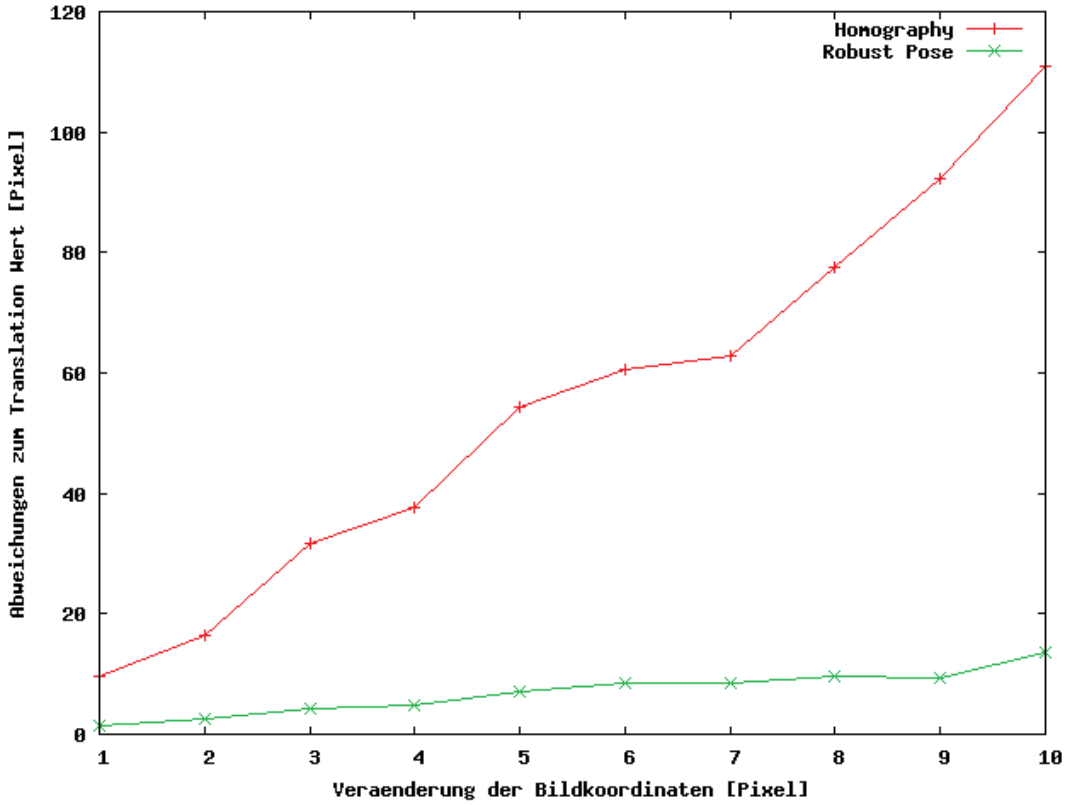


Abbildung 6.12: Vergleich zwischen Translationsberechnung mittels Homographie und der “Robust Pose Estimation from a Planar Target“-Methode.

In Abbildung 6.12 sind die Ergebnisse dieses Tests zu sehen, ist wird deutlich, dass die “Robust Pose Estimation from a Planar Target“-Methode im Schnitt knapp um den Faktor 9 besser ist als die Berechnung der Translation mithilfe der Homographie. Die Tests haben auch gezeigt, dass die “Robust Pose Estimation from a Planar Target“-Methode deutlich langsamer ist als die Berechnung mittels Homographie. In der Regel brauchte die “Robust Pose Estimation from a Planar Target“-Methode auf dem Testsystem 5-10ms. Zur Bestimmung der Position mittels Homographie und anschließender Zerlegung wurde deutlich weniger als 1 ms gebraucht.

Im Framework wurde „Robust Pose Estimation from a Planar Target“-Methode implementiert, da die Berechnungszeit noch ausreichend schnell war, um die Vorgabe der Echtzeit zu erfüllen und die genauere Berechnung der Position das System deutlich robuster macht.

6.10 Editor

Um die Möglichkeit zu haben, Augmented Reality möglichst komfortabel nutzten zu können, sollte ein entsprechender Editor vorhanden sein. Um möglichst wenig Kompatibilitätsprobleme

zu haben, wurde der zur Irrlicht Engine gehörende Editor IrrEdit verwendet. IrrEdit ist ein freier Echtzeit 3D-Welt-Editor und Light Map-Generator. Er bietet die Möglichkeit, verschiedene 3D-Datei-Formate (unter anderem 3D-Studio meshes und Quake 3 Levels) einzulesen und aus verschieden Objekten eine 3D-Welt zu erzeugen. Zusätzlich bietet IrrEdit Features an wie ein Particlesystem oder die Erzeugung von Terrains. Mit IrrEdit erzeugte 3D-Welten können direkt von Irrlich eingelesen werden.

