Ein modulares optisches Trackingsystem für medizintechnische Anwendungen: integrierte Datenflussarchitektur in Hard- und Software und Applikationsframework

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.-Ing. Andreas Köpfle
aus Heidelberg

Mannheim, 2012

Dekan:	Professor Dr. Heinz Jürgen Müller, Universität Mannheim
Referent:	Professor Dr. Reinhard Männer, Universität Heidelberg
Korreferent:	Professor Dr. Holger Fröning, Universität Heidelberg

Tag der mündlichen Prüfung: 11. Februar 2013

für meine Eltern

Zusammenfassung

Die vorliegende Arbeit beschreibt die Entwicklung eines modularen optischen Trackingsystems für medizintechnische Anwendungen. Motivation hierfür waren Schwierigkeiten, die bei der Adaption verfügbarer Systeme für den Einsatz in medizinischen Trainingssimulatoren auftraten. Deren spezielle Anforderungen wie z.B. die Verwendung von Originalinstrumenten, sehr kleine Trackingvolumina, Erfassung deformierbarer Oberflächen und eine einfache Integration in die Simulatorarchitektur lassen sich mit den vorhandenen Lösungen für Standardtrackingaufgaben nur unzureichend umsetzen. Es wurde daher ein Konzept entwickelt für ein universelles, flexibel einsetzbares Baukastensystems, das die Starrheit fertiger Systemlösungen vermeidet und mit moderatem Aufwand an verschiedenartige Aufgabenstellungen anpassbar ist.

Das System abstrahiert hierzu von der spezialisierten Datenverarbeitung üblicher Trackinglösungen und baut auf einer generalisierten, modularen Datenflussarchitektur für alle Arten von Markern mit drei Freiheitsgraden auf. Die Integration FPGA-basierter Bildverarbeitungshardware in den Kameradatenstrom ermöglicht eine frühzeitige Reduktion der Datenmengen und führt zu niedrigen Latenzzeiten. Ein zentraler Punkt des Systemkonzepts ist dabei die Integration der Hardware- und Softwarekomponenten zu einer durchgängigen, flexibel ausgelegten Datenflussarchitektur. Gleichzeitig wird ein breites Spektrum an Verfahren bereitgestellt, die für Bildverarbeitung, Markerdetektion und Rekonstruktion genutzt werden können.

Spezieller Fokus wurde auf die schnelle und einfache Entwicklung von Trackinganwendungen gelegt. So ermöglicht die duale Auslegung der Verarbeitungsmodule in spezialisierter Hardware bzw. PC-basierten Softwarebibliotheken die einfache Adaption der Datenverarbeitungsressourcen an die jeweilige Aufgabenstellung. Ein Simulationsmodus mit virtuellen, in Software nachgebildeten Datenquellen unterstützt bei Entwicklung und Test von neuartigen Trackinganwendungen. Und ein erweiterbares graphisches Softwarefrontend stellt direkten Zugriff auf die Systemfunktionalität bereit für alle Aufgaben im Zusammenhang mit Betrieb, Konfiguration und Entwicklung der Systeme.

Spezielle Eigenschaften der Systemauslegung für einen Einsatz in medizinischen Simulatoren umfassen die optionale Unterstützung von Farbmarkern, die anwendungs-

spezifische Auslegung der Bilderfassungsmodule und eine geringe Ressourcenbelastung des Simulator-PCs. Insbesondere die selten eingesetzte Objektsignalisierung über Farbmarker erwies sich als sinnvolles Konzept für die eingangs genannten Trackingaufgaben, die mit herkömmlichen Markierungstechniken nur unzureichend umzusetzen sind. Farbmarker sind einfach zu applizieren und zu detektieren und die Farbe dient zusätzlich als einfache Codierung für Objektidentifikation und Markerzuordnung. Für offene Trackingsetups und Anwendungen mit höheren Genauigkeitsanforderungen werden zusätzlich infrarotbasierte Signalisierungstechniken unterstützt.

Das System wurde erfolgreich in mehreren Prototypen medizinischer Simulatoren für ophthalmoskopische Diagnostik, ophthalmochirurgische, neurochirurgische und endoskopische Interventionen eingesetzt, darüber hinaus auch für die Positionserfassung eines handgehaltenen Operationsroboters und in allgemeinen VR/AR-Anwendungen. Weitere Einsatzbereiche, auch außerhalb der Medizintechnik, sind durch die universelle Auslegung der Systemarchitektur ohne weiteres zugänglich. Die Bandbreite der realisierten Anwendungen zeigt, dass das Ziel einer universellen Systemauslegung in großen Teilen erreicht wurde. Weiterer Optimierungsbedarf existiert allerdings für Anwendungen mit sehr hohen Anforderungen an die Genauigkeit.

Der Hauptbeitrag der Arbeit besteht in der Schaffung eines Gesamtsystems aus modularer Hard- und Software mit einem erweiterbarem Entwickungsfrontend als Basis für maßgeschneiderte optische Trackinglösungen – insbesondere auch bei unkonventionellen Setups, die mit gängigen Systemen nicht oder nur stark eingeschränkt realisierbar sind. Darüber hinaus liegt der Schwerpunkt auf der Zusammenführung der flexiblen Konzepte modularer Datenflussarchitekturen, wie sie in Trackingbibliotheken der Middleware-Schicht vertreten sind, mit der hohen Performanz spezialisierter Bildverarbeitungshardware, die in den intelligenten Kameras kommerzieller Trackinglösungen zum Einsatz kommt.

Danksagung

Mein Dank gilt allen, die mich während der Arbeiten in der VIPA-Arbeitsgruppe und beim Entstehen dieser Dissertation unterstützt haben. Insbesondere danke ich...

...Herrn Prof. Dr. Reinhard Männer für die Betreuung meiner Promotion und die großen Freiheiten beim Arbeiten an seinem Institut. Außerdem für die Unterstützung in meiner Zeit als Gruppenleiter der VIPA-Gruppe und seine Geduld bei der langen Fertigstellung der schriftlichen Ausarbeitung..

...Herrn Prof. Dr. Holger Fröning für seine Bereitschaft, das Korreferat zu übernehmen.

...Markus Schill für seine Unterstützung als ehemaliger Gruppenleiter und später dann Geschäftsführer der VRmagic GmbH. Seine Aufbauarbeit in der VIPA-Gruppe und die folgende Kooperation von Universität und Firma haben die Durchführung vieler unserer Projekte erst ermöglicht.

...Thomas Ruf, der als Entwicklungsleiter der Trackinglösungen auf VRmagic-Seite viele technische Fragestellungen und essentielle Designentscheidungen mit mir diskutiert hat und immer darauf achtete, dass die entwickelten Lösungen auch ihre Relevanz im praktischen Einsatz behielten.

...Clemens Wagner dafür, dass er zu jeder Zeit bei Problemen als Freund und Ratgeber bereitstand, sei es bei der gemeinsamen Arbeit in der Arbeitsgruppe und bei VRmagic oder auch im privaten Umfeld in unserer Freizeit.

...allen jetzigen und ehemaligen Mitarbeitern der VIPA-Arbeitsgruppe und der VRmagic GmbH, ohne die ein so umfangreiches Projekt nicht durchzuführen gewesen wäre: Stephan Diederich, Florian Beier, Holger Handel, Oliver Schuppe, Yang Yuning, Liu Bing, Olaf Körner, Johannes Grimm, Klaus Rieger, Kathrin Streichert, Ole Jakubik, Stefan Sichler, Jan Hegner, Nikolaj Nock, Michael Schneider und Martin Staiger.

...Andrea Seeger, Christiane Glasbrenner und Dina Geppert für ihre Hilfe beim Umgang mit den oft unverständlichen Verwaltungsvorgängen einer Universität und ein offenes Ohr für Gesprächsthemen jenseits der Informatik.

...Nina Wolfram und Sabrina Muttarrusso für das Korrekturlesen und die moralische Unterstützung.

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
3DOF	drei Freiheitsgrade
6DOF	sechs Freiheitsgrade
AABB	Axis-Aligned Bounding Box
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer Aided Design
CAVE	CAVE Automatic Virtual Environment
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxid Semiconductor
CPU	Central Processing Unit
CT	Computertomographie
DOF	Degree of Freedom
DSP	Digitaler Signalprocessor
EYESI	Eye Surgery Simulation
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
HDI	Hue, Density, Intensity (Farbraum)
HLS	Hue, Luminance, Saturation (Farbraum)
HMD	Head-Mounted Display
I²C	Inter Integrated Circuits
ICM	Institut für computerunterstützte Medizin
IR	Infrarot(licht)
ITD	Intelligent Tool Drive
LED	Licht Emittierende Diode
MOSCOT	Modular Scalable Optical Trackingsystem
MRT	Magnetresonanztomographie

OTS	Optisches Trackingsystem
PC	Personal Computer
Qt	Qt Toolkit
RAD	Rapid Application Development
RAM	Random Access Memory
RGB	Red, Green, Blue (Farbraum)
ROI	Region of Interest
SRAM	Static Random Access Memory
SRG	Spatial Relationship Graph
UCD	User Centered Design
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VIPA	Virtuelle Patientenanalyse
VR	Virtuelle Realität
VRM	Virtuelle Realität in der Medizin

Kapitel 1

Einführung

1.1 Maschinelles Sehen und Optisches Tracking

Der Mensch nimmt den weitaus größen Teil seiner Umgebungseindrücke über den visuellen Kanal wahr. Durch evolutionäre Abläufe optimierte Verarbeitungsprozesse in Retina und visuellem Cortex ermöglichen dabei eine beeindruckende Informationsverarbeitung und Erkennungsleistung. Dieses erfolgreiche Modell der Natur versuchen Wissenschaft und Technik in Form des maschinellen Sehens (engl. *computer vision*) nachzuahmen. Die Ergebnisse bleiben allerdings in vielen Bereichen noch erheblich hinter der Leistungsfähigkeit der visuellen Wahrnehmung des Menschen zurück, insbesondere wenn es um Erkennen und Verstehen komplexer Strukturen geht. Durch Fortschritte beim Verständnis der zugrundeliegenden Wahrnehmungsprozesse und eine ausgereifte mathematische Modellierung des Abbildungsprozesses kann die Computervision mittlerweile jedoch in einigen Anwendungsbereichen das natürliche Vorbild übertreffen.

Eines dieser Gebiete ist die räumliche Rekonstruktion von Szenen aus der Beobachtung mehrerer verschiedener Ansichten, das Anwendungsfeld der klassischen Photogrammetrie [Kraus 1993; Kraus et al. 1997]. Ein Untergebiet, die Nahbereichsphotogrammetrie, spezialisiert sich dabei auf die Erfassung von Objekten im Nahfeld [Luhmann 2010]. Im Unterschied zur Fernerkundung liegt hier ein beschränkter Erfassungsbereich mit nur wenigen, definierten Objekten vor, welche mit speziell gestalteten Passmarken versehen sind. Der Detektionsvorgang für solche wohldefinierten Markierungen ist gut automatisierbar und mit der Verfügbarkeit digitaler Bildsensoren und leistungsfähiger Bildverarbeitungsverfahren auch einer Datenauswertung in Echtzeit zugänglich. Diese dynamische Online-Photogrammetrie wird als optisches Tracking bezeichnet [Foxlin et al. 2002; Bishop et al. 2001] und ist Kernthema der vorliegenden Arbeit. Während viele Aspekte des optischen Trackings wie Kameramodellierung, räumliche Rekonstruktion, Genauigkeitsfragen etc. direkt den Problemstellungen der klassischen Nahbereichsphotogrammetrie entsprechen, kommen hier

durch die Echtzeitanforderung neue Kriterien während des Systementwurfs hinzu, wie etwa Anforderungen hinsichtlich ausreichender Geschwindigkeit und niedriger Latenzzeiten oder zur Robustheit der Systeme gegenüber Störungen des Datenaufnahmestroms.

Fertige Systemlösungen für optisches Tracking existieren mittlerweile in einer Vielzahl von Varianten, die auf ihre jeweiligen Einsatzgebiete zugeschnitten sind. Sie haben sich aufgrund von Preis, Leistungsfähigkeit und Genauigkeit als Standardverfahren etabliert und andere Arten der Bewegungserfassung vielfach verdrängt. So spielt optisches Tracking eine wichtige Rolle in der industriellen Bildverarbeitung bei Automatisierung, Prozessüberwachung und Qualitätskontrolle [Maas 1992; Wiora et al. 2004; Luhmann 2000], aber auch in Bereichen wie der Filmindustrie [Moeslund und Granum 2001; Foxlin et al. 2002] oder der Medizin [Mitchell und Newton 2002].

Die Medizintechnik ist auch das zentrale Anwendungsgebiet des in dieser Arbeit vorgestellten Systems. Die Einsatzmöglichkeiten reichen dabei von der Diagnostik über die Ausbildung bis hin zur Intervention. Bei computernavigierten Operationen ermöglichen optische Trackingsysteme die exakte räumliche Erfassung von Patient und Operationsinstrumenten (Image Guided Surgery) [Grimson et al. 1999; Cinquin et al. 1995]. Dadurch erhält der Chirurg eine präzise Rückmeldung über die korrekte Positionierung der Instrumente und Unterstützung bei der Navigation zu präoperativ geplanten Positionen. In der Diagnostik werden optische Trackingsysteme (OTS) verbreitet zur Erfassung der Bewegungsabläufe von Patienten (Motion Capture) eingesetzt [Zhou und Hu 2004] oder auch zur Überwachung der Patientenposition bei MRT- oder CT-Aufnahmen [Dold et al. 2006]. Ein weiteres großes Anwendungsfeld ist der Einsatz in der Mensch-Maschine-Schnittstelle von medizinischen Trainingssystemen zur Erfassung der Benutzerinteraktion, ein Schwerpunkt der vorliegenden Arbeit [Schuppe et al. 2009; Liu et al. 2007; Sielhorst et al. 2004; Hamza-Lup et al. 2004].

1.2 Motivation

Ausgangspunkt für diese Arbeit war eine Problemstellung, die bei der Entwicklung des ophthalmo-chirurgischen Simulators EYESI [Schill et al. 1999; Wagner et al. 2002], respektive der Bewegungserfassung für dessen Benutzerschnittstelle, auftrat. Unsere Arbeitsgruppe VIPA (Virtuelle Patienten Analyse) entwickelt Technologien und Verfahren für Simulatoren, die mit Hilfe von virtueller Realität (VR) eine medizinische Intervention nachbilden. Solche Simulatoren ermöglichen Chirurgen das Durchspielen von Eingriffen ohne Risiko für den Patienten. Neben der eigentlichen biomechanischen Simulation und dem graphischen Rendering stellt die Interaktion

mit dem Operationssitus einen wesentlichen Faktor für den Echtheitseindruck dieser simulierten Interventionen dar. Bewegungen des Benutzers müssen robust und verzögerungsarm erfasst werden, um den Eindruck einer realen Operation zu schaffen und eine überzeugende Immersion in die künstliche Welt zu gewährleisten. Bei Simulatoren, die keine aktive Kraftrückkopplung (Force-Feedback) benötigen, bieten sich hierfür optische Trackingverfahren an, da sie schnell, präzise und berührungslos arbeiten und gleichzeitig wenig aufwendig in der mechanischen Realisierung sind.

Bei der Entwicklung zeigte sich allerdings, dass verfügbare optische Trackingsysteme nur unzureichend an die speziellen Randbedingungen solcher Trainingssimulatoren adaptierbar sind. Die von EYESI simulierten Eingriffe am Auge beispielsweise umfassen einen Arbeitsbereich von wenigen Kubikzentimetern; die zu trackenden Objekte sind miniaturisierte nadelförmige Instrumente, die sich zudem noch gegenseitig verdecken können. Kommerzielle Komplettsysteme für optisches Tracking mit großen Kameramodulen und großflächigen Objektmarkern sind für ein solches Setup nicht ausgelegt. Der alternative Einsatz von konventionellen digitalen Videokameras zusammen mit existierenden Computervision-Softwarelösungen scheitert an den geforderten hohen Updateraten und niedrigen Latenzzeiten für die flüssige Interaktion mit der virtuellen Umgebung. Offensichtlich erfordern die speziellen Gegebenheiten bei medizinischen Simulatoren maßgeschneiderte Trackinglösungen, wie sie für den Fall von EYESI z.B. in [Ruf 2000] und [Wagner 2003, Kap. 7.3.2] beschrieben sind.

Mit dem Hinzukommen weiterer Projekte in der Arbeitsgruppe mit jeweils sehr individuellen Anforderungen an ein optisches Trackingsystem, entstand die Notwendigkeit, eine aufwendige, wiederholte Neuentwicklung solcher spezialisierten Trackinglösungen zu vermeiden. Es war vielmehr ein Konzept gesucht für ein universelles, flexibel adaptierbares Baukastensystem für optische Trackingaufgaben, das die Starrheit fertiger Systemlösungen vermeidet und mit moderatem Aufwand an die Anforderungen verschiedenartiger medizintechnischer Anwendungen anpassbar ist. Ein solches System und seine Umsetzung beschreibt die vorliegende Arbeit: das Projekt **MOSCOT**, ein **mo**dulares **sc**alierbares **o**ptisches **T**rackingsystem.

1.3 Zielsetzung und Schwerpunkte

Zielsetzung des Projektes MOSCOT war die Schaffung eines universellen wiederverwendbaren Frameworks für die Entwicklung anwendungsspezifischer optischer Trackinglösungen. Mit den Erfahrungen aus dem EYESI-Projekt und weiteren Vorläufersystemen wurden folgende zentrale Anforderungskriterien für MOSCOT aufgestellt:

- **Universalität und Flexibilität:** Das System soll flexibel an verschiedenartige Trackingszenarien anpassbar sein. Die Unterstützung unkonventioneller Trackingsetups mit speziellen Anforderungen (z.B. sehr kleines Trackingvolumen, geringe Latenzzeiten, flexible Rekonstruktionsgeometrien, anwendungsspezifische Markereigenschaften, deformierbare Objekte etc.) soll gegeben sein. Dies bedingt die Auslegung für den Einsatz verschiedenartiger Marker- und Kameratypen sowie die Bereitstellung anpassbarer Verarbeitungsressourcen.
- **Wiederverwendbarkeit und Erweiterbarkeit:** Das System soll einfach erweiterbar sein, um neuartige bzw. weiterentwickelte Kamerasensoren und Bildverarbeitungshardware einsetzen zu können. Bestehende Hard- und Softwarekomponenten sollen zusammen mit existierenden Komponenten unkompliziert in einer Art Baukastensystem zu einer neuen Anwendung integrierbar sein.
- **Skalierbarkeit und Redundanz:** Die Architektur des Systems soll von Anfang an nicht auf eine vorgegebene Anzahl von Kameras oder Markern eingeschränkt sein, sondern mit der Aufgabenstellung skalieren können. Zusätzliche Informationen sollen zum Aufbau von Redundanz in der Erfassung der Trackingszenerie genutzt werden.
- **Niedrige Latenzzeiten:** Das System muß niedrige Verarbeitungslatenzen aufweisen, um einen Einsatz in interaktiven Anwendungen wie medizinischen Simulatoren zu ermöglichen.
- **Geringer Ressourcenbedarf:** Der Hostrechner in einem medizinischen Simulator ist bereits mit Aufgaben wie Gewebesimulation, Kollisionserkennung und grafischem Rendering größtenteils ausgelastet. Der zusätzliche Rechenzeitbedarf für das optische Tracking sollte daher möglichst niedrig gehalten werden.
- **Anpassbare Genauigkeit:** Da Genauigkeit bei Trackingsystemen direkt mit Qualität, Aufwand und damit dem Preis zentraler Komponenten korreliert, soll hier durch Anpassbarkeit dieser Kernkomponenten eine Adaption an die Anforderungen der jeweiligen Applikation möglich sein.
- **Test- und Verifizierbarkeit:** Es soll eine einfach einsetzbare Testumgebung für Trackinganwendungen bereitgestellt werden. Neuartige Trackingansätze sollen einfach verifiziert sowie ihre Ergebnisse auch unabhängig von der Anwendung visualisiert werden können. Eine Simulation des Trackingsetups und -prozesses vor der eigentlichen Anwendungsrealisierung ist sinnvoll.
- **Rapid-Application-Development:** Die Auslegung der Hard- und Softwarearchitektur soll ein Framework für eine schnelle und effiziente Entwicklung schaffen, so dass die Zeit zur Anpassung des Systems an neue Anwendungen deutlich verkürzt werden kann. Die Integration von Hard- und Software soll kurze Entwicklungszyklen ermöglichen.

Der Schwerpunkt der vorliegenden Dissertation liegt damit auf der Entwicklung einer leistungsfähigen modularen Systemarchitektur, die Flexibilität und Anpassbarkeit des Systems sicherstellt, bei gleichzeitiger Auslegung für hohe Geschwindigkeiten und niedrige Latenzen. Darüber hinaus befasst sich die Arbeit mit speziellen Verfahren und Techniken für einzelne Komponenten des Systems, um damit konkrete Trackingproblemstellungen in den medizintechnischen Projekten der Arbeitsgruppe zu lösen. Darunter fallen die Benutzerschnittstelle verschiedener Simulatoren für ophthalmologische, neurochirurgische und endoskopische Operationen, aber auch die Positionsüberwachung eines Operationsroboters.

1.4 Überblick über die Arbeit

Einbettung in die Arbeitsgruppe

Die Arbeitsgruppe VIPA ist am Institut für computerunterstützte Medizin (ICM) der Universitäten Heidelberg und Mannheim angesiedelt. Forschungsschwerpunkte sind die Entwicklung von Simulator- und VR-Technologien für medizinische Anwendungen. Die VRmagic GmbH ist eine Ausgründung aus dieser Gruppe und führt die Arbeiten in den Bereichen anwendungsnahe Forschung, Produktentwicklung und Vermarktung medizinischer Simulatoren und Kameratechnologien weiter. Im Rahmen der gemeinsamen Forschungsarbeiten wurde die VRM-Architektur für medizinische Simulatoren (*Virtual Reality in Medicine*) entwickelt. Diese umfasst Themenschwerpunkte in den Gebieten Simulatorarchitektur, physikalische Simulation, Grafik und Haptik, die bereits in Dissertationen von Schill [2001], Wagner [2003], Körner [2003] und Grimm [2005] vertieft wurden. Die vorliegende Arbeit befasst sich mit dem Teilgebiet des optischen Trackings. Die Softwarekomponenten des Projektes MOSCOT stellen dabei einen Teil der VRM-Gesamtarchitektur dar und bieten den verschiedenen medizintechnischen Anwendungen der Arbeitsgruppe einfachen und direkten Zugriff auf Trackingfunktionalität. Einzelne Themengebiete des optischen Trackings wurden auch im Rahmen von Diplomarbeiten behandelt, unter anderem von Panse [2002], Handel [2004], Hegner [2005], Beier [2006] und Hoffmann [2006]. Weitere Dissertationen in der Arbeitsgruppe befassen sich mit speziellen Teilaspekten eines optischen Trackingsystems: Handel [2009] untersucht den Einsatz komplexerer Marker und eine erhöhte Systemgenauigkeit durch verbesserte Kameramodellierung und Yang [2012] befasst sich mit der Entwicklung leistungsfähiger Hardware für erweiterte Bildverarbeitungsansätze. Abbildung 1.1 gibt einen Überblick über die erwähnten Arbeiten sowie ihren Zusammenhang mit der VRM-Architektur und dem Themengebiet optisches Tracking. In enger Kooperation mit der VIPA-Gruppe arbeiten bei der VRmagic GmbH mehrere Entwickler an Umsetzungen der MOSCOT-

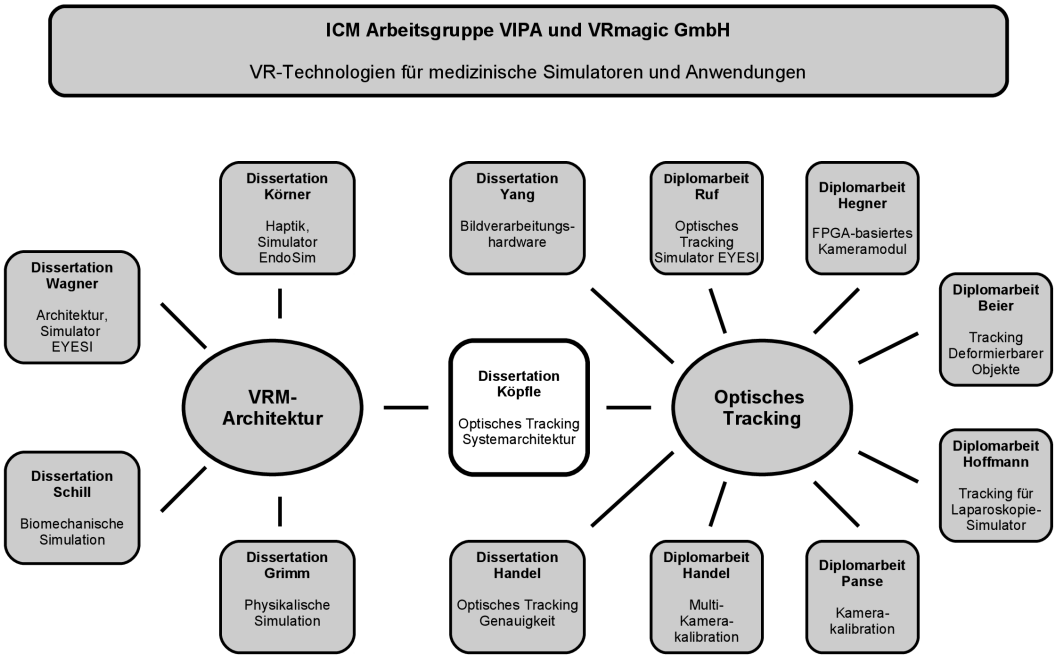


Abbildung 1.1: Überblick über die mit der VRM-Architektur und dem MOSCOT-Trackingsystem in Zusammenhang stehenden Arbeiten.

Systemarchitektur in Simulatorprodukte.

Gliederung der Arbeit

Nach der Einführung in die Thematik im vorliegenden Kapitel werden in Kapitel 2 Vorüberlegungen zum Stand der Technik und zum Systementwurf angestellt. Grundlagen und Grundbegriffe optischer Trackingsysteme, die für die Ausarbeitung eines universellen Systemkonzepts notwendig sind, werden eingeführt. Danach wird ein Überblick über die Technik verfügbarer Komplettsysteme gegeben sowie Softwarebibliotheken für den Einsatz in Trackinganwendungen beschrieben. Nach einer kurzen Diskussion werden Schlussfolgerungen für das grundlegende Konzept des MOSCOT-Systems benannt.

Kapitel 3 untersucht die grundlegenden Datenverarbeitungsvorgänge während des Trackingprozesses. Verschiedene Phasen der Datenverarbeitung werden identifiziert und die Strukturen des Datenflusses herausgearbeitet. Der Rest des Kapitels widmet sich dann einzelnen Methoden von Bildverarbeitung, Markerdetektion und 3D-Rekonstruktion. Die für die bisherigen Anwendungen umgesetzten Verfahren werden ausführlich erläutert und kurz mögliche Erweiterungen skizziert. Ein Abschnitt über Verfahren zum robusten Tracking bei fehlerbehafteten Daten beschließt das Kapitel.

Kapitel 4 erläutert die Umsetzung der Datenverarbeitungsstrukturen in eine Gesamtarchitektur für das MOSCOT-System. Das modulare Systemkonzept wird eingeführt und in seinen zentralen Merkmalen näher beschrieben. Im Anschluss wird auf die Realisierung des Konzepts auf Seiten der Hardware- und der Softwarearchitekur eingegangen. Anhand des Aufbaus einer „intelligenten“ Kamera werden einzelne Hardwarekomponenten exemplarisch erläutert. Auf der Softwareseite wird die Implementierung der Datenflussarchitektur dargestellt. Dazu zählt insbesondere die objektorientierte Modellierung von Verarbeitungs- und Kommunikationsstrukturen sowie von Daten- und Nachrichtenobjekten. Abschließend werden einige Beispielkonfigurationen der Hard- und Softwarearchitektur vorgestellt.

Als graphisches Software-Frontend für den Nutzer stellt das MOSCOT-System die Konfigurations- und Entwicklungsumgebung „Tracklab“ bereit, deren Aufbau in Kapitel 5 kurz dargestellt wird. Ausgehend von den Anforderungen von Anwendern und Entwicklern wird sie als Applikationsframework für Entwicklung und Einsatz von Trackinglösungen entworfen. Es wird auf Softwarearchitektur, Komponenten, Datenfluss und die bereitgestellte Programmierschnittstelle eingegangen. Die Beschreibung einiger exemplarischer Beispielmodule für Trackingbetrieb und Systemkontrolle beschließt das Kapitel.

Kapitel 6 beschreibt die Realisierung einiger konkreter Trackingproblemstellungen mit Hilfe des MOSCOT-Systems. Dabei werden jeweils das Trackingsetup mit den Anforderungen, die Anpassung der Systemarchitektur an die konkrete Aufgabenstellung sowie die Ergebnisse kurz zusammengefasst. Die vorgestellten Anwendungen demonstrieren den Einsatz unter den besonderen Randbedingungen von medizinischen Simulatoren für Intervention und Diagnostik, ebenso wie bei einem eher konventionelles Trackingsetup zur Navigation eines Operationsroboters. Einige spezielle Eigenschaften des MOSCOT-Systems wie die Simulation von Trackingsystemen, Rapid-Application-Development und Erweiterung hinsichtlich unüblicher Aufgabenstellungen wie der Erfassung von Deformationen werden ebenfalls veranschaulicht.

Kapitel 7 fasst im Rahmen einer abschließenden Diskussion die Ergebnisse der vorliegenden Arbeit zusammen und gibt einen Ausblick auf Schwerpunkte der zukünftigen Weiterentwicklung.

Anmerkungen zur Darstellung

Englische Fachbegriffe sind im deutschen technischen Sprachgebrauch alltäglich. In dieser Arbeit werden deutsche Begriffe den englischen vorgezogen, soweit sie gebräuchlich und unter Technikern verständlich sind (bspw. Bibliothek gegenüber Library). Wenn die Übersetzung ins Deutsche die Verständlichkeit mindert, wird stattdessen der englische Terminus in eingedeutscher Schreibweise verwendet (Bsp.: Template-Programmierung gegenüber Schablonen- oder Vorlagenprogrammierung).

Internetverweise werden verwendet, wo aus Gründen der Verfügbarkeit oder aus Mangel an gedruckter Literatur nicht auf sie verzichtet werden kann. Es wurde darauf geachtet, dass zum Veröffentlichungszeitpunkt dieser Arbeit alle Links erreichbar und aktuell waren.

Kapitel 2

Grundlagen und Vorüberlegungen zum Systemkonzept

Dieses Kapitel gibt eine Einführung zu Grundlagen und Stand der Technik optischer Trackingsysteme. Es werden Aufbau, Systemelemente sowie grundlegende Eigenschaften der Systeme beschrieben. Dabei werden auch Vorüberlegungen im Hinblick auf die Entwicklung eines universellen Systems wie MOSCOT und seines späteren Einsatzes in den Projekten der Arbeitsgruppe angestellt. Eine Aufstellung verfügbarer Lösungen für optische Trackingaufgaben bildet den zweiten Teil des Kapitels. Verschiedene Ansätze kommerzieller Systemlösungen werden vorgestellt, die speziell auch im medizinischen Umfeld Verwendung finden. Darüber hinaus wird auf Bibliotheken und Softwaretoolkits eingegangen, die verschiedene Aspekte der Realisierung von Trackinganwendungen abdecken. Eine Diskussion der vorhandenen Lösungen und hieraus resultierende Schlussfolgerungen für den Entwurf eines Systemkonzepts beschließen das Kapitel.[1]

2.1 Optisches Tracking und weitere Verfahren zur Bewegungserfassung

Neben optischem Tracking existiert eine Reihe alternativer Verfahren zur dreidimensionalen Bewegungserfassung. Die Hintergründe, die zur Wahl eines optischen Messverfahrens für die medizintechnischen Anwendungen der Arbeitsgruppe VIPA führten, sollen nun kurz erläutert werden.

[1] An dieser Stelle wird nur auf den Stand der Technik bei Komplettsystemen und Softwarebibliotheken eingegangen. Methoden und Lösungsansätze für einzelne Verarbeitungsschritte werden in Kapitel 3 erläutert.

2.1.1 Übersicht verschiedener 3D-Messverfahren

Die in der Literatur beschriebenen 3D-Messverfahren basieren auf verschiedenen physikalischen Messprinzipien: mechanisch, akustisch, optisch, magnetisch, inertial.

- Mechanische Verfahren auf Basis von Gelenkarmen mit Encodern oder Schnursystemen sind die ältesten Verfahren. Sie erreichen mit kostengünstigen Mitteln eine recht hohe Genauigkeit. Aufgrund der komplexen mechanischen Konstruktion und Einschränkungen bei Freiheitsgraden, Platzbedarf, Verschleiß etc. werden sie aktuell nur noch selten eingesetzt. Vorteilhafte Anwendungsmöglichkeiten ergeben sich immer dann, wenn ohnehin eine mechanische Kopplung mit den zu erfassenden Objekte erforderlich ist, beispielsweise in medizinischen Simulatoren für eine Benutzerschnittstelle mit Kraftrückkopplung (Force Feedback).
- Akustische Verfahren sind ebenfalls eine etablierte, kostengünstige Technologie, die z.B. vielfach in der mobilen Robotik eingesetzt wird. Sie basieren auf der Messung von Laufzeitunterschieden reflektierter Ultraschallsignale. Die erreichbare Genauigkeit ist allerdings unzureichend für die meisten medizintechnischen Anwendungsbereiche und die Systeme sind anfällig für Störungen durch Reflexionen und Interferenzen.
- Optische Verfahren haben sich mit dem Aufkommen digitaler optischer Sensoren weit verbreitet. Sie bieten eine hohe Genauigkeit kombiniert mit hoher Geschwindigkeit bei gleichzeitig moderaten Kosten. Problematisch ist ihre Verschattungsempfindlichkeit, also die Anfälligkeit für Verdeckungen in den Sichtlinien zwischen Sensor und Objekt. Diesem zentrale Problem optischer Verfahren kann durch redundante Auslegung der optischen Sensoren begegnet werden.
- Elektromagnetische Verfahren werden erst seit kurzem in speziellen Anwendungsbereichen eingesetzt. Das Ortungsprinzip basiert auf der Messung der räumlichen Ausrichtung innerhalb eines Magnetfeldes, typischerweise mittels miniaturisierter Spulen, zuweilen auch mittels GMR-Sensoren[2]. Ein großer Vorzug des Messprinzips liegt darin, dass keine freie Sichtlinie zum Messobjekt bestehen muss. Allerdings erreichen die Systeme nicht die Genauigkeit von optischen Verfahren und sind störanfällig gegenüber ferromagnetischen Objekten innerhalb des Magnetfeldes. Es existieren zweckmäßige Kombinationsmöglichkeiten mit optischem Tracking, insbesondere bei der Messung zusätzlicher Freiheitsgrade mit geringeren Genauigkeitsanforderungen (vgl. beispielsweise die in Abschnitt 6.1 beschriebene Anwendung zum Instrumententracking).

[2]Magnetfeldsensoren, die auf dem GMR(Giant-Magneto-Resistive)-Effekt basieren

- Inertiale Messverfahren erfassen mittels Accelerometern und Gyroskopen die translatorischen und rotatorischen Beschleunigungen des Messobjekts. Numerische Integration führt sodann zu den grundlegenden Größen Position und Orientierung. Systemimmanent erfolgt dadurch über die Zeit eine Fehlerakkumulation, die nur unzureichend kompensiert werden kann. Bislang sind die notwendigen Sensoren zudem recht groß und teuer. [3] Vorteilhaft ist in jedem Fall die Autonomie der Sensoren, was zur Vermeidung der Verdeckungsproblematik führt und zu einem praktisch unbegrenzten Arbeitsvolumen verhilft.

Ausführlichere Betrachtungen zu den Verfahren sowie ihrer spezifischen Vor- und Nachteile für ausgewählte Messaufgaben finden sich beispielsweise in [Bishop et al. 2001] und [Foxlin et al. 2002], zusammen mit einer Auflistung einiger verfügbarer Systeme auch in [Buaes 2005], [Zhou und Hu 2004] und [Korb 2007].

Optische Lösungen haben sich in den letzten Jahren in vielen Anwendungsbereichen durchgesetzt und als universelle Standardverfahren etabliert. In der Gesamtheit weisen sie keine größeren Defizite auf, sind kostengünstig und flexibel, einfach zu realisieren und skalieren auch bestens über Volumen und Objektanzahl. Damit sind bestens geeignet für den Einsatz in medizinischen Simulatoren und ähnlichen medizintechnischen Anwendungen und wurden auch als Basis für die Bewegungserfassung in den Projekten unserer Arbeitsgruppe gewählt.

Für die Zukunft scheint auch eine Ergänzung optischer Verfahren durch kleine Inertialsensoren in Form mikro-elektro-mechanischer Systeme vielversprechend, da beide Verfahren sich in ihren zentralen Eigenschaften (absoluter/relativer Fehler, Messgeschwindigkeit, Messung von Position bzw. Beschleunigung, Verdeckungsempfindlichkeit) gegenseitig ergänzen.

2.1.2 Optische 3D-Messverfahren und optisches Tracking

Die optisch basierten 3D-Messverfahren können anhand des spezifischen Messprinzips weiter unterteilt werden. Hier soll eine Einordnung angelehnt an Schwarte et al. [1999] vorgenommen werden, die die Messverfahren in Triangulation, Interferometrie und Laufzeitmessung unterteilt (Abbildung 2.1). Für die Positionsbestimmung von Objekten werden bevorzugt Triangulationsverfahren eingesetzt. Sie basieren auf dem seit dem Altertum bekannten Messprinzip der Geodäsie: sie nutzen die Parallaxe, d.h.

[3] In der Zeit seit dem Beginn dieser Arbeit ist mit der Einführung sog. MEMS (mikro-elektromechanischer Systeme in Form von kostengünstigen und platzsparenden integrierten Baugruppen) auch die Inertialmesstechnik für Trackinganwendungen in medizinischen Anwendungen einsetzbar geworden. Solche Sensoren sind hier noch nicht etabliert, es entfallen jedoch die genannten Nachteile bzgl. Kosten und Raumbedarf.

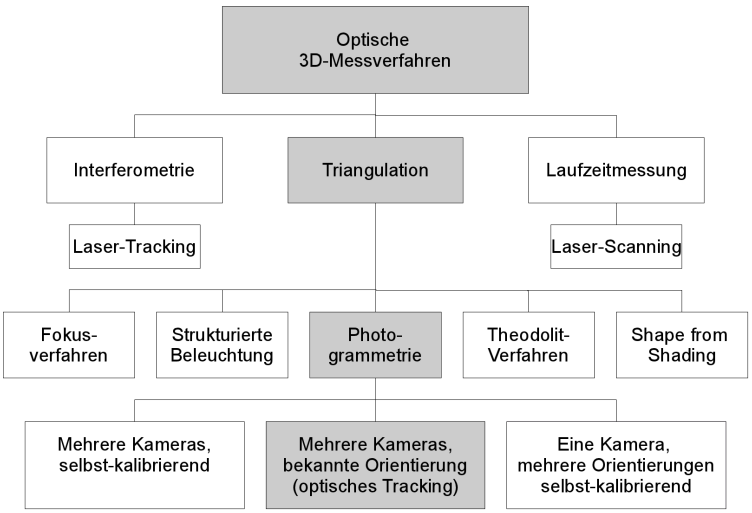


Abbildung 2.1: Schematische Einteilung der optischen 3D-Messverfahren nach Schwarte et al. [1999])

den Winkelunterschied verschiedener Beobachtungspunkte zum zu messenden Objekt aus. Abbildung 2.2 veranschaulicht das Prinzip. Beispiele für Triangulationsverfahren sind stukturierte Beleuchtung, Winkelmessverfahren (Theodolitverfahren) sowie als wichtigstes Anwendungsgebiet die Photogrammetrie.

Aus der Verbindung von klassischer Photogrammetrie im Nahbereich und den digitalen Bildverarbeitungsverfahren der Computer Vision entstammt das optische Tracking, quasi eine digitale Nahbereichsphotogrammetrie unter Echtzeitbedingungen.[4] So interessieren durch den dynamischen Messvorgang neben Aspekten der räumlichen Auflösung und Genauigkeit auch die zeitlichen Parameter des Abtastvorgangs. Redundanz bei der Datenerfassung ist nur durch Parallelmessung und nicht durch sequentielle Vielfachmessung möglich. Und schließlich kommt Aspekten des Systementwurfs wie Systemarchitektur, Echtzeitfähigkeit der Datenverarbeitung und dynamischer Fehlerbehandlung bei optischen Trackingverfahren eine größere Bedeutung zu als in traditionellen Photogrammetrieanwendungen.

[4]Die weiter unten dargestellte Spezialform trinokularer Trackingsysteme mit Zeilenkameras stellt bei rein formaler Betrachtung der Abbildungs- und Rekonstruktionsgeometrie ein Thedolitverfahren und kein photogrammetrisches Verfahren dar. Für den weiteren Verlauf dieser Arbeit ist dieser Unterschied jedoch nicht relevant und soll hier daher nicht weiter ausgeführt werden.

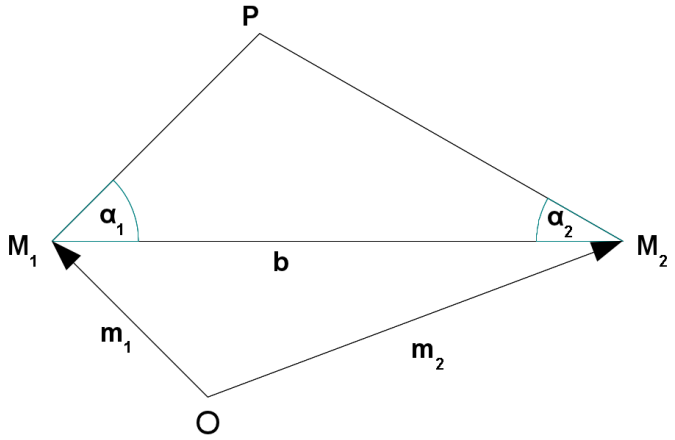


Abbildung 2.2: Grundprinzip der Triangulation (hier im zweidimensionalen Fall). Von zwei verschiedenen Messpunkten $M1$ und $M2$ wird der Objektpunkt P angepeilt. Über die bekannte Basislänge b und die beiden gemessenen Beoboachtungswinkel $\alpha 1$ und $\alpha 2$ kann die Lage von P relativ zu $M1$ und $M2$ bestimmt werden. Bei bekannter Lage von $M1$ und $M2$ folgen daraus die Koordinaten von P relativ zum Koordinatenursprung O.

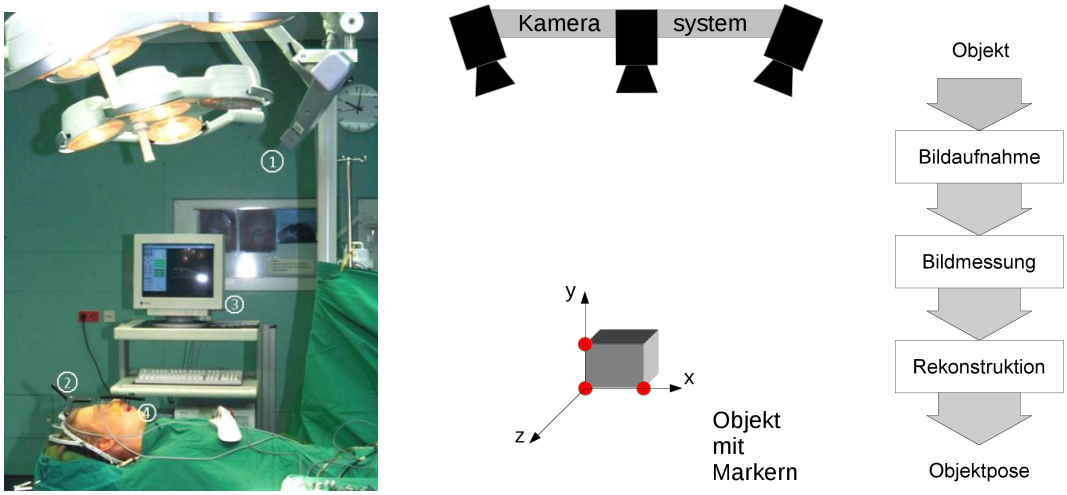
2.2 Grundlagen optischer Trackingsysteme

Im Folgenden werden einige zentrale Aspekte optischer Trackingsystemen erörtert: Grundlagen zu Systemaufbau, Marker- und Kameratechnik, aber auch Randbedinungen der räumlichen und zeitlichen Abtastung. Sie werden im Hinblick auf die intendierten Einsatzgegebenheiten des MOSCOT-Systems analysiert und Schlussfolgerungen für den Entwurf eines Systemkonzepts gezogen.

2.2.1 Prinzipieller Aufbau und Messprinzip

Primäres Ziel des Trackingprozesses ist die kontinuierliche räumliche Erfassung von Position und Orientierung (der Pose) eines oder mehrerer Objekte im zeitlichen Verlauf. Für eine effiziente und robuste Erfassung des Objektes werden einfach detektierbare Markierungen am Objekt angebracht. Der elementare Fall eines kugel- oder punktförmigen Markers definiert eine dreidimensionale Position (Translation) relativ zum Ursprung des Trackingraums. Nicht rotationssymmetrische Objekte, bei denen zusätzlich die Orientierung (Rotation um die drei Raumachsen) bestimmt werden soll, erfordern die gleichzeitige Messung von mehreren Punktmarkern. Dies wird im

Unterschied zur reinen Positionsmessung mit drei Freiheitsgraden (degrees of freedom, 3DOF) als 6DOF-Messung (Position und Orientierung) bezeichnet.[5]



(a) System im medizinischen Einsatz[6]

(b) Schematischer Überblick der Komponenten

(c) Verarbeitungsprozess

Abbildung 2.3: Überblick über Systemaufbau und Messprinzip eines optischen Trackingsystems. Die Kameras (1) erfassen Marker (2), die am Trackingobjekt (im Bild links ein Fixierrahmen am Kopf des Patienten) angebracht sind. Über die Schritte der Bildmessung und Rekonstruktion wird hieraus die Objektpose im Kamerakoordinatensystem bestimmt.

Abbildung 2.3 gibt einen Überblick über den Systemaufbau und den Messvorgang eines optischen Trackingsystems. Mehrere Kameras bilden die Szene aus verschiedenen Beobachtungspositionen ab. Über die Schritte der Bildmessung und Rekonstruktion wird aus den erfassten Bilddaten die räumliche Lage der Objekte rekonstruiert (Abbildung 2.3(c)). In diese Schritte fließen vor Beginn des Trackingprozesses ermittelte Systeminformationen ein (parametrisierte Beschreibungen von Kameras, Markern und Objekten). Die Kameras sind in ihrer räumlichen Konfiguration zueinander fixiert und bilden das Kamerakoordinatensystem. Gewöhnlich ist entweder das Kamera- oder das Objektsystem absolut im Raum fixiert und die Bewegung des jeweils anderen Systems wird erfasst. Es sind jedoch auch Einsätze mit bewegtem Kamerasystem und bewegten Objekten möglich, wenn lediglich die Relativposition

[5] Für die Bestimmung der Rotation ist eine unveränderliche Anordnung der Referenzpunkte notwendig, so dass bei der üblichen Messanordnung nur die Rotation starrer Objekte erfasst werden kann.

[6] Bild von RüdigerMarmulla/commons.wikimedia.org, Lizenz: CC-BY-SA

Relevanz besitzt. Rekonstruierte Objektpositionen werden daher vom Trackingsystem immer relativ zum kamerabasierten Bezugssystem geliefert, eine etwaige Transformation in Weltkoordinaten bleibt der übergeordneten Anwendung vorbehalten.

2.2.2 Formale Beschreibung der räumlichen Konfiguration

Mit der zunehmenden Verbreitung optischer Trackingsysteme in jüngerer Zeit kommen verstärkt Bestrebungen auf, die Beschreibung von Trackingszenarien und -setups zu formalisieren. Solche Ansätze sind noch recht neu und daher nicht weit verbreitet bzw. auch keineswegs standardisiert. Wegen ihrer abstrakten Sichtweise bieten sie nichtsdestotrotz einen guten Ausgangspunkt für die Erarbeitung angepasster Lösungen für verschiedartige Trackingszenarien. Einen Ansatz, der im weiteren Verlauf dieser Arbeit verwendet wird, präsentieren Newman et al. [2003]; Newman et al. [2004] mit den sog. Spatial-Relationship-Graphen (SRG). Pustka et al. [2006] erweitern diese Darstellung auf eine formale Beschreibung für Kalibrierung und automatisches Setup eines optischen Trackingsystems. Im Folgenden wird eine Darstellung angelehnt an die Diagramme aus [Pustka et al. 2006] verwendet.

Bei SRGs werden ähnlich den aus der Computergrafik bekannten Szenegraphen [Strauss 1993] mittels einer Graphstruktur die abstrakten geometrischen Relationen von Objekten, Markern und Kameras beschrieben, sowie die Transformationen zwischen den ihnen zugeordneten Koordinatensystemen. Die Knoten des Graphen entsprechen dabei realen oder virtuellen Entitäten des Trackingsetups (Kameras, Abbildungsebenen, Objekten, Markern, etc.). Die Kanten des Graphen repräsentieren die räumliche Beziehung bzw. den Formalismus zur Transformation zwischen den beteiligten Knoten. Formal definiert sich ein SRG als ein Graph

$$G = (V, E)$$

mit der Knotenmenge V aller Entitäten und der Kantenmenge

$$E \subseteq V \times V$$

der gerichteten räumlichen Relationen. Die Kanten können zusätzliche Attribute tragen wie den Typ der Transformation (6D, 3D, 3D→2D (projektive Abbildung)), Kontinuität der räumlichen Relation (statisch oder dynamisch) oder auch Herkunft der Transformationsinformation (Messung, abgeleiteter Wert, gesuchte Relation). Da mehrere Kanten zwischen zwei Knoten existieren können stellt E eine Multimenge dar. Auslassungspunkte im Graphen steht für mehrere gleichartige Knoten (z.B. Marker) oder Kanten (z.B. Messungen). Abb 2.4 illustriert die Darstellung anhand eines Beispiels.

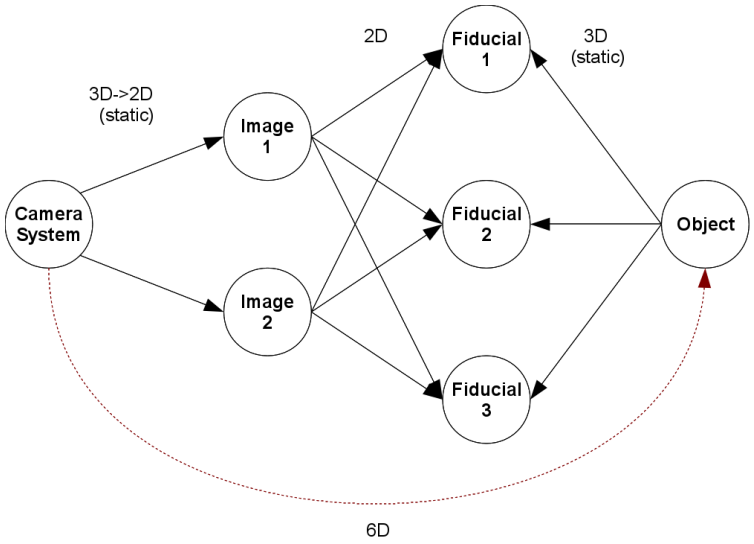


Abbildung 2.4: Beispiel eines Spatial-Relationship-Graphen zur formalen Beschreibung eines optischen Trackingsetups, hier ein Stereokamerasystem mit einem getrackten Objekt mit drei Markern (Fiducials). Gesucht ist die 6DOF-Transformation des Objektes relativ zum Kamerasystem. Ermittelt wird sie aus der bekannten 3D→2D Projektion der Kameraabbildung (bekannt aufgrund der Bestimmung der Kameraparameter während der Kalibrierung), der 2D-Position der Marker in den Kamerabildern (gemessen) sowie der 3D-Anordnung der Marker auf dem Objekt (bekannt aus der Objektregistrierung).

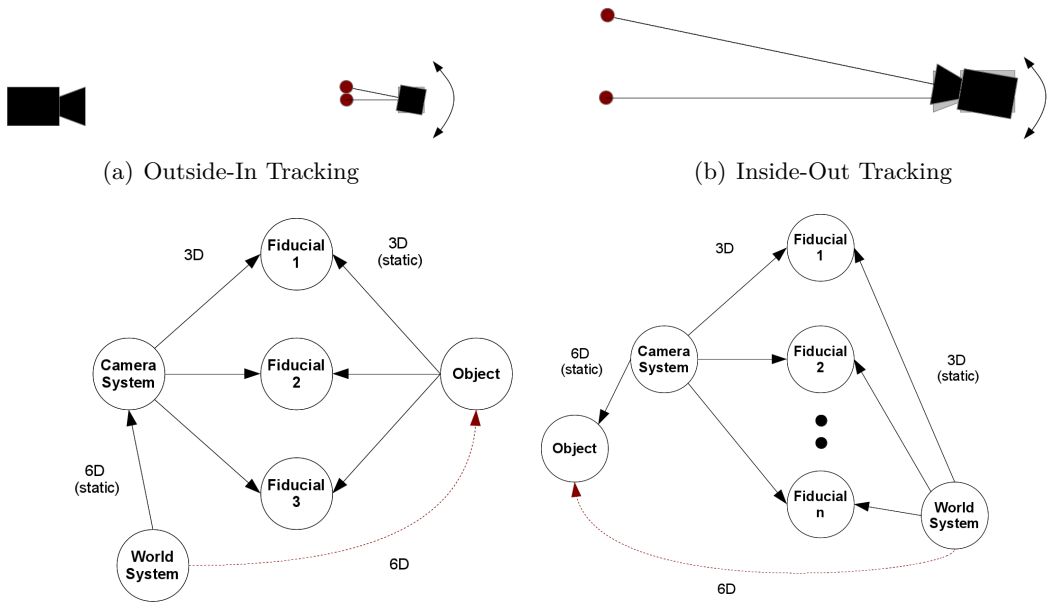
Anwendungsmöglichkeiten existieren sowohl innerhalb eines Systems als auch für die Datenfusion verschiedener Systeme. Obwohl keine Beschreibung des eigentlichen Datenflusses vorliegt, können doch wichtige strukturelle Eigenschaften zu Datenstrukturen und -verarbeitungsalgorithmen für die Systemarchitektur aus der Graphenstruktur entnommen werden. Bei der Beschreibung der Anwendungen in Kapitel 6 werden SRGs zur Beschreibung der Trackingsetups verwendet.

2.2.3 Klassifizierung optischer Trackingsysteme

Optische Trackingsysteme können anhand der Aufnahmekonfiguration, d.h. der relativen Anordung von Markern und Kameras, und anhand der Rekonstruktionsgeometrie in verschiedene Klassen eingeteilt werden:

Aufnahmekonfiguration:

Unter Beachtung eines fixen Weltkoordinatensystems existieren zwei prinzipiell unterschiedliche Anordungen von Kameras und Markern [Ribo 2001, S. 3] [Foxlin et al. 2002, Kap. 8.3.7.1]:



(a) Outside-In Tracking

(b) Inside-Out Tracking

(c) SRG zum Outside-In Tracking

(d) SRG zum Inside-Out Tracking

Abbildung 2.5: Outside-In- bzw. Inside-Out-Aufnahmekonfiguration. Bei Outside-In-Anordnung führt die Rotation des Trackingobjekts zu Markertranslationen im Kamerabild, die proportional zur Ausdehnung der Markeranordnung auf dem Objekt sind. Bei Inside-Out-Anordnung führt die Rotation der mitbewegten Kamera zu Markertranslationen im Kamerabild, die proportional zum Kamera-Marker-Abstand sind. Kleine Rotationsbewegungen lassen sich damit durch die Inside-Out-Konfiguration besser erfassen. Die SRGs zeigen die unterschiedlichen statischen Transformationen von Kameras bzw. Markern zum Weltkoordinatensystem.

