Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Prozessrechentechnik,
Automation und Robotik

Entwicklung eines Kopfverfolgungssystems auf der Basis einer Panoramakamera und künstlicher Landmarken

Diplomarbeit
von

Tim Weyrich

1. Januar 2001 – 30. Juni 2001

Referent:	Prof. Dr.-Ing. R. Dillmann
Koreferent:	Prof. Dr.-Ing. H. Wörn
Betreuer:	Dipl.-Inform. B. Giesler
	Dipl.-Inform. T. Salb

Ich versichere hiermit, die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 30. Juni 2001

Tim Weyrich

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einführung

Am Lehrstuhl für industrielle Anwendungen der Informatik und Mikrosystemtechnik (IAIM) werden zur Zeit Anwendungsgebiete bearbeitet, bei denen unter anderem eine *Augmented-Reality*-Brille zum Einsatz kommen soll. Der Begriff Augmented Reality (AR) umfaßt Techniken, bei denen als Mensch/Maschine-Schnittstelle virtuelle Ansichten mit Bildern der Realität überlagert werden. AR soll die reale Umgebung eines Benutzers durch eine computergenerierte Szene sinnvoll ergänzen. Der Gedanke dabei ist, dem Benutzer auf diesem Wege Informationen zukommen zu lassen, die außerhalb seiner Sinneswahrnehmung liegen.

Zu diesem Zweck wurden unter anderem AR-Brillen geschaffen: Brillen, in denen kleine Displays eingebaut sind, deren Bild auf das Auge gelenkt wird und durch die die Umgebung des Brillenträgers weiterhin sichtbar bleibt (Bild rechts).

Abbildung 1.1: Am IAIM eingesetzte AR-Brille.

Im Idealfall sind Einblendung und Realität perfekt miteinander verwoben, so daß es schwerfällt, zwischen realen und virtuellen Gegenständen zu unterscheiden. Einen wichtigen Beitrag zum Realismus der eingeblendeten Objekte hat dabei ihre stationäre Lage im Raum: Aus Sicht des Benutzers muß der überlagerte Gegenstand eine eindeutige Position und Orientierung im Raum haben, die sich nicht verändert, wenn der Benutzer seinen Kopf bewegt oder in der Szene herumläuft.

Beim Einsatz einer AR-Brille kann dies nur erreicht werden, wenn die Lage des Kopfes des Benutzers im Raum genau bekannt ist. Je nach Kopfposition muß die virtuelle Szene an einer anderen Stelle des Blickfelds eingeblendet werden, um eine konsistente Überlagerung mit der Realität aufrechtzuerhalten.[1]

Dieses Problem der Kopfverfolgung trat auch schon bei „klassischen“ kopfgebundenen Displays auf, wie sie in vielen *Virtual-Reality*-(VR-)Anwendungen eingesetzt werden: Auch hier wird dem Benutzer eine virtuelle Szene ins Blickfeld projiziert, nur mit dem Unterschied, daß die reale Umgebung vollständig ausgeblendet wird. Aus diesem Umfeld stammen denn auch die meisten der kommerziell erhältlichen Kopfverfolgungssysteme (*Head-Tracker*). Da bei VR-Brillen aber keine Überlagerung mit der Umwelt stattfindet, sind die Genauigkeitsanforderungen an diese Systeme nicht sehr hoch: Kleinere Winkelabweichungen werden vom Benutzer kaum wahrgenommen, solange das Bild nur schnell genug nachgeführt wird; die nötige Genauigkeit wird durch den Gleichgewichtssinn des Benutzers vorgegeben, und dieser

[1] Die Stellung der Augen kann weitestgehend vernachlässigt werden, solange die Projektionsebene der Einblendungen nicht zu nahe an der Pupille liegt.

ist maßgeblich durch den Sehsinn beeinflußt. Bei der translatorischen Auflösung bestehen sogar noch größere Toleranzen.

In dem Moment aber, in dem sich der Benutzer gleichzeitig an der Umgebung orientieren kann, fallen Fehler in der Kopfverfolgung schwerer ins Gewicht. In vielen AR-Anwendungen will man mit Einblendungen reale Objekte markieren, was in noch höherem Maße voraussetzt, daß die Markierungen sich nicht relativ zu den Objekten verschieben.

Aus diesem Grunde bringen viele erhältlichen Kopfverfolgungssysteme keine ausreichende Genauigkeit für den Einsatz mit einer AR-Brille mehr mit. Andere, hochgenaue Verfolger sind wieder zu teuer in der Anschaffung und/oder erfordern einen erheblichen Aufwand beim Aufbau des Systems.

1.1 Aufgabe der Arbeit

Daher wurde für diese Diplomarbeit die Aufgabe gestellt, ein alternatives Kopfverfolgungssystem zu entwickeln, das bei verhältnismäßig geringem Hardware-Aufwand eine hohe Auflösung mitbringt. Durch die geplanten Einsatzgebiete waren weitere Randbedingungen gegeben:

Geplante Einsatzgebiete

Am IAIM werden im Moment zwei Szenarien untersucht, bei denen eine AR-Brille Anwendung finden kann.

Computerunterstütztes Operieren: Einem Chirurgen könnten Einblendungen während einer Operation nützliche Dienste leisten. So ließe sich zum Beispiel ein „Röntgenblick“ simulieren, indem dem Patienten CT-Daten überlagert werden. Eine andere Anwendung könnte sein, den Arzt während des Eingriffs über Risikozonen im Gewebe zu informieren [4, 5]. In jedem Fall könnten größere Fehler bei der Kopfverfolgung hier fatale Folgen haben und sind daher unbedingt zu vermeiden.

Programmierung mobiler Roboter: Eine andere Anwendung ist die Mensch/Maschine-Schnittstelle bei der Programmierung mobiler Roboter. Die Einblendungen können als Zustandsanzeigen dienen oder zum Beispiel geplante Trajektorien räumlich darstellen, bevor der Roboter selbst losgeschickt wird.

In diesem Szenario kommt es wiederum weniger auf hohe Präzision als auf Flexibilität im Einsatzort an. Es muß möglich sein, Roboter an den unterschiedlichsten Orten zu programmieren, weshalb keine zu großen Einschränkungen an die Umgebung existieren dürfen und das System schnell aufgebaut sein muß.

Darüberhinaus sind natürlich noch weitere mögliche AR-Anwendungen denkbar. Die Entwurfsentscheidungen dieser Arbeit wurden aber in Hinblick auf diese beiden Fälle getroffen.

1.2 Lösungsansatz

Für die Lösung des Problems habe ich folgenden Ansatz gewählt:

Kern des Kopfverfolgers ist eine sogenannte *Panoramakamera*. Das ist eine Kamera, die sich insbesondere durch den großen Raumwinkel auszeichnet, den ihr Bild abdeckt. Die Kamera wird auf dem Kopf des Benutzers befestigt, so daß ihr Bild etwa die Hemisphäre oberhalb des Kopfes erfaßt.

Zusätzlich werden im näheren Umfeld *künstliche Landmarken* plaziert. Die Landmarken sind Objekte, die sich leicht im Kamerabild detektieren lassen. Ein Rechner verarbeitet die Kamerabilder und ermittelt daraus Peilungen zu den Landmarken.

Bewegt nun der Benutzer den Kopf, bewegt sich die Kamera mit. Damit verändern sich auch diese Peilungen.

Ziel der Diplomarbeit war es nun, aus diesen veränderlichen Peilungen auf die Lage der Kamera im Raum zu schließen und damit auch Position und Orientierung des Benutzerkopfes zu ermitteln.

Dabei sind prinzipiell zwei Ansätze denkbar:

Bekannte Landmarkenpositionen: Im ersten Fall wird jede Landmarke zu Beginn vermessen, so daß ihre Position im Raum genau bekannt ist. Der Kopfverfolger muß dann eine Lage der Kamera im Raum finden, bei der die gemessenen Peilungen genau durch die Markenpositionen gehen. Diese Lage definiert dann die gesuchte Kameraposition und -orientierung.

Das Vermessen der Landmarken kostet Zeit, weshalb dieser Ansatz die Eignung für den mobilen Einsatz etwas einschränkt.

Unbekannte Landmarkenpositionen: Ungleich schwieriger gestaltet sich die Aufgabe, die Kameraposition bei unbekannten Landmarkenpositionen zu bestimmen. Es reicht dann nicht mehr aus, ein einzelnes Bild zu betrachten, um auf die Lage der Kamera zu schließen. Statt dessen muß die von der Kamera gelieferte Bildfolge über die Zeit betrachtet werden. Aufgabe des Kopfverfolgers ist es dann, eine Trajektorie zu finden, auf der sich die Kamera bewegt haben könnte, so daß die aufgenommenen Bilder „zusammenpassen“.

Dieses Problem läßt sich im allgemeinen bis auf einen Skalierungsfaktor des Bezugskoordinatensystems lösen. Es ist eindeutig lösbar, wenn von zwei Landmarken die Entfernung zueinander bekannt ist.

Da bei dieser Variante das Vermessen der Landmarken entfällt, kann das System sehr schnell aufgebaut werden. Leider wird die mit ihm erreichbare Genauigkeit aber unter der des Verfahrens mit bekannten Landmarkenpositionen liegen.

1.3 Umsetzung im Rahmen der Diplomarbeit

Es wurde zunächst nur die Variante bekannter Landmarkenpositionen bearbeitet, um innerhalb der zur Verfügung stehenden Zeit zu einem Ergebnis zu gelangen. Zum Fall unbekannter Markenpositionen wurden lediglich theoretische Überlegungen angestellt.

Für die Arbeit stand anfangs eine Panoramakamera zur Verfügung, die hochauflösende Einzelbilder liefern konnte, aber aufgrund ihrer Bauhöhe für die Befestigung am Kopf ungeeignet war. Mit Hilfe dieser Kamera wurde ein Kalibrierungsverfahren für Panoramakameras entwickelt und untersucht. Eine Kamera kleinerer Bauform wurde bestellt.

Um dennoch schon mit der Arbeit am Kopfverfolger beginnen zu können, wurde eine Kamerasimulation entwickelt, die virtuelle Bilder einer Panoramakamera berechnet. Da bis zum Ende der Arbeiten die bestellte Kamera noch nicht eingetroffen war, konnte der fertige Kopfverfolger nur innerhalb dieser Simulation getestet werden.

Die Kamera wurde schließlich während der Erstellung dieser Ausarbeitung geliefert. Die in Kapitel 6 vorgestellten Simulationsergebnisse konnten daher bereits die geometrischen Daten der neuen Kamera verwenden.

1.4 Aufbau der Ausarbeitung

Kapitel 2 informiert über aktuelle Kopfverfolgungssysteme und Arbeiten, deren Thematik mit dieser Arbeit verwandt ist. Die wichtigsten mathematischen und technischen Grundlagen dieser Arbeit vermittelt Kapitel 3.

In Kapitel 4 wird der allgemeine Entwurf des geplanten Kopfverfolgungssystems vorgestellt, während sich Kapitel 5 der konkreten Realisierung widmet. Anschließend werden in Kapitel 6 und 7 die Ergebnisse der Arbeit vorgestellt und diskutiert.

Kapitel 8 liefert schließlich eine Zusammenfassung und gibt einen Ausblick auf mögliche Weiterentwicklungen des Systems.

Da im Rahmen der Diplomarbeit sehr unterschiedliche Implementierungsaufgaben anfielen, deren ausführliche Beschreibung die eigentliche Ausarbeitung schnell unübersichtlich gemacht hätte, wurden deren Details in Anhänge ausgelagert. Auf die einzelnen Anhänge wird an entsprechender Stelle verwiesen.

Kapitel 2

Stand der Forschung

2.1 Kopfverfolgungssysteme

Bestehende Kopfverfolgungssysteme können danach unterschieden werden, ob die Sensorik stationär im Raum befestigt ist (*periphäre Sensorik*) oder ob sie am Kopf getragen wird (*zentrale Sensorik*):

2.1.1 Periphäre Sensorik

Alle uns bekannten Vertreter der ersten Klasse arbeiten optisch und analysieren mit mehreren Kameras die Bewegung von künstlichen Landmarken, die am Kopf des Benutzers befestigt sind.

Bei eingeschränktem Spielraum der Kopfposition (wenn der Benutzer zum Beispiel vor einem Bildschirm sitzt) genügen passive Marken[1]. In manchen Fällen reicht es sogar aus, Gesichtsmerkmale wie Augen und Nase mit Methoden der Mustererkennung im Bild zu lokalisieren, um darüber auf die Kopfposition zu schließen. Systeme dieser Art sind für unsere Zwecke ungeeignet, da die zu erwartende Bewegung des Benutzer zu große Freiheitsgrade besitzt.

Soll sich der Benutzer frei im Raum bewegen können, greift man gerne zu aktiven Marken in Form von Leuchtdioden oder passiven Retro-targets, die sich leicht am Kopf befestigen lassen. Einige am Markt erhältliche Systeme setzen zwei bis vier Kameras ein, um die Leuchtdioden im Raum zu lokalisieren. Leider haben diese Systeme zwar eine recht hohe translatorische Auflösung, prinzipbedingt ist die rotatorische Auflösung aber recht schlecht, will man nicht auf eine Vielzahl von Kameras zurückgreifen und damit den Hardware-Aufwand in die Höhe treiben.

Nun ist für AR-Anwendungen die rotatorische Auflösung des Kopfverfolgers besonders wichtig: Bei AR-Brillen geht es insbesondere darum, den Bildfehler (*Screen Error*) der Einblendung zu minimieren, das heißt die Soll-Abweichung der eingeblendeten Objekte in der Bildebene (meßbar z.B. in Pixeln) möglichst klein zu halten. Je nach Entfernung zum virtuellen Objekt ist dieser Fehler mehr oder weniger von der translatorischen Genauigkeit des Systems abhängig. Unabhängig von der Objektentfernung haben aber Änderungen in der Orientierung einen sehr starken Einfluß innerhalb der Bildebene.

2.1.2 Zentrale Sensorik

In diesem Sinne sind zentrale Sensoren gegenüber periphär installierten im Vorteil: Da sie sich bei Kopfbewegungen im Rotationszentrum befinden, können sie oft eine viel größere

[1] Zu möglichen Ausführungen künstlicher Landmarken siehe Abschnitt 4.3.1.

rotatorische Genauigkeit erzielen.

Auch vom zentralen Typ sind schon einige Varianten kommerziell erhältlich. Dabei kommen im wesentlichen folgende Varianten zum Einsatz:

Magnetische Sensoren: Ein häufiger Typ Kopfverfolger besteht aus einem stationären Aufbau, der ein elektro-magnetisches Referenzfeld im Raum erzeugt, so daß über Spulen am Kopf des Benutzers dessen relative Lage zum Felderzeuger bestimmt werden kann. Diese Systeme sind besonders kompakt gebaut; außerdem ist es möglich, im gleichen Magnetfeld noch weitere Objekte, wie zum Beispiel einen Datenhandschuh, zu verfolgen.

Der Nachteil magnetischer Kopfverfolger liegt darin, daß das Referenzfeld durch magnetisierbare Gegenstände im Raum beeinflußt wird. Möchte man dann noch eine ausreichende Genauigkeit erzielen, wird ein sehr aufwendiger Kalibrierungsvorgang nötig, der jedesmal wiederholt werden muß, wenn sich die Verteilung ferromagnetischer Materialien im Raum ändert. Dies behindert in unserem Fall den mobilen Einsatz.

Ein recht verbreiteter Vertreter dieses Verfolgertyps ist der Tracker der Firma Polhemus.

Gyroskope/Beschleunigungssensoren: Ein anderer Typ, der vor allem im Low-Cost-Sektor anzutreffen ist, besteht aus einem Kreiselkompaß (*Gyroskop*), der die Orientierungsänderung des Kopfes wahrnimmt. Das Gyroskop wird durch Beschleunigungssensoren ergänzt, die für die Änderung der Position zuständig sind.

Beide Sensormodule sind recht kompakt, haben sehr geringe Latenzzeiten und lassen sich schnell auswerten. Dennoch kommen sie für unseren Fall nicht in Frage, da sie stets nur relative Lageänderungen aufzeichnen können und daher schnell zur Fehlerakkumulation neigen: Je länger das System in Betrieb ist, desto stärker werden die Abweichungen bei der Überlagerung virtueller mit realen Objekten.

Als Ergänzung zu unserem System sind Gyroskope aber denkbar, wie in Abschnitt 7.5 erläutert wird.

Optische Sensoren: Optischen zentralen Sensoren ist unser eigener Ansatz zuzuordnen. Darüberhinaus ist das einzige uns bekannte optisch arbeitende zentrale Verfolgungssystem der HiBall-Tracker, der an der University of North Carolina entwickelt wurde [6].

Der HiBall-Tracker setzt voraus, daß die Decke des Arbeitsraums aus genormten quadratischen Deckenpaneelen besteht, in die jeweils 32 Leuchtdioden eingesetzt sind. Über jedem Deckenpaneel befindet sich eine Steuerelektronik, die mit den anderen vernetzt ist und die Dioden nach einem speziellen zeitlichen Muster pulst, das mit den anderen Paneeleinheiten synchronisiert wird.

Der eigentliche Sensor ist der sogenannte *HiBall*, ein golfball-großes Gerät, das in seinem Inneren sechs Lateraleffektdioden enthält. Lateraleffektdioden sind flächige Photodioden, bei denen sich die Position des Beleuchtungsmaximums auf ihrer Fläche lokalisieren läßt. Den sechs Dioden ist jeweils eine Linse vorgeschaltet, die das Licht der gepulsten Leuchtdioden innerhalb eines gewissen Raumwinkels auf die Diodenfläche lenkt.

Das System liefert 2000 Positions- und Orientierungsschätzungen pro Sekunde. Diese hohe Rate wird dadurch erreicht, daß in einen Schätzungsschritt nicht alle Leuchtidiodenpeilungen eingehen. Lediglich die Dioden, die zum jeweiligen Zeitpunkt gerade aktiv sind, werden berücksichtigt und in einem Kalman-Filter-basierten Verfahren verarbeitet, das die Autoren *Single-Constraint-at-a-Time Tracking (SCAAT)* nennen [7, 8].

Für unsere Anforderungen bringt das System mit einer Genauigkeit von $\pm 0{,}5$ mm in translatorischer und $\pm 0{,}02°$ in rotatorischer Hinsicht die nötige Auflösung mit. Leider verbietet die feste Installation der Deckenpaneelen den mobilen Einsatz. Darüberhinaus bringt der große Hardware-Aufwand für unsere Anwendungen zu hohe Kosten mit sich.

2.2 Systeme mit Panoramakameras

Hin und wieder findet man in der Robotik Ansätze, die auch von einer Panoramakamera gebrauch machen. An bestehenden Übungen kommen der in dieser Arbeit gestellten Aufgabe kameragestützte Navigationssysteme am nächsten. Ich möchte daher einen kurzen Abriß über Panoramakamera-gestützte Navigation liefern, und die dabei verwendeten Kameratypen vorstellen:

2.2.1 Navigation mit 360°-Schnappschüssen

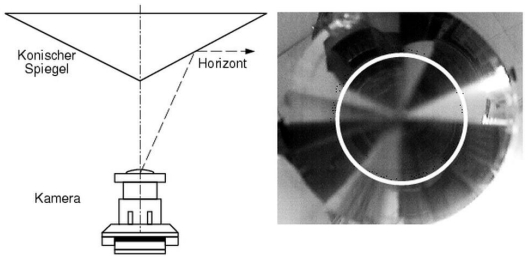

Abbildung 2.1: Prinzip einer konischen Panoramakamera. *Links:* Die Kamera ist auf einen polierten Kegel gerichtet. *Rechts:* Das aufgenommene Bild. Weiß ist der Horizontkreis eingezeichnet.

Am Max-Planck-Institut für Biologische Kybernetik in Tübingen wurde ein Navigationssystem für mobile Roboter entwickelt, das sich anhand von 360°-Aufnahmen der Umwelt orientiert [9, 10].

Für den Navigationsalgorithmus wurden Anleihen am Orientierungssinn von Honigbienen gemacht. Von Bienen vermutet man, daß sie ein gewünschtes Ziel erreichen, indem sie in ihrem Umfeld Orte ansteuern, deren Ansicht sich mit einer früher am Ziel gewonnenen Ansicht (dem *Schnappschuß* aus der Erinnerung) am besten deckt [11].

Die Aufnahmen werden mit einer Panoramakamera gemacht. Die Kamera besteht aus einem handelsüblichen Kameramodul, dessen Optik auf einen polierten Metallkegel gerichtet ist. Dadurch wird ein schmaler Streifen um den Horizont abgebildet (Abbildungen 2.1 und 2.2). Über diesen Streifen wird radial gemittelt, um leichte Verkippungen der Kamera aus der Horizontalen auszugleichen. Letztlich wird die Kamera als eindimensionales 360°-Sensor-Feld verwendet.

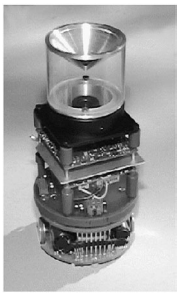

Abbildung 2.2: Konische Panoramakamera.

Für die Navigationsaufgabe muß im Vorfeld bereits eine Reihe von 360°-Bildern gemacht worden sein, die von genau bekannten Positionen aus aufgenommen wurden. Anschließend ist das System in der Lage, einen mit der Panoramakamera ausgestatteten Roboter eine bestimmte Position im Raum anfahren zu lassen. Dazu vergleicht es laufend die Kamerabilder, die der Roboter liefert, mit der Aufnahme, die am Zielort aufgenommen wurde, und bestimmt daraus einen Zielvektor, der den Roboter näher ans Ziel bringt.

Die Arbeit behandelt eine rein zweidimensionale Fragestellung: Der Roboter hat nur drei Freiheitsgrade, und die 360°-Aufnahmen sind auf die horizontale Ebene beschränkt. Das System arbeitet mit natürlichen Landmarken.

Der implementierte Algorithmus eignet sich zur Feinplanung: Der Roboter muß sich innerhalb eines gewissen „Einzugsgebiets“ der Zielaufnahme befinden, um sicher ans Ziel geleitet zu werden. Das System wurde sowohl in einer kleinen Modellwelt als auch in einem Büro getestet. Bei 10 000 Referenzaufnahmen fand der Roboter in den meisten Fällen zum Ziel, solange er nicht zu weit vom Zielort entfernt war.

2.2.2 Verfolgung eines visuellen Pfads mit einer Panoramakamera

Mit [12] liegt eine Arbeit vor, in der ein mobiler Roboter anhand von Panoramabildern seine Position in der Ebene bestimmt. Dabei wird eine Panoramakamera mit sphärischem Spiegel eingesetzt. Das Blickfeld der Kamera deckt den Großteil einer Hemisphäre ab. Abbildung 2.3 zeigt die Abbildungsgeometrie.

Die Kamera wird so auf dem mobilen Roboter befestigt, daß sie den Boden um den Roboter herum abbildet. Nach einer Entzerrung des Bildes liegt eine Aufnahme des Bodens aus der Vogelperspektive vor. Auf dem Boden werden rechteckige Farbflächen als Landmarken aufgeklebt. Eine kantenorientierte Merkmalsextraktion liefert die Umrisse dieser Marken. Bei der in der Arbeit vorgestellten Implementation mußten die zu betrachtenden Landmarken im Bild noch manuell ausgewählt werden.

Während der Roboter sich bewegt, wird die Roboterposition anhand der Markenbilder und ihrer Verschiebung mitgeführt. Die Positionsbestimmung erfolgt auf wenige Zentimeter genau.

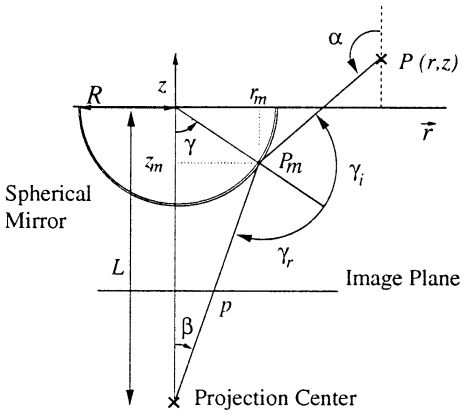

Abbildung 2.3: Strahlengang einer sphärischen Panoramakamera.

Diese Arbeit kommt dem in der Diplomarbeit vorgestellten Ansatz schon recht nahe, kann aber von einer wesentlich stärker eingeschränkten Umgebung profitieren. Es wird wieder eine rein zweidimensionale Fragestellung bearbeitet, und es werden auch nur Bilder einer ebenen Szene, des Bodens, verarbeitet.

2.2.3 Kooperation zwischen einer Panoramakamera und einer mobilen Kamera

Ebenfalls eine sphärische Panoramakamera kommt in [13] zum Einsatz. Die Arbeit stellt ein System vor, mit dem ein mobiler Roboter seine Position in der Ebene in regelmäßigen Abständen neu bestimmen kann, indem er künstliche Landmarken an den Wänden anpeilt. Als Landmarken kommen rote Kreisscheiben vor einem weißen Hintergrund zum Einsatz.

Da eine sphärische Kamera für Peilungsaufgaben schlecht geeignet ist (die Sehstrahlen, die die Kamera verlassen, entspringen keinem gemeinsamen Punkt), wird die Kamera nur eingesetzt, um eine grobe Schätzung zu bekommen, in welcher Richtung eine Marke zu finden ist.

Anschließend wird eine mobile Kamera, die auf einer Schwenk-Kipp-Einheit befestigt ist, in diese Richtung gelenkt, um eine hochaufgelöste Aufnahme der Landmarke zu erhalten. Diese Aufnahme wird verwendet, um eine genaue Peilung zur Marke zu gewinnen. Dabei werden perspektivische Verkürzungen nicht berücksichtigt, als Peilung wird der Pixelschwerpunkt der Marke genommen.

Durch den geringen Öffnungswinkel der mobilen Kamera ist diese Messung sehr präzise. Vernachlässigt man perspektivische Einflüsse, kann angenommen werden, daß der Markenmittelpunkt auf weniger als zwei Pixel genau bestimmt werden kann. In diesem Fall spricht die Arbeit von einer Genauigkeit von 0,018° und 0,013° für die *yaw*- und *pitch*-Winkel der Peilung. Die Positioniergenauigkeit der mobilen Kameraeinheit wird mit 0,014°/0,014° angegeben.

Aus den so gewonnenen Peilungen wird die Roboterposition über Triangulierungsmethoden aus der Vermessungstechnik gewonnen. Die Positionsrekonstruktion wird dadurch erleichtert, daß der Roboter immer auf horizontalem Boden steht.

Durch den kooperativen Ansatz konnte eine sehr hohe Genauigkeit erreicht werden. Allerdings benötigt der Meßvorgang recht viel Zeit, da jede Landmarke von der Kamera einzeln angefahren werden muß.

Kapitel 3

Grundlagen

Dieses Kapitel stellt einige Grundlagen zur Arbeit vor. Die Erklärungen sind bewußt sehr kurz gehalten. Für genauere Beschreibungen wird jeweils auf entsprechende Literatur verwiesen.

3.1 Geometrische Grundlagen

Diese Ausarbeitung setzt Grundkenntnisse in Geometrie und linearer Algebra voraus. Es folgt eine kleine Auswahl an Themen, die nicht jedem Leser voll vertraut sein werden.

3.1.1 Projektive Geometrie

Eine erschöpfende Einführung in die projektive Geometrie würde den Rahmen dieser Arbeit sprengen. Ich gebe daher nur kurz die wichtigsten Tatsachen wieder, die für das Verständnis dieser Arbeit wichtig sein könnten. Der interessierte Leser sei auf die übersichtliche Darstellung in [14] verwiesen. Eine sehr gute und ausführliche Einführung in projektive Geometrie liefert [15].

Ein Punkt im projektiven Raum $\mathcal{P}^n$ wird durch einen Vektor $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_{n+1})^\top$ repräsentiert. Mindestens eine der Komponenten muß ungleich Null sein. Diese Koordinatendarstellung bezeichnet man als *homogenene Koordinaten*. Ich schreibe homogene Vektoren als $\tilde{\boldsymbol{x}}$, während Punkte und Vektoren des euklidischen Vektorraums $\mathbb{R}^n$ als $\boldsymbol{x}$ geschrieben werden.

Jeder Punkt $\boldsymbol{x} \in \mathbb{R}^n$ kann im homogenen Raum von einem Vektor $\tilde{\boldsymbol{x}} \in \mathcal{P}^n$ repräsentiert werden. Diese Repräsentation ist nicht eindeutig; zu jedem $\boldsymbol{x}$ existieren beliebig viele Repräsentanten $\tilde{\boldsymbol{x}}$. Zwei homogene Vektoren $\tilde{\boldsymbol{x}}$ und $\tilde{\boldsymbol{y}}$ bezeichnen den selben euklidischen Punkt, wenn es ein $\lambda \neq 0$ gibt, so das gilt $x_i = \lambda y_i$ für alle $i \in \{1, \ldots, n\}$. Dann kann auch $\tilde{\boldsymbol{x}} \sim \tilde{\boldsymbol{y}}$ geschrieben werden.

Die projektive Ebene $\mathcal{P}^2$

Der projektive Raum $\mathcal{P}^2$ wird auch als die *projektive Ebene* bezeichnet. Ein Punkt in $\mathcal{P}^2$ ist durch einen Vektor $\tilde{\boldsymbol{m}} = (x, y, w)^\top$ gegeben. Analog wird auch eine Gerade $\tilde{\boldsymbol{l}}$ durch einen dreikomponentigen Vektor beschrieben. Ein Punkt $\tilde{\boldsymbol{m}}$ liegt auf einer Geraden $\tilde{\boldsymbol{l}}$, genau dann wenn

$$\tilde{\boldsymbol{l}}^\top \tilde{\boldsymbol{m}} = 0\,. \tag{3.1}$$

In dieser Gleichung dürfen $\tilde{\boldsymbol{m}}$ und $\tilde{\boldsymbol{l}}$ vertauscht werden. Diese Tatsache zeigt, daß in der projektiven Ebene formal kein Unterschied zwischen Punkten und Geraden besteht. Das führt uns zu einem Zusammenhang, der als *Dualitätsprinzip* bezeichnet wird. Es gilt nämlich:

Jede wahre Aussage über die projektive Ebene läßt sich umschreiben, indem man Punkte durch Geraden und Geraden durch Punkte ersetzt, und sie bleibt weiterhin gültig.

Insbesondere gilt: Wenn eine Gerade $\tilde{\boldsymbol{l}}$ durch die Punkte $\tilde{\boldsymbol{m}}_1$ und $\tilde{\boldsymbol{m}}_2$ geht (das heißt also, daß sie diese schneidet), dann schneiden sich auch die beiden *Geraden* $\tilde{\boldsymbol{m}}_1$ und $\tilde{\boldsymbol{m}}_2$ im *Punkt* $\tilde{\boldsymbol{l}}$.

Die Gerade $\tilde{\boldsymbol{l}}$ durch die Punkte $\tilde{\boldsymbol{m}}_1$ und $\tilde{\boldsymbol{m}}_2$ berechnet sich nach

$$\tilde{\boldsymbol{l}} = \tilde{\boldsymbol{m}}_1 \times \tilde{\boldsymbol{m}}_2 \,. \tag{3.2}$$

Aufgrund des Dualitätsprinzips kann Gleichung 3.2 ebenso zur Berechnung eines Geradenschnitts verwendet werden.

Der projektive Raum $\mathcal{P}^3$

Ein Punkt $\boldsymbol{m}$ im $\mathcal{P}^3$ wird als $\tilde{\boldsymbol{m}} = (x, y, z, w)^\top$ geschrieben. Im $\mathcal{P}^3$ ist der duale Partner des Punkts die Ebene. Entsprechend werden auch Ebenen als Vierervektoren geschrieben. Ein Punkt $\boldsymbol{m}$ liegt auf einer Ebenen H, wenn $\tilde{\boldsymbol{h}}^\top \tilde{\boldsymbol{m}} = 0$.

Ist von einer Ebene bekannt, daß sie durch drei Punkte geht, dann erhält man, interpretiert man die Punktrepräsentationen als Ebenen und die Ebenenrepräsentationen als Punkte, drei Ebenen, die sich in einem Punkt schneiden.

Der Schnitt dreier Ebenen berechnet sich wie folgt. Gegeben: Die Ebenen $\tilde{\boldsymbol{g}}$, $\tilde{\boldsymbol{h}}$, $\tilde{\mathbf{i}}$ als Ebenenvektoren $\in \mathcal{P}^3$. Der Schnitt $\tilde{\boldsymbol{x}}$ ergibt sich dann zu

$$\tilde{\boldsymbol{x}} = \left(\begin{vmatrix} \tilde{\boldsymbol{g}}_2 & \tilde{\boldsymbol{h}}_2 & \tilde{\mathbf{i}}_2 \\ \tilde{\boldsymbol{g}}_4 & \tilde{\boldsymbol{h}}_4 & \tilde{\mathbf{i}}_4 \\ \tilde{\boldsymbol{g}}_3 & \tilde{\boldsymbol{h}}_3 & \tilde{\mathbf{i}}_3 \end{vmatrix}, \begin{vmatrix} \tilde{\boldsymbol{g}}_1 & \tilde{\boldsymbol{h}}_1 & \tilde{\mathbf{i}}_1 \\ \tilde{\boldsymbol{g}}_3 & \tilde{\boldsymbol{h}}_3 & \tilde{\mathbf{i}}_3 \\ \tilde{\boldsymbol{g}}_4 & \tilde{\boldsymbol{h}}_4 & \tilde{\mathbf{i}}_4 \end{vmatrix}, \begin{vmatrix} \tilde{\boldsymbol{g}}_1 & \tilde{\boldsymbol{h}}_1 & \tilde{\mathbf{i}}_1 \\ \tilde{\boldsymbol{g}}_4 & \tilde{\boldsymbol{h}}_4 & \tilde{\mathbf{i}}_4 \\ \tilde{\boldsymbol{g}}_2 & \tilde{\boldsymbol{h}}_2 & \tilde{\mathbf{i}}_2 \end{vmatrix}, \begin{vmatrix} \tilde{\boldsymbol{g}}_1 & \tilde{\boldsymbol{h}}_1 & \tilde{\mathbf{i}}_1 \\ \tilde{\boldsymbol{g}}_2 & \tilde{\boldsymbol{h}}_2 & \tilde{\mathbf{i}}_2 \\ \tilde{\boldsymbol{g}}_3 & \tilde{\boldsymbol{h}}_3 & \tilde{\mathbf{i}}_3 \end{vmatrix} \right)^\top \tag{3.3}$$

Aufgrund des Dualitätsprinzips läßt sich mit Gleichung 3.3 auch die Ebene $\tilde{\boldsymbol{x}}$ durch die drei Punkte $\tilde{\boldsymbol{g}}$, $\tilde{\boldsymbol{h}}$ und $\tilde{\mathbf{i}}$ berechnen.

Homogene Koordinaten im *Ray-space*

Eingangs wurde erklärt, daß zu einem euklidischen Punkt eine homogene Repräsentation existiert. Umgekehrt ist die euklidische Interpretation aber nicht die einzige Vorstellung, die man mit einem homogenen Vektor verbinden kann.

Betrachtet man die Menge der möglichen homogenen Repräsentanten eines Vektors $\boldsymbol{x} \in \mathbb{R}^2$ innerhalb des $\mathcal{P}^2$, so bilden sie dort, interpretiert man die Vektoren des $\mathcal{P}^2$ einmal als Punkte des $\mathbb{R}^3$, eine Gerade durch den Ursprung. Entsprechend ist die Gerade $\{\lambda\tilde{\boldsymbol{x}} \,|\, \lambda \in \mathbb{R}\} \subset \mathbb{R}^3$ eine weitere mögliche Deutung eines homogenen Vektors $\tilde{\boldsymbol{X}}$.

Anhang D macht von dieser Interpretation gebrauch.

Koniken

Eine *Konik* im $\mathcal{P}^2$ ist der geometrische Ort aller Punkte $\tilde{\boldsymbol{m}}$, die die homogene quadratische Gleichung

$$\tilde{\boldsymbol{m}}^\top C \tilde{\boldsymbol{m}} = 0 \tag{3.4}$$

erfüllen, wobei $C \in \mathbb{R}^{3\times 3}$ eine symmetrische Matrix ist, die bis auf eine Skalierung definiert ist. Eine Konik hängt daher von fünf Parametern ab.

Interpretiert man die $\tilde{\boldsymbol{m}}$, die die Gleichung erfüllen, als Geraden des *Ray-space* $\mathcal{P}^2$, so bilden diese einen elliptischen Doppelkegel, dessen Spitze im Ursprung liegt.

Wird C als

$$\begin{pmatrix} a & b & d \\ b & c & e \\ d & e & f \end{pmatrix} \tag{3.5}$$

angenommen, so läßt sich Gleichung 3.4 auch als

$$am_1^2 + 2bm_1m_2 + cm_2^2 + 2dm_1m_3 + 2em_2m_3 + fm_3^2 = 0 \tag{3.6}$$

schreiben. Für $m_3 = 1$ entspricht dies der impliziten Darstellung einer Kurve 2. Ordnung oder auch eines *Kegelschnitts* (vgl. [1]). Das ist nicht weiter verwunderlich, da Gleichung 3.6 dann den Schnitt des elliptischen Doppelkegels mit der Ebene $z = 1$ beschreibt.

Einen Kegelschnitt bilden daher auch die Punkte einer Konik, wenn sie als Punkte der projektiven Ebene interpretiert werden.

3.1.2 Das Einheitsparaboloid über der Ebene

Für das Verständnis der Kalibrierungsmethode aus Abschnitt C.2 kann folgender Zusammenhang hilfreich sein.