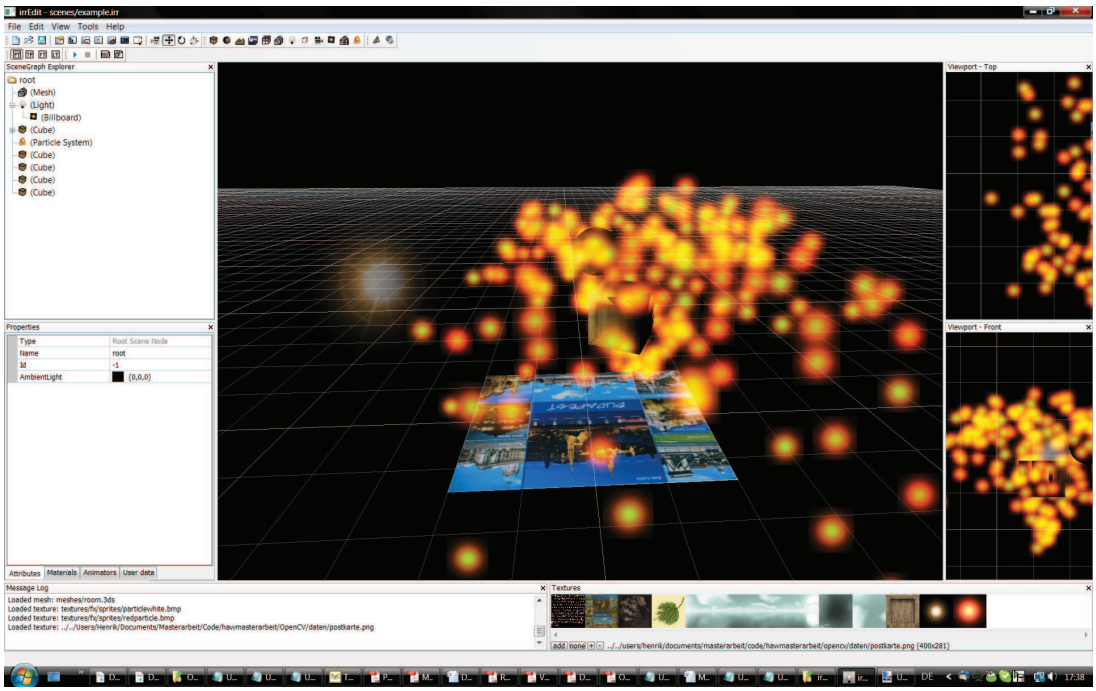


Abbildung 6.13: Szene in IrrEdit.

Mithilfe von IrrEdit ist es möglich, Welten für eine Augmented Reality direkt im Editor zu erzeugen. Hierfür wurde IrrEdit nicht verändert, stattdessen wurden mehrere Designrichtlinien festgelegt. Auf Basis dieser Richtlinien ist es möglich, die IrrEdit Datei einzulesen und korrekt in die 3D-Welt einzupassen. Die wichtigste Regel legt fest, dass ein spezielles Objekt angelegt werden muss. Dieses Objekt dient als Repräsentation des virtuellen Markers und enthält als Textur das Bild, das zum Tracken genutzt wird. So ist es möglich, eine komplette Augmented Reality-Anwendung nur auf Basis einer mittels IrrEdit erstellen 3D-Szene-Datei zu erzeugen.

6.11 3D-Umgebung

Wie im Lösungsansatz in Abschnitt 5.5 erwähnt, sollen neben 2D-Objekten auch 3D-Objekte genutzt werden. Hierfür muss vorher die 3D-Position von SURF-Punkten bestimmt werden. Dies wurde nur testweise in Matlab umgesetzt und aus Zeitgründen nicht im Framework implementiert.

Das Vorgehen hierbei entsprach im Wesentlichen den im Abschnitt 4.8 vorgestellten Verfahren. Es wurden zwei Aufnahmen eines 3D-Objektes genommen. Hieraus wurde die Fundamentalmatrix bestimmt aus der dann mithilfe der Essentialmatrix die beiden Projektionsmatrizen bestimmt

Abbildung 6.14: IrrEdit Szene aus 6.13, dargestellt mithilfe des Frameworks.

wurden. Daraus wurde dann mithilfe der Triangulation die 3D-Position bestimmt. Diese wird Dargestellt in Abbildung 6.15. Abbildung 6.15.a und 6.15.b zeigen die beiden Aufnahmen und 6.15.c zeigt die berechneten 3D-Positionen. Die Punkte wurden an dieser Stelle der Übersicht halber von Hand ausgewählt. Die Punkte befinden sich jeweils auf drei Außenkanten des abgebildeten Kartons.

Ist die 3D-Position der SURF-Punkte bekannt, lässt sich nun mit dem gleichen Verfahren, das hier für die 2D-Bestimmung genutzt wurde, die 3D-Position bestimmen. Somit könnte das 3D-Verfahren relativ leicht mit ins Framework integriert werden.

(a) Kamera Links

(b) Kamera Rechts

(c) 3D Rekonstruktion

Abbildung 6.15: 3D-Rekonstruktion einer Szene aus zwei Kameraaufnahmen.