Outside-In Bei der Outside-In-Anordnung sind im Weltkoordinatensystem feststehende Kameras rund um die mit Markern präparierten Trackingobjekte herum positioniert. Dies ist die bevorzugte Konfiguration für Anwendungen in geschlossenen Räumen mit statischem Kameraaufbau und bewegten Objekten. Die Mindestanzahl an Markern pro Objekt richtet sich nach der Anzahl der zu detektierenden Freiheitsgrade und gewünschter Redundanz. Der Abstand zwischen Kameras und Markern ist groß im Verhältnis zum Abstand der einzelnen

Objektmarker untereinander. Dies führt zu einer hohen Positionsauflösung und zu einer im Vergleich niedrigeren Winkelauflösung (Abbildung 2.5(a)).

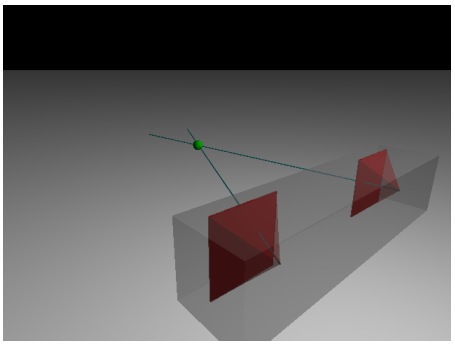
Inside-Out Die Inside-Out-Konfiguration vertauscht die Positionen von Kameras und Markern. Direkt am bewegten Objekt angebrachte Kameras erfassen im Weltkoordinatensystem fixierte Marker, die um die Szene herum im Randbereich des Trackingvolumen angeordnet sind. Die Anzahl an Markern hängt stark von der Ausdehnung des zu erfassenden Raumes ab. Da der Inter-Marker-Abstand im Verhältnis zum Kamera-Marker-Abstand hier größer ausfällt, ergibt sich eine bessere Rotationsauflösung als beim Outside-In-Verfahren. Dieses Setup wird häufig eingesetzt, wenn größere Trackingvolumina abzudecken sind bzw. wenn die zu erfassenden Objekte ohnehin bereits über Strom- und Datenanbindung verfügen, z.B. für die Orientierung mobiler Roboter oder bei der Erfassung der Pose von Head-Mounted-Displays (Abbildung 2.5(b)).

Rekonstruktionsgeometrie:

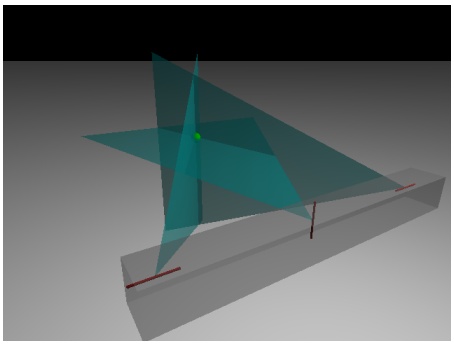
Anhand der Aufnahme- und Rekonstruktionsgeometrie können die Systeme in zwei verschiedene Kategorien klassifiziert werden [Foxlin et al. 2002], [Bishop et al. 2001].

Rekonstruktion durch Geradenschnitt Diese Rekonstruktionsgeometrie resultiert aus dem Einsatz flächiger Bildsensoren (siehe Abschnitt 2.2.5), die eine Abbildung des dreidimensionalen Trackingraumes auf eine zweidimensionale Bildebene ausführen. Jedem Marker wird durch die Abbildungsprojektion der Kamera ein 2D-Bildpunkt auf dem Sensor zugeordnet. Eine Rückprojektion durch das Kamerazentrum definiert einen zugehörigen Sichtstrahl. Der Schnitt korrespondierender Strahlen von mindestens zwei Kameras ergibt die dreidimensionale Rekonstruktion der Markerposition. (Abbildung 2.6(a))

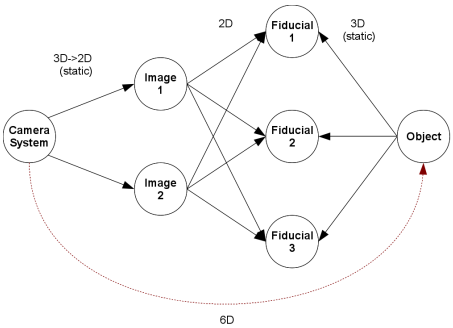
Rekonstruktion durch Ebenenschnitt Hier werden Kameras mit Zeilensensoren (siehe wieder Abschnitt 2.2.5) eingesetzt, die über eine vorgebaute Zylinderoptik eine Abbildung des dreidimensionalen Trackingvolumens auf eine eindimensionale, lineare Sensoranordnung durchführen. Somit ordnet die Kameraprojektion jedem realen Marker eine (eindimensionale) Koordinate auf der Sensorachse zu. Diese kann als Winkelmessung in der Projektionsebene angesehen werden. Die Rückprojektion durch das Kamerazentrum liefert eine Ebene im Raum. Zur Rekonstruktion der Markerposition ist der Schnitt von mindestens drei dieser Sichtebenen erforderlich. (Abbildung 2.6(b))



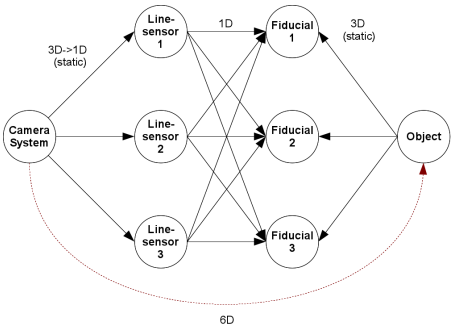
(a) Rekonstruktion über Geradenschnitt bei Flächenkameras



(b) Rekonstruktion über Ebenenschnitt bei Zeilenkameras



(c) SRG zur Rekonstruktion über Geradenschnitt



(d) SRG zur Rekonstruktion über Ebenenschnitt

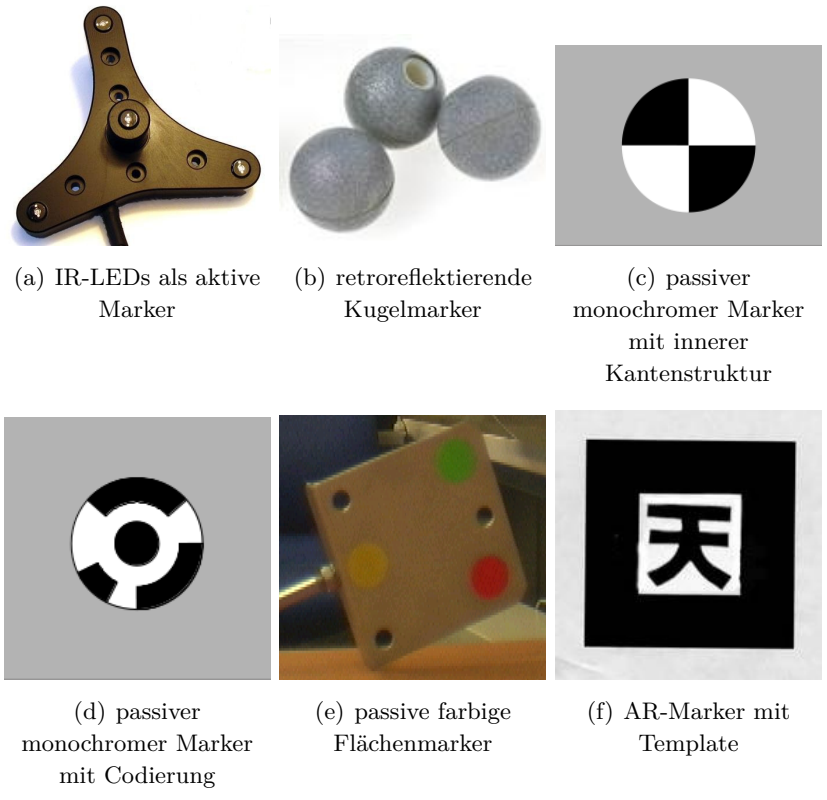
Abbildung 2.6: Rekonstruktion der Markerposition als Schnittpunkt zweier Geraden bzw. dreier Ebenen. Man erkennt die unterschiedlichen Rekonstruktionsgeometrien bzw. Transformationen im SRG.

2.2.4 Markierungs- und Signalisierungstechnik

Zur robusten Erfassung durch das Trackingsystem werden die Messobjekte künstlich signalisiert, d.h. sie werden mit im Kamerabild einfach detektierbaren Markierungen, den Trackingmarkern versehen.[7] Ihre Auswahl stellt das wichtigste Kriterium für die prinzipielle Realisierbarkeit, aber auch die Genauigkeit und Robustheit einer optischen Trackinganwendung dar.

Generell ist auch optisches Tracking ohne Marker möglich, markerbasiertes Tracking bietet jedoch im praktischen Einsatz diverse Vorteile. So sind Marker aufgrund ih-

[7] In der photogrammetrischen Fachliteratur wird der Begriff (Ziel-)Marken (engl. Targets), in der medizinischen, auch der deutschen, Fachliteratur häufig der Begriff Fiducials verwendet.



(a) IR-LEDs als aktive Marker

(b) retroreflektierende Kugelmarker

(c) passiver monochromer Marker mit innerer Kantenstruktur

(d) passiver monochromer Marker mit Codierung

(e) passive farbige Flächenmarker

(f) AR-Marker mit Template

Abbildung 2.7: Beispiele verschiedener Markertypen

rer speziellen Beschaffenheit mit einer höheren Genauigkeit erfassbar, die Detektion benötigt deutlich weniger Ressourcen für die Bildverarbeitung und sie bleibt bei Beleuchtungsänderungen und verschiedenen Kamerasichtwinkeln stabiler. Als Nachteil ist die notwendige Präparation der zu erfassenden Objekte zu nennen. Für die intendierten Anwendungsgebiete im medizintechnischen Umfeld (aber auch in der industriellen Bildverarbeitung) überwiegen jedoch klar die Vorteile des markerbasierten Trackings.

Merkmale verschiedener Markertypen

Im Laufe der Entwicklung optischer Trackingsysteme wurde eine Vielzahl verschiedenartiger Markertypen konzipiert. Abbildung 2.7 zeigt einige gebräuchliche Beispiele. Gemeinsam ist allen Arten von Markern, dass sie über radiometrische oder geometrische Charakteristiken verfügen, anhand derer sie im Kamerabild robust und effizient detektierbar sind. So können selbstleuchtende (Abbildung 2.7(a)) und retro-reflektierende (Abbildung 2.7(b)) Marker anhand des Intensitätsunterschiedes

zwischen Markerbild und Hintergrund isoliert werden. Passive Markertypen werden über radiometrische Unterschiede zwischen Marker und Hintergrund detektiert (Abbildung 2.7(e)) oder über innere Strukturen, die sich robust identifizieren lassen (Abbildungen 2.7(c) und 2.7(d)). Typisch für letztere Art von Markern sind Schwarz-Weiss-Strukturen wie (konzentrische) Kreise, Kreuze oder Kreissektoren. Wichtig ist die gute Segmentierbarkeit der Farbwert- bzw. Kontrastübergänge durch einfache geometrische Primitive wie Geraden oder Kreise. Trotzdem erfordern solche Marker mit innerer Struktur deutlich größtere Ressourcen bei der Bildverarbeitung.

Die folgende Übersicht beschreibt verschiedene Merkmale, anhand derer die verschiedenen Markertypen charakterisiert werden können:

- Art der Signalisierung: Aktive Marker sind selbstleuchtend und werden gewöhnlich in Form von Leuchtdioden (LEDs) realisiert. Retroreflektierende Marker verfügen über eine spezielle Oberfläche, die Licht von um die Kameras herum angebrachten Ringilluminationen direkt zu diesen zurück reflektiert. Passive Marker nutzen das vorhandene Umgebungslicht.
- Radiometrische Charakteristik: Je nach Art der Beleuchtung und Kameras kommt eine Detektion im sichtbaren Spektrum des Lichts oder im nahen Infrarot in Frage. Die Art der ausgewerteten Bildinformation kann sich auf Kontrastwertunterschiede (bei Monochromkameras bzw. Infrarotsignalisierung) oder auf Farbwertunterschiede (für Farbmarker und -kameras) stützen.
- Innere Markerstruktur: Homogene Marker zeigen gleichförmige Bereiche im Kamerabild, die sich über einfache Bildverarbeitungsalgorithmen von der Bildumgebung trennen lassen. Marker mit innerer Struktur erfordern komplexere Algorithmen, sind jedoch meist robuster zu detektieren.
- Verfahren zur subpixelgenauen Detektion: Für hohe Rekonstruktionsgenauigkeit ist es erforderlich, die Markerpositionen im Bild mit Subpixelauflösung zu bestimmen. Im nächsten Abschnitt wird detaillierter auf einzelne Verfahren eingegangen.
- Codierung: Marker mit Codierung sind in den Kamerabildern eindeutig identifizierbar und erleichtern die Zuordnung während der Rekonstruktion (siehe Abschnitt 3.4.2). Möglichkeiten hierzu ergeben sich bei aktiven Markern durch sequentielle Aktivierung, bei farbigen Markern über verschiedene Farbwerte und bei komplexeren Markertypen über die Form bzw. innere Struktur (Abbildung 2.7d-f). Hilfreich ist auch bereits die Zuordnung der Marker zu definierten Teilmengen (Gruppencodierung).
- 2D/3D: Ebene flächenhafte Marker sind einfacher in Herstellung und Montage, leiden jedoch unter abnehmender Genauigkeit bei Blickwinkelabweichungen

von der Senkrechten. Räumlich ausgedehnte z.B. kugelförmige Marker sind robuster unter variablen Sichtwinkeln.

- Anzahl abgebildeter Freiheitsgrade: Die meisten Markertypen repräsentieren ungeachtet ihrer Ausdehnung oder inneren Struktur genau einen Raumpunkt im Trackingvolumen (Abbildung 2.7a-e) und liefern damit Informationen zu 3 Freiheitsgraden (3DOF). Bei Kombination zweier solcher Mmarker an einem Objekt können 5 Freiheitsgrade erfasst werden (Position und Orientierung außer der Rotation um die Verbindungsgerade der Marker - 5DOF), drei Punktmarker erlauben volle 6DOF-Rekonstruktion (Position und Orientierung). Spezielle AR-Marker[8] (Abbildung 2.7f) können die vollständige Pose des Objekts (Lage und Orientierung, 6DOF) erfassen, jedoch mit eingeschränkter Genauigkeit.

Es sind vielfältige Kombinationsmöglichkeiten der aufgeführten Merkmale für den Einsatz als Marker geeignet, wobei je nach Anwendungsbereich ein Markertyp mit geeigneten Eigenschaften auszuwählen ist. Dabei ist oftmals ein Abwägen widersprüchlicher Merkmale notwendig. Die meisten Literaturstellen gehen bevorzugt auf den Einsatz photogrammetrischer Zielmarken bei Standardanwendungen in der industriellen Messtechnik ein (beispielsweise [Ahn und Schultes 1997; Niederöst und Maas 1997; Falcon GmbH 2008]). Einige weitere Markertypen werden unter anderem von Wollnack [2005, Kap. 1.4] und Luhmann [2010, Kap. 3.6 und 5.4] beschrieben. Dem Autor ist jedoch keine umfassende vergleichende Übersicht der verschiedenen in optischen Trackingsystemen eingesetzten Markertypen und ihrer Eignung für spezielle Aufgabenstellungen bekannt. In Anhang C werden daher einige für die Einsatzgebiete des MOSCOT-Systems gebräuchliche Markertypen detaillierter dargestellt und eine Bewertung der individuellen Vor- und Nachteile vorgenommen. Dies kann als Entscheidungshilfe bei der Auswahl geeigneter Markertechniken für verschiedene Trackingszenarien herangezogen werden.

Markerdetektion mit Subpixelauflösung

Zur Steigerung der erreichbaren Auflösung und Genauigkeit existieren für alle beschriebenen Markertypen Mechanismen zur Bildmessung mit Subpixelgenauigkeit. Dabei wird die Position des ausgezeichneten Markerreferenzpunktes im Kamerabild mit einer Auflösung bestimmt, die Zwischenwerte zwischen benachbarten Pixeln annehmen kann (Abbildung 2.8). Wichtige Verfahren diesbezüglich sind [Shortis et al. 1994] [Wang et al. 2004] [Luhmann 2010, Kap. 5.2.4.7]:

- bei kreis- oder ellipsenförmigen Markern die Bestimmung des Schwerpunktes

[8] Erweiterte Realität (*augmented reality*, AR) ist eine virtuelle Realität, die über eine reale Kameraansicht eingeblendet wird.

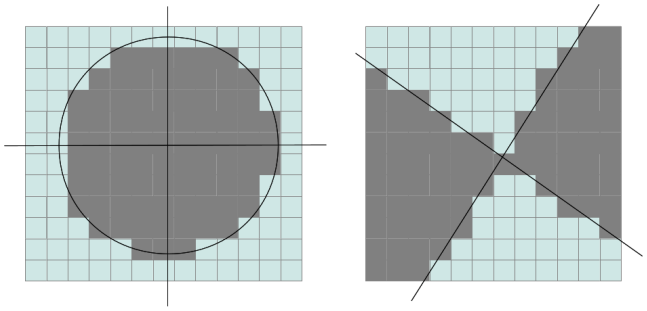


Abbildung 2.8: Bestimmung des Zentrums eines kreisförmigen und eines kantenbasierten Markers mit Subpixelauflösung.

- bei kreis- oder ellipsenförmigen Markern die Bestimmung des Zentrums über spezielle Ellipsenoperatoren oder Kleinste-Quadrate-Schätzung
- bei Markern mit linienartigen Strukturen über Kantenextraktion und Bestimmmung von Schnittpunkten
- bei Markern mit synthetischem Muster (Template) über ein Korrelationsverfahren (Template Matching)

Je nach Verfahren lassen sich so Subpixelgenauigkeiten im Bereich von 0,01–0,1 Pixeln erreichen [Luhmann 1995] [Schwarte et al. 1999, Kap. 7.3.3] [Luhmann 2010, Kap. 5.4.2].

Für das Erreichen der angeführten Subpixelgenauigkeit ist jeweils eine gewisse Mindestgröße der Markerabbildung im Kamerabild notwendig.[9] Das Schwerpunktverfahren bei kreis- bzw. ellipsenförmigen Markern ist dabei bereits bei kleineren Markerbildgrößen (5–10 Pixel Durchmesser) einsetzbar als die anderen Verfahren, welche jedoch teilweise eine höhere Subpixelauflösung liefern. Je nach Anwendung ist eine Abwägung geeigneter Verfahren der Subpixelinterpolation hinsichtlich Faktoren wie erreichbarer Auflösung, Raumbedarf des Markers, Ressourcenaufwand und Laufzeit[10] für die Berechnung etc. notwendig.

Ebene kreisförmige Marker erleiden im Unterschied zu kugelförmigen Markern zusätzlich eine Einschränkung der nutzbaren Auflösung bei Verkippung gegen die Ka-

[9] Die Verfahren der subpixelgenauen Positionsbestimmung basieren darauf, dass der bei der Abbildung auf das Sensorpixelfeld entstehende räumliche Quantisierungsfehler durch Approximationsrechnung über eine größere Umgebungsregion minimiert wird. Hierbei ist häufig eine leichte Defokussierung des Kamerabilds hilfreich, da so harte Kontrastübergänge auf einen größeren Pixelbereich ausgedehnt werden [Kawano et al. 2008].

[10] Dies gilt insbesondere für einige Verfahren, die keine direkte Lösung liefern, sondern iterative Optimierungsschritte ausführen.

merablickrichtung, da der perspektivische Effekt das Markerbild verkürzt. Räumlich ausgedehnte kugelförmige Marker sind darum bei variablen Sichtwinkeln zu bevorzugen.

Farbmarker

Auf den Einsatz von Farbmarkern wird hier im Besonderen eingegangen, da sie bei gängigen Trackinganwendungen selten eingesetzt werden. Bei den speziellen Gegebenheiten in medizinischen Simulatoren stellen sie jedoch durchaus eine sinnvolle Alternative dar. Aufgrund beengter Trackingvolumen und der parallelen Ausführung der Simulationsberechnung werden kleine, einfach zu fertigende und leicht zu fixierende Markertypen gesucht, die mit geringen Ressourcenaufwand aber trotzdem schnell und robust erfasst werden können. Der Hauptnachteil von Farbmarkern, die verringerte Erfassungsgenauigkeit gegenüber monochromatischen Markertypen (insbesondere unter wechselnden Beleuchtungsbedingungen, siehe Abschnitt 3.2.3), fällt hier durch den interaktiven Charakter der Anwendungen weniger ins Gewicht. Vielmehr ergeben sich Vorteile, da Farbmarker auch bei nur geringem Kontrastunterschied mit einfachen Verfahren vom Hintergrund zu segmentieren sind. Gleichzeitig transportiert die Farbinformation eine sehr einfache Codierung, die für die Unterscheidung einzelner Marker oder von Markergruppen (Gruppencodierung) hilfreich ist. Diese Lösung bietet sich hauptsächlich bei Simulatoren mit geschlossenem Trackingaufbau an, die eine gute Kontrolle der Beleuchtungsbedingungen ermöglichen.

Bewertung und Schlussfolgerungen für das Systemkonzept

Die Festlegung auf einen geeigneten Markertyp ist der wichtigste Einzelaspekt bei der Lösung einer optischen Trackingaufgabe. Sie beeinflusst auch direkt die einsetzbare Kamera- und Beleuchtungstechnik. Je breiter das Spektrum unterstützter Markertypen, desto universeller einsetzbar ist ein Trackingsystem. Für das MOSCOT-Projekt ist es daher wichtig, innerhalb der Systemarchitektur möglichst wenige Beschränkungen hinsichtlich der unterstützten Markertypen aufzuerlegen, gleichzeitig jedoch den Entwicklungs- und Adaptionsaufwand für verschiedenartige Markertypen gering zu halten.

Eine wesentliche Gemeinsamkeit der Mehrheit real eingesetzter Markertypen (vgl. Abbildung 2.7a-e und Anhang C) ist ihr quasi punktartiger Charakter mit 3 erfassten Freiheitsgraden. Die Datenverarbeitung dieser Markertypen unterscheiden sich lediglich bei der Bildmessung, nach der Positionsbestimmung des Markerreferenzpunktes in den Kamerabildern verlaufen die weiteren Rekonstruktionsschritte identisch.[11] Eine Unterstützung aller Arten von punktartigen Markern (3DOF-Markern) stellt

[11] Codierte Marker bieten zusätzliche Informationen für die Zuordnung der Markerbilder bei der Rekonstruktion, sie ändern jedoch nichts an der prinzipiellen Vorgehensweise (siehe Abschnitt 3.4.2).

damit einen vorteilhaften Ansatz für das MOSCOT-Baukastensystem dar. Es kann so prinzipiell eine Vielzahl verschiedener Markertypen unterstützt werden; gleichzeitig erfordert die Einbindung neuartiger 3DOF-Marker lediglich die Anpassung der Datenverarbeitungsverfahren für die Bildmessung.

2.2.5 Kameratechnik

Für die Markererfassung in optischen Trackingsystemen steht ein weites Spektrum an Kamerasensoren bereit. Sie unterscheiden sich in Eigenschaften wie Halbleitertechnologie, spektraler Empfindlichkeit, Verschlusstechnik und Anordnung, Anzahl und Größe der Bildelemente. Der Einfluss wichtiger Eigenschaften auf Einsatz und Ergebnis des optischen Trackings wird im Folgenden kurz angerissen. Ausführliche Übersichten zur Kamera- und Sensortechnik finden sich beispielsweise in [Seitz 2000], [Ohta 2008] und [Holst und Lomheim 2007].

Halbleitertechnologie

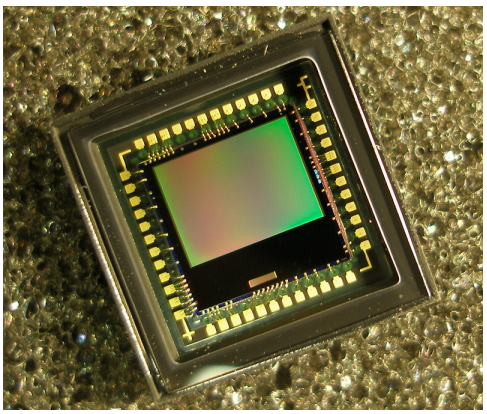


Abbildung 2.9: Ein Bildsensor in CMOS-Technik[12]

Die Bilderfassung in digitalen Kamerasensoren erfolgt über ein Feld von optoelektronischen Sensorelementen, die den einfallenden Photonenstrom über den Belichtungszeitraum sammeln und in elektrische Ladung umsetzen (photoelektrischer Effekt im Halbleiter). Während des Auslesezyklus wird die Ladung der einzelnen Zellen gemessen, digitalisiert und ausgegeben. Zwei unterschiedliche Technologien für den Aufbau solcher hochintegrierter Kamerasensoren haben sich etabliert: CCD (Charge-Coupled Devices) und CMOS (Complementary Metal Oxid Semiconductor). CCD-Sensoren

[12]Bild von Filya1/commons.wikimedia.org, Lizenz: CC-BY-SA-3.0

verfügen traditionell über eine bessere Bildqualität als CMOS, sind durch einen speziellen Fertigungsprozess allerdings auch teurer, aufwendiger in der Beschaltung und weniger flexibel bei der Ansteuerung. Die CMOS-Technologie gewinnt durch ihre bessere Integrierbarkeit mit herkömmlicher Logik und den resultierenden Kostenvorteil an Verbreitung. Durch die intensive Weiterentwicklung verringern sich darüber hinaus die Qualitätsnachteile (z.B. beim Pixelrauschen) gegenüber CCD und auch einige bauartbedingte Vorteile wie die geringe Neigung zum Blooming (Überstrahlen heller Bildbereiche) können für einen Einsatz in optischen Trackingsystemen relevant sein. Im Allgemeinen können hochwertige CMOS-Sensoren daher heute in vielen Anwendungen als qualitativ gleichwertiger Ersatz für CCD-Sensoren herangezogen werden. Abbildung 2.9 zeigt ein Beispiel für einen Bildsensor in CMOS-Technik.

Spektrale Empfindlichkeit

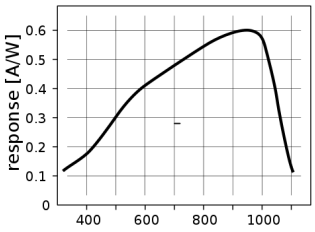
Die spektrale Empfindlichkeit der Photosensoren ist im Wesentlichen unabhängig von der eingesetzten Technologie (CCD vs. CMOS). Sie wird vielmehr durch das verwendete Halbleitermaterial bestimmt. Abbildung 2.10(a) zeigt ein typisches Empfindlichkeitsspektrum für Silizium. Die Integration der mit der Empfindlichkeit $\varphi(\lambda)$ gewichteten Intensität des Lichteinfalls $I(\lambda)$ über das gesamte Wellenlängenspektrum liefert die Quantenausbeute e bzw. den gemessenen Ausgangswert des Bildpixels.

$$e = \int \varphi(\lambda) I(\lambda) d\lambda \tag{2.1}$$

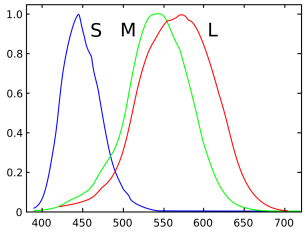
Die größte Empfindlichkeit φ_{max} wird im nahen Infrarotbereich erreicht. Dies hat direkte Auswirkungen auf die Art der Markersignalisierung. Bei Erkennung von Szenen die einem hohen Infrarotlichtanteil ausgesetzt sind, müssen Infrarot-Sperrfilter im optischen Strahlengang installiert werden, um eine natürliche Intensitäts- bzw. Farberfassung zu gewährleisten und ein Überstrahlen IR-heller Bereiche zu vermeiden. Bei Anwendungen mit IR-signalisierten Markern wiederum kann ein an die IR-Frequenz angepasster Bandpassfilter in den optischen Strahlengang eingebracht werden, der Störeinflüsse aus dem sichtbaren Spektralbereich wirksam unterdrückt.

Sensoren für Farbaufnahmen bilden die menschliche Farbwahrnehmung nach. Entsprechend der drei verschiedenen Ausprägungen von frequenzempfindlichen Sinneszellen (Zapfen) in der menschlichen Netzhaut, müssen auch Farbkameras Farbinformation als Linearkombination von drei verschiedenen Spektralverteilungen (z.B. rot, grün und blau für den RGB-Farbraum) erfassen [Mallot et al. 1998; Seitz 2000] (s. Abbildung 2.10(b) und 2.10(c)):

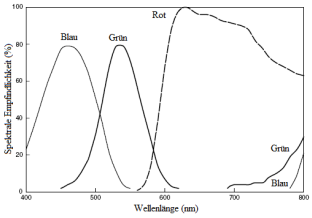
$$e_k = \int \varphi_k(\lambda) I(\lambda) d\lambda, k = 1, 2, 3 \tag{2.2}$$



(a) Siliziumphotoelement[13]



(b) Sinneszellen in der menschlichen Retina[14]



(c) Photoelemente in Farbkamerasensoren[15]

Abbildung 2.10: Spektrale Empfindlichkeiten von menschlichem Auge und digitalen Kamerasensoren

Hochwertige Kameras realisieren dies durch eine Aufspaltung des Strahlengangs an dichroitischen Spiegeln (Prismen) und eine Abbildung durch Spektralfilter auf drei separate Bildsensoren. Der hohe Aufwand und die erforderliche Präzision führen zu sehr hohen Preisen bei diesen Kameras. Die gängige Lösung für den Massenmarkt besteht darin, einzelne Sensorelemente durch vorgeschaltete spektrale Filter für unterschiedliche Wellenbereiche auszulegen. Hierdurch reduziert sich die Auflösung des Sensorfelds. Es existieren verschiedene Schemata für die Anordnung der Farbfilter auf dem Pixelsensorfeld. Das mit Abstand am häufigsten eingesetzte Filtermuster ist das Bayerpattern [Bayer 1976], ein sich wiederholendes 2x2 Farbmuster mit den Spektralkanälen R+G und G+B (Abbildung 2.11). Der Grünkanal als Frequenzbereich mit der höchsten Auflösung und Signifikanz für die menschliche Wahrnehmung ist doppelt ausgeführt. Korrekte Pixelfarbwerte werden nach der Digitalisierung in einer Filterstufe (Demosaicing-Filter, siehe Abschnitt 3.3.3) errechnet.

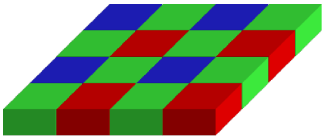


Abbildung 2.11: Anordnung der Farbfiltermatrix nach dem Schema von Bayer

Verschluss

Ähnlich wie bei herkömmlichen Analogkameras ist auch bei digitalen Sensoren ein Verschlussmechanismus zur Begrenzung der Belichtungszeit notwendig. Die meis-

[13] Diagramm von KaiMartin/commons.wikimedia.org, Lizenz: CC-BY-SA-3.0
[14] Diagramm von Vanessaezekowitz/commons.wikimedia.org, Lizenz: CC-BY-SA
[15] Diagramm von Herbertweidner/commons.wikimedia.org, Lizenz: frei

ten digitalen Sensoren implementieren diesen elektronisch, so dass ein mechanischer Verschluss entfallen kann. Kostengünstige Sensoren für den Massenmarkt in CMOS-Technik verfügen häufig über einen Rolling-Shutter-Mechanismus, bei dem die Sensorzellen zeilenweise zu unterschiedlichen Zeitpunkten belichtet werden. Dies führt beim Tracking bewegter Objekte zu verzerrter Abbildung und damit zu verringerter Messgenauigkeit. CCD-Sensoren verfügen bauartbedingt über einen globalen, auf das gesamte Sensorfeld wirkenden Verschluss. Auch aktuelle hochwertige CMOS-Sensoren verfügen mittlerweile über einen solchen Global-Shutter, der alle Sensorzellen gleichzeitig belichtet, und damit diese Problematik umgeht. Für Trackinganwendungen sind daher Sensoren mit globalem Verschlussmechanismus zu bevorzugen.[16]

Zeilenkameras vs. Flächenkameras

Neben den weit verbreiteten zweidimensionalen Sensorfeldern existieren auch eindimensionale Anordnungen von Photozellen, sogenannte Zeilensensoren. Durch die geringe Chipfläche sinkt die Wahrscheinlichkeit von Verunreinigungen und Fehlern und auch sehr hohe Auflösungen können gefertigt werden. Zusammen mit dem eingesetzten CCD-Fertigungsprozess entstehen so Sensoren für hochgenaue Messungen. Da nur geringe Datenmengen anfallen, können diese Sensoren bei sehr hohen Auslesefrequenzen betrieben werden. Aufgrund ihrer Eigenschaften werden Zeilensensoren bevorzugt für linear bewegte Aufnahmesystemen z.B. bei Bildscannern oder Beobachtungssatelliten herangezogen. In optischen Trackingsystemen mit aktiven Markern werden sie im Dreierverbund mit vorgebauten Zylinderlinsen eingesetzt, wobei jeder Sensor jeweils eine Raumwinkelkoordinate misst [Foxlin et al. 2002] (vgl. Sektionen 2.2.3 und 2.3.1). Bei der Abbildung des 3D-Weltkoordinatensystems auf das 1D-Kamerakoordinatensystem entstehen jedoch im Vergleich zu Flächenkameras vermehrt Markerüberdeckungen (alle Marker innerhalb einer Sichtebene werden auf denselben Sensorpunkt abgebildet). Es können somit nur aktive, sequentiell geschaltete Marker eingesetzt werden, jedoch keine Markertypen mit innerer Struktur, Codierung oder Farbe. Im Unterschied dazu bleiben bei der Abbildungsprojektion auf einen Flächensensor alle Arten von Markern identifizierbar.[17]

Bewertung und Schlussfolgerungen für das Systemkonzept

Ähnlich wie die Markierungstechnik ist auch die einzusetzende Kameratechnik eines optischen Trackingsystems stark von der jeweiligen Anwendung abhängig. Im

[16] Ein ähnliches Verhalten tritt bei älteren analogen Videosensoren in Interlace-Technik auf, bei denen zeitversetzt zwei zeilenweise ineinander verschränkte Halbbilder erfasst werden. Auch diese Bildsensoren sind daher für Trackinganwendungen weniger zu vermeiden.

[17] Flächensensoren werden daher in der englischsprachigen Fachliteratur auch als Imaging Sensor bezeichnet, Zeilensensoren hingegen den Non-imaging Sensoren zugeordnet.

Umkehrschluss gilt für ein universelles Trackingsystem, dass die unterstützte Kamerasensortechnik das Spektrum möglicher Anwendungsbereiche nicht einschränken sollte. Zeilenkameras erlauben mit ihren aktiven, sequentiell geschalteten Markern nur ein sehr eingeschränktes Anwendungsgebiet – sie erfordern aufgrund der grundlegend anderen Abbildungsgeometrien jedoch spezialisierte Markerdetektions- und -rekonstruktionsverfahren mit eigenen Datenverarbeitungsstrukturen. Ihre Stärken – sehr hohe Genauigkeit bzw. Auflösung – sind für die primären Einsatzbereiche des MOSCOT-Systems weniger relevant. Bei der Auslegung des MOSCOT-Systemarchitektur kann daher dem Einsatz von Flächensensoren der Vorzug gegeben werden, ohne dass dadurch die Einsatzbandbreite des Systems wesentlich beschränkt wird. Weitere Aspekte der eingesetzten Sensortechnik, so etwa Unterschiede zwischen CMOS- und CCD-Technik oder die Auslegung für Infrarot-, Monochrom- und Farbtracking betreffen nur die unmittelbare Ansteuerung der Bilderfassungshardware, die optischen Komponenten sowie die Bildvorverarbeitung. Die Systemarchitektur selbst bleibt hiervon weitestgehend unabhängig, so dass je nach Einsatzbereich alle Arten von Flächensensoren unterstützt werden können.

2.2.6 Aspekte der räumlichen Abtastung

Die wesentlichen Parameter im Zusammenhang mit dem räumlichen Abtastprozess eines optischen Trackingsystems sind Auflösung, Genauigkeit und Präzision.[18] Genauigkeit bezeichnet die Größe des Messfehlers zwischen dem gemessenem Wert und dem tatsächlichen Referenzwert. Auflösung bezeichnet die kleinstmögliche Positionsänderung, die mit dem System erfasst werden kann. Präzision schließlich ist die statistische Streuung um das Mittel der Messwerte. Sie wird gewöhnlich als Standardabweichung der Positionsmessung angegeben. Die Abweichung der Messwerte wird als Jitter bezeichnet. Abbildung 2.12 illustriert den Sachverhalt. Die gemessenen Werte streuen im Bereich ihrer Präzision, haben jedoch eine möglicherweise größere Abweichung vom tatsächlichen Wert [Luhmann 2010, Kap. 7.2].

Wichtige Einflussfaktoren der Systemgenauigkeit

Die beschriebenen Größen hängen von einer einer Vielzahl die Messung begleitender Faktoren ab und können bei realen Systemen gewöhnlich nicht in geschlossener Form angegeben werden. Weiterhin sind die Größen maßstabs- und konfigurationsabhängig und innerhalb des Messvolumens auch nicht konstant. Für eine detaillierte Darstellung sei auf die Literatur verwiesen (siehe z.B. [Hennes 2007; Bauer et al. 2006;

[18] Die Begriffe Präzision (*precision*) und Genauigkeit (*accuracy*) werden jedoch häufig nicht voneinander getrennt und synonym verwendet. Anstelle von Präzision wird dann von Wiederholgenauigkeit gesprochen.

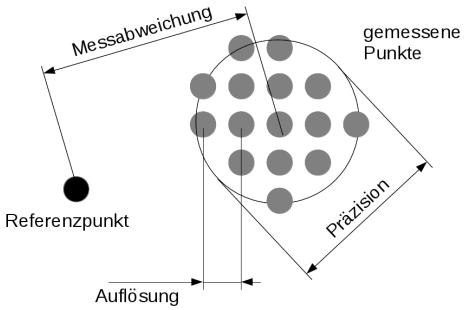


Abbildung 2.12: Veranschaulichung der Zusammenhänge von Auflösung, Präzision, und Genauigkeit als Messabweichung vom Referenzwert (nach Hennes [2007]).

Schlegel 2006; Dold 1997; Wiles et al. 2004]). Für den Systementwurf von MOSCOT soll eine überschlägige Genauigkeitsabschätzung nach Luhmann [2010, Kap. 7.1.2] verwendet werden. Demnach wird die Systemgenauigkeit im wesentlichen von folgenden drei Größen beeinflusst:

- Genauigkeit der Bildmessung s_x
- Abbildungsmaßstab m_b
- Aufnahmekonfiguration q

Die **Genauigkeit der Bildmessung** s_x hängt im wesentlichen ab von der Güte der verwendeten Kameras, sowie ihrer Kalibrierung und der Messgenauigkeit der verwendeten Marker. Je nach verwendetem Kamerasensor und mechanischer Stabilität verfügen Digitalkameras wieder nach der Abschätzung von Luhmann [2010, Kap. 7.1.2] über ein Genauigkeitspotential von 0,2–1 μm (entsprechend 1/50 bis 1/10 Pixel). Geeignete Operatoren zur Subpixelmessung der gefundenen Markerpositionen verbessern dies auf Genauigkeiten zwischen 1/100 und 5/100 Pixel je nach verwendetem Markertyp. Negative Faktoren stellen schiefe Betrachtungswinkel bei ebenen Markern sowie die mit der Größe zunehmende Exzentrizität bei kreisförmigen Markern dar.

Der **Abbildungsmaßstab** m_b beschreibt das Größenverhältnis zwischen Weltkoordinaten und Bildkkoordinaten. Mit den Größen aus Anhang A.3 ist dies gleichbedeutend mit dem Verhältnis von Aufnahmeentfernung zu Brennweite:

$$m_b = \frac{X}{x} = \frac{Z}{f} \tag{2.3}$$

Damit hängt der Abbildungsmaßstab direkt von der Aufnahmekonfiguration ab, also von Überlegungen zu abzudeckendem Messvolumen, Bildformat des Kamerasensors,

Öffnungswinkel des Objektivs etc. Mit dem Abbildungsmaßstab überträgt sich die Messgenauigkeit der Bildmessung in das Trackingvolumen $s_X = m_b s_x$. Übliche Größen bewegen sich im Bereich von ca. 10–1000

Die **Aufnahmekonfiguration** berücksicht die Schnittwinkel der Bildstrahlen, da die Genauigkeit in Kamerablickrichtung s_Z deutlich geringer ist als parallel zur Bildebene $s_{X,Y}$. Bei einem Rundumverband, d.h. einer gleichmäßigen Anordnung vieler Kameras rund um das Trackingvolumen verringert sich dieser Effekt. Der Designfaktor q ist ein Maß für diesen Einfluß der Aufnahmekonfiguration. Er kann bei einem Stereosystem näherungsweise als das Verhältnis der erreichbaren Genauigkeit in Z-Richtung (Kamerablickrichtung) zur Genauigkeit in X,Y-Richtung angesehen werden. Übliche Werte reichen von unter 1 (0.7 für sehr gute Rundumverbände) bis hin zu 1.5 und deutlich schlechter (bis zu 3 oder 4 für schlechte Stereokonfiguration). Als Abschätzung der Messgenauigkeit ergibt sich damit $s_Z = q s_{X,Y}$.

Weitere Faktoren

Weitere zusätzliche Faktoren beeinflussen die erreichbare Genauigkeit. Hierzu zählen die Qualität der Kalibrierung, Veränderung der Abbildungsparameter durch Temperatureffekte (Aufwärmvorgang) oder Alterung, Registrierungsfehler, inhomogene bzw. variable Beleuchtung bei passiven Markern etc. Insbesondere die Präzision wird stark vom Rauschverhalten der Bilderfassung beeinflusst. Das Schwanken der Messwerte wird als Jitter bezeichnet, kann aber durch Filterung der Messwerte (siehe Abschnitt 3.6.1) deutlich eingeschränkt werden.

Angaben zur Genauigkeit optischer Trackingsysteme beziehen sich gewöhnlich auf die anhand der 3DOF-Markerpositionen direkt gemessene Translation. Die Bestimmung der Rotation erfolgt über die Raumpositionen mehrerer Marker und ist in ihrer Genauigkeit demnach neben der Translationsmessung auch von der Geometrie der Markeranordnung abhängig. Prinzipiell kann die Rotationsauflösung erhöht werden, indem die Intermarkerdistanz vergrößert wird (vgl. hierzu auch die Anmerkungen zu Winkel- und Positionsauflösung bei Inside-Out und Outside-In-Systemen in Abschnitt 2.2.3).

Bewertung und Schlussfolgerungen für das Systemkonzept

Anforderungen hinsichtlich der Genauigkeit eines Trackingsetups hängen stark von der jeweiligen Anwendung ab. Für ein System im medizinischen Umfeld kann als Vorgabe die Erfassung manueller Tätigkeiten eines Chirurgen dienen, was zu Anforderungen im Bereich einiger Zehntelmillimeter führt (vgl. die Angaben zu Genauigkeitsanforderungen in [Wagner 2003], [Pott 2007]). Mit den oben angegebenen Genauigkeitsabschätzungen ist dieser Wert bei Trackingvolumina im Bereich von 0,1–1m^3 und

den gängigen Pixelauflösungen optischer Sensoren erreichbar. Für die primären Einsatzbereiche in VR-Anwendungen wie medizinischen Simulatoren sind Stetigkeit der Bewegungsabbildung und Auflösung der Positionserfassung wichtiger als absolute Genauigkeit [Foxlin et al. 2002; Pintaric und Kaufmann 2007], da ja eine korrigierende Rückkopplung über die Benutzerinteraktion selbst erfolgt.[19] Bei Augmented-Reality-Anwendungen mit Head-Mounted-Displays (HMDs) oder Trackingszenarien wie z.B. navigierten Eingriffen ist auch hohe absolute Genauigkeit erforderlich.

Generell sind die bestimmenden Grundlagen der Genauigkeit eines optischen Trackingsystems (Genauigkeit der Bildmessung, Modellierung der Kameraabbildung, mechanische und thermische Stabilität, Anordnung der Kameras) weitgehend unabhängig von Design und Architektur des Gesamtsystems. Sie können vielmehr durch Optimierung einzelner Komponenten und Verfahren wie Kamerasensoren, Objektiven, mechanischem Aufbau, Kalibrierung, etc. beeinflusst werden, wobei viele dieser Faktoren in einem direkten Zusammenhang mit der Qualität der Komponenten und damit dem finanziellen Aufwand stehen. Eine an die Aufgabenstellung angepasste Genauigkeit wird damit am besten von einer flexiblen Auslegung der Systemarchitektur unterstützt, die Auswahl und Tausch einzelner Komponenten und Verfahren ermöglicht. Überlegungen zu Genauigkeitsfragen des MOSCOT-Systems stellen daher im weiteren Verlauf dieser Arbeit keinen Schwerpunkt dar, auf einige wichtige Faktoren wie die Qualität der Kalibrierung und die Kompensation von Temperatureffekten geht eine weitere Dissertation in der Arbeitsgruppe [Handel 2009] ein.

2.2.7 Aspekte der zeitlichen Abtastung

Die zentralen Parameter der zeitlichen Abtastung, die ein Trackingsystem charakterisieren, sind Wiederholrate (engl. *update rate* oder *frame rate*) und Systemlatenz [Foxlin et al. 2002, Kap. 8.2].

Wiederholrate

Die Wiederholrate bzw. -frequenz beschreibt die Aktualisierungsrate, mit der das Trackingsystem neue Daten bereitstellt. Sie entspricht gewöhnlich der Bildwiederholrate der verwendeten Kameras. Die Datenverarbeitung des Systems muss dabei so ausgelegt sein, dass sie die von Kameraseite gelieferte Datenrate verarbeiten kann.

$$f_{System} = f_{Camera} \tag{2.4}$$

[19] Anschaulich wird dies am Beispiel einer Computermaus, bei der die absolute Genauigkeit auch eine untergeordnete Rolle spielt.

Systemlatenz

Die Latenzzeit des Systems beschreibt die zeitliche Verzögerung zwischen dem Zeitpunkt der Bildakquisition und der Bereitstellung der rekonstruierten Daten für die Anwendung. Sie ist die Summe der Datenverarbeitungs- und -übertragungszeiten:

$$T_{System} = T_{Exposure} + T_{CamReadout} + T_{ImageProc} + T_{Reconstruction} + T_{DataTransfer} \quad (2.5)$$

Die Latenzzeiten treten sequentiell innerhalb der Verarbeitungskette auf und können nicht durch parallele Verarbeitung versteckt werden.[20] Die Kameraauslesezeit $T_{CamReadout}$ entspricht meist der reziproken Bildaufnahmerate der Kameras $\frac{1}{f_{Camera}}$. Die Gesamtlatenz des Systems T_{System} kann somit deutlich über $\frac{1}{f_{Camera}}$ liegen. Für viele interaktive Anwendungen ist die Latenz damit die bedeutsamere der beiden Größen. Anders als bei der Genauigkeit des Systems sind für ein optimales zeitliches Verhalten nicht nur einzelne Komponenten, sondern die ausgewogene Auslegung der gesamten Verarbeitungsarchitektur verantwortlich.

Synchronisationslatenz

Abbildung 2.13 illustriert ein Problem, welches bei der Anbindung des Trackingsystems an die Anwendung, z.B. einen VR-Simulator, entsteht. Unterschiedliche Frequenzen bzw. Latenzzeiten von Trackingsystem und Datenverarbeitung der Applikation führen zu einer variablen Verzögerung zwischen Erfassungszeitpunkt des Trackingsystems und der Ausgabe der Applikation. Die Folge sind Probleme wie Artefakte in der graphischen Darstellung eines VR-Simulators oder Instabilitäten eines nachgeschalteten Regelkreises.[21] Bei einer Trackingfrequenz f_T und einer Datenverarbeitungsfrequenz f_{DV} der Anwendung ergibt sich eine variable Gesamtlatenz von

$$T_{sync,min} = \frac{1}{f_T} + \frac{1}{f_{DV}} \quad (2.6)$$

bis

$$T_{sync,max} = 2\frac{1}{f_T} + \frac{1}{f_{DV}}, \quad (2.7)$$

je nachdem, ob der Beginn des Datenverarbeitungsprozesses direkt mit dem Ende des Trackingprozesses zusammenfällt, oder knapp vor diesem liegt, so dass auf das Ergebnis des vorigen Trackingzyklusses zurückgegriffen werden muss (zusätzliche Synchronisationslatenz von $\frac{1}{f_T}$).

[20] Etwaige zusätzliche Synchronisationslatenzen (siehe nächster Abschnitt) sind durch eine vollständig synchrone, sequentielle Auslegung des Trackingdatenflusses vermeidbar.

[21] Der Sachverhalt ist hier vereinfacht wiedergegeben, da auch asynchrone parallele Prozesse innerhalb der Anwendung selbst noch zusätzliche Synchronisationslatenzen induzieren können.

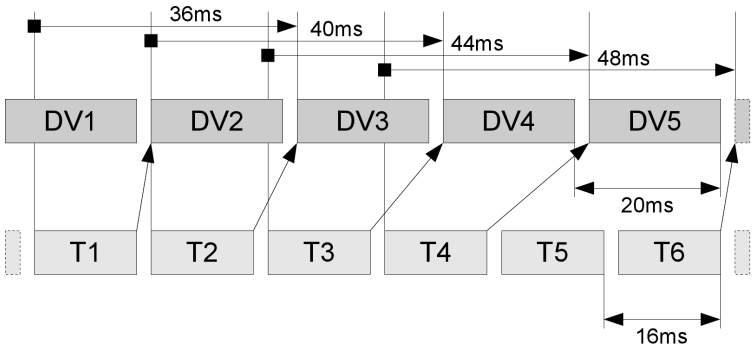


Abbildung 2.13: Variabilität der Synchronisationslatenz von Trackingsystem und Anwendung. Die Verarbeitungszyklen von Tracking (unten) und Anwendung (oben) laufen mit unterschiedlichen Frequenzen (Verarbeitungslatenzen 16ms bzw. 20ms). Der Datenverarbeitungsprozess DVn verwendet jeweils das aktuellste Ergebnis des Trackingprozesses Tn, welches zu seinem Start verfügbar ist. Die Gesamtlatenz schwankt in diesem Beispiel zwischen 36 und 48ms, außerdem wird die Trackinginformation T5 nicht genutzt.

Verschiedene Ansätze zur Verminderung solcher Probleme sind realisierbar, so z.B. eine Synchronisation der Anwendung auf die Bildakquisitionsrate der Trackingkameras oder das in der Studiotechnik verwendete Gen-Locking, d.h. die Synchronisation aller Module auf die Masterwiederholrate der Ausgabegeräte [Foxlin et al. 2002, Kap. 8.6]. Die Synchronisationsmethode ist der jeweiligen Aufgabenstellung bzw. Anwendung anzupassen. Eine ausführliche Diskussion zur Synchronisation innerhalb der VRM-Architektur findet sich bei Wagner [2003, Kap. 2.5].

Dynamischer Fehler

Ein weiteres Problem der zeitlichen Abtastung ist das Auftreten eines dynamischen Fehlers in der Positionserfassung bei Relativbewegungen zwischen Objekt und Aufnahmesystem. Zum einen ist dies Bewegungsunschärfe, d.h. ein Verlust an Genauigkeit bei der räumlichen Abtastung, der im Kamerabild durch die Integration über die Belichtungszeit entstehen. Dem kann durch Wahl einer geeignet kurzen Belichtungszeit (und damit einer passenden Kombination aus schnellem Kamerasensor und lichtstarkem Objektiv) begegnet werden.

Ein weitaus größeres Problem ist der dynamische Rekonstruktionsfehler, der bei nicht synchronen Erfassungszeitpunkten der Kameras entstehen kann. Zwischen den beiden Erfassungszeitpunkten $t, t + \Delta t$ legt das Objekt eine Wegstrecke $\Delta S = v\Delta t$ zurück. Der verspätete Aufnahmezeitpunkt führt direkt zu einem Lagefehler des Markerbildes im zweiten Kamerabild, der bei einer Bewegung quer zur Aufnahme-

richtung maximal wird. Durch die projektive Rückabbildung während der Rekonstruktion kann dieser Parallaxenfehler in Beobachtungsrichtung ein Mehrfaches der ursprünglichen Wegstrecke ΔS betragen. Der durch nicht synchronisierte Kameras entstehende Rekonstruktionsfehler kann damit auch leicht auf ein Vielfaches der eigentlichen Messgenauigkeit des Systems anwachsen [Luhmann 2010, Kap. 6.6.1.2]. Eine Synchronisation der Erfassungszeitpunkte aller Kameras eines Trackingsystems ist daher unerlässlich für die stabile Erfassung bewegter Objekte.

Echtzeitfähigkeit

Wagner [2003, Kap. 1.3.3] fordert für ein medizinisches VR-System als ganzes und damit auch für die optische Trackingkomponente eine weiche Echtzeitfähigkeit des Systems. Weiche Echtzeitfähigkeit wird hier als die Eigenschaft definiert, dass das System auf ein Ereignis *typischerweise* innerhalb eines vordefinierten Zeitrahmens reagiert, so dass die Immersion in die simulierte Szene gewahrt bleibt. Wagner [2003, Kap. 2.5] zeigt weiter, dass diese Forderung unter gewissen, in VR-Simulatoren vorliegenden Bedingungen auch von den Standardbetriebssystemen Windows und Linux eingehalten wird. Für noch anspruchsvollere Aufgabenstellungen (z.B. die Bewegungserfassung eines Operationsroboters) reicht dies jedoch nicht aus und es muss harte Echtzeitfähigkeit gewährleistet werden, d.h. die garantierte Verarbeitung der Daten innerhalb eines vorgegebenen Zeitraums. Datenverarbeitungsmodule, die als Software in einem PC realisiert sind, müssen durch Umsetzung auf ein geeignetes Echtzeitbetriebssystem (z.B. RealTime-Linux [Dietrich und Walker 2005; Ripoll 2002]) an solche Anforderungen angepasst werden.

Bewertung und Schlussfolgerungen für das Systemkonzept

Systemlatenzen und Wiederholraten werden durch das konkrete Trackingszenario vorgegeben, sie können für die primären Einsatzbereiche von MOSCOT-Systemen jedoch bereits grob vorab charakterisiert werden. Bei VR-Anwendungen wie medizinischen Simulatoren sollte das System eine Gesamtlatenz von ca. 50ms nicht überschreiten, um den Eindruck von Präsenz und Immersion der virtuellen Umgebung aufrechtzuerhalten [Wagner 2003]. Da sich die Latenzzeiten für Tracking, Kommunikation, Simulation und Rendern der Grafik aufaddieren, ist z.B. im Falle des Simulators EYESI die für das Tracking verfügbare Latenzzeit auf ca. 20ms beschränkt [Wagner 2003, Kap. 7.4.2]. Bei Augmented Reality Systemen mit Head-Mounted-Displays (HMDs) sind noch geringere Latenzzeiten einzuhalten. Durch die Überlagerung von instantaner Wahrnehmung des Gleichgewichtssinns und verzögertem visuellen Kanal stellen sich sonst Effekte wie Drehschwindel und Übelkeit ein [Allison et al. 2001]. Held und Durlach [1993] geben für HMD eine maximal tolerierbare Verzögerung von 30ms an, so dass bei einer angenommenen gleichmäßigen Aufteilung der zur Verfügung stehenden Latenzzeit zwischen Tracking und AR-Anwendung

eine Trackinglatenz von max. 15ms errreicht werden muss. Die Wiederholfrequenzen eines optischen Trackings bei VR- bzw. AR-Anwendungen sollten im Bereich der Wiederholrate der Visualisierung liegen, d.h. je nach Wiedergabegerät ca. 30–60Hz betragen.

Allerdings stellen diese Werte nur Anhaltspunkte dar – wichtiger ist bei interaktiven VR- und AR-Anwendungen immer die Aufrechterhaltung der Immersion in die künstliche Umgebung. Je nach Anwendungsfall sind daher auch längere Latenz- und Wiederholzeiten tolerierbar, beispielsweise bei langsamen Bewegungen, solange für den Nutzer der Realitätseindruck der Simulation aufrecht erhalten bleibt (vgl. hierzu wieder die Anmerkungen in [Wagner 2003, Kap. 7.4.2]).

Die Systemarchitektur spielt eine essentielle Rolle für die Erreichbarkeit niedriger Latenzen und hoher Updateraten. Sie muss flexibel genug sein, den Einsatz unterschiedlicher Kamera- und Markersignalisierungstechnik zu unterstützen, gleichzeitig aber auch schlank genug, um die Verarbeitungs- und Übertragungszeiten so gering wie möglich zu halten. Diese beiden Forderungen stehen häufig im Widerspruch, so dass ihnen bei der Architekturkonzeption von MOSCOT besondere Beachtung zukommen muss. Als Systemplattform sind sowohl Standardbetriebssystemen wie Windows und Linux für den Einsatz in medizinischen Simulatoren zu unterstützen, für echtzeitkritische Trackingapplikationen ist jedoch bereits eine Portierung auf Echtzeit-Plattformen vorzusehen.