Projiziert man Punkte $p = (x, y)^\top$ aus der Ebene auf das Einheitsparaboloid $z = x^2 + y^2$, so daß $p' = (x, y, x^2 + y^2)^\top$, dann ergeben sich für die resultierende Punktmenge folgende Eigenschaften:

- Waren $p_1, p_2, ..., p_n$ konzentrisch angeordnet, so kommen die p_i' nach der Projektion auf einer Ebene zu liegen.
- Diese Ebene schneidet das Einheitsparaboloid in einer Ellipse, die in die xy-Ebene zurückprojiziert den Kreis ergibt, der die p_i enthält.
- Die Schnittebene mit dem Paraboloid geht durch $(m_x, m_y, m_x^2 + m_y^2 + r^2)^\top$, wenn die p_i auf einem Kreis K mit Mittelpunkt $(m_x, m_y)^\top$ und Radius r liegen. Sie hat in x-Richtung die Steigung $2m_x$ und in y-Richtung $2m_y$. Der Ebenenvektor ist also

$$H_K: \quad (-2m_x, -2m_y, 1, m_x^2 + m_y^2 - r^2)^\top. \tag{3.7}$$

Weitere Eigenschaften, die sich bei der Projektion einer Punktmenge an das Einheitsparaboloid ergeben, können [16] entnommen werden.

3.1.3 Quaternionen

Quaternionen sind ein elegantes Mittel für die Behandlung von Rotationen im Raum. Jede Rotation im Raum kann als Quaternion ausgedrückt werden. Gegenüber anderen Darstellungsformen wie EULER-Winkeln oder 3×3-Matrizen besitzen Quaternionen einige Vorzüge. Dieser Abschnitt stellt nur die Eigenschaften von Quaternionen dar, wie sie für die Rotationsrepräsentation benötigt werden. Eine ausführliche Darstellung des historischen und algebraischen Hintergrunds von Quaternionen findet sich in [17].

Quaternionen werden im allgemeinen geschrieben als

$$\dot{\boldsymbol{w}} = a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d \tag{3.8}$$

und werden als Erweiterung der imaginären Zahlen verstanden. (Tatsächlich ist $\mathbb{C}$ in den Quaternionenraum eingebettet, und Quaternionen mit $c = d = 0$ können wie komplexe

Zahlen (a, b) behandelt werden.) Für die imaginären Einheiten i, j und k gelten die Regeln:

$$\begin{aligned} &\mathrm{ii} = \mathrm{jj} = \mathrm{kk} = -1 \\ &\mathrm{ij} = \mathrm{k} \qquad\qquad \mathrm{ji} = -\mathrm{k} \\ &\mathrm{jk} = \mathrm{i} \qquad\qquad \mathrm{kj} = -\mathrm{i} \\ &\mathrm{ki} = \mathrm{j} \qquad\qquad \mathrm{ik} = -\mathrm{j} \end{aligned} \tag{3.9}$$

Für die weitere Betrachtung ist es wichtig, daß Quaternionen sich auch als vierdimensionale Vektoren der Form

$$\boldsymbol{w} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \tag{3.10}$$

schreiben lassen. Man beachte, daß die Schreibweise $\boldsymbol{w}$ den Vektor zum Quaternion $\dot{\boldsymbol{w}}$ bezeichnet. Es folgen die wichtigsten Definitionen und Rechenregeln zu Quaternionen.

Das **konjugierte** Quaternion ist

$$\dot{\boldsymbol{w}}^* = a - \mathrm{i}b - \mathrm{j}c - \mathrm{k}d. \tag{3.11}$$

Auf Quaternionen sind folgende Operationen definiert:

Die **Multiplikation** kann durch Ausmultiplizieren der Faktoren gefunden werden und ergibt sich zu

$$\begin{aligned} \dot{\boldsymbol{w}}\dot{\boldsymbol{v}} \;=\; & (a_w a_v - b_w b_v - c_w c_v - d_w d_v) + \\ & \mathrm{i}(b_w a_v + a_w b_v + d_w c_v + c_w d_v) + \\ & \mathrm{j}(c_w a_v + d_w b_v + a_w c_v - b_w d_v) + \\ & \mathrm{k}(d_w a_v - c_w b_v + b_w c_v + a_w d_v)\,. \end{aligned} \tag{3.12}$$

Diese Operation kann auch als Matrix/Vektor-Multiplikation geschrieben werden:

$$\dot{\boldsymbol{w}}\dot{\boldsymbol{v}} = W\boldsymbol{v} = \begin{pmatrix} a_w & -b_w & -c_w & -d_w \\ b_w & a_w & -d_w & c_w \\ c_w & d_w & a_w & -b_w \\ d_w & -c_w & b_w & a_w \end{pmatrix} \begin{pmatrix} a_v \\ b_v \\ c_v \\ d_v \end{pmatrix} \tag{3.13}$$

Man beachte, daß die Multiplikation nicht kommutativ ist, d.h. $\dot{\boldsymbol{w}}\dot{\boldsymbol{v}} \neq \dot{\boldsymbol{v}}\dot{\boldsymbol{w}}$. Es gilt aber

$$\dot{\boldsymbol{v}}\dot{\boldsymbol{w}} = \overline{W}\boldsymbol{v} \quad \text{mit} \quad \overline{W} = \begin{pmatrix} a_w & -b_w & -c_w & -d_w \\ b_w & a_w & -d_w & c_w \\ c_w & d_w & a_w & -b_w \\ d_w & -c_w & b_w & a_w \end{pmatrix} \begin{pmatrix} a_v \\ b_v \\ c_v \\ d_v \end{pmatrix}. \tag{3.14}$$

$\overline{W}$ unterscheidet sich von W durch eine transponierte untere rechte 3×3-Submatrix.

Das **Skalarprodukt** ist analog zu seinem Vektorpendant definiert als

$$\dot{\boldsymbol{w}} \bullet \dot{\boldsymbol{v}} = \boldsymbol{w}^\top \boldsymbol{v} = a_w a_v + b_w b_v + c_w c_v + d_w d_v\,. \tag{3.15}$$

Der **Betrag** eines Quaternions ist definiert als

$$\|\dot{\boldsymbol{w}}\| = \sqrt{\dot{\boldsymbol{w}} \bullet \dot{\boldsymbol{w}}}\,. \tag{3.16}$$

Quaternionen mit $\|\dot{\boldsymbol{w}}\| = 1$ werden als **Einheitsquaternionen** bezeichnet.

Einheitsquaternionen haben wichtige Eigenschaften:

- $(\dot{\boldsymbol{w}}\dot{\boldsymbol{v}}) \bullet \dot{\boldsymbol{r}} = \dot{\boldsymbol{w}} \bullet (\dot{\boldsymbol{r}}\dot{\boldsymbol{v}}^*)$ (Das gilt nicht nur für Einheitsquaternionen.)
- Die Rotation eines Vektors $\boldsymbol{r}$ um die Achse $\dot{\boldsymbol{n}} = (n_x, n_y, n_z)^\top$ um den Winkel φ wird durch die Multiplikation

$$\dot{\boldsymbol{w}}\dot{\boldsymbol{r}}\dot{\boldsymbol{w}}^* \tag{3.17}$$

beschrieben, wobei $\dot{\boldsymbol{w}}$ das Einheitsquaternion $\cos\frac{\varphi}{2} + \sin\frac{\varphi}{2}\dot{\boldsymbol{n}}$ ist und die Vektoren $\boldsymbol{v}$ und $\boldsymbol{n}$ in die Quaternionen

$$\begin{aligned} \dot{\boldsymbol{v}} &= 0 + \mathrm{i}v_x + \mathrm{j}v_y + \mathrm{k}v_z \\ \dot{\boldsymbol{n}} &= 0 + \mathrm{i}n_x + \mathrm{j}n_y + \mathrm{k}n_z \end{aligned} \tag{3.18}$$

umgeschrieben wurden. Das Ergebnis ist wiederum ein Quaternion, das als ein analog zu Gleichung 3.18 umgeschriebener Vektor interpretiert werden kann.

3.2 Lösung eines überbestimmten linearen Gleichungssystems

In dieser Arbeit ist häufig ein System der Form

$$A\boldsymbol{x} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \end{pmatrix}, \quad m \geq n \tag{3.19}$$

nach $\boldsymbol{x}$ aufzulösen. Das Gleichungssystem ist im allgemeinen überbestimmt und nicht lösbar. In diesem Fall bestimmen wir $\boldsymbol{x}$ nach der Methode der kleinsten Quadrate. Gesucht ist dabei der Lösungsvektor $\boldsymbol{x}$, der die Summe der Fehlerquadrate

$$\|A\boldsymbol{x} - \boldsymbol{d}\|^2 \tag{3.20}$$

minimiert. Dieses $\boldsymbol{x}$ kann zum Beispiel mit der Technik der *Pseudo-Inversen* bestimmt werden. Die Pseudo-Inverse A^+ ist definiert als

$$A^+ = (A^\top A)^{-1} A^\top. \tag{3.21}$$

A^+ kann explizit berechnet werden. Wenn A quadratisch ist und vollen Rang hat, gilt $A^+ = A^{-1}$. Das gesuchte $\boldsymbol{x}$ ergibt sich in jedem Fall zu

$$\boldsymbol{x} = A^+\boldsymbol{d}. \tag{3.22}$$

Eine gute Einführung in die Technik der Pseudo-Inversen findet sich in [18].

3.3 Parabolische Panoramakameras

Die Arbeit sieht den Einsatz einer Panoramakamera mit konvexem Parabolspiegel vor. Parabolspiegel haben eine wichtige Eigenschaft: Sehstrahlen, die parallel zur Symmetrieachse auf den Spiegel treffen, werden so abgelenkt, daß sie alle dem Fokus des Paraboloids zu entspringen scheinen (siehe Abbildung 3.1).

Entsprechend besitzt eine Panoramakamera, die auf einem parabolischen Spiegel aufbaut, einen Strahlengang, bei dem die Sehstrahlen als Bündel paralleler Strahlen auf die Spiegeloberfläche treffen. Dies kann auf unterschiedliche Weise erreicht werden. Abbildung 3.2 zeigt zwei Kameras mit zwei unterschiedlichen Ansätzen.

Die linke Kamera ist eine Panoramakamera für Einzelbildaufnahmen, die uns zu Beginn der Arbeit zur Verfügung stand.

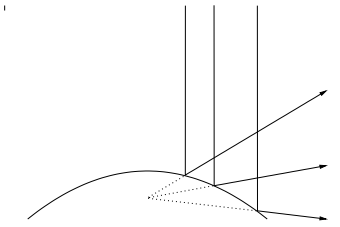

Abbildung 3.1: Ablenkung der Sehstrahlen durch einen parabolischen Spiegel. Gezeigt ist ein Schnitt durch den Spiegel. Die parallel eintreffenden Sehstrahlen scheinen nach der Reflexion alle dem Fokus zu entspringen.

Rechts ist eine Abbildung der wesentlich kompakteren NTSC-Kamera, die später im Kopfverfolgungssystem zum Einsatz kommen soll. (Ihre technischen Daten sind in Anhang K zu finden.)

Abbildung 3.2. Zwei Panoramakameras unterschiedlicher Bauweise. Bei beiden befindet sich unten ein verspiegeltes Kreisparaboloid als Primärspiegel. *Links:* Eine Linsenoptik schickt die parallelisierten Sehstrahlen einer handelsüblichen Einzelbildkamera, die von oben auf das Objektiv geschraubt wird, auf den Spiegel. *Rechts:* Bei diesem Modell befindet sich der lichtempfindliche CCD-Chip im Sockel. Ein konkaver Parabolspiegel, der dem Primärspiegel entgegengesetzt ist, fokusiert die parallel eintreffenden Lichtstrahlen auf ein Loch in dessen Scheitel. Darunter befindet sich ein standardisiertes Kameramodul, das das Bild aufnimmt.

Die oben erwähnte Eigenschaft parabolischer Panoramakameras ist deshalb so wichtig, da über die Kamerabilder Peilungen zu Objekten in der Szene aufgenommen werden sollen. Mehrere Peilungen können aber nur sinnvoll zueinander in Beziehung gesetzt werden, wenn sie sich alle auf den selben Punkt beziehen. Dazu müssen sich die Geraden, auf denen die Sehstrahlen die Kamera verlassen, in einem einzigen Punkt schneiden.

Andere Spiegelformen, zum Beispiel sphärische Spiegel, weisen diese Eigenschaft nicht auf und sind deshalb für die hier vorgestellte Anwendung nicht geeignet.

3.4 Digitale Bildverarbeitung

3.4.1 Farbmodelle

Die Netzhaut des menschlichen Auges verfügt neben den Stäbchen für die Wahrnehmung von Helligkeitsunterschieden über drei verschiedene Arten von Zäpfchen zur Farbwahrnehmung. Durch das Mischen von höchstens drei Grundfarben ist es möglich, jeden subjektiven Farbeindruck zu erzielen.

Je nachdem, ob die Mischung additiv oder subtraktiv erfolgt, hat sich dabei ein System aus Rot, Grün und Blau (*additive Farbmischung*) bzw. deren Komplementärfarben Cyan, Magenta und Gelb (*subtraktive Farbmischung*) als geeignet erwiesen. Die beiden Farbdarstellungsverfahren werden nach den Anfangsbuchstaben ihrer Farbkomponenten *RGB*- bzw. *CMY*-Darstellung genannt. Monitore arbeiten nach dem Prinzip der additiven Farbmischung, weshalb sie auf dem RGB-System aufbauen. Auf Druckern, die ihre Farben im allgemeinen subtraktiv mischen, wird der CMY-Darstellung oft noch ein Kontrastwert K hinzugefügt, um mit Hilfe einer schwarzen Hilfsfarbe starke Kontraste besser wiedergeben zu können. Man spricht dann von einer *CMYK*-Darstellung.

Die Farbmodelle RGB und CMY(K) sind stark an der Art der physikalischen Visualisierung orientiert. Für eine intuitive Beschreibung von Farbeindrücken sind sie dennoch oft nicht geeignet.

Aus diesem Grunde wurden alternative Modelle gesucht, um sichtbare Farben zu beschreiben. Für eine Visualisierung der alternativ beschriebenen Farben auf Monitoren und Druckern müssen für diese Modelle wieder Umrechnungsverfahren von und nach RGB bzw. CMY(K) existieren.

Im Graphik- und Design-Bereich hat sich unter anderem das *HSV*-System etabliert. In diesem System wird eine Farbe anhand ihres Winkels im Farbkreis (*hue*), ihrer Sättigung (*saturation*) und ihrer Intensität (*value*) beschrieben. Die Sättigung bezeichnet auch die *Reinheit* einer Farbe, die um so stärker ist, je geringer der enthaltene *Weißanteil* ist.

Für unsere Zwecke ist das HSV-System insbesondere deshalb interessant, weil Beleuchtungseffekte durch weiße Lichtquellen und Schattenwürfe theoretisch nur Einfluß auf Saturation und Value haben. Hue bleibt weitestgehend unverändert (siehe Abschnitt F).

Mit der Entwicklung des Farbfernsehens kamen weitere Farbmodelle auf. Zwar mischen Fernsehröhren ihre Bilder ausnahmslos nach dem RGB-System. Bei dem Entwurf der Fernsehübertragungsverfahren mußten aber weitere Gesichtspunkte berücksichtigt werden: Einer Fernsehbildübertragung steht nur eine begrenzte Bandbreite zur Verfügung. Ziel ist es natürlich, mit dieser Bandbreite eine möglichst gute Bildqualität zu erreichen. Dabei kamen wieder physiologische Betrachtungen ins Spiel: Weil das Auge empfindlicher für Helligkeit und Kontrast eines Bildes ist als für dessen Farben, hat man Systeme gesucht, in denen diese drei Eindrücke getrennt beschrieben werden, damit man Helligkeit und Kontrast mit einer höheren Qualität und Auflösung übertragen kann als die eigentlichen Farbinformation.

Da diese Darstellung nach der Übertragung wieder nach RGB zu konvertieren ist, suchte man gleichzeitig eine Farbkodierung, die sich mit elektronischer Analogtechnik wieder leicht nach RGB wandeln ließ.

Im europäischen Raum wurde schließlich die PAL-Fernsehnorm entwickelt, die auf dem *YUV*-Farbsystem aufbaut. Die etwas niedriger aufgelöste amerikanische NTSC-Norm (siehe nächster Abschnitt) verwendet hingegen das sogenannte *YIQ*-System.

Beide Systeme stellen eine Gewisse Trennung der genannten Seheindrücke her und gehen durch eine Basistransformation aus dem RGB-System hervor. Diese Transformation ist leicht in Analogtechnik zu realisieren.

Es folgen die Umrechnungsvorschriften für YUV und YIQ.

RGB $\Longleftrightarrow$ YUV Konvertierung

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1{,}140 \\ 1 & -0{,}394 & -0{,}581 \\ 1 & 2{,}028 & 0 \end{pmatrix} \begin{pmatrix} y \\ u \\ v \end{pmatrix} \tag{3.23}$$

$$\begin{pmatrix} y \\ u \\ v \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}147 & -0{,}289 & 0{,}437 \\ 0{,}615 & -0{,}515 & -0{,}100 \end{pmatrix} \begin{pmatrix} r \\ g \\ b \end{pmatrix} \tag{3.24}$$

RGB $\Longleftrightarrow$ YIQ Konvertierung

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0{,}956 & 0{,}621 \\ 1 & -0{,}272 & -0{,}647 \\ 1 & -1{,}105 & 1{,}702 \end{pmatrix} \begin{pmatrix} y \\ i \\ q \end{pmatrix} \tag{3.25}$$

$$\begin{pmatrix} y \\ i \\ q \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}274 & -0{,}322 \\ 0{,}212 & -0{,}523 & 0{,}311 \end{pmatrix} \begin{pmatrix} r \\ g \\ b \end{pmatrix} \tag{3.26}$$

3.4.2 Die NTSC-Fernsehnorm

Die Panoramakamera, die im fertigen System zum Einsatz kommen wird, liefert Bilder in der amerikanischen Fernsehnorm NTSC. Es soll daher kurz auf die wichtigsten Eigenschaften dieses Formats eingegangen werden, sofern diese Arbeit davon betroffen ist.

Bilder in der NTSC-Norm haben eine Auflösung von 640×480 Pixeln. Die Pixelfarbe ist im YIQ-System (siehe vorherigen Abschnitt) definiert.

Jedes Bild ist aus zwei *Halbbildern* zusammengesetzt. Das erste Halbbild erstreckt sich über die ungeradzahligen Bildzeilen 1, 3, 5, ..., das zweite über die Zeilen 2, 4, 6, ... Die Halbbilder eines Vollbilds werden hintereinander übertragen. Der in unserem PC-System für das Auslesen der NTSC-Kamera eingesetzte *Frame grabber* liefert prinzipiell Vollbilder zurück.

Pro Sekunde werden im NTSC-Format 60 Halbbilder übertragen. Das entspricht 30 Vollbildern. Die Aufnahme der Halbbilder geschieht mit dem gleichen zeitlichen Versatz, mit dem sie übertragen werden. Dadurch haben innerhalb eines Vollbilds die geradzahligen Bildzeilen einen anderen Aufnahmezeitpunkt als die ungeradzahligen. Hierdurch kommt es innerhalb eines Vollbilds zu sogenannten *Kamm-Artefakten* an bewegten Objekten (vgl. Abschnitt 5.4.3).

Kapitel 4

Systementwurf

Ein wichtiger Teil der Arbeit war der allgemeine Entwurf eines Panoramakamera-basierten Verfolgungssystems. Dabei wurde vor allem auf Vollständigkeit und Interoperabilität mit ergänzenden Verfolgungssystemen Wert gelegt.

Der Entwurf geht in manchen Teilen über das hinaus, was im Rahmen der Diplomarbeit implementiert werden konnte. Um so wichtiger war es, daß sich die Implementierung an dessen Konzept hält, um eine Weiterentwicklung zu erleichtern.

4.1 Allgemeiner Aufbau

Der Kopfverfolger läßt sich am besten anhand des Datenflusses im System beschreiben. Der Datenfluß über die einzelnen Komponenten soll hier kurz umrissen werden. Die Komponenten selbst werden in Abschnitt 4.2 genauer erklärt.

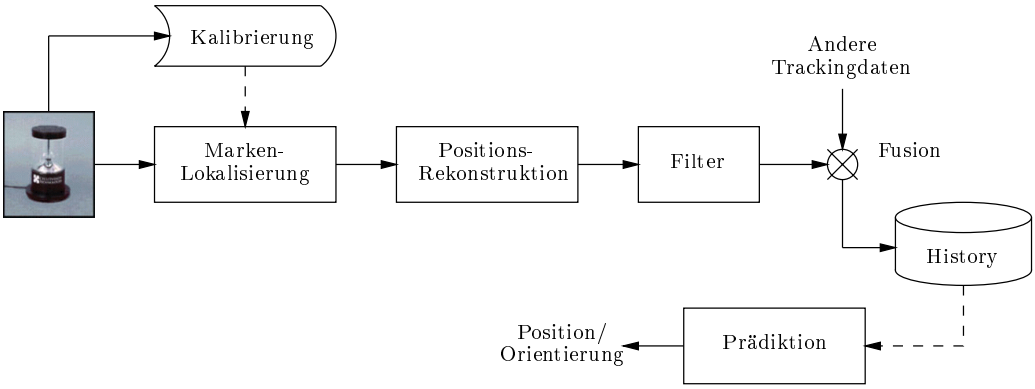

Abbildung 4.1. Allgemeiner Aufbau des Verfolgungssystems. Das System ist als Prozeßablaufdiagramm dargestellt.

Abbildung 4.1 zeigt den groben Aufbau des Systems. Ausgehend von den Bildern der auf dem Kopf befestigten Panoramakamera ermittelt die *Landmarkenlokalisierung* Peilungen zu den sichtbaren Landmarken.

Auf Basis dieser Peilungen und einer Liste der vermessenen Landmarken entwickelt die *Positionsrekonstruktion*[1] eine Hypothese für die aktuelle Kameraposition. Diese Hypothese unterliegt wie jede Messung systematischen (siehe Abschnitt 7.6) und stochastischen Fehlern.

Um die stochastischen Fehler zu verringern, durchlaufen alle von der Positionsrekonstruktion gelieferten Kamerapositionen einen *Filter*. Die resultierenden Positionen können

[1] Ich spreche im folgenden der Einfachheit halber von der *Positions*rekonstruktion, auch wenn sie neben der Position auch die Orientierung der Kamera rekonstruiert.

wahlweise noch mit den Ergebnissen anderer Verfolgungssysteme abgeglichen werden (*Multi-Sensor-Fusion*).

Die so gewonnenen Kamerapositionen könnten bereits als Antwort des Gesamtsystems verwendet werden. Das direkte Abgreifen der rekonstruierten Positionen hat nur einen entscheidenden Nachteil: Durch die Latenzzeit, die durch die Bildakquisition und die Verarbeitung in den vorangegangenen Komponenten unweigerlich entstanden ist, sind die gelieferten Daten stets veraltet. Die Situation wird sogar noch daduch verschärft, daß die Kamera- bzw. Kopfposition dafür benötigt wird, Einblendungen an die Blickrichtung anzupassen. In diesem Fall kommt nämlich noch die Bildgenerierung für die Einblendung als weitere Latenz hinzu: Tatsächlich interessiert uns nie die Kopfposition zum *aktuellen Zeitpunkt*, sondern nur zum *Zeitpunkt der Einblendung*.

Sowohl die Position zum aktuellen Zeitpunkt als auch zukünftige Positionen können aber nur mit Hilfe einer Vorhersageeinheit (*Prädiktion*) geschätzt werden. Dieser stehen die letzten n Messungen zur Verfügung, die in der sogenannten *History* abgelegt sind.

In den meisten Fällen wird man eher auf die Daten der Prädiktion als auf die der eigentlichen Positionsrekonstruktion zurückgreifen.

4.2 Komponenten

Im folgenden sollen die eben skizzierten Teile des Gesamtsystems genauer beschrieben werden. Die Komponenten werden dabei in größtmoglicher Allgemeinheit behandelt. So spielen zum Beispiel die Ausführung der Landmarken, der Aufbau der Kamera und andere Implementierungsentscheidungen noch keine Rolle.

4.2.1 Landmarkenlokalisierung

Die Landmarkenlokalisierung erhält die rohen Kamerabilder der Panoramakamera. Als Hintergrundwissen benötigt sie Information über das Aussehen der verwendeten Landmarken und eine *Kalibrierung* der Kamera, die zu jedem Pixel im Bild den zugehörigen Sehstrahl in die Szene liefert.

Das Lokalisierungsmodul betrachtet die Liste der vermessenen Landmarken, in der zu jeder Landmarke deren *Typ* verzeichnet ist (siehe auch Abschnitt J.1). Nach den zu erwartenden Typen wird nun explizit gesucht (*Detektion*).

Anschließend müssen die im Bild gefundenen Marken exakt angepeilt werden. Je nach Form der Landmarken gestaltet sich dieser Vorgang mehr oder weniger aufwendig: Mit jeder Landmarke ist ein bestimmter *Referenzpunkt* verbunden. Dieser muß robust und unabhängig von perspektivischer Verkürzung und teilweiser Verdeckung gefunden werden. Die Anforderungen an die Markenlokalisierung werden in Abschnitt 4.3.2 genauer diskutiert.

Das Ergebnis der Landmarkenlokalisierung ist schließlich eine Menge von Tupeln (*Markentyp*, *Peilung*).

4.2.2 Positionsrekonstruktion

Nachdem die Peilungen zu den sichtbaren Landmarken ermittelt wurden, fehlt noch ein wichtiger Schritt zur erfolgreichen Positionsrekonstruktion: Die *Markenidentifizierung*.

Markenidentifizierung

Die Landmarkenlokalisierung hat zwar die Information geliefert, in welcher Richtung welcher Landmarken*typ* erkannt wurde. — Die Information, *welche* der bekannten Landmarken an

der fraglichen Stelle zu sehen ist, d.h., welche *Identität* die erkannte Landmarke besitzt, fehlt noch zu einer vollständigen Deutung der gefundenen Peilung.

Existieren von einem Landmarkentyp mehrere Marken, so ist also die plausibelste Zuordnung „Peilung $\longmapsto$ bekannte Landmarke“ zu finden.

Ist im System zu jedem möglichen Landmarkentyp nur eine Marke enthalten, ist dieser Schritt trivial.

Im allgemeinen Fall hingegen ist möglichst viel Umweltwissen in den Identifizierungsprozeß einzubeziehen, damit nicht alle möglichen Permutationen „durchprobiert“ werden müssen (siehe auch Abschnitt 4.3.3). Insbesondere ist es denkbar, daß die Identifizierung eine Positionsschätzung der Prädiktion verwendet, um die Plausibilität einer konkreten Zuordnung zu bestimmen.

Anmerkung: Die Markenidentifizierung ließe sich logisch auch der Landmarkenlokalisierung zuordnen. Ich habe sie aber unter dem Schritt der Positionsrekonstruktion eingeordnet, da Algorithmen denkbar sind, die Mehrdeutigkeiten in den Markenidentitäten dadurch auflösen, daß testweise ein Rekonstruktionsschritt angeworfen wird. In diesem Falle arbeiten Identifizierung und Rekonstruktion sehr eng zusammen.

Ein weiterer Grund, die Markenidentifizierung aus dem Lokalisierungsmodul herauszunehmen, ist, daß die Identifizierung, im Unterschied zur Lokalisierung, unabhängig vom Aussehen der Landmarken ist.

Rekonstruktion der Lage im Raum

Liegt eine Identifikation „Peilung $\longmapsto$ bekannte Landmarke“ vor, so kann aufgrund der bekannten Landmarkenpositionen auf die Kameraposition geschlossen werden.

Für die Positionsrekonstruktion wären drei bekannte Peilungen ausreichend. Im Allgemeinen werden aber noch mehr Landmarken zu sehen sein. Bei der Rekonstruktionsaufgabe handelt es sich also um ein überbestimmtes System.

Natürlich könnte man sich auf die drei verläßlichsten Messungen beschränken, um auf die Position zu schließen. Bezieht man hingegen alle Messungen in die Rekonstruktion mit ein, wird das Verfahren robuster gegen Meßfehler und erreicht in der Regel eine höhere Genauigkeit.

Für Rekonstruktionsverfahren, die ihre Lösung iterativ bestimmen, ist es sinnvoll, als Startwert der Iteration die Positionsschätzung der Prädiktion zu wählen.

4.2.3 Filterung

Die Filterung von Tracking-Daten ist eine häufig gestellte Aufgabe. Oft wird hierfür der KALMAN-Filter ([19,20]) eingesetzt, der bei linear beschreibbaren Systemen in der Regel zu sehr guten Ergebnissen führt. Dennoch ist der Filter für das Filtern von Kopfpositionen nicht besonders gut geeignet, da sich keine guten linearen Modelle der willkürlichen Kopfbewegungen eines Benutzers finden lassen.

Auf keinen Fall sollten die Positionsdaten ungefiltert bleiben, da sonst durch Meßrauschen und kurze „Ausreißer“ in der Positionsrekonstruktion die AR-Einblendungen stets leicht wackeln würden. Dieses Problem könnte aber bereits ein einfacher Tiefpaßfilter lösen.

Speziell für diese Arbeit war ein Filtern der Positionsdaten aber eher unerwünscht, da für eine Evaluierung des Ansatzes vor allem die tatsächlichen Ergebnisse der Positionsrekonstruktion von Interesse waren. Auf eine Implementierung eines Filters wurde deshalb verzichtet.

4.2.4 Multi-Sensor-Fusion

Das Thema der Multi-Sensor-Fusion wurde im Rahmen dieser Diplomarbeit nicht ausdrücklich bearbeitet. Aufgabe der Fusion ist es in unserem Fall, Positionsangaben unterschiedlicher Systeme zu kombinieren und dabei Schwächen des einen Systems mit den Stärken eines anderen auszugleichen. Stehen zum Beispiel Ergebnisse zweier Verfolgungssysteme zur Verfügung, wovon das eine System eine besonders gute translatorische Auflösung hat, während das andere zuverlässigere rotatorische Angaben liefert, ist es naheliegend, die Positions- und Orientierungsdaten der beiden Systeme entsprechend gewichtet miteinander abzugleichen.

4.2.5 Prädiktion

Die Prädiktion setzt ein Modell der zu erwartenden Kopfbewegungen voraus. Auf Basis der bisherigen Messungen muß das Modul auf Kopfpositionen in der Zukunft schließen.

Je nach Wahl des Filters in 4.2.3 kann dieses Modell direkt vom Filter selbst bezogen werden. So ergibt sich zum Beispiel bei der Verwendung des (erweiterten) KALMAN-Filters die Vorhersage direkt aus der filtereigenen Prädiktion.

4.3 Landmarkenentwurf

Die verwendeten Landmarken sind so zu wählen, daß sie möglichst gut mit dem System zusammenarbeiten. Das heißt insbesondere, daß sie von der Lokalisierung (4.2.1) zuverlässig erkannt und angepeilt werden können. Die Landmarkenidentifizierung (4.2.2) wiederum profitiert von einer geeigneten Kodierung der Marken.

Dieser Abschnitt behandelt einige Punkte, die beim Entwurf der künstlichen Landmarken beachtet werden müssen.

4.3.1 Mögliche Ausführungen

Die Marken müssen über Merkmale verfügen, die von der Lokalisierung leicht erkannt werden. Prinzipiell gibt es für künstliche optische Landmarken zwei Herangehensweisen:

Passive Landmarken Dabei handelt es sich um Marken, die lediglich an Form und Farbe zu erkennen sind. Sie werden durch das vorhandene Umgebungslicht beleuchtet.

Ein Spezialfall der passiven Landmarken sind sogenannte *Retro-Targets*, die Licht einer bestimmten Wellenlänge besonders gut reflektieren und von entsprechenden Lampen angestrahlt werden.

Aktive Landmarken Dazu zählen selbstleuchtende Marken, die meist in Zusammenhang mit einem Farbfilter vor der Kamera eingesetzt werden. Aktive Landmarken leuchten bevorzugt im Infraroten (wo auch die CCDs empfindlicher sind). Dadurch sind sie meistens besser zu erkennen als passive Landmarken.

Aktive Landmarken können zusätzlich *zeitlich kodiert* sein. Das heißt, daß sie zeitabhängig ein- und ausgeschaltet werden, was in Synchronisation mit der Kamera zu einer besonders einfachen Detektion führt.

In jedem Fall müssen die Marken gut von anderen Objekten in der Umgebung unterscheidbar sein.

Hat man sich entschieden, ob passive oder akive Landmarken Verwendung finden, stellt sich die Frage nach der geeigneten *Kodierung*. Die Kodierung dient dazu, die Landmarken in *Typen* zu unterteilen. Im günstigsten Falle findet man eine Kodierung, bei der jede einzelne Landmarke ihren eigenen Typ erhält und man schon am Typ erkennt, *welche* Landmarke man vor sich hat (siehe Abschnitt 4.3.3).

Für die Kodierung stehen unterschiedliche Techniken zur Verfügung. Passive Landmarken können nur nach Form und Farbe, respektive Muster, unterschieden werden. Man setzt daher Techniken wie Farbcodes oder Barcode-ähnliche Kodierungen ein.

Bei aktiven Landmarken ergibt sich noch die Möglichkeit der *zeitlichen Typkodierung*. Dabei werden verschiedene Landmarken in unterschiedlichem Rhythmus getriggert. Für die zeitliche Typkodierung benötigt man im allgmeinen aber eine sehr schnelle Sensorik. Diese Form der Kodierung verwendet zum Beispiel das System aus [6], siehe Abschnitt 2.1.

Im folgenden sei noch offengelassen, welche Landmarkenausführung konkret zum Einsatz kommt. Die Anforderungen an das Landmarkendesign sollen erst allgemein diskutiert werden.

4.3.2 Landmarkendesign und Markenlokalisierung

Es soll nun genauer auf das Zusammenspiel zwischen dem Landmarkendesign und der rechnergestützten Markenlokalisierung eingegangen werden. In diesem Zusammenhang können drei Aspekte unterschieden werden.

Fehldetektion

Die Marken müssen so ausgelegt sein, daß möglichst wenig Fehldetektionen auftreten. Es gibt zwei Fälle der Fehldetektion:

Fehler erster Art Eine im Bild sichtbare Landmarke wird nicht erkannt.

Fehler zweiter Art Eine vermeintliche Landmarke wird erkannt, wo keine vorhanden ist.

Im allgemeinen kann das Auftreten der beiden Fehlertypen nicht vollständig verhindert werden. Aber auch eine Reduktion der Fehler kann nicht beliebig erfolgen. Tatsächlich existiert meistens eine gegenläufige Korrelation zwischen den beiden Fehlertypen: Versucht man, den Fehler erster Art zu minimieren, handelt man sich dabei ein vermehrtes Auftreten des zweiten Fehlertyps ein. Reduziert man durch Änderungen am System den Fehler zweiter Art, steigt die Anzahl Fehldetektionen erster Art. Je nach Anwendung ist also ein geeigneter Kompromiß zu finden.

In unserer Anwendung ist der Fall einer nicht erkannten (obwohl vorhandenen) Landmarke aufgrund von möglichen Verdeckungen in der Umwelt nicht zu vermeiden. Insofern muß das System robust gegen „fehlende“ Landmarken, also auch gegen Fehler erster Art sein.

Wenn man weiß, daß das System gegen Fehler erster Art robust sein muß, ist es naheliegend, diese Eigenschaft des Verfolgungssystems auszunutzen, und den Fehler zweiter Art so weit wie möglich zu senken. Der Anstieg des Fehlers erster Art ist dann nämlich eher unkritisch, solange noch genügend andere Landmarken erkannt wurden.

Das Landmarkendesign und der Detektionsalgorithmus sollten daher so aufeinander abgestimmt sein, daß zwar hin und wieder eine Landmarke „übersehen“ wird, eine erfolgte Detektion aber mit hoher Wahrscheinlichkeit korrekt ist.

Teilweise Verdeckung

Im Alltagseinsatz wird das System häufig mit (teilweisen) Verdeckungen der Landmarken zu kämpfen haben. Dabei ist es wünschenswert, daß auch nur partiell sichtbare Marken noch

erkannt werden, ohne die Häufigkeit des Fehlers zweiter Art wieder zu erhöhen (s.o.).

Doch nicht nur die Erkennung muß robust gegen Verdeckungen sein. Auch die Lokalisierung des *Referenzpunkts* der Landmarken (zum Beispiel der Mittelpunkt) sollte von einer teilweisen Verdeckung unberührt bleiben, um ein exakte Peilung zu garantieren.

Eine teilweise Verdeckung spielt natürlich nur bei räumlich ausgedehnten Marken eine Rolle. Insbesondere passive Landmarken benötigen eine gewisse Ausdehnung. Setzt man hingegen aktive Landmarken ein, so bieten sich zum Beispiel Leuchtdioden an, deren annähernd punktförmiges Bild entweder ganz oder gar nicht verdeckt sein wird. Hier muß der Fall der teilweisen Verdeckung nicht betrachtet werden.

Perspektivische Verkürzung

Gerade die Lokalisierung des Referenzpunkts kann durch die perspektivische Verkürzung[2] der Landmarkenbilder erschwert werden. Insbesondere fällt das Bild des Schwerpunkts einer Fläche nicht mehr mit dem Schwerpunkt ihres Bildes zusammen.

Räumlich ausgedehnte Marken müssen also so geformt sein, daß der Referenzpunkt jeweils aus anderen Merkmalen der Landmarke rekonstruiert werden kann. (Dies kann zum Beispiel durch den Schnitt zweier Geraden erfolgen.)

Ist der Referenzpunkt explizit markiert (oder ist die gesamte Marke punktförmig), so kann dieses Problem wieder vernachlässigt werden.

Beleuchtungsartefakte

Zeitlich und räumlich veränderliche Beleuchtungsbedingungen können bei passiven Landmarken zu Schwierigkeiten bei der Erkennung führen. Beleuchtungsartefakte betreffen in erster Linie Farbe, Kontrast und Helligkeit des Landmarkenbilds. Das erschwert nicht nur die Erkennung der Farbe als Landmarkenmerkmal. Scharfe Schatten können auch geometrische Artefakte hervorrufen.