7 Evaluierung des Gesamtergebnisses

An dieser Stelle findet eine Evaluierung des Gesamtergebnisses statt, das System wird mit den in Abschnitt 5.1 festgelegten Zielen vergleichen und die Praxistauglichkeit wird überprüft. Hierfür wird eine kleine Demoanwendung entwickelt und das System in verschiedenen Extremfällen getestet.

7.1 Echtzeitfähig

Das Framework benötigt mit vier Teilbereichen für die SURF-Berechnung auf dem Testsystem im Schnitt 30-45ms zur Berechnung und Darstellung eines Framess somit konnten zwischen 22-33 FPS erreicht werden. Damit wurden die geforderten 24 FPS im Durchschnitt erreicht. Sollten die geforderten FPS auf anderen Systemen nicht erreicht werden, kann das Framework leicht beschleunigt werden, indem mehr Teilbereiche eingestellt werden.

7.2 Verdeckungen

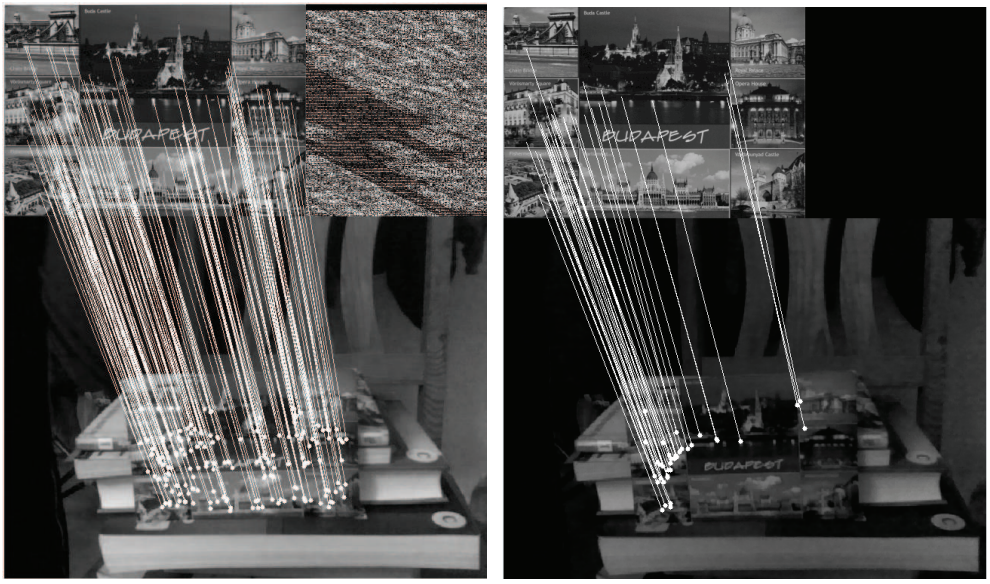
Im laufenden Betrieb kommt es oft vor, dass das zu trackende Objekt nicht ganz im Bild ist oder Teile des Objektes verdeckt werden. In Abbildung 7.1 ist z. B. zu sehen dass die Postkarte, die als Marker dient, nur zur Hälfte zu sehen ist.

Abbildung 7.1: Teilweise verdecktes Objekt.

In der Praxis hat sich gezeigt, dass keine Verschlechterung des Trackings bei Verdeckungen auftritt, vorausgesetzt, es sind ausreichend Punkte zum Tracken vorhanden. Diese ist darauf zurückzuführen, dass das Tracking unabhängig vom Objekt ist. Zum Tracken werden nur einzelne Punkte auf dem Objekt genutzt, wobei es nicht relevant ist, welche Punkte das sind und darum ist es auch nicht relevant, wenn ein Teil der Punkte nicht vorhanden ist, da andere Punkte genutzt werden können.

7.3 Beleuchtungsveränderungen

Normale Veränderungen der Beleuchtung führen nicht zu Problemen und das Objekt kann weiter getrackt werden. Problematisch wird es allerdings, wenn zu wenig Licht vorhanden ist. Das wird in Abbildung 7.2 gezeigt. Im Abbildung 7.2.a befindet sich die Postkarte in einem hell erleuchteten Zimmer. Nach dem Abstellen der Deckenleuchte wird es merklich dunkler und die Anzahl der gefunden Punkte nimmt stark ab. Im normalen Betrieb sollte diese nicht zu Problemen führen. Soll in sehr dunklen Umgebungen gearbeitet werden, könnte eventuell durch Aufhellen des Bildes ein besseres Ergebnis erzeugt werden. Dies müsste aber in der Praxis getestet werden.



(a) Heller Raum (b) Dunkler Raum

Abbildung 7.2: Beleuchtungsveränderungen

7.4 Schnelle Bewegungen

Bei sehr schnellen Bewegungen verschwimmt das Bild (siehe 7.3). Ist dies der Fall, ist es nicht möglich, die Position zu bestimmen. Sobald aber wieder ein klares Bild vorhanden ist, wird das Objekt sofort wieder geortet und die Position kann bestimmt werden. Dem Verschwimmen kann mit guten Kameras entgegengewirkt werden, die auch bei schnellen Bewegungen klare Bilder

erzeugen. Ist diese nicht möglich, stellt z.B. [PLW09] ein Konzept vor mit dem solche Probleme gelöst werden (dies wurde in dieser Arbeit aus Zeitgründen nicht weiter beachtet).