Eine universell ausgelegte Trackingsystemarchitektur sollte weiterhin keine spezielle Synchronisationsmethode voraussetzen, sondern vielmehr an entsprechende Vorgaben der Anwendung anpassbar sein. Folglich ist es sinnvoll, die Trackingarchitektur als in sich vollständig synchrones und sequentielles System auszulegen, das für die Applikationsanbindung sowohl blockierenden als auch nicht blockierenden Betrieb unterstützt sowie auf die Unterstützung extern wie intern generierter Synchronisationssignale vorbereitet ist. In jedem Fall notwendig ist eine hardwareseitige Synchronisation der Kamerasensoren, um den dynamischen Fehler bei schnell bewegten Objekten zu minimieren. Mit dieser Auslegung des Synchronisationskonzept kann das System ein breites Spektrum von Anwendungsfällen unterstützen.

2.3 Komplettsysteme für optisches Tracking

Kommerziell erhältliche Trackingsysteme werden in der Regel als vorkalibrierte monolithische Einheiten bereitgestellt, die für ein eng spezifiziertes Anwendungsszenario ausgelegt sind. Die weitaus meisten Systeme bestehen aus einem Kamerarack, das je nach verwendeter Abbildungs- und Rekonstruktionsgeometrie zwei Flächen-

oder drei Zeilenkameras integriert. Die Systeme arbeiten üblicherweise im Infrarotbereich mit aktiven LED-Markern oder passiven retroreflektierenden Markern mit IR-Beleuchtung an den Kameras. Sie sind ausgelegt für starre Objekte im Größenbereich von einige Zentimetern bis Dezimetern und ein Trackingvolumen von einem halben bis einigen Kubikmetern. Genauigkeiten reichen bis in den Submillimeterbereich und Geschwindigkeiten bis zu einigen 100 Hz werden erreicht.

Es existiert eine Vielzahl an kommerziell verfügbaren Systemen zur Lösung optischer Trackingaufgaben. Eine Übersicht über einige verbreitete Systeme geben beispielsweise Ribo [2001] und Buaes [2005]. Im Folgenden wird im wesentlichen auf Systeme eingegangen, die in der Medizintechnik bzw. bei VR/AR-Anwendungen eingesetzt werden, und damit den vorgesehenen Einsatzbereichen des MOSCOT-Systems ähneln. Systeme mit einem Inside-Out-Ansatz (z.B. für mobile Robotik) oder für sehr große Trackingvolumen und Objektanzahlen (z.B. für Motion-Capture-Aufnahmen in der Filmindustrie) werden nicht berücksichtigt. Je nach Anzahl und Art der zu einem System zusammengefassten Kameras können die Systeme in Klassen mit jeweils ähnlichem Systemaufbau eingeteilt werden. Der Aufbau dieser Systemklassen wird jeweils kurz in Grundzügen beschrieben. Detailliertere Beschreibungen zu einigen ausgewählten Systemen aus dem medizintechnischen Umfeld sind im Anhang B.1 zusammengefasst.

2.3.1 Trinokulare Systeme mit Zeilensensoren

Systeme diesen Typs verfügen alle über einen gleichartigen Systemaufbau mit drei CCD-Infrarot-Zeilensensoren, die in linearer Anordnung in einem gemeinsamen Gehäuse montiert sind. Die beiden Sensoren an den Enden des Gehäuses erfassen dabei die Azimuthwinkel und damit die horizontale Lage des Markers in der Gehäuseebene (waagrechte Ausrichtung des Zeilensensors), der dritte Sensor in der Gehäusemitte erfasst den Deklinationswinkel und damit die Höhe des Markers über der Gehäuseebene (senkrechte Ausrichtung des Zeilensensors). Die Rekonstruktion erfolgt nach dem in Abschnitt 2.2.3 beschriebenen Ebenenschnittverfahren. Durch die definierte Ausrichtung der Zeilensensoren zueinander ist Redundanz mittels zusätzlicher Kameras nur eingeschränkt möglich und muss gewöhnlich durch Hinzunahme weiterer kompletter Systeme realisiert werden. Zur Objektsignalisierung kommen aktive Marker in Form von IR-LEDs zum Einsatz. Um das Problem der Überlagerung verschiedener Markerbilder bei der Zylinderlinsenprojektion zu umgehen, werden die LED-Marker in den Einzelaufnahmen sequentiell fortgeschaltet. Die erforderliche Marker-Kamerasynchronisation erfordert damit eine Kabelanbindung der Marker oder eine komplexe Funksynchronisation. Weiterhin reduziert sich durch die sequentielle Erfassung mit steigender Markerzahl die prinzipiell recht hohe Geschwindigkeit dieser Systeme und die Genauigkeit der Positionsbestimmung bei bewegten Objekten nimmt

ab. Die erhältlichen Systeme verfolgen alle ein identisches Systemkonzept und unterscheiden sich im wesentlichen nur durch die Qualität der eingesetzten Zeilensensoren und die Güte der Systemkalibrierung. Dies beeinflusst direkt die erreichbare Genauigkeit, Geschwindigkeit und die Größe des erfassten Trackingvolumen, und damit indirekt auch Baugröße und Preis der Systeme.

2.3.2 Binokulare Systeme mit Flächensensoren

Solche klassischen Stereobildsysteme arbeiten mit zwei in einem gemeinsamen Gehäuse integrierten Flächenkameras in CCD- oder CMOS-Technik. Die erfasste Markerposition wird hier mit dem Geradenschnittverfahren aus Abschnitt 2.2.3 rekonstruiert. Durch die parallele Erfassung aller Marker ist die Geschwindigkeit (und auch die Genauigkeit) dieser Systeme unabhängig von der Markeranzahl. Meist arbeiten die Systeme mit Kamerasensoren im Infrarotbereich. Sowohl aktive (IR-LEDs) als auch passive Marker (retroreflektierende Marker in Verbindung mit IR-Ringillumination an den Kameras) werden eingesetzt. Auch eine Kombination von passiven und aktiven Markern ist so realisierbar. Die Infrarot-Systeme verfügen alle über einen gleichartigen Systemaufbau. Sie unterscheiden sich ähnlich wie die Zeilenkamerasysteme im wesentlichen anhand der eingesetzten Bildsensoren und damit in den Kriterien Genauigkeit, Geschwindigkeit und erfassbarem Trackingvolumen. Eine andere Variante verwendet Markersignalisierung im sichtbaren Spektralbereich des Lichts. Diese Systeme können oftmals bereits mit der vorhandenen Umgebungsbeleuchtung arbeiten und setzen zur robusten Detektion passive Markertypen mit komplexerer innerer Struktur ein. Sie sind weniger weit verbreitet, es sind hier sehr unterschiedliche Bauformen und Systemausführungen möglich.

2.3.3 Modulare Systeme aus separaten Flächensensormodulen

Diese Systeme arbeiten mit einer variablen Anzahl an separaten Kameramodulen. In den Modulen werden Flächensensoren in CCD- oder CMOS-Technik verwendet, da Zeilenkameras aufgrund ihrer Rekonstruktionsgeometrie größeren Einschränkungen bei der räumlichen Ausrichtung unterliegen. Wie bei den Systemen des vorherigen Abschnitts wird ein Marker über den Schnitt von mindestens zwei Kamerastrahlen rekonstruiert. Die Bildakquisition mit mehr als zwei Kameras bietet eine Redundanz gegenüber Verschattung einzelner Kameras bzw. verringert den Rekonstruktionsfehler durch Mittelung über alle Kameras. Das modulare Konzept erlaubt eine flexible Positionierung der Kameras, erfordert aber auch einen erhöhten Aufwand bei Systemaufbau und -kalibrierung. Ähnlich wie bei den binokularen Systemen können die Systeme für alle Markierungs- und Signalisierungstechniken ausgelegt werden.

Für die Fusion der Kameramoduldaten ist hier ein separater Rechner erforderlich, welcher jedoch häufig mit dem eigentlichen Anwendungsrechner kombiniert werden kann. Ein Problem modularer Systeme ist allerdings der mit der Kamerazahl anwachsende Datenstrom, der hohe Übertragungsbandbreiten und Rechenkapazitäten im zentralen Host-PC erfordert. Ein Lösungsansatz sind sogenannte "intelligente" Kameras (engl. smart cameras), die durch integrierte Bildverarbeitungshardware eine Datenreduktion erzielen und damit den Hostrechner entlasten.

Aufgrund des erhöhten Aufwands für Systemaufbau und -einsatz sind diese Systeme in der Medizintechnik weniger weit verbreitet als komplett integrierte Systeme. Sie verfügen jedoch über eine deutlich größere Flexibilität bei der Anpassung an neuartige Aufgabenstellungen jenseits der Standardanwendungen.

2.4 Softwarebibliotheken für optisches Tracking

Das Spektrum an Softwarelösungen, die sich zumindest in Teilaspekten mit Problemstellungen optischen Trackings befassen, ist deutlich heterogener als bei den Komplettsystemen. Es reicht von hardwarenahen Treibern, über Bildverarbeitungsbibliotheken, abstrahierende Datenflussnetze und Module für die Anbindung an ein VR-Framework bis hin zu Speziallösungen für AR-Applikationen.

Eine große Gruppe stellt die Systemsoftware dar, die mit den Trackingsystemen zusammen ausgeliefert wird. Sie umfasst die Softwareschnittstelle zur Anbindung des Systems an die eigentliche Anwendung. Herstellerspezifische, hardwarenahe Treiber ermöglichen die Kontrolle des Systembetriebs und die Weitergabe der rekonstruierten Trackingdaten. Weiterhin werden mit der Systemsoftware üblicherweise Hilfsanwendungen für grundlegende Funktionen wie die (Nach-)Kalibrierung des Systems oder die Registrierung von Objekten bereitgestellt. Die Systemsoftware ist speziell an das jeweilige System angepasst und kann nicht mit anderen Systemen verwendet werden.

Auf einer höheren Abstraktionsebene arbeiten geräteunabhängige Softwarebibliotheken, die als Middleware zwischen gerätenaher Systemsoftware und der eigentlichen Anwendung vermitteln. Sie bilden eine Abstraktionsschicht über verschiedene Trackingsysteme und stellen generische Datenstrukturen zur Weiterverarbeitung auf Objektebene und vielfach auch Netzwerktransparenz bereit. Einige dieser Ansätze verfügen über direkte Anbindung an VR-Softwareframeworks (wie z.B. VR-Juggler [Bierbaum et al. 2001]). Solche Middleware-Software existiert neben kommerziellen proprietären Varianten auch in Form von freien Softwareprojekten mit offengelegtem Sourcecode (z.B. die VRPN-Bibliothek [Russell M. Taylor et al. 2001]). Den flexibelsten Ansatz bietet hier das OpenTracker-Projekt [Reitmayr und Schmalstieg 2005]. Es ermöglicht über ein Datenflussnetzwerk die Verteilung, Kombination und

Weiterverarbeitung von Trackingdaten unterschiedlicher Quellen, so dass Datenfluss und Datenverarbeitung recht einfach an neue Aufgabenstellungen angepasst werden können. Allen diesen geräteunabhängigen Lösungen ist gemein, dass sie auf einer Abstraktionsebene über den eigentlichen Trackingsystemen ansetzen und mit bereits rekonstruierten Marker- und Objektdaten arbeiten.

Eine größere Flexibilität bei der Anpassung an unkonventionelle Trackingsetups bieten Bibliotheken, die bereits auf tieferer Ebene der Datenverarbeitung ansetzen und die direkte Kontrolle von Bildverarbeitung und Rekonstruktion ermöglichen. Hierzu gehören Projekte wie OpenCV und ARtoolkit. OpenCV [OpenCV 2006; Bradski und Kaehler 2008] ist als Sammlung von Algorithmen und Datenstrukturen zum Thema Computervision konzipiert und bietet daher viele grundlegende Funktionen zu Kamerakalibrierung, Bildauswertung und 3D-Rekonstruktion. Hieraus resultiert eine verhältnismäßig große Flexibilität, der Schwerpunkt der Bibliothek liegt aber bisher mehr bei komplexen Verfahren von Bildverarbeitung und Computervision und weniger bei einfachen Methoden für markerbasiertes latenzarmes Tracking. Diese Ausrichtung zeigt sich auch darin, dass OpenCV reine Funktionalität für den Datenverarbeitungsprozess anbietet, jedoch über keine Mechanismen zur Anbindung von Kameras oder Bildverarbeitungshardware verfügt. Der Einsatzbereich von ARtoolkit [Kato 2001] konzentriert sich auf die Entwicklung einfacher AR-Anwendungen. Neben Funktionen zu Kamerakalibrierung, Bildauswertung und Rekonstruktion wird hier auch ein Systemframework inklusive Kameraanbindung bereitgestellt. Die Trackingfunktionalität beschränkt sich auf spezielle quadratische Templatemarker (siehe Abbildung 2.7(f)), die für die Erfassung mit einer einzelnen Kamera ausgelegt sind. ARtoolkit ist als eine Art RAD-Tool (Rapid-Application-Development) für die schnelle Konzeption und Entwicklung von AR-Setups konzipiert. Durch den gewählten Markertyp und den Einzelkamerabetrieb sind jedoch sowohl Genauigkeit, als auch Stabilität und erreichbare Latenzzeiten für komplexere Trackinganwendungen nicht ausreichend.

Detailliertere Informationen zu den genannten sowie weiteren geräteunabhängigen Softwarelösungen finden sich in Anhang B.2.

2.5 Diskussion und Schlussfolgerungen für ein Systemkonzept

Diskussion zum Stand der Technik

Optische Verfahren haben sich gegenüber anderen 3D-Messverfahren als universelle Standardlösung etabliert. Optische Trackingverfahren zur Positionsbestimmung

zeichnen sich insbesondere auch durch ihre hohe Flexibilität bei unkonventionellen Aufgabenstellungen aus, an die sie durch die Auswahl von Typ, Anzahl und Positionierung geeigneter Kameras und Marker gut anpassbar sind – eine wichtige Voraussetzung für den geplanten Einsatz in medizintechnischen Projekten, insbesondere den medizinischen Simulatoren unserer Arbeitsgruppe VIPA.

Genau diese hohe Flexibilität ist jedoch mit verfügbaren kommerziellen Komplettsystemen nur schwierig umzusetzen. Sie präsentieren sich als abgeschlossene monolithische Systeme, die auf die Bewegungserfassung beim Motion Capture, in einer Cave[22], im Operationssaal und ähnlich gearteten Trackingumgebungen mit großen Kamerablöcken, einem Trackingvolumen im Kubikmeterbereich und starren Markeranordnungen an den Objekten ausgerichtet sind. Die Systeme zielen dabei prinzipiell auf hinreichende Genauigkeit und Geschwindigkeit für solche Standardeinsätze ab und erfüllen die zugehörigen Anforderungen auch exzellent. Auf Schwierigkeiten stößt der Einsatz in unüblichen Trackingumgebungen wie der Benutzerschnittstelle eines Simulators, insbesondere bei Szenarien, die eine Adaption des Systems erfordern. Zum einen resultieren diese Probleme aus der Festlegung der Systeme auf einen Markertyp, vorgegebene Kamerageometrien und starre Objekte. Anwendungen, die sehr kleine oder unkonventionelle Marker, kleine oder leichte Kameras, besondere Beleuchtungsbedingungen oder die Erfassung nicht-starrer Objekte erfordern, sind nur schwer zu realisieren. Zum anderen stellen die Systeme ausschließlich rekonstruierte 3D-Daten von Markern bzw. Objekten bereit, gewähren aber keinen Zugriff auf interne Daten der Verarbeitungsstufen, die z.B. bei komplexeren Rekonstruktionsverfahren hilfreich sein können. Am flexibelsten präsentieren sich Systeme mit Einzelkameramodulen, die zumindest den Zugriff auf vorverarbeitete Daten einzelner Kameramodule ermöglichen. Allerdings sind auch sie gewöhnlich auf einen dedizierten Marker- und Kameratyp festgelegt. Weiterhin bieten nicht alle der Systeme eine ausreichend niedrige Systemlatenz für den Einsatz in interaktiven Anwendungen wie VR-Simulatoren.

Die meisten geräteunabhängigen Softwarebibliotheken für optisches Tracking befassen sich mit Datenabstraktion und Weiterverarbeitung bereits rekonstruierter Daten innerhalb eines größeren, oft auch verteilten VR-Gesamtsystems. Sie sind damit weniger starr an Standardanwendungen gebunden als die Komplettsysteme. Datenflussbasierte Änsatze bieten dabei die größte Flexibilität bei der Fusion und Verteilung rekonstruierter Daten. Eine solche Architektur ist auch für die Realisierung des MOSCOT-Systems zweckmäßig, allerdings sollte die Datenflusskontrolle sich nicht auf rekonstruierte Daten beschränken, sondern vielmehr den gesamten Datenverarbeitungsprozess von der Bilderfassung bis zum rekonstruierten Objekt umfassen. Hin-

[22] Cave Automatic Virtual Environment, dreidimensionale Projektionsumgebung für VR-Anwendungen

gegen ist die von einigen Bibliotheken bereitgestellte Netzwerktransparenz bei den intendierten medizintechnischen Setups irrelevant, da diese üblicherweise auf einer einzelnen Maschine ablaufen. Für die Adaptabilität an unkonventionelle Trackingszenarien ist eine direkte Manipulation der grundlegenden Datenverarbeitungsschritte für Bildverarbeitung und Rekonstruktion notwendig. Eine solche Kontrolle über Algorithmen auf Bildebene stellen Computervision-Bibliotheken bereit. Damit existiert eine Teillösung für die Bereiche der Bildverarbeitung und auch Kameramodellierung, allerdings fehlt die notwendige Integration in ein größeres Rahmenwerk, das z.B. Hardwareanbindung und Objektverwaltung gewährleistet. Das Konzept von ARtoolkit entspricht am ehesten den Anforderungen an ein Trackingframework mit einer Gesamtsystemarchitektur. Es bietet Kameraanbindung mit Bildakquisition und Kalibrierung und Funktionen für Definition und Datenverarbeitung von Markern und Objekten. Seine Unterstützung für die schnelle Entwicklung neuartiger Anwendungen ist auch Entwicklungsziel für das MOSCOT-System, ohne dabei jedoch bei der Zahl der Kameramodule oder den unterstützten Markertypen Einschränkungen aufzuerlegen.

Schlussfolgerungen für das Systemkonzept

Angestrebt wird ein universelles, flexibel anpassbares Baukastensystem für optische Trackingaufgaben. Für ein breit angelegtes Einsatzspektrum ist die Unterstützung möglichst vieler unterschiedlicher Markertypen, Signalisierungen und Marker- bzw. Kameraanordnungen bei gleichzeitig anpassbarer, erweiterbarer, aber auch schlanker Systemarchitektur notwendig. Die Analyse verschiedener Klassen von Komplettsystemen favorisiert ein Systemdesign aus Einzelkameramodulen mit Flächenbildsensoren. Separate Kameramodule ermöglichen variable Positionierung, freie Wahl von Inside-Out- bzw. Outside-In-Konfiguration und Anpassung an unterschiedliche Trackingvolumina. Flächenkameras unterstützen alle vorhandenen Marker- und Signalisierungstechniken, so dass je nach Problemstellung geeignete Markertypen eingesetzt werden können. Eine Beschränkung auf Marker mit Punktcharakter (Repräsentation eines Raumpunkts mit drei Freiheitsgraden) ermöglicht eine breite Variabilität an unterstützten Markertypen bei gleichzeitig geringem Anpassungsbedarf der Datenverarbeitungsmodule. Weitergehende Einschränkungen hinsichtlich der Art und Technologie der Kamerasensoren sind nicht notwendig, da sie die Auslegung der Systemarchitektur nicht beeinflussen. Wichtig ist vielmehr die Unterstützung eines Spektrums verschiedener Sensortypen aus allen Leistungsbereichen, da für viele Anwendungen insbesondere die Kosten der optischen Trackinglösung eine entscheidende Rolle spielen. Die Option intelligenter Kameramodule mit integrierten Bildverarbeitungsressourcen ist sinnvoll zur Einschränkung von Bandbreiten- und Rechenleistungsanforderungen im Host-Rechner, da dieser bei den intendierten Anwendungen ja bereits mit der VR-Simulation größtenteils ausgelastet ist.

Die Datenverarbeitungsarchitektur selbst soll verschiedene Teilaspekte der vorgestellten Softwarelösungen vereinen, so eine frei konfigurierbare Datenflussarchitektur, die aber unterhalb des Middleware-Niveaus ansetzt und auch grundlegende Verfahren der Bildverarbeitung und Rekonstruktion umfasst. Sie sollte weiterhin eine enge Integration der Hardwarekomponenten ermöglichen. Da universell ausgelegte Systemarchitekturen häufig mit hohem Ressourcenbedarf und einem komplexen Aufbau einhergehen, ist es notwendig, von Beginn an auf die Umsetzung niedriger Systemlatenzen für den Einsatz in interaktiven VR-Anwendungen zu achten.

Kapitel 3

Datenfluss und Datenverarbeitungsmethoden

Als Ausgangspunkt für die Entwicklung einer universellen Systemarchitektur untersucht dieses Kapitel die dem Trackingvorgang zugrundeliegenden Datenverarbeitungsprozesse. Ziel ist die Identifikation von Strukturen und Verfahren, die die Basis einer generalisierten Datenflussarchitektur bilden können. Ausgehend von einer exemplarischen Trackinganwendung wird eine Unterteilung des Trackingprozesses in verschiedene Verarbeitungsphasen vorgenommen. Eine abstrakte Sicht auf den Datenfluss zeigt die den einzelnen Phasen zugehörigen Datenstrukturen auf und unterteilt sie nach dynamischen Trackingdaten und statischen Informationen zum System. Die Bestimmung dieser statischen Systemparameter während des Systemsetups wird kurz beschrieben. Der Hauptteil des Kapitels widmet sich sodann den einzelnen Verfahren und Algorithmen, die den identifizierten Datenverarbeitungsschritten zugrundeliegen; ausgehend von der Verarbeitung der Rohbilddaten bis hin zum rekonstruierten Objekt.

Die beschriebenen Methoden stehen beispielhaft für Trackinganwendungen, die bisher mit dem MOSCOT-Systembaukasten realisiert wurden. Sie stellen jedoch nur einen Ausschnitt aller von der Systemarchitektur unterstützten Lösungen dar. Am Ende der einzelnen Abschnitte werden daher jeweils kurz Ansätze für alternative oder erweiterte Verfahren aufgezeigt, auf die bei der Entwicklung zukünftiger Trackinganwendungen im Rahmen der MOSCOT-Architektur zurückgegriffen werden kann.

Die Ausführungen in diesem Kapitel erfolgen so weit als möglich unabhängig von der späteren konkreten Umsetzung in eine Systemarchitektur. Diese wird zusammen mit einer Beschreibung der implementierten Hard- und Softwaremodule erst im folgenden Kapitel 4 eingeführt. Die für die Ausführungen relevanten mathematischen Grundlagen der projektiven Geometrie sowie der Kameramodellierung finden sich im Anhang A.

3.1 Übersicht über den Datenfluss

3.1.1 Datenverarbeitungsschritte an einem Beispiel

Zur Veranschaulichung der Überlegungen in den folgenden Abschnitten wird zunächst der prinzipielle Ablauf der Verarbeitungsschritte bei einer für MOSCOT typischen optischen Trackinganwendung dargestellt (beispielsweise die optische Erfassung von Objektbewegungen in der Benutzerschnittstelle eines VR-Simulators). Abbildung 3.1 illustriert den Datenverarbeitungspfad von der Bildakquisition der Kameras bis hin zur Ausgabe der rekonstruierten Objektpose. Neben den aufeinanderfolgenden Datenverarbeitungsschritten (linke Spalte) sind die jeweiligen Ergebnisse (mittlere Spalte) und eine Repräsentation der dynamischen Trackinginformationen auf der jeweiligen Stufe dargestellt (rechte Spalte). Ausgehend von den Bilddaten werden nach einigen Bildvorverarbeitungsschritten typische Strukturen der Marker in den Bildern identifiziert. Mit ihnen werden die Bildkoordinaten der Marker bestimmt und nach weiteren Verarbeitungsschritten die räumlichen Markerpositionen rekonstruiert. Schließlich erfolgt die Bestimmung der Objektposen mittels dieser Markerpositionen. Charakteristisch für die fortlaufende Datenverarbeitung während des Trackingprozesses ist die im Verlauf abnehmende Datenmenge bei gleichzeitiger Zunahme der Komplexität der datenverarbeitenden Operationen.

Je nach Trackingszenario, Marker-, Kamera- und Objekttyp ändern sich Details in Aufbau und Abfolge einzelner Verarbeitungsschritte. Der im Diagramm illustrierte prinzipielle Ablauf bleibt jedoch für alle relevanten Einsatzszenarien des MOSCOT-Systems erhalten. Ähnliche Abläufe beschreiben auch andere Autoren in ihren Arbeiten zu optischen Trackingsystemen: vgl. beispielsweise die Darstellungen bei Luhmann [2010, Abb. 1.3 und 5.1] sowie Dorfmüller und Wirth [1998, Abb. 3] und Pintaric und Kaufmann [2007, Abb. 4]).

Für die Ausrichtung hinsichtlich einer universellen, flexibel einsetzbaren Datenflussarchitektur ist eine weitergehende Abstraktion vorteilhaft. Hierzu wird in der vorliegenden Arbeit eine Aufteilung basierend auf dem Fluss der dynamischen Trackingdaten durch die Verarbeitungsstufen sowie ihrer Verküpfung mit den statischen Informationen der Systembeschreibung vorgenommen. Abbildung 3.2 skizziert eine solche abstrahierende Sicht auf den Ablauf des Trackingprozesses.

3.1.2 Abstrakte Sicht auf den Datenfluss

Es lassen sich drei elementare Phasen der Datenverarbeitung charakterisieren: **Markerdetektion**, **Markerrekonstruktion** und **Objektrekonstruktion**.

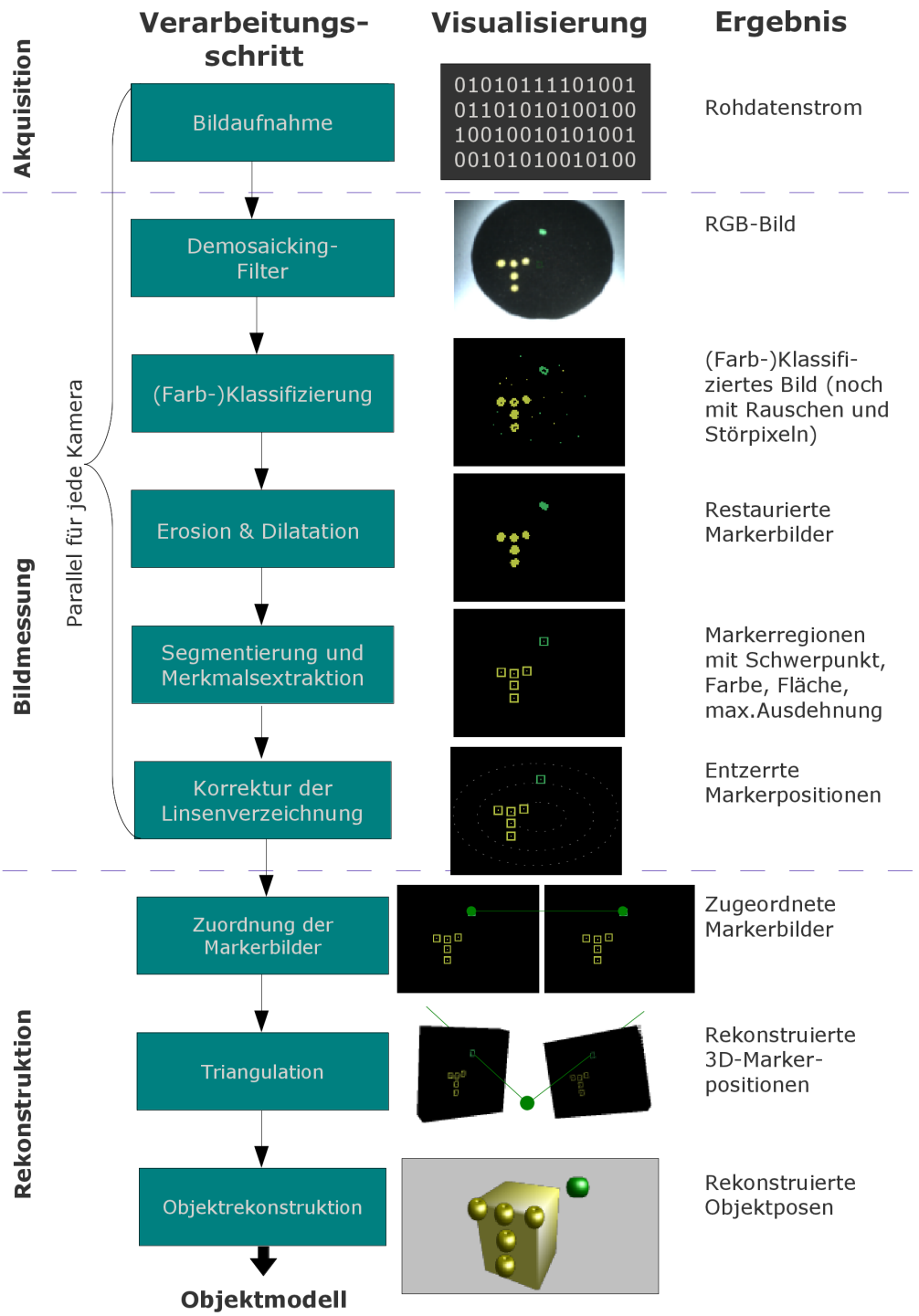


Abbildung 3.1: Beispiel für die Datenverarbeitungsschritte während des Trackingvorgangs in einer typischen Anwendung von MOSCOT in einem VR-Simulator.

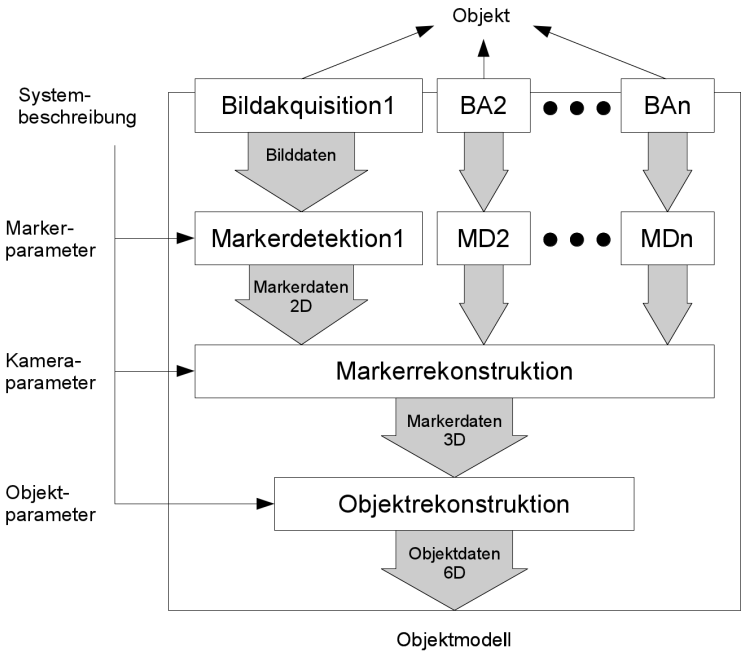


Abbildung 3.2: Abstrakte Sicht auf den Trackingprozess. Sie zeigt einen Überblick über die wesentlichen Datenverarbeitungsphasen, die Repräsentation der dynamische Trackingdaten als Schnittstellen zwischen den einzelnen Phasen und die Aufteilung der statischen Systeminformationen.

Jede der drei Phasen kann über die Repräsentation der Ein- und Ausgangsinformationen sowie die zur Verarbeitung der Trackingdaten erforderlichen systembeschreibenden Informationen (im Folgenden als Kamera-, Marker- bzw. Objektparameter bezeichnet) definiert werden:

Die Phase der Bildmessung oder **Markerdetektion** umfasst alle Verarbeitungsschritte, die die Positionen der Trackingmarker in den Kamerabildern bestimmen. Sie untergliedert sich weiter in einzelne Schritte zur Bildvorverarbeitung, bei der relevante Bildinformationen verstärkt und der Bildhintergrund unterdrückt werden sowie die eigentliche Erkennung, die die charakteristischen Strukturen der Marker im Bild identifiziert und lokalisiert. Diese Abläufe werden für jede Kamera des Trackingsystem parallel ausgeführt. Eingangsdaten sind die Rohbilddaten der jeweiligen Kamera, Ausgangsdaten beinhalten eine Liste der detektierten Markerpositionen. Der Detektionsvorgang erfordert Informationen zu den eingesetzten Markertypen in Form der Markerparameter.

Die Phase der **Markerrekonstruktion** rekonstruiert die 3D-Weltkoordinaten der Trackingmarker aus den detektierten Markerpositionen. In diesem Schritt werden

auch die bisher getrennten Kameradatenströme zu einer gemeinsamen Datenrepräsentation vereint. Der Rekonstruktionsvorgang ordnet die Markerbilder der verschiedenen Kameras einander zu und ermittelt daraufhin die Weltkoordinaten durch Triangulation. Eingangsdaten sind die detektierten Markerdatensätze der verschiedenen Kameras, der Ausgangsdatensatz umfasst eine Liste mit rekonstruierten 3D-Markerpositionen. Für die Rekonstruktion müssen die Parameter des Kamerasystems bekannt sein.

Die Phase der **Objektrekonstruktion** berechnet schließlich Position und Orientierung der Trackingobjekte. Dabei werden rekonstruierte Marker ihren Trackingobjekten zugeordnet und deren Pose über die Schätzung der absoluten Orientierung bestimmt. Eingangsdaten beinhalten die Liste rekonstruierter Markerkoordinaten, Ausgangsdaten umfassen die Objektposen. Objektparameter, d.h. Informationen zur relativen Anordnung der Marker eines Trackingobjektes, müssen verfügbar sein.

3.1.3 Entkopplung von Systembeschreibung, Daten und Datenverarbeitung

Abbildung 3.2 illustriert weiter auch die Differenzierung des Informationsflusses in statische Parameter, die das System und seine Entitäten (Kameras, Marker, Objekte) beschreiben und dynamische Daten, die die erfassten Trackingdaten während der Verarbeitung repräsentieren. Die Systemparameter werden für den Trackingvorgang als konstant betrachtet und müssen vor dessen Beginn während des Systemsetups bestimmt werden (Kalibrierung und Konfiguration). Ihre Unterteilung in Marker-, Kamera- und Objektparamenter korreliert mit der Aufteilung der Datenverarbeitung in drei Phasen. Für die dynamischen Trackingdaten lassen sich mit der Ende des letzten Kapitels getroffenen Festlegung auf punktartige 3DOF-Marker vier grundlegende Repräsentationen während der verschiedenen Datenverarbeitungsphasen definieren: Bilddaten, zweidimensionale Markerpositionen in den Einzelbildern, rekonstruierte räumliche Markerpositionen und schließlich rekonstruierte Objektposen. Die Trackingdaten werden während des Trackingprozesses fortlaufend in Echtzeit erfasst und synchron mit der Aufnahmerate der Kameras in den einzelnen Datenverarbeitungsstufen verarbeitet.

Durch die blockweise funktionale Aufteilung findet eine Entkoppelung verschiedener Bereiche des Datenverarbeitungsprozesses statt. Die unterschiedlichen Repräsentationen der dynamischen Daten dienen dabei als klar definierte Schnittstellen dieser Blöcke. Damit können Abhängigkeiten innerhalb des Datenflusses reduziert und eine lokale Kapselung der Verarbeitungsstrukturen erreicht werden. Letztendlich erleichtert sich damit die Modularierung der einzelnen Datenverarbeitungsschritte und ihre Anpassung und Wiederverwendung bei neuartigen Aufgabenstellungen.

Zwei zentrale Beschränkungen des Systemkonzepts ermöglichen diese Herangehensweise. Die Festlegungen auf punktartige Marker mit drei Freiheitsgraden und auf den Einsatz flächiger Bildsensoren erlaubt es, die Eigenheiten der verschiedenen Markertypen innerhalb der Markerdetektionsphase zu kapseln und die weiteren Verarbeitungsschritte auf einem generischen 3DOF-Markermodell aufzubauen. Damit bleibt die Geometrie der Markerrekonstruktion erhalten und die zugehörigen Rekonstruktionsverfahren hängen nur von der Art der Markercodierung und der Redundanz der Kameraansichten und Objektmarker ab. Anpassungen des MOSCOT-Systems an neue Aufgabenstellungen finden über die Auswahl geeigneter Markertypen und Kameras bzw. die Festlegung einer geeigneten Marker-Kamera-Anordnung statt. Anpassungen der Datenverarbeitung werden nur beim Einsatz veränderter Markertypen notwendig und auch diese Änderungen bleiben lokal beschränkt auf die Phase der Markerdetektion. Hier sind einzelne Bildvorverarbeitungsschritte möglicherweise wiederverwendbar, die Erkennung der spezifischen Markermerkmale in den Bildern bleibt jedoch sehr stark auf den jeweiligen Markertyp zugeschnitten.

3.1.4 Repräsentation der dynamischen und statischen Daten des Trackingprozesses

Im Folgenden wird die Zusammenstellung der am Trackingprozess beteiligten dynamischen und statischen Informationen näher erläutert. Tabelle 3.1 gibt einen Überblick über den Aufbau der dynamischen Datensätze. Sie repräsentieren wie bereits erläutert die erfassten Informationen über die Trackingszene, jeweils auf den verschiedenen Stufen des fortlaufenden Datenverarbeitungsprozesses und stellen auch gleichzeitig das Schnittstellenformat zwischen den Datenverarbeitungsphasen dar. [1]

Kameradatensätze umfassen die Rohdaten der Kameras, je nach Sensortyp als Pixelfeld von Rohbilddaten, Intensitäts- oder Farbwerten. Die 2D-Markerdaten beinhalten eine Liste der detektierten Markerpositionen sowie ihre Codierungsinformationen, jeweils für jedes Kamerabild. Das generische Markermodell aus Position und Codierungsinformation wird hier ergänzt um markerspezifische Merkmale, die den nachfolgenden Verarbeitungsschritten bei Zuordnung und Rekonstruktion der Marker helfen. Die 3D-Markerdaten umfassen die Liste der rekonstruierten 3D-Markerpositionen im Trackingvolumen. Bei codierten Markern werden auch Codierungsinformationen zur Unterstützung der Objektrekonstruktion gespeichert. Objektdaten schließlich bestehen aus Position und Orientierung der Objekte im Trackingvolumen. [2]

[1] Eine detaillierte Auflistung einzelner Elemente mit einer Abschätzung des Speicherbedarfs sowie die Umsetzung in Klassenstrukturen folgen in Kapitel 4.

[2] Der Spezialfall der Pose und Verformung bei nicht-starren Objekten ist stark anwendungsabhängig

dynamische Trackingdaten	Bilddaten (pro Kamera)	Feld (Zeilen x Spalten) mit Pixelwerten
		(je nach Sensortyp: Rohdaten, RGB, Monochrom)
	Markerdaten 2D (pro Kamera)	Liste detektierter Markerpositionen im Kamerabild
		Codierungsinformationen[1]
		zusätzliche markerspezifische Merkmale[2]
	Markerdaten 3D	Liste rekonstruierter 3D-Weltkoordinaten der Marker
		Codierungsinformationen[1]
	Objektdaten 6DOF	Liste mit Objektposen (Position & Orientierung)

[1] bei codierten Markern
[2] z.B. Fläche, max. Ausdehnung, etc.

Tabelle 3.1: Repräsentation der dynamischen Daten während des Trackingprozesses

statische Systeminformationen	Kameraparameter	Parameter des Lochkameramodells
		Parameter der Linsenverzeichung
	Objektparameter	Positionen von 3 Referenzmarkern[1]
		optional Positionen weiterer Objektmarker
	Markerparameter	Markertyp
		Codierungsinformationen[2]
		typspezifische Parameter zur Markerdetektion[3]

[1] Definition des lokalen Objektkoordinatensystems
[2] bei codierten Markern
[3] z.B. Bildregionen, Farbregionen, Schwellwerte, etc.

Tabelle 3.2: Repräsentation der statischen Informationen des Trackingprozesses

Die statischen Systeminformationen werden durch die Parameter der systembestimmenden Entitäten Kameras, Marker und Objekte beschrieben. Tabelle 3.2 gibt einen Überblick.

Eine Kamera wird definiert durch die Parameter des Lochkameramodells. Zur Verbesserung der Rekonstruktionsgenauigkeit werden zusätzlich Parameter der Linsenverzeichnung berücksichtigt (siehe Abschnitt 3.2.1). Ein Trackingobjekt wird bestimmt durch einen Satz von mindestens drei Markern, die ein lokales Objektkoordinatensystem aufspannen. Weitere redundante Marker können zur Erhöhung von Genauigkeit und Stabilität eingesetzt werden (siehe Abschnitt 3.2.2). Ein Marker wird allgemein beschrieben durch den Markertyp und die Codierungsinformationen (bei codierten Markern). Hinzu kommen markertypspezifische Parameter, die die Lokalisation im Kamerabild ermöglichen. Dies sind beispielsweise radiometrische und

und kann deshalb nicht in dieser allgemeinen Datenstruktur abgebildet werden. Für ein Beispiel sei auf die Anwendung in Abschnitt 6.2 verwiesen.

geometrische Eigenschaften, aber auch Parameter zu Bildvorverarbeitungsoperationen (siehe Abschnitt 3.2.3).

3.2 Bestimmung der statischen Systemparameter

Die Systemparameter werden vor dem eigentlichen Trackingbetrieb in einem separat durchgeführten Konfigurationsvorgang bestimmt, im Folgenden als Systemsetup bezeichnet. Die dabei vorgenommenen Kalibrierungs- und Konfigurationsschritte sind nicht zeitkritisch und erfordern daher im Unterschied zum Trackingbetrieb keine Datenverarbeitung unter Echtzeitbedingungen. Nach dem Systemsetup steht ein definiertes Modell des Kamerasystems und der zu erfassenden Objekte und Marker zur Verfügung.

Da die vorliegende Arbeit sich auf die Datenflussarchitektur des Online-Trackingprozesses konzentriert, wird auf die Bestimmung der Systemparameter nur kurz eingegangen. Kamerakalibierung und Objektregistrierung sind in der Literatur vielfältig beschriebene Aufgabenstellungen und es existieren etablierte Standardverfahren für ihre Lösung. Der Vorgang der Markerkonfiguration ist stark vom eingesetzten Markertyp abhängig. Der Spezialfall von Farbmarkern wird ausführlicher dargestellt, da sie bei herkömmlichen Trackinglösungen selten eingesetzt werden, bei den Anwendungen des MOSCOT-Sytems in medizinischen VR-Simulatoren jedoch eine wichtige Rolle einnehmen.

3.2.1 Kameraparameter

Die dreidimensionale Rekonstruktion einer Szene aus zweidimensionalen Bilddaten erfordert ein mathematisches Modell der Kameraabbildung. Ein solches Modell stellt die Projektive Geometrie bereit (siehe Anhang A.1). Dabei werden Kameras als Lochkameras modelliert.

Die allgemeine Form der linearen Abbildung einer Lochkamera wird durch die folgende Projektionsgleichung in homogenen Koordinaten beschrieben, bei der der Weltpunkt $\mathbf{X} = (X, Y, Z, 1)^T$ auf den Bildpunkt $\mathbf{x} = (x, y, w)^T$ abgebildet wird:

$$\mathbf{x} = \mathbf{P}\mathbf{X}, \qquad \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \mathbf{P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.1}$$

Die homogene 3x4 Matrix $\mathbf{P}$ wird als Projektionsmatrix bezeichnet und umfasst die Parameter des Kameramodells. Sie kann in eine 3x3 Matrix $\mathbf{K}$ und eine 3x4 Matrix

$[\mathbf{R}|\mathbf{t}]$ zerlegt werden.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{3.2}$$

Die Matrix $\mathbf{K}$ beschreibt die innere Orientierung der Kamera in Form der internen Parameter des Kameramodells. Es sind die Brennweiten in x- und y-Richtung f_x und f_y, der Hauptpunkt $(p_x, p_y)^T$ als Durchstoßpunkt der optischen Achse mit der Bildebene sowie die Schiefe s (die Verkippung der Bildkoordinatenachsen gegeneinander).

$$\mathbf{K} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$\mathbf{R}$ und $\mathbf{t}$ beschreiben die äußere Orientierung der Kamera, die Rotation und Translation gegenüber dem Weltkoordinatensystem. Sie werden auch als externe Kameraparameter bezeichnet.

Eine ausführlichere Darstellung der Herleitung der Parameter des Lochkameramodells findet sich in Anhang A.3.[3]

Lochkameramodell und obige Projektionsgleichung stellen ein lineares Modell des Abbildungsprozesses dar. Die nichtlinearen Abbildungseigenschaften realer Linsen können damit nicht modelliert werden. Erweiterte Modelle berücksichtigen daher noch die nichtlineare Linsenverzeichnung über einen zusätzlichen Korrekturterm $\mathbf{x}_{dist}$, der im Anschluss an die perspektivische Projektion nach Gl. (3.1) ausgeführt wird:

$$\mathbf{x}_d = \mathbf{x}_u + \Delta\mathbf{x}_{dist}(\mathbf{x}_u) \tag{3.4}$$

$\mathbf{x}_d = (x_d, y_d)^T$ bezeichnen hier die verzeichneten Bildkoordinaten, $\mathbf{x}_u = (x_u, y_u)^T$ die ursprünglichen, unverzeichneten. Der Korrekturterm setzt sich zusammen aus einer radialen und einer tangentialen Komponente:

$$\Delta\mathbf{x}_{dist}(\mathbf{x}_u) = \Delta\mathbf{x}_{rad}(\mathbf{x}_u, k_1, k_2) + \Delta\mathbf{x}_{tan}(\mathbf{x}_u, t_1, t_2) \tag{3.5}$$

$\Delta\mathbf{x}_{rad}$ modelliert die radiale Linsenverzeichung mittels zweier Parameter k_1, k_2 und $\Delta\mathbf{x}_{tan}$ die tangentiale Linsenverzeichnung durch die zwei Parameter t_1, t_2, jeweils an der Position $\mathbf{x}_u$. Damit erweitern sich die Kameraparameter des Lochkameramodells um die Verzeichnungsparameter k_1, k_2, t_1, t_2. Die Berechnung der radialen und axialen Korrekturterme wird in Anhang A.4 dargestellt.

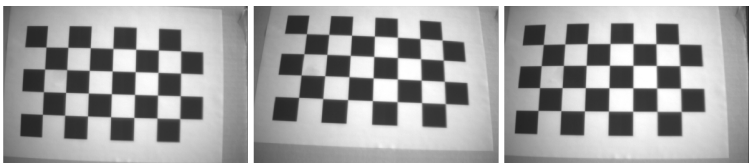


Abbildung 3.3: Planares Kalibrierfeld aus drei verschiedenen Kameraansichten

Kamerakalibrierung

Die Bestimmung der Kameraparameter, die sog. Kamerakalibrierung ist ein Standardproblem der Computer Vision und wird in der Literatur ausführlich beschrieben, so z.B. in [Tsai 1987; Zhang 1998; Hartley und Zisserman 2000]. Eine Übersicht über Kamerakalibrierung in der Photogrammetrie findet sich in [Luhmann 2010, Kap. 7.4].

Die Kalibrierung des MOSCOT-Trackingsystems beschreibt Handel [2009]. Beim aktuellen Entwicklungsstand wird der Kalibriervorgang vor dem eigentlichen Trackingprozess mittels eines bekannten Kalibrierobjektes durchgeführt (Testfeldkalibrierung). Verschiedene Kalibrierungsverfahren wurden für MOSCOT implementiert. Als hauptsächliches Verfahren findet die Methode nach Zhang [1998] Anwendung. Sie verwendet ein bekanntes zweidimensionaled Kalibrierfeld, welches unter verschiedenen Aufnahmewinkeln innerhalb des Trackingvolumens eingemessen wird (Abbildung 3.3). Durch seine Stabilität, freie Verfügbarkeit und einfachen Einsatz hat es sich als weit verbreitetes Standardverfahren etabliert. Auch die im Abschnitt 2.4 beschriebenen Computer-Vision-Bibliothek OpenCV greift auf dieses Verfahren für die Kalibrierung der Kameras zurück. Weitere Ansätze wurden innerhalb des MOSCOT-Projekts getestet, so z.B. für spezielle Anwendungsgebiete eine Kalibrierung mit einem eindimensionalen Kalibrierobjekt [Zhang 2004] oder einzelnen Kalibrierpunkten [Azarbayejani und Pentland 1995]. Beide Verfahren eignen sich für Anwendungen bei denen die Verwendung eines großflächigen Kalibrierobjektes im Trackingvolumen Handhabungsprobleme aufwirft. Die erreichbare Genauigkeit ist jedoch geringer. Handel [2004] beschreibt Grundlagen und Implementierung dieser Alternativverfahren innerhalb des MOSCOT-Systems.

[3]Das Lochkameramodell und die hier dargestellten Abbildungsgleichungen gelten nur für Kameras mit punktförmigem Projektionszentrum. Kameras mit Abbildung auf ein lineares Sensorfeld (Zeilenkameras mit Zylinderlinsen, Pushbroomkameras) erfordern eine Anpassung der Projektionsgleichungen. Da sie bei der Realisierung des MOSCOT-Systems keine Berücksichtigung finden, wird hier nur auf die Zentralprojektion bei Flächenkameras eingegangen.

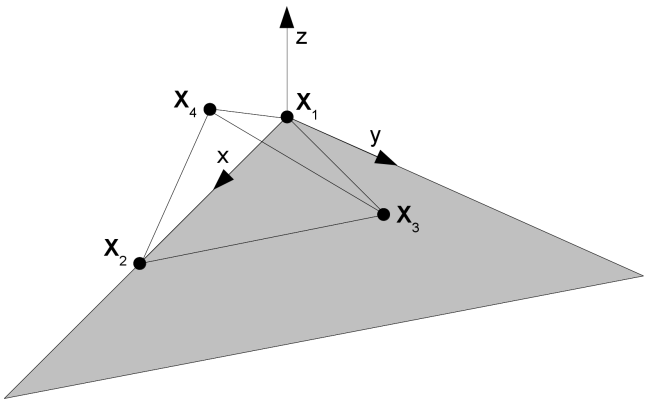


Abbildung 3.4: Definition des Objektkoordinatensystems durch die Objektmarker

3.2.2 Objektparameter

Die Objektparameter beschreiben die Anordnung der Marker auf den Trackingobjekten. Für den allgemeinen Fall eines frei beweglichen Objekts für das sowohl Position als auch Orientierung zu bestimmen sind (6DOF-Tracking), sind mindestens drei 3DOF-Marker erfoderlich. Bei zwei Markern bleibt die Rotation um die Verbindungsachse unbekannt (5DOF-Tracking), bei nur einem 3DOF Marker ist nur die Positionsbestimmung möglich. Über zusätzliche Randbedingungen (z.B. weitere Objektpunkte an definierten Positionen oder einen eingeschränkten Bewegungsraum) bzw. weitere Sensorik können möglicherweise trotzdem Objektposen erfasst werden, es soll hier aber der allgemeine Fall mit mind. 3 Markern pro Objekt erläutert werden.

Ein allgemeines 6DOF-Trackingobjekt wird beschrieben durch einen Satz von mindestens drei Markern $(\mathbf{X_1}, \mathbf{X_2}, .., \mathbf{X_n}), n \geq 3$, die ein lokales Objektkoordinatensystem definieren. $\mathbf{X_1}$ bestimmt den Referenzpunkt des Objekts (den Ursprung des Objektkoordinatensystems), $\mathbf{X_2}$ und $\mathbf{X_3}$ spannen das Objektkoordinatensystem auf. Die x-Achse ist gegeben durch $\mathbf{X_2} - \mathbf{X_1}$, die y-Achse als Orthogonale zur x-Achse in der von $(\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3})$ aufgespannten Ebene und die z-Achse als Senkrechte zu dieser Ebene (s. Abbildung 3.4). Die drei Marker dürfen demnach nicht kollinear angeordnet sein, wobei in diesem Fall auch kein 6DOF-Tracking möglich wäre. Weitere redundante Trackingmarker dienen zur Erhöhung von Genauigkeit und Stabilität. Während der Objektrekonstruktion wird mittels der Objektparameter zum einen das Objekt identifiziert, zum anderen seine äußere Orientierung (Lage und Ausrichtung) im Kamerakoordinatensystem bestimmt.[4]

[4]Einen Spezialfall stellen deformierbare Objekte dar, für die kein markerdefiniertes Objektkoordinatensystem angegeben werden kann. Daher erfolgt bei ihnen lediglich eine codierungsgestützte Zuordnung der Marker zum Objekt ohne Berücksichtigung der Markerpositionen.

Position und Anordnung der Objektmarker definieren sowohl das Objekt selbst als auch sein Referenzkoordinatensystem. Je nach Anwendungsfall, Art der zu trackenden Objekte oder zusätzlicher Ausrüstung können verschiedene Möglichkeiten zur Bestimmung der Objektparameter genutzt werden:

- Vorgefertigte Markeranordnungen (z.B. Markersterne für medizinische Anwendungen) werden vom Hersteller bereits mit einem geeigneten Parametersatz ausgeliefert, der die Markeranordnung beschreibt.
- Bei speziell angefertigten Trackingobjektion können häufig die Referenzpunkte der Markerpositionen aus CAD-Daten der Objektfertigung entnommen werden. Fertigungstoleranzen beeinflussen jedoch direkt die erreichbare Rekonstruktionsgenauigkeit und es ist wichtig, dass die mechanischen Referenzpunkte des CAD-Modells direkt den Passmarken des optischen Trackings entsprechen.
- Ein weiterer Ansatz besteht im Einmessen der Markerpositionen über ein Referenzsystem. Ideal ist eine identische Erfassung der Objektmarker im Referenz- und im MOSCOT-System. Alternativ besteht die Möglichkeit der Antastung der Markerpositionen über einen getrackten Pointer.
- Eine häufig eingesetzte Methode bei Anwendungen mit unkritischen Genauigkeitsanforderungen (so auch bei einigen Simulatoranwendungen des MOSCOT-Systems) ist das Einmessen der Referenzpositionen der Objektmarker über das Trackingsystem selbst. Offensichtlich reduziert sich hier die erreichbare Genauigkeit, da bereits die Objektreferenzdaten mit einem Fehler im Bereich der Messgenauigkeit des Trackingsystems behaftet sind.

Auf die Auswahl geeigneter Markeranordnungen für robuste Objektrekonstruktion und -identifikation wird an dieser Stelle nicht weiter eingegangen (bspw. müssen Anordnungen aus drei gleichartigen Markern ein unregelmäßiges Dreick bilden, um eine eindeutige Orientierungsbestimmung zu ermöglichen). Pintaric und Kaufmann [2008] befassen sich ausführlich mit dem Entwurf robuster starrer Anordnungen aus 3–4 Markern. Weitere Überlegungen zu der Thematik finden sich außerdem in [Dorfmüller 1999; Pintaric und Kaufmann 2007; Jansen et al. 2007; Steinicke et al. 2007].

3.2.3 Markerparameter

Während das Kameramodell und die Objektdefinitionen[5] für alle Anwendungsbereiche des MOSCOT-Systems unverändert bleiben, sind die Markerparameter stark vom jeweils verwendeten Markertyp abhängig. Ein Marker im Kontext der universellen Systemauslegung von MOSCOT wird dabei definiert über alle Informationen, die

[5] für Starrkörper

seine Lokalisierung im Kamerabild ermöglichen. Das sind zum einen Markertyp, Codierung und radiometrische und geometrische Eigenschaften, zum anderen auch Parameter (Schwellwerte, Toleranzbereiche) der Bildvorverarbeitungsoperationen, die für eine robuste Erkennnung notwendig sind (siehe die Erläuterung der Markerdetektionsmethoden in Abschnitt 3.3). Viele dieser Informationen können während des Systemsetups direkt vorgegeben werden (so der Typ des Markers, Farb- bzw. Monochromaufnahme, Art der Referenzpunktbestimmung über Schwerpunkt, Linienschnitt etc.). Bei einigen zusätzlichen Parametern der Markerdetektionsverfahren ist es jedoch notwendig, sie während der Markerkonfiguration in einem interaktiven Vorgang einzumessen, so z.B. die Farbwerte der Farbmarker.