Aber auch aktive Landmarken können von den Beleuchtungsbedingungen betroffen sein. Helles Umgebungslicht vermindert den zur Detektion nötigen Kontrast, und farbige Lichtquellen können als Landmarken erkannt werden. Insbesondere Infrarot-Marken sind schwer von Wärmequellen in der Umgebung zu unterscheiden. Flackerndes Licht verfälscht die Erscheinung zeitlich kodierter Landmarken.

Schließlich treten durch die Eigenschaften einer klassischen CCD-Kamera weitere Probleme auf: Die Kamera führt je nach Umgebungslicht einen *Weißabgleich* durch, um Farbverfälschungen durch die Beleuchtung zu kompensieren. Oft führt aber gerade dieser Kompensationsversuch dazu, daß sich Farben von einem Bild zum nächsten verändern, wenn zum Beispiel starke Lichtquellen mit hohem Blauanteil (so der Fall bei Neon-Röhren) ins Bild kommen.

Helle Lichtquellen können aber noch mehr anrichten: Um die Lichtquellen selbst herum kommt es zu „Übersprechern“ zwischen den einzelnen CCD-Pixeln, und übersteigt die Gesamthelligkeit einen gewissen Betrag, wird die Kamera geblendet, so daß nur noch weitgehend weiße Bilder aufgenommen werden.

All diese Aspekte müssen beim Entwurf geeigneter Landmarken brücksichtigt werden. Die Qualität eines Landmarkendesigns ist in jedem Fall vom Einsatzumfeld abhängig.

[2] Von perspektivischer Verkürzung wird normalerweise nur bei ebenen Projektionen der Umwelt gesprochen. Tatsächlich betrachte ich in dieser Arbeit aber keine ebenen Projektionen mehr, sondern arbeite ausschließlich im *Ray-space*. Dennoch gibt es auch hier ein Äquivalent der persp. Verk.: Kollineare Strecken gleicher Länge können unter unterschiedlichem Winkel erscheinen.

Es ist aber zulässig, die Problematik der Anschauung halber am Bild einer ebenen Projektion zu betrachten, da sie uns meistens vertrauter ist.

Anzahl der Markentypen m	Marken pro Typ k	Max. Anzahl Zuordnungen max $\lvert S\rvert$
1	24	$\sim 6 \cdot 10^{24}$
2	12	$\sim 2 \cdot 10^{17}$
3	8	$\sim 7 \cdot 10^{13}$
4	6	$\sim 3 \cdot 10^{11}$
6	4	191 102 976
8	3	1 679 616
12	2	4 096
24	1	1

Tabelle 4.1. Beispielhafte Aufwandsabschätzung für die Landmarkenidentifizierung bei 24 Marken. Der Aufwand steht in Abhängigkeit zu der Anzahl vorhandener Landmarkentypen m.

Letztendlich darf die Wahl der Landmarken aber nicht ohne den zugehörige Lokalisierungsalgorithmus bewertet werden.

4.3.3 Markenkodierung und die Markenidentifizierung

Ein Punkt ist für die Geschwindigkeit des Gesamtsystems besonders wichtig: Da der Aufwand für die Markenidentifizierung mit der Anzahl gleichartiger Landmarken extrem ansteigt, muß die Zahl der Marken mit gleichem Aussehen möglichst gering gehalten werden.

Um das zu verdeutlichen: Wird keine weitere Umweltinformation berücksichtigt, so sind bei der Markenidentifizierung sämtliche Permutationen der Marken, die zu den erkannten Landmarkentypen passen, als mögliche Lösung in Betrach zu ziehen. Befinden sich zum Beispiel von m unterschiedlichen Landmarkentypen jeweils k bekannte Marken im System und sind von jedem Typ jeweils v Marken sichtbar, so sind

$$|S| = \left[v! \binom{k}{v} \right]^m = \left[\frac{k!}{(k-v)!} \right]^m \tag{4.1}$$

Lösungen zu betrachten. $|S|$ wird maximal für $v = k$. Im schlimmsten Fall sind also

$$|S| = k!^m \tag{4.2}$$

Zuordnungen „Peilung $\longmapsto$ bekannte Landmarke“ zu untersuchen. Diese Abschätzung zeigt klar, daß von jedem Landmarkentyp nur sehr wenige Marken vorhanden sein dürfen, will man den Aufwand für die Markenidentifizierung nicht ins Uferlose treiben. — Die Komplexität eines „naiven“ Algorithmus liegt bei $O((n/m)!^m)$ für n Landmarken und m Markentypen.

Zur Veranschaulichung zeigt Tabelle 4.1 noch einmal die Abschätzungen von $|S|$ für ein Ensemble von 24 Landmarken bei variierendem m.

Kapitel 5

Implementierung

Die konkrete Umsetzung des Kopfverfolgers erfolgte entsprechend dem Entwurf aus Kapitel 4. Die Wahl der Mittel richtete sich dabei nach den Anforderungen an das System, soweit sie im Vorfeld bekannt waren.

5.1 Anforderungen an die Implementierung

5.1.1 Modularität

Der allgemeine Ansatz sollte sich auch im Programmentwurf widerspiegeln. Das heißt insbesondere, daß die einzelnen Komponenten, wie sie in 4.2 vorgestellt wurden, beliebig austauschbar sein müssen, ohne daß davon eine andere als die ausgewechselte Komponente betroffen ist.

Da damit zu rechnen ist, daß das Programm noch weiterentwickelt werden wird, sind umso mehr Wartbarkeit und gute Dokumentation gefordert.

5.1.2 Vorgesehene Hardware-Umgebung

Das Kopfverfolgungssystem mußte so dimensioniert werden, daß es in folgender Hardware-Umgebung Echtzeit erreicht:

Kamera

Die Panoramakammera soll während des Einsatzes vom Benutzer auf dem Kopf getragen werden und muß folglich kompakt gebaut sein. Ein Modell mit zwei parabolischen Spiegeln (siehe Abschnitt 3.3) eignet sich daher für diese Verwendung am besten.

Ursprünglich war geplant gewesen, die Kamera selbst zu bauen. Nachdem auch schon erste Kontakte mit einer Werkstatt geknüpft waren, die die Spiegel herstellen sollte, sind wir noch auf ein Angebot der Firma *CycloVision Technologies* gestoßen.

Diese Firma hatte bereits eine Panoramakamera im Programm, die unseren Vorstellungen bereits sehr gut entsprach. Wir verwarfen daher die Pläne zum Selbstbau und orderten die Kamera „ParaCamera™ S-360c“. Das Datenblatt findet sich in Anhang K.

Die CycloVision-Kamera liefert Bilder in der amerikanischen Fernsehnorm NTSC (siehe Abschnitt 3.4.2).

Anmerkung: Leider hatte CycloVision Lieferprobleme, weshalb die Kamera erst nach Abschluß der Arbeit in Karlsruhe eingetroffen ist. (Umstellungen in der Fertigung sollen die Verzögerung verursacht haben.) Aus diesem Grunde konnte der Kopfverfolger nur mit Hilfe des Kamera-Simulationssystems getestet werden.

Rechner

Bildverarbeitung und Positionsrekonstruktion werden unter Linux auf einem Wearable Computer laufen, der mit einem Intel Celeron, 800 MHz, und 128 MB Hauptspeicher ausgerüstet ist. Als Graphikkarte steht dem PC eine Karte mit dem GeForce-Chip der Firma nVidia zur Verfügung.

Diese Ressourcen teilt sich der Kopfverfolger noch mit der eigentlichen AR-Applikation, so daß er das System nicht vollständig auslasten darf.

Während der Diplomarbeit stand der Wearable Computer allerdings noch nicht zur Verfügung. Die Entwicklung fand daher auf einem dual Pentium III, 450 MHz, und einem Athlon 600 System statt. Alle Zeitmessungen beziehen sich auf letzteres System.

5.1.3 Betriebsystem

Bezüglich des Betriebsystems soll das Programm möglichst wenig Einschränkungen unterliegen. Unbegingt notwendig ist die Lauffähigkeit unter Linux, da der Wearable Computer mit diesem Betriebssystem konfiguriert sein wird. Wünschenswert ist eine Unterstützung der Win32-Plattform. Teile des Systems sollten auch unter IRIX64 laufen.

5.1.4 Wahl der Programmiersprache

Als Programmiersprache soll ANSI C++ und/oder Python eingesetzt werden.

5.1.5 Netzwerkumgebung

Der Kopfverfolger wird im Rahmen eines verteilten, CORBA-gestützten Systems am IAIM zum Einsatz kommen. Daraus leiten sich weitere Randbedingungen an den Objektentwurf ab:

- Alle Schnittstellen müssen netzwerkfähig ausgelegt werden. Das heißt insbesondere, daß sämtliche Datentypen, die zum Austausch gelangen, für den Export als CORBA-Objekte geeignet sein müssen.
- Der Netzwerkverkehr muß minimiert werden. Das heißt, daß die Gliederung des Programms so erfolgen sollte, daß zwischen den einzelnen Moduln möglichst wenig Datenverkehr stattfindet.
- Das System muß reentrant und unempfindlich gegenüber Nebenläufigkeiten sein.
- Alle Zeitreferenzen müssen sich auf einen im Netz vorhandenen Zeit-Server beziehen.

5.2 Entwurfsentscheidungen

Die Implementierung zur Diplomarbeit berücksichtigt alle Anforderungen aus 5.1. Allerdings wurde aus Zeitgründen vorerst auf die konkrete CORBA-Anbindung verzichtet.

Die Entwicklung erfolgte vollständig in ANSI C/C++ (egcs) unter Linux. Bei der Wahl der verwendeten Libraries wurde darauf geachtet, daß sie auf möglichst vielen Plattformen vorhanden sind.

So erfolgt die Visualisierung ausschließlich mit OpenGL; die Fenster- und Menüverwaltung wird an GLUT delegiert.

Komplexere numerische Aufgaben werden, sofern sie nicht „von Hand“ für die konkrete Aufgabe optimiert kodiert wurden, mit Hilfe der freien Bibliothek „GSL“ (***G**nu **S**cientific*

L**ibrary*) gelöst. Die Library ist auf allen gängigen Plattformen vorhanden und ist in der Lage, eine vorhandene BLAS-Installation (B**asic **L**inear **A**lgebra **S**ystem*) mitzubenutzen.

Darüberhinaus kamen zwei selbst entwickelte Bibliotheken zum Einsatz: libglutviewer stellt eine Hilfsklasse zur Navigation in der 3D-Umwelt zur Verfügung; libasgeom ist eine Library für Projektive Geometrie und den Umgang mit Quaternionen.

5.3 Kamerakalibrierung

Will man mit Hilfe der Panoramakamera Peilungen bestimmen, so müssen die Abbildungsparameter der Kamera natürlich bekannt sein. Die eingesetzte Kamera entspricht in ihren Abbildungseigenschaften dem allgemeinen Fall einer parabolischen Panoramakamera, wie er in Abschnitt 3.3 vorgestellt wurde.

Da ich in der Literatur nichts Brauchbares zur Kalibrierung dieses speziellen Typs einer Panoramakamera gefunden hatte, habe ich im Rahmen der Diplomarbeit ein eigenes Kalibrierungsverfahren bentwickelt:

In parabolischen Panoramakameras werden Strecken in der Umwelt auf Kreissegmente im Kamerabild abgebildet (Beweis siehe Anhang C). Aber nicht jedes mögliche Kreissegment kann Bild einer Geraden sein. Mein Kalibrierungs-Algorithmus macht sich diese Eigenschaft zunutze: Aus mindestens drei Kreissegmenten allgemeiner Lage, von denen bekannt ist, daß sie Bilder von Strecken in der Umgebung sind, kann auf die Geometrie des Spiegelparaboloids geschlossen werden.

Zur Zeit erfolgt dieser Kalibrierungsvorgang manuell. Im einzelnen sind folgende Schritte nötig:

- In einem Vektor-Zeichenprogramm (xfig) wird ein Kamerabild in den Hintergrund einer neuen Zeichnung gelegt.
- Anschließend müssen mindestens drei Bilder von Strecken in der Umwelt mit dem „Kreissegment durch drei Punkte“-Werkzeug nachgezogen werden. Gerade auf Innenaufnahmen sind durch die Kanten zwischen zwei Wänden, bzw. Wänden und Decke, meist genügend gerade Strecken vorhanden.

  Im Zeichenprogramm können die Stützstellen bei hoher Vergrößerung noch nachjustiert werden.
- Die Rahmenapplikation zur Diplomarbeit ist in der Lage, die so erstellte xfig-Datei einzulesen und daraus die Kamerakalibrierung abzuleiten.

Es ist sinnvoll, mehr als drei Kreissegmente zu markieren, da so durch Ausgleichsrechnung eine höhere Genauigkeit erreicht wird. Die eigentliche Kalibrierung ist in Anhang K beschrieben.

Abbildung 5.3 zeigt Eingabe und Ergebnis des Kalibrierungsvorgangs anhand eines Kamerabilds, das mit der Einzelbildpanoramakamera aus Abschnitt 3.3 aufgenommen wurde.

5.4 Kamerasimulation

Eine Bewertung des Verfahrens zur Positionsrekonstruktion wird vereinfacht und in manchen Fällen überhaupt erst möglich, wenn sämtliche Parameter der abgebildeten Umwelt genauestens bekannt sind. (Siehe auch Abschnitt 5.4.4.)

Aus diesem Grunde, und vor allem auch, weil wie erwähnt die NTSC-Panoramakamera bis zum Schluß noch nicht verfügbar war, wurde zunächst eine Kamerasimulation entwickelt.

Das Simulationssystem ist in der Lage, dem Verfolgungssystem Bilder einer gedachten Kamera in einer virtuellen Umgebung zu liefern. Auf diese Weise wurde es möglich, den

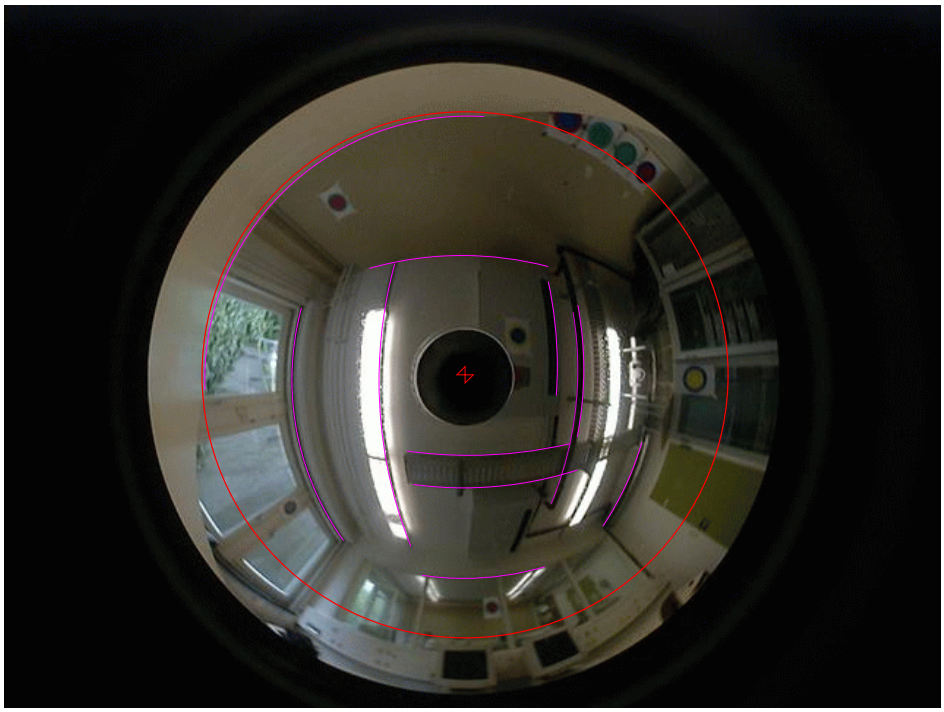

Abbildung 5.1. Kamerabild mit Kalibrierungsmarkierungen. Magentafarben sind die Kreissegmente nachgezogen, die in den Kalibrierungsvorgang eingegangen sind. Rot eingezeichnet sind der daraus abgeleitete Scheitelpunkt des Paraboloids und der Horizontkreis. Letzterer beschreibt die Kalibrierung vollständig.

Kopfverfolger in unterschiedlichen Szenarien zu erproben. Nicht zuletzt durch die Simulation ist die schnelle Fertigstellung des Kopfverfolgers erst möglich geworden.

5.4.1 Virtuelle Umgebung

Szenenaufbau

Die virtuelle Umgebung wird durch einen primitiven Szenen-Graphen beschrieben. Für die Evaluation des Verfolgungssystems war es ausreichend, die „Welt“ aus einfarbigen Quadern zusammenzusetzen. Zur Laufzeit wird der Szenengraph um die Landmarken ergänzt, wie sie in der Landmarkendatei gefunden wurden (siehe Anhang J.1). Auf diese Weise lassen sich schon einfache Räume mit unterschiedlichen Landmarkenkonfigurationen simulieren.

Eine ausführlichere Beschreibung des Szenen-Graphen findet sich in Anhang H.

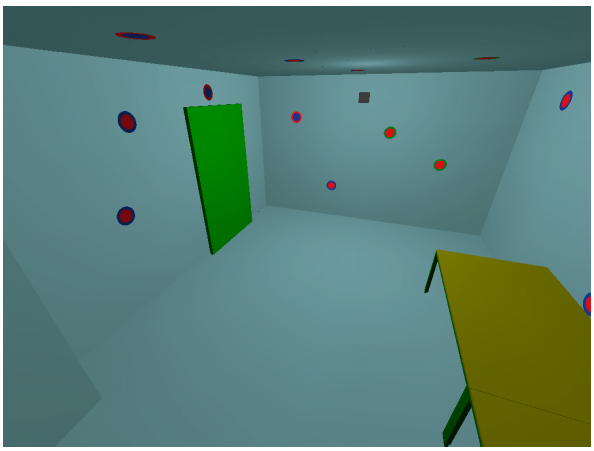

Abbildung 5.2: Bild der virtuellen Testumgebung.

Visualisierung

Die Visualisierung des fertigen Szenen-Graphen erfolgt in OpenGL. Dabei können Abbildungsparameter und Lichtquellen aus dem OpenGL-Kontext übernommen oder frei definiert werden. Abbildung 5.2 zeigt eine so gerenderte Testumgebung.

Zur Zeit ist OpenGL die einzige unterstützte Graphikausgabe. Der Scene-Viewer ist den-

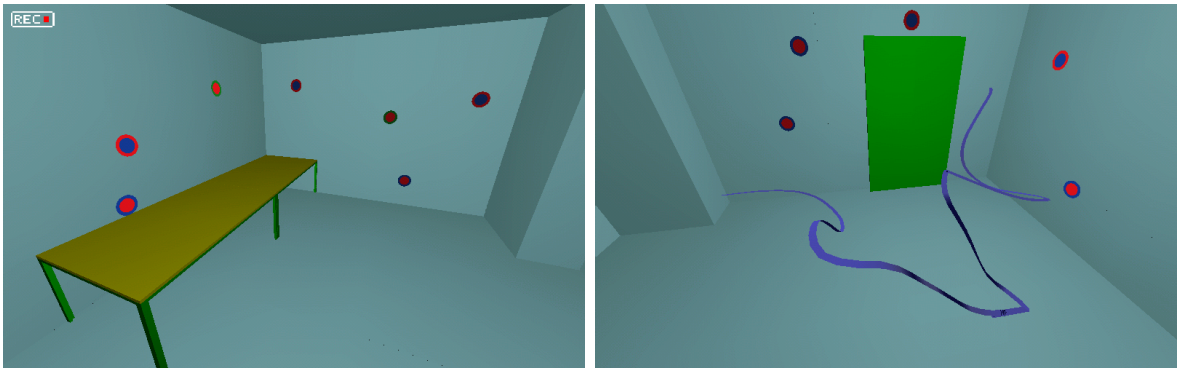

Abbildung 5.3. Aufnahme einer Trajektorie. Während der Benutzer sich durch die Simulation bewegt, wird die dabei durchlaufene Bahn aufgezeichnet (Bild links). Anschließend kann die Trajektorie in die simulierte Umgebung eingeblendet werden (Bild rechts).

noch unabhängig von der verwendeten Graphik-Library geschrieben.

Mehr zur Szenenvisualisierung in Anhang H.

Interaktiver „Walk-Through“

Die Rahmenapplikation erlaubt es dem Benutzer, sich in der virtuellen Umgebung frei zu bewegen. Die Steuerung unterstützt „Flüge“ durch den Raum. Auf diese Weise kann der Raum genauer erkundet werden.

Die Steuerung ist in Anhang I beschrieben.

5.4.2 Trajektorienaufzeichnung

Der Kopfverfolger kann nur an einem dynamischen System aussagekräftig untersucht werden. Es muß daher möglich sein, die simulierte Kamera entlang vorgegebener Bahnen zu bewegen.

Der Einfachheit halber können solche Bahnen direkt in der Rahmenapplikation eingegeben werden: Der Benutzer steuert, wie oben beschrieben, durch den Raum, während die zurückgelegte Trajektorie aufgezeichnet wird. Um Mausruckeln zu eliminieren, wird die fertig aufgenommene Trajektorie geglättet und in einer Datei gesichert.

In Hinblick auf eine realistische Simulation ist die Glättung zulässig, da auch ein menschlicher Kopf aufgrund seiner natürlichen Trägheit nicht beliebig stark beschleunigt werden kann. Die Glättung erfolgt durch Gauß-Filterung. Die Positionen werden im Euklidischen Raum gefiltert, die Rotationen auf der Einheitskugel des Quaternionenraums.

Abbildung 5.3 zeigt den Aufnahmevorgang und die fertige Trajektorie.

Die Trajektorie ist eine Folge von Tripeln (*Zeitpunkt*, *Position*, *Orientierung*). Ihre Wiedergabe — sei es für die Kamerasimulation, für das Abspielen der Bewegung innerhalb der interaktiven Ansicht oder einfach nur zur Visualisierung wie in Bild 5.3 — erfolgt interpoliert: Existiert für einen Zeitpunkt kein „Meßwert“, so werden Position und Rotation aus den beiden benachbarten Messungen linearkombiniert (Rotationen wieder im Quaternionenraum).

Die Wiedergabe wird durch einen frei wählbaren Zeitgeber gesteuert. Sie kann also zum Beispiel wahlweise in „Echtzeit“ erfolgen oder in „Frame-Time“, was heißt, daß die Bilder der Kamerasimulation aufeinander unabhängig von der Berechnungsdauer folgen, als wären sie im Abstand von (bei unserer NTSC-Kamera) 1/30 Sekunde aufgenommen worden.

5.4.3 Simulation der Kamerabilder

Ist die virtuelle Umgebung definiert, können darin Kamerabilder aus beliebigen Positionen simuliert werden. Die Simulation verwendet dabei die zuvor bestimmten Kalibrierungspara-

meter einer gegebenen Panoramakamera.

Für die Erstellung des simulierten Kamerabilds könnte ein naiver Algorithmus verwendet werden (der in seiner Allgemeinheit für jeden Kameratyp geeignet wäre):

Algorithmus 5.1: Berechnung eines Kamerabilds I

Für alle Pixel im zu berechnenden Kamerabild

Bestimme über die Kalibrierung den Sehstrahl, der mit dem entsprechenden Pixel korrespondiert.

Schicke diesen Strahl, ausgehend von der aktuellen Kameraposition, in die Szene und bestimme das erste getroffene Objekt.

Die Farbe, die das Objekt (unter Berücksichtigung aller Beleuchtungseffekte) an der Auftreffstelle hat, bestimmt die Farbe des Pixels im Kamerabild.

Diese Vorgehensweise entspräche der eines *Ray-Tracers*. Stände dem System ein Ray-Tracer zur Verfügung, der direkte Strahlanfragen für die Simulationsumgebung beantworten könnte, wäre dessen Einsatz auch die erste Wahl, um hochqualitative Kamerabilder zu erzeugen.

Nun sind Ray-Tracer, verglichen mit Rendering-Methoden, wie sie zum Beispiel von üblicher Graphik-Hardware verwendet werden, verhältnismäßig langsam. Für die Kamerasimulation wäre es aber wünschenswert, ein möglichst schnelles Verfahren zur Hand zu haben, um in akzeptabler Zeit umfangreiche Bildfolgen erzeugen zu können.

Außerdem existieren im Programm bereits Teile, die via OpenGL Ansichten der Simulationsumgebung rendern. Es ist also naheliegend, diese Visualisierungsroutinen auch für die Erzeugung der Kamerabilder zu verwenden.

Kubische Umgebungskarte

Hierfür bietet sich eine Technik an, die von der Hardware-unterstützten Darstellung verspiegelter Flächen bekannt ist: Der Einsatz einer kubischen Umgebungskarte (*cubic environment map*). Sie ist nicht nur für die schnelle Berechnung von Spiegelungen, sondern auch für die Lösung unseres konkreten Ray-Tracing-Problems geeignet.

Eine Umgebungskarte kartiert zu einem Referenzpunkt im Raum alle von ihm ausgehenden Sehstrahlen. Sie hält für jeden dieser Strahlen die Farbe des Objekts, auf das er trifft. Sie enthält also insbesondere auch alle Sehstrahlen, die wir für die Simulation des Kamerabildes benötigen.

Um mit endlichem Speicher auszukommen, beschränkt man die Karte allerdings auf eine Auswahl repräsentativer Sehstrahlen. Die unterschiedlichen Techniken des *environment mappings* unterscheiden sich in der Wahl dieser Repräsentanten. Die jeweiligen Objektfarben werden in einem Pixelbild abgelegt.

Eine einfache, weit verbreitete Spielart ist die *sphärische Umgebungskarte* (Abb. 5.4 a)). Sie besteht aus dem Pixelbild einer perfekt verspiegelten Kugel mit infinitesimalem Radius unter orthogonaler Projektion. Liegt die Umgebung eines Objekts als sphärische Umgebungskarte vor, läßt sich das Objekt mit verhältnismäßig geringem Aufwand mit verspiegelter Oberfläche darstellen (Abb. 5.4 b)).

Der Hauptnachteil dieser Methode liegt darin, daß die sphärische Umgebungskarte für manche Blickrichtungen eine ungünstige Parametrisierung darstellt (siehe Abb. 5.4 c)). Zudem wäre in unserem Fall für die Erstellung der Karte wieder das Ray-Tracing-Problem zu lösen, womit nichts gewonnen wäre.

(a) (b) (c)

Abbildung 5.4. Beispiel einer sphärischen Umgebungskarte und ein verspiegeltes Objekt. (a) Jeder Sehstrahl, der vom Mittelpunkt der verspiegelten Kugel ausgeht, ist in einem Punkt des kreisförmigen Bilds der Kugel wiederzufinden. Lediglich der Sehstrahl, der in der optischen Achse des Betrachters liegt, wird auf den Umfang des Kreises abgebildet. (b) Verspiegelte Teekanne, deren Bild mit Hilfe der Umgebungskarte berechnet wurde. (c) Geringe Bildqualtität in der Nähe der Singularität.

Bezeichnung	*Anzahl Look-ups pro Pixel*	*Effektive Anzahl Look-ups*	*Gauß-Filterung*
NearestNeighbour	1	1	nein
Super2 (iso)	2	2	nein
Super2 (aniso)	2	2	nein
Super4	4	4	nein
Gauss4	4	2	ja
Gauss9	9	4	ja

Tabelle 5.1. Mögliche Super-Sampling Methoden bei der Kamerabilderzeugung. Die Gauß-gefilterten Varianten beziehen jeweils Abtastpunkte der umliegenden Zielpixel mit ein, so daß einige Quellpixel mehrfach abgetastet werden. Die effektive Anzahl Look-ups pro Pixel für das gesamte Bild wird damit kleiner als die Anzahl Pixel, die jeweils mit der Gauß-Maske gefaltet werden. Somit wird das Quellbild zum Beispiel bei *Gauss9* genauso fein abgetastet wie bei *Super4*. Dafür sind aber wesentlich weniger Aliasing-Artefakte zu beobachten.

Demgegenüber läßt sich eine *kubische Umgebungskarte* wesentlich einfacher erstellen: Sie besteht aus sechs perspektivischen Projektionen der Umgebung, deren Projektionsflächen sich zu einem Würfel ergänzen. Ihr Brennpunkt liegt jeweils im Referenzpunkt, und perspektivische Projektionen lassen sich wieder bequem von gängiger Hardware rendern, weshalb wir hierfür wieder den OpenGL-Renderer der Simulationsumgebung verwenden können.

Als Zwischenschritt der Kamerabilderzeugung wird also eine kubische Umgebungskarte erstellt. Abbildung 5.5 zeigt eine solche Karte.

Verzerrung

Tatsächlich ließe sich das Bild der Panoramakamera jetzt bereits erzeugen, indem man die 3D-Hardware anweist, unter orthogonaler Projektion ein Paraboloid zu zeichnen, das mit der eben erstellten *environment map* überzogen ist. (Denn nichts anderes „sieht“ der CCD-Chip in der Panoramakamera: Ein verspiegeltes Paraboloid in Parallelprojektion.) Zum einen unterstützt aber nicht jede OpenGL-Implementierung das technisch aufwendigere *cubic environment mapping*, zum anderen wären beim derzeitigen Stand der OpenGL-API zwei weitere Pixel-Transfers zwischen Hauptspeicher und Graphikkarte notwendig, um das fertige Kamerabild zu erhalten. Insbesondere auf PC-Hardware sind solche Pixel-Transfers noch relativ teuer.

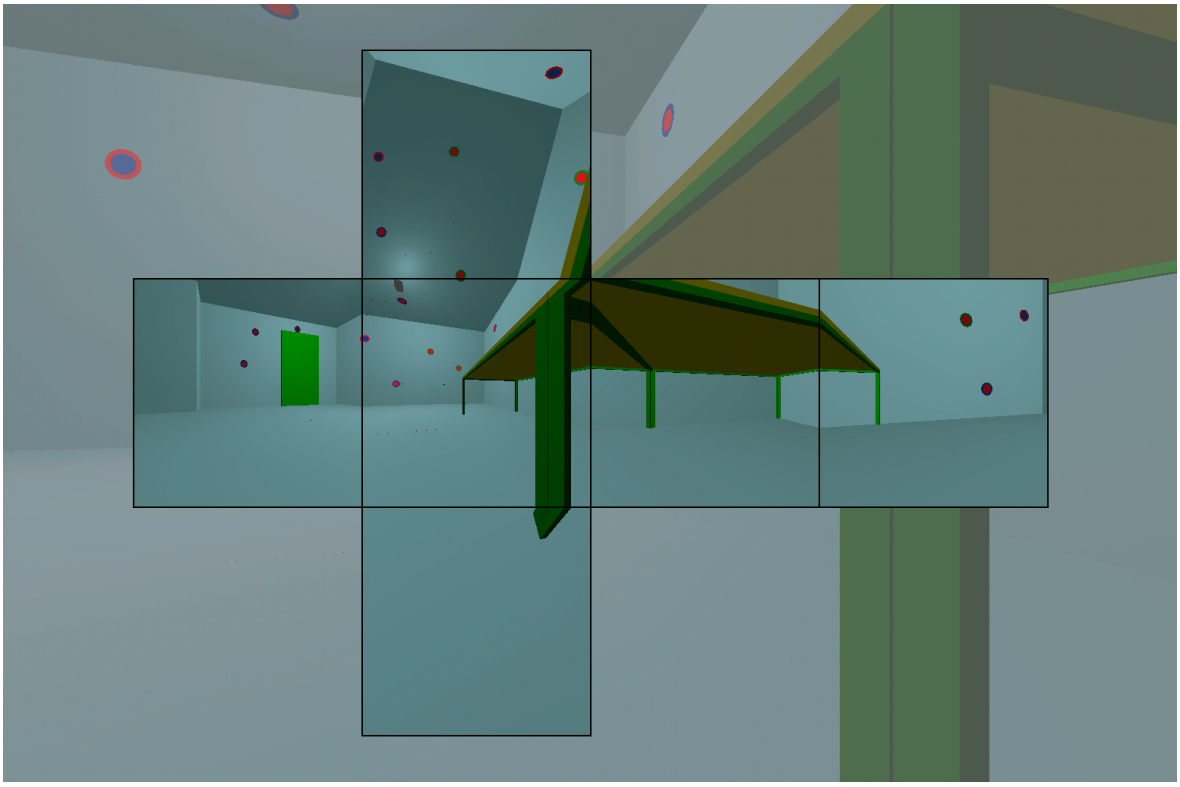

Abbildung 5.5. Kubische Umgebungskarte für die Kamerasimulation. Die einzelnen Projektionen sind zur Veranschaulichung auf einem abgewickelten Kubus dargestellt. Im Zentrum befindet sich die Front-Ansicht — die Sicht in Blickrichtung des Benutzers.

Es hat sich daher als schneller erwiesen, die Abbildung der Umgebungskarte auf das Kamerabild „von Hand“ durchzuführen. Zu diesem Zweck muß Algorithmus 5.1 modifiziert werden, und man erhält:

Algorithmus 5.2: Berechnung eines Kamerabilds II

Erstelle kubische Umgebungskarte.

Für alle Pixel im zu berechnenden Kamerabild

Bestimme über die Kalibrierung den Sehstrahl, der mit dem entsprechenden Pixel korrespondiert.

Beziehe die Pixelfarbe zu diesem Sehstrahl aus der Umgebungskarte.

Der Vorgang, die Pixelfarbe zu einem Sehstrahl aus der Umgebungskarte zu beziehen, sollte nach Möglichkeit interpoliert und/oder per Super-sampling stattfinden. Die gegenwärtige Implementierung verzichtet auf Interpolation, unterstützt aber sechs Arten von Super-sampling (siehe Tabelle 5.1). In Abbildung 5.6 ist ein simuliertes Kamerabild zu sehen, das mit dieser Methode berechnet wurde.

Kamm-Artefakte

Bei Fernsehkameras treten durch das Zeilensprungverfahren an bewegten Objekten sogenannte *Kamm-Artefakte* auf. Dies rührt daher, daß die Bildzeilen 1, 3, 5, ... eines Bildes zu einem

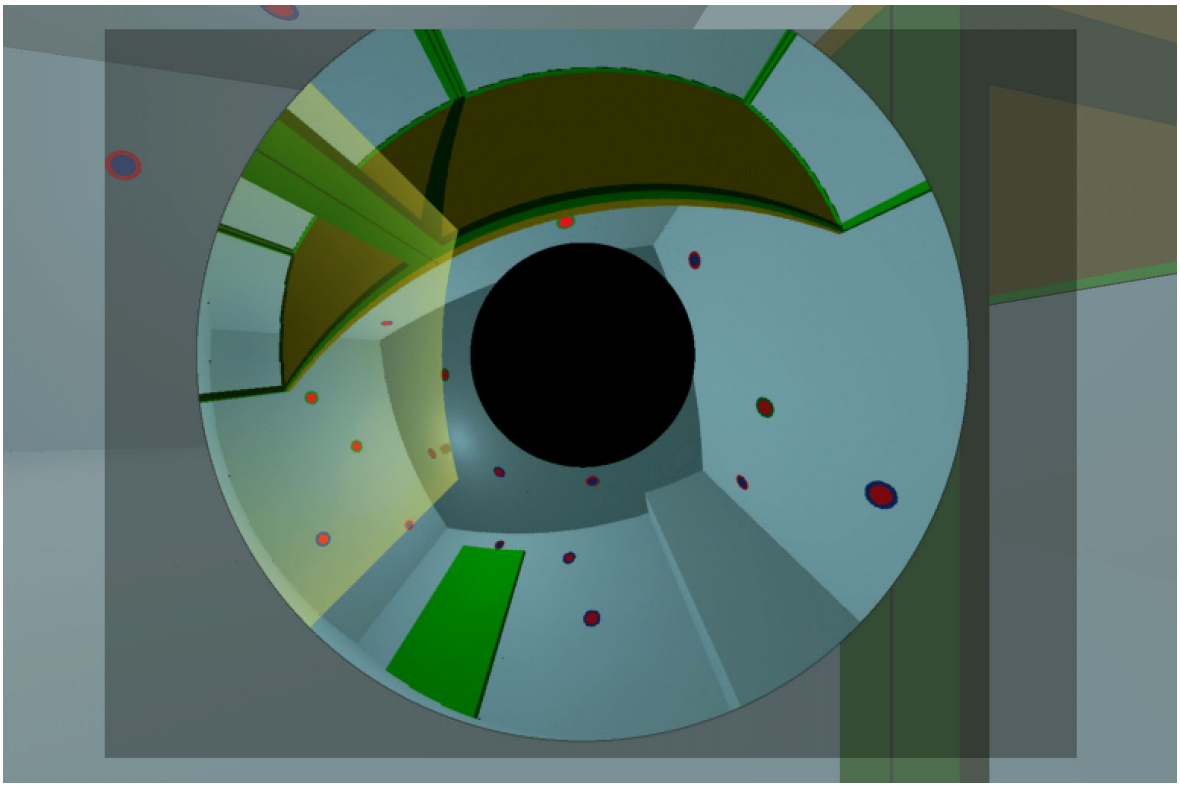

Abbildung 5.6. Simuliertes Kamerabild. Das Bild ist zusammengesetzt aus den verzerrten Projektionen der kubischen Umgebungskarte. Das Abbild der Front-Ansicht ist zur Orientierung gelb eingefärbt.