(a) Verschwommendes Bild aufgrund von sehr schneller Bewegung.

(b) Sobald die Bewegung beendet ist, kann die Position wieder korrekt bestimmt werden.

Abbildung 7.3: Unscharfes Bild.

7.5 Genauigkeit

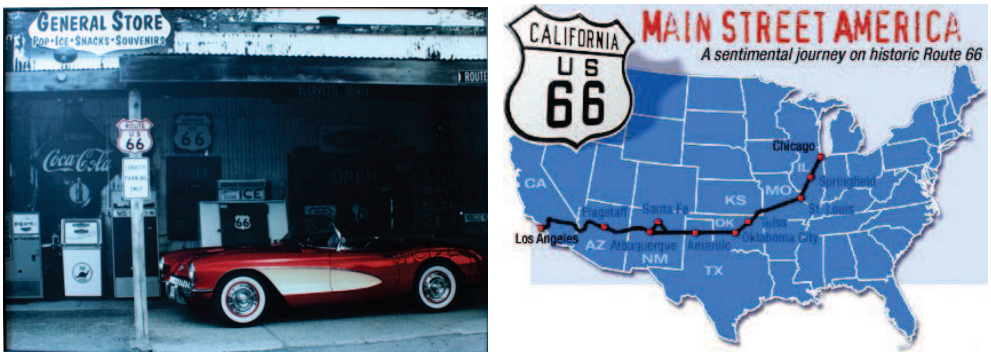
In Abschnitt 6.9.2 wurden zum Thema Genauigkeit bereits Tests durchgeführt, die an dieser Stelle nochmal erwähnt werden sollen. Bei den Tests wurde gezeigt, dass mit dem verwendeten Algorithmus zur Positionsbestimmung selbst bei Verschiebung der Featurepunkte um 10 Pixel in x,y und z Richtung eine hohe Genauigkeit erreicht wird (Gesamtverschiebung, also $x + y + z$ war im Schnitt 7 Pixel). Um die Genauigkeit des Gesamtsystems nun tatsächlich beurteilen zu können, muss festgestellt werden, wie großer der Fehler bei der Bestimmung der Featurepunkte ist.

Das Framework beurteilt einen Featurepunkt als falsch zugeordnet, wenn der quadratische Abstand zum Modell größer ist als 10. Das bedeutet wenn gilt $((\Delta x)^2 + (\Delta y)^2) > 10$, ist ein Featurepunkt falsch und wird nicht zur Berechnung der Position genutzt. Wenn dieses Wissen über die maximale Abweichung nun mit den Tests aus Abschnitt 6.9.2 verglichen wird, lässt sich daraus schließen, dass eine maximale Verschiebung in x, y und z Richtung von 3 Pixeln auftreten kann. Dies kann bei 640×480-Bildern als sehr gering beurteilt werden (eine Abweichung von ca. 0.5 %) und ist in der Praxis in der Regel nicht zu bemerken.

7.6 Entwicklerfreundlichkeit

In diesem Abschnitt soll die Entwicklerfreundlichkeit getestet werden, hierfür wird eine Demoanwendung entwickelt. Als Demoanwendung wurde ein Szenario gewählt, das an das in Abschnitt 1.2 vorgestellt Motivationsszenario angelehnt ist. Dabei ging es darum, Besucher eines Museums mit zusätzlichen Informationen zu den ausgestellten Stücken zu versorgen. Hierzu soll ein

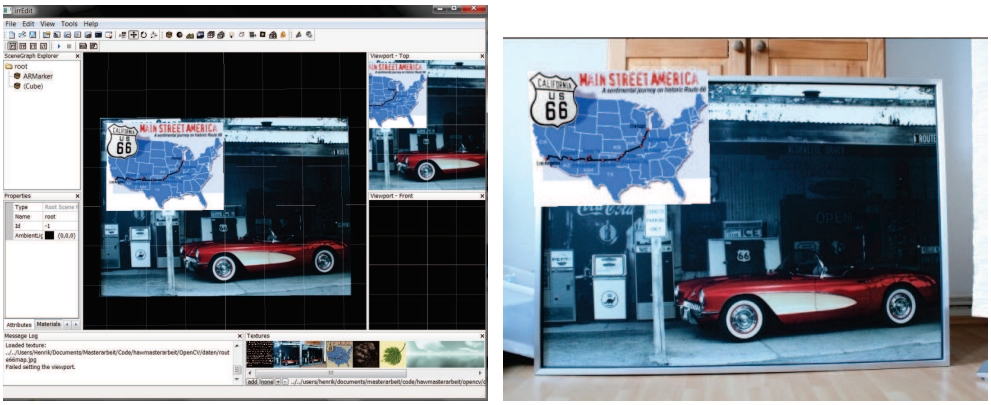
Bild, das eine ältere Corvette an einer Tankstelle auf der Route 66 zeigt (siehe 7.4.a), um zusätzliche Informationen erweitert werden. Hierfür wurde ein Karte der Route 66 gewählt (siehe 7.4.b), die in der linken oberen Ecke des Bildes eingeblendet wird. Die Karte informiert den Nutzer über den Verlauf der Route 66.