Zentraler Schritt der Markererkennung ist die Differenzierung relevanter Markerstrukturen von der Umgebung. Die Detektion der meisten monochromen Markertypen basiert auf Intensitätsunterschieden von Marker und Hintergrund bzw. von Regionen innerhalb des Markers selbst (vgl. Abbildung 2.7), so dass die Markerstrukturen mittels einfacher Schwellwertfunktionen bestimmt werden können. Durch den hohen Kontrast ist diese Vorgehensweise auch robust gegenüber Veränderungen der Beleuchtungssituation. Die Konstruktion einer ähnlichen Schwellenfunktion für die Abgrenzung von Farbwerten bei der Farbmarkerdetektion erweist sich als deutlich komplexer, da Farbwerte im Unterschied zum Kontrast einen mehrdimensionalen Parameterraum repräsentieren und sich durch Effekte wie Schattierungsverläufe innerhalb der Markerstrukturen, inhomogene bzw. veränderliche Beleuchtung, Kameracharakteristik etc. in diesem Parameterraum verschieben (Problem der Farbkonstanz [Jain et al. 1995, Kap. 10.5] [Funt et al. 1998]). Verschiedene teilweise recht komplexe Modelle zur robusten Farbdefinition unter variablen Umgebungsbedingungen wurden vorgestellt (bspw. von McKenna et al. [1999] und Álvarez et al. [2010]). Für den Einsatz bei MOSCOT wird demgegenüber eine einfache Repräsentation der Markerfarben gesucht, die unter kontrollierten Beleuchtungsbedingungen in geschlossener Umgebung eine ausreichend robuste und gleichzeitig ressourceneffiziente Erkennung der Markerstukturen erlaubt.

Repräsentation und Konfiguration der Markerfarben

In Abschnitt 2.2.5 wurde bereits kurz die Farberfassung von menschlichem Auge und Farbkamerasensoren über drei getrennte Spektralkanäle erörtert, die in der Kombination zu einem dreidimensionalen Farbraum führen.[6] Je nach Wahl der Basis des

[6] Die Dreidimensionalität der Farbraumdarstellung hat also keinen physikalischen Hintergrund, sondern ist eine Eigenschaft der trichromatischen Sinneswahrnehmung des Menschen. Die meisten Säugetiere erfassen beispielsweise nur zwei verschiedene Spektralkanäle und verfügen damit nur über einen 2D-Farbraum, Vögel hingegen über einen vierdimensionalen Farbraum [Jacobs und Rowe 2004].

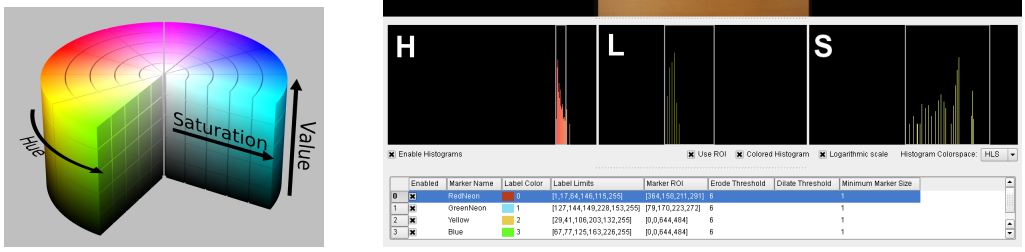
Farbraums existieren unterschiedliche Farbmodelle. Das durch einen Farbkamerasensor gelieferte Bild gibt die Rot-, Grün- und Blauanteile des erfassten Spektrums nach Gl. (2.2) wieder, und ist demnach im RGB-Farbraum kodiert. Der Farbwert eines Markers entspricht einem Punkt dieses Farbraums. Unter den oben angeführten Beleuchtungseinflüssen ist der Farbwert jedoch nicht konstant, so dass für eine eindeutige Klassifizierung der Markerfarbe die Definition einer ganzen Farbregion notwendig wird.

Für eine robuste Konfiguration dieser Farbregionen bieten sich Farbräume an, die im Unterschied zu RGB die reine Farbinformation H (Hue, Farbton) als getrennte Dimension darstellen. Leicht unterschiedliche Parametrisierungen in den verbleibenden zwei Dimensionen Helligkeit (L,V,I) und Sättigung (S,D) führen dann zu verschiedenen, konzeptionell ähnlichen Farbräumen (z.B. HSV, HSI, HDI, HSL) [Kravtchenko 1999; Bovik 2009]. Der Farbton H wird bei allen als Winkel innerhalb des Farbspektrums dargestellt (Abbildung 3.5(a) zeigt als Beispiel eine Darstellung des Farbraums HSV). Durch Auswahl enger Grenzen in der Farbwertkomponente und breiterer Bereiche für die Helligkeit- und Sättigungskomponente kann auf einfache Weise eine Farbregion definiert werden, die eine recht robuste Kompensation von Beleuchtungsänderungen ermöglicht [Vezhnevets et al. 2003; Azad et al. 2008]. Diese an das menschliche Farbempfinden angelehnten Modelle ermöglichen darüberhinaus dem Nutzer eine intuitive Visualisierung und Konfiguration der Grenzen der Schwellwertfunktionen. Verschiedene solcher Farbdefinitionen werden von MOSCOT unterstützt, bei den meisten Anwendungen wird eine Konfiguration der Markerfarbe als achsenparallele disjunkte Region im HSL-Farbraum eingesetzt. Abbildung 3.5(b) zeigt als Beispiel die Histogrammauswertung der verschiedenen Farbwerte eines roten passiven Farbmarkers im HSL-Farbraum und die Konfiguration der zugehörigen Grenzen (H wird hier aus Visualisierungsgründen linear dargestellt).

Für eine effiziente Klassifizierung der Markerfarbwerte während des Trackingvorgangs ist es sinnvoll, die Farbraumkonversion vorab zu berechnen und mit der Definition der Markerbereiche im jeweiligen Zielfarbraum in einer Lookuptabelle zusammenzufassen. Diese ordnet dann den von den Kameras gelieferten RGB-Pixelwerten direkt Markerfarbregionen zu:

$$(R, G, B) \rightarrow Farbregion1, Farbregion2, Farbregion3, ..., undefiniert$$

Die Erzeugung dieser Lookup-Tabelle aus der Definition der Markerfarbregionen in einem farbtonbasierten Farbraum erfolgt bereits vor dem eigentlichen Trackingprozess während der Systemkonfiguration. Die folgenden Gleichungen beschreiben hierzu die Konvertierung zu den bei MOSCOT hauptsächlich eingesetzten Farbräumen



(a) Darstellung des Farbwinkels H am Beispiel des Farbraums HSV[7]

(b) Konfiguration der Markerfarben über Grenzen im HSL(HLS)-Histogramm eines Markerbildes

Abbildung 3.5: Farbkonfiguration über getrennte Einstellung von Farbwert H und Helligkeits- bzw. Sättigungsbereichen.

$(R,G,B) \rightarrow (H,S,L)$ bzw. $(R,G,B) \rightarrow (H,S,I)$.

$$
\begin{aligned}
H &= \begin{cases} 0 & \text{wenn } max(R,G,B) = min(R,G,B) \\ \frac{G-B}{max(R,G,B)-min(R,G,B)} & \text{wenn } max(R,G,B) = R \\ \frac{B-R}{max(R,G,B)-min(R,G,B)} + 2 & \text{wenn } max(R,G,B) = G \\ \frac{R-B}{max(R,G,B)-min(R,G,B)} + 4 & \text{wenn } max(R,G,B) = B \end{cases} \\
I &= \frac{1}{3}(R+G+B) \\
L &= \frac{1}{2}(max(R,G,B) + min(R,G,B)) \\
S &= \begin{cases} 0 & \text{wenn } max(R,G,B) = min(R,G,B) \\ \frac{max(R,G,B)-min(R,G,B)}{1-|max(R,G,B)+min(R,G,B)-1|} & \text{sonst} \end{cases}
\end{aligned}
\tag{3.6}
$$

Durch den Einsatz von Lookup-Tabellen können prinzipiell beliebige Beschreibungen der Markerfarbregionen eingesetzt werden, solange sie statisch vorberechnet werden können. Beim Einsatz bedruckter passiver Farbmarker kann z.B. eine komplexere Definition der Markerfarbregionen mittels eines dichromatisches Farbmodells im HSI-Farbraum eine bessere Kompensation von Beleuchtungsänderungen erzielen [Shafer 1985; Kravtchenko 1999].

[7] Diagramm von SharkD/commons.wikimedia.org, Lizenz: CC-BY-SA-3.0

Prinzipiell zeigte sich während der Entwicklung des MOSCOT-Systems, dass das Tracking von Farbmarkern nur bei kontrollierter Beleuchtungssituation eine ausreichende Farbkonstanz für robuste Markererkennung aufweist. Es empfiehlt sich daher hauptsächlich bei Simulatoren mit geschlossenem Trackingaufbau, die eine gute Kontrolle der Beleuchtungsbedingungen ermöglichen. Auch dann ist allerdings nur eine begrenzte Zahl verschiedener Farbregionen sicher voneinander trennbar (ca. 4–8 je nach Beleuchtungs- und Objekteigenschaften). Die Verwendung von Farbmarkern in offenen Setups ist problematisch, da die Farbwerte sich bei Beleuchtungsänderung stärker verschieben, und so keine ausreichend robuste und genaue Erfassung der Markerstrukturen und damit der Markerposition im Bild zulassen. In einer möglichen Weiterentwicklung des Systems könnte dies über die dynamische Definition von Farbregionen kompensiert werden, die sich mit etwaigen Beleuchtungsänderungen mitverschieben. Es kann dann allerdings keine Repräsentation mehr über vorberechnete Lookup-Tabellen erfolgen.

3.3 Markerdetektion

Der Prozess der Markerdetektion überführt die Kamerarohdaten als bildhafte Repräsentation der Trackingszene in eine nicht-bildhafte, symbolische Darstellung. Dadurch reduziert sich der Informationsgehalt des Kamerabildes auf die für den weiteren Trackingprozess relevanten Informationen in Form einer Liste von detektierten Markerstrukturen.

Durch die Beschränkung des Systemkonzepts auf 3DOF-Marker ist die Markerdetektion der einzige Teil der Datenverarbeitung, der für die Unterstützung eines neuen Markertyps angepasst werden muss. Eigenheiten des Einsatzes spezieller Markertypen mit komplexeren radiometrischen bzw. geometrischen Merkmalen (z.B. Farbmarker, Marker mit innerer Struktur) bleiben innerhalb dieser Phase gekapselt und sind im Ergebnis für die weiteren Verarbeitungsstufen nicht von einfachen punktartigen Markern (wie z.B. retroreflektierenden IR-Marken) zu unterscheiden.

Die Markerdetektionsverfahren in dieser Arbeit konzentrieren sich auf die Detektion punktförmiger (3DOF) Farb- und Monochrom-Marker ohne innere Struktur sowie deren Schwerpunktsbestimmung über Flächenmomente. Der Einsatz von Codierungsinformationen zur Markeridentifikation und Rekonstruktion wird unterstützt, die Erfassung ist bisher aber nur für farbcodierte Marker implementiert. Auf mögliche Erweiterungen zur Detektion von 3DOF-Markertypen mit inneren Kantenstrukturen bzw. komplexere Verfahren zur Merkmalsbestimmung wird kurz am Ende dieses Abschnitts eingegangen.

3.3.1 Kriterien zur Auswahl geeigneter Bildverarbeitungsalgorithmen

Ohne zu sehr der Implementierung im folgenden Kapitel vorzugreifen, muss bereits bei der Auswahl der Datenverarbeitungsalgorithmen die Forderung nach niedrigen Latenzzeiten und geringem Ressourcenbedarf des MOSCOT-Systems berücksichtigt werden. Das betrifft in besonderem Maße die Auslegung der Algorithmen der Markerdetektionsphase. Es kommen daher vornehmlich einfache Standardmethoden der Bildverarbeitung zum Einsatz wie Punktoperatoren und lokale Operatoren mit kleinem Fensterbereich (3x3 oder 5x5). Eine Einführung geben z.B. [Jain et al. 1995; Shapiro und Stockman 2001; Jähne 2005]. Diese einfachen Operatoren können direkt auf dem Pixeldatenstrom der Kameras arbeiten und erfordern keine Zwischenspeicherung der Bilder.[8] Solche Zugriffsmuster kommen sowohl der Verarbeitung in spezialisierter Bildverarbeitungshardware, die beispielsweise in intelligenten Kameras (vgl. Abschnitt 2.3.3) zum Einsatz kommt, als auch der Cacheverwaltung moderner Prozessoren entgegen und bringen damit Vorteile für die Hard- und Softwareimplementierung. Wenig geeignet sind dagegen Algorithmen, die einen wahlfreien Zugriff auf verschiedene Bildbereiche, Rekursionen oder mehrfache Durchläufe (multi-pass) über das Bild erfordern.

Eine wichtige Technik zur Realisierung komplexer Punktoperatoren stellt die Vorabberechnung aller möglichen Ergebniswerte und ihre Speicherung in einer Lookuptabelle dar. Dadurch können auch umfangreichere Berechnungsschritte (wie z.B. die Gammakorrektur oder die Farbraumkonvertierungen aus Abschnitt 3.2.3) durch einen einfachen Tabellenzugriff ersetzt werden und damit für die Markerdetektion eingesetzt werden.

3.3.2 Ablauf der Einzelschritte der Markerdetektion

Abbildung 3.6 zeigt auf der linken Seite den Ablauf der Markerdetektion mit den typischen Verarbeitungsschritten digitaler Bildverarbeitung. Die Bildvorverarbeitung optimiert das erfasste Bild im Hinblick auf eine robuste und ressourceneffiziente Objekterfassung. Relevante Strukturen werden verstärkt und Störeinflüsse unterdrückt. Allen Bildvorverarbeitungsschritten gemein ist, dass sie aus Bilddaten wieder neue Bilddaten generieren. Die nachfolgenden Segmentierungsverfahren trennen schließlich die Markerstrukturen von der Bildumgebung und die Merkmalsextraktion erzeugt die nicht-bildhafte Repräsentation der Bildinhalte in Form einer Liste von Markerstrukturen mit zugehörigen Eigenschaften.

[8] Ein kleiner Zeilencache der letzten n-1 Zeilen ist für die Datenhaltung des nxn-Fensterbereichs bei den lokalen Operatoren notwendig.

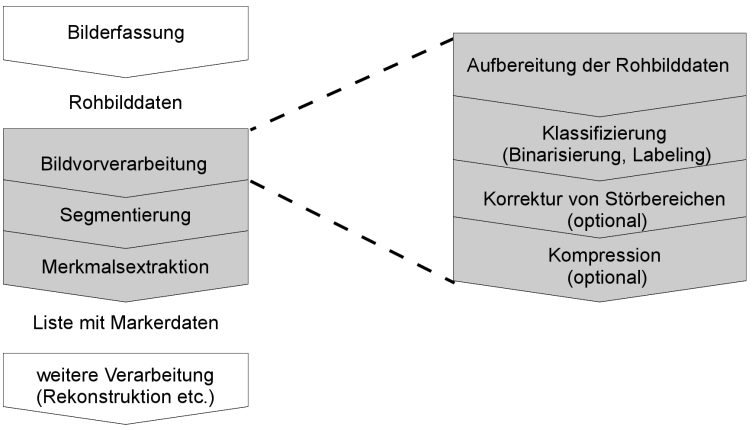


Abbildung 3.6: Unterteilung der Markerdetektionsphase in Bildvorverarbeitung, Segmentierung und Merkmalsextraktion. Die Bildvorverarbeiung unterteilt sich wiederum in weitere Einzelschritte (vgl. auch Abbildung 3.1).

Die Bildvorverarbeitung selbst unterteilt sich in weitere Einzelschritte zur Aufbereitung der Rohbilddaten, Klassifizierung in Vordergrund- und Hintergrundstrukturen und Unterdrückung von Störbereichen (Abbildung 3.6 rechts). Je nach Marker- bzw. Kamerasensortyp muss zwischen der Verarbeitung von Farb- und Graustufenbildern unterschieden werden. Zur Vereinfachung der Datenflusstruktur wurde darauf geachtet, dass beide Bildtypen einen prinzipiell gleichartigen Ablauf erfahren und sich nur in einzelnen Verarbeitungsschritten unterscheiden.

Den zentralen Schritt der Bildvorverarbeitung stellt die Differenzierung relevanter Markerstrukturen vom Bilduntergrund dar. Hierfür wird anhand der Pixelwerte eine Klassifizierung in Vorder- und Hintergrundstrukturen vorgenommen. Bei Graustufenbildern erfolgt diese Differenzierung über die Intensitätswerte, anhand derer die Pixel zwei Klassen H, V (Vordergrund und Hintergrund) zugeteilt werden.[9] Bei Farbbildern erfolgt die Klassifizierung über den Farbwert der Pixel. Dabei existieren jedoch mehrere Vordergrundfarbklassen $H, V_1, V_2, ..., V_n$, die den verschiedenen Markerfarbregionen entsprechen (vgl. die Beschreibung der Farbmarkerkonfiguration in Abschnitt 3.2.3). Der Klassifizierungsschritt stellt demnach bei Monochrombildern eine Binarisierung, bei Farbbildern ein Farblabeling dar. Zur einheitlichen Beschreibung der nachfolgenden Verarbeitungsschritte werden die relevanten Markerbereiche als Vordergrundregionen mit Vordergrundklasse V_k bezeichnet. Für die binarisierten

[9] Bei Markern mit innerer Struktur wird keine Pixelklassifizierung in Vordergrund und Hintergrund, sondern zu verschiedenen Regionen des Markers vorgenommen (z.B. weiße und schwarze Sektoren). Der prinzipielle Ablauf ändert sich dadurch jedoch nicht, so dass auch zur Bildvorverarbeitung solcher Markertypen die hier vorgestellten Methoden eingesetzt werden können.

Bilder entspricht dies der Klasse V der hellen Pixel, für das Farblabeling entsprechend den verschiedenen Farbklassen V_k aus der Definition der Markerfarbregionen.

Die übrigen Verarbeitungsschritte verlaufen beim Tracking von Farb- bzw. monochromatischen Markern größtenteils analog. Bei Farbsensoren mit Bayerfiltermatrix muss vor allen weiteren Schritten zusätzlich die Aufbereitung der Farbbilder aus den Rohbilddaten der Kameras erfolgen, das Demosaicing. Die weiteren Schritte von Störbereichsunterdrückung und Segmentierung & Merkmalsbestimmung sind bei der Farbmarkerdetektion jeweils auf die Verarbeitung mehrerer Vordergrundklassen $V_1, V_2, ..., V_n$ erweitert. Die Vorgehensweise entspricht einer kombinierten Ausführung der Verarbeitung von Monochrombildern parallel für alle Vordergrundfarbklassen $V = V_k$. Die aufgeführten Einzelschritte werden in den folgenden Abschnitten nun detaillierter beschrieben.

3.3.3 Aufbereitung der Rohbilddaten

Demosaicing-Filter

Der Demosaicing-Filter führt die durch die Bayerfiltermatrix (Abbildung 2.11) getrennt erfassten Spektralkanäle pixelweise wieder zu einem RGB-Wert zusammen. Aufgrund seiner Bedeutung bei der Generierung von Farbaufnahmen existiert eine Vielzahl von Algorithmen mit Unterschieden bei Qualität und Berechnungsaufwand [Holst und Lomheim 2007, Kap. 7.4.3] [Gunturk et al. 2005]. Bei den üblichen Einsatzbereichen des Farbmarkertrackings von MOSCOT ist eine lineare Interpolation innerhalb eines 3x3-Pixelfensters ausreichend. Abbildung 3.7 zeigt die vier möglichen Pixelanordnungen der Bayermaske für ein 3x3-Pixelfenster. Die jeweils fehlenden Spektralkanäle des zentralen Pixels werden nach folgender Vorschrift aus den umliegenden Pixeln ermittelt:

Berechnung der Rot- und Blauanteile für ein Grünpixel (Abb. 3.7a+b):

$$R = \frac{R_1 + R_2}{2} \qquad B = \frac{B_1 + B_2}{2} \tag{3.7}$$

Berechnung der Grün- und Blauanteile für ein Rotpixel (Abb. 3.7c):

$$G = \frac{G_1 + G_2 + G_3 + G_4}{4} \qquad B = \frac{B_1 + B_2 + B_3 + B_4}{4} \tag{3.8}$$

Berechnung der Rot- und Grünanteile für ein Blaupixel (Abb. 3.7d):

$$R = \frac{R_1 + R_2 + R_3 + R_4}{4} \qquad G = \frac{G_1 + G_2 + G_3 + G_4}{4} \tag{3.9}$$

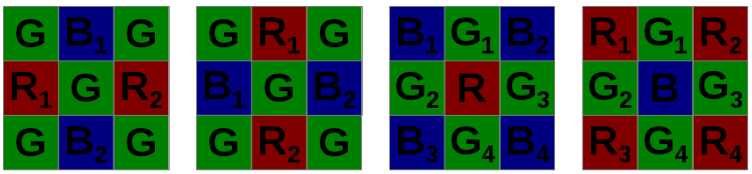


Abbildung 3.7: Die vier verschiedenen Anordnungen der Pixelumgebung beim Farbfilter nach Bayer.

Für höhere Anforderungen sind auch 5x5-Filter möglich, die zusätzlich die Kantenvorzugsrichtung berücksichtigen, und so das Auftreten von Farbsäumen einschränken [Gunturk et al. 2005; Luhmann 2010]. Für solche Artefakte anfällige, hochfrequente Schwarz-Weiss-Übergängen treten allerdings beim Farbmarkertracking in geschlossenen Simulatorsetups gewöhnlich nicht auf. In Versuchen zeigte sich das Farbmarkertracking mehr durch unzureichende Farbkonstanz und Fehlklassifizierungen limitiert als durch Artefaktbildung des Demosaicingfilters.

Gammakorrektur

Die Gammakorrektur dient der Anpassung der Helligkeitscodierung des Bildes an die nicht-lineare Helligkeitswahrnehmung des Menschen [Poynton 1998; Shapiro und Stockman 2001; Jähne 2005]. Als reine Skalierung der Intensitätswerte hat sie keine Auswirkung auf die folgenden Bildverarbeitungsoperationen. Für einen natürlichen Bildeindruck bei der manuellen Prozesskontrolle und Steuerung durch den Benutzer erweist sie sich allerdings als sinnvoll (z.B. bei der interaktiven Festlegung von Farbregionen, Abbildung 5.5) und ist daher auch in die Bildverarbeitungskette des Trackings integriert. Die Definition der Gammafunktion lautet:

$$GAMMA(I) = I_{max} \left(\frac{I}{I_{max}} \right)^{\gamma}, \tag{3.10}$$

wobei I_{max} die Obergrenze des Wertebereichs ist (255 bei 8bit Darstellung der Komponenten) und γ der Gammakorrekturfaktor. Ein gängiger Wert für γ bei der Wiedergabe auf Computermonitoren ist 2.2, so dass für einen korrekten visuellen Eindruck in der Kamera eine Gammakorrektur mit einem Wert von $\frac{1}{2.2} \approx 0.45$ erfolgen muss. Bei Farbbildern wird die Korrektur komponentenweise durchgeführt. Die Implementierung kann aus Effizienzgründen wieder in einer Lookup-Tabelle erfolgen.

Rauschunterdrückungsfilter

Rauschunterdrückungsfilter können in manchen Trackingsetups zum Einsatz kommen, bei denen aufgrund unzureichender Beleuchtungsbedingungen (insbesondere

beim Farbmarkertracking) Bildartefakte durch das Elektronenrauschen der Kamerasensoren entstehen. Faltungsoperatoren sind eines der Standardinstrumente, das die Computer Vision zur Rauschunterdrückung einsetzt. Im diskreten zweidimensionalen Bildraum ist die Faltung mit einem Filter h definiert als:

$$g'(x,y) = g(x,y) * h(x,y) = \frac{1}{f} \sum_{u=-k}^{k} \sum_{v=-l}^{l} g(x-u, y-v) \cdot h(u,v) \tag{3.11}$$

Dabei bezeichnet $h(u,v)$ das Filterkernel der Größe $(2k+1) \times (2l+1)$. Meist wird ein quadratischer Kernel eingesetzt, so dass $k = l$. Der Normierungsfaktor $1/f$ verhindert ein Überschreiten des zulässigen Wertebereichs.

Einen einfachen Rauschunterrückungsoperator stellt der Mittelwertfilter dar, welcher den Pixelwertdurchschnitt in einer 3x3-Umgebung berechnet und somit eine glättende Wirkung hat (Tiefpassfilter):

$$MEAN = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.12}$$

Häufig wird auch ein Filterkernel mit Koeffizienten vorgeschlagen [Shapiro und Stockman 2001; Jain et al. 1995], die an die zweidimensionale Gaussfunktion angelehnt sind, und eine effektivere Störungsunterdrückung erlauben:

$$GAUSS = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.13}$$

Bei allen Faltungsoperatoren kann durch Vergrößerung des Kernels auf 5x5 wieder eine verbesserte Filterfunktion zulasten des Rechenaufwandes erreicht werden.

Bei Versuchen zeigte sich, dass die in Abschnitt 3.3.5 Störbereichsunterdrückung mittel morphologischer Filter im Anschluss an die Pixelklassifikation effektiver ist als diese vorgelagerten Methoden der Rauschunterdrückung. Letztere werden hier daher nur als zusätzliche Option aufgeführt, in den aktuellen Anwendungen von MOSCOT aber nicht eingesetzt.

3.3.4 Pixelklassifizierung

Die Pixelklassifizierung ordnet jedem einzelnen Bildpixel anhand seines Pixelwerts ein Klassenlabel von Vordergrund- bzw. Hintergrundklassen zu: $I \rightarrow H, V$ bzw. $(R,G,B) \rightarrow H, V_1, V_2, ..., V_n$.

Für Graustufenbilder entspricht dies einer Binarisierung mit dem Schwellwert $I_{threshold}$ aus der Markerdefinition:

$$BIN(I) = \begin{cases} 1 & \text{wenn } I >= I_{threshold} \\ 0 & \text{sonst,} \end{cases} \tag{3.14}$$

für Farbbilder einem Farblabeling, wobei die Zuordnung der RGB-Werte zu Farbklassen anhand der bei der Markerkonfiguration festgelegten Definition erfolgt:

$$LABEL(R,G,B) = \begin{cases} k & \text{mit } k = 1, 2..., n \text{ für} (R,G,B) \in \text{Farbregion } V_1, V_2, .., V_n \\ 0 & \text{sonst.} \end{cases} \tag{3.15}$$

Ganz allgemein kann die Umsetzung der Pixelklassifizierung wie im Abschnitt zur Markerkonfiguration erläutert über eine Lookuptabelle LUT erfolgen:

$$k(x,y) = LUT(g(x,y)), \tag{3.16}$$

wobei $g(x,y)$ den Pixelwert und $k(x,y)$ das klassifizierte Pixel an der Stelle (x,y) bezeichnen. Die Binarisierung kann so als Spezialfall des Farblabelings mit nur einer Vordergrundklasse V behandelt werden. Ebenso werden in den folgenden Verarbeitungsschritte die morphologischen Operatoren und Segmentierungsoperationen von rein binärer Verarbeitung auf die Bearbeitung gelabelter Bilder mit mehreren Vordergrundklassen V_k erweitert.

3.3.5 Unterdrückung von Störbereichen durch morphologische Operatoren

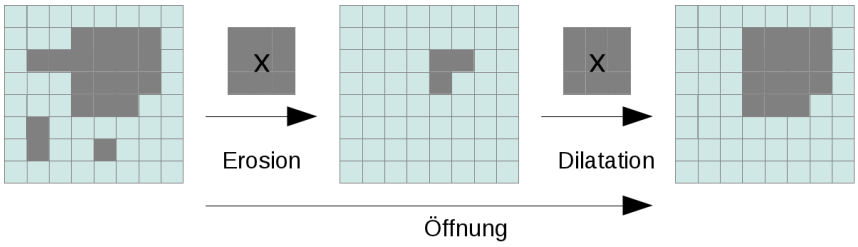


Abbildung 3.8: Morphologische Erosion, Dilatation und die morphologische Öffnung als Kombination von beiden.

Da auch fälschlicherweise als Vordergrund klassifizierte Störpixel bei der Segmentierung als potentielle Markerstrukturen betrachtet werden, kann eine effiziente Störpixelunterdrückung Ressourcen in den nachfolgenden Verarbeitungsstufen einsparen.

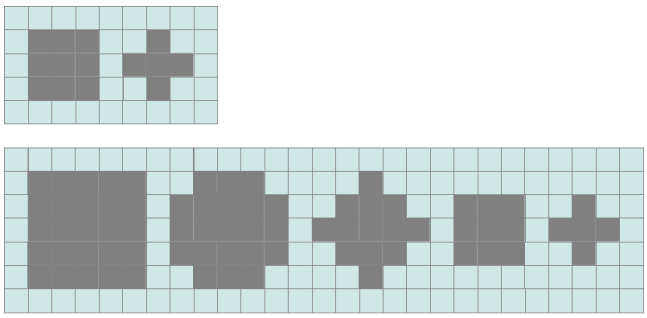


Abbildung 3.9: Die möglichen isotropen strukturierenden Elemente bei einem 3x3- bzw. 5x5-Kernel

Eine universell einsetzbare Methode zur Filterung von Bildfehlern in binarisierten Bildern sind morphologische Bildoperatoren [Soille 2003; Jähne 2005; Snyder und Qi 2010]. Charakteristisch für sie ist ein strukturierendes Element B (ein kleines Binärbild), das sequentiell über das Ursprungsbild A bewegt wird, wobei eine binäre Faltung zwischen dem strukturierendem Element und den Bildpixeln durchgeführt wird.

Die grundlegenden Operationen morphologischer Bildverarbeitung sind Erosion ε_B und Dilatation δ_B:

$$\varepsilon_B(A) = A \ominus B = \{z|B_z \subseteq A\} \delta_B(A) = A \oplus B = \{z|B_z \cap A \neq \emptyset\} \tag{3.17}$$

Die Erosion stellt die Menge aller Punkte z dar, für die das strukturierende Element B vollständig in der Menge der Vordergrundpixel A enthalten ist, wenn sich der Bezugspunkt des strukturierenden Elements B an der Stelle z befindet. Die Dilatation stellt die Menge aller Punkte z dar, an denen das strukturierende Element B die Menge A der Vordergrundpixel schneidet. Abbildung 3.8 veranschaulicht diese Sachverhalte.

Eine Erosion eignet sich damit sehr gut, um verbleibende Störpixel und -bereiche aus den binarisierten bzw. gelabelten Trackingbildern herauszufiltern. Gleichzeitig verdünnt die Erosion jedoch auch die relevanten Vordergrundstrukturen. Da für die Merkmalsextraktion eine hohe Pixelzahl der Markerstrukturen erwünscht ist, kann eine eine nachfolgende Dilatation diesen Nebeneffekt ausgleichen. Eine solche Kombination aus Erosion und Dilatation wird als morphologische Öffnung γ_B bezeichnet:

$$\gamma_B(A) = A \circ B = (A \ominus B) \oplus B \tag{3.18}$$

Die morphologische Öffnung kann damit dargestellt werden als die Vereinigung aller Verschiebungen des strukturierenden Elements B, bei denen es vollständig in die Menge der Vordergrundpixel hineinpasst:

$$\gamma_B(A) = A \circ B = \bigcup \{B | B \subseteq A\} \tag{3.19}$$

Mit der morphologischen Erosion und Öffnung stehen damit zwei Verfahren bereit, um kleine Störbereiche oder -pixel im Ursprungsbild zu unterdrücken. Als strukturierende Elemente (Strukturelement) kommen nur kleine isotrope Formen in Frage, da zum einen der Berechnungsaufwand niedrig zu halten ist, zum anderen die Schwerpunkte der Markerbilder erhalten bleiben müssen. Abbildung 3.9 zeigt oben die beiden möglichen Strukturelement bei einem 3x3-Kernel. Sie lassen sich als unterschiedliche „Stärken“ der Störungsunterdrückung ansehen, da ein größeres Strukturelement auch größere Störbereiche unterdrückt.[10] Je nach Anzahl und Auswahl der Markerfarben kann jedoch eine bessere Steuerung der Filterwirkung nützlich sein.

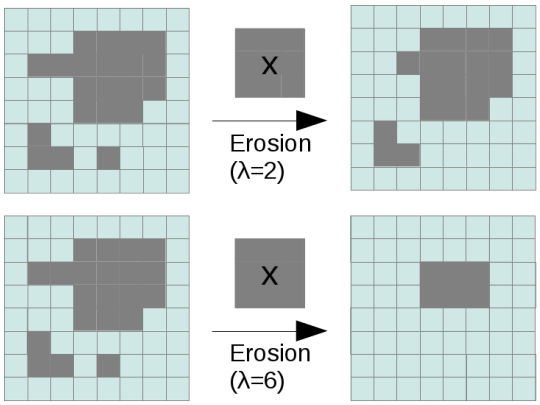


Abbildung 3.10: Parametrische Erosion als feinjustierbare Methode zur Störbereichsfilterung. Der Parameter λ gibt die notwendige Überdeckung des Strukturelements an (siehe Text)

Eine Lösung für eine solche feiner justierbare Störungsunterdrückung bietet sich mit der parametrische Erosion ε_λ [Soille 2003], die auch bei nur teilweiser Überdeckung des Strukturelements eine Filterfunktion aufweist. Das Beispiel in Abbildung 3.10 zeigt, dass hier im Unterschied zur gewöhnlichen Erosion das zentrale Pixel bereits bei einer Überdeckung von $k > \lambda$ Pixeln entfernt wird. Anschaulich entspricht dies beim isotropen Standardelement einer Filterung des zentralen Pixels bei weniger

[10] Bei einer Vergrößerung des Kernels auf 5x5 existieren mehr isotrope Strukturelemente, allerdings wächst auch der Ressourcenaufwand quadratisch mit der Kernelgröße.

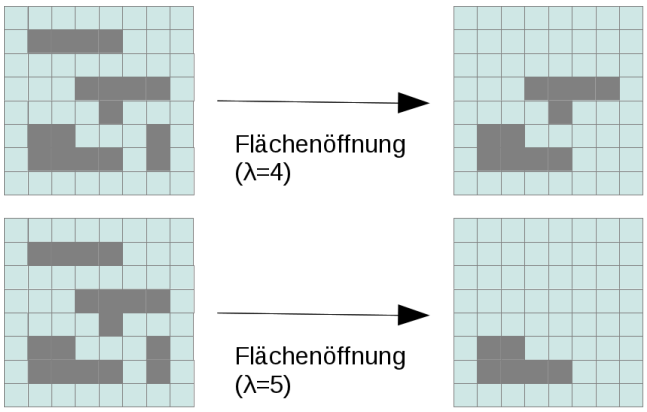


Abbildung 3.11: Flächenöffnung zur Filterung von beliebigen Störbereichen mit gegebener maximaler Größe λ

als λ Nachbarpixeln und erlaubt damit eine Einstellung der Filterstärke über acht Stufen. Das Verfahren kann mit wenig Ressourcenaufwand implementiert werden und verringert effektiv die Anzahl von Störbereichen für die nachfolgende Segmentierung:

$$\varepsilon_\lambda(x,y) = \begin{cases} k & \text{wenn } k(x,y) = k \wedge n > \lambda \\ 0 & \text{sonst,} \end{cases} \tag{3.20}$$

wobei n die Anzahl der jeweiligen Nachbarpixel von (x,y) mit Vordergrundklasse V_k bezeichnet.

Eine weitere Möglichkeit, auch größere Störbereiche mit wählbarer Pixelzahl zu entfernen, besteht in der algebraische Flächenöffnung γ_λ Vincent [1994]; Bovik [2009]:

$$\gamma_\lambda(A) = \bigvee_i \{\gamma_{B_i}(A) | \text{Fläche}(B_i) = \lambda\} \tag{3.21}$$

Sie entfernt zusammenhängende Vordergrundbereiche, deren Fläche kleiner gleich λ Pixeln ist und kann als die Vereinigung aller morphologischen Öffnungen mit einem Strukturelement B der Größe λ aufgefasst werden (Abbildung 3.11). Eine effiziente Implementierung ist über das im folgenden Abschnitt vorgestellte Connected-Component-Labeling möglich, wenn detektierte Strukturen mit einer Fläche $\leq \lambda$ verworfen werden.

Die letzten beiden Methoden werden bei MOSCOT der gewöhnlichen Erosion und Öffnung vorgezogen. Allgemein sind die morphologischen Filtermethoden insbesondere bei der Farbmarkerdetektion sinnvoll, da hier häufig Störbereiche die Farbklassifizierung überstehen. Im Falle von aktiver bzw. retroreflektierender IR-Signalisierung

mit anschließender Klassifizierung über einer Binarisierungsschwelle treten solche Artefakte nur selten auf.

3.3.6 Lauflängenkodierung

Zur Datenreduktion der vorverarbeiteten pixelklassifizierten Bilder kann eine Lauflängenkodierung eingesetzt werden. Sie bietet sich an, wenn bei der im folgenden Abschnitt vorgestellten Konnektivitätsanalyse ein lauflängenbasierter Algorithmus zum Einsatz kommen soll. Zusätzlich kann so eine deutliche Reduktion der zu übertragenden Datenmengen erreicht werden, falls die nachfolgende Segmentierung und Merkmalsextraktion in getrennten Verarbeitungsressourcen implementiert sind. (Solche aufgeteilten Datenflussarchitekturen werden in Kapitel 4 erläutert.)

Der Algorithmus tastet das Bild zeilenweise ab und ersetzt jeweils Sequenzen von n Pixeln der Vordergrundklasse k ab der Spalte x_{start} durch die Datenstruktur (y, x_{start}, n, k) [Jain et al. 1995; Steger et al. 2008]. Nach der Bildvorverarbeitung vorliegende gleichförmige Bereiche von Vordergrundpixeln sind mit dieser Methode sehr effizient zu kodieren. Die erreichbare Kompressionsrate der Lauflängenkodierung hängt allerdings sehr stark von der Anzahl der Vordergrundstrukturen und damit von der effizienten Unterdrückung von Störpixeln ab. Je nach Auflösung des Originalbildes, Anzahl der Vordergrundklassen, Anzahl der verbleibenden Vordergrundstrukturen kann bei vorverarbeiteten Trackingbildern ein Datenreduktionsfaktor von ca. 3–100 erreicht werden.

3.3.7 Segmentierung und Merkmalsextraktion

Segmentierung und Merkmalsextraktion sind neben der Pixelklassifizierung die essentiellen Arbeitsschritte der Markerdetektion. Sie identifizieren die Markerstrukturen innerhalb der als Vordergrund klassifizierten Pixel und bestimmen ihre Schwerpunktskoordinaten sowie zusätzliche Merkmale wie Fläche, maximale Ausdehnung und Codierungsinformationen (Abbildung 3.12). Während Segmentierung und Merkmalsextraktion beim klassischen Ablauf von Bildverarbeitungssystemen zwei getrennte Schritte darstellen, werden sei bei MOSCOT aus Effizienzgründen in einem gemeinsamen Arbeitsschritt zusammengefasst. Die zu bestimmenden Markermerkmale sind so gewählt, dass sie eine solche Zusammenführung der Arbeitsschritte ermöglichen. Nachfolgend wird zuerst die Merkmalsbestimmung einer Markerstruktur über Flächenmomente erläutert und danach deren Ermittlung während eines Durchlaufs über das vorverarbeitete Bild.

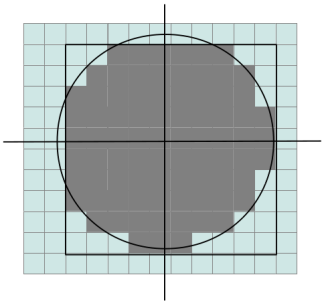


Abbildung 3.12: Fläche, (subpixelgenauer) Schwerpunkt und umschließendes Rechteck einer Markerregion.

Mit m_{pq} als Flächenmoment der Ordnung $(p+q)$ einer Region R

$$m_{pq} = \sum_{(x,y)\in R} x^p y^q \tag{3.22}$$

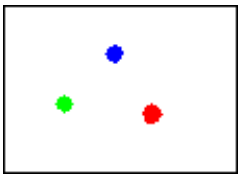
ergeben sich die folgenden Zusammenhänge für die Merkmale der Markerstrukturen:

$$\begin{aligned} AREA(R) &= m_{00} \\ COG(R) &= (\bar{x}, \bar{y}) = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \\ AABB(R) &= (\min_{(x,y)\in R} x, \max_{(x,y)\in R} x, \min_{(x,y)\in R} y, \max_{(x,y)\in R} y) \\ CODE(R) &= ID(V_k(R), (\bar{x}, \bar{y}) \in ROI_j(V_k)) \end{aligned} \tag{3.23}$$

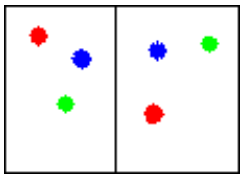
$AREA$ bezeichnet die Berechnungsvorschrift für die Fläche, COG für den Schwerpunkt (centre of gravity), $AAAB$ für das umschließende Rechteck (axis aligned bounding box) und $CODE$ für die Codierungsinformation des Markers. Im aktuellen Entwicklungsstand von MOSCOT wird die Codierungsinformation zur Markeridentifikation nur anhand der Farbklasse der Markerstruktur V_k und optional aus ihrer Bildregion $ROI_j(V_k)$ (siehe unten) bestimmt.

R bezeichnet eine ganze Klasse bzw. eine zusammenhängende Teilregion einer Klasse von Vordergrundpixeln. Für die Bestimmung der Objektregion R kommen je nach Komplexität des Trackingszenarios drei verschiedene Segmentierungsverfahren zum Einsatz (Abbildung 3.13):

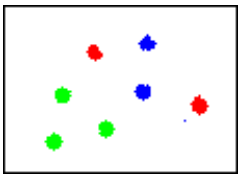
Globale klassenbezogene Merkmalsbestimmung Es werden alle Pixel einer Vordergrundklasse V_k im gesamten Bild betrachtet. Das Verfahren ist damit auf Bilder mit nur einem Marker pro Vordergrundklasse beschränkt, wie sie z.B. bei



(a) Globale klassenbezogene Merkmalsbestimmung, ein Marker pro Vordergrundklasse V_k



(b) Bildregionen- und klassenbezogene Merkmalsbestimmung, ein Marker pro Vordergrundklasse V_k und Bildregion ROI_j



(c) Lokale klassenbezogene Konnektivitätsanalyse und Merkmalsbestimmung, mehrere Marker der Vordergrundklasse V_k

Abbildung 3.13: Darstellung der drei verschiedenen Verfahren der Segmentierung und Merkmalsbestimmung, Erläuterung siehe Text.

Trackingsetups mit verschiedenfarbigen Farbmarkern bzw. sequentiell geschalteten aktiven Markern auftreten. Da alle Pixel $k(x, y) = k$ im gesamten Bild in die Berechnung der Flächenmerkmale einbezogen werden, ist das Verfahren sehr anfällig für Störpixel, die in den Filterstufen nicht komplett ausgefiltert wurden. Vorteilhaft ist die extrem ressourceneffiziente Implementierung, das Verfahren kann aber nur bei sehr stabilen Beleuchtungsverhältnissen zum Einsatz kommen.

$$R_{V_k} = \{(x, y)|k(x, y) = k\} \tag{3.24}$$

Bildregionen- und klassenbezogene Merkmalsbestimmung Obiges Verfahren kann auf mehrere Marker pro Farbe ausgedehnt werden, falls die Marker jeweils definierte Bildregionen (ROI, region of interest) nicht verlassen.[11] Den j verschiedenen Markern einer Vordergrundklasse werden dazu disjunkte rechteckförmige Bildregionoen zugewiesen, die den Bewegungsbereich des jeweiligen Markers abdecken:

$$ROI_j(V_k) = (x_{min}, y_{min}, x_{max}, y_{max}) \tag{3.25}$$

$$\text{mit } ROI_j(V_k) \cap ROI_i(V_k) = \emptyset \qquad \text{für } j \neq i$$

Die Merkmalsberechnung wird dann bezogen auf die jeweilige Bildregion $ROI_j(V_k)$ durchgeführt. Probleme mit Störpixeln bleiben analog zum ersten

[11] Dieser Spezialfall ist für Benutzerschnittstellen von medizinischen Simulatoren interessant, wo bei manchen Objekten ein eingeschränkter Bewegungsraum vorliegen kann (vgl. die Anwendung in Abschnitt 6.1.)

Verfahren vorhanden, so dass auch dieses Verfahren nur bei stabilen Beleuchtungsverhältnissen einsetzbar ist.

$$R_{V_k,ROI_j} = \{(x,y)|k(x,y) = k \wedge (x,y) \in ROI_j(V_k)\} \tag{3.26}$$

Lokale klassenbezogene Konnektivitätsanalyse und Merkmalsbestimmung Universeller einsetzbar, aber auch deutlich aufwendiger ist die Segmentierung mittels Konnektivitätsanalyse (Connected Component Labeling, CCL). Mit dieser können verschiedene Marker derselben Vordergrundklasse im Bild getrennt werden. Abgesetzte Störpixel werden als eigenständige Strukturen erkannt. Wegen der Relevanz für die robuste Segmentierung komplexerer Trackingsetups wird dieses Verfahren nachfolgend detailliert beschrieben.

Konnektivitätsanalyse

Zusammenhängende, gleichförmige Bereiche von Vordergrundpixeln werden als Blobs oder Connected Components bezeichnet. Die Konnektivitätsanalyse (Connected Component Labeling - CCL, auch Blobsegmentierung oder Blobextraktion [Jain et al. 1995; Shapiro und Stockman 2001]) identifiziert solche Bereiche im Eingangsbild und versieht sie im Ergebnisbild mit einem jeweils eindeutigen Labelwert (Abbildung 3.14). Aufgrund ihrer fundamentalen Bedeutung für die Segmentierung von Binärbildern existiert eine Vielzahl von Algorithmen zur Umsetzung der CCL (einen kurzen Überblick geben z.B. Suzuki et al. [2003] und He et al. [2009].) Bei einer pixelstrombasierten Implementierung wie bei MOSCOT, die ohne wahlfreien Zugriff auf die Bilddaten auskommen muss, können nur sequentielle Verfahren eingesetzt werden, die das Bild zeilenweise von oben nach unten durchlaufen. Für ein korrektes Labeling sind dann zumindest zwei Durchläufe erforderlich. Abbildung 3.15 illustriert dieses klassische Verfahren des Connected Component Labeling nach Rosenfeld und Pfaltz [1966]. Der erste Durchlauf kann lediglich ein vorläufiges gelabeltes Bild erzeugen, da z.B. bei U-förmigen Strukturen, Teilbereiche anfangs mit unterschiedlichen Labeln gekennzeichnet werden, die erst im weiteren Verlauf der Bildabtastung als zusammenhängend erkannt werden. Zusammengehörige Labelwerte der Teilbereiche werden in einer Äquivalenztabelle verzeichnet, um dann im zweiten Durchlauf durch einen eindeutigen endgültigen Labelwert ersetzt zu werden.

In der Markerdetektionsstufe eines optischen Trackingsystems kann jedoch auf die Ausgabe korrekt gelabelter Bilder verzichtet werden, da als Ergebnis lediglich die identifizierten Markerstrukturen mit ihren Merkmalen von Interesse sind. Damit wird eine Single-Pass-Implementierung möglich, falls die relevanten Bereichsmerkmale nach Gl. (3.23) bereits während des ersten Durchlaufs fortlaufend berechnet und bei der Vereinigung zweier Teilbereiche zusammengeführt werden können. Das

0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	2	0	2	0	0	3	0
0	1	1	1	0	2	2	2	0	0	0	0
0	0	1	1	0	0	0	0	0	4	4	0
0	0	0	0	0	0	4	4	4	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Abbildung 3.14: Ergebnis eines Connected Component Labeling. Die verschiedenen identifizierten Strukturen erhalten jeweils eindeutige Labelwerte zugewiesen.[12]

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	2	0	0	0	1	0
0	0	0	3	0	0	2	0	4	1	1	0
0	5	3	0	3	0	0	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(a) Vorläufig gelabelten Bildes nach dem ersten Durchlauf. Äquivalenzklassen sind 1,2,4 und 3,5

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	2	0	0	1	0	1	1	1	0
0	2	2	0	2	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(b) Endgültig gelabeltes Bild mit zusammengefassten Äquivalenzklassen

Abbildung 3.15: Darstellung der zwei Durchläufe des klassischen CCL-Algorithmus zur korrekten Erfassung U-förmiger Teilstrukturen.

ist sowohl für die Flächenmomente nullter und erster Ordnung als auch für das umschließende Rechteck möglich. Wie man sich leicht veranschaulicht, können die Labels in einer Pixelnachbarschaft zu maximal zwei verschiedenen Äquivalenzklassen gehören, so dass pro Pixelschritt maximal eine Vereinigung von Merkmalsinformationen durchzuführen ist. Bailey und Johnston [2007] und Johnston und Bailey [2008] beschreiben solche Single-Pass-Ansätze für die Konnektivitätsanalyse.

Algorithmus 3.1 gibt den Pseudocode für ein Connected-Component-Labeling in einem einzigen Durchlauf wieder. Für den zentralen Berechnungsschritt, die Erfassung und Auflösung der Labeläquivalenzen, werden in der Literatur verschiedene Verfahren vorgeschlagen (vgl. z.B. die Übersicht in [He et al. 2009]). In der Arbeitsgruppe wurden zwei Ansätze implementiert, die sich beide für eine Single-Pass-Anpassung eignen. Yang [2012] beschreibt eine Variante zum Einsatz in einer intelligenten Ka-

[12]Die Labels des CCL-Verfahrens sind von den bisher eingeführten Farblabels der Vordergrundpixelklassen zu unterscheiden. So können die Strukturen im Bild durchaus alle dieselbe, bei der Klassifizierung zugewiesene, Vordergrundpixelklasse V_k darstellen.

Algorithmus 3.1 Single-Pass Connected-Component Labeling Algorithmus

```
1:  for all pixels (x, y) do
2:      if k(x, y) = V_k then
3:          N ← GetPriorNeighborPixels(x, y)
4:          if IsEmpty(N) then                    ▷ no prior labels in neighborhood
5:              l(x, y) ← label++
6:              InitMarkerFeatures(x, y, l(x, y))
7:          else if SameLabel(N) then             ▷ just one label in neighborhood
8:              l(x, y) ← GetMinLabel(N)
9:              UpdateMarkerFeatures(x, y, l(x, y))
10:         else                                  ▷ different labels in neighborhood
11:             l(x, y) ← GetMinLabel(N)
12:             UpdateMarkerFeatures(x, y, l(x, y))
13:     ▷ register all labels in neighborhood as equivalent and merge region features
14:             for all X in labels(N) ∧ X ≠ l(x, y) do
15:                 RegisterEquivalentLabels(X, l(x, y))
16:                 MergeFeatures(X, l(x, y))
```

mera, bei der die Verwaltung der Äquivalenzklassen über eine Lookup-Tabelle und einen Parallel-Search-Multiple-Update-Algorithmus zum Vereinigen der Labelstrukturen realisiert wird (konzeptionell entspricht dieser Ansatz dem Verfahren mit Repräsentationslabeln und Äquivalenzmengen aus [He et al. 2007]). Vom Kooperationspartner VRmagic wurde ein Ansatz für lauflängenkodierte Bilder auf Basis des Union-Find-Algorithmus [Lapeyre und Strandh 2004] [Wu et al. 2009] implementiert (vgl. die Anwendung in Abschnitt 6.5).

3.3.8 Mögliche Erweiterungen der Markerdetektionsverfahren

Für erhöhte Genauigkeitsanforderungen können komplexere Verfahren zur subpixelgenauen Mittelpunktsbestimmung von kreis- bzw. ellipsenförmigen Markerbildern herangezogen werden (vgl. Abschnitt 2.2.4): Ellipsenoperator, Kleinste-Quadrate-Schätzung, Template-Matching [Luhmann 1995; Dold 1997; Shortis et al. 1994], [Steger et al. 2008, Kap. 3.11]. Einige dieser Optimierungsverfahren benötigen bereits eine Schätzung als Startwert für die Mittelpunktsbestimmung, so dass sie auf den Ergebnissen des Connected-Component-Labeling aufbauen können. Eine andere Möglichkeit zur Erhöhung der Genauigkeit bei monochromen Markern besteht in der Erweiterung der Schwerpunktsbestimmung auf grauwertbasierte Momentenbestimmung, die die Intensitätswert der Pixel miteinbezieht [Steger et al. 2008,

Kap. 3.5.2][Jähne 2005, Kap. 19.3]. Dies bietet sich insbesondere für retroreflektierende Kugelmarker oder aktive IR-LEDs an. Die Bestimmung zentraler (positionsinvariante) Momente zweiter Ordnung [Shapiro und Stockman 2001, Kap. 3.6], [Hu 1962] kann zusätzliche Informationen liefern über die Orientierung und Exzentrizität der Markerstrukturen und damit Hinweise für Korrespondenzanalyse und Güte der Markerdetektion geben. Sie erfordern aufgrund ihrer positionsinvarianten Berechnung jedoch einen zweiten Durchlauf über die Objektregion mit dem bereits ermittelten Schwerpunkt.

Bei sehr hoher Objektanzahl und damit häufiger Überlappung von Markerbildern derselben Farbklasse kann eine Detektion und Auflösung dieser Überlappungen sinnvoll sein. Für kreisförmige Marker kann die Identifikation z.B. über eine eine Hough-Transformation erfolgen [Ballard 1981]. Pintaric und Kaufmann [2007] beschreiben eine solche Implementierung für ein passives IR-Trackingsetup.

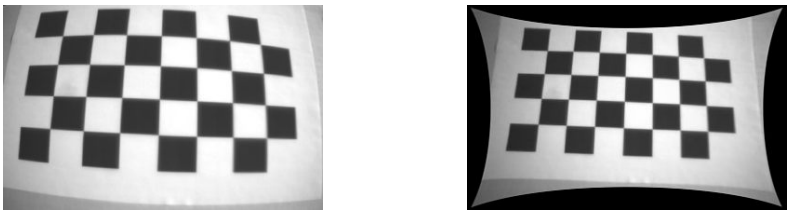
Weitere Erweiterungsmöglichkeiten der Markerdetektion bestehen hinsichtlich des Einsatzes von kantenbasierten Markern (vgl. wieder Abschnitt 2.2.4). Verfahren zu ihrer Detektion umfassen z.B. Kantenoperatoren, das KLT-Verfahren und Template Matching [Lucas, Kanade et al. 1981; Shi und Tomasi 1994; Kim et al. 1999; Wiora et al. 2004]

Markerdetektionsverfahren mittels Ellipsen- bzw. Kantenoperatoren und ihre Implementierung im Rahmen der MOSCOT-Trackingarchitektur finden sich bei Handel [2009].

3.4 Markerrekonstruktion

Die Markerrekonstruktion bestimmt aus der Liste der detektierten 2D-Markerpositionen aller Kameras die rekonstruierten 3D-Positionen der Marker im Trackingvolumen. Hierfür greift sie auf die während des Kamerakalibrierungsprozesses bestimmten Parameter der inneren und äußeren Kameraorientierung sowie die Verzeichnungskoeffizienten des Linsenmodells zurück. Zusätzliche extrahierte Markermerkmale können bei redundanten Daten zur Bewertung der 2D-Markerdatensätze herangezogen werden (beispielsweise können detektierte Markerstrukturen mit zu kleiner Fläche vor der Rekonstruktion ausgefiltert werden).

Die Rekonstruktionsverfahren in dieser Arbeit konzentrieren sich auf das Tracking einer kleinen Anzahl von Objekten wie z.B. Instrumente und Patientenphantom bei einem medizinischen Simulator. Die Objekte werden erfasst durch starre Anordnungen von wenigen Markern (3–5 Marker für 6 Freiheitsgrade mit Redundanz, bei Sonderfällen mit eingeschränkten Freiheitsgraden auch weniger). Zusätzlich wird beim



(a) Originalbild (b) Entzerrtes Bild

Abbildung 3.16: Starke radiale Linsenverzeichnung bei einem einfachen Kameraobjektiv

Farbmarkertracking die Farbcodierung zur Identifikation einzelner Marker und Objekte herangezogen. Aufwendigere Verfahren für eine Erweiterung des Systems zur Unterstützung größerer Anzahlen von gleichartigen Markern und Objekten werden wieder kurz am Ende des Abschnitts in 3.4.6 erörtert.