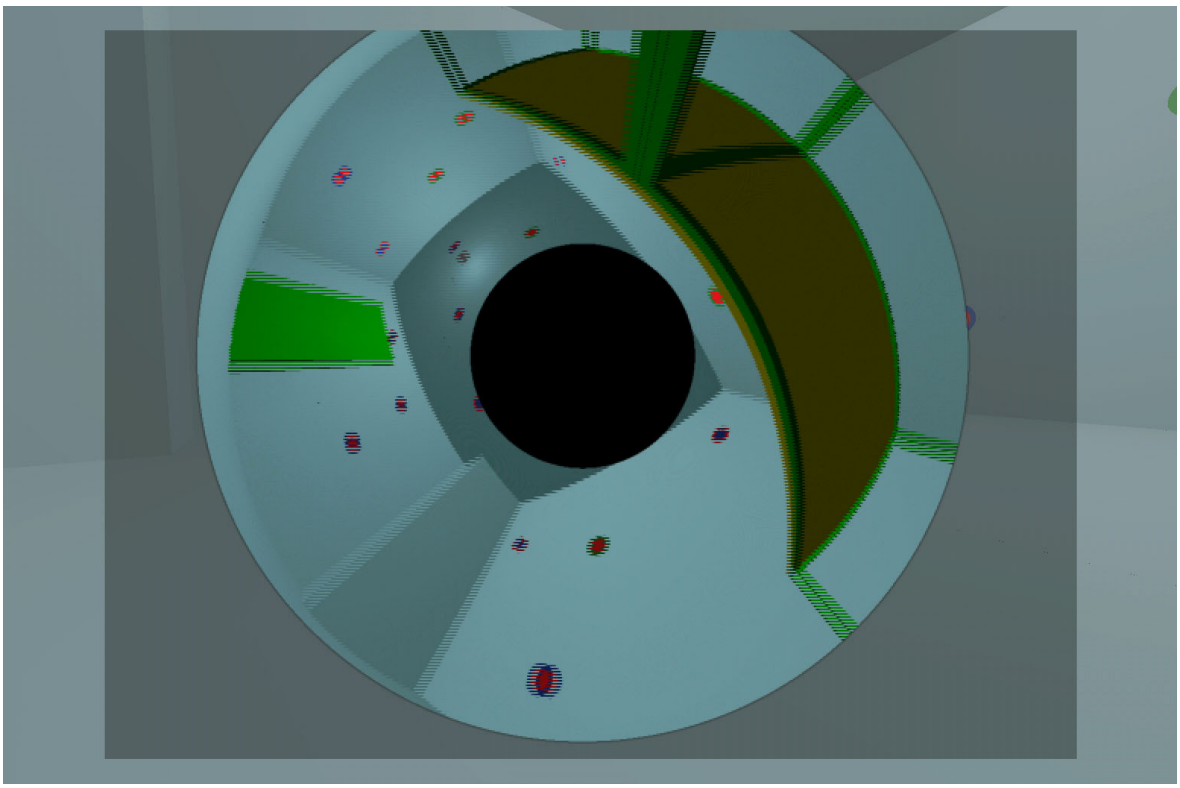

Abbildung 5.7. Simuliertes Kamerabild bei bewegter Kamera. Die Zeitverschiebung zwischen der Aufnahme der gerad- und ungeradzahligen Bildzeilen führt zu Kamm-Artefakten.

anderen Zeitpunkt aufgenommen worden sind als die Zeilen 2, 4, 6, ... Dieser Effekt erschwert die Bildverarbeitung enorm. Leider sind Kameras, die diesen Effekt nicht aufweisen (sog. *Progressive-Scan-Kameras)*, um ein Vielfaches teurer als handelsübliche PAL- oder NTSC-Kameras. Aus diesem Grunde sind sie selten anzutreffen. Insbesondere die Panoramakamera, die für unsere Versuche bestellt worden war, weist als NTSC-Kamera Kamm-Artefakte auf.

Es war deshalb unerläßlich, diese Artefakte in der Kamerasimulation nachzubilden. Dies wird dadurch erreicht, daß für zwei, um eine halbe Bildwiederholdauer verschobene Zeitpunkte die Umgebungskarten berechnet und auf die ungerad-, bzw. geradzahligen Bildzeilen abgebildet werden. Abbildung 5.7 zeigt ein simuliertes Kamerabild mit Kamm-Artefakten.

5.4.4 Grenzen und Möglichkeiten der Simulation

Natürlich wird der Realismus der simulierten Daten nie an den realer Kameradaten heranreichen. Deshalb sind Ergebnisse, die mit der Simulation erreicht wurden, immer mit Vorsicht zu interpretieren. Insbesondere unterliegt die Aussagekraft der Simulation folgenden Einschränkungen:

- Die virtuelle Umgebung enthält nicht die gleiche Vielfalt an Farben, Lichteffekten und anderen Störeinflüssen, wie sie in der Realität anzutreffen ist. Robustheit und Leistungsfähigkeit der Detektions- und Lokalisierungsalgorithmen müssen deshalb immer wieder an realen Kameradaten getestet werden.

- Die geometrische Genauigkeit der simulierten Daten wird von keiner Kamera erreicht werden, mithin wird die Präzision des Kopfverfolgers in der Realität geringer sein als in der Simulation.

- Elektronische Kameraartefakte wie Übersprechen zwischen benachbarten CCD-Pixeln, sich ändernder Weißabgleich und Übersteuerung treten in der Simulation gar nicht auf, können in ihrer Wirkung also noch nicht abgeschätzt werden.

Dennoch liefert die Simulation wertvolle Hinweise, zumal gilt: Ein Verfahren, daß in der Simulation nicht funktioniert, wird dies im realen Einsatz erst recht nicht tun. Einen besonderen Mehrwert liefert die Simulation durch die neuen Möglichkeiten, die sie mit sich bringt:

- Unterschiedliche Räume mit beliebigem Grundriß und unterschiedlicher Verdeckungssituation lassen sich austesten.

- Landmarkentypen und -anordnungen können rasch gewechselt werden. Der zeitaufwendige Teil, die aufgeklebten Marken zu vermessen, entfällt, wenn man eine alternative Markenkonfiguration betrachten möchte.

- Neue Trajektorien sind schnell aufgenommen und können "on the fly" simuliert werden. Gleichzeitig kann eine Kamerafahrt beliebig oft reproduziert werden, um die Ergebnisse verschiedener Verfahren im selben Kontext betrachten zu können.

- Der Einfluß von Kameratyp und -parametern auf die Qualität der Kopfverfolgung kann untersucht werden. Es ist ohne weiteres möglich, eine hochauflösende Hochgeschwindigkeitskamera zu simulieren, um herauszufinden, ob deren Anschaffung sich überhaupt lohnen würde.

- Auch die verfügbare Rechenzeit kann beliebig erhöht werden, indem die Simulation verlangsamt wird. Gerade weil die durchschnittliche Leistung der Hardware allgemein steigt, ist es interessant zu untersuchen, ob ein schnellerer Rechner bessere Ergebnisse liefern würde.

- Die Aufnahmepositionen der Kamera sind exakt bekannt und können mit den rekonstruierten verglichen werden. In der Realität sind Position und Orientierung der Kamera nur mit gewissem Aufwand zu bestimmen.

- Während der Entwicklung können Komponenten des Kopfverfolgers durch „allwissende“ Platzhalter ersetzt werden, da das Verfolgungssystem prinzipiell jede nötige Information bei der Simulation direkt abgreifen kann. Dadurch kann die Leistungsfähigkeit einzelner Komponenten des Verfolgers gezielt untersucht werden. Einflüsse anderer Komponenten werden minimiert.

5.5 Der Verfolger

Bei der Umsetzung des Verfolgers wurden zunächst nur die elementaren Komponenten implementiert. Auf Filterung und Multi-Sensor-Fusion wurde verzichtet.

Eine Filterung der Tracking-Daten ist in der Applikation zwar vorgesehen, wurde aber im Rahmen der Diplomarbeit noch nicht eingesetzt, da die Filterung eine objektive Bewertung des Rekonstruktionsalgorithmus eher behindert hätte. Das Thema der Multi-Sensor-Fusion wurde in dieser Arbeit überhaupt nicht bearbeitet.

5.5.1 Landmarken

Wie in Abschnitt 4.3 ausführlich diskutiert, spielt der Landmarkenentwurf für das Verfolgungssystem eine wichtige Rolle. Die Wahl fiel schließlich auf ein Ensemble passiver Landmarken gleicher Form und unterschiedlicher Farbkodierung. Die Marken werden mit einem Farbdrucker auf DIN A4 Papier gedruckt und an Wände und Decke des Raums geheftet, in dem der Kopfverfolger zum Einsatz kommen soll.

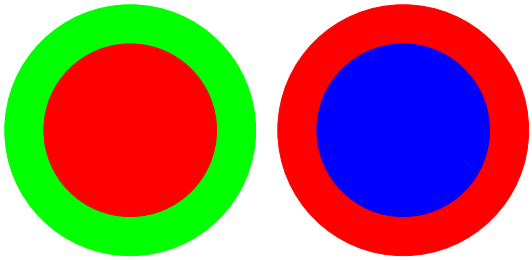

Abbildung 5.8: Beispiele des eingesetzten Landmarkentyps.

Abbildung 5.9: Landmarke mit zwei konzentrischen Ringen.

Jede Marke besteht aus einer einfarbigen Kreisscheibe, die von einem oder mehreren konzentrischen Farbringen umgeben ist. Benachbarte Ringe müssen sich natürlich in der Farbe unterscheiden. Referenzpunkt für die Peilungen ist der Markenmittelpunkt.

Meine bisherigen Experimente arbeiteten ausschließlich mit einfach umrandeten Landmarken, wie sie in Abbildung 5.8 zu sehen sind; die innere Kreisscheibe hat einen Durchmesser von 13,0 cm, die gesamte Marke ist 18,8 cm hoch. Ein Ergebnis der Arbeit war aber, daß aufgrund der Identifikationsproblematik (4.3.3) Marken mit mehreren Ringen verwendet werden müssen, um die Anzahl möglicher Farbkodierungen zu erhöhen (Abbildung 5.9).

Von dieser Wahl des Markentyps verspreche ich mir folgende Vorteile:

- Passive Marken, die auf Papier gedruckt werden, sind günstig in der Herstellung und schnell produziert. Es ist daher möglich, beim mobilen Einsatz flexibel auf die farblichen Bedingungen vor Ort zu reagieren. Soll das System in einem grün gestrichenen Raum zum Einsatz kommen, wird man sich Landmarken ausdrucken, die diese Farbe umgehen.

- Die Marken sind rotationssymmetrisch, haben also unabhängig von der Betrachtungsrichtung ähnliche Abbilder im Kamerabild. Kreisscheiben werden von der parabolischen

Verzerrung des Spiegels auf Pseudo-Ellipsen abgebildet, deren Proportionen nicht zu stark von denen einer planaren Projektion der Kreisscheiben abweichen. Dies erleichtert die Detektion.

- Die Kodierung der Marken, die Abfolge der Farbringe, ist auch bei sehr kleinen Markenbildern, wie sie aufgrund des großen Öffnungswinkels der Kamera schnell entstehen, noch ablesbar. Bar-Codes oder ähnliche Techniken wären bei Landmarken, die nur noch wenige Pixel im Kamerabild einnehmen, nicht mehr auszuwerten gewesen.

Wie schon angedeutet, sollte die Farbkodierung voll ausgeschöpft werden, um die Markenidentifizierung zu erleichtern. Für den vorgestellten Markentyp gilt allgemein: Für Landmarken mit n konzentrischen Ringen um die zentrale Kreisscheibe und k verfügbaren Farben lassen sich

$$m = k \cdot (k-1)^n \tag{5.1}$$

unterschiedliche Kodierungen erzeugen. Tabelle 5.2 zeigt die Werte von m für unterschiedliche Belegungen von n und k. Die Zahlen lassen hoffen, daß es möglich sein wird, für jede Marke einen eigenen Kode festzulegen. Allerdings sei gesagt, daß es schon schwierig wird, vier bis fünf gut unterscheidbare Farben zu finden, die in der Umgebung nicht zu oft vorkommen. Realistisch sind eher die Werte für $k = 3$.

$n \backslash k$	*2*	*3*	*4*	*5*
1	2	6	12	20
2	2	12	36	80
3	2	24	108	320

Tabelle 5.2: Anzahl Kodierungsmöglichkeiten der Landmarken, je nach Zahl der verfügbaren Farben k und Farbringe n.

Will man die Anzahl der Kodierungsmöglichkeiten noch weiter erhöhen, bleibt also nur, die Zahl Ringe zu vergrößern. Es ist aber fraglich, ob Landmarken mit mehr als drei Ringen noch zuverlässig erkannt werden können, da die Ringe bei gleicher Landmarkengröße dann verhältnismäßig schmal ausfallen.

Zumindest für kleine Räume reicht es aber aus, etwa 20 Markentypen unterscheiden zu können. Für größere Szenrarien müssen neue Landmarken entworfen werden.

5.5.2 Landmarkenlokalisierung

Das Modul zur Landmarkenlokalisierung ist stark an die konkrete Ausführung der Landmarken angepaßt. Ausgehend vom Kamerabild werden folgende Schritte ausgeführt:

Farbklassifizierung

Anfangs werden alle Pixel des Kamerabilds gesammelt, die möglicherweise zu einer Landmarke gehören könnten. Dazu wird jedes Pixel auf seine Farbe hin untersucht. Verwenden die Marken zum Beispiel die Farben Rot, Grün und Blau, so wird jedes Kamerapixel einer der Klassen schwarz, weiß, rot, grün, blau und unbekannt zugeordnet. (schwarz und weiß sind zwar für die Markendetektion nicht notwendig, können aber zum Beispiel verwendet werden, um Veränderungen des Farbraums durch den automatischen Weißabgleich zu erkennen.)

Die Farberkennung erfolgt dabei weitestgehend unabhängig von der Beleuchtung. Aufgrund der hohen Zahl zu klassifizierender Pixel, mußte die Farrbklassifizierung stark auf Geschwindigkeit optimiert werden, um die Echtzeitfähigkeit des Programms aufrechtzuerhalten. Details zum Klassifizierungsalgorithmus stehen in Anhang F.

Zusammenhangskomponenten

Im nächsten Schritt werden Zusammenhangskomponenten unter den Pixeln einer Farbklasse gesucht. Für das Auffinden zusammenhängender Pixelgruppen, sogenannter *Blob*s, existiert

bereits eine große Zahl von Algorithmen. (Exemplarisch seien [21, 22] genannt; weitere Verfahren werden in [23] vorgestellt.)

Durch Störeinflüsse wie Kamerarauschen oder teilweise Verdeckung werden die klassifizierten Landmarkenbilder aber mit ziemlicher Sicherheit „Löcher“ aufweisen. Das kann so weit führen, daß Pixel, die zu ein und derselben Landmarke gehören, in disjunkten Zusammenhangskomponenten einer Farbklasse landen. Um dieses „Auseinanderreißen“ der Zusammenhangskomponenten zu verhindern, führt man in der Bildverarbeitung üblicherweise den morphologischen *Schließen*-Operator auf dem klassifizierten Bild durch.

Eine Menge zu schließen heißt, sie erst zu *dilatieren*, um sie dann unter Beibehaltung des strukturierenden Elements zu *erodieren*. Dieses Vorgehen wird zusammen mit anderen morphologischen Operatoren in [23] vorgestellt.

Diese im allgemeinen übliche Vorgehensweise hat nur einen entscheidenden Nachteil: Der *Schließen*-Operator muß global für das gesamte Bild parametrisiert werden: Die maximale Größe der Löcher und Fehlstellen, die der Operator zu schließen vermag, ist für das ganze Bild gleich. Nun sind die Größenunterschiede der Landmarkenbilder durch den großen Öffnungswinkel der Kamera sehr hoch: Während die eine Landmarke eine Ausdehnung von 100–200 Pixeln haben kann, wird eine andere nur auf eine Handvoll Bildpunkte abgebildet. Zwar werden Löcher, die durch Kamerarauschen entstanden sind, in ihrer Dichte auf allen Landmarken ähnlich ausfallen. Die Ausdehnung von Artefakten, die durch starke Glanzlichter oder teilweise Verdeckung entstehen, schwankt aber mit der Landmarkengröße. Aus diesem Grunde wird sich für den *Schließen*-Operator keine Parametrisierung finden lassen, die für alle Landmarkenbilder gleichermaßen geeignet ist.

Daher habe ich einen eigenen Algorithmus entwickelt, um die Zusammenhangskomponenten zu ermitteln. Der Algorithmus überbrückt Löcher in den Pixelkomponenten abhängig von der Gesamtgröße einer Zusammenhangskomponente. Eine ausführliche Schilderung des Verfahrens kann Anhang G entnommen werden.

Landmarkenkriterium

Die Farbklassifizierung hat Zusammenhangskomponenten von Pixeln geliefert, die ihrer Farbe nach zu einer Landmarke gehören könnten. Jetzt gilt es, herauszufinden, welche davon sich zu einer Landmarke ergänzen und welche keiner Marke angehören.

Dazu wird eine Heuristik verwendet, die ich hier für den Fall einfach umrandeter Landmarken beschreiben will.

Das Verfahren betrachtet lediglich die *Bounding boxes* der Komponenten. Wie im Algorithmus aus Anhang G werden dabei pro Komponente *zwei* Bounding boxes betrachtet: Die tatsächliche und eine, die gegenüber der tatsächlichen um einen konstanten Prozentsatz vergrößert wurde. Auf eine Landmarke wird nun geschlossen, wenn

(*i*) zwei Komponenten jeweils die Farbe des äußeren Rings, bzw. die der zentralen Kreisscheibe besitzen. (Ich nenne sie die *innere*, bzw. die *äußere Komponente*.)

(*ii*) die *tatsächliche* Bounding box der inneren Komponente vollständig innerhalb der *vergrößerten* Box der äußeren liegt.

(*iii*) der Flächeninhalt der inneren Bounding box mindestens 4% der Fläche der äußeren mißt.

(*iv*) darüberhinaus keine weitere Komponente mit ihrer tatsächlichen Bounding box die tatsächliche der Äußeren schneidet.

Sind alle Bedingungen erfüllt, kann der Landmarkenkode an den Farben der beteiligten Komponenten abgelesen werden. Komponenten, die sich nicht zu einer Marke ergänzen, werden

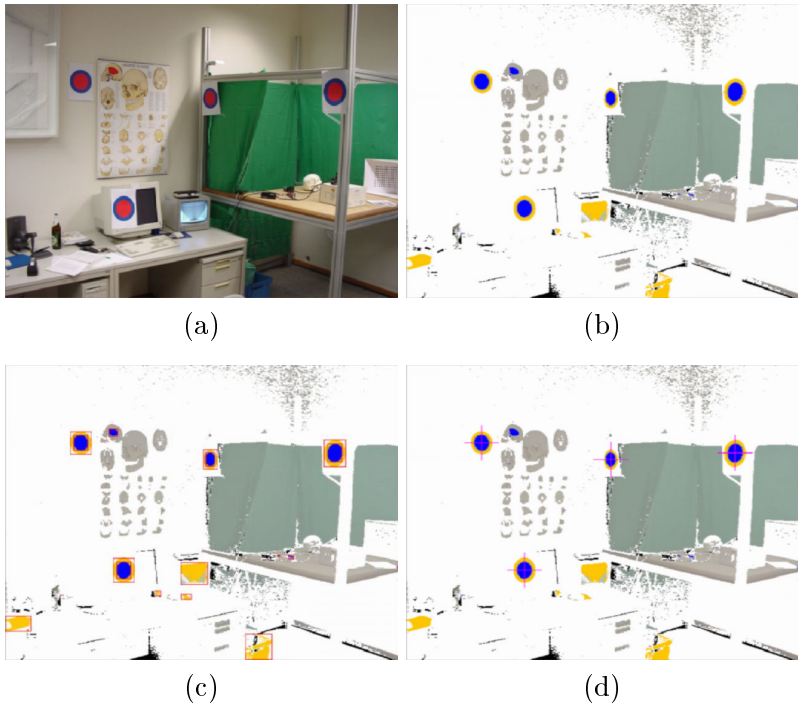

Abbildung 5.10. Vorgang der Landmarkendetektion am Beispiel einer Testaufnahme mit einer herkömmlichen CCD-Kamera. Zum Einsatz kam nur ein Landmarkentyp. In (a) ist das Quellbild zu sehen. (b) zeigt das farbklassifizierte Bild. Zur Verdeutlichung ist das Bild falschfarben. Wie zu sehen ist, erfüllen auch Pixelgruppen das Farbkriterium, die zu keiner Landmarke gehören. In (c) wurden die (tatsächlichen) Bounding boxes der gefundenen Farbkomponenten eingezeichnet. Im letzten Bild (d) sind die detektierten Landmarken markiert.

ignoriert.

Diese einfache Heuristik hat sich bis jetzt bewährt. Auf Probeaufnahmen mit einer gebräuchlichen CCD-Photokamera wurden die Landmarken auch in sehr bunten Umgebungen zuverlässig erkannt, ohne daß Fehldetektionen auftraten.

Nur wenn eine der folgenden **Plazierungsvorschriften** mißachtet wurde, kann eine Landmarke nicht erkannt werden:

- Zwischen zwei Landmarken muß mindestens ein Landmarkendurchmesser Platz gelassen werden.
- Auch von Objekten in der Szene, die eine Farbe der Landmarke enthalten, muß ein Sicherheitsabstand eingehalten werden. Der Abstand beträgt etwa den maximalen Durchmesser dieses Objekts.

Solch eine Fehldetektion ist dann allerdings ein *Fehler erster Art*, der laut 4.3.2 toleriert werden kann.

In Abbildung 5.5.2 ist der gesamte Vorgang bis zur Landmarkendetektion visualisiert.

Peilung

Nun sind die Bilder der Landmarken, sowie deren Markentyp (in unserem Fall die Farbkodierung) ermittelt. Als Vorbereitung zur Positionsrekonstruktion müssen jetzt exakte Peilungen zu den Marken vorgenommen werden. Dazu werden, je nach Größe der Landmarke im Bild, zwei Verfahren eingesetzt:

Pixelschwerpunkt: Bei Landmarken, die im Bild verhältnismäßig klein erscheinen, genügt es, den Pixelschwerpunkt der zentralen Kreisscheibe im Kamerabild zu berechnen. Der Schwerpunkt wird als Bild des Markenzentrums angenommen und anhand der Kamerakalibrierung in einen Sehstrahl umgerechnet. Dieser Strahl ist die Peilung zur Landmarke.

Da Peilungen nur bei sehr kleinen Markenbildern über den Pixelschwerpunkt berechnet werden, lohnt es sich nicht, hier einen höheren Aufwand zu betreiben[1].

Konikanpassung: Erst bei Landmarken, deren Abbild eine gewisse Größe erreicht hat, wird eine ausgefeiltere Methode gewählt. Der Idee ist dabei, zuerst das Bild der Begrenzungskurve zwischen der inneren Kreisscheibe und dem ersten Farbring der Marke zu rekonstruieren.

Diese Begrenzungskurve ist ein stabiles Merkmal. Auch bei teilweise verdeckten Marken sind die Punkte, die auf der Kurve liegen, eindeutig identifizierbar, auch wenn die Kurve als Ganzes nicht mehr sichtbar ist.

Würden wir mit einer gebräuchlichen Kamera arbeiten, deren Bild eine planare Projektion der Umwelt ist, so würde die Grenzkurve (in der Realität ein Kreis) auf eine Ellipse abgebildet werden. Die Rekonstruktionsaufgabe bestünde also darin, eine Ellipse zu finden, die durch die sichtbaren Punkte der Kurve geht. Diese *Ellipsenanpassung* wird gerade in der Bildverarbeitung häufig benötigt. Sie wird zum Beispiel in [24] benutzt, wo sich eine ausführliche Darstellung der Problematik findet. Ein qualitativer Vergleich bekannter Methoden wird in [25] angestellt.

Leider bildet die Panoramakamera Kreise nicht auf Ellipsen ab — die Bilder eines Kreises ähneln nur *annähernd* diesem Kegelschnitt. Nun könnte man das Bild der Grenzkurve zuerst auf eine gedachte Projektionsebene umrechnen, um dann eine Ellipsenanpassung durchzuführen. Tatsächlich habe ich aber einen Ansatz gefunden, der ohne diesen Umweg auskommt.

Dabei betrachte ich den Anpassungsvorgang nicht in der Projektion, sondern wechsle in den *Ray-space*, den Raum, den die Sehstrahlen der Kamera bilden. In diesem Raum bilden die Sehstrahlen, die zur kreisförmigen Grenzkurve der Landmarke führen, einen Kegel mit elliptischem Grundriß (Beweis siehe [26].) Der Ursprung des Ellipsenkegels liegt im Fokus der Kamera. Vervollständigt man die Sehstrahlen zu Geraden, so bilden sie einen elliptischen Doppelkegel um den Ursprung, eine sogenannte *Konik* (siehe Abschnitt 3.1.1).

Die Aufgabe ist es jetzt, eine Konik zu finden, die die Sehstrahlen zu den sichtbaren Punkten der Grenzkurve enthält. Dieser Anpassungsvorgang ist in Anhang D beschrieben und liefert eine Kleinste-Quadrate-Lösung.

Tatsächlich benötigen wir aber nicht die Konik, sondern eine Peilung zum Mittelpunkt der Landmarke, respektive des Grenzkreises zwischen den zwei Farben. Die Konik enthält diesen Kreis. Im allgemeinen liegt der Kreismittelpunkt aber nicht auf ihrer Symmetrieachse. Stattdessen ist er durch die perspektivische Verkürzung leicht verschoben. Laut [26][2] gibt es zu einer Konik im allgemeinen zwei Geraden, auf denen die Mittelpunkte aller Kreise liegen, die die Konik enthält.

Anhang D beschreibt das Verfahren, diese beiden Geraden zu berechnen. Die Geraden gehen durch den Ursprung, also den Fokus der Kamera. Damit korrespondieren theoretisch vier mögliche Peilungen zum Landmarkenmittelpunkt. Aufgrund der groben Richtung, in

[1] Man könnte zum Beispiel überlegen, dem Umstand Rechnung zu tragen, daß jedes Kamerapixel einen anderen Raumwinkel (Steradiant) abdeckt, und den Schwerpunkt als gewichtete Summe berechnen; auch eine mögliche perspektivische Verkürzung könnte berücksichtigt werden. Bei Marken, die nur aus wenigen Pixeln bestehen, bleibt es aber fraglich, ob sich auf diese Weise die Genauigkeit steigern läßt.

[2] Die Quelle liefert nur einen Existenzbeweis. Für eine konstruktive Herleitung sei wieder auf Anhang D verwiesen.

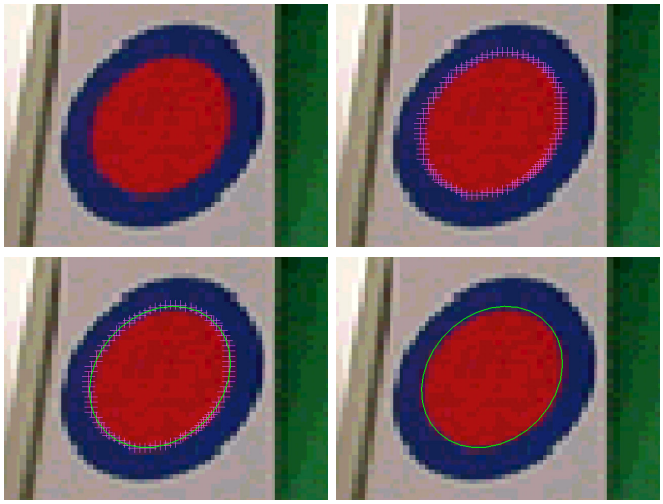

Abbildung 5.11: Anpassungsschritte bei der Lokalisierung einer Landmarke. *Oben Links:* Das Bild der Landmarke. Die Marke wurde mit einer „klassischen“ Kamera aufgenommen. Daher wird der Rand der inneren Kreisscheibe auf eine Ellipse abgebildet. *Oben rechts:* Rekonstruktion der Grenzkurve zwischen rot und blau. Die gefundenen Randpunkte sind magentafarben markiert. *Unten links:* Die Konik wird an die Randpunkte angepaßt. In der aktuellen Projektion erscheint die Konik als Ellipse (grün). *Unten rechts:* Die fertige Rekonstruktion der Randkurve.

der die Landmarke entdeckt wurde, können davon zwei ausgeschlossen werden. Mit jeder der verbleibenden zwei Peilungen korrespondiert jeweils ein Normalenvektor, den ein Kreis in dieser Richtung haben muß.

Anhand des a-priori Wissens über die Landmarkenanordnung ließe sich insbesondere dieser Normalenvektor nutzen, um herauszufinden, welche der beiden Peilungen plausibler ist. Vorerst habe ich aber darauf verzichtet, zumal die Peilungen meistens recht nahe beieinander liegen. Aus diesem Grund wird zur Zeit einfach das Mittel der beiden Peilungen genommen (welches wieder die Symmetrie- oder Hauptachse der Konik ist).

Dieser recht hohe Aufwand bei der Lokalisierung größerer Marken ist vor allem deswegen gerechtfertigt, weil die Peilung auf diese Weise robuster gegen teilweise Verdeckung ist, wie in Abbildung 6.3.3 zu sehen ist.

Abbildung 5.11 zeigt den Anpassungsvorgang an einer Landmarke. Da es sich um ein Bild einer „klassischen“ Kamera handelt, ist der Markenumriß eine Ellipse.

5.5.3 Positionsrekonstruktion

Markenidentifizierung

Die Landmarkenidentifizierung ist vorerst relativ rudimentär ausgeführt: Alle möglichen zu den erkannten Markentypen passenden Zuordnungen „Peilung $\longmapsto$ bekannte Landmarke“ werden durchprobiert. Dabei wird jedesmal versucht, mit der jeweiligen Zuordnung auf die augenblickliche Lage im Raum zu schließen. Diejenige Zuordnung, die mit dem kleinsten Fehler angepaßt werden konnte, wird als abschließende Identifikation weitergereicht.

Denkbare Optimierungen, wie die Ausnutzung einer Frame-to-Frame Kohärenz, wurden weggelassen.

Das führt, wie in Abschnitt 4.3.3 geschildert, zu einem echten Performance-Einbruch, sobald einige Markentypen mehrfach vergeben werden. Deshalb kann der Kopfverfolger zum jetzigen Zeitpunkt nur sinnvoll eingesetzt werden, wenn eine eindeutige Markenkodierung vorliegt.

In der Simulation existiert noch die Möglichkeit, eine *allwissende* Markenidentifizierung einzusetzen. Dabei wird die Kenntnis der Simulation dazu ausgenutzt, die gemessenen Peilungen mit der korrekten Identifikation zu versehen. Dieser Trick kann benutzt werden, um die restlichen Komponenten zu testen. Im wirklichen Einsatz ist er natürlich nicht anwendbar.

Rekonstruktion der Lage im Raum

Die Rekonstruktion der Lage im Raum erfolgt durch ein iteratives Mischverfahren. Ausge-

hend von einer Startposition werden im Wechsel eine rotatorische und eine translatorische Anpassung der Messung an die bekannten Landmarken vorgenommen. Dabei geht es jeweils darum, eine Position, bzw. Orientierung im Raum zu finden, so daß die gemessenen Peilungen möglichst genau durch die vermessenen Landmarkenpositionen gehen.

Dieser Vorgang wird so lange wiederholt, bis die Positionsänderung durch die letzte translatorische Anpassung unter einem zehntel Millimeter liegt. In diesem Falle erfolgt noch eine letzte rotatorische Anpassung; das Ergebnis ist die rekonstruierte Kameraposition.

Die Startposition kann prinzipiell willkürlich gewählt werden. Eine vernünftige initiale Wahl ist die Mitte des Raums, in dem der Verfolger eingesetzt wird. In besonders ungünstigen Fällen kann es leider trotzdem passieren, daß das Verfahren nicht konvergiert. In diesem Fall wird die Rekonstruktion nach 48 Iterationen abgebrochen, und eine Schätzung der Prädiktion wird zurückgeliefert. Bis jetzt trat eine Divergenz jedoch nur sehr selten auf.

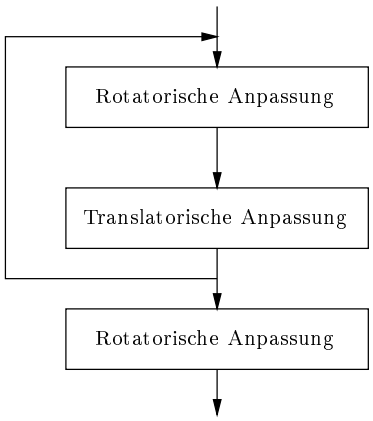

Abbildung 5.12: Schematische Darstellung des Rekonstruktionsvorgangs.

Im laufenden System wird als Startposition die Schätzung der Prädiktion genommen, was einen großen Geschwindigkeitsvorteil bringt, da die Iteration schneller konvergiert.

Rotatorische und translatorische Anpassung werden in Anhang E genauer erklärt.

5.5.4 Prädiktion

Bis jetzt verwende ich eine sehr einfache Prädiktion, die nur auf den letzten beiden Messungen aufbaut:

- Die Position wird aus den beiden letzten Positionen linear fortgeschrieben.
- Die gesuchte Rotation wird ebenfalls linear, entlang eines Großkreises der Quaternionen-Einheitskugel extrapoliert.

Das Modell, das dieser Prädiktion zugrunde liegt, lautet also: *Der zu verfolgende Kopf bewegt sich seit der letzten Messung mit konstanter (Winkel-)Geschwindigkeit weiter.*

5.6 Rahmenapplikation

Der Verfolger, sowie Kamerakalibrierung und -simulation sind gemeinsam in einer Applikation zusammengefaßt. Die einzelnen Moduln sind zwar absolut unabhängig voneinander, der gemeinsame Rahmen erleichtert aber Entwicklung und Tests.

Hauptfunktionen der Applikation sind

- Quellansicht: Die Darstellung des „Kamerabilds“ aus einer frei wählbaren Quelle.
- Berechnung der Kamerakalibrierung aus einer xfig-Datei.
- Entzerrung des Kamerabilds zu einer Panoramasicht. Im Panorama kann sich der Benutzer in allen Richtungen umsehehen.
- Der eigentliche Kopfverfolger: Aufgrund der Kamerabilder wird eine Positionsrekonstruktion für die Panoramakamera vorgenommen.
- Interaktive Begehung der Simulationsumgebung:

  - Interaktive Kamerasimulation. (Man steuert durch den Raum und sieht gleichzeitig die Bilder einer simulierten Panoramakamera, als ob die sich an der Benutzerposition befinden würde.)
  - Aufzeichnung einer Trajektorien in der Simulationsumgebung.
  - Abspielen der Trajektorie mit gleichzeitiger Kamerasimulation.

- Die Bilder der Kamerasimulation können wieder in die Panoramasicht und das Verfolgungssystem eingespeist werden.

- Während der Kopfverfolger läuft, können wahlweise Markierungen eingeblendet werden, die über den Landmarkenlokalisierungsprozeß und die rekonstruierte Kameraposition informieren. Die Einblendungen sind, jeweils entsprechend entzerrt, sowohl in der Quellansicht, als auch im Panorama-Modus und in der virtuellen Umgebung sichtbar.

Eine Übersicht über die Menüstruktur und alle möglichen Tastenkombinationen liefert Anhang I.

Kapitel 6

Ergebnisse

6.1 Kamerakalibrierung

Zu Beginn der Arbeit stand für kurze Zeit eine Panoramakamera zur Verfügung, mit der hochauflösende Einzelbilder aufgenommen werden konnten. Die Kamera ist in Abschnitt 3.3 beschrieben.

Das in Abschnitt 5.3 vorgestellte Verfahren zur Kamerakalibrierung wurde an Bildern dieser Kamera getestet. Eines dieser Bilder ist in Abbildung 6.1, *links*, zu sehen. Mit Hilfe der schon in Abbildung 5.3 gezeigten Kalibrierung wurden daraus Ansichten des Raumes rekonstruiert, aus dem die Aufnahme stammt. Eine der Ansichten ist in Abbildung 6.1, *rechts*, wiedergegeben.

Es ist schwierig, eine quantitative Aussage über die Genauigkeit des Kalibrierungsverfahrens zu machen, solange keine wohldefinierten Testmuster mit der Kamera analysiert wurden. Zumindest konnte aber folgende Beobachtung gemacht werden: Je nachdem, welche Geradenbilder einer Aufnahme in die Kalibrierung eingegangen sind, konnten sich die jeweiligen Abbilder des rekonstruierten Scheitelpunkts des Spiegelparaboloids in ihrer Lage um bis zu 10 Pixel unterscheiden.

Dies könnte bedeuten, daß manche Geradenbilder nicht genau genug nachgezeichnet wurden. Es kann aber auch heißen, daß die Abbildungseigenschaften der Kamera nicht ideal sind (siehe Anhang C). Das ist in unserem Fall sogar zu vermuten, da bei der Kamera, mit der die gezeigten Kalibrierungstests gemacht wurden, die zentrale Metallstange leicht verbogen war, die Sehstrahlen also nicht mehr exakt parallel zur Symmetrieachse des Paraboloids verlaufen sind.

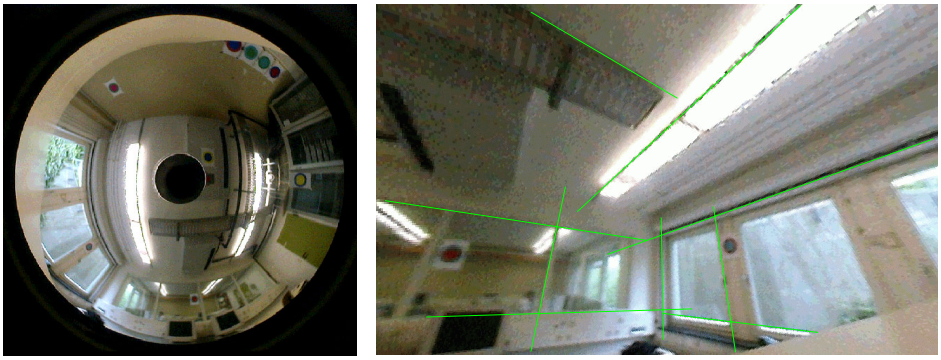

Abbildung 6.1. Mit Hilfe der Kamerakalibrierung rekonstruierte Raumansicht. *Links* ist das Quellbild zu sehen, *rechts* die entzerrte Ansicht. In der Raumansicht sind für eine qualitative Kontrolle der Kalibrierung gerade Linien mit grünen Strecken nachgezogen.