(a) Bild von Corvett (b) Karte der Route 66

Abbildung 7.4: Corvette und Route 66.

Mithilfe von IrrEdit wurde eine Irrlich Szene-Datei erstellt, die direkt vom Framework geladen werden kann. Dafür wurde im ersten Schritt ein Rechteck erzeugt, dem die dicke 0 zugewiesen wurde und das als Texture das Bild der Corvette erhalten hat. Dieses Rechteck wurde mit “ARMarker“ bezeichnet. Im zweiten Schritt wurde ein weiteres Rechteck erzeugt, auf dem als Texture die Karte der Route 66 gewählt wurde. Dieses Rechteck wurde entsprechend platziert. Die Irrlicht Szene-Datei kann jetzt in Framework eingelesen werden und funktioniert als vollwertige Anwendung.



(a) IrrEdit (b) Augmented Reality Anwendung

Abbildung 7.5

Bewertung

Mit IrrEdit können einfach Anwendung sehr einfach und komfortable erstellt werden. Werden die Anwendungen allerdings komplexer sodass Nutzer mit den Objekten interagieren können -

denkbar wäre hier z.B., dass der Nutzer auf einen Knopf drückt und weitere Informationen angezeigt bekommt - ist das nicht möglich. Sobald mehr gefordert wird als einfaches Anzeigen, ist der Entwickler gezwungen, dies selbst zu implementieren. Für die Zukunft könnten zusätzliche Tools erstellt werden, mit den auch Anwendungen erstellt werden können, die komplexer sind.

7.7 Weiterentwicklung für die Zukunft

Um das Framework zu komplettieren, müsste als Erstes die fehlende 3D-Unterstützung umgesetzt werden. Diese beinhaltet im Wesentlichen die Umsetzung des in Abschnitt 4.8 vorgestellten Verfahrens in C++ und den Umbau des Frameworks von 2D-Punkten zu 3D-Punkten.

Daraufhin könnte ein Editor entwickelt werden, der auch mit 3D-Welten umgehen kann sodass 3D-Objekte direkt bearbeitet werden können. Hierfür könnte z.B. das Verfahren genutzt werden, das Qi Pan in [PRD09] entwickelt hat. Diese Verfahren ermöglicht es, aus Aufnahmen von Objekten der realen Welt direkt 3D-Polygonnetze zu erzeugen. Diese könnten dann direkt in den Editor eingelesen und zur Erstellung einer 3D-Welt genutzt werden.

Eine weitere zukünftige Weiterentwicklung könnte die Portierung des Frameworks auf tragbare Endgeräten wie Smartphones sein. Hierfür könnte das in [WRM+08] entwickelte PhonySIFT-Verfahren den SURF-Algorithmus ersetzten. Dadurch würde vermutliche eine deutliche höhere Performance erreicht sodass das Framework auch fließend läuft.

Dies würde die Möglichkeit eröffnen, tatsächlich Anwendungen für “Jedermann“ zu entwickeln, da leistungsfähige Smartphones schon heute sehr verbreitet sind und in Zukunft ihre Anzahl vermutlich noch stark zunehmen wird. So könnte ein Museum eine Anwendung zum Download anbieten, die dann jeder interessierte Nutzer auf sein Handy lädt und direkt nutzt.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Framework entwickelt, mit dem Augmented Reality-Anwendungen entwickelt werden können. Hierfür wurde zunächst ein Überblick über den momentanen Stand der Technik gegeben. Danach wurden die nötigen Grundlagen erarbeitet, die sich einmal in die Objekterkennung und Objektverfolgung und in die Positionsbestimmung aufteilen. Bei der Objekterkennung und Objektverfolgung wurden verschiedene Verfahren wie SURF und KLM untersucht, um festzustellen welches Verfahren am besten geeignet ist, um die Lage im Bild von bekannten Punkten zu bestimmen. Im Abschnitt Positionsbestimmung wurden dann verschiede Grundlagen erläutert, mit denen aus den bekannten Punkten die Position ermittelt werden kann.

Im nächsten Schritt wurde dann auf Basis des Wissens, das in den vorangegangenen Abschnitten erarbeite wurde, ein Lösungsansatz entwickelt. Dieser beinhaltete nicht nur die verschiedenen Schritte, die zur Positionsbestimmung nötig sind, sondern noch zusätzliche Elemente wie die Entwicklung eines Editors. Außerdem wurde Anforderungen aufgestellt, die das Framework erfüllen sollte.

Im nächsten Abschnitt wurde erläutert, wie das Framework umgesetzt wurde. Als erstes wurde der Gesamtaufbau des Frameworks erläutert und dann wurden die einzelnen Teilbereiche ausführlich beschrieben. Hierbei wurden teilweise Tests durchgeführt, die verschiedene Techniken verglichen haben, wobei die beste für das Projekt ausgewählt wurde.