Während die Markerdetektion lediglich die Bilddaten einer Kamera berücksichtigt, und damit auch lokal in einer intelligenten Kamera ablaufen kann, ist es für die Markerrekonstruktion notwendig, die Daten aller Kameras in einem vorgeschalteten Datenfusionsschritt zusammenzuführen. Dabei werden die extrahierten Markerdatensätze aller Kameras in einer Datenstruktur zusammengefasst und für die anschließenden Rekonstruktionsschritte bereitgestellt.

Die 3D-Rekonstruktion der Markerpositionen unterteilt sich in drei Phasen:

- Entzerrung der 2D-Markerpositionen bei Einsatz eines erweiterten Kameramodells mit Modellierung der Linsenverzeichung.[13]
- Lösung des Korrespondenzproblems, der gegenseitigen Zuordnung der Markerbilder in den verschiedenen Kameraansichten.
- 3D-Rekonstruktion der Markerpositionen bei bekannter Kameraorientierung über Triangulation.

3.4.1 Korrektur der Linsenverzeichnung

Für Anwendungen mit höheren Anforderungen an die Genauigkeit ist die Linsenverzeichung bei der Rekonstrukion der Markerpositionen zu berücksichtigen. Ebenso

[13]Die Entzerrung der Bildkoordinaten verläuft für die Datensätze jeder Kamera unabhängig vom übrigen System, so dass sie auch als abschließender Verarbeitungsschritt der Markerdetektionsphase zugeordnet werden könnte. Da die Verzeichnungskoeffizienten jedoch Teil der erweiterten Kameraparameter sind, wird in der datenflusszentrischen Architektur von MOSCOT die logische Anordnung zu Beginn der Markerrekonstruktion bevorzugt.

gilt dies beim Einsatz von einfachen, kostengünstigen Objektiven, die konstruktionsbedingt eine stärkere Verzeichnung aufweisen als qualitativ hochwertigere Exemplare (Abbildung 3.16).[14] Zur Entzerrung wird die Verzeichnung nach Gl. (3.4) geschätzt und die detektierten Bildkoordinaten der Markerschwerpunkte damit korrigiert (siehe auch Anhang A.4):

$$\mathbf{x}_u = \mathbf{x}_d - \Delta\mathbf{x}_{rad}(\mathbf{x}_u) - \Delta\mathbf{x}_{tan}(\mathbf{x}_u) \tag{3.27}$$

$\mathbf{x}_d = (x_d, y_d)^T$ bezeichnet wieder die verzeichneten (gemessenen) Bildkoordinaten der Marker, $\mathbf{x}_u = (x_u, y_u)^T$ die unverzeichneten (hier korrigierten) Bildkoordinaten. $\Delta\mathbf{x}_{rad}(\mathbf{x}_u)$ bzw. $\Delta\mathbf{x}_{tan}(\mathbf{x}_u)$ modellieren die Verzeichnung der Linse am Punkt $\mathbf{x}_u$ in radialer und tangentialer Richtung nach Gl. (A.19) und (A.21). Die Berechnung der Korrekturwerte $\Delta\mathbf{x}_{rad}$ und $\Delta\mathbf{x}_{tan}$ basiert auf den unverzeichneten Bildkoordinaten $\mathbf{x}_u$, welche jedoch nicht bekannt sind. Als Näherung werden die Korrekturwerte an den unkorrigierten Koordinaten $\mathbf{x}_d$ berechnet. Bei sehr hohen Genauigkeitsanforderungen (insbesondere bei großen Korrekturkoeffizienten) besteht die Möglichkeit verbesserte Schätzwerte für $\mathbf{x}_u$ iterativ über eine anschließende nicht-lineare Optimierung zu bestimmen [Luhmann 2010, Kap. 3.3.3]:

$$\underset{\mathbf{x}_u}{\operatorname{argmin}} \quad \|\mathbf{x}_d - (\mathbf{x}_u + \Delta\mathbf{x}_{rad}(\mathbf{x}_u) + \Delta\mathbf{x}_{tan}(\mathbf{x}_u))\|^2 \tag{3.28}$$

3.4.2 Korrespondenzanalyse zur Markerzuordnung

Vor der Ausführung des eigentlichen Rekonstruktionsschrittes ist eine gegenseitige Zuordnung der Markerbilder in verschiedenen Kameraansichten der Szene notwendig, die Lösung des sogenannten Korrespondenzproblems. Es gehört in die Klasse der schlecht gestellten Probleme (*ill-posed problems*), d.h. es kann nicht für alle Eingangsdaten sichergestellt werden, dass eine eindeutige Lösung existiert [Poggio und Koch 1985; Faugeras 1993]. Dadurch ist bisher auch kein allgemeines, robustes und universell einsetzbares Verfahren zur Korrespondenzbestimmung bekannt. Im Folgenden werden Ansätze beschrieben, die beim Vorhandensein gewisser Randbedingungen eine Lösung ermöglichen.

Prinzipiell sind es die folgenden zwei Sonderfälle, die eine eindeutige Lösung erschweren oder verhindern:

[14] Häufig ist es bereits ausreichend, die dominierende radialsymmetrische Verzeichnungskomponente zu berücksichtigen.

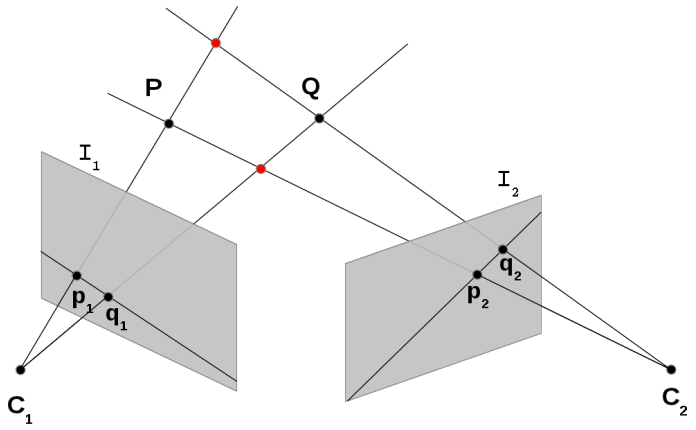


Abbildung 3.17: Korrespondenzproblem bei der Stereorekonstruktion: Bei falscher Zuordnung der Bildpunkte p und q entstehen die fehlerhaften Rekonstruktionen von Weltpunkten an den rot markierten Positionen.

- zum Marker in einem Kamerabild existieren mehrere Kandidaten. Bei falscher Zuordnung der Markerbilder werden fehlerhafte Weltpositionen rekonstruiert (Abbildung 3.17 veranschaulicht dieses Problem).
- zum Marker in einem Kamerabild existiert kein Kandidat in anderen Kameraansichten (durch Verdeckung, oder einen unterschiedlichen Bildausschnitt)

Damit kommt der Identifizierbarkeit von Markern eine Schlüsselrolle zu.

Eindeutige Markerkorrespondenzen

Die Bestimmung korrespondierender Marker ist trivial bei eindeutiger Identifizierbarkeit der Marker in den Kamerabildern. Im Falle des optischen Trackings ist dies gegeben bei:

- aktiven Markern mit eindeutiger Zuordnung durch sequentielle Aktivierung
- Markern mit eindeutiger Codierung (z.B. Passmarken mit Codierung)
- Farbmarkern bei Einsatz von global nur einem Marker je Farbe (Farbcodierung)
- Restriktion gleichartiger Marker auf jeweils eindeutige disjunkte 2D-Bildregionen (vgl. das Markermerkmal CODE aus Abschnitt 3.3.7)

Diese direkt bestimmbaren Markerkorrespondenzen können aus der Menge der zuzuordnenden Kandidaten entfernt und für die Rekonstruktion vorgemerkt werden.

Auch Fälle von fehlenden Korrespondenzen durch Verdeckungen oder unterschiedliche Bildausschnitte sind dabei direkt detektierbar und bei vorhandener Redundanz in den Datensätzen weiterer Kameras rekonstruierbar. Verbleibende Markerkorrespondenzen müssen mit den nachfolgend beschriebenen Verfahren ermittelt werden.

Korrespondenzanalyse bei nicht eindeutiger Markerzuordnung

Da der kombinatorische Suchraum möglicher Markerbildzuordnungen mit steigender Zahl an Markern sehr schnell anwächst ($\mathcal{O}(n!)$ bei n Markern im Stereofall), besteht eine prinzipielle Strategie darin, die Anzahl der in Frage kommenden Korrespondenzpartner zu beschränken. Dabei gilt es, sich die speziellen Gegebenheiten bei markerbasiertem optischen Tracking zu Nutze zu machen, so die bekannte Kalibrierung der Kamerasysteme, die Codierung von Einzelmarkern oder Markergruppen und schließlich die bekannte Markeranordnung bei starren Objekten. Abbildung 3.18 gibt einen Überblick der Vorgehensweise, die in den nächsten Abschnitten genauer beschrieben wird.

Bei Trackingsetups mit geeigneter Markercodierung besteht eine einfache Möglichkeit zur Komplexitätsreduktion in der Einteilung der Marker in Gruppen, die auf eindeutigen Merkmalen beruhen (analog zu den aufgeführten Kriterien für eindeutige Identifikation im letzten Abschnitt). Als Beispiel sei hier ein System mit jeweils mehreren Markern zweier verschiedener Farbklassen genannt. Es ist klar, dass dabei nur Markerbilder innerhalb derselben Farbgruppe als Kandidaten für gegenseitige Zuordnung berücksichtigt werden müssen.

Eine weitere entscheidende Vereinfachung der Korrespondenzanalyse kann bei optischen Trackingsystemen mit vorkalibrierten Kameras durch die bekannte Geometrie der Stereobildanordnung erreicht werden, die sogenannte Epipolargeometrie (siehe auch Anhang A.5). Damit wird der Suchbereich für Markerkandidaten vom Gesamtbild auf einen kleinen Toleranzbereich entlang einer Linie eingeschränkt. Details dieser Vorgehensweise werden im folgenden Abschnitt 3.4.3 beschrieben.

Mittels Gruppencodierung und Epipolargeometrie ist für die meisten Anwendungsszenarien des MOSCOT-Systems bereits eine ausreichende Einschränkung des Lösungsraums für die Markerzuordnung erreichbar. Beim Tracking einer kleinen Anzahl bekannter Objekt mit starrer Markerandordnungen ist nämlich eine vollständige Lösung des Korrespondenzproblems nicht zwingend notwendig. Vielmehr können die nach Anwendung des Epipolarkriteriums innerhalb einer Gruppe verbleibenden wenigen Mehrdeutigkeiten toleriert, und alle noch kombinatorisch möglichen Markerpaarungen einander zugeordnet werden. Bei der Rekonstruktion entstehen damit durch die fehlerhaften Zuordnungen zusätzliche Marker (im Folgenden als *ghost marker*

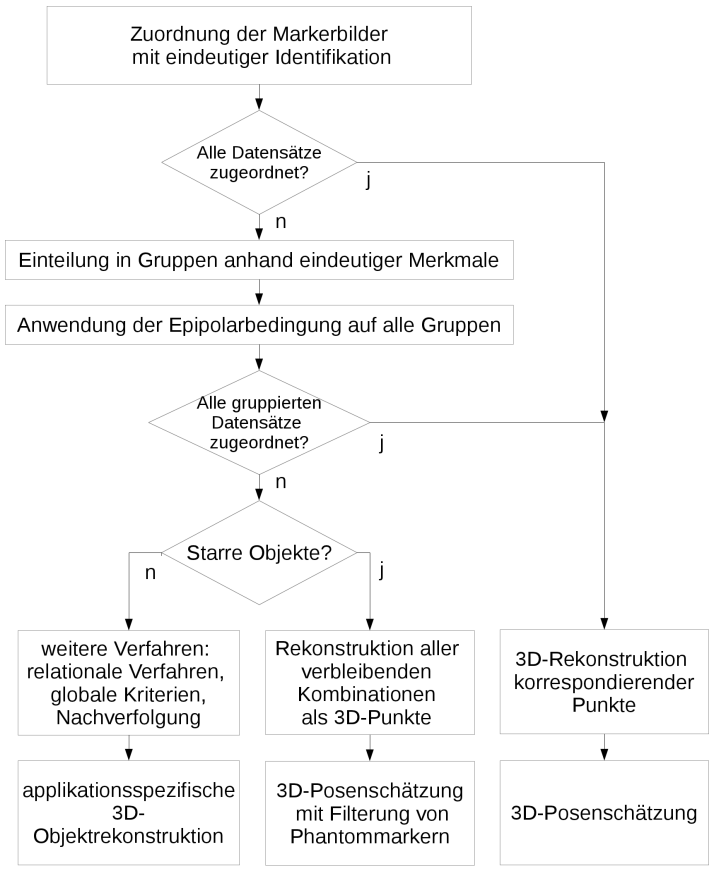


Abbildung 3.18: Ablaufdiagramm zur Korrespondenzanalyse

oder Phantommarker bezeichnet). Während der anschließenden Objektrekonstruktion werden die bekannten starren Markerkonfigurationen der Trackingobjekte in der rekonstruierten Punktemenge identifiziert, wobei fehlerhaft rekonstruierte, zusätzliche Marker herausfallen. Wie man sich leicht veranschaulicht, wächst bereits im Stereofall die Zahl möglicher Konstellationen solcher Phantommarker faktoriell mit der Zahl nicht eindeutig zuzuordnender Markerbilder, so dass diese Herangehensweise auf typische Trackingsetups mit einer geringen Zahl von Markern und wenigen Kameraansichten beschränkt ist. Abschnitt 3.5.1 zur Rekonstruktion starrer Objekte beschreibt diese Vorgehensweise.

Beim Tracking von Einzelmarkern bzw. deformierbaren Objekten kann diese Strategie aufgrund der fehlenden starren Markeranordnung nicht zum Einsatz kommen. Abschnitt 3.4.4 beschreibt einige komplexere Korrespondenzanalyseverfahren und weitere Ansätze zur Handhabung solcher Fälle.

3.4.3 Reduktion von Mehrdeutigkeiten mittels Epipolargeometrie

Zwei Kameras

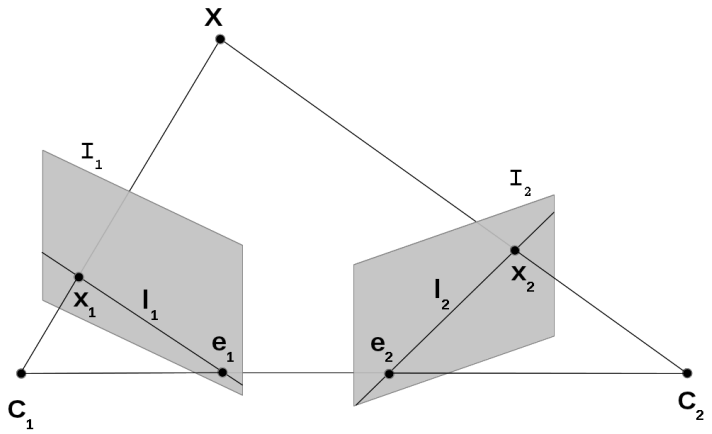


Abbildung 3.19: Epipolargeometrie einer Stereoaufnahme.

Die Epipolargeometrie [Longuet-Higgins 1981; Faugeras 1993; Hartley und Zisserman 2000] beschreibt die geometrischen Gegebenheiten bei der Abbildung eines Punktes $\mathbf{X}$ mittels zweier Kameras (Abbildung 3.19). Bei bekannter Kamerakalibrierung kann aus den Projektionsmatrizen $\mathbf{P}_1, \mathbf{P}_2$ die Fundamentalmatrix $\mathbf{F}$ errechnet werden. Sie liefert über die Gleichung

$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0 \tag{3.29}$$

die mathematische Beschreibung der Beziehung zwischen den Bildpunkten $\mathbf{x}_1$ und $\mathbf{x}_2$. Geometrisch veranschaulicht spannt der Weltpunkt $\mathbf{X}$ zusammen mit den Kamerazentren $\mathbf{C}_1$ und $\mathbf{C}_2$ die Epipolarebene auf. Deren Schnittlinien mit den Bildebenen der Kameras bilden die Epipolarlinien $\mathbf{l}_1$ und $\mathbf{l}_2$. Damit beschränkt sich die Suche nach Korrespondenzkandidaten für $\mathbf{x}_1$ vom gesamten Bildbereich I_2 auf die zugehörige Epipolarlinie $\mathbf{l}_2 = \mathbf{F}\mathbf{x}_1$ im Bild I_2 (bzw. einen schmalen Toleranzbereich um $\mathbf{l}_2$ zur Kompensation von Ungenauigkeiten der Bildaufnahme und Kamerakalibrierung). Anhang A.5 geht ausführlicher auf Epipolargeometrie und Fundamentalmatrix ein.

Durch die Anwendung des Epipolarkriteriums wird eine deutliche Reduktion der Zahl möglicher Korrespondenzkandidaten erreicht. Eine Abschätzung für die Anzahl verbleibender Mehrdeutigkeiten gibt Maas [1992, Gl. 6.5]:

$$N_a = (n^2 - n)\frac{2c\epsilon b_{12}(Z_{max} - Z_{min})}{F Z_{max} Z_{min}} \tag{3.30}$$

Dabei bezeichnet n die Anzahl der Marker, c die Kamerakonstante (entspricht den Brennweiten f_x bzw. f_y des in dieser Arbeit eingesetzten Kameramodells), ϵ die zulässige Toleranz um die Epipolarlinie, b_{12} die Basislänge des Kamerapaars, F die Bildfläche sowie Z_{max} und Z_{min} den Tiefenbereich der Objektausdehnung. Mit dieser Abschätzung verringert sich also die Komplexität der Korrespondenzsuche durch die Anwendung der Epipolarbedingung bei Stereokameras von $\mathcal{O}(n!)$ auf $\mathcal{O}(n^2)$.

Drei und mehr Kameras

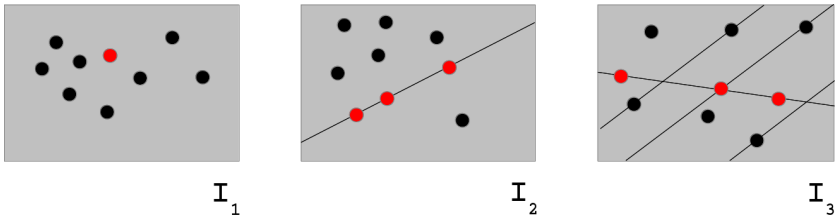


Abbildung 3.20: Trifokales Setup mit Kandidatenverifizierung über Epipolarlinienschnitt

Durch die Hinzunahme weiterer Kameras können verbleibende Mehrdeutigkeiten aufgelöst werden. Abbildung 3.20 illustriert eine solche Erweiterung der Epipolarbedingung auf drei Kameras. Es werden mögliche Korrespondenzpartner zum roten Marker im Bild I_1 gesucht, auf der zugehörigen Epipolarlinie im Bild I_2 stehen aber drei Kandidaten zur Wahl. Bei Hinzunahme der dritten Ansicht I_3 können dort die Epipolarlinien des Markers aus I_1 und der drei Kandidaten aus I_2 eingezeichnet werden. Die Suche nach Korrespondenzkandidaten beschränkt sich nun auf die Schnittpunkte dieser Epipolarlinien (bzw. einen kleinen Tolerenzbereich um diese herum). Im Beispiel befindet sich nur am mittleren Schnittpunkt in I_3 ein Markerbild, so dass der zugehörige mittlere rote Kandidat aus Ansicht I_2 der gesuchte Korrespondenzpartner ist. Algorithmus 3.2 beschreibt die Vorgehensweise [Maas 1992; Luhmann 2010].

Algorithmus 3.2 Epipolarlinienschnittverfahren für 3 Kameras

```
1: for all p_i in I1 do
2:     e12 ← CalcEpipolarLine(I1, I2, p_i)
3:     e13 ← CalcEpipolarLine(I1, I3, p_i)
4:     for all p_j in I2 do
5:         if Dist(p_j, e12) ≤ ε then                          ▷ p_j on e12
6:             e23 ← CalcEpipolarLine(I2, I3, p_j)
7:             for all p_k in I3 do
8:                 if Dist(p_k, e13) ≤ ε ∧ Dist(p_k, e23) ≤ ε then   ▷ p_k on e13 ∩ e23
9:                     CandidateList.add(p_i, p_j, p_k)          ▷ add tripel to candidates
```

Auch beim trinokularen Setup kann es noch zu Mehrdeutigkeiten in der Markerzuordnung kommen, insbesondere bei der Zuordnung einer großen Anzahl gleichartiger Marker. Maas [1992, Gl. 6.17] gibt für nicht kollineare Dreikamerasysteme eine Abschätzung der erwarteten Mehrdeutigkeiten an:[15]

$$N_a = \frac{4(n^2 - n)\varepsilon^2}{F sin\alpha}(1 + \frac{b_{12}}{b_{13}} + \frac{b_{12}}{b_{23}}) \tag{3.31}$$

Die Variablenbezeichungen entsprechen dem Fall mit zwei Kameras aus Gl. (3.30), wobei α den Schnittwinkel der Epipolarlinien beschreibt. Eine optimale Anordnung für drei Kameras entsteht demnach bei $\alpha = 60°$und $b_{12} = b_{23} = b_{13}$, einer Anordnung der Kameras als gleichseitiges Dreieck (vgl. auch die Anwendung in Abschnitt 6.2). Eine weitere deutliche Reduktion noch verbleibender Mehrdeutigkeiten ergibt sich bei zunehmender Kameraanzahl, wobei allerdings der Rechenzeitbedarf durch die Kombination aller möglichen Epipolarlinienschnitte sehr stark ansteigt.

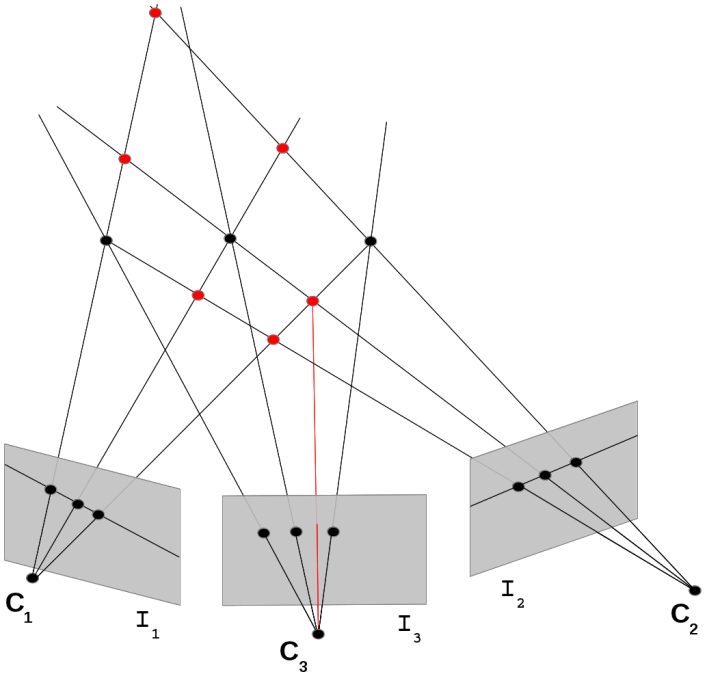


Abbildung 3.21: Trifokales Setup mit Rückprojektion der Rekonstruktionskandidaten

Ein gleichwertiger Ansatz zum Epipolarlinienschnitt bei mehreren Kameras besteht in der spekulativen Stereorekonstruktion mit anschließender Rückprojektion in eine

[15]Kollineare Systeme mit den drei Kamerazentren auf einer Linie ergeben etwas ungünstigere Geometrien und damit eine etwas höhere Zahl an Mehrdeutigkeiten.

dritte Kameraansicht [Forsyth und Ponce 2002]. Abbildung 3.21 zeigt die Vorgehensweise bei diesem Rückprojektionsverfahren. Die Zuordnung der drei Marker in den Kameraansichten I_1 und I_2 kann trotz Epipolarbedingung nicht aufgelöst werden, da alle in derselben Epipolarebene liegen. Neben den drei tatsächlichen Markern (schwarz) sind daher auch Fehlrekonstruktionen an den roten Positionen möglich. Eine Rückprojektion dieser Punkte in die Kameraansicht I_3 (am Beispiel des roten Phantommarkers rechts unten angedeutet) zeigt, dass am Schnittpunkt mit der Bildebene für die fehlzugeordneten roten Marker kein Markerbild zu finden ist und sie daher verworfen werden können. Nur die korrekt zugeordneten (schwarzen) Marker können auch im dritten Kamerabild an ihren Markerbildern verifziert werden. Dieser Rückprojektionsansatz kann auch direkt auf Systeme mit mehr als drei Kameras erweitert werden. Luhmann [2010, Kap. 5.5.3.3] beschreibt hierzu den Algorithmus 3.3.

Algorithmus 3.3 Epipolarbedingung mit Rückprojektion

```
1:  for all p_i in I_a do
2:      TempList.clear()
3:      for all I_b with b > a do
4:          e_ab ← CalcEpipolarLine(I_a, I_b, p_i)
5:          for all p_j in I_b do
6:              if Dist(p_j, e_ab) ≤ ε then                    ▷ p_j on e_ab
7:                  TempList.add(b, j)                  ▷ add candidate j in image b
8:                  Counter_bj ← 0                      ▷ init counter for candidate bj
9:                  for all I_c with c > a + b do
10:                     p_ijc ← ReconstructAndReproject(p_i, I_a, p_j, I_b, I_c)
11:                     for all p_k in I_c do
12:                         if Dist(p_k, p_ijc) ≤ ε then    ▷ ∃p_k at reprojected p_ij in I_c
13:                             Counter_bj ++   ▷ count verified point for candidate bj
14:     Max ← 0
15:     for all (b, j) ∈ TempList do          ▷ search candidate with highest count
16:         if Counter_bj > Max then
17:             Max ← Counter_bj
18:             Candidate ← (p_j in I_b)                   ▷ select (b,j) as candidate
19:     CandidateList.add(p_i, Candidate)                          ▷ add candidate
```

3.4.4 Weitere Verfahren zur Korrespondenzanalyse

Neben den geometrischen Randbedingungen durch die Epipolargeometrie können weitere Lösungsansätze für spezielle Konfigurationen des Korrespondenzproblems eingesetzt werden. Einführungen zu Verfahren der Korrespondenzanalyse in der Computer Vision finden sich bspw. in [Faugeras 1993; Lane und Thacker 1998; Luhmann 2010]. Die Verfahren unterscheiden sich zum einen nach der Anzahl der betrachteten Korrespondenzpunkte in dicht (*dense correspondence problem*) bzw. schwach besetzte Felder von Korrespondenzkandidaten (*sparse correspondence problem*). Zum anderen werden nach der Art der Korrespondenzbestimmung drei Gruppen von Verfahren unterschieden (vgl. [Maas 1997; Faugeras 1993; Scharstein und Szeliski 2002]):

- **Bild- bzw. flächenbasierte Verfahren** stützen sich auf die photometrische Auswertung von Pixelumgebungen und vergleichen deren Ähnlichkeit anhand verschiedener Kostenfunktionen. Sie betrachten gewöhnlich jedes, oder fast jedes Bildpixel bei der Bestimmung der Korrespondenzkandidaten.
- **Merkmalsbasierte Verfahren** extrahieren geeignete Merkmalsregionen (z.B. Kanten, Ecken, etc.) mittels spezieller Interestoperatoren und ermitteln deren Ähnlichkeit anhand von Gütekriterien, die sich auf die extrahierten Merkmale stützen. Dies führt gewöhnlich zu einer beschränkten Anzahl an Korrespondenzkandidaten.
- **Relationale Verfahren** versuchen eine Zuordnung durch die Betrachtung der relativen Lage von Bildmerkmalen zueinander. Auch hier wird nur mit einer beschränkten Anzahl an Korrespondenzkandidaten gearbeitet.

Die meisten in der Literatur aufgeführten Verfahren befassen sich mit Lösungsansätzen zur dicht besetzten bildbasierten Korrespondenzbestimung (vgl. die Übersichten in [Scharstein und Szeliski 2002] und [Seitz et al. 2006]). Im Falle der vorverarbeitenen Daten des MOSCOT-Datenflusses stehen nach der Markerdetektion allerdings keine Bildinformation mehr zur Verfügung, sondern lediglich Bildmerkmale in Form der detektierten Markerdatensätze. Ein Einsatz flächenbasierter Verfahren scheidet somit aus, wäre aber auch aufgrund der gleichartigen Marker und des homogenen Hintergrundes bei den meisten optischen Trackingszenarien wenig erfolgversprechend. Der Einsatz merkmalsbasierter Verfahren muss sich durch die gleichartigen Strukturen bei markerbasiertem Tracking auf relevante Merkmale wie Farbe oder Codierung stützen. Andere Merkmale wie Fläche und Ausdehnung, aber auch die Flächenmomente höherer Ordnung weisen nur eine geringe Korrelation bei korrespondierenden Bildern von homogenen Punktstrukturen auf (vgl. [Maas 1992, Kap. 6.2.2]). Codierung und Farbe wurden jedoch bereits im ersten Schritt bei der Gruppenzuordnung berücksichtigt. Einzig die Distanz zur Epipolarlinie kann bei mehreren Markerkandidaten als sinnvolle Kostenfunktion herangezogen werden [Maas 1992; Pintaric und

Kaufmann 2007]. Damit verbleiben im wesentlichen relationale Verfahren, die auf der relativen Lage ansonsten identischer Bildmerkmale aufbauen.

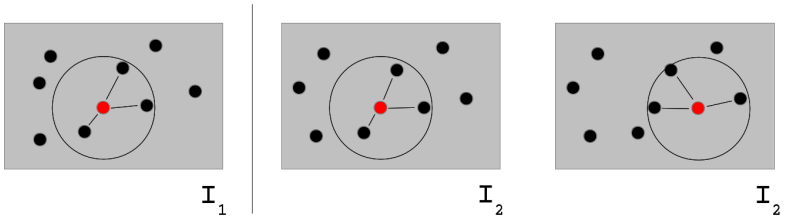


Abbildung 3.22: Beispiel für das relationale Zuordnungsverfahren nach Zhang et al. [1995]. Mittig ein Zuordnungskandidat mit hoher Ähnlichkeit zum Kandidaten im linken Kamerabild, rechts ein Zuordnungskandidat mit niedriger Ähnlichkeit,

Relationale Verfahren

Eine Einführung zu relationalen Verfahren geben z.B. Faugeras [1993]; Shapiro und Stockman [2001]. Im Folgenden soll als Beispiel eine Methode nach Zhang et al. [1995] beschrieben werden, die in leicht adaptierter Form auch beim Tracking einer deformierbaren Oberfläche mit dem MOSCOT-System zum Einsatz kommt (siehe Abschnitt 6.2). Dabei wird ein Gütemaß für Korrespondenzkandidaten über die Ähnlichkeit ihrer Umgebungen bestimmt. Abbildung 3.22 veranschaulicht die Vorgehensweise. Es wird der Korrespondenzpartner zum roten Marker in Ansicht I_1 gesucht. Dazu werden die Distanzen zu weiteren Markern in der Umgebung betrachtet. Eine hohe Übereinstimmung bei dem roten Marker in Ansicht I_2 (mitte) führt zu einem höheren Ähnlichkeitsmaß gegenüber dem in Ansicht I_2 (rechts) betrachteten Marker. Das Ähnlichkeitsmaß S_M für zwei Punkte m_{1i} und m_{2j} in ihren jeweiligen Nachbarschaften $\mathcal{N}$ im linken bzw. rechten Bild definieren Zhang et al. als

$$S_M(m_{1i}, m_{2j}) = c_{ij} \sum_{n_{1k} \in \mathcal{N}(m_{1i})} \left[\max_{n_{2l} \in \mathcal{N}(m_{2j})} \frac{c_{kl}\delta(m_{1i}, m_{2j}, n_{1k}, n_{2l})}{1 + dist(m_{1i}, m_{2j}, n_{1k}, n_{2l})} \right]. \quad (3.32)$$

Dabei bezeichnen $n_{1k} \in N(m_{1i})$ die in der Nachbarschaft des Punktes m_{1i} vorhandenen Punkte. c_{ij} sind zuvor mittels anderer Verfahren bestimme Korrelationswerte der verschiedenen Kandidaten (diese werden im Fall von markerbasiertem Tracking zu 1 gesetztz). $dist(m_{1i}, m_{2j}, n_{1k}, n_{2l}) = d(m_{1i}, n_{1k}) + d(m_{2j}, n_{2l})$ ist die mittlere Distanz der beiden Paare und

$$\delta(m_{1i}, m_{2j}, n_{1k}, n_{2l}) = \begin{cases} e^{-r/\epsilon_r} & \text{wenn } r < \epsilon_r \\ 0 & \text{sonst} \end{cases} \quad (3.33)$$

eine Gewichtungsfunktion mit

$$r = \frac{|d(m_{1i}, n_{1k}) - d(m_{2j}, n_{2l})|}{dist(m_{1i}, m_{2j}, n_{1k}, n_{2l})} \tag{3.34}$$

als relativem Distanzunterschied.

Es wird also die Summe über alle Markerpunkte der Umgebung gebildet, jeweils für den Kandidaten mit dem niedrigsten Distanzunterschied. Die Gewichtungsfaktoren sorgen dafür, dass nähere Kandidaten stärker in das Ergebnis einfließen. Schließlich wird eine Energiefunktion als Summe der Ähnlichkeitsmaße S_M aller Kandidaten aufgestellt:

$$\mathcal{I} = \sum_{(m_{1i}, m_{2j})} S_M(m_{1i}, m_{2j}) \tag{3.35}$$

Das Problem der bestmöglichen Zuordnung der Marker ist dann äquivalent zur Minimierung dieser Energiefunktion. Die Autoren beschreiben hierzu eine Relaxierung mit einer speziell angepassten Strategie („*some-winners-take-all*“) zur Auswahl passender Zuordnungen im Iterationsschritt. Als Ergebnis erhalten sie eine schnelle und zuverlässig konvergierende Lösung der Markerzuordnungen.

Globale Kriterien

Zusätzlich zu den aufgeführten Verfahren können noch weitere globale Kriterien zu einer Eingrenzung des Korrespondenzproblems herangezogen werden [Forsyth und Ponce 2002; Faugeras 1993]. Das Glattheitskriterium von Oberflächen (disparity gradient limit) [Pollard et al. 1985] fordert, dass sich die Tiefeninformation nur stetig und nicht sprunghaft ändern darf. Damit eignet es sich für deformierbare Oberflächen, aber nicht für die Rekonstruktion unabhängig verteilter Einzelmarker. Das Kriterium eindeutiger Zuordnung kann eingesetzt werden, um kombinatorisch mögliche Markerpaarungen in Gruppen zu unterteilen, die sich gegenseitig ausschließen. So können in dem Beispiel aus Abbildung 3.17 entweder die beiden schwarzen oder die beiden roten Markerpositionen korrekt sein, nicht jedoch ein rote und eine schwarze Position, da für diesen Fall ein Markerbild mehrfach zugeordnet sein müsste. Die Rekonstruktionsmethoden können diese sich ausschließenden Gruppen von Markerkandidaten berücksichtigen. Eine weitere allgemeine Methode bei ausreichend hoher Bildaufnahmefrequenz ist eine Nachverfolgung korrekt zugeordneter Marker aus den vorigen Frames, so dass nur rekonstruierte Positionen akzeptiert werden, die in unmittelbarer Nähe eines Markers des Vorgängerbildes liegen.

Generell ist das Vorgehen stark vom Anwendungsfall abhängig und auch eine Kombination verschiedener Verfahren ist möglich (siehe die Anwendung zum Deformationstracking in Abschnitt 6.2).

Rektifizierung

Bei der Anwendung vieler Stereokorrespondenzalgorithmen werden die Kamerabilder vor der Zuordnung korrespondierender Merkmale rektifiziert. Damit wird eine Transformation der Bilder in die Anordnung eines Stereonormalfalls bezeichnet, d.h. die achsenparallele Ausrichtung der Kamerabildebenen nebeneinander in einer Ebene [Hartley und Zisserman 2000; Luhmann 2010]. Die Epipolarlinien werden so zu horizontalen Linien mit identischem y-Koordinatenwert in beiden Bildern. Da insbesondere für bildbasierte Zuordnungsverfahren sehr viele Berechnungen über Bildbereiche entlang der Epipolarlinien ausgeführt werden müssen, ermöglicht die Rektifizierung eine signifikante Verringerung der Rechenzeit.

Bei markerbasiertem Tracking liegen allerdings nach der Markerdetektionsphase keine Bilddaten sondern bereits Koordinateninformationen vor. Des weiteren ist die wesentliche zu berechnende Größe der epipolaren Zuordnungsverfahren die Distanz zwischen Markerbild und den Epipolarlinien möglicher Zuordnungskandidaten. Diese epipolare Distanz kann auch in nicht-rektifizierten Bildern effizient bestimmt werden, so dass eine vorgeschaltete Rektifizierung keine merklichen Vorzüge aufweist. Ihr Einsatz ist jedoch sinnvoll bei Zuordnungsverfahren mit aufwendigeren Berechnungsschritten wie z.B. den oben angeführten relationalen Verfahren. Anhang A.7 beschreibt ein verbreitetes Verfahren zur Rektifizierung.

3.4.5 Triangulation

Die abschließende Schritt der räumliche Rekonstruktion der Marker basiert auf dem Triangulationsprinzip [Longuet-Higgins 1981; Faugeras 1993]. Dabei wird der Schnittpunkt der beiden aus den Kamerazentren und Markerbildpunkten rekonstruierten Raumstrahlen gesucht.[16] Nur bei idealen Bildpunktkoordinaten und Kameraparametern existiert eine exakte Lösung, bei realen fehlerbehafteten Daten verlaufen die Raumstrahlen windschief. Dies führt zu einem unerwünschten, im Trackingvolumen inhomogenen und von den Schnittwinkeln der Aufnahmekonfiguration abhängigen Rekonstruktionsfehler. Abbildung 3.23 zeigt diesen Fall.

Hartley und Sturm [1997] vergleichen verschiedene Methoden, den optimalen Punkt für die räumliche Rekonstruktion zu bestimmen. Sie geben als optimales Verfahren die Minimierung der Summe der reprojizierten 2D-Fehlerquadrate an (L2-optimale Triangulation). Für euklidische Rekonstruktion im Falle bekannter Kamerakalibrierung zeigt sich jedoch, dass auch das triviale Verfahren, den Mittelpunkt der orthogonalen Verbindung der windschiefen Sehstrahlen zu bestimmen, eine hinreichende

[16] In der Photogrammetrie wird der Rekonstruktionsschritt auch als räumlicher Vorwärtsschnitt bezeichnet.

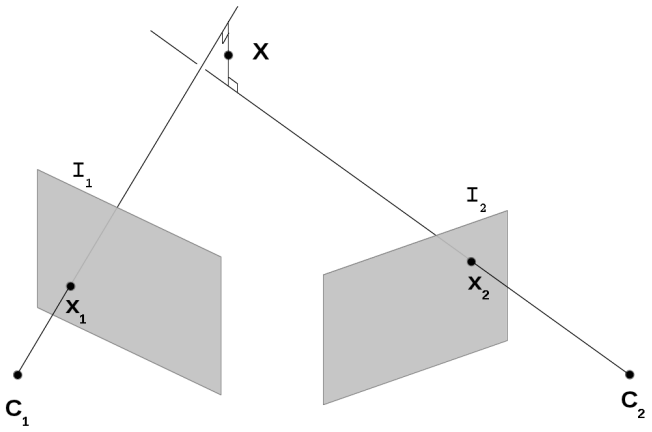


Abbildung 3.23: Durch fehlerbehaftete Daten treffen sich die Raumstrahlen bei der Stereorekonstruktion nicht in einem Raumpunkt sondern verlaufen windschief. Eine einfache Näherung kann über den Mittelpunkt der orthogonalen Verbindung der Raumstrahlen bestimmt werden.

Genauigkeit erreicht. Viele Quellen schlagen daher diese Methode als guten Kompromiss von Genauigkeit und Berechnungsaufwand vor [Beardsley et al. 1994; Forsyth und Ponce 2002; Luhmann 2010].

Einen anderen, rein algebraischen Lösungsansatz ohne geometrischen Hintergrund beschreiben Hartley und Zisserman [2000] als lineare Triangulation. Sie leiten aus den Projektionsgleichungen der Markerbilder $\mathbf{x}_1 = \mathbf{P}_1\mathbf{X}$ und $\mathbf{x}_2 = \mathbf{P}_2\mathbf{X}$ unter Verwendung der Kreuzproduktteigenschaft $\mathbf{x}_j \times (\mathbf{P}_j\mathbf{X}) = \mathbf{0}$ ein lineares Gleichungssystem her. Dieses hat die Form

$$\mathbf{AX} = \mathbf{0}, \text{ mit } \mathbf{A} = \begin{bmatrix} x_1\mathbf{p}_1^{3T} - \mathbf{p}_1^{1T} \\ y_1\mathbf{p}_1^{3T} - \mathbf{p}_1^{2T} \\ x_2\mathbf{p}_2^{3T} - \mathbf{p}_2^{1T} \\ y_2\mathbf{p}_2^{3T} - \mathbf{p}_2^{2T} \end{bmatrix}, \tag{3.36}$$

wobei $\mathbf{p}_j^{iT}$ die Zeilen der Projektionsmatrizen $\mathbf{P}_j$ bezeichnen und jeweils zwei Gleichungen pro Kamerabild verwendet werden. Eine Lösung kann mittels gängiger Methoden zur Lösung linearer Gleichungssysteme bestimmt werden (z.B. über Singulärwertzerlegung [Press et al. 1986; Golub und Van Loan 1996]). Ein Vorteil an diesem Ansatz ist, dass er direkt auf Trackingszenarien mit mehr als 2 Kameras erweiterbar ist, indem für jede neue Kamera zwei zusätzliche Zeilen zum Gleichungssystem hinzufügt werden.

Das Ergebnis kann bei erhöhten Genauigkeitsanforderungen durch eine nichtlinere Optimierung verbessert werden, wobei der Bildabstand aus rückprojiziertem Welt-

punkt $\mathbf{P}_j\mathbf{X}$ und gemessenem Markerbild $\mathbf{x}_j$ als Minimierungskriterium dient [Hartley und Sturm 1997]:

$$\underset{\mathbf{X}}{\operatorname{argmin}} \sum_j \|\mathbf{x}_j - \mathbf{P}_j\mathbf{X}\|^2 \tag{3.37}$$

Die Berechnung erfolgt über Standardverfahren der nichtlinearen Optimierung, z.B. das Levenberg-Marquardt-Verfahren [Press et al. 1986; Golub und Van Loan 1996]. Beim Einsatz in den medizinischen Simulatoren der Arbeitsgruppe kann auf diese Optimierung verzichtet werden.

3.4.6 Mögliche Erweiterungen der Markerrekonstruktionsverfahren

Markerzuordnung

Mögliche Erweiterungen der Korrespondenzanalyseverfahren für größere Anzahl an gleichartigen Markern bzw. auch für nicht-starre Objekte wurden bereits in Abschnitt 3.4.4 aufgeführt. Für eine größere Zahl an Kameras sei hier noch die Methode nach Okutomi und Kanade [1993] erwähnt. Sie erlaubt es bei speziellen Kameraanordnungen in allen Bildern simultan nach Zuordnungen zu suchen, indem sie bei rektifizierten Bildern statt der Disparität des Kandidatenpaares die inverse Distanz zum zugehörigen Weltpunkt betrachtet. Dadurch vermeidet sie die bei vielen Verfahren notwendige Betrachtung einzelner Stereopaare.

Rekonstruktion

Zur Optimierung des Rekonstruktionsergebnisses wird in der Literatur häufig eine Bündelausgleichung (*bundle adjustment*) vorgeschlagen [Hartley und Zisserman 2000; Pintaric und Kaufmann 2007; Luhmann 2010]. Dabei werden neben allen rekonstruierten Weltpunkten auch die Projektionsmatrizen der Kameras in die Optimierung nach Gl. (3.37) miteinbezogen [Triggs et al. 2000]:

$$\underset{\mathbf{X}_i,\mathbf{P}_j}{\operatorname{argmin}} \sum_j \sum_i \|\mathbf{x_{i,j}} - \mathbf{P}_j\mathbf{X}_i\|^2 \tag{3.38}$$

Die Lösung dieses nicht-lineraren Optimierungsproblems ist durch die hohe Anzahl beteiligter Parameter allerdings mit sehr hohem Rechenaufwand verbunden [Hartley und Zisserman 2000] und wurde im Rahmen der MOSCOT-Projekte bislang nicht untersucht.

3.5 Objektrekonstruktion

Die Objektrekonstruktion ermittelt die Posen der Trackingobjekte aus den 3D-Positionen der rekonstruierten Marker. Dazu greift sie auf die aus der Registrierung bekannte Markeranordnung der Objekte (die Objektparameter) zurück. Die Vorgehensweise ist abhängig von der Anzahl der Marker und Objekte und der Identifikation einzelner Marker in den Zuordnungsverfahren.

Die Objektrekonstruktionsverfahren in dieser Arbeit beschränken sich auf die vorherrschenden Anwendungsfälle des MOSCOT-Systems mit einer geringen Anzahl starrer Objekten mit jeweils wenigen Objektmarkern, deren Identifikation häufig noch über eine farbbasierte (Gruppen-)Codierung erleichtert wird (z.B. Simulatorsetups mit 2–3 Objekten aus je 3–5 gleichfarbig kodierten Markern). Für diese Problemstellung existieren einfache Strategien der Objektrekonstruktion, die im folgenden Abschnitt 3.5.1 dargestellt werden. Die Rekonstruktion nicht-starrer, deformierbarer Objekte muss demgegenüber individuell auf die Besonderheiten des jeweiligen Anwendungsfalls zugeschnitten werden. Abschnitt 3.5.2 gibt hierzu einige allgemeine Hinweise. Mögliche Erweiterungen der Verfahren für die Objektrekonstruktion bei einer großen Anzahl gleichartiger Marker werden in Abschnitt 3.5.3 aufgeführt.

3.5.1 Bestimmung der absoluten Orientierung

Im günstigsten Falle der Objektrekonstruktion ist die Zuordnung der gemessenen Markerpunkte zu den Markern der Objektregistrierung bekannt (z.B. durch Farbcodierung), so dass lediglich die Transformation des Objektkoordinatensystems bestimmt werden muss. Der einfachste Ansatz verwendet den ersten Marker direkt als Ursprung des Objektsystems (vgl. die Definition der Objektparameter in Abschnitt 3.2.2) und bestimmt die Richtung der Koordinatenachsen über die Kreuzprodukte der Vektoren der beiden anderen Basismarker.

Für eine genauere Schätzung bei fehlerbehafteten Daten sollten jedoch die Positionen aller Marker (auch weiterer redundanter) gleichermaßen in das Ergebnis einfließen. Dies entspricht der Bestimmung der Transformation (Translation und Rotation) zwischen gemessenen ($\mathbf{X_i}$) und aus der Registrierung bekannten Markerpositionen ($\hat{\mathbf{X}}_\mathbf{i}$) des Objekts:

$$\mathbf{X_i} = \mathbf{R}\hat{\mathbf{X}}_\mathbf{i} + \mathbf{t} \tag{3.39}$$

Diese 3D-3D-Posenschätzung, auch als Bestimmung der absoluten Orientierung oder 3D-Registrierung bezeichnet, ist ebenso wie das Stereokorrespondenzproblem ein zentrales Problem der Computer Vision [Jain et al. 1995, Kap. 12.3] [Forsyth und Ponce

2002, Kap. 12.1]. Die Schwierigkeit im allgemeinen Fall rührt daher, dass gleichzeitig die Transformation zwischen den beiden Punktwolken sowie die Zuordnung der Einzelpunkte zueinander geschätzt werden müssen. Eine Übersicht verschiedener Ansätze findet sich z.B. bei Li und Hartley [2007]. Für den Einsatz beim optischen Tracking kann das Problem durch die Codierung der Marker bzw. die Kontrolle über die Anordnung der Objektmarker vereinfacht werden.

Eine verbreitete Lösung bei bekannter Zuordnung der Einzelpunkte besteht in der Minimierung der Summe der Fehlerquadrate:

$$\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_i \|\mathbf{X_i} - \mathbf{R}\hat{\mathbf{X}}_\mathbf{i} - \mathbf{t}\|^2 \tag{3.40}$$

Verfahren hierzu beschreiben Arun et al. [1987] und Horn [1987]. Eggert et al. [1997] vergleichen verschiedene Methoden und ihre Ergebnisse zeigen, dass Genauigkeit und Robustheit der Methoden ebenso wie die Laufzeiten bei kleinen Punktzahlen in ähnlichem Rahmen liegen. Für das MOSCOT-Projekt wurde die Lösung nach Arun et al. [1987] über Singulärwertzerlegung implementiert.

Bei unbekannter Zuordnung der Punkte $\mathbf{X_i}$ und $\hat{\mathbf{X}}_\mathbf{i}$ muß diese vorab bestimmt werden. Für gleichartige Marker bietet sich eine Identifikation über die Bestimmung der paarweisen Markerdistanzen an, die ja aus den Objektparametern bekannt sind. Bei geeignet konstruierten Markergeometrien mit deutlich unterscheidbaren Intermarkerdistanzen und einer begrenzten Zahl an Markern ist diese Lösung einfach realisierbar, insbesondere da die verschiedenen Objekte meist räumlich getrennt sind und so eine Vorgruppierung der Marker vorgenommen werden kann. Die Suche beginnt mit zusammenhängenden Pfaden über drei Marker, die unregelmäßige Dreicke bilden [Dorfmüller 1999]. Diese können dann leicht auf Objekte mit 4–5 Markern erweitert werden, wobei Distanzmatrizen die bekannten Markerdistanzen innerhalb der Objekte repräsentieren. Bei n Markern sind insgesamt $n(n-1)/2$ Distanzen zu bestimmen und den bekannten Distanzen der Objektmarker zuzuordnen. Die Bestimmung der Markerdistanzen ist darüber hinaus auch geeignet, in der Punktwolke enthaltene Phantommarker zu identifizieren, da die mit ihnen gebildete Distanzmatrix keinem bekannten Objekt zuzuordnen ist. Abbildung 3.24 veranschaulicht dies an einem Beispiel.

Sollte die Distanzmethode bei ungünstiger Position der Objekte nicht für alle Marker zu einem eindeutigen Ergebnis führen, kann aus der Menge aller möglichen Zuordnungen über ein RANSAC-Verfahren (Random Sample Consensus) die korrekte Zuordnung bestimmt werden [Fischler und Bolles 1981]. Bei diesem allgemeinen Verfahren werden Modelle von Messwerten auf Basis einer Stichprobe geschätzt und anhand der Übereinstimmung mit der gesamten Gruppe der Messwerte bewertet. In Abschnitt 3.6 wird allgemein auf RANSAC als wichtiges Werkzeug zur Eliminie-

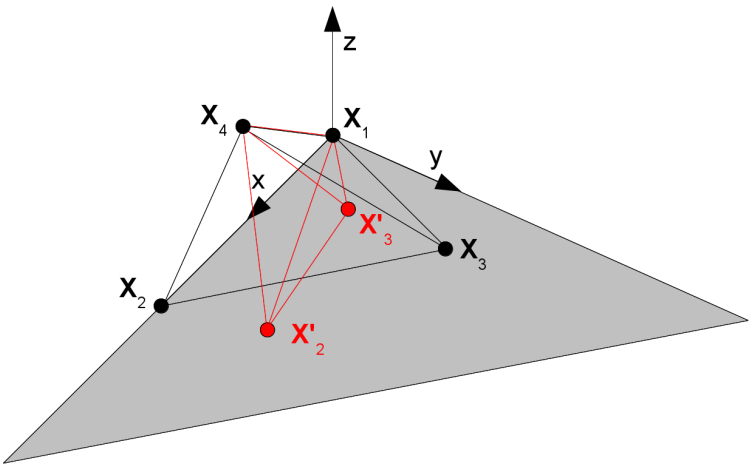


Abbildung 3.24: Objektrekonstruktion im Falle mehrdeutiger Markerbildkorrespondenzen. Die Stereobilder der Marker $\mathbf{X_2}$ und $\mathbf{X_3}$ sind nach Anwendung der Epipolarbedingung nicht eindeutig zuzuordnen, so dass neben den tatsächlichen Markern auch die Phantommarker $\mathbf{X'_2}$ und $\mathbf{X'_3}$ rekonstruiert werden. Durch die bekannten Intermarkerdistanzen aus den Objektparametern für $(\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, \mathbf{X_4})$ lässt sich die Markerkonstellation $(\mathbf{X_1}, \mathbf{X'_2}, \mathbf{X'_3}, \mathbf{X_4})$ ausschließen und damit $\mathbf{X'_2}$ und $\mathbf{X'_3}$ verwerfen.

rung von Ausreißern eingegangen. Forsyth und Ponce [2002, Kap. 12] beschreiben die Anwendung bei der 3D-Posenschätzung:

1. Bestimme zufällige Untermengen aus der Menge der rekonstruierten Marker im Umfang von 3 Markern und ordne sie Referenzmarkern zu
2. Schätze die 3D-Transformation nach obigem Verfahren
3. Teste alle Punkte mit der geschätzten Transformation und bewerte den Fehler
4. Wiederhole die Schritte 1–3, bis die Fehlerschranke unterschritten ist bzw. die maximale Zahl an Iterationen erreicht ist

Da bei einer falschen Zuordnung die resultierende Transformation die meisten Marker weit ab von einer Referenzmarkerposition abbildet, liegt beim Unterschreiten der Fehlerschranke mit sehr hoher Wahrscheinlichkeit eine richtige Zuordnung der ausgewählten Punkte vor. Die absolute Orientierung wird sodann wieder nach obigen Verfahren bestimmt.

Mit steigender Zahl gleichartiger Marker stößt die beschriebene Vorgehensweise durch die quadratisch anwachsenden Zahl der zu betrachtenden Markerdistanzen an Grenzen. Einige alternative Verfahren für solche Fälle sind am Ende des Abschnitts auf-

geführt. Eine eindeutige Entscheidung, ab welchen Marker- bzw. Objektzahlen ein Einsatz der erweiterten Verfahren notwendig wird, ist jedoch nicht möglich. Dies ist vielmehr abhängig von der Robustheit der jeweiligen Anwendung gegenüber fehlerhaften Objektposen und auch von den zu erwartenden Bewegungsmustern. Letztere beeinflussen direkt das Auftreten von Phantommarkern nach Anwendung der Epipolarfilterung und auch die Trennung verschiedener Objekte durch Markergruppierung. In den beschriebenen Setups für medizinische Anwendungen sind die vorgestellten Methoden ausreichend.

3.5.2 Besonderheiten der Rekonstruktion nicht-starrer Objekte

Für die Rekonstruktion deformierbarer Objekte bzw. Oberflächen sind individuelle Lösungen notwendig, die auf das Tracking von Einzelmarkern aufbauen. Da keine starre Markeranordnung vorliegt, können die bisher vorgestellten Verfahren der Objektrekonstruktion nicht eingesetzt werden. So kann zum einen keine Filterung von Phantommarkern über die bekannte Markeranordnung durchgeführt werden, so dass bereits die Zuordnungsverfahren solche Fehlrekonstruktionen minimieren müssen. Gängige Ansätze für das Tracking von Einzelmarkern arbeiten daher mit einer größeren Zahl von Kameras zur Ausnutzung des erweiterten Epipolarkriteriums oder mit sequentiell geschalteten aktiven Markern. Eine weitere Möglichkeit stellt die Nachverfolgung von rekonstruierten Markern dar. Ein zweites Problem beim Fehlen einer starren Markeranordnung ist, dass die Verfahren der absoluten Orientierung nicht anwendbar sind. Hier kann durch Ausnutzung von A-Priori-Informationen über das Objekt versucht werden, unveränderliche Teilstrukturen zu identifizieren, deren Pose dann bestimmt wird (z.B. ein Gelenkmodell für eine Bewegungsrekonstruktion [Gavrila 1999] oder ein Oberflächenmodell bei einem nur teilweise deformierten elastischen Körper).

In Abschnitt 6.2 wird die Umsetzung eines solchen spezialisierten Anwendungsfalls am Beispiel der Deformationserfassung einer elastischen Kugeloberfläche gezeigt.

3.5.3 Mögliche Erweiterungen der Objektrekonstruktionsverfahren

Umeyama [1991] stellen eine Erweiterung des Verfahrens von Arun et al. [1987] für die Bestimmung der absoluten Orientierung beim vermehrten Auftreten fehlrekonstruierter Marker vor. Die Lösung vermeidet falsche Rotationsbestimmungen, die bei ungünstigen Markerkonstellationen auftreten können und ist daher robuster beim Vorhandensein von Ausreißern oder falschen Zuordnungen in den Markerdaten [Dorfmüller-Ulhaas 2003].