Für die Simulation hingegen ist jede Kalibrierung eine perfekte Kalibrierung, da sowohl die Kamerasimulation als auch der Bildverarbeitungsteil auf identische Kalibrierungsdaten zurückgreifen. Die Kamerakalibrierung hatte also keinen Einfluß auf die Qualität der Ergebnisse in den Simulationsläufen. Eine abschließende Beurteilung der Kalibrierungsmethode kann daher noch nicht erfolgen.

6.2 Testszenario

Die vorgestellten Ergebnisse beziehen sich bereits auf die Kalibrierung der mittlerweile gelieferten NTSC-Kamera. Sie unterscheidet sich in ihrer Abbildungsgeometrie leicht von der Einzelbildkamera, wie sich an der in Abbildung 6.3 gezeigten Aufnahme sehen läßt. Der durch die Bauweise nach oben hin verdeckte Bereich (die zentrale dunkle Scheibe) umfaßt einen größeren Raumwinkel als bei der Einzelbildkamera. Dafür deckt das Bild mehr Pixel ab, womit die Kameraauflösung besser ausgenutzt wird.

Abbildung 6.3: Bild der NTSC-Panoramakamera.

Der virtuelle Raum, in dem die Testläufe stattgefunden haben, ist eine $4 \times 5\,\mathrm{m}$ große Szene mit zwei Tischen, mit denen Verdeckungstests durchgeführt wurden. Tür und Tischbeine sind bewußt in einer Farbe gehalten, die auch in den Landmarken wieder auftaucht, um die Landmarkendetektion zu erschweren.

An Wänden und Decke des Raums wurden 20 Landmarken willkürlich verteilt. Lediglich bei der Zuteilung der Landmarkentypen wurde darauf Wert gelegt, daß kein Markentyp neben einem Objekt plaziert wird, das eine Farbe der Marke wieder aufgreift. Abbildung 6.2 zeigt den beschriebenen Raum mit angebrachten Landmarken.

Wie zu sehen ist, wurden nur sechs verschiedene Landmarkentypen eingesetzt. Dies liegt daran, daß der Markendetektionsalgorithmus zur Zeit nur einfach umrandete Landmarken unterstützt und nicht mehr als drei Farben verwendet werden sollten, da es unrealistisch ist, in realen Kamerabildern mehr als drei Farbklassen vernünftig trennen zu wollen. Nach Tabelle 5.2 ergeben sich daher sechs Kodierungsmöglichkeiten für die Marken.

Die im folgenden geschilderten Erfahrungen mit dem System wurden in diesem Testszenario mit den unterschiedlichsten Trajektorien gesammelt. Dabei war neben der Genauigkeit des Verfolgungssystems vor allem die Echtzeitfähigkeit ein wichtiger Gesichtspunkt, weshalb

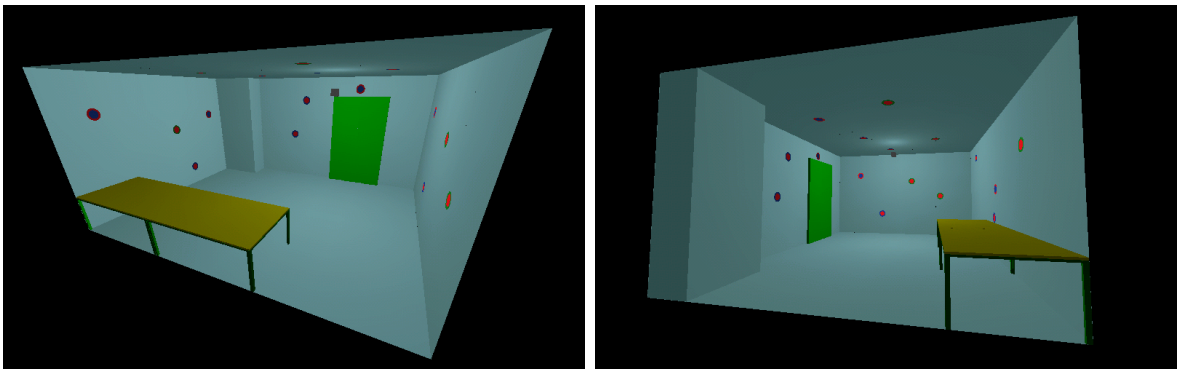

Abbildung 6.2. Virtuelle Testumgebung mit angebrachten Landmarken.

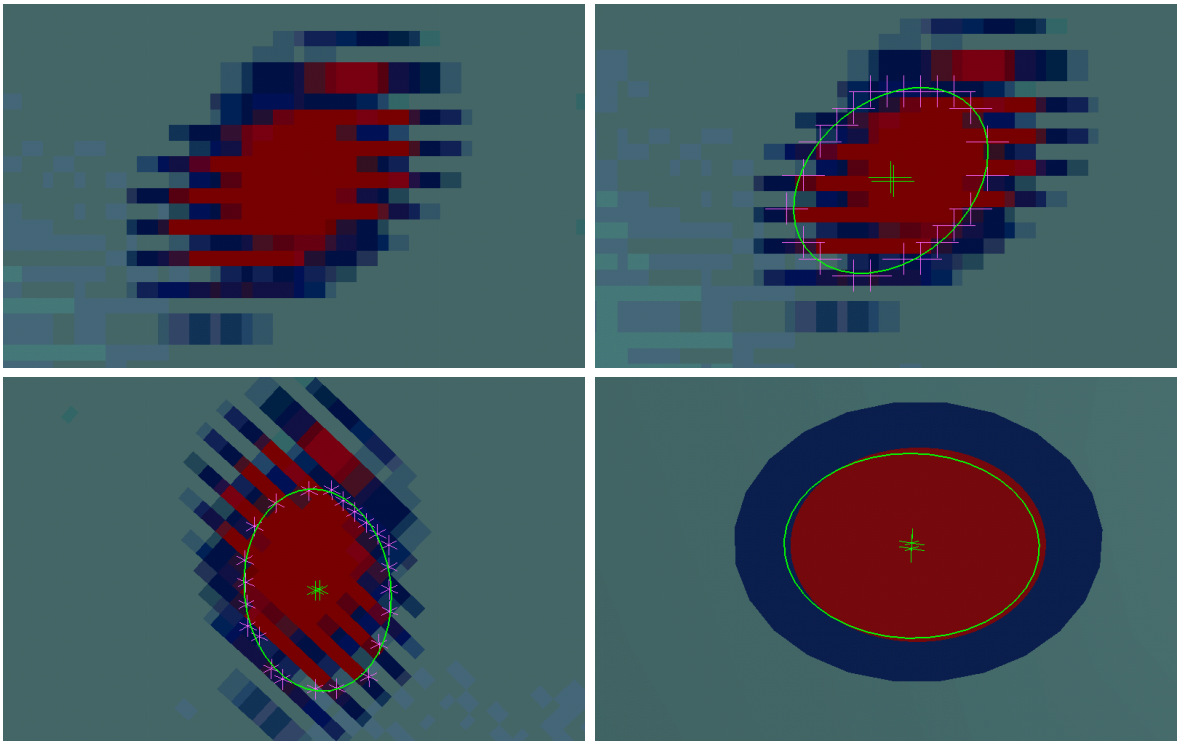

Abbildung 6.4. Detektion und Lokalisierung bei starken Kammartefakten. *Oben links:* Durch einen schnellen Kameraschwenk bedingt treten an einer Marke starke Kammartefakte auf. *Oben rechts:* Bei der Bestimmung der Randpixel (magentafarben) gingen nur ungeradzahlige Bildzeilen ein. In Grün ist das Ergebnis der daraus abgeleitete Konikanpassung zu sehen. Die beiden oberen Bilder sind Ausschnitte des Kamerabilds. *Unten links:* Die gleiche Marke in einer entzerrten Teilansicht des Panoramas. *Unten rechts:* Die rekonstruierte Konik wurde innerhalb der Simulationsumgebung mit der tatsächlichen Landmarke überlagert.

ich auf die Geschwindigkeit der der einzelnen Verarbeitungsschritte besonders eingehe.

6.3 Landmarkenlokalisierung

6.3.1 Farbklassifizierung

Dank intensiver Optimierungen konnte die Farbklassifizierung, die ursprünglich den größten Anteil an der Gesamtrechenzeit des Systems hatte, so weit beschleunigt werden, daß sie auf dem Referenzsystem (Athlon 600, 128 MB RAM, siehe auch Abschnitt 5.1.2) nur noch etwa 5 bis 6 ms benötigt.

Dieser Wert bezieht sich auf die Klassifizierung im RGB-System, da die Kamerasimulation RGB-Daten liefert. Tatsächlich ist die Farbklassifizierung im YIQ-System, dem Farbsystem, mit dem die Bilder von der NTSC-Kamera erhalten werden, geringfügig schneller, da die Look-up-Tabelle für eindeutig klassifizierte YIQ-Voxel dichter besetzt ist als die für RGB-Voxel (vgl. Anhang F).

Die Qualität der Farbklassifizierung ist zufriedenstellend. Auf Aufnahmen realer Landmarken, die mit einer handelsüblichen CCD-Einzelbildkamera aufgenommen wurden, konnten vor allem die Landmarkenfarben rot, grün und blau zuverlässig erkannt werden. Die Klassifizierung blieb weitestgehend unabhängig von Unterschieden in der Beleuchtung.

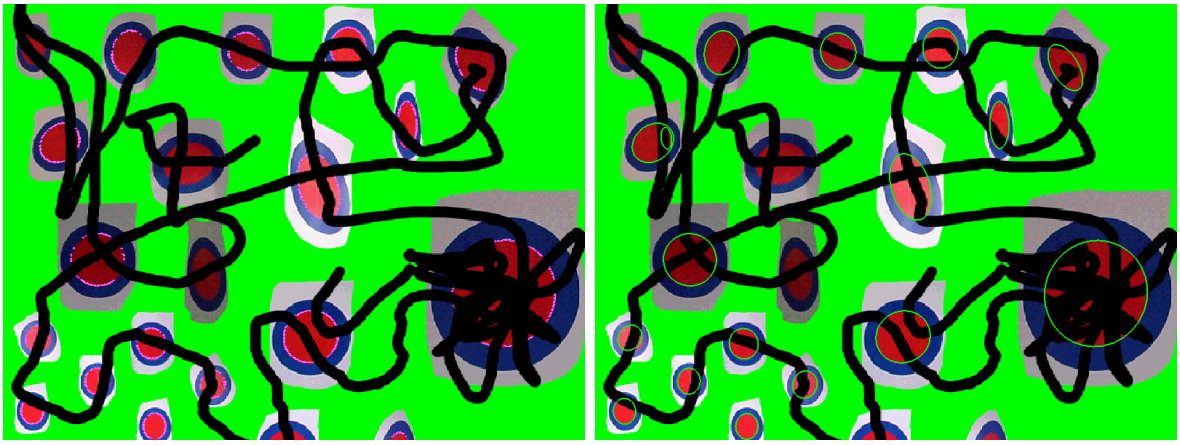

Abbildung 6.5. Konikanpassung bei teilweise verdeckten Landmarken. Als Testbild wurden mehrere Aufnahmen des gleichen Landmarkentyps in einem Bild zusammengestellt. Mit einem schwarzen Pinsel-Werkzeug wurde eine partielle Verdeckung der Marken simuliert. *Links:* Das Quellbild, bei dem die erkannten Punkte der Grenzkurve magentafarben markiert wurden. *Rechts:* Die Rekonstruktion der Grenzkurven durch die Konikanpassung. *Anmerkung:* Da die hier gezeigten Landmarkenbilder mit einer „klassischen" Kamera aufgenommen wurden, sind die Grenzkurven tatsächlich Ellipsen. Das Bild wurde erzeugt, indem die Konikanpassung mit der Kalibrierung einer Lochkamera parametrisiert wurde. Ein Bild der Anpassung bei einem Panoramabild findet sich in Abbildung 6.4.

6.3.2 Landmarkendetektion

Die Kammartefakte des Kamerabilds stellen für die Detektion eine unlösbare Schwierigkeit dar. Abbildung 6.4 verdeutlicht dies an einem Bild, das während eines schnellen Kameraschwenks aufgenommen wurde. Aus diesem Grunde blieb schließlich nichts anderes übrig, als die gesamte Bildverarbeitung auf die ungeradzahligen[1] Bildzeilen zu beschränken.

Die Landmarkendetektion, die maßgeblich von Algorithmus G.1 Gebrauch macht, benötigt je nach Anzahl Farbkomponenten im Bild 6 bis 8 ms.

Bei Realaufnahmen, die mit der handelsüblichen CCD-Einzelbildkamera in unterschiedlichen Räumen des IAIM gemacht wurden, gab es sehr wenige Fehldetektionen und ausschließlich Fehler erster Art. In der Simulation werden fast alle Marken erkannt; Fehldetektionen zweiter Art gab es auch hier nicht.

Damit erfüllt der Detektionsschritt alle Anforderungen.

6.3.3 Lokalisierungsschritt

Die Konikanpassung ist recht robust gegen unterbrochene Landmarken, wie in Abbildung 6.3.3 zu sehen ist. Bei Marken, die größtenteils verdeckt sind und von denen nur noch ein kleiner Randausschnitt sichtbar ist, liefert die Konikanpassung aber hin und wieder stark von der tatsächlichen Landmarkenform abweichende Anpassungen. Aus diesem Grunde verwirft die Implementierung Koniken, deren Achsenverhältnis unsinnige Werte annimmt. Durch diese Maßnahme konnten starke Fehlschätzungen der Markenmittelpunkte weitestgehend ausgeschlossen werden.

Erschwert wird die Lokalisierung natürlich dadurch, daß, wie in Abbildung 6.4 gezeigt, aufgrund der Kammartefakte nur jede zweite Zeile verwendet werden kann, um Randpixel zu detektieren.

Zur Veranschaulichung der Tatsache, daß zu jeder Landmarke *zwei* potentielle Mittelpunkte existieren, zeigt Abbildung 6.6 noch einmal eine Marke, bei der die potentiellen Mittel-

[1] Bei Zählung ab 0. In diesem Fall sind nämlich die Bildzeilen 1, 3, 5, ... eine sechzigstel Sekunde aktueller als die geradzahligen Bildzeilen.

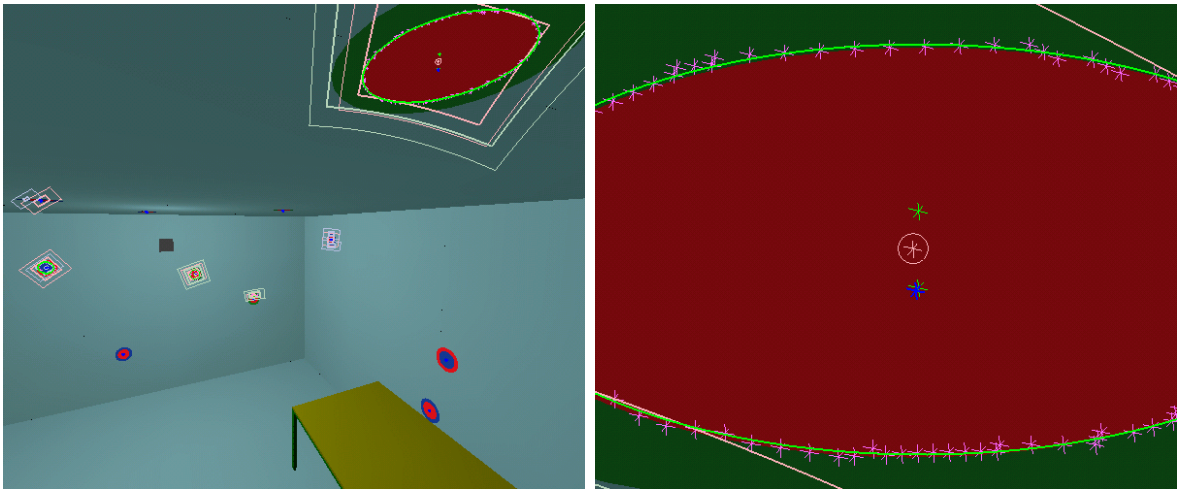

Abbildung 6.6. Bild der beiden potentiellen Mittelpunkte einer Landmarke. *Links:* Durch die besonders flache Aufsicht auf die Marke kommen die potentiellen Mittelpunkte recht weit voneinander entfernt zu liegen. Das Bild zeigt eine Überlagerung der an das Pixelbild angepaßten Konik mit der Simulationsumgebung. *Rechts:* Die Vergrößerung der Marke zeigt noch einmal die beiden Mittelpunktsschätzungen der Konikanpassung in Grün; blau eingetragen ist das tatsächliche Landmarkenzentrum, wie es aus der Simulation bekannt ist.

punkte gut voneinander zu unterscheiden sind. Im Großteil der Fälle liegen sie aber wesentlich näher beeinander, so daß es nur einen kleinen Fehler einbringt, zwischen den beiden Punkten zu mitteln.

Der Aufwand für die Konikanpassung wächst mit der Anzahl Landmarken und der Anzahl beteiligter Randpixel. Im genannten Testszenario belief sich ihre Laufzeit in der Regel auf 6 bis 8 ms.

6.4 Positionsrekonstruktion

6.4.1 Landmarkenidentifizierung

Mit der naiven, permutativ arbeitenden Landmarkenidentifizierung nimmt bei den sechs Markentypen in zwanzig Landmarken ein Identifizierungsschritt mehrere Minuten in Anspruch. Das ist auch nicht weiter verwunderlich, schaut man sich Tabelle 4.1 an. Nicht umsonst wird in Abschnitt 4.3.3 darauf hingewiesen, daß möglichst jede Landmarke einen eineindeutigen Typ besitzen sollte.

Da das aktuelle Detektionsmodul aber leider nur einfach umrandete Landmarken unterstützt, konnten keine 20 verschiedenen Markentypen eingesetzt werden (vgl. Abschnitt 6.2).

Das Identifizierungsmodul wurde daher vorerst gegen das „allwissende“ Modul ausgetauscht, das die Information, welche Identität eine gefundene Marke besitzt, direkt aus der Simulation bezieht.

Damit ist die von der Markenidentifizierung benötigte Zeit natürlich verschwindend gering.

6.4.2 Rekonstruktion der Lage im Raum

Rekonstruktionsfehler

Die Qualität der Positionsrekonstruktion läßt sich anhand der mittleren Fehler der translatorischen, bzw. rotatorischen Anpassung beurteilen. Sie berechnen sich wie folgt.

Seien $\boldsymbol{p}_i$ die n Positionsrekonstruktionen entlang einer Trajektorie und $\boldsymbol{p}_i^0$ die tatsächlichen Positionen, wie sie aus der Simulation bekannt sind. Der mittlere Fehler der Positionsrekonstruktion ergibt sich dann zu

$$\sigma_{\mathrm{tr}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{p}_i^0\|^2} \,. \tag{6.1}$$

Analog bestimmt man den mittleren rotatorischen Fehler über

$$\sigma_{\mathrm{rot}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} \Delta_\varphi(\dot{\boldsymbol{r}}_i, \dot{\boldsymbol{r}}_i^0)^2} \,, \qquad \Delta_\varphi(\dot{\boldsymbol{q}}_1, \dot{\boldsymbol{q}}_2) := 2\arccos\left([\dot{\boldsymbol{q}}_1 \dot{\boldsymbol{q}}_2^*]_1\right) \,, \tag{6.2}$$

wobei $\dot{\boldsymbol{r}}_i$ und $\dot{\boldsymbol{r}}_i^0$ die Rotationsquaternionen der Rotationsrekonstruktionen bzw. bekannten Orientierungen sind; $\Delta_\varphi(\cdot,\cdot)$ ist hier die Winkeldifferenz zwischen zwei Quaternionen, $[\cdot]_1$ liefert die erste Komponente des Arguments, in diesem Fall eines Rotationsquaternions.

Testläufe

In den meisten Fällen konvergiert die Positionsrekonstruktion zufriedenstellend: Liegt durch vorhergehende Rekonstruktionen bereits eine Schätzung der aktuellen Kameralage vor, so ist die geforderte Genauigkeit nach vier bis fünf Iterationsschritten erreicht. Für die initiale Rekonstruktion, sozusagen nach dem „Einschalten“ des Systems, werden etwa zwanzig bis dreißig Iterationszyklen benötigt. Je nachdem entfallen damit auf den Rekonstruktionsschritt etwa 1–8 ms.

Nur in seltenen Fällen divergiert das Verfahren. Die aktuelle Lage im Raum wird dann mit Hilfe der Prädiktion geschätzt.

Abbildung 6.7 zeigt die Rekonstruktionsfehler für zwei repräsentative Trajektorien. In der rechten Spalte ist ein Durchlauf mit zeitweiliger Divergenz des Rekonstruktionsalgorithmus zu sehen. In allen Testläufen waren translatorische und rotatorische Abweichung stark korreliert.

Die Wirkung der beiden mittleren Fehler auf den Fehler innerhalb der Bildebene der AR-Brillem, hängt insbesondere von der Entfernung der sichtbaren Objekte ab. Um den resultierenden *Screen error* einer AR-Einblendung einschätzen zu können, kann in der Simulationsumgebung eine Enblendung wie in Abbildung 6.8 vorgenommen werden. Zu sehen ist ein grüner Würfel mit zehn Zentimetern Kantenlänge, der während der Kamerafahrt entlang der aufgezeichneten Trajektorie stets genau einen halben Meter vor der Kameraposition eingeblendet wird. Sein Bild befindet sich also immer an der gleichen Stelle. Zusätzlich wird ein weiterer, roter Würfel einen halben Meter vor der *rekonstruierten* Kameraposition eingezeichnet. Wäre die Rekonstruktion perfekt, kämen die beiden Würfel zur Deckung. Durch die Fehler in der Rekonstruktion verschieben sich die beiden Bilder gegeneinander. Das Ausmaß dieser Verschiebung entspricht dem *Screen error*, der zwischen Bildern eingeblendeter und realer Gegenstände entsteht, die einen halben Meter vom Benutzer entfernt sind.

6.5 Geschwindigkeit des Gesamtsystems

Zusammengenommen benötigen von der Bildverarbeitung bis zur Positionsrekonstruktion alle Komponenten zwischen 20 und 30 Millisekunden, womit das System in der Lage ist, zwischen 33 und 50 Bildern pro Sekunde zu verarbeiten. In den meisten Situationen erreicht die Auswertung etwa 43 Bilder pro Sekunde auf dem Athlon 600-System. Es ist zu erwarten, daß sich auf dem geplanten Celeron 800-System ähnliche Zeiten ergeben werden.

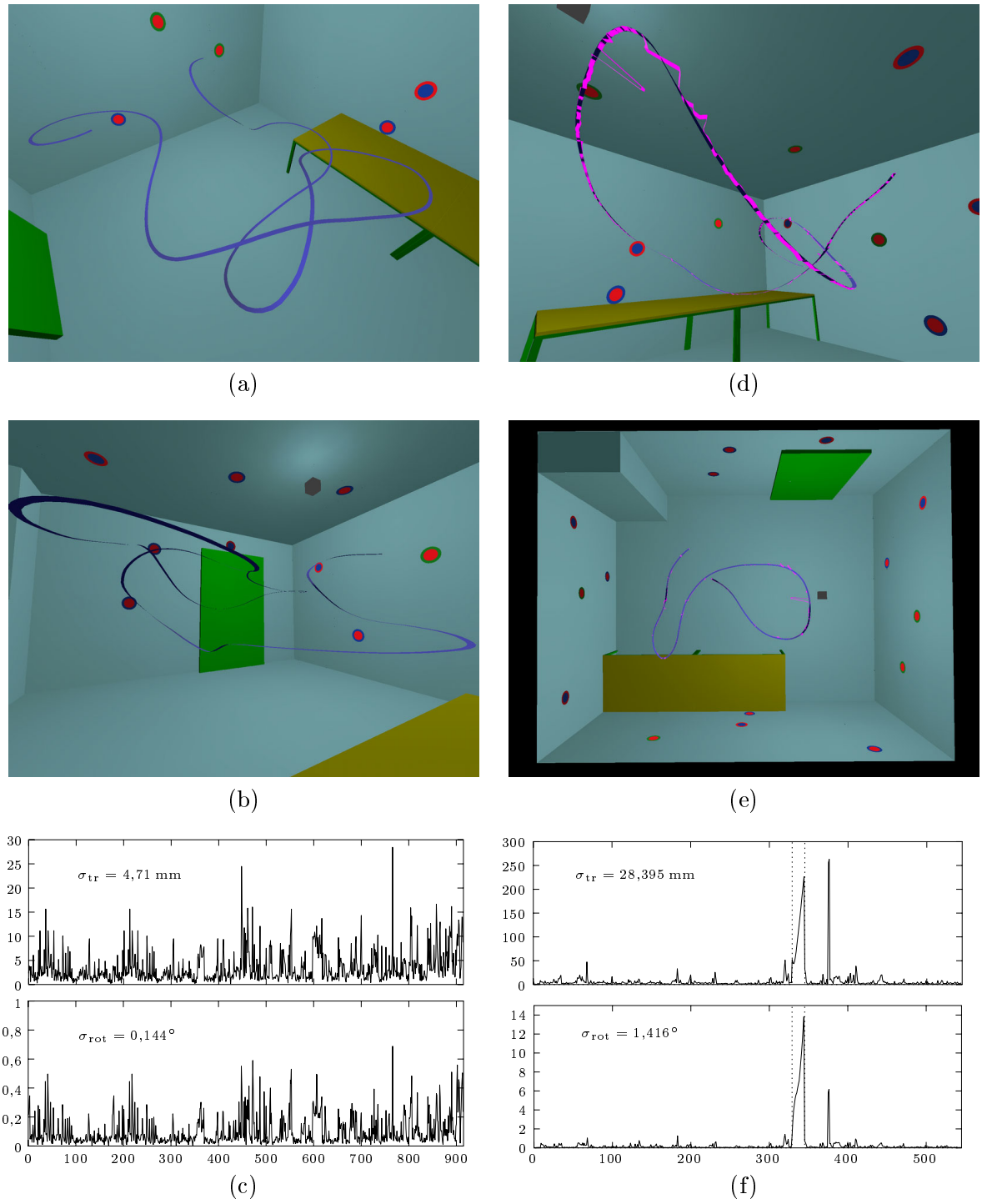

Abbildung 6.7. Rekonstruktionsfehler für zwei verschiedene Trajektorien. *Linke Spalte:* (a) und (b) zeigen die Kameratrajektorie aus verschiedenen Perspektiven. In (c) sind der translatorische (*oben*) und der rotatorische (*unten*) Rekonstruktionsfehler auf dieser Bahn zu sehen. Nach rechts ist die Bildfolgennummer abgetragen. *Rechte Spalte:* (d) und (e) zeigen wieder eine Trajektorie, die diesmal mit der rekonstruierten Kameratrajektorie (magentafarben) überlagert ist. In Deckennähe divergierte der Positionsrekonstruktionsalgorithmus, was in der Überlagerung gut zu erkennen ist. Abbildung (f) zeigt die zugehörigen Rekonstruktionsfehler; die Divergenz trat innerhalb der gestrichelten Linien ein. Die steil ansteigende Flanke zeigt hier, wie Prädiktion und „Realität“ auseinanderlaufen, bis der Algorithmus erneut konvergiert und der Fehler umgehend wieder minimal wird.

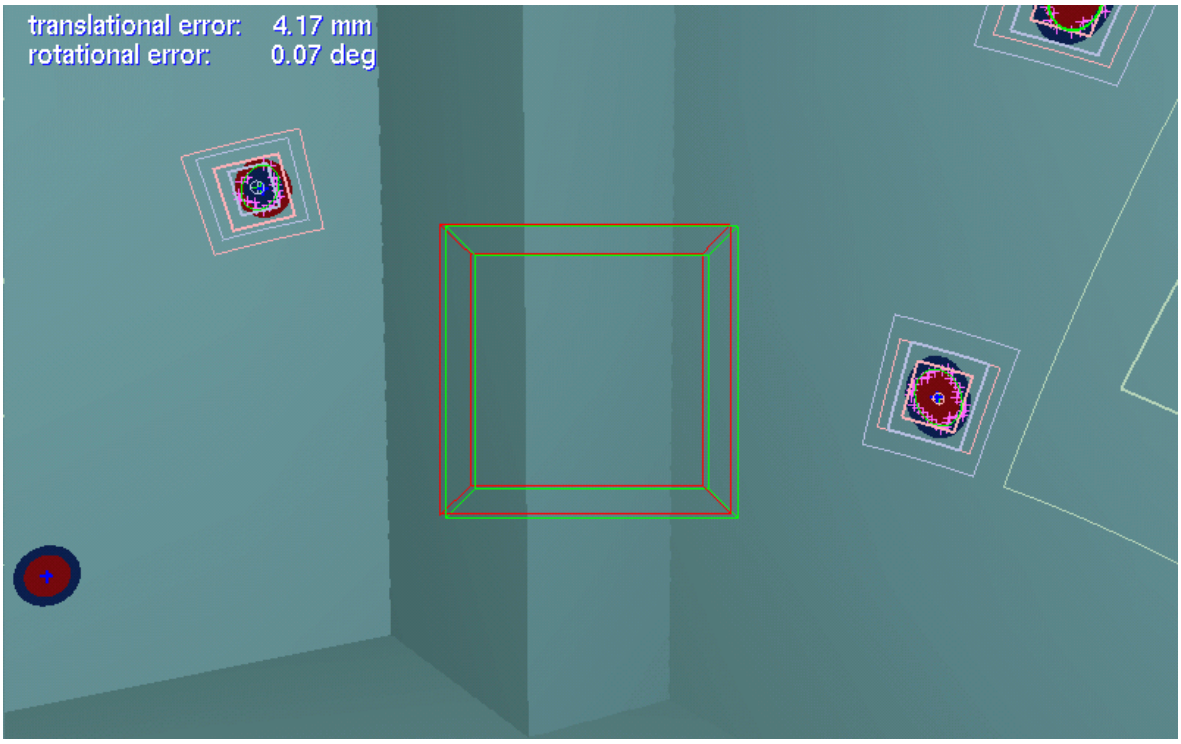

Abbildung 6.8. Visualisierung des *Screen errors* einer Einblendung. Zu sehen ist die aus Fehlern der Positionsrekonstruktion entstandene Verschiebung zwischen einem eingeblendeten (*rot*) und einem „realen“ Würfel (*grün*). Beide Würfel haben zehn Zentimeter Kantenlänge und befinden sich einen halben Meter vor der Kameraposition.

Das heißt, daß eine Echtzeitverarbeitung der von der Panoramakamera gelieferten 30 Bilder pro Sekunde durchaus möglich ist. Es bleibt sogar noch genügend Rechenzeit für einfachere AR-Anwendungen.

6.6 Kameraauflösung und Genauigkeit

Alle vorgestellten Testläufe wurden für eine Kamera mit NTSC-Auflösung (640×480 Pixel, Zeilensprungverfahren) simuliert. Die Kamerabildberechnung lief ohne Super-Sampling, für die Verzerrung der Umgebungskarte wurde das Nearest-Neighbour-Verfahren angewandt.

Verbesserungen, die durch Super-Sampling und/oder Gauß-Filterung zustande kamen, waren vernachlässigbar.

Der Schritt zu einer Auflösung von 1024×768 Pixeln und “progressive scan”-Abtastung führte bei der Trajektorie der linken Spalte der Abbildung 6.7 zu einem mittleren translatorischen Fehler von 4,25 mm (10,9% Verbesserung), bzw. 0,14° (4,4%). Verglichen mit der Auflösungssteigerung von 60% in horizontaler und 220% in vertikaler Richtung zeigen diese Ergebnisse: Mit einer Auflösungsteigerung alleine kann die Genauigkeit des Verfolgers nicht wesentlich verbessert werden.

Kapitel 7

Diskussion

7.1 Kamerakalibrierung

Die ersten Ergebnisse, die mit der Kamerakalibrierung gewonnen wurden, sind vielversprechend.

Eine bestechende Eigenschaft des Kalibrierungsverfahrens ist, daß man kein spezielles Testmuster benötigt, da das Verfahren lediglich das Vorhandensein gerader Linien in der Umgebung voraussetzt. Dies ist bei den meisten Innenraumaufnahmen schon allein durch die architektonischen Gegebenheiten der Fall.

Für eine besonders präzise Kalibrierung bietet es sich dennoch an, spezielle Kalibrierungsmuster zu verwenden, in denen gerade Linien zu finden sind, die sich besonders gut lokalisieren lassen. Es wäre zum Beispiel denkbar, ein schachbrettartiges Muster zu verwenden, bei dem die Linien an ausgeprägten Bildgradienten zu erkennen sind. Weiterhin ist es aber nicht notwendig, die genauen geometrischen Parameter des Musters zu kennen. Für das Kalibrierungsverfahren genügt es zu wissen, daß die sichtbaren Kreissegmente ursprünglich Geradenstücke waren.

In diesem Zusammenhang ist zu überlegen, ob die Kamerakalibrierung nicht automatisch, ohne die manuelle Angabe der Kreissegmente erfolgen könnte. Dazu müßte eine Methode gefunden werden, die automatisch Kreissegmente im Gradientenbild erkennt. In [24] werden Verfahren zur automatischen Detektion von Bildellipsen vorgestellt. In unserem Fall ist das Problem stärker eingeschränkt, da echte Kreissegmente gesucht sind. Es müßte daher reichen, die Kreissegmente über die HOUGH-Transformation zu gewinnen.

Eine automatisierte Kreissegmenterkennung hätte zudem den Vorteil, daß durch eine Modellierung der Ebenen durch Gerade und Fokus die Genauigkeit der Kreissegmentlokalisierung durch eine anschließende Modellanpassung noch weiter erhöht werden kann. Eine Sub-Pixel-Auflösung ist dabei ohne weiteres zu erreichen wie in [27] anhand einer klassischen Kamerageometrie gezeigt wurde.

Ein weiterer Vorteil des in der Arbeit entwickelten Kalibrierungsverfahrens ist, daß beliebig viele Geradenbilder in die Berechnung eingehen können und die Genauigkeit auf diesem Wege erhöht wird.

Leider konnte bis jetzt keine quantitative Beurteilung der Kalibrierung erfolgen. Dazu müßten erst wohldefinierte Testmuster ausgewertet werden.

Ferner ist zu überlegen, ob das Kameramodell erweitert werden sollte. So könnte man zum Beispiel eine schräge Sicht der Kamera auf das Paraboloid miteinkalkulieren und den Fall einer nicht orthonormalen CCD-Matrix im Kameramodell berücksichtigen.

Effizienter wird es aber wohl sein, die Abweichung der Kamera von den idealen Abbildungseigenschaften als eine nicht weiter spezifizierte Verzerrung der Bildebene zu betrachten. Dann wäre ein Ansatz die Gitteranpassung: Nach einem Kalibrierungsschritt, der eine idea-

le Kamera voraussetzt, lege man ein regelmäßiges quadratisches Gitter über das Kamerabild und versuche, Modell und Kamerabild zur Deckung zu bringen, indem man sukzessive Knoten des Netzes und mit ihnen die darunter liegenden Bildbereiche verschiebt.

Schließlich sei noch erwähnt, daß der Einsatz einer Panoramakamera für den Kopfverfolger nicht zwingend notwendig ist. Das System kann im Prinzip auch mit einer gewöhnlichen Kamera betrieben werden. Dazu müßte lediglich eine passende Kamerakalibrierung angegeben werden; die restlichen Systemkomponenten blieben unverändert. Dieses Vorgehen hätte nur einen gravierenden Nachteil. Da klassische Kameras nur einen verhältnismäßig kleinen Raumwinkel abdecken, sind dann entsprechend weniger Landmarken im Bild zu sehen. Damit sinkt die Genauigkeit der Positionsrekonstruktion.

7.2 Kamerasimulation

Grenzen und Möglichkeiten der Simulation wurden schon ausführlich in Abschnitt 5.4.4 diskutiert. Es sei hier aber noch einmal darauf hingewiesen, daß die Simulation über den Ersatz der ursprünglich nicht verfügbaren Panoramakamera hinaus eine wichtige Rolle bei der Entwicklung des Kopfverfolgers gespielt hat und wohl noch spielen wird.

Zum einen wird natürlich die Entwicklung durch die Möglichkeit, wechselnde Rahmenbedingungen schnell durchspielen zu können, enorm beschleunigt. Zum anderen, und das ist mit der wichtigste Punkt, können die verwendeten Verfahren im Simulator ohne geometrische und elektronische Störeinflüsse untersucht werden. Auf diese Weise ist das mit einem konkreten Rekonstruktionsverfahren theoretisch Machbare zu ermitteln. Ebenso kann der Einfluß der Meßgenauigkeit auf die Genauigkeit des Gesamtsystems untersucht werden, was mit einem realen Panoramasensor nicht möglich ist.

Schließlich stellt die Simulation, solange kein weiteres, hochgenaues Kopfverfolgungssystem zur Verfügung steht, die einzige Möglichkeit dar, die Genauigkeit der Rekonstruktion quantitativ zu bewerten.

7.3 Landmarkenlokalisierung

7.3.1 Farbklassifizierung und Markendetektion

In Abschnitt 6.3.1 wurde bereits berichtet, daß sich die Farben Rot, Grün und Blau in ihren Reintönen am besten als Landmarkenfarben geeignet haben. Sie wurden am zuverlässigsten erkannt. Das mag daran liegen, daß die Kamera gerade für diese drei Farben Sensoren besitzt. Es ist aber eher zu vermuten, daß die gute Detektion darin begründet liegt, daß die drei Farben vom Drucker als Mischfarben aus Cyan, Magenta und Gelb erzeugt werden. Dabei haben sie, verglichen mit anderen Farbtönen, die auf demselben Drucker ausgedruckt wurden, maximale Deckung, erscheinen also am intensivsten.