Im letzten Abschnitt wurde dann eine Evaluierung des Gesamtergebnisses durchgeführt. Hierfür wurden die im Abschnitt Lösungsansatz aufgestellten Anforderungen mit dem tatsächlichen Ergebnis verglichen. Dabei hat sich gezeigt, dass das Framework im Großen und Ganzen die geforderten Anforderungen erfüllt.

Ausblick

Diese Arbeit hat gezeigt dass es möglich ist nur auf Basis von 3D Modellen und bildbasierten Algorithmen die Position im Raum zu bestimmen. Es ist somit möglich Augmented Reality-Anwendungen nur mithilfe einer Kamera und eines leistungsfähigen Computers zu erstellen. Durch die ständige Leistungssteigerung wird es in der Zukunft möglich sein, dies auch auf kleinen mobilen Endgeräten zu nutzen. Das bietet für die Zukunft die Möglichkeit Augmented Reality mit Systemen wie Google Street View zu verbinden.

Google bietet in Street View Bilder von Straßenzügen an. In der Zukunft dürfte es möglich sein diese Aufnahmen zu kartographieren und ein großes Modell der Welt bzw. von bestimmten Städten zu erstellen. Damit könnte ein System, wie das hier in der Arbeit vorgestellte, die Position des Nutzers bestimmen und entsprechend um virtuelle Elemente erweitern.

Diese virtuellen Objekte könnten z.B. von Google Earth stammen. Google Earth bietet die Möglichkeit 3D-Modelle von Gebäuden auf einem virtuellen Globus zu platzieren. Dafür gibt es schon mehrere Modelle, z.B. Wiederauferstehung des antiken Roms in 3D [Rom10] . In der Zukunft sollte es möglich sein genau aufgrund dieser Modelle und der vorher erwähnten Aufnahmen aus Street View eine Augmented Reality-Anwendung zu erstellen, die dem Nutzer die Möglichkeit gibt durch Rom zu laufen und zu sehen wie es vor 2000 Jahren dort aussah.

Abbildung 8.1: Kolosseum in Rom. Auf dem Tablett-PC wird gezeigt wie das Kolosseum vor 2000 Jahren aussah.

Literaturverzeichnis

[ART09a] *ARTag*. Webseite, Stand: 22. Februar 2009. – www.artag.net/

[ART09b] *ARToolKit*. Webseite, Stand: 22. Februar 2009. – www.hitl.washington.edu/artoolkit/

[ART09c] *ARToolKit+*. Webseite, Stand: 22.12.2009. – http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php

[ASTM08] AZZARI, Pietro ; STEFANO, Luigi D. ; TOMBARI, Federico ; MATTOCCIA, Stefano: *Markerless Augmented Reality using Image Mosaics*. Paper, 2008

[ATA09] *ARTag*. Webseite, Stand: 22.2.2009. – http://www.artag.net/

[Bau10] *Kd-Bäume*. Webseite, Stand: 16. Februar 2010. – http://www.lehre.informatik.uniosnabrueck.de/dbs/2001/skript/node34.html

[BBG02] BRÜDERLIN, Prof. Dr. B. ; BEIER, Dipl.-Inf. D. ; GASSMANN, Dipl.-Ing. F.: *Modellbasierte Objekterkennung in einem AR-System*. Paper, 2002

[Bic03a] BICHLER, Simon: *ARToolkit*. Webseite, 1. Dezember 2003. – http://wwwnavab.in.tum.de/twiki/pub/Far/AugmentedRealityBenutzerschnittstellenWiSe2003/hauptseminar-artk.pdf

[Bic03b] BICHLER, Simon: *ARToolkit*. Paper, 2003

[BK08] BRADSKI, Gary ; KAEHLER, Adrian: *Learning OpenCV: Computer Vision with the OpenCV Library*. Cambridge, MA : O'Reilly, 2008

[BTGL06] BAY, Herbert ; TUYTELAARS, Tinne ; GOOL, Van ; L.: *SURF: Speeded Up Robust Features*. Graz Austria, May 2006

[CBL08] CHOI, Changhyun ; BAEK, Seung-Min ; LEE, Sukhan: *Real-time 3D Object Pose Estimation and Tracking for Natural Landmark Based Visual Servo*. 2008

[CKM08] CASTLE, R. O. ; KLEIN, G. ; MURRAY, D. W.: Video-rate Localization in Multiple Maps for Wearable Augmented Reality. In: *Proc 12th IEEE Int Symp on Wearable Computers, Pittsburgh PA, Sept 28 - Oct 1, 2008*, 2008, S. 15–22

[CMC03] COMPORT, Andrew I. ; MARCHAND Éric ; CHAUMETTE, François: *A real-time tracker for markerless augmented reality*. Paper, 2003

[CMPC06] COMPORT, Andrew I. ; MARCHAND, Eric ; PRESSIGOUT, Muriel ; CHAUMETTE, Francois: *Real-Time Markerless Tracking for Augmented Reality: The Virtual Visual Servoing Framework*. Paper, 2006