Pintaric und Kaufmann [2007] beschreiben Verfahren für die Rekonstruktion einer größeren Anzahl von Markern und Objekten (bis zu 50 Marker und 10 Objekte). Die Objekte werden durch definierte Anordnungen von jeweils 4–5 Objektmarkern gebildet. Es wird auch hier eine Komplexitätsreduktion anhand der Intermarkerdistanzen durchgeführt. Aufgrund der großen Markerzahlen werden aus den Objektkonstellationen Korrelationstabellen der Markerdistanzen aufgestellt und über eine Distanzmatrix der rekonstruierten Marker akkumuliert. Nach einer Schwellwertfilterung bleiben so nur wenige Kandidaten für jedes Objekt übrig. In diesen wird eine Maximum-Cliquen-Suche ausgeführt und schließlich das Objekt rekonstruiert. Auch dieser Ansatz beruht auf geeignet gewählten Anordnungen und Abständen der Objektmarker [Pintaric und Kaufmann 2008].

Allgemeine Verfahren zur Bestimmung der absoluten Orientierung bei unbekannten Punktkorrespondenzen unter der Tolerierung von Ausreißern beschreiben Gold et al. [1998] mit dem SoftAssign-Algorithmus und Li und Hartley [2007] mit dem Box-and-Ball-Algorithmus. Beide Methoden versprechen robuste Ergebnisse auch unter ungünstigen Bedingungen, es bleibt jedoch zu untersuchen, inwiefern die Verfahren zumindest bei begrenzten Markerzahlen unter den Latenz- und Ressourcenanforderungen des MOSCOT-Systems einsetzbar sind.

3.6 Robustes Tracking bei fehlerbehafteten Daten

Ein zentrales Thema bei vielen Trackinganwendungen ist die Robustheit gegenüber unvollständiger oder fehlerbehafteter Datenerfassung und Rekonstruktion [Hartley und Zisserman 2000, Kap. 4.7] [Forsyth und Ponce 2002, Kap. 10.4]. Einen Teil des Problemfelds stellen zufällige Messfehler dar, die aufgrund des Rauschverhaltens von Kameraelektronik und Beleuchtung oder Jitter von Takt- und Synchronisationssignalen entstehen und zu einem fehlerbehafteten Rekonstruktionsergebnis führen. Da diese Zufallsprozesse gewöhnlich einer Normalverteilung genügen, können die Abweichungen durch Filterstufen deutlich verringert werden (siehe Abschnitt 3.6.1). Schwerwiegender sind Ausreißer in den gemessenen bzw. rekonstruierten Daten, die auf fehlende, falsch erkannte oder falsch zugeordnete Markerbilder zurückzuführen sind. Sie unterliegen anders als zufällige Fehler keiner Normalverteilung und behindern damit insbesondere übliche Verfahren zur Parameterschätzung wie die Methode der kleinsten Quadrate (LSE). Das RANSAC-Verfahren als wichtige Möglichkeit zur Erkennung und Eliminierung von Ausreißern wird in Abschnitt 3.6.3 vorgestellt.

3.6.1 Kalmanfilter und andere Filterverfahren

Filter erfüllen drei wesentliche Funktionen im Datenfluss des Trackingprozesses: Sie ermöglichen eine Schätzung von Daten, die aufgrund von Verdeckungen nicht gemessen werden können. Sie erlauben eine Verbesserung der Datenqualität bei verrauschten Messwerten. Bestimmte Filtertypen ermöglichen durch Prädiktion die Cachierung von Latenzen. Die Standardlösung für alle drei Teilaufgaben stellt der Kalmanfilter [Kalman 1960] dar. Aufgrund seiner Bedeutung für die Zustandsschätzung bewegter Systeme wird dieser Algorithmus am Ende des Abschnitts ausführlich beschrieben.

Weitere diskrete Filtertypen, wie einfache Mittelwertfilter oder Tiefpassfilter erweisen sich jedoch für manche Einsätze bei Simulatoren als sinnvoll, da sie weniger Ressourcen benötigen und für den Nutzer einfacher zu verstehen und zu konfigurieren sind. In [Reitmayr und Schmalstieg 2005] werden noch eine Reihe weiterer Filtertypen für den Einsatz im Datenfluss optischer Trackingsysteme aufgeführt, so z.B. Begrenzungsfilter (*clamp filter*), die Werte auf einen zulässigen Wertebereich begrenzen und Halteglieder (*store-and-forward filter*), die bei ausbleibenden Werten einfach den letzten gültigen Wert vorhalten.

Durch den Einsatz individueller Filter und Filterparameter für verschiedene Trackingobjekte kann auf spezielle Anforderungen der Anwendung eingegangen werden, z.B. eine direkte latenzarme Reaktion der Instrumente gegenüber einer gedämpften langsameren Reaktion der Operationsumgebung zur Rauschunterdrückung. Je nach Bedarf der jeweiligen Anwendung stellt das MOSCOT-System verschiedene Filtertypen bereit, die von Haltegliedern über Mittelwert- und Tiefpass- bis hin zu Kalmanfiltern in ihren verschiedenen Ausprägungen reichen.

Kalmanfilter

Der von Kalman [1960] bereits 1960 vorgestellte Algorithmus stellt das wohl wichtigste Verfahren zur Bewegungsschätzung dar. Er verbindet die dynamische Zustandsmodellierung eines linearen Systems mit einem Kleinste-Quadrate-Schätzer und benennt für die fortlaufende Berechnung einen rekursiven Algorithmus, so dass er sich gut für den Einsatz in digitalen Systemen eignet. Aufgrund dieser Faktoren hat sich der Kalmanfilter zum weit verbreiten Standardfilterverfahren für alle Arten von linearen Systemen entwickelt.

Das System wird dazu über eine Reihe von nicht direkt bestimmbaren Zustandsvariablen beschrieben sowie über ein normalverteiltes Rauschen von System und Messprozess (siehe auch Anhang A.8). Der Kalmanfilter liefert dann eine Schätzung für die unbekannten Zustandsvariablen anhand der Reihe der rauschbehafteten Messwerte. Diese fortlaufende Schätzung wird jeweils mittels der aktuellen Messdaten korrigiert.

Die eigentliche Operation des Filters unterteilt sich also in zwei einander abwechselnde Phasen, die Schätzung des neuen Zustands samt Messwert aus dem bisherigen (Prädiktionsphase, *time update*) und die Aktualisierung der Zustandsschätzung anhand eines neuen Messwerts (Korrekturphase, *measurement update*).

Die Bedeutung des Kalmanfilters rührt daher, dass er neben der reinen Filterfunktion auch eine Vorhersage zukünftiger Zustände ermöglicht. Damit können einerseits fehlende Zustände aus der Vergangenheit des Systems heraus geschätzt werden, andererseits die Latenzen des Messsystems versteckt werden. Weiterhin zeigt sich der Filter auch als robust gegenüber ungenauer oder fehlerhafter Modellierung der Systemparameter [Welch und Bishop 1995; Foxlin et al. 2002]. Durch seine rekursive zeitdiskrete Berechnung in zwei Phasen ist der Filter auch sehr flexibel bei der Verarbeitung von nicht synchronen Messdaten. So kann ein schnell laufender Kalmanfilter kontinuierlich Prädiktionsschritte berechnen, die nur beim Vorliegen aktueller Messwerte aus unterschiedlichen Sensoren durch Korrekturschritte aktualisiert werden. Damit ist der Kalmanfilter auch das Mittel der Wahl bei der Datenfusion hybrider Trackingsysteme, so z.B. für die Fusion von inertialen Messsystemen mit optischem Tracking.[17]

Für den Einsatz in nichtlinearen Systemen existiert eine Reihe von Erweiterungen in Form des erweiterten Kalmanfilters (EKF) und des iterativen erweiterten Kalmanfilters (IEKF) [Gelb 1999; Welch und Bishop 1995]. Dabei werden die nichtlinearen Systemgleichungen durch eine Taylorentwicklung in der Umgebung des aktuellen Zustands linearisiert und mit einem gewöhnlichen Kalmanfilter approximiert. Dorfmüller-Ulhaas [2003] beschreibt den Einsatz eines solchen erweiterten Kalmanfilters für die Vorhersage der Objektposen bei einem optischen Trackingsystem.

3.6.2 Strategien für die Behandlung von Ausreißern

Im wesentlichen können drei Ursachen für systematische Fehler bzw. Ausreißer in den Daten ausgemacht werden:

- fälschlich als Markerbilder erkannte Artefakte der Bildaufnahme oder Bildverarbeitung
- fehlerhafte Zuordnungen während der Korrespondenzanalyse
- fehlende Markerbilder durch Verdeckung

Durch die Signalisierung der Marker und die kontrollierten Beleuchtungsbedingungen sind fehlerkannte Markerbilder ein seltenes Phänomen, welches meist durch Analyse von Merkmalen wie Fläche, umschließendem Rechteck und ihren Proportionen

[17] Auf das Potential einer solcher Kombination wird im Ausblick am Ende dieser Arbeit eingegangen.

erkannt und verworfen werden kann. Verbleibende Fehler werden ähnlich wie fehlerhafte Zuordnungen bei der Korrespondenzanalyse als Phantommarker rekonstruiert. Diese können bei der Objektrekonstruktion mittels der dort genannten Strategien (z.B. RANSAC) identifiziert und verworfen werden. Bei Starrkörpern kann durch Wahl geeigneter Markeranordnungen und eine begrenzte Zahl von Objekten die Problematik von Mehrdeutigkeiten bei der Korrespondenzanalyse kontrolliert werden.

Gravierender sind Fehler, die durch Verdeckungen der Markerbilder hervorgerufen werden. Sie können durch redundante Informationen in weiteren Kameras kompensiert werden. Falls weniger als zwei Ansichten vorliegen, kann der entsprechende Marker jedoch nicht mehr rekonstruiert werden. Als Ersatz können z.B. Kalmanfilter zumindest eine Schätzung der Position ermöglichen. Bei redundanter Auslegung der Objektmarker (mehr als drei Marker für ein 6DOF-Tracking) kann die Objektpose auch über die anderen Marker ermittelt werden.

3.6.3 Eliminierung von Ausreißern durch das RANSAC-Verfahren

Ein weit verbreiteter Algorithmus zur Eliminierung von Ausreißern ist das RANSAC-Verfahren (*random sample consensus*) [Fischler und Bolles 1981] [Meer et al. 1991]. Seine Grundidee basiert darauf, Modelle auf Basis von Teilmengen der gegebenen Daten zu erstellen und diese dann im Kontext des Gesamtdatensatzes zu bewerten, in der Absicht, Teilmengen ohne Ausreißer zu identifizieren. Aus den zufällig bestimmten Teilmengen (*random sample*) wird jeweils ein Modell geschätzt und geprüft, wie gut das Modell den gesamten Datensatz erklärt. Modelle, bei deren Bestimmung Ausreißer mit einbezogen wurden, beschreiben den Gesamtdatensatz schlechter und werden dadurch schlechter bewertet. Abschließend werden alle Messpunkte bestimmt, die sich in Übereinstimmung mit dem besten ermittelten Modell befinden (*consensus set*), und aus diesen das finale Modell geschätzt. Die Vorgehensweise ist in Algorithmus 3.4 zusammengefasst.

Einsatzmöglichkeiten beim optischen Tracking ergeben sich wie beschrieben bei der Objektrekonstruktion zur Eliminierung fehlrekonstruierter Marker, sei es durch Artefakte bei der Detektion oder durch Mehrdeutigkeiten bei der Korrespondenzanalyse. Bei begrenzter Zahl von fehlrekonstruierten Markern endet das RANSAC Verfahren nach wenigen Iterationen, so dass den Ressourcen- und Latenzanforderungen von MOSCOT entsprochen wird. (Detaillierte Abschätzungen zu den notwendigen Iterationsschritten und anderen Parameter des RANSAC-Verfahrens finden sich in [Hartley und Zisserman 2000, Kap. 4.7.1].) Darüberhinaus findet RANSAC auch Anwendung bei der Kalibrierung der Kamerasysteme (vgl. die Ausführungen in [Handel 2009]).

Algorithmus 3.4 RANSAC Verfahren

```
1: C ← ∅
2: for j in 1 to N do
3:     subset ← GenerateRandomSubset(S)
4:     model ← EstimateModel(subset)
5:     eval ← EvaluateModel(model)
6:     tmp ← ∅
7:     for all S_i in S do
8:         if ‖S_i − eval_i‖ ≤ THRESHOLD then  ▷ datapoint S_i fits current model
9:             tmp ← tmp ∪ S_i
10:    if |tmp| > |C| then
11:        C ← tmp
12: if |C| > 0 then
13:    model ← EstimateModel(C) return model
14: else
15:    return ∅                                  ▷ no valid model found
```

Kapitel 4

Systemarchitektur

Kapitel 4 erörtert Systemaufbau und Architektur des MOSCOT-Systems. Die im vorigen Kapitel beschriebenen Datenverarbeitungsverfahren werden hier in konkrete Hard- und Softwarestrukturen umgesetzt. Zu Beginn wird kurz der Stand anderer Arbeiten und Projekte mit Bezug zur Systemarchitektur optischer Trackingsysteme dargelegt. Vorüberlegungen zu den in der Einleitung formulierten Systemanforderungen, zur Datenbandbreite und zu Realisierungsmöglichkeiten der Datenverarbeitungsressourcen folgen in Abschnitt 4.1. Im Anschluss wird hieraus das Konzept einer universellen modulbasierten Systemarchitektur entwickelt und dessen maßgebliche Eigenschaften erläutert (Abschnitt 4.2). Die Umsetzung in einzelne Komponenten einer Hardwarearchitektur wird im darauffolgenden Abschnitt 4.3 beschrieben, während Abschnitt 4.4 auf die Implementierung der Softwarearchitektur eingeht. Einige exemplarische Anpassungen der Systemarchitektur an verschiedenartige Aufgabenstellungen beschließen das Kapitel.

4.1 Vorüberlegungen zum Systementwurf

4.1.1 Analyse der Anforderungen

In der Einführung in Kapitel 1 wurden grundlegende Anforderungen an das MOSCOT-System formuliert. Sie sind in der folgenden Auflistung noch einmal aufgeführt:

- Universalität und Flexibilität
- Wiederverwendbarkeit und Erweiterbarkeit
- Skalierbarkeit und Redundanz
- Niedrige Latenzzeiten
- Geringer Ressourcenbedarf

- Anpassbare Genauigkeit
- Test- und Verifizierbarkeit
- Rapid-Application-Development

Im wesentlichen betreffen diese Punkte nichtfunktionale Anforderungen, die von der Auslegung des Systems für verschiedenartige Einsatzszenarien herrühren und es auch von den vorgestellten Komplettsystemen aus Sektion 2.3 unterscheiden. Selbstverständlich steht neben diesen Punkten die zentrale funktionale Anforderung, die dreidimensionale Erfassung und Rekonstruktion der Objektbewegungen. Aus den genannten nichtfunktionalen Anforderungen heraus können jedoch wesentliche Kriterien für den Entwurf der Systemarchitektur hergeleitet werden:

So bedingen **Flexibilität** und **Universalität** des Gesamtsystems ebenso wie die **Wiederverwendbarkeit** und **Erweiterbarkeit** einzelner Komponenten eine variable Systemstruktur aus anpassbaren Einzelkomponenten mit wohldefinierten Schnittstellen. Auch **Skalierbarkeit** und **Redundanz** des Systems profitieren von einer solchen modularen Auslegung, da hierdurch eine bedarfsgerechte Parallelisierung von Datenverarbeitungsressourcen möglich wird.

Genauigkeit, Auflösung und Präzision hingegen sind weitgehend unabhängig von der speziellen Ausgestaltung der Systemarchitektur. Vielmehr werden sie beeinflusst von den Eigenschaften einzelner Komponenten: der Qualität von Objektiven und Kamerasensoren, dem mechanischen Aufbau, aber auch beispielsweise durch die eingesetzten Kalibrierungsverfahren. Damit spielen sie bei den weiteren Überlegungen zur Systemarchitektur nur eine untergeordnete Rolle.

Forderungen nach **schneller Entwicklung von Applikationen** sowie guter **Test- und Verifizierbarkeit** betreffen Aspekte, die über die reine Systemarchitektur hinausgehen. Eine auf Erweiterbarkeit und Zugänglichkeit ausgelegte Basisarchitektur erleichtert jedoch die Umsetzung dieser Aspekte. Darauf aufbauend kann ein Softwarefrontend bereitgestellt werden, welches die Entwicklung neuer Trackinglösungen vereinfacht (so z.B. die im nachfolgenden Kapitel vorgestellte Konfigurations- und Entwicklungsumgebung Tracklab). Vorteilhaft sind wiederverwendbare erweiterbare Datenverarbeitungsmodule und stabile Datenschnittstellen mit Option zu Serialisierung und Deserialisierung. Eine große Hilfestellung bei der schnellen Applikationsentwicklung ist immer auch die Bereitstellung einer Möglichkeit zur Systemsimulation, die es erlaubt, ganze Bereiche der Datenverarbeitungskette mit synthetischen Testdatensätzen zu simulieren und verifizieren.

Die noch ausstehenden Punkte der Anforderungsliste beziehen sich auf die Organisation und Verarbeitung der erfassten Daten. Dazu gehören neben der bereits aufgeführten Forderung nach **Skalierbarkeit** die Realisierung **niedriger Latenzzeiten** bei gleichzeitig **geringen Ressourcenanforderungen**. Die Umsetzung dieser

Datenstruktur	Entität	Datenmenge System 1 2 Kam., 2 Obj., 8 Marker	Datenmenge System 2 6 Kam., 5 Obj., 20 Marker
Bild (Pixeldaten)	Kamera	$640 \times 480 \times 1B =$ **307KB**	$1280 \times 1024 \times 3B =$ **3,93MB**
Monochrom-/Farbpixel: 1Byte / 3 Byte	System	$2 \times 307KB =$ **614KB**	$6 \times 3,93MB =$ **23,59MB**
klassifiziertes Bild	Kamera	$640 \times 480 \times 1bit =$ **38,4KB**	$1280 \times 1024 \times 4bit =$ **655KB**
Monochrom-/Farbpixel: 1bit / 4bit	System	$2 \times 38,4KB =$ **76,8KB**	$6 \times 655KB =$ **3,93MB**
Marker 2D (2D-Pos, Größe, Bound.Box)	Kamera	$8Marker \times 20B =$ **160B**	$20Marker \times 20B =$ **400B**
X:2 Y:2 Größe:4 BBox:8 Misc:4 $\Sigma : 20B$	System	$2 \times 160B =$ **320B**	$6 \times 400B =$ **2400B**
Marker 3D (3D-Pos)	Kamera		
X:4 Y:4 Z:4 Misc:4$\Sigma : 16Byte$	System	$8Marker \times 16B =$ **128B**	$20Marker \times 16B =$ **320B**
Objekt (3D-Pos, 3D-Orientierung)	Kamera		
X:4 Y:4 Z:4 Ψ:4 Θ:4 Φ:4 Misc:6 $\Sigma : 30B$	System	$2Objekte \times 30B =$ **60B**	$5Objekte \times 30B =$ **150B**

Tabelle 4.1: Abschätzung der Datenmenge auf verschiedenen Stufen des Datenverarbeitungsprozesses. Die Zahlen in den Zeilen für Marker 2D/3D und Objekt geben Abschätzungen für den jeweiligen Speicherbedarf in Byte (Koordinaten, weitere Merkmale von Markern/Objekten, Abschätzungen zur Qualität, Verwaltungsinformationen)

Punkte setzt eine effiziente Datenverarbeitungsarchitektur voraus, deren Struktur an Datenfluss und Datenmenge anpassbar ist. Insbesondere ist ein Kompromiss zu finden aus einer schlanken Lösung mit hoher Performanz einerseits und einer universellen, adaptierbaren Gesamtstruktur andererseits. Kernpunkt bei der Abwägung dieser Aspekte und damit auch bei der Konzeption einer geeigneten Gesamtsystemarchitektur ist die zu erwartende Datenbandbreite auf den verschiedenen Verarbeitungsstufen.

4.1.2 Abschätzung der Datenbandbreiten

Für eine Beurteilung der notwendigen Datenbandbreiten im Verlauf des Trackingprozesses werden exemplarisch zwei fiktive Trackingsysteme untersucht: ein eher konservativer Ansatz mit einem Stereo-System aus monochromen VGA-Kameras mit 8 Trackingmarkern (bspw. zwei Starrkörper mit jeweils vier Markern), sowie ein hochperformantes System mit 6 Farbkameras mit 1.3 Megapixelsensor und 100Hz-Updaterate sowie 20 Trackingmarkern. Tabelle 4.1 gibt eine Abschätzung der für die Datenrepräsentation benötigten Speichermenge. Tabelle 4.2 zeigt die hieraus resultierenden Datenbandbreiten in den jeweiligen Verarbeitungsstufen. Die betrachteten Datenstrukturen bzw. Verarbeitungsstufen entsprechen dem in Kapitel 3 eingeführten Datenflussmodell.

Diese Betrachtung stellt nur eine Näherungsrechnung dar, bei anderer Datenrepräsentation sind leicht abweichende Werte möglich. Bereits an der überschlägigen Näherungsrechnung der beiden Beispielsysteme zeigt sich jedoch, dass der wesentliche Schritt der Datenreduktion auf der Stufe der Markerdetektion vollzogen wird, so

Datenstruktur	Entität	System 1 Datenmenge	System 1 Datenbandbreite	System 2 Datenmenge	System 2 Datenbandbreite
Bilddaten	Kamera	$307KB$	$307KB \times 50Hz = \mathbf{15{,}4MB/s}$	$3,9MB$	$3,9MB \times 100Hz = \mathbf{0{,}39GB/s}$
	System	$614KB$	$614KB \times 50Hz = \mathbf{30{,}7MB/s}$	$23,6MB$	$23,6MB \times 100Hz = \mathbf{2{,}36GB/s}$
klassifizierte Bilddaten	Kamera	$38,4KB$	$38,4KB \times 50Hz = \mathbf{1{,}92MB/s}$	$655KB$	$655KB \times 100Hz = \mathbf{65{,}5MB/s}$
	System	$76,8KB$	$76,8KB \times 50Hz = \mathbf{3{,}84MB/s}$	$3,93MB$	$3,93MB \times 100Hz = \mathbf{393MB/s}$
Markerdaten 2D	Kamera	$160B$	$160B \times 50Hz = \mathbf{8KB/s}$	$400B$	$400B \times 100Hz = \mathbf{40KB/s}$
	System	$320B$	$320B \times 50Hz = \mathbf{16KB/s}$	$2400B$	$2400B \times 100Hz = \mathbf{240KB/s}$
Markerdaten 3D	Kamera	$128B$	$128B \times 50Hz = \mathbf{6{,}4KB/s}$	$320B$	$320B \times 100Hz = \mathbf{32KB/s}$
	System	$128B$	$128B \times 50Hz = \mathbf{6{,}4KB/s}$	$320B$	$320B \times 100Hz = \mathbf{32KB/s}$
Objektdaten 6DOF	Kamera	$60B$	$60B \times 50Hz = \mathbf{3KB/s}$	$150B$	$150B \times 100Hz = \mathbf{15KB/s}$
	System	$60B$	$60B \times 50Hz = \mathbf{3KB/s}$	$150B$	$150B \times 100Hz = \mathbf{15KB/s}$

Tabelle 4.2: Abschätzung der resultierenden Bandbreitenanforderungen auf verschiedenen Stufen des Datenverarbeitungsprozesses

dass die notwendigen Datenbandbreiten auf den nachfolgenden Verarbeitungsstufen mehrere Größenordnungen niedriger ausfallen.

Eine Abschätzung der benötigten Verarbeitungsressourcen ist in dieser allgemeinen Form nicht möglich, da die Algorithmen und Rechenoperationen stark von Randbedingungen wie dem zu detektierenden Markertyp, Filterstufen, Art der 3D-Rekonstruktion etc. abhängen. Durch die große Reduktion der Datenbandbreite ist es jedoch eine zulässige Annahme, dass sich auch der wesentliche Ressourcenbedarf der Datenverarbeitung in der Markerdetektionsstufe manifestiert. Gleichzeitig sind die hier erforderlichen Bildverarbeitungsfunktionen jedoch recht einfache Operationen, so dass sie effizient in spezialisierter Hardware implementierbar sind. Die verschiedenen Realisierungsmöglichkeiten der Datenverarbeitungsressourcen werden im folgenden Abschnitt näher untersucht.

4.1.3 Realisierungsmöglichkeiten der Datenverarbeitungsressourcen

Im Unterschied zu gängigen Trackingsystemen ist bei den meisten Anwendungen von MOSCOT-Systemen kein dedizierter Rechner bzw. Prozessor für die Datenverarbeitung vorgesehen. Vielmehr wird eine Co-Nutzung der Ressourcen des Simulator-PCs angestrebt. Da diese jedoch bereits mit Aufgaben wie Simulation, Rendering etc. größtenteils ausgelastet sind, können nur Rechenoperationen mit geringen Ressourcenanforderungen sinnvoll im Simulator integriert werden. Mit den eben getätigten Abschätzungen ist dies insbesondere für die Datenverarbeitung der Markerdetektionsstufe nicht gegeben. Unabhängig von der Rechenlast der Simulator-CPU ergibt sich mit der erwarteten Datenbandbreite von Beispielsystem 2 aus dem letzten Abschnitt auch ein Problem hinsichtlich der Bandbreite gängiger PC-Schnittstellen.

Tabelle 4.3 zeigt die Bruttodatenraten einiger PC-Schnittstellen, die für die Anbindung von MOSCOT-Hardware in Frage kommen. Erreichbare Nettodatenraten

Schnittstelle	Datenbandbreite
USB2	60MB/s
(USB3)[1]	500MB/s
Firewire	100MB/s
(FWS3200)[1]	400MB/s
Cameralink	250MB/s
(Cameralink Full)[1]	680MB/s
GBit-Ethernet	120MB/s

[1] aktuellste Version zum Vergleich

Tabelle 4.3: Datenbandbreiten einiger PC-Schnittstellen [USB Implementers Forum, Inc. 2000; IEEE 2008; Automated Imaging Association 2004; Brady 2006]

liegen teilweise noch um bis zu 30% niedriger [Benz und Feddern 2012]. Verbreitete Schnittstellen wie USB2 sind selbst mit dem schwachen Beispielsystem 1 bereits ausgelastet. Selbst aktuellste Versionen der Standardtechnologien USB und Firewire sind für die Anbindung eines Hochleistungssystems nicht ausreichend, so dass bei diesen in jedem Fall eine externe Datenreduktion notwendig wird.

Für die Lösung beider Problemfelder (Verarbeitungsressourcen und IO-Datenbandbreite) bietet sich eine Verlagerung der erforderlichen Bildverarbeitungsoperationen der Markerdetektionsstufe in spezialisierte Bildverarbeitungshardware an. Hierzu stehen zwei prinzipiell unterschiedliche Technologien zur Verfügung: Field Programmable Gate Arrays (FPGAs) und digitale Signalprozessoren (DSPs) [Kisačanin et al. 2009; Meyer-Baese 2004]. Beide Technologien sind weit verbreitet als Basis spezialisierter Bildverarbeitungssysteme und werden in intelligenten Kameras eingesetzt. Sie bieten durch die direkte Abbildung von Bildverarbeitungsroutinen auf Hardwarestrukturen (FPGA) bzw. durch einen datenflussoptimierten Prozessoraufbau (DSP) hohe Datenraten und Rechenkapazitäten. DSPs erreichen IO-Datenbandbreiten von mehr als 1GB/s ([Spectrum Digital Inc. 2006]) und FPGAs skalieren durch ihren massiv parallelen Aufbau mit einer großen Anzahl externer Datenleitungen noch deutlich höher [Xilinx Inc. 2009].

Abschnitt 4.3.2 geht detaillierter auf Merkmale und Unterschiede beide Technologien ein. Wichtig für die Erfordernisse der MOSCOT-Systemarchitektur ist die Fähigkeit beider Technologien, direkt an die Datenschnittstelle der Kamerasensoren anbinden zu können und damit eine Datenreduktion bereits vor dem Einlesen in den PC durchzuführen. Die Verarbeitung erfolgt dann direkt auf dem Pixeldatenstrom der Kameras und ermöglicht auch eine Reduktion von Verarbeitungs- und Übertragungslatenzen. Dedizierte Bildverarbeitungshardware ist auch einfach in intelligente Kameras integrierbar und erleichtert so die Parallelisierung der Verarbeitungsres-

sourcen. Damit stellt sie für die MOSCOT-Systemarchitektur eine sinnvolle Option zur Auslagerung von Bildverarbeitungskapazitäten vom Simulator-Host-PC in Richtung intelligenter Kameras dar. Die verbleibenden komplexeren Datenverarbeitungsschritte der Marker- und Objektrekonstrukion werden weiterhin im zentralen Host-PC ausgeführt. Eine optionale PC-seitige Implementierung der Bildverarbeitungsverfahren bleibt ebenfalls sinnvoll, da sowohl Kosten als auch Entwicklungszeiten der hardwarebasierten Lösungen deutlich höher ausfallen und somit nicht für alle Anwendungsfälle vorzuziehen sind.

Zur Abgrenzung von der üblichen (softwarebasierten) Bildverarbeitung in General-Purpose-CPUs im PC wird der Einsatz spezialisierter Bildverarbeitungshardware im weiteren Verlauf der Arbeit als hardwarebasierte Bildverarbeitung bezeichnet.[1]

4.2 Konzept und Entwurf der Systemarchitektur

4.2.1 Stand der Technik

Während mathematische Grundlagen und Verfahren des optischen Trackings in der Literatur zur Computervision ausführlich beschrieben sind, finden sich nur wenige Referenzen, die auf die konkrete Umsetzung in eine Systemarchitektur eingehen. So sind dem Autor keine Übersichten zur Hard- und Softwarearchitektur kommerziell verfügbarer Trackingsysteme bekannt, wie sie in Sektion 2.3 bzw. Anhang B.1 beschrieben sind. Oftmals entscheidet jedoch der Systemaufbau über die erfolgreiche Anwendung der theoretischen Konzepte in der Praxis (vgl. hierzu z.B. auch die Erläuterungen von Foxlin et al. [2002]). Im Folgenden wird daher näher auf die Architekturkonzepte einiger der in Sektion 2.4 genannten Softwarebibliotheken eingegangen. Darüber hinaus werden Arbeiten zu optischen Trackingsystemen aus dem Forschungsbereich angeführt, die zumindest in Teilbereichen Angaben zur Umsetzung der Systemarchitektur machen.

Reine Softwarearchitekturen:

Kato et al. [2005] beschreiben die Implementierung von Datenstrukturen und Datenfluss im AR-Projekt ARToolkit. Es ist ausgerichtet auf das effiziente Durchreichen von Videodaten in verschiedenen Formaten und deren Überlagerung mit gerenderten

[1] Selbstverständlich muss auch Bildverarbeitungshardware über eine Art Software programmiert werden (Hardware-Beschreibungsprachen bzw. DSP-Software) und die softwarebasierte Bildverarbeitung läuft auf der darunterliegenden PC-Hardwareplattform ab. Zur Unterscheidung der beiden Konzepte werden jedoch die genannten Begriffe verwendet.

Ansichten der getrackten AR-Objekte. Während ARToolkit eine vollständige Architektur zur Umsetzung eigener AR-Anwendungen bereitstellt, bleiben Datenfluss und die bereitgestellten Datenformate auf die Verarbeitung der vorgegebenen AR-Marker festgelegt.

Einen Ansatz zur Abstraktion von Datenformaten und Geräten liefern als Middleware bezeichnete Pakete wie z.B. die VRPN-Bibliothek für Eingabegeräte [Russell M. Taylor et al. 2001]. Sie stellt den nachgelagerten Anwendungen definierte generische Schnittstellen zur Weitergabe von rekonstruierten Daten der Eingabesysteme zur Verfügung.

Wesentliche Arbeiten zur Realisierung einer flexibleren Datenflussarchitektur für optische Trackingsysteme wurden im Rahmen des OpenTracker-Projektes publiziert. So beschreiben Reitmayr und Schmalstieg [2001] eine verteilte Datenflussstruktur als Grundlage für die flexible Auslegung von Anwendungen optischer Trackingsysteme. In [Reitmayr und Schmalstieg 2005] wird dies weiter ausgeführt als modulare Datenflussarchitektur mit generischen Datentypen und verschiedenen Arten von Datenknoten, die die Verarbeitungsfunktionen enthalten. Die Datenflussstruktur wird dabei über Konfigurationsfiles definiert. Newman et al. [2003] und Newman et al. [2004] führen diesen Ansatz unter der Bezeichnung „Ubiquitous Tracking“ fort in Richtung auf eine verteilte, dynamisch rekonfigurierbare Softwarearchitektur, die die einfache Hinzunahme und Veränderung von Objekten und Trackingsystemen im laufenden Betrieb ermöglicht. Huber et al. [2007] beschreiben eine weitere Umsetzung verteilter Datenflussarchitekturen für Ubiquitous-Tracking-Umgebungen, die auf einer Client-Server-Architektur beruht. Wie in Abschnitt 2.4 erörtert, arbeiten alle diese auf dem ursprünglichen OpenTracker-System aufsetzenden Erweiterungen als Middleware mit bereits rekonstruierten Datensätzen und sehen keine Möglichkeit der Integration von Hardwarekomponenten und von Datenverarbeitungsmethoden der Markerdetektions- und Rekonstruktionsphase vor. Die Weiterverarbeitung der Daten auf Applikationsebene auf Basis generischer Datentypen mit einer freien Verknüpfung der Datenverarbeitungsknoten zeigt allerdings deutlich die Vorteile modularer Datenflussarchitekturen bei der universellen Auslegung von Trackingsystemen auf.

Weitere Arbeiten gehen speziell auf Softwarearchitekturen für die Datenfusion verschiedener Trackingsysteme ein. Ribo et al. [2004] beschreiben ein hybrides System aus optischen und inertiellen Messsensoren und die flexible Realisierung der Datenfusion der verschiedenen Sensorsysteme. Hoff und Golden [1999] stellen eine Softwarearchitektur für die Datenfusion von mobilen und ortsfesten optischen Trackingsystemen vor.

Vollständige Systemarchitekturen ohne Spezialhardware:

Einige weitere Autoren beschreiben dedizierte Anwendungen optischer Trackingsysteme mit vollständigem Systemaufbau, einschließlich der Bildverarbeitungs- und Rekonstruktionsschritte. Mehrere Arbeiten älteren Datums basieren auf einer Realisierung mittels analoger IR-Kameras und Digitalisierung der Bilder im PC über Framegrabbermodule. Bildverarbeitung und Rekonstruktion sind hier speziell auf das jeweilige Anwendungsszenario zugeschnitten und werden vollständig in Software auf der PC-Seite ausgeführt. Beispiele solcher Systeme finden sich bei Dorfmüller und Wirth [1998], Dorfmüller [1999] und Ribo et al. [2001]. Sie beschreiben eine Softwarearchitektur mit ähnlichen Verarbeitungsphasen wie in Kapitel 3 aufgeführt, erwähnen jedoch keine Modularisierung der zugrundeliegenden Softwarearchitektur. Die Hardwareintegration beschränkt sich auf die Framegrabber, die zur Digitalisierung der Videodatenströme eingesetzt werden. Aktuelle Ansätze verwenden meist digitale Kameras mit Firewire-Schnittstelle und können so auf den Einsatz von Framegrabbern verzichten (z.B. [Mulder et al. 2003; Pintaric und Kaufmann 2007]). Das System aus [Pintaric und Kaufmann 2007] ist dabei nicht auf einen speziellen Anwendungsfall ausgelegt, sondern bietet eine generische Softwarearchitektur für die Rekonstruktion und Unterscheidung einer Vielzahl von Objekten aus Konfigurationen gleichartiger Marker. Der Datenfluss bleibt allerdings auf die vorgegebenen IR-reflektiven Kugelmarker und eine rein PC-seitige Datenverarbeitung festgelegt.

Systemarchitekturen mit Spezialhardware:

Allen bisher aufgeführten Arbeiten ist gemein, dass sie die Datenverarbeitung ausschließlich auf der PC-Plattform realisieren. Im Unterschied dazu setzen die folgenden Arbeiten spezialisierte Datenverarbeitungshardware in Form von Digitalen Signalprozessoren (DSPs) bzw. programmierbaren Logikbausteinen (FPGAs) ein.

Den Aufbau einer intelligenten Kamera für Trackinganwendungen präsentieren Muehlmann et al. [2004]. Ein CMOS-Bildsensor wird hier mit einem FPGA zur Bildverarbeitung gekoppelt und über einen USB-Mikrokontroller an den PC angebunden. Die Kamera kann selektiv kleine Fensterbereiche mit sehr hoher Geschwindigkeit (>1000Hz) auslesen und im FPGA ein Blobtracking (Connected-Component-Labeling) ausführen. Ausführlich wird der Einsatz von FPGAs für die Detektion von Farbmarkern in Arbeiten von Johnston et al. [2005a] und Johnston et al. [2005b] erläutert. Der Fokus liegt auf der Implementierung des Datenstroms der Bildverarbeitungsverfahren innerhalb des FPGA. Dabei wird eine Datenverarbeitungspipeline aufgebaut, die ähnlich den Verfahren in Kapitel 3 eine Farbklassifizierung, morphologische Filter und Segmentierungsverfahren implementiert. Die FPGA-basierte Bildverarbeitungskette wird ausführlich beschrieben, jedoch auch hier ohne auf weitergehende Schritte der Rekonstruktion einzugehen oder eine Integration in ein Ge-

samtsystem vorzustellen.

Einen allgemein gehaltenen Überblick über das Themengebiet intelligenter Kameras im Kontext verteilter Datenverarbeitung geben Rinner und Wolf [2008]. Sie erläutern den Aufbau der Hardware aus verschiedenen Modulen mit Sensoreinheit, Verarbeitungseinheit und Kommunikationseinheit und geben eine Übersicht zur Implementierung verteilter Algorithmen. Weiterhin beschreiben sie übersichtsartig verschiedene Anwendungsbereiche und stellen eine allgemeine Softwarearchitektur für den Aufbau eines verteiltes Netzwerks aus intelligenten Kameras vor. Die einzelnen Kameraeinheiten sind bei diesem Ansatz als gleichberechtigte Agenten in einem größeren Verarbeitungsnetz ausgeführt und es gibt keine Anbindung an einen zentralen Host.

Ein integriertes Gesamtsystem aus Hard- und Softwarearchitektur wird von Schlessman et al. [2006] für ein eingebettetes Trackingsystem auf Basis des optischen Flusses vorgestellt. Besonders ressourcenintensive Teile der Bildverarbeitung (der KLT-Tracker) werden hier in ein FPGA ausgelagert, während weitere höhere Verarbeitungsschritte des eigentlichen Objekttrackings in Software im PC verbleiben. Das FPGA führt Farbraumkonvertierung, Faltungsoperatoren sowie weitere Schritte des KLT-Trackers (Gradientenmatrix, Inversion und Flussvektorbestimmung) aus. Es ist nicht bekannt, ob das System auch für einen Einsatz mit markerbasiertem optischen Tracking adaptiert wurde.

4.2.2 Übersicht über das Architekturkonzept

Aufbauend auf dem Stand der Technik und den Vorüberlegungen aus Abschnitt 4.1 wird ein Konzept für die Systemarchitektur von MOSCOT entwickelt. Dieses Konzept stützt sich auf folgende zentrale Punkte:

- Unterteilung des Datenverarbeitungsprozesses in wiederverwendbare Module, die den einzelnen Phasen und Verarbeitungsschritten aus Kapitel 3 zugeordnet werden können.
- Aufteilung der zugrundeliegenden Datenverarbeitungsressourcen auf verschiedene Technologien - ein zentraler PC-basierter Hostrechner für allgemeine, komplexere Berechnungsschritte und eine optionale, direkt an die Kameramodule angebundene Spezialhardware für einfachere, breitbandige Bildverarbeitungsoperationen.
- Komplette Simulation des Trackingprozesses mittels virtueller, in Software nachgebildeter Datenquellen für effiziente Entwicklung von Trackinglösungen.
- Auslegung für niedrige Latenzzeiten durch Realisierung einer schlanken Datenflussarchitektur in Software und die Option direkter Hardwarebildverarbeitung auf dem Kameradatenstrom.

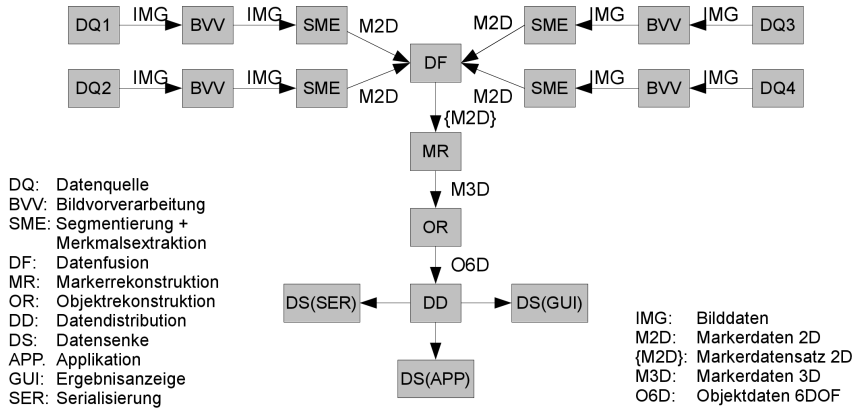


Abbildung 4.1: Verteilte Datenflussarchitektur des MOSCOT-Systems ausgehend von Datenquellen über die verschiedenen Datenverarbeitungsschritte bis hin zu den Datensenken.

In Abbildung 4.1 ist eine schematische Übersicht der Systemarchitektur wiedergegeben. Sie bildet die Struktur des Datenflusses in Form einer (teilweise parallel ausgelegten) Kette von Datenverarbeitungsmodulen nach: ausgehend von der Kamerasensoren als Datenquelle, über die Stufen der Markerdetektion (Bildvorverarbeitung und Segmentierung & Merkmalsextraktion), Markerrekonstruktion und Objektrekonstruktion, bis hin zur jeweiligen Anwendung als Datensenke.

Unabhängig hiervon ist die Aufteilung der darunterliegenden Hardwareplattformen. Während die zentralen Berechnungsschritte nach der Fusion der Kameradatenströme (d.h. Marker- und Objektrekonstruktion) immer auf einer PC-basierten Hostplattform ausgeführt werden, existiert für die Markerdetektion die Möglichkeit, sie ganz oder teilweise in spezialisierte Hardware zu verlagern.[2] Die Zuordnung dieser dedizierten Verarbeitungseinheiten zu den Kameramodulen führt direkt zum Konzept intelligenter Kameras. Abbildung 4.2 zeigt verschiedene Möglichkeiten der Aufteilung, so die komplette Ausführung der Markerdetektion in der Bildverarbeitungshardware (Zeile 1), die alleinige Verlagerung der Bildvorverarbeitungsschritte (Zeile 2) und schließlich der Verzicht auf spezialisierte Hardware mit Datenverarbeitung ausschließlich auf dem Host-PC (Zeile 3). Bei zusätzlicher Nachbildung der Datenquellenfunktionalität als virtuelle Datenquellen in Software ist die Simulation der kompletten Datenverarbeitungskette für die Anwendungsentwicklung möglich (Abbildung 4.2, unterste Zeile).

[2]Die Markerdetektionsphase wird dafür unterteilt in die Schritte der Bildvorverarbeitung sowie der Segmentierung und Merkmalsextraktion.

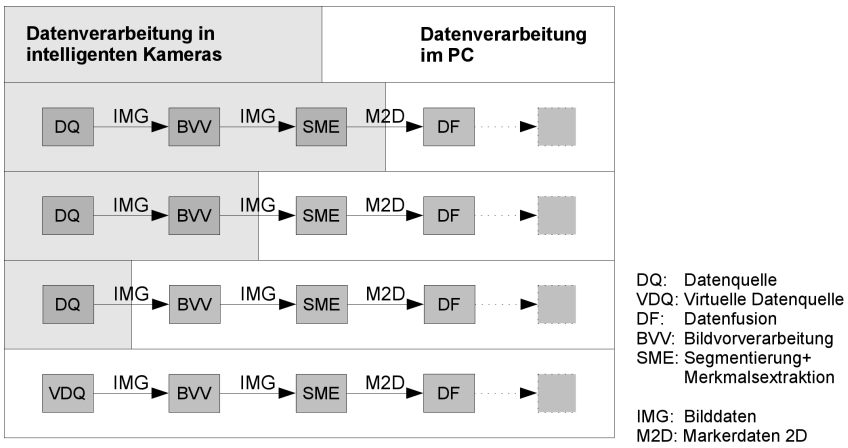


Abbildung 4.2: Die uniforme Auslegung der Datenverarbeitungsstrukturen ermöglicht die Umsetzung einzelner Bildverarbeitungsoperationen sowohl in spezialisierter Bildverarbeitungshardware als auch in Software im Host-PC. Eine Nachbildung der realen Datenquellen als virtuelle Einheiten führt zur Simulierbarkeit des Gesamtsystems in Software.

Konzeptionell erweitert die MOSCOT-Systemarchitektur damit den datenflussbasierten Ansatz von Softwarebibliotheken der Middleware-Schicht auf die vorgelagerten Schritte von Markerdetektion und Rekonstruktion. Gleichzeitig integriert sie hardwarebasierte Bildverarbeitungsressourcen in die rein softwarebasierten herkömmlichen Datenflussarchitekturen.

Bedingung für die Modularisierung der Datenverarbeitungsschritte und ihre Austauschbarkeit zwischen Hard- und Softwareressourcen ist die Festlegung klar definierter Schnittstellenformate für den Parameter- und Datenaustausch. Hierfür wird auf die bereits im letzten Kapitel eingeführten Datenstrukturen für die dynamischen und statischen Informationen des Trackingprozesses zurückgegriffen (vgl. Tabellen 3.1 und 3.2).

Die vier zentralen Aspekte der MOSCOT-Systemarchitektur werden in den folgenden Abschnitten näher erläutert.

4.2.3 Modularisierter Aufbau der Gesamtarchitektur

Voraussetzung für die Realisierung einer flexiblen Systemarchitektur ist die durchgehende Modularisierung aller Komponenten. Durch den Austausch von Modulen – Hardwarekomponenten oder auch Datenverarbeitungsverfahren – kann das Gesamtsystem an unterschiedliche Aufgabenstellungen angepasst werden. So ermöglicht etwa

der Einsatz verschiedener Hardwaremodule wie Kamerasensoren oder Bildverarbeitungseinheiten die Anpassung an unterschiedliche Trackingszenarien, Markertypen oder Genauigkeits- und Geschwindigkeitsanforderungen. Mit der Auswahl geeigneter Bildverarbeitungshardware können Datenverarbeitungs- und -übertragungsresourcen projektspezifisch skaliert werden. Und durch den Einsatz verschiedener Datenübertragungsmodule kann auch die Datenschnittstelle zwischen Kamerahardware und Host-PC auf den jeweiligen Anwendungsfall eingestellt werden (z.B. für Ethernet-, CameraLink- oder USB-Anbindung).

Bei den logischen Modulen der Datenverarbeitungskette selbst erlauben Austausch und Neukombination die Adaption an anwendungsspezifische Erkennungs-, Rekonstruktions- und Filteralgorithmen und damit wiederum an spezielle Marker- und Objekteigenschaften. Dabei bleibt der Integrationsaufwand neuer Verarbeitungsmodule durch die definierten Schnittstellenformate und Datenpfade gering. Die Entkopplung der Datenverarbeitungsschritte erleichtert ebenfalls die Wiederverwendung einzelner Komponenten. So können etwa Module zur Detektion spezieller Markertypen wiederverwendet werden, unabhängig davon, wie diese Marker zu Objekten zusammengestellt sind. Auf der Hardwareseite können passende Module mit Kamerasensoren, Bildverarbeitungshardware und Kommunikationsschnittstellen wiederverwendet und anwendungsspezifisch neu kombiniert werden.

In der Gesamtheit führt der modulare Ansatz zu einem breiteren Spektrum an Einsatzmöglichkeiten bei gleichzeitig begrenztem Adaptions- und Entwicklungsaufwand für spezielle Trackinglösungen. Ein zusätzlicher positiver Aspekt der Modularisierung ist die resultierende Skalierbarkeit der Systemarchitektur. Durch zusätzliche Datenquellen bzw. Verarbeitungsmodule in der Datenflussarchitektur kann das System recht einfach hinsichtlich der Anzahl von Kameras, Markern und Objekten skaliert werden.

4.2.4 Duale Auslegung von Bildverarbeitungskomponenten in Hardware und Software

Die Möglichkeit, ressourcenintensive Verarbeitungsschritte von Bildvorverarbeitung und Markerdetektion bei Bedarf in dedizierte Hardware auszulagern, erfordert eine einheitliche Schnittstellenspezifikation und Repräsentation der Daten und Systeminformationen in den zugehörigen Hardware- bzw. Softwaremodulen. Damit können einzelne Verarbeitungsschritte gleichartig in Hard- und Software ausgelegt werden und je nach Bedarf in die Datenverarbeitungskette eingefügt werden. So sind maßgeschneiderte Lösungen möglich: von spezialisierten Ansätzen bei denen das Gros der Datenverarbeitung in dedizierter Hardware abläuft bis hin zu Konzepten mit einfachen Kameras und reiner Softwaredatenverarbeitung, die kurzfristig und günstig

realisierbar sind (vgl. Abbildung 4.2).

Weitere Vorteile ergeben sich beim Entwicklungsprozess, da Anpassungen der Datenverarbeitung bei einer Softwareimplementierung schneller zu realisieren und zu testen sind als in Hardware. Die Implementierung kann so schrittweise von Software in Hardwaremodule übernommen und Verarbeitungsergebnisse aber auch Geschwindigkeit und Latenzzeiten direkt miteinander verglichen werden. Wird diese Herangehensweise auch auf die Datenquellen selbst ausgeweitet, so führt das zum Konzept virtueller Trackingsysteme, welches im folgenden Abschnitt beschrieben ist.

4.2.5 Simulation von Trackingsystemen durch virtuelle Datenquellen und -senken

Das Konzept virtueller Trackingsysteme entstand im Verlauf der Arbeiten an neuen Komponenten und Einsatzbereichen für MOSCOT-Trackingsysteme. Es wurde eine Lösung gesucht, mit der die (Software)-Datenverarbeitungskette mit vordefinierten Datensätzen getestet bzw. Trackingsetups vor Verfügbarkeit von Hardwarekomponenten oder dem endgültigen Trackingaufbau evaluiert werden können. Während Unit- und Modultests nur einzelne Komponenten abdecken, ermöglicht die Einführung virtueller Datenquellen eine vollständige Simulation der Datenverarbeitungskette von virtuellen Datenquellen (Kameras) bis hin zu den Datensenken (Anwendungen). Virtuelle Datenquellen vervollständigen quasi die duale Auslegung von Hard- und Sofwarekomponenten, indem sie als Softwareimplementierung der Kameras die Einspeisung vorgegebener Trackingdaten ermöglichen.[3]

Die Simulation von Systemen mittels serialisierter realer oder auch generierter synthetischer Daten erlaubt es, Fehlersituationen im Trackingverlauf nachzubilden und ähnlich wie in einem Software-Debugger die Datenverarbeitungskette zu pausieren, weiterzuschalten und die jeweiligen Ergebnisse zu untersuchen. Damit sind virtuelle Systeme auch ein wichtiger Baustein für das im folgenden Kapitel vorgestellten Entwicklungsfrontend Tracklab zur schnellen Applikationsentwicklung (RAD – Rapid-Application-Development) mit dem MOSCOT-System.

Über die Systemsimulation hinaus bietet der Mechanismus virtueller Datenquellen auch eine einfache Möglichkeit, Trackinghardware von Drittanbietern an die MOSCOT-eigene Datenflussarchitektur anzubinden. Hilfreich ist der Anschluß solcher externen Systeme z.B. für den direkten Vergleich mit einem Referenzsystem oder auch für eine Kombination erweiterter Rekonstruktionsverfahren von MOSCOT mit Fremdsystemen.

[3] Virtuelle Datenquellen unterstützen aufgrund ihrer Platzierung in der softwarebasierten Datenverarbeitungskette auf PC-Seite nur Tests der Softwarekomponenten. Eine Erweiterung auf Hardware-Testdatenquellen würde das Konzept vervollständigen, dies ist aktuell jedoch noch nicht realisiert.

4.2.6 Vermeidung von Latenzen

In Abschnitt 2.2.7 wurde bereits die Bedeutung niedriger Latenzzeiten für den Einsatz eines optischen Trackingsystems in VR- und AR-Simulatoren herausgestellt. Dieser Aspekt muss trotz der universellen Auslegung der MOSCOT-Systemarchitektur bei der Implementierung aller Komponenten sichergestellt werden.

Maßnahmen zur Minimierung von Latenzzeiten betreffen zum einen die Architektur des Gesamtsystems, zum anderen die Implementierung einzelner Datenverarbeitungskomponenten. Während Ausführungen zur Implementierung bei den einzelnen Komponentenbeschreibungen erfolgen, werden an dieser Stelle Überlegungen zur Architekturauslegung angestellt.

Eine generelle Strategie ist die Vermeidung überflüssiger Zwischenspeicher- und Kopiervorgänge innerhalb der gesamten Datenverarbeitungskette. In der Hardwarearchitektur von MOSCOT wird dies erreicht durch ein durchgängiges direktes Weiterreichen der Bilddaten. So entsteht ein kontinuierlicher Pixeldatenstom auf dem Datenverarbeitungsoperationen „on-the-fly“ ausgeführt werden. Alle Verarbeitungsoperationen arbeiten direkt hintereinander auf einer Datenpipeline, während diese kontinuierlich mit den Kameradaten gespeist wird. Im Unterschied zum klassischen Framegrabber-Einsatz im PC können so die Verarbeitungslatenzen größtenteils in der Kameraauslesezeit „versteckt“ und auch die Übertragungslatenzen minimert werden. Ähnlich muss auch innerhalb der Softwarearchitektur auf der PC-Seite ein möglichst direktes Durchreichen der Daten angestrebt und Umkopieren von Datenstrukturen vermieden werden.

Ein weitere Ansatzpunkt zur Latenzverringerung besteht in der Bereitstellung ausreichender Übertragungs- und Verarbeitungsressourcen auf allen Datenverarbeitungsstufen. Dem trägt die Architektur durch das beschriebene Konzept verteilter Rechenressourcen Rechnung, das bei Bedarf eine Parallelisierung der Datenverarbeitung in intelligenten Kameramodulen ermöglicht – und damit eine Skalierung auf der Stufe, die mit den Bilddaten die höchste Datenbandbreite bewältigen müssen. Die damit einhergehenden Datenreduktion reduziert darüber hinaus auch die Übertragungslatenzen, die einen signifikanten Anteil der Gesamtsystemlatenz ausmachen können.

Ebenfalls relevant für das Erreichen niedriger Systemlatenzen ist die in Abschnitt 2.2.7 beschriebene Synchronisationslatenz zwischen der Anwendung und dem Trackingsystem. Durch Synchronisation auf den Mastertakt der Anwendung stellt die Systemarchitektur sicher, dass diese zusätzliche Latenzzeit minimal bleibt.

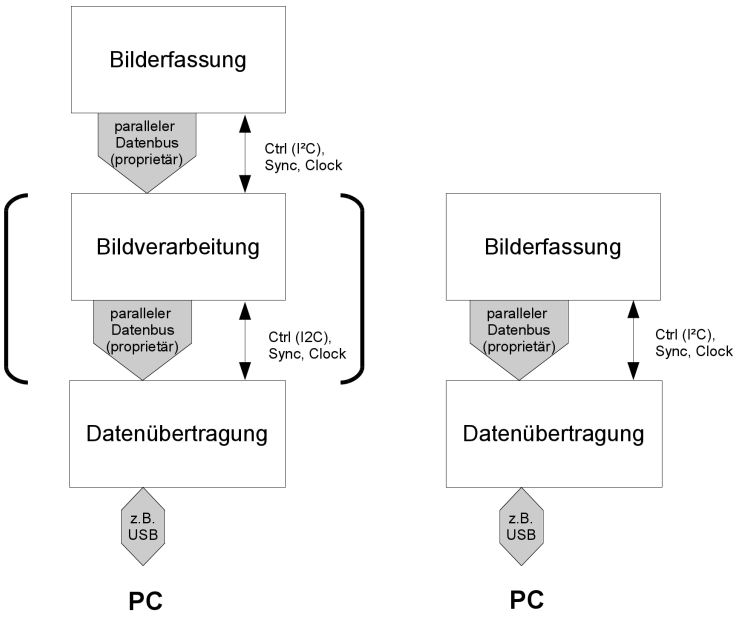


Abbildung 4.3: Überblick über die Module der Hardwarearchitektur sowie die definierten Schnittstellen zum Datenaustausch. Links eine intelligente Kamera mit dem optionalen Bildverarbeitungsmodul, rechts eine reine Kameraauslegung mit Bildverarbeitung im Host-PC.