Letztlich ist die Frage nach den für die Farbklassifizierung am besten geeigneten Landmarkenfarben eine Frage des Zusammenspiels zwischen Drucker und Kamera-CCD. Die optimalen Farben können also nur im Experiment für eine konkrete Kamera bei gegebenem Drucker bestimmt werden. Verzichtet man darauf, die Landmarken komfortabel ausdrucken zu können, bieten sich natürlich noch weitere Möglichkeiten, besonders rein gefärbte Landmarken zu gewinnen.

Insgesamt arbeitet die Landmarkendetektion in ihrer derzeitigen Fassung schnell und zuverlässig. Als besonders erfolgreich ist zu vermerken, daß in allen Testphasen keine einzige Fehldetektion zweiter Art aufgetreten ist. Das heißt, daß auch auf Realaufnahmen keine Objekte für Landmarken gehalten wurden, die keine waren.

7.3.2 Lokalisierungsschritt

Die Konikanpassung führt in den meisten Fällen zu sehr guten Mittelpunktschätzungen. Der Fehler, der durch die Mittelung der beiden potentiellen Kreismittelpunkte entsteht, ist wesentlich kleiner, als er zum Beispiel bei der Verwendung des Pixelschwerpunkts wäre.

Die Lokalisierung der Landmarkenmittelpunkte hingegen ist noch nicht zufriedenstellend. Abhilfe könnte ein neuer Landmarkentyp bringen, dessen Mittelpunkt implizit gekennzeichnet ist. Dabei muß die Marke aber auch bei kleinem Abbildungsmaßstab noch gut zu dekodieren sein. Ein Vorschlag ist in Abbildung 7.1 gezeigt.

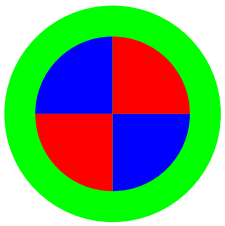

Abbildung 7.1: Alternativer Landmarkenvorschlag

Leider steht aber zu befürchten, daß die Detektion bei komplexeren Markenbildern nicht mehr so zuverlässig funktioniert.

7.4 Rekonstruktionsmethoden

Die Markenidentifizierung hat sich wie zu erwarten als kritischer Punkt herausgestellt. Abhilfe könnte ein intelligenterer Identifizierungsalgorithmus leisten, der mehr Umweltwissen einbringt, um zur korrekten Markenzuordnung zu gelangen. Auf jeden Fall sollte aber die Landmarkenkodierung verbessert werden. Sobald mehr Landmarkentypen zur Verfügung stehen, verliert das Identifizierungsproblem seine Brisanz.

Die eigentliche Positionsrekonstruktion arbeitet sehr zuverlässig und präzise, solange genügend Landmarken im Kamerabild zu sehen sind. Sind dagegen nur wenige Marken sichtbar, die sich zudem in einer extremer Lage zur Kamera befinden, kann es passieren, daß das Verfahren nicht konvergiert. Ob unter diesen Umständen andere Rekonstruktionsverfahren bessere Ergebnisse liefern, muß noch untersucht werden.

7.5 Prädiktion

Die in der Diplomarbeit eingesetzte Prädiktion wurde bewußt sehr einfach gewählt, da die Prädiktion von Kopfpositionen ein sehr umfangreiches Thema ist, das im Rahmen dieser Arbeit nicht ausreichend behandelt werden konnte. Es gibt mit Sicherheit treffendere Modelle als das der linearen Prädiktion, um menschliche Kopfbewegungen vorherzusagen. Insbesondere mag es sinnvoll sein, die menschliche Anatomie zu berücksichtigen. Dieses Thema wurde schon oft bearbeitet, mir ist dennoch kein Modell bekannt, das diese Aufgabe bis jetzt zufriedenstellend löst. Das Hauptproblem bei der Vorhersage der Kopfposition ist die Willkürlichkeit menschlicher Kopfbewegungen.

Es mag daher noch am aussichtsreichsten sein, das in dieser Arbeit verwendete lineare Modell zu einem quadratischen oder kubischen zu erweitern. Auch wäre zu überlegen, die Rotation nicht im Quaternionenraum zu extrapolieren, sondern auf die Roll/Pitch/Yaw-Domäne zurückzugreifen, die der menschlichen Anatomie noch ein wenig näher kommt.

Sehr wichtig für eine gute Prädiktion ist die schnelle Reaktion auf plötzliche Änderungen der Kopfbewegung. Prinzipbedingt wird die Prädiktion nie schneller reagieren können, als es die Gesamtlatenz des Systems erlaubt.

Abhilfe kann hier eine hardwaretechnische Maßnahme bringen: Setzt man zusätzlich zur Panoramakamera ein Gyroskop und Beschleunigungssensoren ein, so läßt sich damit eine hochpräzise Prädiktion realisieren. Dazu wird die relative Lageänderung, wie sie von den zusätzlichen Sensoren erfaßt wird, zur letzten absoluten, kamerabasierten Positionsbestimmung hinzugerechnet.

7.6 Genauigkeit des Gesamtsystems

Natürlich sind Ergebnisse, die auf rein simulierten Daten aufsetzen, immer mit Vorsicht zu genießen. Dennoch liefert diese Arbeit schon gewisse Eckdaten, die auf die zu erwartende Genauigkeit im realen Einsatz schließen lassen.

Prinzipiell kann bemerkt werden, daß die Ortsauflösung des Systems vom durchschnittlichen Abstand zu den Landmarken abhängt: Der translatorische Fehler skaliert linear mit der Entfernung zu den Landmarken. Der rotatorische Fehler bleibt hingegen unverändert.

In unserem $4 \times 5\,\mathrm{m}$ großen Testszenario ist eine translatorische Auflösung im unteren Zentimeterbereich und eine rotatorische Auflösung von etwa 0,2° erreicht worden. Der rotatorische Fehler wirkt sich dabei direkt auf den *Screen error* der Brillen-Einblendung aus. Inwieweit der translatorische Fehler dabei zu Buche schlägt, hängt davon ab, wie weit die eingebleten Objekte von der Benutzerposition entfernt sind.

Diese Genauigkeit genügt bereits für Anwendungen wie den geplanten Einsatz bei der Programmierung mobiler Roboter. Für den medizinischen Einsatz hingegen würde man sich eine etwas genauere tanslatorische Rekonstruktion wünschen. Dies kann zum Teil durch den Einsatz einer höher auflösenden Kamera erreicht werden (siehe Abschnitt 6.6).

Darüberhinaus ist es denkbar, wie in Abschnitt 7.5 geschildert, Beschleunigungssensoren hinzuzunehmen. Sollen diese nicht nur für die Prädiktion verwendet werden, sondern auch direkt in die Positionsrekonstruktion eingehen, bietet sich eine Fusion der beiden Sensordaten an: Man filtere sowohl die aus den Peilungen als auch aus den Beschleunigungsdaten zurückgerechneten Positionen in der Zeit, so daß man von den Peilungsrekonstruktionen die niederfrequenten und von den Beschleunigungsdaten die hochfrequenten Anteile behält. Eine anschließende gewichtete Mittelung beiden Datensätze müßte zu einer erhöhten Genauigkeit führen.

Der Fehler in der Positionsrekonstruktion wird vor allem von Kamerageometrie und -auflösung bestimmt. Der Einfluß geometrischer Kamerafehler ist in der Simulation natürlich ausgeschlossen. Die verbleibenden Fehler rühren also hauptsächlich von der Pixelung der Kamerabilder her. Durch Techniken wie die Konikanpassung wird bereits versucht, aus den diskreten Daten eine möglichst genaue Rekonstruktion der Wirklichkeit zu gewinnen. Eventuell lassen sich aber noch weitere Methoden finden, die Kamerabilder auf Sub-Pixel-Ebene auszuwerten.

Einen kleinen Anteil am Gesamtfehler hat schließlich noch der Umstand, daß zur Zeit noch beide potentiellen Mittelpunktpeilungen gemittelt werden, um zur Landmarkenpeilung zu kommen. Der damit eingebrachte systematische Fehler bewegt sich aber in einem Bereich, der bei den meisten Landmarkenbildern in Sub-Pixel-Größenordnung liegt.

Eventuell wird eine genauere Analyse, welche Kamerabilder zu größeren Rekonstruktionsfehlern führen, weitere Antworten liefern.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Im Rahmen der Arbeit wurde ein neuartiges Kopfverfolgungssystem entwickelt. Das System arbeitet mit künstlichen Landmarken in der Einsatzumgebung und einer Panoramakamera, die auf dem Kopf des Benutzers befestigt wird.

Der konkreten Umsetzung ging ein allgemeiner Entwurf für ein Panoramakamera-basiertes Verfolgungssystem voraus. Die Implementierung hat sich stark an diesem Entwurf orientiert.

Zu Anfang wurde eine Methode entwickelt, parabolische Panoramakameras zu kalibrieren, und anhand einer Panoramakamera für Einzelbildaufnahmen evaluiert. Da anfangs keine Panoramakamera zur Verfügung stand, die Bildfolgen hätte liefern können, mußte eine Kamerasimulation entwickelt werden, die Panoramaaufnahmen einer virtuellen Umgebung generiert. Die Simulation konnte aber insbesondere auch dazu eingesetzt werden, Genauigkeitsangaben über das Verfolgungssystem zu gewinnen.

Parallel zur Simulationsumgebung wurde ein Bildverarbeitungsteil entwickelt, der einen bestimmten Landmarkentyp in photographischen Aufnahmen realer Szenen detektiert und lokalisiert. Die präzise Lokalisierung der Marken wurde durch die Anpassung einer Konik an das Landmarkenbild erreicht.

Mit Hilfe der fertigen Kamerasimulation war es schließlich möglich, den Entwurf des Kopfverfolgungssystems in ein lauffähiges Programm umzusetzen.

Kern des Kopfverfolgers ist ein Algorithmus zur Positionsrekonstruktion aus den Peilungen zu den Landmarken. Es handelt sich dabei um ein iteratives Verfahren, das alternierend geschätzte Position und Orientierung der Kamera an die Messung anpaßt.

Leider ist die benötigte Panoramakamera erst während des Zusammenschreibens der Arbeit geliefert worden. Es liegen daher nur Ergebnisse aus der Simulationsumgebung vor.

Diese Ergebnisse sind vielversprechend und konnten neben der prinzipiellen Funktionsfähigkeit des Systems vor allem zeigen, daß mit der vorhandenen Kameraauflösung eine brauchbare Genauigkeit zu erzielen ist. Die Kameraposition konnte unter Simulationsbedingungen mit einer mittleren Abweichung rekonstruiert werden, die sich im Translatorischen im Bereich von ein bis zwei Zentimetern und im Rotatorischen um etwa 0,2° bewegt.

Die für den realen Einsatz geplante Hardware-Ausstattung hat sich als ausreichend erwiesen, um die Kopfverfolgung in Echtzeit zu bewältigen. Tatächlich wird noch genügend Rechenzeit vorhanden sein, um einfachere Anwendungen parallel laufen zu lassen. Für aufwendige Anwendungen, die zur gleichen Zeit auf dem Wearable-Computer laufen sollen, muß aber zumindest ein weiterer Prozessor eingeplant werden.

8.2 Ausblick

Angesichts der Komplexität des Kopfverfolgungssystems konnten einige Bereiche nicht erschöpfend untersucht werden. So kann zum Beispiel die Methode zur Kamerakalibrierung noch weiter verfeinert werden. Vorschläge finden sich in Abschnitt 7.1. Ferner müssen noch Analysen durchgeführt werden, welche Präzision die mit der realen Kamera gewonnenen Peilungen besitzen.

Auch die Frage, welche Landmarken sich am besten eignen, kann noch weiter untersucht werden. Es gilt, neue Markentypen zu finden, deren Referenzpunkt besser zu lokalisieren ist und für die es mehr Kodierungsmöglichkeiten gibt.

Für den Einsatz mir einer AR-Brille sind außerdem noch eine Reihe von Detailproblemen zu lösen. So muß zum Beispiel noch von der Kamera- auf die Augenposition geschlossen werden. Dies erfordert einen weiteren Kalibrierungsvorgang, in den der Benutzer mit einbezogen werden muß.

Umfangreichen Stoff würde auch die Suche nach einem verbesserten Algorithmus zur Positionsrekonstruktion geben. Es wäre wünschenswert, einen Rekonstruktionsalgorithmus zu haben, der die translatorische und rotatorische Anpassung in einem Schritt erledigt und ein besseres Konvergenzverhalten aufweist. Die größte Herausforderung wäre es aber, den Fall unbekannter Landmarkenpositionen in Angriff zu nehmen: Zur Zeit müssen alle Landmarken vor Inbetriebnahme des Kopfverfolgers genau vermessen werden. Gelänge es, ein Rekonstruktionsverfahren zu entwickeln, das die Lage der Kamera relativ zu Landmarken unbekannter Position bestimmen kann, würde das eine Reihe weiterer Anwendungsfelder eröffnen.

Dann wäre es auch an der Zeit, natürliche Landmarken statt der künstlichen zu verwenden, um damit den Einsatz des Kopfverfolgers im freien, unbekannten Gelände zu ermöglichen.

Die vorliegende Arbeit hat gezeigt, daß aber auch schon mit den bisherigen Mitteln eine praktikable Kopfverfolgung möglich sein muß. Testläufe mit der jetzt verfügbaren Kamera werden schließlich quantitative Aussagen hierzu liefern können.

Anhang A

Konventionen

A.1 Satz

Mathematische Strukturen sind einheitlich gesetzt. Ich folge folgender Konvention:

a, α	Skalar
$\tilde{\boldsymbol{x}}$	homogener Vektor
$\dot{\boldsymbol{q}}$	Quaternion
A	Matrix

A.2 Vektoren

Ich verwende, sofern nicht anders deklariert, ausschließlich Spaltenvektoren. Die Anzahl der Elemente eines Vektors ist seine Dimension. Ein n-dimensionaler Vektor kann auch als Matrix aus $\mathbb{R}^{n\times 1}$ verstanden werden. Ein hochgestelltes $^\top$ wird verwendet, um die Transponierte eines Vektors oder einer Matrix zu bezeichnen.

Im folgenden verwende ich 3-dimensionale Vektoren, um die Vektoroperationen zu erklären.

Es seien $\boldsymbol{a} = (a_1, a_2, a_3)^\top$, $\boldsymbol{b} = (b_1, b_2, b_3)^\top$, $\boldsymbol{c} = (c_1, c_2, c_3)^\top$ und λ ein beliebiger Skalar.

A.2.1 Vektoroperationen

$\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$	Vektornorm (Länge oder Betrag eines Vektors)
$\boldsymbol{a} + \boldsymbol{b} = (a_1 + b_1, a_2 + b_2, a_3 + b_3)^\top$	Vektoraddition
$\lambda\boldsymbol{a} = (\lambda a_1, \lambda a_2, \lambda a_3)^\top$	S-Multiplikation
$\boldsymbol{a} \bullet \boldsymbol{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$	Skalarprodukt
$[\boldsymbol{x}]_i$	i-te Komponente eines Vektors/Quaternions

Anhang B

Koordinatensysteme

B.1 Verwendete Koordinatensysteme

Alle Koordinatensysteme sind, bis auf das Kamerabild-System, Rechtskoordinatensysteme. Es existieren folgende Systeme:

B.1.1 Kamerabildsystem

Parametrisierung: (x_C, y_C)

Sämtliche Pixelkoordinaten liegen im Kamerabildsystem vor. Es handelt sich dabei um ein Linkskoordinatensystem, dessen Ursprung im Pixel*zentrum* der linken oberen Ecke liegt. x geht nach rechts, y nach unten.

Längeneinheit ist die Pixelbreite bzw. -höhe.

B.1.2 Normierte Bildebene

Parametrisierungen: (x_I, y_I), (φ_I, r_I)

Nach der Rektifizierung liegt das Kamerabild in der normierten Bildebene. Das normierte Bildsystem ist ein Rechtskoordinatensystem mit seinem Ursprung in der Symmetrieachse des Spiegelparaboloids. x verläuft nach rechts, y nach oben.

Von der *normierten* Bildebene spreche ich, da hier bereits die Kamerakalibrierung eingegangen ist: Die Skalierung ist so gewählt, daß der Horizontkreis auf den Einheitskreis fällt (vgl. C.3). Umrechnung von der normierten Bildebene in Sehstrahlen ist also möglich, ohne die Kamerakalibrierung explizit zu kennen.

Wahlweise können Punkte der normierten Bildebene auch in Polarkoordinaten angegeben werden. Es gilt dann:

$$x_I = r_I \cos\varphi, \quad y_I = r_I \sin\varphi \tag{B.1}$$

B.1.3 Kugelsystem

Parametrisierungen: (x_S, y_S, z_S), (φ_S, ψ_S)

Sehstrahlen werden bevorzugt im Kugelsystem beschrieben: Ein Sehstrahl wird von seinem Schnittpunkte mit der Einheitskugel repräsentiert. Man kann das Kugelsystem daher auch als *Ray-space* $\mathcal{P}^2$ interpretieren, in dem alle Vektoren normiert auftreten. Die Normierung erfolgte aus Bequemlichkeit und erleichtert einige Operationen der Implementierung.

Das Kugelsystem ist ein Rechtskoordinatensystem mit Ursprung im optischen Zentrum (Fokus des Paraboloids). Seine y-Achse zeigt entlang der Summetrieachse aus dem Spiegelparaboloid heraus. Die x-Achse fällt mit der y-Achse der normierten Bildebene zusammen;

z mit deren x [1].

Naturgemäß sind die Kugelkoordinaten (x_S, y_S, z_S) einheitslos.

In manchen Fällne bietet sich die polare Darstellung (φ_S, ψ_S) an. Sie unterscheidet sich insofern von der üblichen Konvention für Kugelkoordinaten, als daß ψ vom Pol zum Äquator zählt, anstatt in umgekehrter Richtung. Diese Parametrisierung der Kugel ist in unserem Fall geeigneter, da nun die Singularität im „Nordpol" der Kugel, respektive im Scheitel des Spiegelparaboloids, entfällt.

Es gilt

$$\begin{aligned} x_S &= \sin\varphi_S \sin\psi_S \\ y_S &= \cos\psi_S \\ z_S &= \cos\varphi_S \sin\psi_S \,. \end{aligned} \tag{B.2}$$

B.1.4 Paraboloidsystem

Parametrisierung: (x_P, y_P, z_P)

Das Paraboloid- oder auch Spiegelkoordinatensystem findet in der Implementierung nur als Zwischenschritt bei der Koordinatentransformation Verwendung (vgl. Abschnitt B.2). Ferner macht Anhang C von diesem System ausgiebig Gebrauch.

Es entspricht bis auf eine Koordinatenpermutation und die Normierung der Strahlenvektoren dem Kugelsystem: Das Paraboloidsystem enthält die Schnittpunkte der Sehstrahlen mit dem *Spiegelparaboloid* statt mit der Enheitskugel. Punkte in diesem System können also auch als Auftreffpunkte der parallelen Sehstrahlen auf den Spiegel gedeutet werden.

Die x-Achse des Rechtskoordinatensystems liegt diesmal auf der x-Achse der normierten Bildebene; y verläuft mit deren y, und z zeigt mit der Symmetrieachse aus dem Paraboloid heraus.

B.1.5 Weltsystem

Parametrisierung: (x_W, y_W, z_W)

Das Weltsystem sind das allgemeine Bezugsystem für alle beteiligten Koordinatensysteme. Mein Weltkoordinatensystem folgt der OpenGL-Konvention: y zeigt nach „oben", die Horizontale liegt in der zy-Ebene.

B.1.6 Extrinsische Kamerakoordinaten

Entsprechend gilt für eine im Raum positionierte Kamera: y zeigt nach „oben", x nach „rechts"; „vorne" ist in Richtung der negativen z-Achse.

B.2 Umrechnungen

Häufig muß zwischen den verschiedenen Koordinatensystemen umgerechnet werden. Tabelle B.2 gibt die wichtigsten Umrechnungen explizit an.

[1] Dies mag wie eine unnötige Permutation aussehen; auf diese Weise bleibt das System aber rechtshändig und ist vor allem konsistent mit dem Weltkoordinatensystem, in dessen Kontext es meistens eingesetzt wird.

Quell- \ Ziel- system	*Kamerabild* (x_C, y_C)	*Bildebene* (x_I, y_I)	*Bildebene* (φ_I, r_I)	*Kugel* (x_S, y_S, z_S)	*Kugel* (φ_S, ψ_S)	*Paraboloid* (x_P, y_P, z_P)
Kamerabild (x_C, y_C)	id	$\begin{pmatrix} x_I \\ y_I \\ 1 \end{pmatrix} := R \begin{pmatrix} x_C \\ y_C \\ 1 \end{pmatrix}$	–	–	–	–
Bildebene (x_I, y_I)	$\begin{pmatrix} x_C \\ y_C \\ 1 \end{pmatrix} := R^{-1} \begin{pmatrix} x_I \\ y_I \\ 1 \end{pmatrix}$	id	$\varphi_I := \operatorname{atan2}(y_I, x_I)$ $r_I := \sqrt{x_I^2 + y_I^2}$	–	–	$x_P := y_P$ $y_P := \frac{1}{2}(1 - x_P^2 - y_P^2)$ $z_P := x_P$
Bildebene (φ_I, r_I)	–	–	id	–	$\varphi_S := \varphi_I$ $\psi_S := \operatorname{atan2}(r_I, -\frac{1}{2}r_I^2 + \frac{1}{2})$	–
Kugel (x_S, y_S, z_S)	–	$x_I := \lambda z_S$ $y_I := \lambda x_S$ mit $\lambda := \frac{1 - y_S ph}{x_S^2 + z_S^2}$	–	id	$\varphi_S := \operatorname{atan2}(x_S, z_S)$ $\psi_S := \operatorname{atan2}(\sqrt{z_S^2 + x_S^2}, y_S)$	–
Kugel (φ_S, ψ_S)	–	–	$\varphi_I := \varphi_S$ $r_I := \frac{1 - \cos \psi_S}{\sin \psi_S}$	$x_S := \sin \varphi_S \cdot \sin \psi_S$ $y_S := \cos \psi_S$ $z_S := \cos \varphi_S \cdot \sin \psi_S$	id	–
Paraboloid (x_P, y_P, z_P)	–	–	–	$\begin{pmatrix} x_S \\ y_S \\ z_S \end{pmatrix} := \frac{[x_P, y_P, z_P]^\top}{\|[x_P, y_P, z_P]^\top\|}$	–	id

Tabelle B.1. Die wichtigsten Koordiantentransformationen. In nicht aufgeführten Fällen ist es das einfachste, den Umweg über ein anderes System zu nehmen. Die Rektifizierungsmatrix $R \in \mathbb{R}^{3\times 3}$ ist die Abbildung der Kamerabildkoordinaten in die normierte Bildebene. Durch die Normierung des Horizonts auf den Einheitskreis enthält diese Matrix zusätzlich alle Information über die Kamerakalibrierung (siehe Abschnitt C.3).

Anhang C

Kamerakalibrierung

Die Kalibrierung der Panoramakamera, wie sie hier geschildert wird, setzt zunächst einmal voraus, daß die Abbildungseigenschaften der Kamera ideal sind. Das bedeutet, daß

- das Kamerabildsystem (die Anordnung der Pixel-Matrix) orthonormal ist.
- die Sehstrahlen der Kamera wirklich orthogonal auf das Spiegelparaboloid treffen.
- die Symmetrieachse des Paraboloids exakt parallel zur optischen Achse ist.
- das Paraboloid keine Toleranzen aufweist.

Diese Annahmen sind strenggenommen unrealistisch. In der Realität werden Fertigungstoleranzen die Abbildungseigenschaften beeinträchtigen. Darüberhinaus sind die erhältlichen Panoramakameras keine reinen Spiegel-Kameras. An einer Stelle befindet sich immer noch ein Linsensystem im Strahlengang, das wieder eine mehr oder weniger starke sphärische Aberration mit sich bringt.

Dennoch soll zunächst von einer idealen Kamera ausgegangen werden.

Die Kalibrierungsaufgabe besteht darin, für das abgelichtete Spiegelparaboloid das Bild des Scheitelpunkts $\boldsymbol{p}$ zu finden und den Öffnungsparameter a des Paraboloids zu bestimmen. Nach unserer Konvention des Spiegelkoordinatensystems (siehe Abschnitt B.1) besitzt das Spiegelparaboloid die Form

$$\mathcal{P}\colon z = ax^2 + ay^2 - \frac{1}{4a}, \quad a < 0. \tag{C.1}$$

Wie schon angedeutet, nutzt die Kalibrierungsmethode den Umstand aus, daß Strecken bzw. Geraden aus der Umwelt im Kamerabild auf Kreissegmente abgebildet werden. Dieser Zusammenhang soll nun hergeleitet werden.

Betrachten wir die Abbildung einer Geraden. Die (virtuellen) Sehstrahlen vom Brennpunkt zur Geraden bilden eine Ebene. Sei diese Ebene $H = (h_1, h_2, h_3, h_4)^\top$. Was im Kamerabild zu sehen ist, sind die Schnittpunkte der Sehstrahlen mit dem Paraboloid in orthogonaler Projektion. Das Bild der Geraden ist also Teil der projzierten Schnittkurve zwischen H und $\mathcal{P}$. Ich nenne H deshalb auch die *Erzeugendenebene*.

Betrachten wir nun die Projektion dieses Schnitts in die Bildebene unserer Kamera ($\mathcal{P}$ in H einsetzen und $\{x, y\}$ als Bildsystem denken),

$$h_3 a x^2 + h_3 a y^2 + h_1 x + h_2 y + \left(h_4 - \frac{1}{4a} h_3\right) = 0\,. \tag{C.2}$$

C.2 ist eine Gleichung zweiten Grades und beschreibt daher einen Kegelschnitt. In allgemeiner Form notiert lautet sie

$$\underbrace{h_3 a}_{a'} x^2 + \underbrace{0}_{\frac{1}{2}b'} xy + \underbrace{h_3 a}_{c'} y^2 + \underbrace{h_1}_{\frac{1}{2}d'} x + \underbrace{h_2}_{\frac{1}{2}e'} y + \underbrace{\left(h_4 - \frac{1}{4a} h_3\right)}_{f'} = 0. \tag{C.3}$$

Diese Kurve ist laut [1] genau dann ein Kreis, wenn

$$\Delta = \begin{vmatrix} a' & b' & d' \\ b' & c' & e' \\ d' & e' & f' \end{vmatrix} > 0, \quad \delta = \begin{vmatrix} a' & b' \\ b' & c' \end{vmatrix} \neq 0 \quad \text{und} \quad (a' - c') + 4b^2 = 0. \tag{C.4}$$

Dies ist offensichtlich der Fall für $h_4 \neq 0$, womit gezeigt wäre, daß alle Ebenen, die nicht parallel zur Symmetrieachse des Paraboloids liegen, dieses in einer Kurve schneiden, die, in othogonaler Projektion entlang der Symmetrieachse betrachtet, als Kreis erscheint.

Ebene Schnitte, die parallel zur Achse des Paraboloids verlaufen ($h_4 = 0$), erscheinen als Geraden — ein Grenzfall des Kreises, wenn man den Radius gegen ∞ gehen läßt.

Da die Sehstrahlen zu einer Geraden nur eine Halbebene abdecken, deren Rand durch die Symmetrieachse geht, werden Geraden in der Umwelt also auf Halbkreise, Teilgeraden (also Strecken) auf Kreissegmente mit einem Segmentwinkel $\leq 180°$ abgebildet. □

Für die Kamerakalibrierung wird nun das Wissen genutzt, daß die Erzeugendenebene eines Kreissegments[1] durch den Fokus des Paraboloids geht. In unserem Spiegelkoordinatensystem heißt das, daß $h_4 = 0$ ist.

Es sind nun je nach Vorwissen zwei Kalibrierungsansätze denkbar, die ich hier vorstellen möchte.

C.1 Bekannter Scheitelpunkt des Paraboloids

Wenn das Bild $\boldsymbol{p}'$ des Scheitelpunkts $\boldsymbol{p}$ bekannt ist, muß nur noch der Parameter a bestimmt werden. Der Fall, daß wir die Position des Scheitels kennen, ist nicht unrealistisch. Er kann zum Beispiel aus dem kreisförmigen Bild des Spiegelrands bestimmt werden — bei den Kameras, die uns zur Verfügung standen, ist die Kante der verspiegelten Paraboloidkappe im Bild sichtbar, so daß der Mittelpunkt dieses Kreises mit dem Bild des Scheitelpunkts zusammenfallen sollte.

Im Falle des bekannten Scheitelpunkts kann bereits aus dem Bild einer einzigen Geraden auf a geschlossen werden:

Sei also $\boldsymbol{p}'$ bekannt, dann ist damit auch schon den Ursprung unseres Bildkoordinatensystems gegeben. Weiter kennen wir das Bild einer Geraden, deren Erzeugendenebene wir wieder $H = (h_1, h_2, h_3, h_4)^\top$ nennen. Das Geradenbild genügt der Gleichung C.2.

Mit dem Wissen, daß h_4 Null ist, können wir Gleichung C.2 nun nach a auflösen:

$$\begin{aligned} h_3 a^2 x^2 + h_3 a^2 y^2 + h_1 a x + h_2 a y &= \frac{1}{4} h_3 \\ (h_3 x^2 + h_3 y^2) a^2 + (h_1 x + h_2 y) a &= \frac{1}{4} h_3 \\ a^2 + \frac{h_1 x + h_2 y}{h_3 x^2 + h_3 y^2} a &= \frac{1}{4(h_3 x^2 + h_3 y^2)} h_3 \end{aligned}$$

[1] Das Kreissegment muß nach wie vor als Bild einer Strecke im Raum entstanden sein! Es sind andere Kurven im Raum denkbar, die auch auf ein Kreissegment abgebildet werden, deren Sehstrahlen aber nicht in einer Ebene liegen.

$$a^2 + \frac{h_1 x + h_2 y}{h_3 x^2 + h_3 y^2} a = \frac{1}{4x^2 + 4y^2} \tag{C.5}$$

Transformation nach Polarkoordinaten liefert

$$a^2 + \frac{h_1 \cos\varphi + h_2 \sin\varphi}{h_3 r^2} a = \frac{1}{4r^2}\,. \tag{C.6}$$

Erweitern mit r^2 und Substitution mit

$$\alpha := a^2, \quad \lambda := \frac{h_1}{h_3} a, \quad \mu := \frac{h_2}{h_3} a \tag{C.7}$$

führt zu der Form

$$r^2\alpha + \lambda\cos\varphi + \mu\sin\varphi = \frac{1}{4}. \tag{C.8}$$

Demnach genügen im allgemeinen[2] drei Punkte (φ_i, r_i) des Geradenbilds, um das Gleichungssystem

$$\begin{aligned} r^2\alpha + \lambda\cos\varphi_1 + \mu\sin\varphi_1 &= \frac{1}{4} \\ r^2\alpha + \lambda\cos\varphi_2 + \mu\sin\varphi_2 &= \frac{1}{4} \\ r^2\alpha + \lambda\cos\varphi_3 + \mu\sin\varphi_3 &= \frac{1}{4} \end{aligned} \tag{C.9}$$

zu lösen. Für die Bestimmung von a ist anschließend α gemäß C.7 rückzusubstituieren. □

Da wir vermutlich mehr als drei Punkte des sichtbaren Kreissegments kennen werden, liegt es nahe, die Lösung wieder als Kleinste-Quadrate-Optimierung über

$$\begin{pmatrix} r_1^2 & \cos\varphi_1 & \sin\varphi_1 \\ r_2^2 & \cos\varphi_2 & \sin\varphi_2 \\ & \vdots & \\ r_n^2 & \cos\varphi_n & \sin\varphi_n \end{pmatrix} \begin{pmatrix} \alpha \\ \lambda \\ \mu \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \tag{C.10}$$

zu bestimmen.

Die Tatsache, daß diese Kalibrierungsmethode nur ein einziges Geradenbild benötigt, ist Vor- und Nachteil zugleich. Es ist komfortabel, nur eine Strecke im Bild identifizieren zu müssen. Auf der anderen Seite ist das Verfahren gerade dadurch besonders empfindlich gegen sphärische Verzeichnungen durch eine Linsenoptik.

Aus diesem Grunde habe ich auf die Implementierung dieser Variante verzichtet. Stattdessen kommt die folgende Methode zum Einsatz.

[2] System C.9 ist unterbestimmt, wenn die polare Darstellung der drei Punkte linear abhängig ist. In unserem Fall kann das nur bei der Horizontlinie oder dem entarteten Bild einer Geraden auftreten, die die Symmetrieachse schneidet. Im ersten Fall enthält die Horizontlinie bereits die nötige Kalibrierungsinformation (siehe Abschnitt C.3); in letzterem kann die Kalibrierung nicht bestimmt werden.

C.2 Scheitelpunkt des Paraboloids unbekannt

Eine andere Kalibrierungsmethode kommt ohne die Kenntnis des Scheitelpunkts aus. Sowohl $\boldsymbol{p}$ als auch a lassen sich aus dem Bild von mindestens drei Geraden herleiten.

C.2.1 Herleitung

Gegeben sind n Kreissegmente S_i. Wir suchen wieder die Parameter des Spiegelparaboloids.

Legen wir den Fokus des Paraboloids nach $\boldsymbol{p} = (p_1, p_2, 0)^\top$, so hat es die Form

$$\mathcal{P}: \quad z = a(x - p_1)^2 + a(y - p_2)^2 - \frac{1}{4a}. \tag{C.11}$$

Ausmultipliziert erhalten wir

$$\mathcal{P}: \quad z = ax^2 + ay^2 - 2ap_1x - 2ap_2y - \left(\frac{1}{4a} - ap_1^2 - ap_2^2\right). \tag{C.12}$$

Die Bilder der Geraden, die wir im Kamerabild sehen, sind Bilder von Ebenenschnitten mit diesem Paraboloid. Die Erzeugendenebenen H_i schneiden sich wiederum alle im Fokus von $\mathcal{P}$, also in $\boldsymbol{p}$.

Wechseln wir nun in homogene Koordinaten. $\mathcal{P}$ läßt sich dann schreiben als

$$\begin{aligned} \mathcal{P}: \quad ax_1^2 + ax_2^2 - 2ap_1x_1 - 2ap_2x_2 - x_3 - \left(\frac{1}{4a} - ap_1^2 - ap_2^2\right) x_4 &= 0 \qquad \overset{a\neq 0}{\Longleftrightarrow} \\ x_1^2 + x_2^2 - 2p_1x_1 - 2p_2x_2 - \frac{1}{a}x_3 - \left(\frac{1}{4a^2} - p_1^2 - p_2^2\right) x_4 &= 0. \end{aligned} \tag{C.13}$$

Der (noch unbekannte) Fokus liegt in homogenen Koordinaten bei $\tilde{\boldsymbol{p}} = (p_1, p_2, 0, 1)^\top$.

Die affine Transformation

$$T: \quad \tilde{\boldsymbol{x}} \longmapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2p_1 & 2p_2 & \frac{1}{a^2} & \frac{1}{4a^2} - p_1^2 - p_2^2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tilde{\boldsymbol{x}} \tag{C.14}$$

bildet $\mathcal{P}$ auf das Einheitsparaboloid

$$\mathcal{P}_0: \quad z = x^2 + y^2 \tag{C.15}$$

ab, wie anhand C.13 leicht nachgerechnet werden kann.

In der Matrix-Darstellung in Gleichung C.14 ist gut zu erkennen, daß T eingeschränkt auf $\{x_1, x_2, x_4\}$ die Identität ist. Das heißt insbesondere:

> *Die orthogonale Kameraprojektion der $H_i \cap \mathcal{P}$ bleibt unverändert unter T, ist also identisch mit $T(H_i \cap \mathcal{P}) = T(H_i) \cap \mathcal{P}_0$.* (C.16)

Dies ist der Schlüssel zur Kamerakalibrierung aus den sichtbaren Kreissegmenten.

Wir können jetzt nämlich die S_i aus der Kamera-, d.h. xy-Ebene, gemäß Abschnitt 3.1.2 auf das Einheitsparaboloid P_0 projizieren und jeweils eine Ebene H_i' hindurchlegen. Wegen C.16 gilt $H_i' = T(H_i)$. Also schneiden sich die H_i' in

$$\tilde{\boldsymbol{p}}' = T(\tilde{\boldsymbol{p}}) = \begin{pmatrix} p_1 \\ p_2 \\ p_1^2 + p_2^2 + \frac{1}{4a^2} \\ 1 \end{pmatrix} \tag{C.17}$$

Sei $\bigcap_i H_i' = \tilde{\boldsymbol{c}} = (c_1, c_2, c_3, 1)^\top$, dann wissen wir also schon

$$\boldsymbol{p} = \begin{pmatrix} c_1 \\ c_2 \\ 0 \end{pmatrix} . \tag{C.18}$$

Einsetzen in C.17 liefert

$$\begin{aligned} c_1^2 + c_2^2 + \frac{1}{4a^2} &= c_3 \\ a^2(c_1^2 + c_2^2 - c_3) + \frac{1}{4} &= 0 \\ a^2 &= -\frac{1}{4(c_1^2 + c_2^2 - c_3)}, \end{aligned}$$

und, da wir ein $a < 0$ suchen

$$a = -\frac{1}{2\sqrt{c_3 - c_1^2 - c_2^2}} . \tag{C.19}$$

Damit sind die Parameter des Paraboloids gefunden. □

Dieses Kalibrierungsverfahren wurde in der Implementierung eingesetzt. Der besseren Übersicht halber sei noch einmal das „Kochrezept“ für die Kalibrierung bei unbekanntem Scheitelpunkt gegeben.

C.2.2 Zusammenfassung

Gegeben in der xy-Ebene n Kreissegmente $S_1, ..., S_n$, $n \geq 3$, von denen bekannt ist, daß sie Bilder von Strecken sind.