[Esc06] ESCHENBURG, Jonas: *Optisches Kameratracking anhand natürlicher Merkmale*. 2006

[Ewe06] EWERING, Dag: *Modellbasiertes Tracking mittels Linien- und Punktkorrelationen*. Diplomarbeit, 2006

[FB81] FISCHLER, Martin A. ; BOLLES, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Commun. ACM* 24 (1981), June, Nr. 6, 381–395. http://dx.doi.org/10.1145/358669.358692. – DOI 10.1145/358669.358692. – ISSN 0001–0782

[Fär06] FÄRBER, Markus: *Markerbasiertes Tracking für Augmented Reality Applikationen*. Paper, 2006

[FTG01] FERRARI1, V. ; TUYTELAARS, T. ; GOOL, L. V.: *Markerless Augmented Reality with a Real-time Affine Region Tracker*. Paper, 2001

[GRS+02] GENC, Y. ; RIEDEL, S. ; SOUVANNAVONG, F. ; AKINLAR, C. ; NAVAB, N.: *Marker-less Tracking for AR: A Learning-Based Approach*. Washington, DC, USA, 2002

[Hab09] HABELITZ, Thorsten: *Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis der Ähnlichkeitsbestimmung photorealistischer Bilder*. 2009

[Har92] HARRIS, C.:: *Tracking with Rigid Objects*. 1992

[Hey05] HEYMANN, Sebastian: *Implementierung und Evaluierung von Video Feature Tracking auf moderner Grafikhardware*. Diplomarbeit, 2005

[HS80] HORN, Berthold K. ; SCHUNCK, Brian G.: Determining Optical Flow. Cambridge, MA, USA : Massachusetts Institute of Technology, 1980. – Forschungsbericht

[HS88] HARRIS, C. ; STEPHENS, M.: A Combined Corner and Edge Detection. In: *Proceedings of The Fourth Alvey Vision Conference*, 1988, 147–151

[HZ04] HARTLEY, R. I. ; ZISSERMAN, A.: *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004

[IRR08] *IrrAR*. Webseite, Stand: 18. Oktober 2008. – http://sourceforge.net/projects/irrar/

[IRR09] *Irrlicht 3D*. Webseite, Stand: 22. Februar 2009. – http://irrlicht.sourceforge.net/

[JJB+05] JUAN, M.C. ; JOELE, D. ; BAÑOS, R. ; BOTELLA, C. ; MAST, M. Alcañiz C. d.: *A Markerless Augmented Reality System for the treatment of phobia to small animals*. Paper, 2005

[Jot01] JOTZO, Joachim: *Aktive Landmarken zur Positionsbestimmung von autonomen Fahrzeugen*. Dissertation, 2001

[Jun06] JUNG, Frank: *Objekterkennung mit SIFT-Features*. Bachelorarbeit, 2006

[KB99] KATO, Hirokazu ; BILLINGHURST, Mark: *Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System*. Paper, 1999. – http://portal.acm.org/citation.cfm?id=858134

[KIN03] KIND, ANDREAS: *Positionsbestimmung Seminar Mobile Systeme, SS03*. Paper, 2003

[Kle06] KLEIN, Georg: *Visual Tracking for Augmented Reality*. Dissertation, 2006

[Kle08] KLEIN, Georg: *Parallel Tracking and Mapping for Small AR Workspaces*. Paper, 2008

[LF05] LEPETIT, Vincent ; FUA, Pascal: *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*. Paper, 2005

[LH08] LEE, Taehee ; HÖLLERER, Tobias: *Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality*. Paper, 2008

[LK81a] LUCAS, Bruce D. ; KANADE, Takeo: *An Iterative Image Registration Technique with an Application to Stereo Vision*. Paper, 1981

[LK81b] LUCAS, Bruce D. ; KANADE, Takeo: An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, 1981, S. 674–679

[Low04] LOWE, David G.: *Distinctive image features from scale-invariant keypoints*. Paper, 2004. – http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8899

[LPF04] LEPETIT, Vincent ; PILET, Julien ; FUA, Pascal: *Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation.* 2004

[Mar04a] MARCHAND, Muriel Pressigout´and E.: *Model-free augmented reality by virtual visual servoing*. Paper, 2004

[Mar04b] MARCZOK, Hagen: *Multiple-View Geometry - Studienarbeit*. Universität Rostock, Fachbereich Informatik, 2004

[ML09] MUJA, Marius ; LOWE, David G.: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In: *VISSAPP (1)*, 2009, S. 331–340

[MYN07] MOOSER, Jonathan ; YOU, Suya ; NEUMANN, Ulrich: *Real-Time Object Tracking for Augmented Reality Combining Graph Cuts and Optical Flow*. Paper, 2007

[PLW09] PARK, Youngmin ; LEPETIT, Vincent ; WOO, Woontack: *ESM-Blur: Handling and Rendering Blur in 3D Tracking and Augmentation*. 2009

[PRD09] PAN, Q. ; REITMAYR, G. ; DRUMMOND, T.: ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In: *Proc. 20th British Machine Vision Conference (BMVC)*. London, September 2009