4.3 Realisierung der Hardwarearchitektur

Schwerpunkte der hardwareseitigen Umsetzung der Systemanforderungen betreffen die Realisierung des Modulaufteilung und die Sicherstellung niedriger Latenzzeiten des Trackingprozesses. Letzteres wird erreicht durch die Bereitstellung dedizierter Bildverarbeitungshardware, die direkte Anbindung an den Kameradatenstrom und die Synchronisation der Kameras.

Ähnlich der logischen Datenverarbeitungsstruktur des MOSCOT-Systems ist auch die physische Hardware als Modulsystem ausgeführt. Wie beschrieben existiert als zentrale Hardwarekomponente ein Host-PC, der die meisten Datenverarbeitungsprozesse realisiert.[4] Angebunden an den zentralen Host sind die Kameras, die in Modulbausweise ausgeführt sind. Je nach Aufgabenstellung sind so verschiedene Bildsensoren einsetzbar und die Übertragungstechnologie zur Host-PC-Anbindung kann frei

[4]Für Anwendungen mit erhöhten zeitlichen Anforderungen (z.B. navigierte Operationen) ist hier auch der Einsatz eines eingebetteten Systems mit Echtzeitfähigkeit vorgesehen. Diese Option ist bisher nicht realisiert, Hard- und Softwarearchitektur sind aber für eine direkte Portierbarkeit ausgelegt.

gewählt werden. Optional ist durch ein Bildverarbeitungsmodul die Erweiterung zu intelligenten Kameras möglich, die dann bereits die Majorität der Datenreduktion implementieren.

Abbildung 4.3 gibt einen Überblick über die Komponenten der Hardwarearchitektur sowie die Schnittstellen zwischen den Modulen. In der Gesamtheit unterteilt sich die Hardwarearchitektur von MOSCOT damit in funktional getrennte Module für

- Bilderfassung
- Bildverarbeitung (optional)
- Kommunikation und Datenübertragung[5]

sowie die Rekonstruktion und Systemsteuerung im Hostrechner.

Durch definierte Schnittstellenspezifikation zwischen den Modulen wird die Rekombinierbarkeit von Kamera-, Datenverarbeitungs- und Datenübertragungseinheit sichergestellt. Während die Nutzdaten den Strukturen aus Kapitel 3 entsprechen, kann die physikalische Schnittstelle und das Übertragungsprotokoll frei gewählt werden. Die Intermodulkommunikation zwischen Bilderfassung, Bildverarbeitungs- und Kommunikationsmodul ist als proprietäre Schnittstelle realisiert. Sie umfasst einen parallelen Bus für die Bilddatenübertragung (je nach Datenformat des Sensor-ICs 8bit bzw. 16bit breit), serielle Bussysteme (z.B. I²C [Philips Semiconductors 2000]) für Kontrollfunktionen, sowie weitere Verbindungen für Takt, Synchronisation und Stromversorgung. Für die Anbindung der Kameras an den Hostrechner bieten sich die in Abschnitt 4.1.3 aufgeführten PC-Technologien an (Ethernet, USB, auch CameraLink).

Die Hardwaremodule sind für flexible Kombination von Bilderfassung, Bildverarbeitung und Datenübertragung als getrennte Einheiten ausgelegt. So ist für räumlich beengte Einsatzorte beispielsweise eine bauliche Trennung des Sensorkopfs vom Kamerakörper denkbar, es kann aber auch eine Kombination der Module auf einer gemeinsamen Trägerplatine realisiert werden (siehe Abbildung 4.10 und Abbildung 6.2(b)). Der logische Aufbau und das Zusammenwirken der Module bleiben hiervon unberührt. Das optionale Bildverarbeitungsmodul kann zwischen Bilderfassungs- und Kommunikationsmodule eingefügt werden. Die Schnittstellen zwischen den Modulen sind so gestaltet, dass im Falle seines Fehlens die Bilderfassungsmodule direkt an die Kommunikationsmodule angebunden werden können und eine einfache (nichtintelligente) Kamera entsteht. Eine direkte durchgängige Verbindung des Datenstroms zum PC ist dabei mit und ohne das Bildverarbeitungsmodul sichergestellt. Die Hardwarearchitektur sieht weiter eine freie Wahl der Synchronisationsmethode

[5] Da die Übertragungsmodule für den Datenverarbeitungsprozess transparent erscheinen sind sie in der bisherigen logischen Übersichtsdarstellungen der Datenflussarchitektur nicht dargestellt.

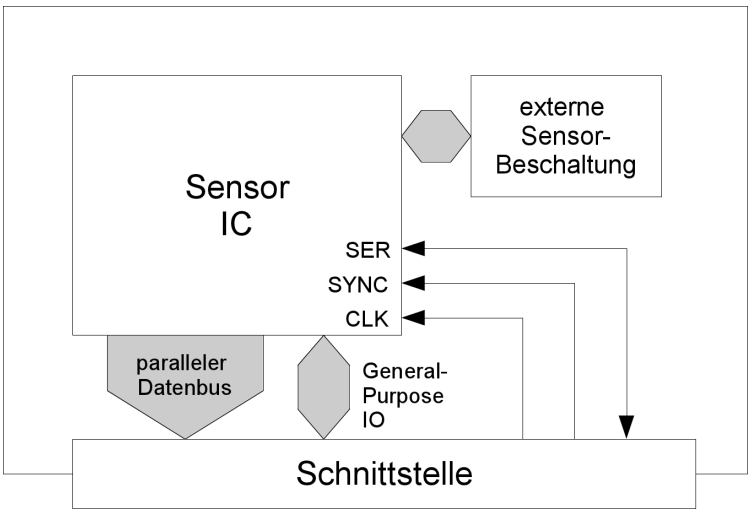


Abbildung 4.4: Schematischer Überblick über den logischen Aufbau eines Kamerasensormoduls

zur Anwendung hin vor, indem sie die Verteilung von intern oder extern generierten Synchronisationssignalen zu allen Modulen und zwischen den Kameras bereitstellt (siehe die Beschreibung des Datenübertragungsmoduls in Abschnitt 4.3.3).

4.3.1 Bilderfassungsmodule

Bilderfassungsmodule stellen die nicht-virtuellen Datenquellen des Systems dar. Sie umfassen damit in erster Linie Module mit verschiedenen Typen von Kamerasensoren, darüber hinaus aber auch Ausführungen mit Anschlussmöglichkeiten für externe Geräte (siehe unten). Die Bilderfassungsmodule beherbergen weiterhin die notwendige Außenbeschaltung der Kamerasensoren wie Hilfsspannungsgeneratoren, externe Kapazitäten etc. Zur Anbindung an die weiteren Hardwaremodule implementieren sie die proprietäre Intermodulschnittstelle. (Abbildung 4.4 zeigt den schematischen Aufbau eines Moduls.)

Die Belegung der physischen Schnittstelle mit logischen Signalen wird durch den jeweiligen Sensor-IC bestimmt. Sensor-ICs mit einem generischen 8 bzw. 16bit breiten Datenbusmodell mit Synchronisationssignalen und Konfiguration über einen I²C-Bus können direkt über das Datenübertragungsmodul angesprochen werden. Sensoren mit komplexeren Datenschnittstellen und zusätzlichen Signalleitungen erfordern eine passende Glue-Logic[6] zur Anbindung, die z.B. in der programmierbaren Logik

[6] Als Glue-Logic werden logische Schaltungen bezeichnet, die nicht direkt schnittstellenkompatible

des Bildverarbeitungsmoduls realisierbar ist.

Im Rahmen des MOSCOT-Projekts wurde eine Reihe von Bilderfassungsmodulen mit verschiedenen Kamerasensoren umgesetzt, von preiswerten VGA-Sensoren bis hin zu hochauflösenden Hochgeschwindigkeitssensoren. Abbildung 4.5 zeigt Umsetzungen des Bilderfassungsmoduls mit verschiedenen Kamerasensoren und Formfaktoren. Optional können für eine Anbindung von externen Kameras auch spezielle Bilderfassungsmodule mit Videoinputprozessoren (z.B. für analoge Kameras) oder USB-Host-ICs für USB-Kameras eingesetzt werden.

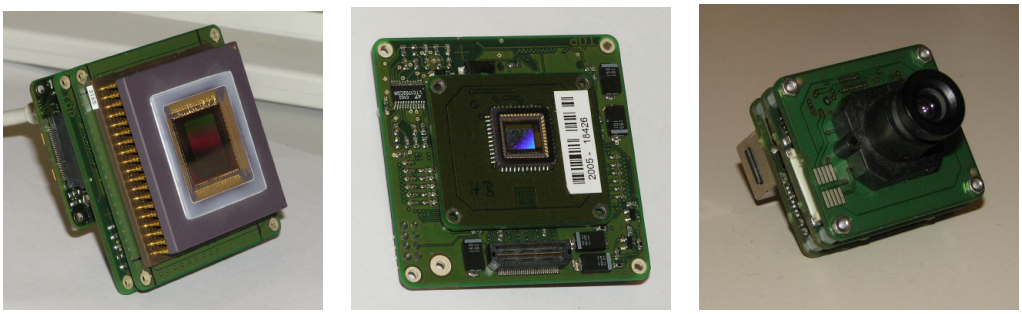


Abbildung 4.5: Beispiele für Bilderfassungsmodule mit verschiedenen Kamerasensor-ICs (rechts ein Modul mit integriertem Platinenobjektiv)

4.3.2 Bildverarbeitungsmodule

Zum Aufbau spezialisierter Hardwarekomponenten für Bildverarbeitungsaufgaben stehen zwei grundlegend unterschiedliche Technologien bereit: FPGAs (Field Programmable Gate Arrays) und DSPs (Digitale Signalprozessoren) [Kisačanin et al. 2009; Meyer-Baese 2004].

Field Programmable Gate Arrays

FPGAs (Field Programmable Gate Arrays) sind programmierbare Logikbausteine. Sie lassen sich über sog. Hardwarebeschreibungssprachen (HDL) konfigurieren und verhalten sich dann wie Spezialprozessoren. Damit verknüpfen sie die Flexibilität einer Softwarelösung mit der Geschwindigkeit von spezialisierter Hardware. FPGAs eignen sich vorzüglich für Bildverarbeitungsanwendungen, bei denen sehr geringe Latenzzeiten erforderlich sind. Durch ihre konfigurierbare Hardwareschnittstelle können sie direkt an die Kamerasensorbausteine angeschlossen werden und Bildverarbeitungsoperationen on-the-fly auf dem Bilddatenstrom vornehmen. FPGAs können

Baugruppen miteinander verbinden.

Bildverarbeitungsoperationen durch zwei Strategien beschleunigen, die auf gewöhnlichen General-Purpose-CPUs nicht zur Verfügung stehen: Pipelining und Parallelisierung. Bei der Parallelisierung werden verschiedene Bildbereiche durch identische Verarbeitungsstufen im FPGA parallel verarbeitet, Pipelining hingegen ermöglicht die Aufteilung der Datenverarbeitung in mehrere aufeinanderfolgende Stufen. Dadurch kann mit jedem Taktzyklus ein neues Datenwort angenommen und verarbeitet werden, bevor das vorherige Datenwort alle Stufen durchlaufen hat. Je nach Aufgabenstellung werden diese beiden Techniken auch kombiniert. Auf dem FPGA integrierter Speicher steht als Cache für die schnelle Speicherung von Zwischenergebnissen zur Verfügung. FPGA-basierte Bildverarbeitungssysteme sind weitverbreitet und werden vielfach in intelligenten Kameramodulen eingesetzt. Nachteilig ist die Programmierung über HDL, die deutlich aufwendiger und langwieriger ist als reine Softwareentwicklung z.B. für einen Signalprozessor.

Digitale Signalprozessoren

DSPs sind Spezialprozessoren, deren innere Struktur als Harvardarchitektur ausgelegt ist und damit eine effiziente Datenverarbeitung bei hohem Datendurchsatz ermöglicht. Wesentlich ist das Vorhandensein mehrerer Busse und spezieller Befehle, die zentrale Rechenoperationen in einem Taktzyklus ermöglichen (Multiply-Accumulate MAC). Durch angepasste Programme kann so eine hohe Auslastung aller Rechenwerke und Datenbusse erreicht werden. Häufig exisistieren auch schnelle integrierte Speicher für Zwischenergebnisse, so dass in jedem Taktzyklus ein Datenwort bearbeitet werden kann (Pipelining). Im Unterschied zu FPGAs werden hier keine Hardwarestrukturen an die spezielle Aufgabenstellung angepasst, dafür kann die Programmierung konventionell in Assembler bzw. C erfolgen. DSPs verfügen über generische Schnittstellen zur Anbindung externer Datenquellen, so dass viele Arten von Kamerasensoren direkt angebunden werden können. Falls dies durch Unterschiede in Schnittstelle oder Protokoll nicht möglich ist (insbesondere bei Hochleistungsbildsensoren) können zusätzliche Logikbausteine (Glue-Logic) für die Anbindung nötig werden.

Aufbau eines FPGA-Bildverarbeitungsmoduls

Für die erste Realisierung von Bildverarbeitungshardware im Rahmen des MOSCOT-Projekts wurde FPGA-Technologie ausgewählt, da sie flexibler in der Anbindung verschiedener Arten von Kamerasensoren ist und auch große Bildsensoren mit breitbandigen Schnittstellen direkt ansteuern kann. Die zunehmende Integration von DSP-Einheiten in FPGAs erlaubt zukünftig auch die leichte Erweiterung auf eine Kombination beider Technologien in einem FPGA. Letzlich fiel die Wahl auch auf FPGAs, da in der Arbeitsgruppe bereits Erfahrung mit ihrem Einsatz vorhanden war.

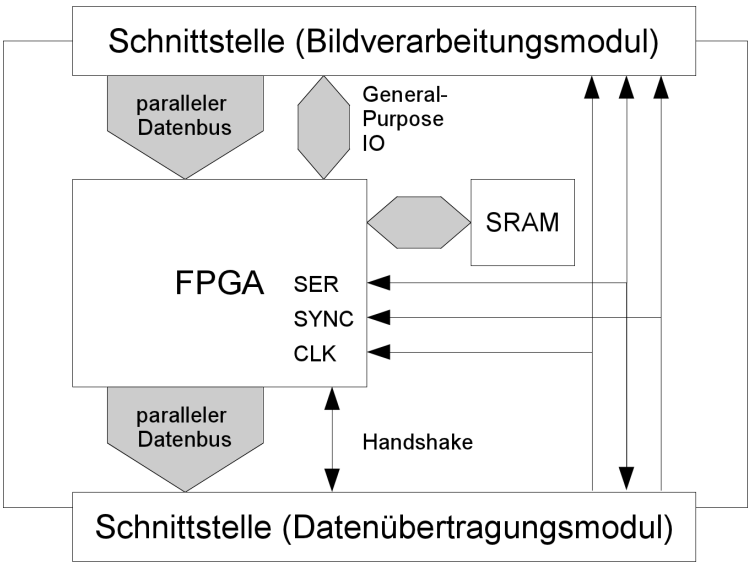


Abbildung 4.6: Schematischer Überblick über den logischen Aufbau eines FPGA-Bildverarbeitungsmoduls

Abbildung 4.6 zeigt das schematische Blockschaltbild eines Bildverarbeitungsmoduls mit FPGA. Angebunden an den zentralen FPGA-Baustein ist ein schneller SRAM-Speicher (z.B. für das Ablegen von Lookuptabellen). Des weiteren muss das Modul die Spannungs- und Taktversorgung des FPGAs bereitstellen. Die Anbindung an die Bilderfassungs- und Datenübertragungsmodule erfolgt wieder über die proprietäre Schnittstelleodule direkt mit der Kommunikationseinheit zu koppeln. Für komplexere Kamerasensoren besteht die Möglichkeit zusätzliche Datenverbindungen über General-Purpose-IO des FPGA zu schalten.

Abbildung 4.7 zeigt das innere Blockschaltbild eines FPGA-Designs zur Markerdetektion. Man sieht die Komponenten für die Ansteuerung des SRAMs, die Kommunikation mit dem Schnittstellenmodul, die Anbindung des Kamerasensors sowie das eigentliche Bildverarbeitungsmodul. Letzteres umfasst die komplette Datenverarbeitungskette aus Bildvorverarbeitung, Pixelklassifizierung, Filterung und Connected-Component-Labeling und ist direkt an die Kameraschnittstelle angebunden. Die Pipeline verarbeitet die Pixeldaten direkt ohne Zwischenspeicherung der Bilder, lediglich zur Ausführung von nxn-Faltungsoperatoren werden die letzten n-1 Zeilen in einem Zeilencache vorgehalten.

Die FPGA-Implementierung muss ausreichend schnell sein, um Pixeldaten im Kameraauslesetakt durch die Datenverarbeitungspipeline zu schieben. Die Gesamtlatenz setzt sich damit zusammen aus der Kameraauslesezeit für das Gesamtbild, einer Ver-

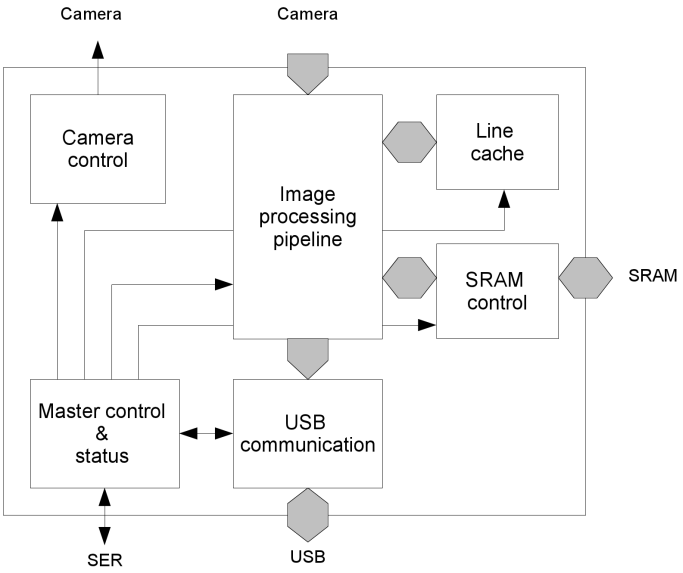


Abbildung 4.7: Schematischer Überblick über den internen logischen Aufbau des FPGA-Designs. Die wesentlichen Funktionsblöcke sind die Bildverarbeitungspipeline mit Zeilencache und Speicheranbindung, die USB-Kommunikation, die Kamerasteuerung und die zentrale Steuer- und Kontrolleinheit.

zögerung von n-1 Zeilen, bevor die Ausgabe aus dem Zeilencache startet[7] und die eigentliche Verarbeitungslatenz der Datenpipeline. Dabei sind sowohl die Verzögerung durch den Zeilencache (einige tausend Pixeltakte) als auch die Verarbeitungslatenz selbst (wenige Pixeltakte) vernachlässigbar gegenüber der Kameraauslesezeit (Million Pixeltakte bei einem Megapixelsensor). Damit wird der große Vorteil hardwarebasierter Bildverarbeitung deutlich, die die Verarbeitungszeit fast vollständig in der Kameraausleselatenz verstecken kann (siehe Abbildung 4.8).

Die konkrete Implementierung der Bildverarbeitungsmodule, insbesondere des FPGA-Designs, ist nicht Gegenstand dieser Arbeit. Die Implementierung von Faltungsoperatoren und Punktoperatoren ist Standardtechnik FPGA-basierter Bildverarbeitung [Bailey 2011; Kisačanin et al. 2009; Dillinger 2004]. Veröffentlichungen, die die Implementierung eines Farbmarkertrackings auf FPGA-Basis beschreiben, existieren z.B. von Johnston et al. [2005a]; Johnston et al. [2005b], speziell zur Implementierung des Connected-Component-Labeling von Jablonski und Gorgon [2004]; Johnston und Bailey [2008]. Details der Umsetzung innerhalb des MOSCOT-Projekts können den Arbeiten von Yang [2012] und Hegner [2005] entnommen werden.

[7] Bei mehreren aufeinanderfolgenden Faltungsoperatoren müssen mehrere Zeilencaches eingesetzt werden.

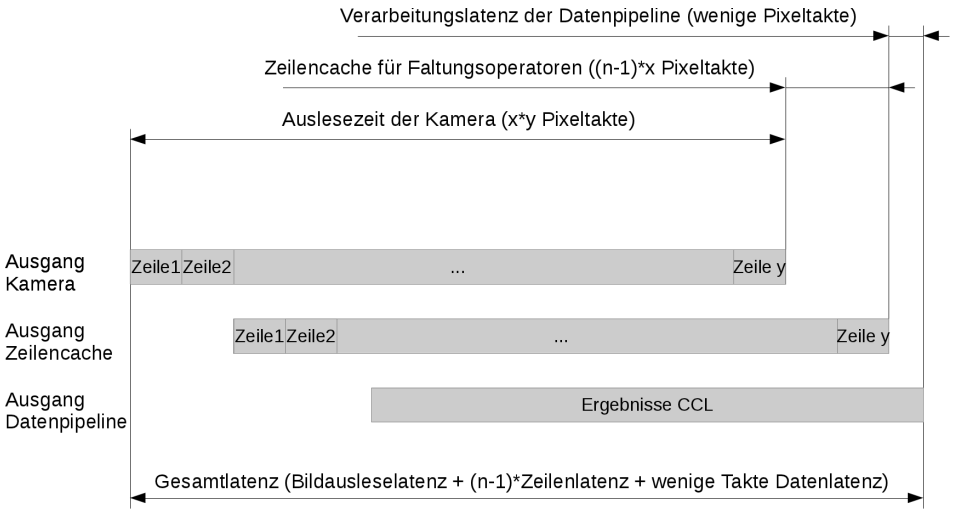


Abbildung 4.8: Aufteilung der Latenzzeiten der Hardwarebildverarbeitung: Bildauslesezeit, Zeilencache und Datenpipeline. Offensichtlich liegt die zusätzliche, über die Bildauslesezeit hinausgehende Latenzzeit der hardwarebasierten Bildverarbeitung im Bereich von wenigen Prozent der Bildausleselatenz.

4.3.3 Datenübertragungs- und Schnittstellenmodule

Die Datenübertragungs- und Schnittstellenmodule binden die Kamerahardware an den zentralen Host-PC an. Neben der Übertragung der Kameradaten müssen von diesen Modulen auch Steuerungsbefehle vom PC empfangen und weitergereicht werden. Als weitere Funktionalität verfügen sie über Schnittstellen zur externen Synchronisation der Kameras und stellen Takt- und Synchronisationssignale für die Bilderfassungs- und -verarbeitungsmodule bereit. Für die Unterstützung verschiedener Synchronisationsmethoden müssen die Module in der Lage sein, eine eigene Masterclock zu erzeugen und nach außen über Schnittstellen weiterzugeben, oder auch im Slavebetrieb ein externes Clocksignal entgegenzunehmen.

Durch die Integration passender Module können die MOSCOT-Kameraeinheiten an unterschiedliche Datenübertragungsstandards angepasst werden. Die wichtigste Schnittstellentechnologie zum Host ist sicherlich der Universal Serial Bus (USB) [USB Implementers Forum, Inc. 2000]. In der Arbeitsgruppe werden aber auch Module zur Kommunikation über CameraLink und Gigabit-Ethernet Kommunikation entwickelt [Yang 2012].

Abbildung 4.9 zeigt das Blockschaltbild eines Datenübertragungsmoduls zur USB-Kommunikation. Die Anbindung an die vorgelagerten Bilderfassungs- bzw. -verarbeitungsmodule erfolgt wieder über die beschriebene proprietäre Schnittstelle zur

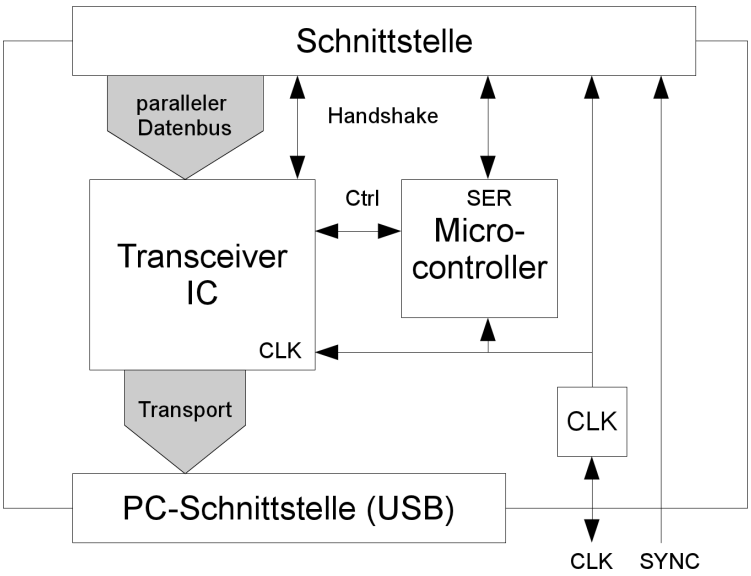


Abbildung 4.9: Schematischer Überblick über den logischen Aufbau eines Datenübertragungs- und Schnittstellenmoduls für USB-Anbindung

Intermodulkommunikation. Ein Mikrokontroller implementiert zusammen mit einem Treiberbaustein die USB-Anbindung und stellt einen I²C-Bus zu den weiteren Modulen bereit. Weiterhin stehen externe Schnittstellen für Takt- und Synchronisationssteuerung zur Verfügung.

4.3.4 Exemplarische Umsetzung der Hardwarearchitektur

Zur Veranschaulichung des Hardwaremodulkonzepts wird die konkrete Umsetzung einer intelligenten Trackingkamera dargestellt, die für das in Sektion 6.3 beschriebene Medizinrobotikprojekt „Intelligent Tool Drive (ITD)“ entwickelt wurde (Abbildung 4.10). Auf dem Bilderfassungsmodul kommt hier ein CMOS-Sensor der Firma Micron (MT9M413 [Micron Technology Inc. 2004]) zum Einsatz. Der Sensor hat eine Auflösung von 1280x1024 Pixeln, und erreicht dabei Bildwiederholraten von bis zu 400Hz. Abbildung 4.10 (links oben) zeigt den Aufbau der Platine. Zur Anbindung der breitbandigen Datenschnittstelle des Sensors sowie zur Datenreduktion in der intelligenten Kamera dient ein FPGA-basiertes Bildverarbeitungsmodul (Abbildung 4.10 mitte). Es integriert einen Xilinx Spartan 3 FPGA [Xilinx Inc. 2009], der die Busanbindung und die Bildvorverarbeitungsschritte implementiert, 4MiB SRAM und Schaltungen zur Generierung der Versorgungsspannungen. In [Hegner 2005] wird die Realisierung des Moduls und die Integration zu einer intelligenten Kamera beschrie-

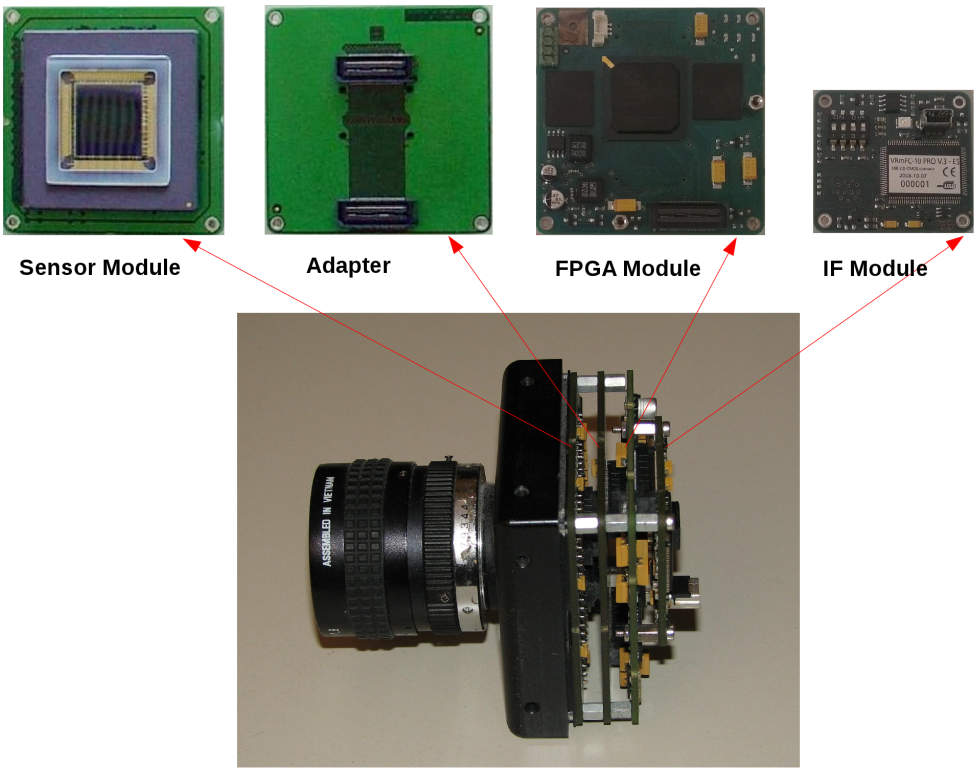


Abbildung 4.10: Beispiel der intelligenten Kamera für das Projekt ITD mit Bildsensor-, FPGA-Bildverarbeitungs- und USB-Kommunikationmodul sowie Teilgehäuse mit C-Mount-Objektiv

ben. Als Schnittstellenmodul kommt ein USB-Camera-Adapter-Board des Kooperationspartners VRmagic vom Typ UCA3 zum Einsatz (Abbildung 4.10 rechts). Dieses Modul basiert auf einem Cypress FX2 USB2.0-Microcontroller und stellt eine schnelle 16-bittige Schnittstelle für die Datenübernahme des Bildverarbeitungsmoduls zur Verfügung. Für Steuerungs- und Kontrollbefehle in Gegenrichtung stehen mehrere serielle I²C-Busse bereit.

Für den Aufbau der Kamera werden die Leiterplatten der einzelnen Modulgruppen über Platinensteckverbinder und Schraubverbindungen zu einem kompakten Gesamtaufbau übereinandergestapelt und in ein Teilgehäuse mit Objektivfassung integriert (Abbildung 4.10 unten). Die einzelnen Platinen haben ABmessungen von 62mm×62mm, so dass ein kompakter Kameraaufbau entsteht.

4.4 Realisierung der Softwarearchitektur

4.4.1 Einbettung in die VRM-Gesamtarchitektur

Die Softwarearchitektur von MOSCOT fügt sich als Baustein in die VRM-Architektur für medizinische Simulatoren [Wagner 2003, Kap. 2] ein. Diese bildet die softwaretechnische Basis der von VIPA-Arbeitsgruppe und VRmagic GmbH gemeinsam realisierten medizintechnischen Projekte. Sie untergliedert sich in eine Reihe von C++-Softwarebibliotheken, die jeweils Teilbereiche der Funktionalität zusammenfassen (z.B. optisches Tracking, physikalische Simulation, graphisches Rendering, etc.). Abbildung 4.11 gibt einen Ausschnitt der VRM-Bibliotheken wieder sowie ihre Anordnung auf verschiedenen funktionalen Ebenen. Das Gros der in diesem Kapitel aufgeführten Datenverarbeitungsstrukturen finden Eingang in die `Tracking`-Bibliothek. Komponenten der Bildverarbeitung sind der `Imageprocessing`-Bibliothek zugeordnet. Wichtige Hilfsfunktionen für die Trackingbibliothek existieren in den allgemeinen Datenstrukturen und Diensten der `Utils`-Bibliothek sowie mathematischen Klassen und Methoden der `Maths`-Biblothek.

Aufsetzend auf der obersten Bibliotheksschicht findet sich die Anwendungsschicht, also die Software des jeweiligen Simulators bzw. auch das Konfigurations- und Entwicklungsfrontend Tracklab. Für diese Anwendungsschicht werden spezielle Softwaremodule mit graphischer Nutzerschnittstelle bereitgestellt, beispielsweise für die Nutzerinteraktion bei Konfiguration und Betrieb der Trackingsysteme. Sie sind als eigenständige Tracking-GUI-Bibliothek ebenfalls Teil der VRM-Bibliotheken. Einige dieser Softwaremodule werden zusammen mit Tracklab im nachfolgenden Kapitel 5 beschrieben.

4.4.2 Anbindung an die Hardware

In einer Schicht unterhalb der Trackingbibliothek finden sich grundlegende Funktionen zur Datenkommunikation mit den Kameras in den Bibliotheken `Grabber, Hardware, Usb` und `Serial`. Darunter folgen die betriebssystemspezifischen Gerätetreiber zur Anbindung der Hardwarekomponenten (aktuell existieren Implementierungen für Linux und Windows). Die Treiber sind als schlanke Zwischenschicht ausgeführt, um den Umfang betriebssystemspezifischen Codes möglichst klein zu halten.[8] Die Datenschnittstelle selbst ist als Modul ausgelegt und für verschiedene

[8] Die Treiber sind im Unterschied zum Rest der Softwareimplementierung in der systemnahen Programmiersprache C implementiert.

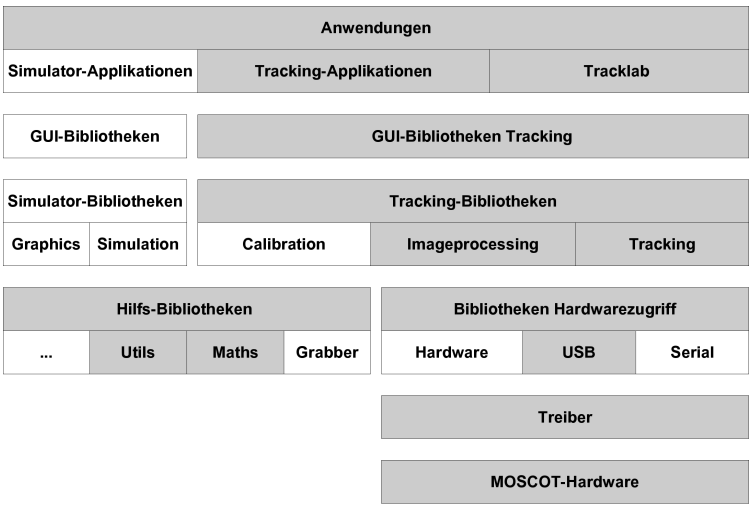


Abbildung 4.11: Einbettung der MOSCOT-Softwarekomponenten in die VRM-Architektur. Die Bibliotheken sind auf mehrere Ebenen aufgeteilt und bauen jeweils auf den tieferliegenden Schichten auf. Über den Bibliotheken folgt die Anwendungsschicht mit VR-Simulatoren und dem Softwarefrontend Tracklab. Die im Rahmen dieser Arbeit entwickelten Strukturen und Verfahren der MOSCOT-Systemarchitektur sind in den grau unterlegten Komponenten implementiert.

Übertragungsstandards austauschbar. Die wesentliche protokollspezifische Funktionalität wie Handshake, Fehlerbehandlung etc. sowie datenspezifische Übertragungsfunktionen sind in den genannten betriebssystemunabhängigen Kommunikationsbibliotheken implementiert.

Die Gesamtheit aus Hardware-Schnittstellenmodul, physikalischer Anbindung, Treibern und Kommunikationsbibliotheken kann aus Sicht der Datenflussarchitektur als transparente Kommunikationschnittstelle zwischen Hard- und Software angesehen werden. Aktuell ist die Anbindung von Kameras über USB und serielle Schnittstellen (nur Kamerakontrolle) implementiert, Varianten für Camera-Link und Gigabit-Ethernet sind in Entwicklung. Für die Zukunft sind (bei Realisierung entsprechender Schnittstellenmodule auf der Hardwareseite) auch weitere Übertragungstechnologien möglich.

4.4.3 Zusätzliche Anforderungen an die Softwarearchitektur

Die eingangs des Kapitels zitierte Anforderungsliste für das Gesamtsystem bildet auch die Grundlage für die Realisierung der Softwarearchitektur. Hinzu treten zu-

sätzliche Anforderungen, die spezifisch für die softwaretechnische Umsetzung der Systemarchitektur sind. Dazu gehören die Forderung nach Unabhängigkeit von Plattform bzw. Betriebssystem und Fragestellungen hinsichtlich des Zeitverhaltens:

Plattformunabhängigkeit

Die Forderung nach Unabhängigkeit von einer Betriebssystemplattform resultiert unmittelbar aus den intendierten Einsatzbereichen des MOSCOT-Systems. So werden die medizinischen Simulatoren der Arbeitsgruppe auf Standard-PC-Hardware unter Microsoft Windows betrieben. Andere Anwendungen (z.B. der in Abschnitt 6.3 beschriebene Operationsroboter ITD) erfordern von Beginn die Auslegung für eine spätere Portierung auf eingebettete Plattformen unter einem echtzeitfähigen Betriebssystemen wie z.B. ein Realtime-Linux (siehe unten). Mit der Unabhängigkeit von der zugrundeliegenden Systemplattform erweitert sich somit auch das Einsatzspektrum des Systems.

Echtzeitfähigkeit

Mit den Ausführungen aus Abschnitt 2.2.7 genügt eine weiche Echtzeitfähigkeit für Anwendungen in medizinischen Simulatoren, wobei Wagner [2003, Kap. 2.5] zeigt, dass diese für die speziellen Gegebenheiten in VR-Simulatoren auch von gewöhnlichen Betriebssystemen wie Windows oder Linux erfüllt wird. Anspruchsvoller sind Anwendungen wie das eben angeführte Positionstracking eines Operationsrobotern, die einer harten Echtzeitfähigkeit bedürfen und damit unter einem Echtzeitbetriebssystem ablaufen müssen. Während der Entwicklung solcher Anwendungen bietet ein Standard-Linux-System mit Low-Latency-Patches [Molnar 2000] eine ausreichende Basis, für die finale Ausführung muss jedoch die Portierbarkeit auf ein Echtzeitbetriebssystem (z.B. Realtime-Linux) sichergestellt sein.

Synchronisation

Der Synchronisation des Systems kommt auch auf der Softwareseite besondere Beachtung zu. Mit der in Abschnitt 2.2.7 geforderten Anpassbarkeit der Synchonisationsmethode an die Vorgaben der Anwendung muss die Softwarearchitektur intern einen vollständig synchronen Betrieb unterstützen und nur an den Schnittstellen zur Hardware bzw. zur Applikation je nach Anforderung zwischen synchronem und asynchronen Verhalten unterscheiden. Den Treibern kommt dabei für die Synchronisation wichtige Funktion zu. Je nach Betriebsart der Anwendung können sie synchrone bzw. asynchrone Transfers unterstützen. Die Anwendung kann damit den Treiber in einem blockierenden Modus benutzen, um sich auf die Verfügbarkeit von Daten im Takt der Trackinghardware zu synchronisieren, oder – falls ein asynchroner Betrieb von Tracking und Anwendung vorteilhaft ist – im nicht blockierenden Modus jeweils direkt mit den letzten verfügbaren Daten weiterarbeiten.

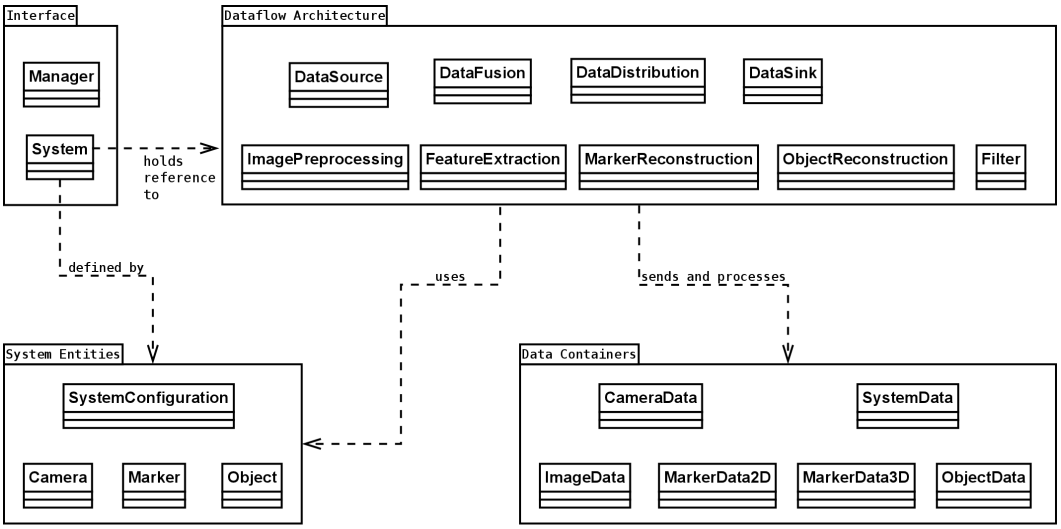


Abbildung 4.12: Gruppierung der wichtigsten Klassen der MOSCOT-Softwarearchitektur anhand ihrer Rolle beim Datenverarbeitungsprozess. Das System wird definiert über die Beschreibung seiner Entitäten und hält eine Referenz auf die Datenflussarchitektur. Diese wiederum transportiert und verarbeitet die Datenstrukturen unter Verwendung der Vorgaben aus der Systembeschreibung.

4.4.4 Überblick über die Realisierung

Allgemein folgt die Realisierung den Prinzipien objektorientierter Programmierung (OOP) [Booch et al. 2007] in der Programmiersprache C++ [Stroustrup 1997]. Entwurfsmusterbasierte Ansätze [Gamma et al. 2001] kommen ebenso wie generische Programierung zur Anwendung. Es wird auf grundlegende weitverbreitete Bibliotheken wie die C++-Standardbibliothek mit der Standard-Template-Library (STL) [Austern 1998] aufgebaut. Ausgewählte Teile der Boost-Bibliothek [Dawes und Abrahams 2000–2007] [Karlsson 2005], die eine Auswahl- und Evaluierungsplattform für zukünftige Aufnahmen in die C++-Standardbibliothek darstellt, werden ebenfalls eingesetzt. Das von der STL, aber auch von Boost verfolgte Konzept der Trennung von Daten und Algorithmen wird auch innerhalb der MOSCOT-Datenflussarchiktektur umgesetzt. Die Abstraktion von den zugrundeliegenden Betriebssystemplattformen Windows und Linux erfolgt durch die genannten plattformübergreifenden Bibliotheken und einige weitere Softwarebibliotheken, die betriebssystemspezifische Eigenheiten kapseln, z.B. die Qt-Bibliothek für graphische Benutzeroberflächen [Blanchette und Summerfield 2006; Digia Plc 2012a].

Objektorientierte Modellierung

Viele der allgemeinen Anforderungen an die Systemarchitektur werden auf der Softwareseite bereits durch die objektorientierte Modellierung umgesetzt. Sie unterstützt im Besonderen die zentralen Punkte Modularität, Wiederverwendbarkeit und Skalierbarkeit. Im Rahmen von objektorientierter Analyse und Design (OOAD) [Booch et al. 2007] sind elementare Strukturen zu identifizieren, die als Klassenobjekte modellierbar sind. Im Kontext der MOSCOT-Datenflussarchitektur bietet sich hierzu drei Arten von Elementen an, die den Elementen des Datenflussdiagramms aus Abbildung 3.2 entsprechen: Klassen, die das System selbst und die statischen Systemparameter beschreiben (System Entities – Abschnitt 4.4.5), Datenhaltungsklassen, die die dynamischen Datenstrukturen des Trackingprozesses vorhalten (Data Containers – Abschnitt 4.4.7) und Datenverarbeitungsklassen, die den Rahmen und die Komponenten der Datenflussarchitektur bereitstellen (Dataflow Architecture – Abschnitte 4.4.8ff). Abbildung 4.12 gibt einen Überblick über diese Aufteilung und die zugehörigen Klassen.

Für einen vereinfachten Zugriff auf die Trackingfunktionalität wurde die zentrale systembeschreibende Klasse `VtrkSystem`[9] als Fassadenentwurfsmuster [Gamma et al. 2001, S.212ff] für den Zugang zu allen trackingrelevanten Funktionen ausgelegt. Für übergeordneten Zugriff auf mehrere Systeme existiert noch eine allgemeine Verwaltungshilfsklasse `VtrkManager`, die als Singleton implementiert ist. Zusammen gewähren sie den Zugang zur Funktionalität der Trackingibliothek, die Steuerung der Systemkomponenten und Zugriff auf die Datenflussstrukturen und ihre Ergebnisse

4.4.5 Modellierung von System und Systemkonfiguration

Abbildung 4.13 gibt einen detaillierteren Überblick über die Klassen, die ein Trackingsystem innerhalb der Softwarearchitektur repräsentieren.[10] Die Klasse `VtrkSystem` stellt das zentrale Interface zum Ansprechen des Systems bereit. Ihre Attribute umfassen die Systemkonfiguration `VtrkSystemConfig`, die verketteten Instanzen der Datenflussarchitektur (Spezialisierungen von `VtrkTransceiverBase`), sowie Spezialisierungen der Klasse `VtrkDeviceBase` als Datenquellen. Die Systemkonfiguration fasst die statischen Informationen eines Trackingsystems in einer Da-

[9] Da die Ursprünge der VRM-Architektur auf eine Zeit zurückgehen, als noch nicht alle gängigen Compiler C++-Namensräume vollständig unterstützten, sind Namensräume innerhalb der VRM-Architektur als Teil des Klassennamens in der Form VlibnameClassname codiert (bspw. VtrkSystem). Mit der Verwendung aktueller Compiler würde dieses Beispiel als vrm::trk::System codiert.

[10] Die UML-Klassendiagramme in diesem Kapitel zeigen aus Gründen der Übersichtlichkeit vereinfachte Darstellungen der realen Gegebenheiten. So können Klassen zusammengefasst, Beziehungen vereinfacht oder Hilfsklassen, Attribute und Methoden ganz ausgelassen werden.

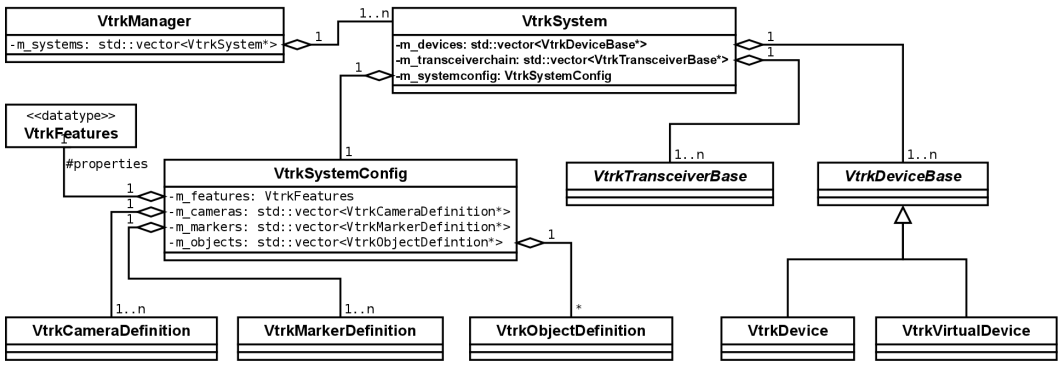


Abbildung 4.13: Klassenstruktur von System und Systemkonfiguration

tenstruktur zusammen. Ihre Attribute sind Klassen mit den in Tabelle 3.2 aufgeführten Systemparametern (`VtrkCameraDefinition`, `VtrkMarkerDefinition`, `VtrkObjectDefinition`). Zusätzlich beschreibt das Attribut `VtrkFeatures` weitere Eigenschaften des Systems, die etwa zum Aufbau der Datenverarbeitungskette notwendig sind. Eine Instanz von `VtrkSystemConfig` steht allen am Datenverarbeitungsprozess beteiligten Modulen als referenzgezählter unveränderbarer Datensatz zur Verfügung, so dass die Konfigurationsinformationen in der gesamten Datenflussarchitektur verfügbar sind. `VtrkDeviceBase` und `VtrkTransceiverBase` sowie ihre Spezialisierungen werden in den Abschnitten 4.4.8ff näher beschrieben.

Die Persistenz der zu einem System gehörigen Systemkonfiguration wird sichergestellt durch Serialisierung im Dateisystem des Host-PCs.[11] Damit ist auch eine einfache Austauschbarkeit von Teilen der Konfiguration zwischen verschiedenen Systemen gegeben, z.B. Kameradefinitionen zur Basiskalibrierung identischer Systeme oder Marker- und Objektdefinitionen für die Übernahme eines Trackingobjekts in ein anderes Trackingsystem.

4.4.6 Entwurf der Datenflussarchitektur

Die Modellierung der dynamischen Strukturen des Datenflusses stellt den Kern der MOSCOT-Softwarearchitektur dar. Konzept und wesentliche Komponenten werden in den folgenden Abschnitten beschrieben. Zuvor werden ähnliche Datenverarbeitungsstrukturen in der Literatur untersucht.

[11] Für spezielle Anwendungen unterstützt die MOSCOT-Systemarchitektur auch zusätzlich die Persistenz von Teilen der Systemkonfiguration in den systemeigenen Hardwaremodulen (z.B. Kamerakalibrierung oder Markerdefinitionen in intelligenten Kameras eines Simulators).

Stand der Technik

Die Verarbeitungsschritte des Trackingsprozesses sind in Form einer fortlaufenden Kette von Datenflussmodulen organisiert. Entwurfsmuster für derartige Datenverarbeitungsstrukturen treten in der Literatur unter verschiedenen Termini auf. Schmidt et al. [2000] beschreiben das Interceptor-Entwurfsmuster als eine Möglichkeit, einen Verarbeitungsprozess transparent durch neue Methoden zu erweitern, unabhängig von den übrigen Verarbeitungsschritten. Buschmann [1996] führt die gesamte Verarbeitungskette als „Pipes and Filters“ (Abbildung 4.14) ein, wobei die Pipes Verbindung und Weiterleitung der Daten zwischen aufeinanderfolgenden Filtern übernehmen. Die Filter sind feingranulare unabhängige Datenverarbeitungseinheiten mit einheitlicher Schnittstelle, die Wiederverwendung und Neukombination ermöglicht. Es gibt keine übergeordnete Kontrollinstanz. Java kennt ähnliche Entwurfsmuster als Intercepting Filter [Alur et al. 2003] bzw. Decorating Filter [Bien 2003], welche hauptsächlich für das Prä- und Postprocessing von Webserveranfragen eingesetzt werden. Während die Verarbeitungskette einer Pipes and Filters-Struktur ähnelt, erfolgt die Verwaltung hier zentral über eine Filtermanagerinstanz, die den Datenfluss kontrolliert.

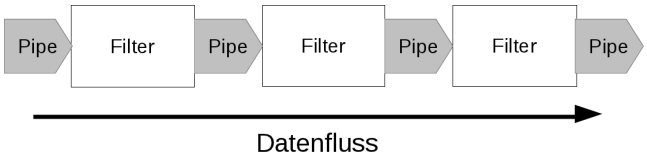


Abbildung 4.14: Schematische Darstellung der Struktur des „Pipes und Filter“-Entwurfsmuster zum Aufbau flexibler Datenflussarchitekturen

Das FXEngine-Projekt [SMProcess 2011] stellt ein Beispiel für die Umsetzung einer plugin-basierten Datenflussarchitektur dar. Es stehen Datenquellen, Datensenken und Verarbeitungseinheiten bereit und verschiedene vordefinierte Datentypen werden unterstützt. Die Verwaltung und Konfiguration der Verarbeitungsstrukturen erfolgt auch hier über eine zentrale Verwaltungsklasse. Ein weiteres Projekt, welches auf einer modularen Datenflussarchitektur aufbaut, ist das Insight Toolkit (ITK) [Ibanez et al. 2005] für Segmentierung und Registrierung medizinischer Bilddaten. Datenobjekte werden hier von Verarbeitungsfiltern bearbeitet, die in einer rekonfigurierbaren Pipelinestruktur angeordnet sind. Die Abarbeitung der Ereignisse wird über ein Command/Observer-Entwurfsmuster [Gamma et al. 2001, S.273ff] realisiert.

Konzeption der MOSCOT-Datenflussarchitektur

Ähnlich den beschriebenen Architekturmustern realisiert auch die Datenflussarchitektur des MOSCOT-Systems eine Kette aneinandergereihter, voneinander unabhän-

giger Datenverarbeitungsmodule. Die resultierende Entkopplung der unterschiedlichen Funktionsbereiche vereinfacht Wiederverwendung und Rekombination der Module. Im Unterschied zu den meisten der vorgestellten Referenzen ist jedoch keine zentrale Kontrollinstanz vorhanden, die die Datenverarbeitung in den verschiedenen Stufen organisiert bzw. die Datenpakete weiterreicht. Vielmehr wird während der Initialisierung des Systems ausgehend von den Datenquellen eine passende Struktur von miteinander verketteten Datenverarbeitungsmodulen aufgebaut. Der Datentransport wird von den Datenquellen aus angestoßen: speziellen Datenflussmodulen, die den Trackinggeräten zugeordnet sind und direkt von diesen mit Daten beschickt werden. Je nach den von den Hardwareverarbeitungseinheiten bereits ausgeführten Verarbeitungsschritten speisen die Geräte entsprechende Datenstrukturen in die Kette der Verarbeitungsmodule ein. Diese verarbeiten die für sie relevanten Daten und reichen die Datenpakete an das nächste Modul in der Kette weiter. Da keine zentrale Instanz den Datenfluss organisiert, existieren spezielle Module zur lokalen Steuerung des Datenflusses. Sie verfügen wie die Datenquellen über keine Datenverarbeitungsfunktion im eigentlichen Sinne, sondern realisieren Verwaltungsfunktionen wie Verteilung, Fusion und Synchronisation von Datenpaketen.

Die Konfiguration und Initialisierung der einzelnen Datenverarbeitungsmodule wird ebenfalls dezentral über die Datenflussarchitektur selbst realisiert. Es ist naheliegend, hierfür dieselben Kommunikationsstrukturen einzusetzen, die auch die Trackingdatensätze transportieren. Diese parallele Nutzung der Datenflussarchitektur für die Propagierung von Kontroll- und Systemnachrichten (neben dem Transport der reinen Trackingdaten) ist ein neuer Aspekt der MOSCOT-Systemarchitektur im Unterschied zu bestehenden Datenflussarchitekturen. Gleichzeitig eröffnet sich damit eine Reihe zusätzlicher Möglichkeiten: Benachrichtigungen über veränderte Systemkonfiguration ermöglichen die Rekonfiguration einzelner Datenverarbeitungsmodule (z.B. zur Nachkalibrierung der Kameras durch Bündelblockausgleichung, Hinzunahme neuer Objekte oder Generierung neuer Lookuptabellen bei Änderung der Markerfarbdefinition). Neben der Systemkonfiguration können auch weitere Ereignisse wie Änderungen des Systemzustands (Start und Stop des Trackingvorgangs) kommuniziert werden, um einzelne Module ein Prä- und Postprocessing durchführen zu lassen (z.B. zum Zurücksetzen von Filtern oder Auswerten gesammelter Datensätze).

Eine weitere Besonderheit ergibt sich bei der Organisation der Daten in den Containerklassen, die für den Datentransport eingesetzt werden. Die vier grundlegend verschiedenen Repräsentationsformen der Trackingdaten aus Tabelle 3.1 werden in zwei verschiedenen Containerstrukturen zusammengefasst, je nachdem, ob sie einer Kamera (Bilddaten, 2D-Markerdaten) oder dem ganzen System (3D-Markerdaten, Objektdaten) zugeordnet sind. Um auch Datenverarbeitungsmodulen in der Applikationsschicht (außerhalb der VRM-Bibliotheken) Zugriff auf den gesamten Datenverlauf zu gewähren, werden die Datensätze in den Containern nicht überschrieben,

sondern akkumuliert. Bei Daten desselben Typs (z.B. Bilddaten) wird so eine Historie der Datensätze gebildet, so dass Zugriff auf alle Zwischenergebnisse besteht. Diese Maßnahme ist wichtig für die Realisierung der externen Entwicklungsumgebung Tracklab, die Zugriff auf die internen Daten des Trackingprozesses erfordert, ebenso aber auch für die Serialisierung ausgewählter Zwischenschritte als Datenbestand für virtuelle Systeme.[12]

Die detaillierte Beschreibung in den folgenden Abschnitten umfasst damit die Ereignis- und Datenhaltungsklassen (4.4.7), die Realisierung der verketteten Datenflussmodule (4.4.8), Module der Datenflussorganisation wie z.B. Datenquellen (4.4.9) und schließlich die Implementierung der Datenverarbeitungsmethoden selbst (4.4.10).