(*i*) Projiziere die S_i parallel zur z-Achse an das Einheitsparaboloid $z = x^2 + y^2$. Lege durch jedes projizierte S_i eine Ebene H_i.

(*ii*) Sei $\boldsymbol{c} = (c_1, c_2, c_3)^\top$ der Schnitt der H_i.

(*iii*) Für die Paraboloidparameter gilt nun

$$\boldsymbol{p} = \begin{pmatrix} c_1 \\ c_2 \\ 0 \end{pmatrix}, \quad a = -\frac{1}{2\sqrt{c_3 - c_1^2 - c_2^2}} . \tag{C.20}$$

C.2.3 Bemerkungen

Es folgen noch ein paar implementierungsspezifische Bemerkungen.

Bestimmung der H_i: Teilschritt (*i*) besteht darin, ein sichtbares Kreissegment S auf das Einheitsparaboloid zu projizieren und eine Ebene H hindurchzulegen.

Ist S als Kreissegment durch drei Punkte $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ und $\boldsymbol{p}_3 \in \mathbb{R}^2$ gegeben, kann H direkt bestimmt werden: H ist die Ebene durch die drei Punkte $\hat{\boldsymbol{p}} = (p_{i,1}, p_{i,2}, p_{i,1}^2 + p_{i,2}^2)$ und kann zum Beispiel gemäß Gleichung 3.3 berechnet werden.

Sind mehr als drei $\boldsymbol{p}_i$ bekannt, das ist zum Beispiel der Fall, wenn S als Pixelbild gegeben ist, sollten dennoch alle diese Punkte aufprojiziert werden. H ist dann als Ausgleichsebene durch die resultierende Punktwolke zu bestimmen.

Wenn Kreismittelpunkt $\boldsymbol{m} = (m_1, m_2)^\top$ und Radius r zu S bekannt sind, ergibt sich H zu

$$H: \quad (-2m_1, -2m_2, 1, m_1^2 + m_2^2 - r^2)^\top \tag{C.21}$$

(vergleiche 3.1.2).

Schnitt der H_i: Für $n = 3$ kann $\boldsymbol{c}$ über Gl. 3.3 bestimmt werden; für $n > 3$ ist eine Ausgleichsrechnung zu betreiben.

Im Dualraum läuft das wieder auf das Problem einer Ausgleichsebene durch n Punkte hinaus. Dabei kann jedem S_i ein Gewicht w_i zugeordnet werden, das bestimmt, wie stark H_i in die Berechnung von $\boldsymbol{c}$ eingeht. Es ist zum Beispiel möglich, $\boldsymbol{c}$ proportional zum Flächeninhalt des Kreissegments (der Fläche, die entsteht, wenn man die beiden Endpunkte des Segments verbindet) oder zu dessen Länge zu wählen.

C.3 Der Horizontkreis

Prinzipiell besteht noch eine weitere Möglichkeit, auf die Geometrie des Spiegelparaboloids zu schließen. Die Sehstrahlen, die das Paraboloid genau senkrecht zur Symmetrieachse verlassen, werden nämlich auf einen scheitelzentrierten Kreis mit Radius

$$R = \frac{1}{2a} \tag{C.22}$$

abgebildet. Dieser Kreis ist der *Horizontkreis*, so genannt, da er das Bild des Horizonts ist, wenn die Kamera exakt waagerecht steht.

Hätte man also das kreisförmige Bild des Horizonts, ließe sich von dessen Ursprung auf $\boldsymbol{p}$ und von dessen Radius mit C.22 direkt auf a schließen.

Leider ist diese Beobachtung eher von akademischem Wert, da man die Kamera nie exakt genug ins Wasser stellen könnte und der Horizont tückischerweise auch bei einer schrägstehenden Kamera auf einen Kreis abgebildet wird — einen anderen als den Horizontkreis.

Der praktische Nutzen des Horizontkreises besteht eher darin, daß mit seiner Hilfe eine Kalibrierung des Spiegelparaboloids gut visualisiert werden kann (siehe Abbildung 5.3). R ist intuitiver zugänglich als a, weshalb in meiner Implementierung das Paraboloid wahlweise statt mit $(\boldsymbol{p}, a)$ auch mit $(\boldsymbol{p}, R)$ parametrisiert werden kann.

Anhang D

Landmarkenlokalisierung über die Konikanpassung

Die Konikanpassung, wie sie in Abschnitt 5.5.2 eingeführt wurde, soll nun ausführlicher vorgestellt werden. Die Problematik wurde dort ausreichend diskutiert, weshalb ich ohne Umschweife mit der Beschreibung der Anpassung beginne.

D.1 Konikanpassung

apx:conicmatchMatch

Es ist eine ein Konik $\mathcal{C} \subset \mathcal{P}^2$ zu finden, die alle Sehstrahlen $\boldsymbol{x}_i = (x_i, y_i, z_i)^\top$ zum Rand der zentralen Kreisscheibe enthält. Zur Zeit verwende ich einen sehr einfachen Kleinste-Quadrate-Ansatz. In [25] wird aber gezeigt, daß dies gerade bei Stützdaten, die nur in einem kleinen Ausschnitt gegeben sind (das tritt bei uns im Falle einer halb abgeschatteten Landmarke auf), zu ungenauen Ergebnissen führt. Fürs erste erfüllt das Verfahren dennoch seinen Zweck.

Kleinste-Quadrate-Anpassung

$\mathcal{C}$ wird durch die implizite Gleichung:

$$\mathcal{C}: \quad ax^2 + 2bxy + cy^2 + 2dxz + 2eyz + fz^2 = 0, \tag{D.1}$$

beschrieben, in Vektorschreibweise

$$\mathcal{C}: \quad \boldsymbol{x}^\top C \boldsymbol{x} = 0 \quad \text{mit} \quad C = \begin{pmatrix} a & b & d \\ b & c & e \\ d & e & f \end{pmatrix}. \tag{D.2}$$

Setzt man zum Beispiel $f = 1$, so bestimmen sich die Parameter von C über

$$\begin{pmatrix} x_1^2 & 2\,x_1y_1 & y_1^2 & 2\,x_1z_1 & 2\,y_1z_1 \\ x_2^2 & 2\,x_2y_2 & y_2^2 & 2\,x_2z_2 & 2\,y_2z_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & 2\,x_ny_1 & y_n^2 & 2\,x_nz_n & 2\,y_nz_n \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} -z_1^2 \\ -z_2^2 \\ \vdots \\ -z_n^2 \end{pmatrix}. \tag{D.3}$$

Das System ist im allgemeinen überbestimmt. Lösung, zum Beispiel mit der Technik der Pseudo-Inversen, liefert $\mathcal{C}$.

Wie schon angedeutet, ist diese Form der Anpassung nicht ideal. Das rührt im wesentlichen daher, daß hierbei der *algebraische Fehler* und nicht der *geometrische* minimiert wird. Aus

diesem Grund hat leider auch die Enscheidung, welche Variable gleich Eins gesetzt wird, Einfluß auf das Ergebnis.

Die beste Wahl für die festgehaltene Variable hängt von der ungefähren Haupt-Achsenrichtung ab, in der die Landmarke gesehen wird: Die Landmarkenlokalisierung liefert bereits eine erste Schätzung $\boldsymbol{d}$, für die Richtung, in der die Marke liegt. Nun betrachte man die betragsmäßig größte Komponente von $\boldsymbol{d}$. Ist es die x-, y-, bzw. z-Komponente, ist jeweils a, c oder f auf Eins zu setzen.

Alternativ kann auch eine Koordinatenpermutation vorgenommen werden, so daß die betragsmäßig größte Komponente von $\boldsymbol{d}$ zur z-Komponente wird.

D.2 Potentielle Kreismittelpunkte

Ist $\mathcal{C}$ ermittelt, sind die möglichen Mittelpunkte der zentralen Landmarkenkreisscheibe gesucht. Der Ansatz sieht wie folgt aus:

Ansatz

Man finde die Menge $\mathcal{H}$ der Ebenen, deren Schnitt mit $\mathcal{C}$ Kreisform hat. In einer Ebene $H \in \mathcal{H}$ muß sich die Landmarke befinden. Der Mittelpunkt von $H \cap \mathcal{C}$ ist der Landmarkenmittelpunkt.

$\mathcal{H}$ besteht aus zwei Ebenenscharen $\mathcal{H}_i = \{H : H \parallel H_i,\ H \cap H_i = \emptyset\}$, $i \in \{1, 2\}$, die von zwei Ebenen H_1 und H_2 erzeugt werden, die durch den Ursprung gehen[1].

Zur Bestimmung der potentiellen Kreismittelpunkte und der zugehörigen Normalenvektoren der Landmarke sind also diese zwei Ebenen zu finden.

Normalform der Konik

Nun interessiert uns die Diagonalform der Konik, die wir durch Rotation unseres Koordinatensystem mit S aus C erhalten:

$$C_0 = S^\top C S, \quad S^\top S = I, \quad C_0 \text{ diagonal.} \tag{D.4}$$

S erhält man durch die Bestimmung des Eigensystems, zum Beispiel mit dem JACOBI-Verfahren[2]. Haben alle Eigenwerte das gleiche Vorzeichen, so handelt es sich um eine imaginäre Konik. Dieser Fall kann nur auftreten, wenn die Sehstrahlen, die in den Anpassungsvorgang eingegangen sind, beim besten Willen nicht auf einer Konik lagen[3].

OBdA seien also zwei der Eigenwerte positiv. (Dies kann erzwungen werden, indem C negiert wird — was keinen Einfluß auf $\mathcal{C}$ hat.)

Zudem fordere ich, daß die Eigenwerte auf der Diagonalen in absteigender Sortierung vorliegen. Auch diese Operation ist erlaubt: Sie dürfen frei permutiert werden, solange man dabei auch die Spalten von S mitpermutiert.

Sind diese Bedingungen sichergestellt, erhält man also

$$C_0 = S^\top C S = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}, \quad \alpha \geq \beta > 0 > \gamma. \tag{D.5}$$

C_0 beschreibt die Konik $\mathcal{C}_0 = S^{-1}(\mathcal{C})$.

[1] Ohne Beweis. Der interessierte Leser sei auf [26] verwiesen.

[2] Es ließe sich auch auf das HOUSEHOLDER-Verfahren oder, da wir wissen, daß S orthonormal ist, auf die Singulärwertzerlegung zurückgreifen. Dieser Aufwand wäre aber bei einer 3×3-Matrix nicht gerechtfertigt.

[3] In diesem Falle wird die Kreismittelpunktsuche einfach abgebrochen, und die Landmarkenlokalisierung greift auf den Pixelschwerkpunkt der Marke zurück oder verwirft die Landmarke komplett.

$\mathcal{C}_0$ ist ein gerader Ellipsenkegel um die z-Achse, dessen Spitze im Nullpunkt sitzt und dessen elliptischer Schnitt mit den Ebenen $z = \pm\sqrt{-\gamma}$ jeweils Hauptachsen mit den Längen $\sqrt{\alpha}$ und $\sqrt{\beta}$ hat. Ich nenne $\mathcal{C}_0$ die *Normalform* der Konik $\mathcal{C}$.

Schnittebenen...

Nun sind die Ebenenscharen gesucht, deren Schnittkurven mit der normalisierten Konik $\mathcal{C}_0$ kreisförmig sind. Wie oben erwähnt, genügt es, die zwei Repräsentantenebenen H_1 und H_2 zu finden.

Von den beiden Ebenen wissen wir, daß sie durch den Ursprung gehen. Wie man sich leicht veranschaulichen kann, müssen sie ferner parallel zur Hauptachse des elliptischen xy-Querschnitts von $\mathcal{C}_0$ liegen, um einen kreisförmigen Schnitt mit $\mathcal{C}_0$ zu erzielen. Für die Suche nach den Erzeugenden setze ich also eine Ebene H_t mit Parameter t an,

$$H_t\colon\ (t\ 0\ 1)\ \boldsymbol{x} = 0\,. \tag{D.6}$$

Parallelen zu H_t schneiden $\mathcal{C}_0$ in einem Kreis, wenn der Schnitt von $\mathcal{C}_t = R_t^{-1}(\mathcal{C}_0)$ mit der Ebene $z = 1$ kreisförmig ist, wobei

$$R_t = \frac{1}{\sqrt{t^2+1}} \begin{pmatrix} 1 & 0 & -t \\ 0 & \sqrt{t^2+1} & 0 \\ t & 0 & 1 \end{pmatrix} \tag{D.7}$$

die Rotation darstellt, die die xy-Ebene in H_t überführt.

Es gilt $\mathcal{C}_t = R_t^{-1}(\mathcal{C}_0)$, also

$$\mathcal{C}_t\colon\ C_t = {R_t}^{\top} C_0 R_t = \frac{1}{t^2+1} \begin{pmatrix} \alpha+\gamma t^2 & 0 & -\alpha t+\gamma t \\ 0 & \beta(t^2+1) & 0 \\ -\alpha t+\gamma t & 0 & -\alpha t^2+\gamma \end{pmatrix}. \tag{D.8}$$

Diese Konik schneidet $z = 1$ in einem Kreis, wenn[4]

$$(a'-c') + 4b'^2 = 0 \qquad \text{mit} \quad C_t = \begin{pmatrix} a' & b' & d' \\ b' & c' & e' \\ d' & e' & f' \end{pmatrix}. \tag{D.9}$$

Einsetzen liefert

$$\begin{aligned} \frac{1}{t^2+1}(\alpha+\gamma t^2-\beta(t^2+1)) &= 0 \qquad \Longleftrightarrow \\ (\gamma-\beta)t^2+(\alpha-\beta) &= 0 \qquad \Longleftrightarrow \\ t^2 &= \frac{\alpha-\beta}{\beta-\gamma}\,. \end{aligned} \tag{D.10}$$

Die beiden Lösungen für t liefern, eingesetzt in H_t, die gesuchten H_i.

... und zurück

Für die Landmarkenlokalisierung werden nun die beiden Sehstrahlen $\boldsymbol{m}_{1/2}$ benötigt, auf denen die Mittelpunkte der Schnittkreise zu den Ebenenscharen $\mathcal{H}_i$ liegen.

[4] Eingeschränkt auf $z = 1$ ist die Konikgleichung identisch zur impliziten Form eines Kegelschnitts. Dieser Kegelschnitt muß also ein Kreis sein, was zum Ansatz D.9 führt (vgl. [1]).

Im lokalen Koordinatensystem (nach Anwendung von $R_t^{-1}S^{-1}$) geht jeweils ein Sehstrahl durch

$$\boldsymbol{m}_t = \begin{pmatrix} m_x \\ 0 \\ 1 \end{pmatrix} \qquad \text{mit} \qquad m_x = \frac{1}{\delta}\begin{vmatrix} b' & c' \\ d' & e' \end{vmatrix}, \quad \delta = \begin{vmatrix} a' & b' \\ b' & c' \end{vmatrix}. \tag{D.11}$$

Eliminieren von t liefert

$$m_x = \pm\frac{\alpha - \gamma}{\alpha\sqrt{\frac{\beta-\gamma}{\alpha-\beta}} + \gamma\sqrt{\frac{\alpha-\beta}{\beta-\gamma}}}\,. \tag{D.12}$$

Im Weltsystem gilt dann

$$\boldsymbol{m}_{1/2} = SR_t\boldsymbol{m}_t\,. \tag{D.13}$$

Für die Landmarkenidentifizierung können schließlich noch die Normalen der H_i von Interesse sein, denn sie entsprechen der Normalen der Landmarkenscheibe. Für diese gilt in Weltkoordinaten

$$\boldsymbol{n}_{1/2} = S(H_{1/2})^\top = \left[(t\ 0\ 1)\ S^\top\right]^\top = S\begin{pmatrix} \pm\sqrt{\frac{\alpha-\beta}{\beta-\gamma}} \\ 0 \\ 1 \end{pmatrix}. \tag{D.14}$$

Anhang E

Positionsrekonstruktion

Die allgemeine Vorgehensweise bei der Rekonstrukion der Kameralage im Raum wurde bereits in 5.5.3 vorgestellt. Hier werden die beiden Anpassungsschritte, die dabei Verwendung finden, genauer beschrieben.

In beiden Fällen geht es darum, eine konkrete Messung $\mathcal{Y}$ an ein Modell $\mathcal{X}$ der Landmarkenverteilung anzupassen.

In beiden Anpassungsschritten ist nun eine Positions-, bzw. Orientierungs-Hypothese gesucht, so daß die n Peilungen $\boldsymbol{y}_i$ möglichst exakt durch die bekannten Landmarken verlaufen.

E.1 Translatorische Anpassung

Im translatorischen Anpassungschritt wird die hypothetische Kameraposition bei gegebener Orientierung so verschoben, daß die euklidischen Abstandsquadrate zwischen den Peilstahlen und den bekannten Landmarkenpositionen in der Summe minimiert werden[1].

Seien $\boldsymbol{y}_i \in \mathbb{R}^3$ die Peilungen, die von Landmarkenlokalisierung und -identifizierung geliefert wurden, und $\boldsymbol{x}_i$ die bekannten Positionen der zugehörigen Landmarken. Zunächst wähle ich zu jedem $\boldsymbol{y}_i$ zwei Zeilenvektoren

$$\boldsymbol{u}_i, \boldsymbol{v}_i \in \mathbb{R}^{1\times 3}, \quad \{\boldsymbol{u}_i, \boldsymbol{v}_i\} \perp R(\boldsymbol{y}_i), \quad \|\boldsymbol{u}_i\| = \|\boldsymbol{v}_i\| = 1, \tag{E.1}$$

wobei R die gegebene Kameraposition ist. Der Abstand eines Peilstrahls $\boldsymbol{y}_i$ von einer Landmarkenposition $\boldsymbol{x}$ beträgt nun

$$d(\boldsymbol{y}_i, \boldsymbol{x}) = \sqrt{(\boldsymbol{u}_i\boldsymbol{x})^2 + (\boldsymbol{v}_i\boldsymbol{x})^2}. \tag{E.2}$$

Lösung des Systems

$$\begin{pmatrix} \boldsymbol{u}_1 \\ \boldsymbol{v}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{u}_n \\ \boldsymbol{v}_n \end{pmatrix} \boldsymbol{p} = \begin{pmatrix} \boldsymbol{u}_1\boldsymbol{x}_1 \\ \boldsymbol{v}_1\boldsymbol{x}_1 \\ \boldsymbol{u}_2\boldsymbol{x}_2 \\ \boldsymbol{v}_2\boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{u}_n\boldsymbol{x}_n \\ \boldsymbol{v}_n\boldsymbol{x}_n \end{pmatrix} \tag{E.3}$$

per Kleinste-Quadrate-Optimierung liefert daher die gesuchte Position $\boldsymbol{p} \in \mathbb{R}^3$.

[1] Tatsächlich wäre es sinnvoller, statt der Abstände die Winkelabweichung der Peilungen zu minimieren. Für Marken, deren Abstand zur Kamera in der gleichen Größenordnung liegt, führt das aber zu vergleichbaren Ergebnissen.

Die Translatorische Anpassung setzt voraus, daß für die Kameraorientierung bereits eine sinnvolle Schätzung vorliegt. Das ist der Grund, warum bei der Positionsrekonstruktion die rotatorische Anpassung zuerst durchgeführt wird.

Allgemein ist diese Methode etwas anfällig gegen lokale Minima. In den wenigen Fällen, in denen die Positionsrekonstruktion keine Konvergenz gezeigt hat, war die translatorische Anpassung verantwortlich zu machen.

E.2 Rotatorische Anpassung

Für die rotatorische Anpassung habe ich das Verfahren nach [28] aufgegriffen, das auch Teil des ICP-Algorithmus aus [29] ist. Eine kurze Übersicht über das Verfahren liefert [23]. Eine weitere schöne Darstellung kann auch in [30] gefunden werden.

Der eigentliche ICP-Algorithmus, der dazu gedacht ist, dreidimensionale Flächen(-stücke) aneinander anzupassen, ist für unser Problem nicht geeignet. Dafür konnte sein rotatorischer Anpassungsteil direkt übernommen werden. Er sei an dieser Stelle gleich in der Form wiedergegeben, in der er in diese Arbeit Einzug gehalten hat.

Wir wollen zu einer festen Position eine Orientierungshypothese für die Kamera finden, so daß gemessene Peilungen $\boldsymbol{y}_i$ und bekannte Landmarken zueinander passen. Dazu werden zunächst exakte Peilungen $\boldsymbol{x}_i$ zu den bekannten Landmarken berechnet, wie sie eine Kamera in „Nullstellung“ sehen würde. Anschließend wird eine Rotation gesucht, die alle $\boldsymbol{y}_i$ und $\boldsymbol{x}_i$ zur Deckung bringt. Die inverse Rotation ist die gesuchte Orientierung.

Anfangs bestimme man die Richtungen $\boldsymbol{x}_i$, unter denen die bekannten Landmarken, die von den $\boldsymbol{y}_i$ angepeilt werden, erscheinen würden, wäre die Kamerarotation auf Null gesetzt.

Nun sei R die gesuchte Rotation, beschrieben vom Einheitsquaternion $\dot{\boldsymbol{r}}$. Dann liegen $\boldsymbol{x}_i$ und $\boldsymbol{y}_i$ bestmöglich aufeinander, wenn

$$\sum_{i=1}^{n} \boldsymbol{x}_i \bullet R(\boldsymbol{y}_i) \tag{E.4}$$

maximal wird. Wird die Rotation mit Einheitsquaternionen vorgenommen, so schreibt sich das als

$$\sum_{i=1}^{n} \dot{\boldsymbol{x}}_i \bullet \dot{\boldsymbol{r}}\dot{\boldsymbol{y}}_i\dot{\boldsymbol{r}}^*. \tag{E.5}$$

Dabei wurden die $\boldsymbol{x}_i$ in Quaternionen $\dot{\boldsymbol{x}}_i = (0, x_{i,1}, x_{i,2}, x_{i,3})^\top$ umgeschrieben, die $\boldsymbol{y}_i$ analog. Mit Hilfe der Eigenschaften des Einheitsquaternions, wie sie in Abschnitt 3.1.3 beschrieben wurden, und unter Ausnutzung der Kommutativität des Skalarprodukts läßt sich Gleichung E.5 nach

$$\sum_{i=1}^{n} (\dot{\boldsymbol{x}}_i\dot{\boldsymbol{r}}) \bullet (\dot{\boldsymbol{r}}\dot{\boldsymbol{y}}_i) \tag{E.6}$$

umformen. Bedient man sich für die Quaternionenmultiplikation der Matrix-Vektor-Schreibweise nach Gleichung 3.13 bzw. 3.14, erhält man

$$\sum_{i=1}^{n} (X_i\boldsymbol{r}) \bullet (\overline{Y}_i\boldsymbol{r}) \tag{E.7}$$

und daraus

$$\sum_{i=1}^{n} \boldsymbol{r}^\top X_i^\top \overline{Y}_i\boldsymbol{r}\,. \tag{E.8}$$

Da $\boldsymbol{r}$ unabhängig von i ist, darf es aus der Summe gezogen werden. Wir bekommen dann

$$\boldsymbol{r}^\top \underbrace{\left(\sum_{i=1}^{n} X_i^\top \overline{Y}_i \right)}_{N} \boldsymbol{r} \,. \tag{E.9}$$

Gesucht ist also ein Einheitsquaternion $\dot{\boldsymbol{r}}$, das Gleichung E.9 maximiert. Als symmetrische Matrix hat $N \in \mathbb{R}^{4\times 4}$ vier reelle Eigenwerte. Die zugehörige Eigenbasis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_4\}$ ist orthonormal. Als Quaternionen interpretiert bilden die $\dot{\boldsymbol{e}}_i$ eine Basis des Quaternionenraums. Jedes Quaternion läßt sich also als

$$\dot{\boldsymbol{q}} = \alpha_1 \dot{\boldsymbol{e}}_1 + \alpha_2 \dot{\boldsymbol{e}}_2 + \alpha_3 \dot{\boldsymbol{e}}_3 + \alpha_4 \dot{\boldsymbol{e}}_4, \quad \alpha_i \in \mathbb{R} \tag{E.10}$$

schreiben. Für ein Einheitsquaternion gilt ferner

$$\dot{\boldsymbol{q}} \cdot \dot{\boldsymbol{q}} = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2 = 1 \,. \tag{E.11}$$

Daraus folgt, daß alle α_i betragsmäßig ≤ 1 sind.

Setzt man $\boldsymbol{q}$ für $\boldsymbol{r}$ an, erhält man durch schrittweises Ausmultiplizieren der Gleichung E.9

$$N \cdot \boldsymbol{q} = \alpha_1\lambda_1\boldsymbol{e}_1 + \alpha_2\lambda_2\boldsymbol{e}_2 + \alpha_3\lambda_3\boldsymbol{e}_3 + \alpha_4\lambda_4\boldsymbol{e}_4$$

und

$$\boldsymbol{q}^\top \cdot N \cdot \boldsymbol{q} = \alpha_1^2\lambda_1 + \alpha_2^2\lambda_2 + \alpha_3^2\lambda_3 + \alpha_4^2\lambda_4 \,. \tag{E.12}$$

Sei oBdA[2] λ_1 der größte Eigenwert. Dann kann der Ausdruck in Gleichung E.12 nie größer werden als λ_1. Mit E.11 wird er demnach maximal für $\alpha_1 = \pm 1$ und $\alpha_2 \ldots \alpha_4 = 0$.

Damit wissen wir, daß der Eigenvektor zum betragsmäßig größten Eigenwert von N das gesuchte optimale Rotationsquaternion ist. Das Eigensystem kann wie in Abschnitt D.2 wieder wahlweise mit dem JACOBI-Verfahren, über das HOUSEHOLDER-Verfahren, oder mit Hilfe einer Singulärwertzerlegung bestimmt werden. Für die 4×4-Matrix N hat sich dabei das JACOBI-Verfahren als das schnellste erwiesen.

An diesem Verfahren ist gut zu sehen, daß der Weg über den Quaternionenraum klare Vorteile gegenüber der Rotationsbeschreibung in Form einer Matrix hat: Die Einheitslänge der Quaternionen ist einfacher zu garantieren als die Orthonormalität von Matrizen.

Die vorgestellte Lösung des Rekonstruktionsproblems war ursprünglich eher eine „Verlegenheitslösung“. Es erschien mir ungünstig, den Anpassungsprozeß in rotatorische und translatorische Einzelanpassungen aufzuteilen. Wenn man in der Lage wäre, beide Aspekte in einem gemeinsamen Schritt zu optimieren, hätte man sich eventuell das Iterieren sparen können.

Deshalb habe ich immer wieder nach Möglichkeiten gesucht, Position und Orientierung in einer gemeinsamen Domäne[3] zu optimieren, bin aber zu keinem zufriedenstellenden Ergebnis gekommen.

Schließlich hat sich das hier vorgestellte Verfahren dann aber doch als geeignet erwiesen und legt eine recht gute Konvergenzgeschwindigkeit an den Tag.

[2] Wie in schon in D.2 erläutert, dürfen die Eigenwerte umsortiert werden, solange das Eigensystem mitpermutiert wird.

[3] Konkret habe ich versucht, die Peilungen als Plücker-Geraden darzustellen, um die Anpassung dann innerhalb der Plücker-Domäne durchzuführen.

Anhang F

Farbklassifizierung

Die Farbklassifizierung wurde in Abschnitt 5.5.2 kurz vorgestellt. Ich möchte hier auf ihre Implementierungsdetails eingehen.

Die Aufgabe der Farbklassifizierung ist es, eine gegebene Pixelfarbe (r, g, b) den Klassen schwarz, weiß, unbekannt oder einer der interessierenden Farbklassen zuzuordnen. Die Farbklassen sind jeweils durch einen Klassenrepräsentanten $(r_{\mathsf{klasse}}, g_{\mathsf{klasse}}, b_{\mathsf{klasse}})$ definiert. Eine Farbe wird einer Klasse zugerechnet, wenn sie durch Beleuchtungseffekte aus dem Repräsentanten hervorgegangen sein kann.

Die Unempfindlichkeit gegenüber Veränderungen der Lichtverhältnisse hatte ich ursprünglich durch eine Konvertierung der Bilddaten in das HSV-Format (siehe Abschnitt 3.4.1) erreicht: Unterschiede in Helligkeit und Kontrast, wie sie durch unterschiedliche Beleuchtungswinkel und Abschattungen entstehen, wirken sich im HSV-System nur auf *Saturation* und *Value* aus. *Hue*, der Winkel im Farbkreis, bleibt theoretisch unverändert. Eine Pixel wurde also einer Farbklasse zugeordnet, wenn dessen *Hue* bis auf ein Epsilon dem *Hue* des Klassenrepräsentanten entsprochen hat.

Leider erwies sich die RGB→HSV-, bzw. YIQ→HSV-Konversion als zu zeitaufwendig. Darum habe ich versucht, die eben geschilderte Klassifizierung innerhalb des RGB-Würfels durchzuführen[1]. Exakt das gleiche Klassifizierungsverhalten in RGB nachzubilden, hätte aber nur wieder eine implizite RGB→HSV-Konversion bedeutet; es wäre nichts gewonnen gewesen. Daher wird die HSV-Klassifizierung nur näherungsweise nachgebildet.

Grundidee ist es, die unterschiedlichen Beleuchtungseffekte als Linearkombination zwischen Schwarz, Weiß und dem Farbrepräsentanten zu deuten. Damit lägen alle Farbwerte der gleichen Farbklasse in einer Ebene des RGB-Würfels, die durch diese drei Farben geht. Verbietet man einen negativen Beitrag des Farbrepräsentanten (das entspräche der Beleuchtung durch die Komplementärfarbe), bleibt nur noch die Halbebene, die durch die *Schwarz-Weiß-Achse*, der Geraden durch die Farbwerte $(0, 0, 0)$ und $(r_{\mathsf{weiß}}, g_{\mathsf{weiß}}, b_{\mathsf{weiß}})$, begrenzt ist[2].

An die Stelle des *Hue*-Epsilons tritt nun ein Epsilon-Keil, dessen Kante mit der Schwarz-Weiß-Achse zusammenfällt: Liegt eine Farbe innerhalb dieses Keils, wird sie der Klasse zugeordnet.

Diese Klassifizierung kommt der im HSV-System schon sehr nahe. Der Hauptunterschied liegt in der Epsilontik: Der Epsilon-Keil bildet das *Hue*-Epsilon nur unvollständig nach.

Eine Farbe wird dann als schwarz angesehen, wenn sie in der Ecke des RGB-Würfels

[1] Ich spreche jetzt nur noch vom RGB-Würfel, obwohl die Klassifizierung je nach Kameradatenstrom auch auf YIQ-Daten stattfinden kann. Dies ist zulässig, da das YIQ-System durch eine Basistransformation aus dem RGB-System hervorgeht (siehe Abschnitt 3.4.1) und im folgenden nur lineare Klassifikatoren zum Einsatz kommen, die ebenso der Basistransformation unterworfen werden könnnen.

[2] Weiß wird nicht automatisch als $(1, 1, 1)$ angenommen. Der alternative Weiß-Repräsentant (r_w, g_w, b_w) kann für einen eigenen Weißabgleich verwendet werden.

liegt, die von der sogenannten *Schwarz-Ebene*, einer Ebene, die senkrecht zur Schwarz-Weiß-Achse liegt, begrenzt wird. weiß ist alles, was einen Mindestabstand zur Schwarz-Weiß-Achse unterschreitet und nicht schwarz zugerechnet wird.

Der Vorteil dieser Vorgehensweise liegt darin, daß die Farbklassifizierung jetzt weitestgehend linearisiert ist, zur Bestimmung der Farbklasse also hauptsächlich nur noch Skalarprodukte eines $(r, g, b, 1)$-Quadrupels mit Ebenenvektoren berechnet werden müssen[3]. Im Gegensatz zur HSV-Konversion mit anschließender Klassifizierung, die eine hohe Anzahl bedingter Sprünge enthält, die auf heutigen Prozessoren zu Geschwindigkeitseinbrüchen führen, läßt sich diese Klassifizierung schnell berechnen.

Um die Farbklassifizierung noch weiter zu beschleunigen, wird der RGB-Würfel in einem Vorverarbeitungsschritt in grobe Voxel unterteilt. Enthält ein Voxel nur RGB-Tripel der selben Klasse, wird das in einer Tabelle festgehalten. Bevor eine unbekannte Farbe die lineare Klassifizierung durchläuft, wird erst in dieser Tabelle nachgeschaut, ob die Farbe in ein eindeutig klassifiziertes RGB-Voxel fällt. In diesem Fall kann die Klassifizierung noch weiter abgekürzt werden.

[3] Es ist daher eine naheliegende Optimierung, die Klassifizierung für die SIMD-Einheiten moderner Prozessoren zu optimieren. Diesen Schritt habe ich selbst nicht durchgeführt; er ist aber dennoch empfehlenswert, da die Farbklassifizierung durch die große Anzahl zu klassifizierender Pixel einen nennenswerten Beitrag zur Gesamtrechenzeit des Systems beisteuert (siehe Kapitel 6).

Anhang G

Zusammenhangskomponenten in Binärbildern

Für die Landmarkenlokalisierung wird ein Verfahren benötigt, das zusammenhängende Pixelkomponenten gleicher Farbe findet (vgl. Abschnitt 5.5.2). Von dem Verfahren wird verlangt, tolerant gegen kleine Lücken zwischen zwei Komponenten zu sein. Die Toleranzgrenze soll abhängig von der Größe der beteiligten Komponenten sein.

Aus diesem Grunde war es nicht möglich, die Standard-Vorgehensweise anzuwenden und das Bild vor der Komponentensuche zu *schließen*. Statt dessen setze ich ein anderes, einfaches Verfahren ein, daß ich nun vorstellen möchte.

G.1 Das Verfahren

Der Komponentensuche ist eine Farbklassifizierung vorgeschaltet, so daß formal der Fall eines Binärbilds behandelt werden kann. Es gibt *gesetzte* und *nicht gesetzte* Pixel, und Aufgabe des Algorithmus ist es nun, eine *Abschätzung* der Zusammenhangskomponenten gesetzter Pixel zu finden.

Von einer Abschätzung ist die Rede, da der Algorithmus, indem er kleinere Pixellücken überbrückt, eher mehr Pixel als notwendig einer Komponente zuweisen soll. Dieses Verhalten ist in unserem Fall ausdrücklich erwünscht.

G.1.1 Definitionen

Für die Ermittlung der Zusammenhangskomponenten seien Pixel ausnahmsweise[1] als flächige Rechtecke aufgefaßt; wir verwenden ein Koordinatensystem, dessen Ursprung in einer Pixel*ecke* liegt. Mit dieser Konvention wird definiert:

Komponente: Eine Komponente ist eine Menge gesetzter Pixel, die nicht notwendigerweise zusammenhängend sein muß.

Tatsächliche Bounding box: Die *tatsächliche Bounding box* (BB) einer Komponente ist das kleinste achsenparallele Rechteck, das alle ihre Pixel vollständig enthält.

Erweiterte Bounding box: Die *erweiterte Bounding box* (BB_x) einer Komponente ist um deren tatsächliche BB zentriert und besitzt ihr gegenüber etwas größere Ausmaße. Breite w_x

[1] In den anderen Teilen der Arbeit werden Pixel des Kamerabilds als Abtast*punkte* aufgefaßt. Entsprechend liegt der Ursprung des Kamerabild-Koordinatensystem dort in den Pixelmittelpunkten.

und Höhe h_x von BB_x berechnen sich aus den Maßen w, h von BB gemäß

$$w_x := c_w + a_w w, \quad h_x := c_h + a_h h, \qquad c_w, c_h > 0, \quad a_w, a_h > 1\,. \tag{G.1}$$

Verschmelzen von Komponenten: Zwei Komponenten k_1 und k_2 werden verschmolzen, indem ihre Pixelmengen vereinigt werden. Dabei entsteht eine neue Komponente $k := k_1 \oplus k_2$, deren tatsächliche Bounding box das minimale Rechteck ist, das die tatsächlichen BBs der ursprünglichen Komponenten enthält. Es gilt

$$\begin{aligned} w_x(k) - w(k) &\geq \max\{w_x(k_1) - w(k_1), w_x(k_2) - w(k_2)\} \qquad \text{und} \\ h_x(k) - h(k) &\geq \max\{h_x(k_1) - h(k_1), h_x(k_2) - h(k_2)\}\,. \end{aligned} \tag{G.2}$$

G.1.2 Der Algorithmus

Mit diesen Definitionen ist der eigentliche Algorithmus schnell formuliert:

Algorithmus G.1: Abschätzung der Zusammenhangskomponenten in Binärbildern

Initial bildet jedes gesetzte Pixel eine Komponente.

Sei $\mathcal{K}$ die Menge dieser Ein-Pixel-Komponenten.

Solange $\exists k_1, k_2 \in \mathcal{K}\colon \mathrm{BB}_x(k_1) \cap \mathrm{BB}_x(k_2) \neq \emptyset$

$\mathcal{K} := \mathcal{K} \backslash \{k_1, k_2\} \cup (k_1 \oplus k_2)$

$\mathcal{K}$ enthält nun die gesuchten Zusammenhangskomponenten.

G.2 Bemerkungen

Zu dem Verfahren können folgende Eigenschaften festgehalten werden:

(*i*) Es läßt sich zeigen, daß das Ergebnis unabhängig von der Reihenfolge ist, in der überlappende Komponenten verschmolzen werden.

(*ii*) Die Tatsache, daß der Algorithmus mit zunehmender Komponentengröße immer größere Lücken zu überbrücken vermag, liegt in Gleichung G.2 begründet.

(*iii*) Ein Nachteil der Abschätzung mit Bounding boxes ist, daß das Verfahren nicht unabhängig gegenüber der Orientierung des Koordinatensystems ist: In Richtung der Diagonalen werden größere Lücken geschlossen als in Richtung der Koordinatenachsen.