[PTA09] *PTAM*. Webseite, Stand: 22.9.2009. – http://www.robots.ox.ac.uk/~gk/PTAM/

[RBG94] ROLLAND, Jannick P. ; BAILLOT, Yohan ; GOON, Alexei A.: *A SURVEY OF TRACKING TECHNOLOGY FOR VIRTUAL ENVIRONMENTS*. 1994

[RD06a] REITMAYR, Gerhard ; DRUMMOND, Tom W.: *Robust Model-based Tracking for Outdoor Augmented Reality*. Paper, 2006

[RD06b] ROSTEN, Edward ; DRUMMOND, Tom: Machine learning for high-speed corner detection. In: *In European Conference on Computer Vision* Bd. 1, 2006, 430–443

[Rom10] *Das alte Rom in 3D. Webseite,*. Webseite, Stand: 20. Februar 2010. – http://earth.google.de/rome/

[SAH08] SILPA-ANAN, C. ; HARTLEY, R.: *Optimised KD-trees for fast image descriptor matching*. 2008

[SB02] SIMON, Gilles ; BERGER, Marie-Odile: *Reconstructing while registering: a novel approach for markerless augmented reality*. Paper, 2002

[Sch05] SCHREER, Oliver: *Stereoanalyse und Bildsynthese*. Springer, Berlin, 2005

[Sch06a] SCHNEIDER, Matthias: *Markerloses Posentracking für Augmented Reality Anwendungen*. Paper, 2006

[Sch06b] SCHWEIGHOFER, Gerald: Robust Pose Estimation from a Planar Target. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (2006), December, Nr. 12, 2024–2030. http://dx.doi.org/10.1109/TPAMI.2006.252. – ISSN 0162–8828

[SFZ00a] SIMON, Gilles ; FITZGIBBON, Andrew W. ; ZISSERMAN, Andrew: *Markerless Tracking using Planar Structures in the Scene.* 2000

[SFZ00b] SIMON, Gilles ; FITZGIBBON, Andrew W. ; ZISSERMAN, Andrew: *Markerless tracking using planar structures in the scene*. Munich, Germany, October May–June 2000

[STT09] *Studierstube Tracker*. Webseite, Stand: 22.2.2009. – http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbtracker.php

[Sut03] SUTHAU, Tim: *Augmented Reality Techniken für den Einsatz in der Leberchirurgie*. Paper, 2003

[Teg06] TEGTMEIER, André: *Augmented Reality als Anwendungstechnologie in der Automobilindustrie*. Dissertation, 2006. – http://diglib.uni-magdeburg.de/Dissertationen/2007/andtegtmeier.pdf

[TTV05] *Tracking Technologies for Virtual Environments*. Paper, 2005. – http://campar.in.tum.de/twiki/pub/Chair/TeachingSS05ARProseminar/Tracking_A.pdf

[VLF04a] VACCHETTI, L. ; LEPETIT, V. ; FUA, P.: *Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking.* 2004

[VLF04b] VACCHETTI, L. ; LEPETIT, V. ; FUA, P.: *Stable Real-Time 3D Tracking using Online and Offline Information.* 2004

[Wag06] WAGNER, Thomas: *Qualitative sicht-basierte Navigation in unstrukturierten Umgebungen*. Dissertation, 2006

[WEB09] *Charlotte: A Simple Web Server for Microsoft Windows*. Webseite, Stand: 22. Februar 2009. – http://www.acm.org/crossroads/xrds6-4/charlotte.html

[Wei06] WEIDENHAUSEN, Jens-Martin: *Mobile Mixed Reality Platform*. Dissertation, 2006

[WLS08] WAGNER, Daniel ; LANGLOTZ, Tobias ; SCHMALSTIEG, Dieter: *Robust and Unobtrusive Marker Tracking on Mobile Phones*. Paper, 2008

[WRM+08] WAGNER, Daniel ; REITMAYR, Gerhard ; MULLONI, Alessandro ; DRUMMOND, Tom ; SCHMALSTIEG, Dieter: *Pose Tracking from Natural Features on Mobile Phones*. Paper, 2008

[WS07] WAGNER, Daniel ; SCHMALSTIEG, Dieter: *ARToolKitPlus for Pose Tracking on Mobile Devices*. 2007

[Yua06a] YUAN, Chunrong: *Markerless Pose Tracking for Augmented Reality*. Paper, 2006

[Yua06b] YUAN, Chunrong: *A tracking by detection approach for robut markerless tracking*. Paper, 2006

[Zha95] ZHANG, Zhengyou: *Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting*. 1995

[Zha99] ZHANG, Zhengyou: Flexible Camera Calibration by Viewing a Plane from Unknown Orientations. In: *Computer Vision, IEEE International Conference on* 1 (1999), S. 666. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/ICCV.1999.791289. – DOI http://doi.ieeecomputersociety.org/10.1109/ICCV.1999.791289. ISBN 0–7695–0164–8

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw. §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25. Februar 2010

Ort, Datum

Unterschrift