(*iv*) Punkt (*iii*) kann vermieden werden, wenn statt der Bounding boxes *konvexe Hüllen* der Pixelmengen eingesetzt werden. Eigenschaft (*i*) bleibt dabei glücklicherweise bestehen.

(*v*) *Nicht* bestehen bleibt (*i*), wenn stattdessen *minimale Kreisscheiben* [31] verwendet würden...

G.3 Implementierung

Abschließend noch zwei Bemerkungen zur konkreten Implementierung des Verfahrens:

- Eigenschaft (i) erlaubt der Implementierung folgende Optimierung: Sie arbeitet nicht auf Pixel-Ebene, sondern betrachtet bereits initial zusammenhängende Pixel-scan-lines, für die auch die Bounding boxes einfach berechnet werden können.

  Ferner wird schon verschmolzen, während $\mathcal{K}$ noch aufgebaut wird. Dadurch bleibt der Speicherverbrauch des Verfahrens bei dem meisten Bildern sehr gering.

- Der Überlappungs-Test zweier Bounding boxes konnte durch ein Zellrasterverfahren [32] beschleunigt werden.

Auf diese Weise konnte eine recht schnelle Implementierung geschaffen werden, die die nötigen Echtzeitanforderungen ohne Probleme erfüllt (siehe Abschnitt 6.3.2).

Anhang H

Scene-viewer für die Kamerasimulation

Für die Kamerasimulation wurde die Möglichkeit benötigt, schnell und unkompliziert kleinere Szenen in OpenGL zu visualisieren. Dazu habe ich den kleinen Scene-viewer „QuickScene“ geschrieben, der gerade so viel leistet, wie von der Simulation benötigt wird. Es folgt eine knappe Beschreibung.

H.1 Der Szenengraph

Die zu visualisierende Umgebung wird über einem Szenengraphen definiert. Der Aufbau des Graphen ist stark an den des OpenInventor-Systems angelehnt, da diese Graphenstruktur die Generierung der nötigen OpenGL-Aufrufe wesentlich vereinfacht.

Das Hauptmerkmal dieser Art von Szenengraph ist, daß die Abarbeitungsreihenfolge des Graphen wohldefiniert ist (nämlich *pre-order* und von links nach rechts) und daß dabei zuerst behandelte Knoten Seiteneffekte auf alle nachfolgenden Knoten haben können. Beispiel: Ein *Materialknoten* legt das Material aller in der Abarbeitungsreihenfolge hinter ihm liegenden Knoten fest, unabhängig von ihrer genauen Position im Graphen. Auch nach einem Aufstieg aus dem Teilgraphen, in dem sich der Materialknoten befand, bleibt dessen Materialeinstellung erhalten, bis eine neue Materialdefinition oder ein *Separator*-Knoten aufgetaucht ist (s.u.).

Bei dem Graphen handelt es sich im übrigen um den allgemeinen Fall eines gerichteten, azyklischen Graphens. Das heißt insbesondere, daß ein Knoten bei der Abarbeitung mehrfach besucht werden kann. QuickScene kennt folgende Knotentypen:

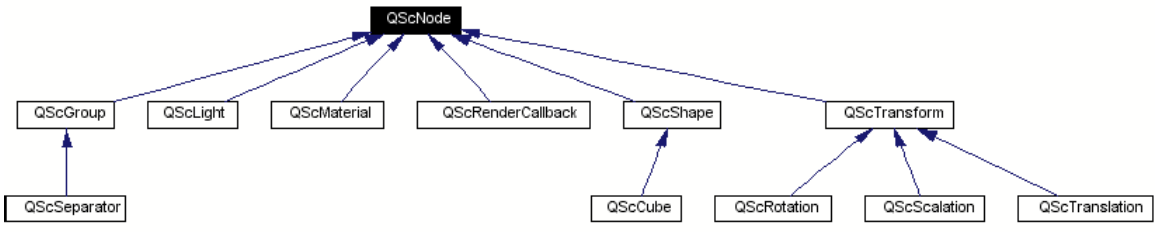

Abbildung H.1. Klassenhierarchie der Knotentypen.

Gruppenknoten (QScGroup)

Gruppenknoten sind die einzigen Knoten, die Nachfolgerknoten besitzen können. Alle anderen Knotentypen können nur an Blättern (Knoten mit Ausgangsgrad 0) auftreten. Gruppenknoten werden daher benutzt, um den Graphen zu strukturieren.

Separator (QScSeparator) Ein Sonderfall des Gruppenknotens ist der Separator: Wird er beim Rendern traversiert, so werden an dieser Stelle alle Material-, Lichtquellen- und Transformationseinstellungen gesichert. Kehrt der Renderer schließlich wieder von den Kinderknoten des Separators zurück, werden diese Einstellungen wiederhergestellt.

Die Wirkung eines Material-, Lichtquellen- oder Transformationsknotens kann innerhalb des Graphens immer nur bis zum nächsten Separator hochreichen. Dadurch können die Seiteneffekte, die durch die Graphenkonvention entstehen, kontrolliert werden.

Sichtbare Oberflächen (QScShape)

Alle Knoten, die eine sichtbare Repräsentation in der Szene besitzen, gehören der Klasse QScShape an. Da es für meine Anwendung bisher gereicht hat, wird zur Zeit nur ein einziger QScShape-Knotentyp unterstützt, nämlich

Quader (QScCube) Der QScCube-Knoten steht für einen achsenparallelen Quader, dessen sechs Seiten mit dem aktiven Material versehen sind.

Transformationsknoten (QScTransform)

Die Transformationsknoten dienen dazu, alle in der Abarbeitungsreihenfolge nach ihnen erscheinende Objekte einer Transformation zu unterwerfen.

Am Transformationsknoten tritt die Konvention zur Graphen-Abarbeitung am augenfälligsten zutage. Manchem würde es nämlich intuitiver erscheinen, wenn ein Transformationsknoten in der Hierarchie prinzipiell oberhalb des Knotens, den er beeinflussen soll, plaziert würde. Das ist hier aber *nicht* der Fall. Es sind alle Objekte betroffen, die in der Abarbeitung nach ihm liegen, bevor der nächste Separator-Knoten wieder verlassen wird.

Liegen bis dort noch weitere Transformationsknoten, wird deren Transformation mit der aktuellen (durch *Rechts*multiplikation) verknüpft.

Ein QScTransform-Knoten kann nur in einer der folgenden Formen auftreten:

Rotation (QScRotation) Der Rotationsknoten ist leider immer noch nicht implementiert.

Skalierung (QScScalation) Der Skalierungsknoten skaliert die nachfolgenden Knoten in x-, y- und z-Richtung.

Translation (QScTranslation) Der Translationsknoten verschiebt die nachfolgenden Knoten in x-, y- und z-Richtung.

Materialeigenschaften (QScMaterial)

Die Materialeigenschaften der nachfolgenden Objekte werden durch einen QScMaterial-Knoten beschrieben. Der Viewer unterstützt einfarbige, PHONG-schattierte Oberflächen. Im Materialknoten sind (r, g, b, a)-Werte für ambiente, diffuse und glänzende Beleuchtung gespeichert.

OpenGL kann im allgemeinen nur GOURAUD-schattierte Dreiecke zeichnen. PHONG-Schattierung wird daher durch Echtzeittesselierung in Abhängigkeit von den Lichtquellenpositionen erreicht.

Lichtquellen (QScLight)

Keine Bilder ohne Licht. In der virtuellen Umgebung dürfen daher mehrere Lichtquellen positioniert werden. Zur Zeit werden nur punktförmige Lichtquellen unterstützt.

Eine Lichtquelle wird durch ihre Position und ihre (r, g, b, a)-Farbe für ambiente, diffuse und glänzende Beleuchtung definiert.

Call-back-Knoten für OpenGL-Rendering (QScRenderCallback)

Der Scene-viewer wurde vor allem entwickelt, um eine möglichst unbürokratische Verbindung zwischen der Szenengraph-Visualisierung und diversen OpenGL-Tricks zu ermöglichen.

Daher existiert auch ein Call-back-Knoten, mit dem direkt in den Render-Vorgang eingegriffen werden kann. Der Knoten erhält einen Einsprung, wenn er vom Renderer während der Graphenabarbeitung traversiert wird.

Zum Beispiel wurden die Landmarkenbilder mit Hilfe dieses Knotentyps in die Szene eingefügt. Dabei mußte mit OpenGL etwas getrickst werden, da die Landmarken exakt in der Wand liegen und daher leicht Opfer von z-Buffer-Aliasing geworden wären.

Eine zusammenfassende Darstellung der Knotenhierarchie findet sich in Abbildung H.1.

Um den Aufbau des Graphen und das Dateiformat so unkompliziert wie möglich zu halten, erfolgt die Verzeigerung des Graphen rein symbolisch; jeder Knoten erhält einen eindeutigen Namen, und statt über Pointer-Referenzen werden alle Knoten nur über diese Namen adressiert.

H.2 Aufruf

Der QuickScene-Viewer steht in Form der C++-Klasse QuickScene zur Verfügung.

QuickScene stellt Methoden zur Graphenmanipulation bereit. Dadurch ist es theoretisch möglich, den kompletten Szenengraphen „on-the-fly“ zusammenzubauen. Komfortabler ist es jedoch, den Graphen aus einer Datei zu lesen. Siehe dazu Abschnitt J.2.

Die fertige Szene kann in einen bestehenden OpenGL-Kontext gerendert werden. Dabei können Kameraeinstellung und Lichtquellen wahlweise vom OpenGL-Kontext übernommen werden.

H.3 Known Bugs

Die nötige Garbage-Collection wurde noch nicht implementiert. Das heißt, daß intensive Manipulationen am Szenengraphen den Speicherverbrauch erhöhen.

Anhang I

Rahmenapplikation

Dieser Anhang stellt den Funktionsumfang der Rahmenapplikation kurz anhand der Menüstruktur und der verfügbaren Tastenbefehle vor. Dennoch kann er keine vollständige Benutzerdokumentation bieten.

I.1 Menüstruktur

Die wichtigsten Aktionen sind über ein Pop-Up-Menü mit weiteren Untermenüs zugänglich (rechte Maustaste). Es folgt eine Auflistung dieser Menüstruktur zusammen mit kurzen Erklärungen:

- View...
  - *Auswahl der aktuellen Ansicht. Verfolger und/oder Kamerasimulation laufen unabhängig von der Ansicht weiter.*
  - Source Image
  - Panorama
  - Simulation Environment
- Calibration by...
  - *Stößt eine Kamerakalibrierung an. Sie wird von der Panoramasicht, der Kamerasimulation und dem Verfolger benötigt. Die Kalibrierung kann im laufenden System gewechselt werden hat sofortige Wirkung.*
    Über das Untermenü ist der gewünschte Kalibrierungsweg zu wählen. Das Programm erwartet dann auf der Kommandozeile...
  - prepared xfig-file...
    - *... den Namen einer* xfig*-Datei.*
  - specifying p and a manually...
    - *... die Kalibrierungsparameter* $\boldsymbol{p}$ *und* a*.*
  - specifying p and R manually...
    - *... die Kalibrierungsparameter* $\boldsymbol{p}$ *und* R*.*
- Image source...
  - *Auswahl der Bildquelle, von der die Kamerabilder bezogen werden.*
  - None
  - Camera
  - Camera Simulation
  - Movie
- Image sink...
  - *Auswahl der Bildsenke. Die Kamerabilder aus der Bildquelle können wahlweise zum Verfolger geleitet oder als Bildfolge abgelegt werden.*

- None
- Motion Tracker
- Movie

- Camera simulation...
  - Camera resolution...

    Über dieses Menü läßt sich die Kameraauflösung aus einer Reihe vorgegebener Alternativen wählen. Möchte man weitere Möglichkeiten testen, müssen diese zur Zeit noch im Code eingetragen werden:

    - 640x480, interlaced (NTSC)
    - 640x480, progressive scan
    - 800x600, progressive scan
    - 1024x768, progressive scan
  - Sampling quality...

    Hierüber läßt sich steuern, nach welchem Abtastverfahren die Verzerrung des Kamerabilds berechnet werden soll. Die zur Wahl stehenden Verfahren wurden schon in Abschnitt 5.4.3 vorgestellt:

    - NearestNeighbour
    - Super 2 (iso)
    - Super 2 (aniso)
    - Super 4
    - Gauss 4
    - Gauss 9
- Redraw

  Erzwingt ein Neuzeichnen des Fensterinhalts.
- Quit

  Beendet das Programm.

I.2 Tastenbefehle

Aktuelle Sicht

`P`	Panorama
`S`	Quellbild
`E`	Simulationsumgebung

Allgemeine Kürzel

Programm

`?`	Gibt eine kleine Hilfe nach stdout aus.
`q`	Beendet das Programm.

Debug-Einblendungen

Mit den Ziffern `1`–`5` können folgende Einblendungen ein- und ausgeschaltet werden:

`1`	Bounding boxes
`2`	detektierte Randpixel
`3`	Konik-Anpassung
`4`	durch die Simulation bekannte Landmarkenzentren
`5`	Ergebnis der Positionsrekonstruktion

Weiter gibt es folgende Tasten:

)	Löscht alle Einblendungen.
&	Läßt alle Fadenkreuze rotieren, damit man sie besser sieht.

Quellbildansicht

Bildsteuerung

Der sichtbare Ausschnitt des Kamerabilds läßt sich bei gedrückter linker Maustaste verschieben. Der Maßstab der Anzeige kann über die Zoom-Tasten + und - verändert werden.

Simulationsumgebung

Bewegungssteuerung

Die Navigation im Raum erfolgt vorzugsweise mit der Maus. Bei gedrückter linker Maustaste kann man sich durch Bewegung der Maus umschauen. Bei gedrückter mittlerer Maustaste bewegt mach sich in Blickrichtung vorwärts.
Darüberhinaus stehen noch folgende Tasten für die Navigation zur Verfügung:

↑, s	vorwärts
↓, Space	rückwarts
←	nach links drehen
→	nach rechts drehen
a	side-step links
d	side-step rechts
x	hoch
z	runter

Mit Insert läßt sich außerdem noch die Kamera auf die Bildebene einrasten. Die Steuerung ist dann analog zu der im Quellbild-Modus.
Die Brennweite der Benutzeransicht (*nicht* die der simulierten Kamera!) kann über die Zoom-Tasten + und - verändert werden.

Einblendungen

v	Überlagert das Bild der virtuellen Umgebung mit dem Bild mit dem simulierten Kamerabild.
c	Wie v, nur zeigt es die "cubic environment-map", die für die Bildberechnung benutzt wird.
f	Markiert in der v- oder c-Sicht den Bereich, der aus der Sicht des Benutzers „vorne“ ist.
t	Zeigt die aufgezeichnete Trajektorie und, sofern vorhanden, die rekonstruierte.

Trajektorie

`r` Beginnt die Trajektorienaufzeichnung bzw. hält sie an. Schickt nachträglich einen Gauß-Filter über die rekonstruierte Trajektorie.

`>` Stößt die Wiedergabe der Trajektorie an bzw. beendet sie.

Bilderzeugung

`/` Schickt ein Einzelbild der aktuellen Positon zur Bildsenke.

`|` Legt die Kamerasimulation permanent als Bildquelle fest. (Erst dann werden bei der Trajektorienwiedergabe Bilder erzeugt.)

`@` Schaltet für die Aufzeichnug zwischen *real-time* und *frame-time* um.

Panoramaansicht

Die Navigation in der Panoramaansicht ist identisch zur Steuerung in der Simulationsumgebung. Die einzigen nur in der Panoramaansicht verfügbaren Tasten sind:

Utah Teapot

`o` Blendet eine verspiegelte Teekanne ein, in der sich die Umgebung, wie sie vom Kamerabild aufgezeichnet wurde, spiegelt.

`i` Kippt die Teekanne vom Benutzer weg.

`m` Kippt die Teekanne zum Benutzer hin.

`j` Dreht sie nach links.

`k` Dreht sie nach rechts.

`w` Zeigt die Teekanne in Drahtgitterdarstellung.

Anhang J

Dateiformate

Die Formate der meisten Ein-/Ausgabedateien, von denen in dieser Arbeit Gebrauch gemacht wird, sind sehr einfach aufgebaut und lassen sich teilweise schlecht erweitern. Dies rührt daher, daß sie zu einem Zeitpunkt entstanden sind, als noch nicht abzusehen war, ob sie als Dateien überhaupt bestehen bleiben würden.

Gerade *weil* sie sich noch in diesem unausgereiften Entwicklungsstand befinden, will ich die wichtigsten hier genauer dokumentieren.

J.1 Landmarkenpositionen

Die vermessenen Landmarken müssen in der Landmarken-Datei bereitstehen. Diese Datei wird sowohl von der Positionsrekonstruktion als auch vom Simulatonssystem benötigt. Als Dateiendung sei `.lml` (*landmark list*) vorgeschlagen.

Inhalt der Datei

Die Datei beschreibt für jede Landmarke Typ, Position und Ausrichtung:

Typ Der Landmarkentyp wird durch einen integer-Index beschrieben. Die Interpretation des Indexes ist im Programm „fest verdrahtet“[1]. Zur Zeit gilt folgende Zuordnung:

Index	*Innere Farbe*	*Äußere Farbe*
0	rot	blau
1	blau	rot
2	rot	grün
3	grün	rot
4	blau	grün
5	grün	blau

Es werden im Moment also nur Marken unterstützt, die aus einer inneren Scheibe und einem konzentrischen Ring bestehen.

Position Die Position des Landmarkenmittelpunkts wird in Form von drei Gleitkommazahlen (x, y, z) angegeben. Die Längeneinheit wird für die ganze Datei global festgelegt (s.u.).

Ausrichtung In manchen Fällen wird neben der Position der Marke auch deren Ausrichtung benötigt. Diese wird über ihre Normale definiert (wieder drei Gleitkommazahlen). Der angegebene Vektor muß nicht normiert sein.

[1] Die Zuordnungstabelle findet sich in `lmdefaults.cc`.

Darüberhinaus finden sich in der Datei noch eine Angabe, in welcher Längeneinheit alle Positionsangaben zu interpretieren sind, sowie ein Offset, der auf alle Positionen addiert wird. (Auch dieser Offset muß in der globalen Einheit angegeben sein.)

Syntax

Der Aufbau der Datei sieht wie folgt aus:

Kommentare beginnen mit dem Kommentarzeichen '`#`'.

Der Inhalt einer Zeile ab dem Kommentarzeichen (einschließlich) wird ignoriert.

Leerzeilen und Zeilen, die nach Entfernen des Kommentars nur Leerzeichen („*white-spaces*“) enthalten, werden *ignoriert*.

Für die verbleibenden Zeilen gilt:

- In der ersten Zeile steht eine Gleitkommazahl, die die verwendete Längeneinheit in Metern angibt. (Z.B. würde `0.001` die globale Einheit auf Millimeter setzen.)
- Die zweite Zeile enthält den Positionsoffset in Form von drei Gleitkommazahlen (x, y, z).
- Alle weiteren Zeilen beschreiben jeweils eine Landmarke. Das Zeilenformat ist:

Einträge	*Typ*	*Bedeutung*
1	`integer`	Markentyp
2–4	3 × `float`	Markenposition
5–7	3 × `float`	Markenausrichtung

Alle Zahlenangaben sind durch Leerzeichen zu trennen.

Beispiel

Im folgenden sei exemplarisch eine Landmarkendatei aufgeführt:

```
# -*- shell-script -*- (Emacs-hints to
#               highlight my comments ;-)

# Landmark List for my Test Chamber
#

# The room is 500/400 cm wide in z/x-
# direction and 240 cm tall (y).
#

# Units (1cm):
0.01

# Translation:
-120 -170 -250

# Marks on the northern wall:
#
0 400 207  83  -1 0 0
1 400 120 256  -1 0 0
0 400  69 243  -1 0 0
2 400 170 380  -1 0 0

# Marks on the ceiling:
#
0 180 240  75  0 -1 0
2 299 240 162  0 -1 0
1 100 240 331  0 -1 0
2 230 240 375  0 -1 0
0 140 240 200  0 -1 0

# Marks on the southern wall:
#
0   0 107 300  1 0 0
1   0 215 144  1 0 0
0   0 190 270  1 0 0

# Marks on the western wall:
#
0 140  60   0  0 0 1
2 313 113   0  0 0 1
1  70 177   0  0 0 1
2 230 158   0  0 0 1
```

```
# Marks on the eastern wall:
#
0 160  40 500  0 0 -1
2 193 133 500  0 0 -1
1  90 157 500  0 0 -1
1 340 178 500  0 0 -1
```

J.2 Szenengraph für virtuelle Umgebung

Der Aufbau des Szenengraphen, der für die Simulationsumgebung verwendet wird, wurde bereits in Anhang H beschrieben. Es ist möglich, den Szenengraphen anhand einer Textdatei zu beschreiben. Das zugehörige Format soll hier erklärt werden. Die empfohlene Dateiendung lautet `.qsc` (***q**uick**sc**ene*).

Inhalt

Über das Dateiformat steht bis jetzt nur ein Teil der Möglichkeiten zur Verfügung. Es gibt Szenengraphen, die nicht im Dateiformat beschreibbar sind. Das betrifft besonders die Parametrisierung mancher Knoten. So ist es zum Beispiel nicht möglich, aus der Datei heraus den Alpha-Anteil einer Farbe zu setzen; hier werden nur (r, g, b)-Tripel akzeptiert.

Syntax

Das Dateiformat ist wieder zeilenorientiert. Die Syntax ist:

Kommentare beginnen mit dem Kommentarzeichen '`#`'.

Der Inhalt einer Zeile ab dem Kommentarzeichen (einschließlich) wird ignoriert.

Leerzeilen und Zeilen, die nach Entfernen des Kommentars nur Leerzeichen („*white-spaces*“) enthalten, werden *ignoriert*.

Alle anderen Zeilen bestehend aus white-space separierten Tokens. Sie müssen einer der folgenden Formen entsprechen:

`UNIT` ⟨*Längeneinheit*⟩
Legt die aktuelle Längeneinheit fest. ⟨*Längeneinheit*⟩ ist eine Gleitkommazahl. Alle nachfolgenden Längen- und Positionsangaben werden als ⟨*Längeneinheit*⟩ Meter interpretiert.

(`DEFINE` | `GROUP`) ⟨*Bezeichner*⟩
Definiert einen Gruppenknoten des Namens ⟨*Bezeichner*⟩. Alle nachfolgenden Zeilen bis zur nächsten Gruppen- oder Separatordefinition definieren Kinder des Gruppenknotens. Die Schlüsselwörter `GROUP` und `SEPARATOR` sind gleichbedeutend.

`SEPARATOR` ⟨*Bezeichner*⟩
Definiert einen Separatorknoten des Namens ⟨*Bezeichner*⟩. Alle nachfolgenden Zeilen bis zur nächsten Gruppen- oder Separatordefinition definieren Kinder des Separatorknotens.

⟨*Gruppenbezeichner*⟩
Fügt den Kindern des aktuellen Gruppen-/Separatorknoten einen Gruppen-/Separatorknoten des Namens ⟨*Gruppenbezeichner*⟩ hinzu. Der hinzugefügte Knoten muß noch nicht bekannt sein; er kann auch später erst definiert werden.

Weitere Knoten werden über im Programm fest eingebaute *Templates* generiert. Die allgemeine Form eines Template-Aufrufs ist:

⟨*Templatebezeichner*⟩ ⟨*Argumentliste...*⟩

Der generierte Knoten wird den Kindern des aktuellen Gruppen- oder Separatorknoten hinzugefügt.

Zur Zeit stehen folgende Templates zur Verfügung:

`tCube` [$\langle x_o \rangle$ $\langle y_o \rangle$ $\langle z_o \rangle$ $\langle w \rangle$ $\langle h \rangle$ $\langle d \rangle$]
Generiert einen achsenparallelen Quader, mit der Diagonalen $(x_o, y_o, z_o)^\top \longleftrightarrow (x_o{+}w, y_o{+}h, z_o{+}d)^\top$. Entfallen die optionalen Argumente, wird der Einheitswürfel mit $(0,0,0)^\top \longleftrightarrow (1,1,1)^\top$ gebaut.

`tMaterial` $\langle r \rangle$ $\langle g \rangle$ $\langle b \rangle$
Erzeugt einen Materialknoten mit $0.3(r,g,b)$ als ambiente und $0{,}6(r,g,b)$ als diffuse Farbe.

`tMaterial` $\langle r_a \rangle$ $\langle g_a \rangle$ $\langle b_a \rangle$ $\langle r_d \rangle$ $\langle g_d \rangle$ $\langle b_d \rangle$
Erzeugt einen Materialknoten mit (r_a, g_a, b_a) als ambiente und (r_d, g_d, b_d) als diffuse Farbe.

`tLight` $\langle x \rangle$ $\langle y \rangle$ $\langle z \rangle$ [$\langle r \rangle$ $\langle g \rangle$ $\langle b \rangle$]
Setzt eine rein diffuse Lichtquelle der Farbe $0{,}6(r,g,b)$ an die Position (x,y,z). Entfallen die Farbparameter, wird die Farbe $(1,1,1)$ gewählt.

`tLight` $\langle x \rangle$ $\langle y \rangle$ $\langle z \rangle$ $\langle r_a \rangle$ $\langle g_a \rangle$ $\langle b_a \rangle$ $\langle r_d \rangle$ $\langle g_d \rangle$ $\langle b_d \rangle$ [$\langle r_s \rangle$ $\langle g_s \rangle$ $\langle b_s \rangle$]
Setzt eine rein diffuse Lichtquelle der ambienten/diffusen/glänzenden Farbe (r_a, g_a, b_a)/(r_d, g_d, b_d)/(r_s, g_s, b_s) an die Position (x,y,z). Entfällt die Angabe von (r_s, g_s, b_s), so wird hierfür $(0,0,0)$ angenommen.

`xTransl` [$\langle x_t \rangle$ $\langle y_t \rangle$ $\langle z_t \rangle$] Erzeugt einen Translationsknoten mit der Verschiebung um $(x,y,z)^\top$. Entfallen die Argumente, dann wird eine Verschiebung um den Nullvektor generiert.

Und speziell für meine Anwendung stellen

`mWall`

`mCeiling`

`mFloor`

`mDoor`

`mMetal`

`mPlastic`

gleich eines von sechs vorgegebenen Materialien ein.

Beispiel

Als Beispiel folgt die Szenendatei, mit der die Experimente zur Arbeit stattgefunden haben:

```
# -*- shell-script -*- (Emacs-hints for
#                   high-lighting my comments ;-)

# Important Note:
#
#   Due to a bug, please don't use any
#   transformation inside the root node, or your
#   landmarks will be shifted away...
#
```

```
# Test Chamber
#

# The room is 500/400 cm wide in z/x-direction and
# 240 cm tall (y).
#
# Names:
#
```

```
#   tFoo        built-in template
#   mFoo        material (all materials are
#                         built-ins at the moment)
#   xFoo        transformation

# unit node (we're using cm; one cm is 0.01 meters)
#
UNITS 0.01

# select root node
#
ROOT root0

# root node
#
SEPARATOR root0
  root

SEPARATOR root
  xTransl     -120 -170 -250
  lights
  room
  tables

GROUP lights
  # I chose a GROUP, since SEPARATORs still switch
  # off the lights :-(
  xTransl     200 210 150
  light
  #xTransl     150 0 300
  #light
  #bulb
  #xTransl     -150 0 -300
  bulb
  xTransl     -200 -210 -150

GROUP bulb
  tMaterial     0.4 0.4 0.4  0 0 0
  tCube         -5 -5 -5 10 10 10

GROUP light
  tLight        0 0 0    0.4 0.4 0.4  0.6 0.6 0.6
  #tLight        0 0 0    0.2 0.2 0.2  0.3 0.3 0.3


# architecture
#
```

```
SEPARATOR room
  mWall
  # walls
  tCube          400 240 500     -400 -240 -500
  # corner
  tCube            0   0 400       50  240  100

  mDoor
  # closed door
  tCube            5   0  80        5  200  120

# Furniture
#
SEPARATOR tables
  xTransl  300 0 340
  table
  xTransl  0 0 -160.2
  table

# Table
#
DEFINE table
  mMetal
  # front-left leg
  tCube            0   0   0        4   58    2
  tCube            0   0   0        2   58    4
  # front-right leg
  tCube            0   0 158        4   58    2
  tCube            0   0 156        2   58    4
  # back-left leg
  tCube           96   0   0        4   58    2
  tCube           98   0   0        2   58    4
  # back-right leg
  tCube           96   0 158        4   58    2
  tCube           98   0 156        2   58    4
  # front frame
  tCube            0  56   0        2    2  160
  # back frame
  tCube           98  56   0        2    2  160
  # left frame
  tCube            0  56   0      100    2    2
  # right frame
  tCube            0  56 158      100    2    2

  mPlastic
  # top
  tCube            0  58   0      100    2  160
```

J.3 Weitere Dateiformate

Darüberhinaus existieren noch weitere Dateiformate, auf deren ausführliche Beschreibung aus Platzgründen verzichtet wurde. Da es sich aber um Formate handelt, von denen nicht zu erwarten ist, daß ein Benutzer sie von Hand editieren wird, sollte dies zu verschmerzen sein. Es folgt eine Kurzbeschreibung der verbleibenden Formate:

Extension	Kurzbeschreibung
`.myrgba`	Ein sehr primitives Bilddateiformat, in dem Testbilder eingelesen werden. Die Datei beginnt mit einer `\n`-terminierten Beschreibungszeile, die von binären Rohdaten gefolgt wird.
`.trj`	Ein Format zur Trajektorienaufzeichnung. Die Dateien enthalten im wesentlichen eine Folge von Positionen/Orientierungen im Raum, die mit einem Zeitstempel versehen sind.
`.mov`	In Verzeichnissen mit dieser Endung werden mehrere `.myrgba`-Dateien und eine Indexdatei abgelegt.

Anhang K

Technische Daten der Panoramakamera

REMOTEREALITY™
Immerse Yourself...

ParaCamera™ S-360c
Specifications

ParaCamera S360c™ – 360-degree color video camera - The ParaCamera S360c is Remote Reality's most compact 360-degree color video camera. The S360c is less than 100mm tall and 70mm wide and provides an unobstructed panoramic view. The S360c can be used to provide complete coverage in small, difficult to monitor areas like ATM lobbies and cash

Component	Description
Optical	
♦ Field of view (FOV)	(30°- 90°) x 360°
♦ F-number	2.5
♦ Focal Length	0.98mm
♦ Blind spot	+/- 30°
❖ Vertical angular resolution	at 30°: 8.7 mrad (30.0') at 90°: 4.6 mrad (16.0')
❖ Horizontal angular resolution	at 30°: 7.4 mrad (25.5') at 90°: 4.0 mrad (13.7')
Image Sensor (CCD)	
♦ Size	1/3"
♦ Sensitive Area	4.8 mm x 3.6 mm (0.1890" x 0.1415")
♦ Number of pixels	768(H) x 494(V)
♦ Pixel Size	6.25 µm (H) x 7.3 µm (V)
♦ Sensitivity	2 lux
Video	
♦ Signal type	NTSC (color)
♦ Resolution	450 TV lines
Power	
♦ Power	12 VDC (adapter provided)
Physical	
♦ Height	3.850" (97.8mm)
♦ Max. Body diameter	2.61" (66.3mm)
♦ Min. Body diameter	2.15" (54.6mm)
♦ Base plate diameter	3.00" (76.2mm)
♦ Weight	9.4 oz (267g)
♦ Mounting	Four (4) 4-40 screw clearance holes built into baseplate on a 2.790" HBC[1].
Software	
♦ Version	ParaPlayer™ 1.1
♦ Operating System	Windows®95™, 98, NT™4.0
♦ CPU	Intel® Pentium® 233Mhz (or higher), or equivalent processor
❖ Frame grabber	Matrox® Meteor®, Meteor PPB, Pulsar®

[1] HBC Hole Bolt Center

Abbildung K.1. Datenblatt der Panoramakamera.

Literaturverzeichnis

[1] I. N. Bronstein, A. Semendjajew, G. Musiol und H. Mühlig. *Taschenbuch der Mathematik.* Verlag Harri Deutsch, 3. Aufl., Frankfurt am Main · Thun 1997.

[2] R. Dillmann und M. Huck. *Informationsverarbeitung in der Robotik.* Springer Verlag, Berlin · Heidelberg · New York 1991.

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling und B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 2nd edition, Cambridge · New York · Melbourne 1992.

[4] T. Salb, T. Weyrich, and R. Dillmann. *Preoperative planning and training simulation for risk reducing surgery.* Proceedings of Conference: International Training and Education Conference (ITEC), The Hague, April 1999.

[5] T. Weyrich. *Bewertung und Optimierung von Schnitten in Weichgewebe — ein risikobasierter Ansatz.* Studienarbeit, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe, Karlsruhe 1999.

[6] G. Welch, G. Bishop, L. Brumback, et al. *The HiBall Tracker: High-Performance Wide-Area Tracking for Virutal and Augmented Environments.* Symposium on Virtual-Reality Software and Technology, 1999, University College London, London 1999.

[7] G. Welch, G. Bishop. *SCAAT: Incremental Tracking With Incomplete Information.* SIGGRAPH 97 Conference Proceedings, Annual Conference Series, ACM SIGGRAPH August 1997, Los Angeles, CA.

[8] G. Welch, G. Bishop. *SCAAT: Incremental Tracking With Incomplete Information.* University of North Carolina at Chapel Hill, doctoral dissertation, Technical Report TR 96-051, `http://www.cs.unc.edu/~welch/publications.html`, 1996.

[9] M. O. Franz, B. Schölkopf, H. A. Mallot, H. H. Bülthoff und A. Zell. *Navigation mit Schnappschüssen.* P. Levi, R.-J. Ahlers, F. May, M. Schanz (Hrsg.): Mustererkennung 1998. Proc. 20. DAGM-Symposium: S. 421–428, Springer Verlag, Berlin 1998.

[10] M. O. Franz, B. Schölkopf und H. H. Bülthoff. *Homing by Parameterized Scene Matching.* P. Husbands, I. Harvey (Eds.): Proc. 4th European Conf. on Artificial Life, pp. 236–245, MIT Press, Cambridge 1997.

[11] B. A. Cartwright and T. S. Collett. *Landmark Learning in Bees.* J. of Comp. Physiol. A 151, 521–543, 1983.

[12] J. Gaspar and J. S. Victor. *Visual Path Following with a Catadioptric Panoramic Camera.* Proceedings of the 7th International Symposium on Intelligent Robotic Systems (SIRS'99), University of Coimbra, 1999.

[13] F. Wawak, N. Katevas and F. Mat´ia. *Co-operation between Omni-directional Camera and Mobile Camera for the Localisation of a Mobile Platform.* Proceedings of the 7th International Symposium on Intelligent Robotic Systems (SIRS'99), University of Coimbra, 1999.

[14] O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint.* The MIT Press, Cambridge, Mass., 1993.

[15] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, Cambridge · New York · Melbourne · Madrid 2000.

[16] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applikations.* Springer Verlag, Berlin · Heidelberg · New York 2000.

[17] H.-D. Ebbinghaus. *Zahlen.* Springer Verlag, 3. Aufl., Berlin · Heidelberg 1992.

[18] F. Locher. *Numerische Mathematik.* Springer-Verlag, Berlin · Heidelberg · New York 1993.

[19] R. E. Kalman. *A New Approach to Linear Filtering and Prediction Problems.* Transactions of the ASME — Journal of Basic Engineering, pp. 35–45, 1960.

[20] G. Welch and G. Bishop. *An Introduction to the Kalman Filter.* University of North Carolina at Chapel Hill, Department of Computer Science, TR 95-041, `http://www.cs.unc.edu/~welch/publications.html`, 1995.

[21] M. B. Dillencourt, H. Samet, and M. Tamminen. *A General Approach to Connected-Component Labelling for Arbitrary Image Representations.* Journal of the ACM, 39(2): 253–280, 1992.

[22] H. Samet, M. Tamminen. *An Improved Approach to Connected Component Labelling of Images.* IEEE Computer Society Conference on Computer Vision and Pattern-Recognition (CVPR), pp. 312–318, Miami Beach, Fla., 1986.

[23] G. Lohmann. *Volumetric Image Analysis.* John Wiley & Sons / B.G. Teubner, Chichester / Stuttgart 1998.

[24] V. Gengenbach. *Einsatz von Rückkopplungen in der Bildauswertung bei einem Hand-Auge-System zur automatischen Demontage.* Dissertation Universität Karlsruhe, Dissertationen zur künstlichen Intelligenz, Infix Verlag, Sankt Augustin 1994.

[25] Z. Zhang. *Parameter estimation techniques: a tutorial with application to conic fitting.* Image and Vision Computing 15 (1997) 59–76, 1997.

[26] J. L. Mundy und A. Zisserman (Hrsg.). *Geometric Invariance in Computer Vision.* The MIT Press, Cambridge, Mass. 1992.

[27] A. Lohr. *Eine robuste Kalibriermarkenerkennung für den Kamera-Roboter CaRo.* Studienarbeit, Institut für Betriebs- und Dialogsysteme, Universität Karlsruhe, Karlsruhe 1999.

[28] B. K. Horn. *Closed-form solution of absolute orientation using unit quaternions.* Journal Optical Society of America, 4(4): 629–642, April 1987.

[29] P. J. Besl and N. D. McKay. *A Method for Registration of 3-D Shapes.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2): 239–256, February 1992.

[30] G. Bleiker. *Eine Methode zur automatischen Registrierung von Oberflächenscans.* Semesterarbeit, Institut für Wissenschaftliches Rechnen, Eidgenössische Technische Hochschule Zürich, Zürich 2000.

[31] E. Welzl. *Smallest enclosing disks (balls and ellipsoid).* Lecture Notes in Computer Science, Band 555, Springer Verlag, Berlin · Heidelberg · New York 1991.

[32] A. Schmitt, O. Deussen und Marion Kreeb. *Einführung in graphisch-geometrische Algorithmen.* Teubner, Stuttgart 1996.