4.4.7 Modellierung der Ereignis- und Datencontainerstrukturen

Wie beschrieben soll die Datenflussarchitektur neben den eigentlichen Datensätzen auch Konfigurations- und Statusinformationen zu den einzelnen Datenflussmodulen verteilen. Dazu wird eine Basisklasse `VtrkEvent` definiert. Sie stellt Zugriffsmethoden sowie einen Eventheader mit Verwaltungsinformationen zur korrekten Verteilung und Abarbeitung der Ereignisse durch die Datenflussmodule bereit. Die weitere Spezialisierung der `VtrkEvent`-Basisklasse unterteilt sich dann in Ereignisklassen und Datenhaltungsklassen (Abbildung 4.15).

Datenklassen beinhalten die Trackingdatensätze einzelner Kameras oder des ganzen Systems. Die Klasse `VtrkCameraData` repräsentiert den Datensatz einer Kamera mit Bild und 2D-Markerdaten. `VtrkSystemData` beinhaltet die zusammengefassten Daten des Systems (2D-Markerdaten aller Kameras und nach der Rekonstruktion 3D-Markerdaten sowie Objektposen). Die Datensätze selbst werden jeweils auf Stapelspeicher abgelegt, so dass z.B. ein neuer Satz 3D-Daten aus einer Filterstufe den letzten Datensatz überdeckt. Über spezielle Methoden besteht jedoch weiterhin Zugriff auf die Ergebnisdaten vorgelagerter Verarbeitungsstufen. Als Zugriffsschutz-Proxy-Muster sind Varianten der Datenhaltungsklassen `VtrkConstCameraData` bzw. `VtrkConstSystemData` mit eingeschränkten Zugriffsrechten implementiert, so dass die enthaltenen Daten

[12] Die Akkumulation bereits verarbeiteter Daten für externe Datenverarbeitungsmodule scheint zunächst der Forderung nach Datenreduktion zu widersprechen. Da diese Daten auf Seiten der Softwarearchitektur jedoch ohnehin im Speicher vorgehalten werden, wirkt sich diese Designentscheidung nicht nachteilig aus, solange sichergestellt ist, dass nur Referenzen der Daten und keine Kopien weitergereicht werden (vgl. die Implementierungsdetails in Abschnitt 4.4.11).

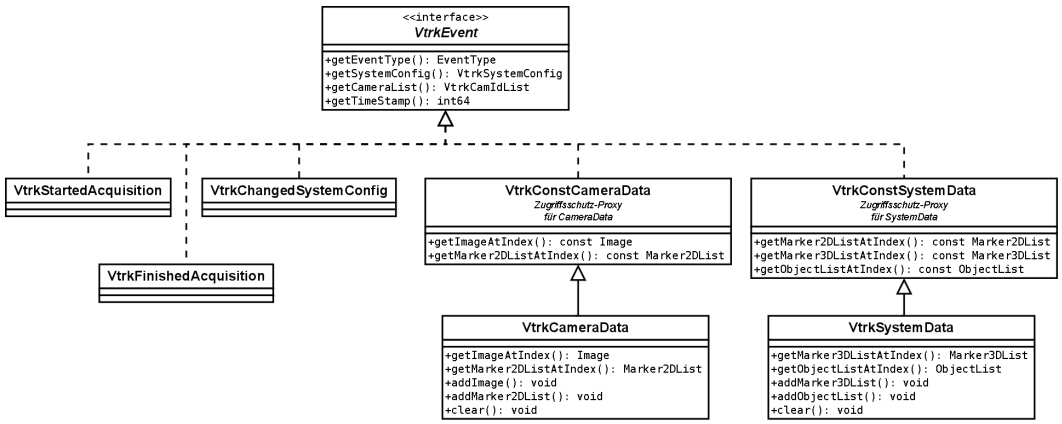


Abbildung 4.15: Klassenstruktur der Unterklassen für Datencontainer und Ereignisse

nicht verändert werden können. Damit können die Schreibrechte der Datenflussmodule kontrolliert werden.[13]

Ereignisklassen werden nur als Benachrichtigungstypen ohne eigene Atribute eingesetzt. `VtrkStartedAcquisition` und `VtrkFinishedAcquisition` signalisieren Beginn und Ende eines Trackingsvorgangs. `VtrkChangedSystemConfig` propagiert Änderungen des Systemzustands an die Module.

Beim Erstellen und beim Transport der Datenstrukturen durch die Verarbeitungskette ist Sorge zu tragen, dass die Ressourcenanforderungen und notwendigen Latenzzeiten möglichst niedrig bleiben. In Abschnitt 4.4.11 werden dazu allgemeine Maßnahmen beschrieben, die insbesondere für die Implementierung der Datencontainerklassen Relevanz besitzen.

4.4.8 Modellierung der Datenflussmodule

Die grundlegende Funktionalität zur Verkettung der Datenflussmodule ist in der Basisklasse `VtrkTransceiverBase` implementiert. Ähnlich wie viele der Architekturmuster aus Abschnitt 4.4.6 basiert auch der Kern der Implementierung bei MOSCOT auf einem modifizierten Decorator-Entwurfsmuster [Gamma et al. 2001, S.199ff]. `VtrkTransceiverBase` implementiert als Decorator die Schnittstellenklasse `VtrkTransceiverInterface` und hält gleichzeitig eine Referenz auf das nächste Objekt, welches diese Schnittstelle implementiert (Abbildung 4.16). In der Decorator-

[13]Da sich beide Klassen nur durch die Schreibzugriffsmethoden unterscheiden, wird hier einfache Vererbung einem vollständigen Proxy-Entwurfsmuster mit zusätzlicher abstrakter Schnittstellendefinition vorgezogen.

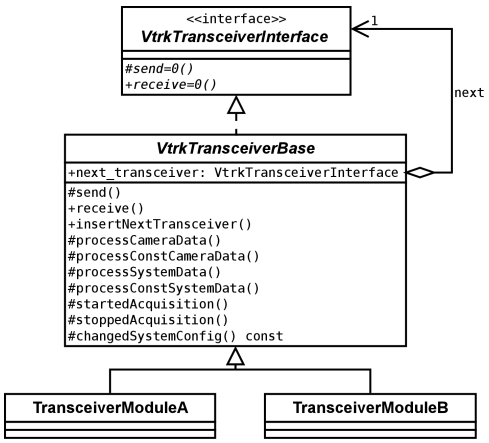


Abbildung 4.16: Schnittstellenklasse und Basisklasse der Datentransceiver für den Aufbau einer verketteten Datenflussstruktur. Die eigentliche Datenverarbeitungsfunktion wird in den Kindklassen implementiert, angedeutet als TransceiverModuleA/B.

Klasse sind die Grundfunktionen zur Verkettung mit anderen `VtrkTransceiverBase`-Objekten realisiert. Alle weiteren Datenflussmodule leiten von `VtrkTransceiverBase` ab und implementieren ihre Verarbeitungsfunktionalität in virtuellen Methoden. Es existieren Elementfunktionen für die Reaktion auf Konfigurationsänderungen und Systemereignisse (`startedAcquisition, stoppedAcquisition, changedSystemConfig`). Weitere Methoden implementieren die modulspezifische Datenverarbeitung der Module, unterschieden nach Kameradaten und Systemdaten. Reiner Lesezugriff wird über getrennte Elementfunktionen in Zusammenhang mit den Zugriffschutzproxies der Datenhaltungsklassen angeboten (`process***Data`).

Abbildung 4.17 veranschaulicht die Interaktion der Datemflussmodule anhand eines Sequenzdiagramms. Das Durchreichen der Ereignisse (Spezialisierungen von `VtrkEvent`) an das jeweils nächste Modul lässt eine fortlaufende Kette von Verarbeitungsknoten entstehen. Die `receive`-Methode des empfangenden Moduls ruft die zum Ereignis passende Methode auf, bevor sie ihrerseits das verarbeitete Ereignis über einen `send`-Aufruf weiterreicht. Der Ausgangspunkt der Kette liegt in der Datenquelle, das Ende der Kette ist in den Datensenken erreicht.

Verkettung der Datenflussmodule

Die folgende Übersicht ordnet die Datenflussmodule in der Reihenfolge ihres Aufrufs entlang der Datenflusskette an. Die Reihenfolge der Modulanordnung entspricht den Datenverarbeitungsschritten aus Kapitel 3. Zusätzlich zu den eigentlichen Datenverarbeitungsmodulen existieren Module, die keine Daten verarbeiten, sondern lediglich

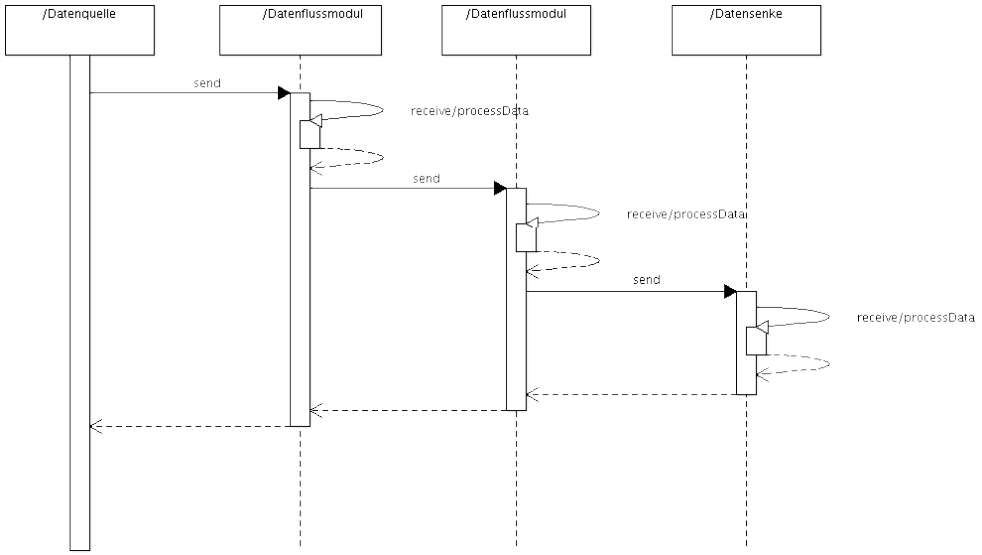


Abbildung 4.17: Aufrufmuster der Kette der Datentransceiver. Aus der `send`-Methode des sendenden Transceiver-Objekts wird `receive` des nachfolgenden Datenflussmoduls aufgerufen. Diese führt je nach Art des Ereignises die entsprechende spezialisierte Elementfunktion (`process***Data`, etc.) aus, bevor ein `send`-Aufruf das nächste Ergebnis an das nächste Modul weiterreicht.

eine Datenflusssteuerung ausführen (DFS). Einige Module sind als optional gekennzeichnet, da ihre Funktionalität entweder schon in den Hardwareeinheiten ausgeführt wurde, oder sie nicht für alle Anwendungsbereiche erforderlich sind (z.B. Filter).

- (DFS) Datenquellen
- Bildvorverarbeitung (optional)
- Merkmalsextraktion (optional)
- Verzeichnungskorrektur (optional)
- Filter-2D (optional)
- (DFS) Fusion der Kameradaten
- Markerrekonstruktion
- Filter-3D (optional)
- Objektrekonstruktion
- Filter-6DOF (optional)

- (DFS) Datendistribution (optional)
- (DFS) Datenserialisierung (optional)
- (Applikation)

Alle Module sind als abgeleitete Klassen von `VtrkTransceiverBase` implementiert. Abbildung 4.18 gibt einen Überblick unterteilt in Datenverarbeitungsklassen und Klassen zur Datenflusssteuerung.

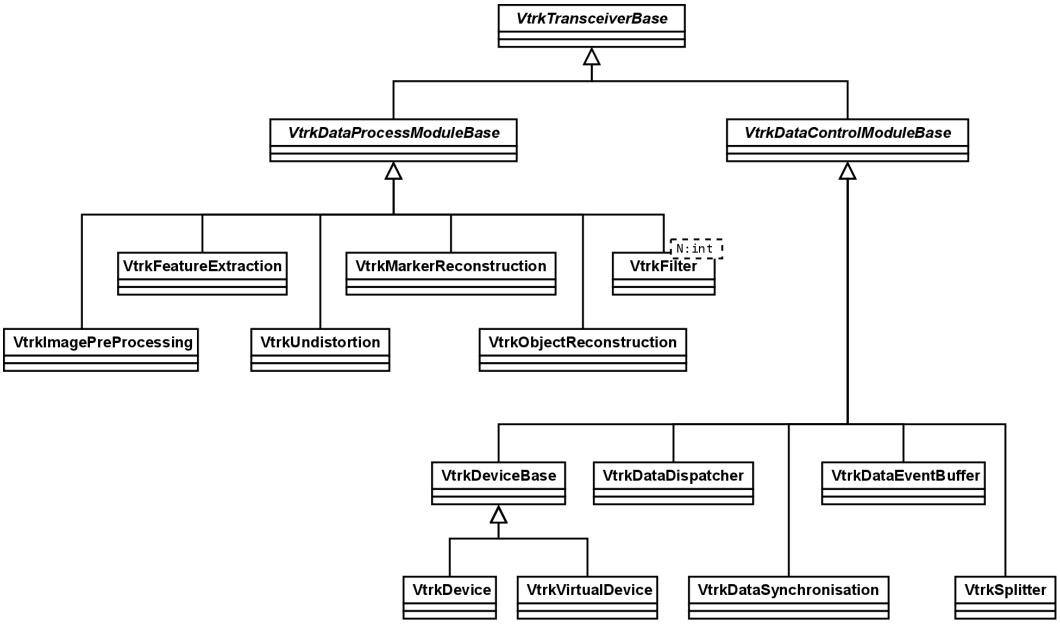


Abbildung 4.18: Datentransceiver-Basisklasse und die abgeleiteten Klassen für Datenverarbeitung und Datenflusssteuerung

4.4.9 Module zur Steuerung des Datenflusses

In diesem Abschnitt werden die Module zur Steuerung des Datenflusses, d.h. ohne Datenverarbeitungsfunktionen, beschrieben. Neben den Datenquellen sind dies Fusions- und Verzweigungspunkte des Datenflusses sowie die Datensenken, die die Datenpakete serialisieren, visualisieren oder an die Anwendungsschicht weiterreichen (Abbildung 4.19).

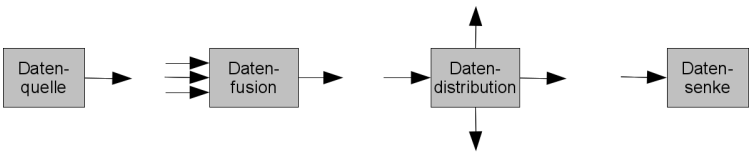


Abbildung 4.19: Die verschiedenen Module zur Steuerung des Datenflusses im schematischen Überblick: Quellen, Senken, Verteilung und Aggregation.

Datenquellen

Datenquellen aus Sicht der Softwarearchitektur sind die Einspeisungspunkte der Daten in die Kette der VtrkTransceiver-Module.[14] Eingespeiste Daten sind die von der MOSCOT-Hardware empfangenen Datensätze bzw. generierte oder serialisierte Daten für die in Abschnitt 4.2.5 vorgestellten virtuellen Systeme.

Die Implementierung basiert auf der abstrakten Basisklasse `VtrkDeviceBase`, die die Schnittstelle für alle Klassen mit Datenquellenfunktionalität deklariert. Instanzen der hiervon abgeleiteten Klassen `VtrkDevice` bzw. `VtrkVirtualDevice` implementieren die Funktionalität zur Einspeisung der Daten. Bei realen Kameras wird eine Instanz von `VtrkCameraData` mit den von den darunterliegenden Schichten bereitgestellten Daten der Hardwareeinheiten (Rohbilder, vorverarbeitete Bilder oder Listen von 2D-Markerdaten) befüllt und über die `send`-Methode verschickt. Bei virtuellen Geräten werden Daten nach einem vorgegebenen Schema generiert oder aus Dateien eingelesen, wobei Datensätze aller Verarbeitungsstufen möglich sind. Je nach Art der Daten und des virtuellen Geräts werden die Daten als `VtrkCameraData`-Objekte (bei Bilddaten oder 2D-Markerdaten einer virtuellen Kamera) bzw. als `VtrkSystemData`-Objekte (bei 3D-Markerdaten oder Objektdaten eines virtuellen Systems) in die Verarbeitungskette eingespeist. Durch die Serialisierung können virtuelle Systeme die aufgezeichneten Daten eines realen Systems mit identischer Systemkonfiguration erneut verarbeiten. Da sie ähnlich einem Software-Debugger komplett steuerbar sind durch Pausieren und schrittweises Fortschalten des Trackingprozesses, besteht so die Möglichkeit einer einfachen Inspektion und Verifikation der Verarbeitungsschritte während der Entwicklung neuer Trackinganwendungen. Virtuelle Systeme können auch das Zeitverhalten realer Systeme nachbilden, ebenso wie deren verschiedene Synchronisationsmethoden.[15]

Der Mechanismus virtueller Systeme ermöglicht es der MOSCOT-Systemarchitektur auch, Kameras oder Trackingsysteme von Drittanbietern zu integrieren. Dabei wer-

[14] Im Unterschied hierzu stellen die Kameras die Datenquellen für das Gesamtsystem dar.

[15] Dies ist natürlich nur gewährleistet, falls die PC-Basis die zeitlichen Anforderungen der Originaldaten erfüllen kann. Evtl. kann daher auch eine Verlangsamung der Dateneinspeisung sinnvoll sein.

den die Daten der externen Hardware über einen Softwareadapter in die virtuelle Datenquelle eingelesen (z.B. über USB- oder Netzwerkschnittstelle) und von dort in den MOSCOT-eigenen Datenfluss eingespeist. Je nach Anwendungsfall können das Bilder von Kameras, 2D-Trackingpositionen oder 3D-Markerdaten sein. Bereits vom Fremdsystem ausgeführte Schritte werden in der Datenverarbeitungskette ignoriert und die restlichen Schritte wie Filterung, Objektrekonstruktion etc. analog zu einem nativen MOSCOT-System ausgeführt. Zur Realisierung wird das externe Gerät als spezielles MOSCOT-Gerät behandelt, für das eine eigene angepasste Systemkonfiguration erstellt wird.

Die Einbindung externer Systeme ermöglicht auf einfachem Weg die Kombination von MOSCOT und anderen Trackingsystemen, z.B. zum Abgleich mit einem Referenzsystem oder zur Integration eines Standardtrackingsystems mit MOSCOT bei einer Simulatoranwendung.

Datenfusion

Das Modul zur Datenfusion und -synchronisation (`VtrkDataSynchronisation`) bildet den Endpunkt der parallelen Datenverarbeitungpfade der Kamerabilddaten und den Start der Rekonstruktionskette (vgl. auch Abbildung 4.1). Es wird ein Systemdatensatz (`VtrkSystemData`) aus den einzelnen `VtrkCameraData`-Instanzen eines Aufnahmezeitpunkts zusammengestellt, sobald alle im Datenfusionsmodul empfangen wurden. Es ist damit gleichzeitig auch die Datenquelle für die `VtrkSystemData`-Objekte. Die Datenfusion muss auch Fehlerfälle korrekt handhaben, in denen Daten einzelner Kameras z.B. durch Datenübertragungsfehler verloren gehen. Im einfachsten Fall werden dabei die Daten aller Kameras verworfen, bei ausreichender Redundanz ist auch die Weiterverarbeitung mit einem unvollständigem Sytemdatensatz möglich.

Datendistribution

Datendistributionspunkte (Instanzen von `VtrkSplitter`) bieten eine optionale Verteilung der Daten auf mehrere parallele Datenketten. Anwendung findet dies innerhalb der Applikationsschicht, wo verschiedene Datensenken parallel und unabhängig voneinander Ergebnisdaten bearbeiten oder darstellen. Eine weitere Aufgabe ist die selektive Weiterleitung unter vordefinierten zeitlichen Randbedingungen. So kann es sinnvoll sein, einem Visualisierungsmodul nur einen Teil der Trackingdaten mit der Bildfrequenz der Visualisierung zur Verfügung zu stellen, während beispielsweise Filter oder ein Regelungssystem die volle Datenrate erhalten.

Datensenken

Datensenken stellen die Endpunkte der Datenverarbeitungskette dar. Es handelt sich dabei um Übergabepunkte der Ergebnisse in die jeweilige Anwendung, um Visualisierungsmodule, die die Daten dem Nutzer graphisch aufbereitet präsentieren, und um Module zur Datenserialiserung. Letztere implementieren die kurz- bzw. langfristige Archivierung von Trackingdaten, sowie die Weiterleitung an Datensenken außerhalb der Applikationsschichten der VRM-Architektur (z.B. Netzwerk, Dateisystem, serielle Schnittstellen).

Die `VtrkDataDispatcher`-Klasse serialisiert Ergebnisdaten aller Verarbeitungsstufen, um sie in eine Datei abzuspeichern oder über Netzwerk und andere Schnittstellen an externe Empfänger zu übertragen. Virtuelle Datenquellen können ihre Datensätze aus diesen Dateien oder über die Netzwerkschnittstelle beziehen. Die Ein- und Ausgabemethoden der Datencontainerklassen sind zu diesem Zweck als symmetrische Serialiserungs- bzw. Deserialsierungsfunktionen implementiert.

Die `VtrkDataEventBuffer`-Klasse realisiert eine Archiv oder Cache-Funktion für Datensätze. Sie ist als Ringbuffer ausgeführt und archiviert die letzten N Datensätze jedes Datentyps. Diese Datenreihen können Kontrollmodulen für komplexe Datenanalysefunktionen zur Verfügung gestellt werden, wie z.B. zur automatischen Konfiguration von Kameraparametern oder Markerfarbregionen.

Beispiele für Datensenken in Form von Visualisierungsmodulen werden im folgenden Kapitel bei der graphischen Benutzeroberfläche von MOSCOT beschrieben.

4.4.10 Module zur Datenverarbeitung

In den Datenverarbeitungsmodulen werden die Methoden aus Kapitel 3 umgesetzt. Im wesentlichen handelt es sich um die Bildvorverarbeitung, Segmentierung und Merkmalsextraktion, Verzeichnungskorrektur, Markerrekonstruktion, Objektrekonstruktion sowie diverse Filter (vgl. auch die Auflistung in Abschnitt 4.4.8). Die Module empfangen zu Beginn des Trackingprozesses die aktuelle Systemkonfiguration und greifen hierüber auf die benötigten statischen Systemparameter zu (Kamera-, Marker-, Objektparameter). Während des Trackingbetriebs werden die Daten (`VtrkCameraData` bzw. `VtrkSystemData`) über die Transceiverstruktur verteilt und von den Modulen in ihren spezialisierten `process***Data`-Methoden verarbeitet. Die Module fügen ihre Verarbeitungsergebnisse dem Datencontainer hinzu und rufen das nächste Modul der Kette auf. Implementierungsdetails für einige ausgewählte Module werden im Folgenden erörtert:

Die Module zur Markerdetektion (Bildvorverarbeitung und Segmentierung mit Merkmalsextraktion) bilden die softwarebasierte Entsprechung des Hardware-Bildverar-

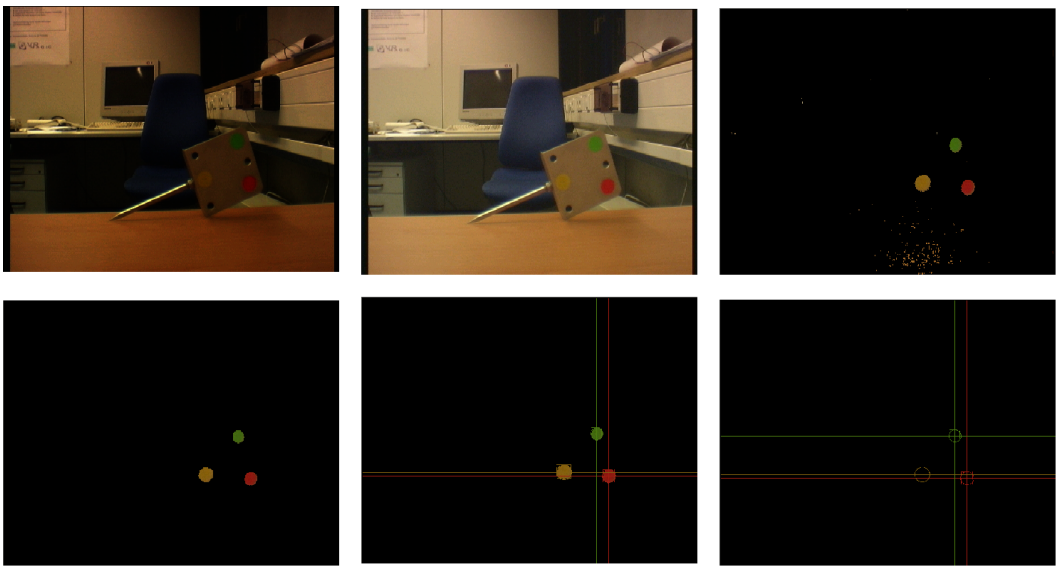


Abbildung 4.20: Ergebnisse der verschiedenen Stufen einer softwarebasierten Farbmarkerdetektion: Originalbild, Gammakorrektur, Farbklassifizierung, morphologische Erosion, Segmentierung und Merkmalsextraktion, 2D-Markerdaten.

beitungsmoduls aus Abschnitt 4.3.2. Für einen freien Wechsel der Datenverarbeitungsressourcen zwischen Soft- und Hardware ist hier auf die Unterstützung identischer Ein- und Ausgangsdatenstrukturen sowie Konfigurationsinformationen zu achten. Insbesondere müssen alle Bilddatenformate der Kameras bzw. Bildverarbeitungshardware interpretierbar sein, die bei der Hardware-Implementierung zum Einsatz kommen. Es sind also auch Rohbildformate und Demosaicing-Filter zu unterstützen. Das Softwaremodul arbeitet intern mit einem universellen Bilddatenkonverter, der in ein Standardbildformat (32bit-ARGB) wandelt. Auf diesem Format basieren die Bildverarbeitungsoperationen, die in der Imageprocessing-Bibliothek zusammengefasst sind. Durch die akkumulierenden Datencontainer der Softwarearchitektur können die Ergebnisse der einzelnen Bildverarbeitungsschritte einfach eingesehen und überprüft werden. Abbildung 4.20 zeigt als Beispiel die Ergebnisse der verschiedenen Stufen einer softwarebasierten Farbmarkerdetektion.

Das Filtermodul implementiert die in Abschnitt 3.6.1 beschriebene Filterfunktionalität. Um alle Anforderungen abzudecken, werden die Filter über eine ganze Klassenstruktur implementiert (Abbildung 4.21). Alle Filterklassen leiten dazu von einer Basisklasse `VmFilterBase<T,int>` ab. Generische Programmierung erlaubt die Verarbeitung von n-dimensionalen Datensätzen verschiedener Präzision und damit den Einsatz im 2D- und 3D-Bereich der Datenverarbeitungskette. Die einzel-

nen Filter werden zu Ketten `VmFilterChain<T,int>` aneinandergereiht, so dass Kombinationen von einfachen Filtertypen möglich werden (z.B. ein Average- mit einem Clampfilter). Im übergeordneten Filtermodul der Datenverarbeitungskette `VtrkMarkerFilter<T,int>` schließlich ist jedem Marker eine Filterkette zugeordnet. In seiner `process***Data`-Methode werden die gefilterten Werte berechnet und jeweils dem Datencontainer als neuer Datensatz auf dem Verlaufsstapelspeicher hinzugefügt.

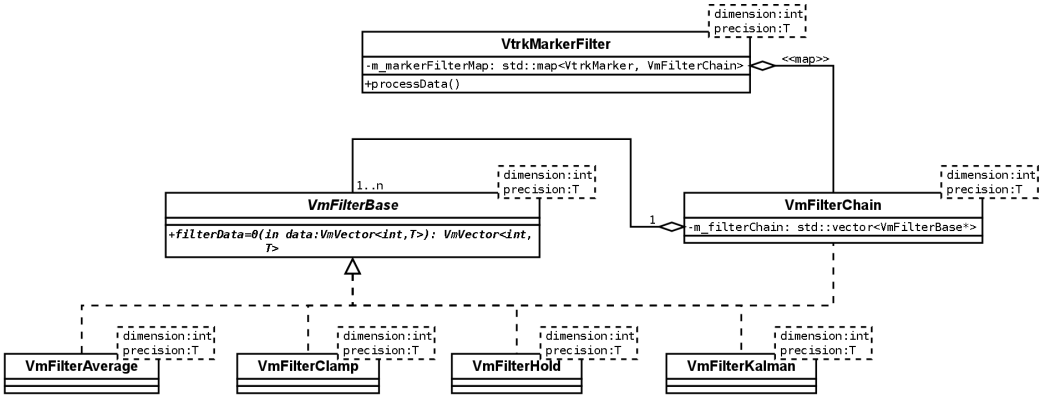


Abbildung 4.21: Filterketten für Einsatz auf 2D- und 3D-Datensätzen

Allgemeine Implementierungsstrategien für alle Datenverarbeitungsmodule werden im folgenden Abschnitt erläutert.

4.4.11 Implementierungsstrategien

Essentiell bei der Implementierung der Softwarearchitektur ist die Auslegung für niedrige Latenzzeiten und geringe Ressourcenanforderungen. Konkrete Maßnahmen hierzu umfassen folgende Punkte (vgl. hierzu auch [Azad et al. 2008, Kap. 2.12] und [Drepper 2007]):

Vermeidung von Kopiervorgängen: Ein direktes Durchreichen der Datensätze vermeidet einen Großteil der üblichen Kopiervorgänge. Bilddaten werden bereits vom Treiber direkt in die für die Weiterverarbeitung relevanten Speicherbereiche eingelesen und nicht zwischen Kernel- und Userdatenraum kopiert. Auch während der weiteren Bearbeitung werden durch Maßnahmen wie referenzgezählte Proxy-Entwurfsmuster, Dirty-Flags und Copy-on-write-Policy unnötige Kopiervorgänge vermieden und die Daten nach Möglichkeit nur einmal im Arbeitsspeicher vorgehalten.

Wiederverwendung von Speicher: Die Datenstrukturen (z.B. für Bilder oder auch Markerlisten) werden nach jedem Durchlauf der kompletten Datenverarbeitungskette geleert und wiederverwendet. Zum einen kann so eine beständige Freigabe und Neuallokation vermieden werden. Zum anderen verhindert dies die akkumulierende Aufsplitterung des verfügbaren freien Speichers.[16]

Cacheoptimierte Datenzugriffe: Für die effiziente Implementierung von datenintensiven Algorithmen (insbesondere bei der Bildverarbeitung) ist auf eine gute Ausnutzung der Datencaches des Prozessors zu achten, da die Zugriffe dort um Größenordnungen schneller ablaufen als im Hauptspeicher. Eine optimierte Anordnung der Speicherzugriffe (z.B. in inneren Schleifen mit Zugriffen auf fortlaufende Cachelinien und das Ausnutzen von Prefetch-Mechanismen der Prozessoren) ermöglicht Rechenzeiteinsparungen, die deutlich über das hinausgehen, was durch Verbesserung von Algorithmen erreichbar ist. Eine sehr gute Übersicht zur Relevanz solcher Optimierungsmethoden bietet Drepper [2007]. Ein weiteres Beispiel ist die Auslegung von Datenstrukturen auf 32- bzw. 64-bit-Grenzen, an Stellen an denen die Compiler dies nicht selbst umsetzen (z.B. Auffüllen von RGB-Daten der Hardware im 3x8bit-Format auf volle 32bit für jedes Bildpixel).

Verwendung von Lookup-Tabellen: Zur Vermeidung von wiederholter Neuberechnung von Pixeloperatoren (z.B. für die Farbklassifizierung oder Farbraumkonvertierungen) werden deren Werte vorberechnet und in Lookup-Tabellen gespeichert. Dabei kann eine Einschränkung der Farbtiefe sinnvoll sein, um den notwendigen Speicherbedarf in Grenzen zu halten (RGB 3×8bit: 16777216 Werte, RGB 3×6bit: 262144 Werte, RGB 565: 65536 Werte). Durch das ohnehin vorhandene Farbrauschen der Kameras in den niederwertigen Bits bleibt diese Einschränkung ohne relevante Auswirkung auf die Ergebnisse des Trackingprozesses.

4.5 Beispielkonfigurationen von Hard- und Softwarearchitektur

Ziel der modularen Auslegung von MOSCOT ist die flexible Anpassung der Datenverarbeitungsarchitektur an unterschiedliche Anforderungen. In den Abbildungen 4.22

[16] Die Speicherverwaltung moderner Betriebssysteme versucht bereits selbst, durch Wiederverwendung der an das System zurückgegebenen Speicherbereiche eine Fragmentierung zu vermeiden. Da jedoch auch andere Prozesse zu beliebigen Zeiten Speicher anfordern, kann dieses Ziel nicht immer erreicht werden. Dieses Problem entfällt nach einer Portierung auf ein Echtzeitbetriebssystem.

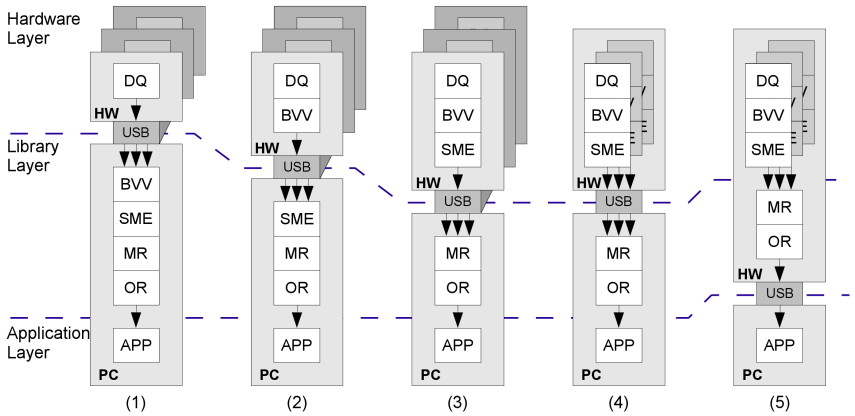


Abbildung 4.22: Überblick einiger Adaptionsmöglichkeiten der Hard- und Softwarearchitektur von MOSCOT für verschiedene Anwendungsszenarien. (Die Kürzel entsprechen den in Abbildung 4.1 verwendeten.)

und 4.23 ist eine Reihe möglicher Konfigurationen der Hard- und Softwarearchitektur für verschiedene Anwendungsszenarien skizziert.

Beispiel 1 zeigt die einfachste aller denkbaren Konfigurationen, bei der die Kamerasensoren direkt über ein Schnittstellenmodul mit dem Host-PC verbunden sind. Dort wird dann die vollständige Datenverarbeitungskette in Software ausgeführt.

Beispiel 2 erweitert dieses Beispiel um eine Bildverarbeitungshardware, die aber lediglich die rechenintensiven Bildvorverarbeitungsoperationen ausführt. Die vorverarbeiteten klassifizierten Bilder werden (evtl. lauflängenkodiert) über das Schnittstellenmodul an den Host-PC übertragen, wo die Verarbeitungsschritte ab der Merkmalsextraktionsstufe in Software ausgeführt werden. Diese Konfiguration kommt z.B. bei dem Diagnostiksimulator in Abschnitt 6.5 zum Einsatz.

Die Beispiele 3 und 4 zeigen eine vollständige Markerdetektion in Bildverarbeitungshardware, so dass nur detektierte 2D-Markerposition an den Host-PC übermittelt werden. Der Rest der Verarbeitung läuft wie bisher im Host-PC in Software ab. Im Unterschied zum vorhergehenden Szenario, bei welchem die intelligenten Kameras nur Bildvorverarbeitungsfunktionen ausführen, beschreibt dieses Beispiel vollständige Trackingkameras. Dies ist die Standardkonfiguration vieler MOSCOT-Anwendungen. Die Hardwaremodule können dabei wie in Beispiel 3 als separate intelligente Kameras zusammengestellt werden (vgl. die intelligente Kamera aus Abbildung 4.10 für die Anwendung ITD in Abschnitt 6.3). Oder die Hardwaremodule werden wie in Beispiel 4 für die Benutzerschnittstelle eines Simulators auf einer Platine vereint (vgl. das Tracking für den Simulator EYESI in Abschnitt 6.3). In letzterem Fall wäre als zukünftige Erweiterung der Systemarchitektur auch die Verlagerung der

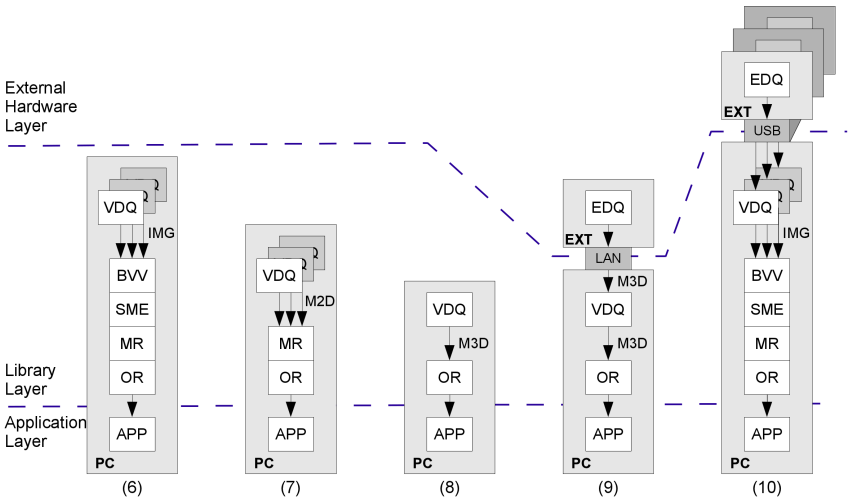


Abbildung 4.23: Überblick einiger Adaptionsmöglichkeiten virtueller Systeme für verschiedene Anwendungsszenarien. (EDQ bezeichnet hier aus Sicht des MOSCOT-Systems externe Datenquellen.)

PC-seitigen Datenverarbeitung auf einen im FPGA integrierten Prozessor denkbar, so dass ein komplett integriertes Trackingsystem entsteht, welches fertig rekonstruierte Objektdaten zum Host-PC des Simulators sendet (Beispiel 5).

Die Beispiele 6 und 7 zeigen während der Entwicklung neuer Trackinganwendungen eingesetzte Systemkonfigurationen mit virtuellen Kameras. Alle Datenverarbeitungsschritte laufen in Software ab, das virtuelle System generiert oder deserialisiert Kameradaten in Form von Bildern oder 2D-Markerdaten. Diese Konfiguration wird bei der Systemsimulation der erweiterten ITD-Anwendung in Abschnitt 6.4 eingesetzt.

Beispiel 8 zeigt im Unterschied hierzu ein virtuelles Gesamtsystem, das bei der Entwicklung komplexerer Rekonstruktionsprozesse eingesetzt werden kann. Hier werden aufgezeichnete oder generierte 3D-Markerdaten in die weitere Rekonstruktionskette eingespeist. Komplexere Verfahren zur Objektrekonstruktion können so implementiert und getestet werden. Die Beispiele 6, 7 und 8 stellen auch die Simulationskonfiguration für die Entwicklung neuer Anwendungen mit der Tracklab-Umgebung aus dem nächsten Kapitel dar.

Beispiel 9 zeigt die Einbindung eines vollständigen Trackingsystems eines Drittherstellers als externes System, das bereits rekonstruierte 3D-Markerdaten liefert. Die Daten werden entgegengenommen (hier über eine Netzwerkverbindung) und dann über ein virtuelles System in den bekannten MOSCOT-Trackingdatenfluss eingespeist. Anwendungen bestehen bei der Kombination von MOSCOT mit Fremdsystemen oder auch für den einfachen Datenvergleich mit einem Referenzsystem.

Beispiel 10 schließlich zeigt die mögliche Anbindung externer Kameras von Drittherstellern, z.B. über USB, zusammen mit der weitereren Datenverarbeitung in den Bibliotheken der MOSCOT-Software (ähnlich Beispiel 1).

Kapitel 5

Konfigurations- und Entwicklungsumgebung

Die im vorigen Kapitel beschriebenen Hard- und Softwarestrukturen stellen bereits alle Funktionalität für den Aufbau maßgeschneiderter Trackingsysteme bereit. Für eine schnelle Umsetzung von Trackinglösungen ist für Anwender und Entwickler jedoch eine weitergehende Unterstützung auf höherer Ebene hilfreich – ein graphisches Softwarefrontend für Einsatz, Anpassung und Weiterentwicklung der Systeme.

Mit dieser Zielstellung wurde die integrierte Konfigurations- und Entwicklungsumgebung „Tracklab“ konzipiert.[1] Sie unterstützt bei Systemkonfiguration und Komponentenentwicklung und stellt eine graphische Benutzerschnittstelle zur Kontrolle der Trackingsysteme bereit. Im Folgenden wird ein Überblick über Konzept, Softwarearchitektur und Realisierung von Tracklab gegeben und im Anschluss kurz einige Anwendungsbeispiele präsentiert.

5.1 Analyse des Nutzungskontexts

Zentrale Eigenschaften der MOSCOT-Systemarchitektur sind die flexible Adaption von Trackingsetups und die einfache, schnelle Entwicklung neuer Trackinganwendungen. Hierzu werden eine Reihe von Funktionen bereitgestellt, die bei der Anwendungsentwicklung unterstützen (Simulation mittels virtueller Trackingsysteme, duale Hard- und Softwarearchitektur, gleiche Datenformate für hard- und softwareseitige Serialisierung, akkumulierende Datencontainer mit vollem Zugriff auf Zwischenergebnisse). Es ist wichtig, für ein solches Gesamtsystem eine effiziente Nutzerschnittstelle bereitzustellen. Die Betriebssoftware von Trackingsystemen besteht jedoch oft aus

[1] Die Software Tracklab wird in Kooperation mit dem Entwicklungsteam der VRmagic GmbH entwickelt. Der Name selbst geht auf eine frühe Vorläuferversion der Kalibrierungssoftware für das Tracking des Simulators EYESI zurück.

mehreren Einzelanwendungen für Konfiguration, Kalibration, Trackingbetrieb etc., die vom Nutzer durch unterschiedliche Oberflächen häufige Programmwechsel und Bedienungsumstellungen erfordern. Für den Entwickler existieren wiederum separate Anwendungen, die Zugriff auf den Datenfluss des Systems gewähren. Für das MOSCOT-System soll im Gegensatz dazu mit der Tracklab-Anwendung eine zentrale Programmumgebung bereitgestellt werden, die für Anwender wie Entwickler alle mit dem System auszuführenden Aufgaben unter einer einheitlichen Oberfläche zusammenführt.

Reine Anwender der Trackingsysteme benötigen ein Werkzeug für Konfiguration und Kalibration der Systeme, für Online-Kontrolle und Verifikation des Trackingvorgangs. Entwickler neuer Trackinganwendungen bedürfen zusätzlich einer Entwicklungsumgebung, die bei Entwurf und Realisierung neuer Komponenten und Anwendungen unterstützt. Die Verbindung beider Nutzungsaspekte auf Basis einer gemeinsamen Programmumgebung bietet sich an, da auch Entwickler während Entwicklung und Tests von Komponenten Zugriff auf die Basisfunktionalität für Anwender benötigen. Des Weiteren kann neuentwickelte Funktionalität in diesem Rahmen schnell dem Anwender bereitgestellt werden.

Zur Strukturierung der verschiedenartigen Anwendungsfälle und der resultierenden Anforderungen ist es hilfreich, die einzelnen Nutzerklassen der MOSCOT-Systeme zu definieren, sowie die jeweiligen Aufgaben zu identifizieren, die diese mit dem System ausführen wollen. Mit einem solchen nutzerorientierten Ansatz des Softwareentwurfs befasst sich das User-Centered-Design (UCD) [Norman und Draper 1986] [Abras et al. 2004][2] Im Rahmen des UCDs sind bereits zu Beginn der Entwicklung zentrale Fragen hinsichtlich des Nutzerkreises und Einsatzes der Anwendung zu klären. Dabei werden verschiedene typische Nutzungssituation skizziert, in Form von Nutzerprofilen (sogenannten Personas) und Nutzungsszenarien. Sie erleichtern Analyse und Definition der Anforderungen, sowie Konzeption und Entwurf der Nutzerinteraktion und Programmstruktur.

5.1.1 Nutzerprofile und Nutzungsszenarien

Aus den Erfahrungen bei Einsatz und Entwicklung von Trackingsystemen in den Projekten der Arbeitsgruppe lassen sich im wesentlichen drei voneinander abgegrenzte Nutzergruppen identifizieren:

- Anwender[3] (beispielsweise Projektpartner in Kooperationen), die MOSCOT-Trackinglösungen einsetzen, aber nicht selbst weiterentwickeln. Sie benötigen

[2] Die Internationale Standardisierungsorganisation ISO standardisiert diesen Designansatz in [ISO/IEC 1999] als "nutzerorientierten Gestaltungsprozess".

[3] Der Begriff Anwender beschreibt im folgenden Kontext eine Nutzerprofilklasse der Tracklaban-

Funktionalität in den Bereichen Konfiguration, Kalibration und Kontrolle der Trackingsysteme.

- Anwendungsentwickler (z.B. Studenten in der Arbeitsgruppe), die neue Trackinglösungen auf Basis, aber außerhalb der Tracking-Bibliotheken des MOSCOT-Systembaukastens entwickeln. Beispiele sind spezielle Rekonstruktionsschritte für Instrumente oder Operationssitus neuer Simulatortypen. Die Entwickler benötigen Zugriff auf die maßgeblichen Ergebnisse des Rekonstruktionsprozesses, Erweiterungsmöglichkeiten der Rahmenanwendung hinsichtlich neuer Problemlösungen sowie die Kontrolle elementarer Systemfunktionen.
- Kernentwickler (meist Mitarbeiter der Arbeitsgruppe oder der VRmagic GmbH), die Kernkomponenten austauschen oder erweitern. Sie implementieren beispielsweise die Datenverarbeitungsmethoden zur Unterstützung neuer Typen von Kamerasensoren oder Markern und benötigen dafür vollen Zugriff auf interne Datenstrukturen und Kontrolle aller grundlegenden Datenverarbeitungsschritte und Prozessparameter.

Eine ähnliche Unterteilung ergibt sich auch bei einer abstrakten Betrachtung der Interaktion mit dem Trackingprozess: Anwender verändern und kontrollieren lediglich Prozessparameter des Trackings, Anwendungsentwickler erweitern die Trackingprozesskette bzw. Rekombinieren ihre Teilglieder, während Kernentwickler den Prozess selbst verändern und Teile davon ersetzen.

Nutzungsszenarien der Anwender

Allgemeine Konfiguration und Kalibration: Der Anwender muss Trackingsysteme an spezifische Aufgabenstellungen anpassen können. Konkrete Aufgabenbereiche sind die Definition der Systeme als solche mit Kameras, Markern und Objekten und die Einstellung von Parametern zu Bildverarbeitungs- und Rekonstruktionsalgorithmen. Darüberhinaus müssen vor der Inbetriebnahme verschiedene Parameter des Systems kalibriert werden. Dazu gehören Kameraparameter- und Bildeinstellungen, Markerfarb- und -intensitätswerte sowie evtl. die relativen Koordinaten der Objektmarker (vgl. Abschnitt 3.2). Über verschiedene graphische Module soll das einfache Konfigurieren der Eigenschaften eines MOSCOT-Trackingsystems ermöglicht werden sowie der Anwender bei der Systemkalibration von einem softwaregeführten Prozess unterstützt werden.

Systemspezifische Konfiguration und Kalibration: Spezielle Trackingsysteme erfordern eine weitergehende Konfiguration und Kalibration, die über das grundlegende Setup hinausgeht. So bedarf möglicherweise das Tracking in medizinischen

wendung. Anwender bilden also eine Teilmenge der Nutzer und die Begriffe werden nicht synonym verwendet.

Simulatoren einer zusätzlichen Registrierung des Operationsinterfaces bzw. der Instrumente (siehe bspw. Abschnitt 6.1 zum Trackingaufbau des Augenoperationssimulators EYESI). Tracklab soll den Anwender bei diesem systemspezifischen Setup unterstützen und die Funktion innerhalb derselben einheitlichen Rahmenapplikation bereitstellen.

Online-Kontrolle, Test und Verifikation: Während des eigentlichen Trackingvorgangs wünscht der Anwender die direkte Kontrolle des laufenden Trackingprozesses. Visualisierungen von Daten der verschiedenen Verarbeitungsstufen wie Rohbilder, verarbeitete Bilder, 2D-Markerkoordinaten und rekonstruierte 3D-Marker- und Objektpositionen müssen darstellbar sein. Bei neuen Trackingaufbauten und anwendungen will der Anwender die Ergebnisse live überprüfen und Ergebnisse sichten. Eine Datenaufnahme der erfassten Datensätze ist für weitergehende Offline-Analysen nützlich.

Nutzungsszenarien der Anwendungsentwickler

Entwicklung anwendungsspezifischer Trackingerweiterungen: Entwickler neuer Trackinganwendungen benötigen eine Umgebung in der sie weiterführende, auf den vorhandenen Module aufbauende Trackingmethoden z.B. angepasste Rekonstruktionsverfahren für spezielle Objekte oder Systeme entwickeln und testen können. Die Entwicklung erfordert Zugriff auf die Rohdaten der Trackingbibliothek und die Möglichkeit ihre Ergebnisse zu visualisieren und schnell zwischen Kalibration, Setup und Test umzuschalten. Weiterhin benötigt diese Nutzergruppe die Möglichkeit, die Tracklab-Benutzerschnittstelle um neue Funktionalität für den Anwender zu erweitern, beispielsweise für die angeführte systemspezifische Konfiguration und Kalibration oder die Darstellung anwendungsspezifischer Rekonstruktionsergebnisse (siehe bspw. in Abschnitt 6.2 das Entwicklungsmodul zum Deformationstracking).

Entwicklung von Stand-Alone-Trackingapplikationen: Für einige Anwendungen des MOSCOT-Systems, die keine eigene Benutzeroberfläche (wie z.B. Simulatoren) mitbringen, ist eine zugehörige Betriebssoftware nötig, die Konfiguration und Betrieb der Systeme steuert und die Trackingergebnisse darstellt. Die Entwicklung solcher spezialisierter Applikationssoftware auf Basis der Tracklab-Rahmenanwendung vermeidet wiederholte Neuimplementierung von Funktionalität wie Aufsetzen und Kontrolle der Trackingsysteme.

Nutzungsszenarien der Kernentwickler

Entwicklung neuer MOSCOT-Kernkomponenten: Kernentwickler entwickeln und erweitern das MOSCOT-System durch neue Funktionalität in der Hardware

und der VRM-Trackingbibliothek. Beispiele dafür sind spezielle Bildverarbeitungskomponenten zur Unterstützung neuer Markertypen oder komplexere Algorithmen für die Rekonstruktion spezieller Objekte in Simulatorschnittstellen. Kernentwickler benötigen die Möglichkeit an Kernkomponenten zu arbeiten, ohne vorhandene Funktionalität der Trackingbibliothek zu beeinflussen. Dafür benötigen sie Zugriff auf alle internen Datensätze innerhalb und außerhalb der Trackingbibliothek, den direkten Datenvergleich zwischen neuentwickelten und etablierten Komponenten und schnelles Umschalten verschiedener Trackingsetups (bspw. zwischen Hardware- und Software-Bildverarbeitung). Nach erfolgreichen Tests außerhalb der Trackingbliothek können neuentwickelte Verarbeitungsmodule in diese integriert werden.

Gemeinsame Nutzungsszenarien von Anwendungs- und Kernentwicklern:

Simulation von Trackingsetups: Die Entwickler benötigen Bereitstellung von Funktionalität für die Simulation von Systemen. Bevor Systeme in der Praxis eingesetzt werden bzw. neue Algorithmen, Objekte und Markertypen implementiert werden, möchte der Entwickler die Ergebnisse simulieren. Zu diesem Zweck benötigt er eine Umgebung die die Features virtueller Systeme transparent unterstützt und gleichartig auf echte Systeme abbilden kann.

Alle Nutzungsszenarien der Anwender: Während Entwicklung, Simulation und Tests neuer Funktionalität treten auch viele der bei den Anwendern beschriebenen Nutzungsszenarien auf. Dazu zählen unter anderem Konfiguration und Kalibration von Systemen und die Visualisierung von Daten. Sie müssen so integriert sein, dass für den Entwickler ein schnelles Umschalten zwischen Entwicklung und Anwenderfunktion möglich ist.

5.1.2 Allgemeine Anforderungen

Unabhängig von den aufgeführten Nutzungsszenarien sollte Tracklab als Frontend für die MOSCOT-Trackingarchitektur Unterstützung für deren spezielle Eigenschaften bieten und diese dem Nutzer zugänglich machen. Hier sind insbesondere zu nennen:

- Serialisierung (Laden, Speichern, Generieren) von Datensätzen aller Verarbeitungsstufen der Trackingpipeline (Kamerabilder, 2D-Markerpositionen, 3D-Markerpositionen, Objektpositionen)
- Unterstützung der dualen Datenflussarchitektur für Simulation, Verifikation und Test neuer Setups. Direktes Umschalten zwischen verschiedenen Hard- und Softwarekomponenten zu Bildverarbeitung, Markerdetektion, Rekonstruktion.
- Transparente Integration von virtuellen Systemen zur Simulation und Verfikation realer Systemsetups

- Skalierbarkeit durch einfaches Hinzufügen zusätzlicher Entitäten wie Kameras, Marker oder Objekte zu bestehenden Setups
- Bereitstellung von Standard-GUI-Komponenten für wiederkehrende Aufgaben wie Datenanzeige, Konfiguration, etc. von MOSCOT-Systemen
- Möglichkeit zum direkten Umschalten zwischen verschiedenen Nutzungsszenarien

5.1.3 Schlussfolgerungen

Die Konfigurations- und Entwicklungsumgebung muss die allgemeinen, für alle Systeme erforderlichen Grundfunktionen von Systemkonfiguration und Kontrolle des Trackingvorgangs bereitstellen. Darüberhinaus soll die Anwendungsumgebung um zusätzliche Funktionalität für die Unterstützung spezialisierter Trackingsetups erweiterbar sein. Da Anforderungen von Anwendern wie Entwicklern im Rahmen einer einzigen Applikation abgedeckt werden, ist eine deutliche Abgrenzung der einzelnen Funktionen für den Nutzer sinnvoll. Für die Belange der Entwickler ist zusätzlich ein Rahmen zu schaffen, der Zugriff auf alle relevanten Daten der internen Datenverarbeitungsstufen bereitstellt – dabei aber auch eine unabhängige Entwicklung außerhalb der Trackingbibliotheken zulässt.

Die graphische Oberfläche selbst soll ein einheitliches Benutzerinterface zur Verfügung stellen, das die Kontrolle aller Systemfunktionalität umfasst. Gleichzeitig muss ein schnelles Umschalten zwischen verschiedenen Aufgaben und eine Einbindung neuer Funktionen in die gegebene Nutzeroberfläche möglich sein. Für einfache Anwendungen sollen nicht erforderliche Funktionen aus Tracklab entfernbar sein, so dass eine eingeschränkte Tracklab-Umgebung als Ersatz für eine dedizierte Anwendung fungieren kann.

5.2 Architektur

5.2.1 Konzept

Zur Realisierung der geforderten Flexibilität und Erweiterbarkeit wird Tracklab als modulares Applikationsframework konzipiert. Die eigentliche Rahmenanwendung kann dabei über Plugins mit neuer Funktionalität erweitert werden. Damit findet das Konzept des Baukastensystems der MOSCOT-Gesamtarchitektur auch im Softwarefrontend seine Fortsetzung.

Innerhalb der Tracklab-Anwendung werden die Plugins auch als Tracklabmodule bezeichnet. Sie implementieren jeweils sowohl Funktionalität als auch Benutzeroberfläche für eines der beschriebenen Nutzungsszenarien. Tracklab wird durch die jeweiligen Plugins gleichsam in verschiedene spezialisierte Nutzerschnittstellen umgeformt, die den für das jeweilige Nutzungsszenario passenden Zugriff auf die Trackingsysteme bereitstellen. Das modulare Konzept ist gerade auch bei der Entwicklung einer Stand-Alone-Anwendung auf Tracklab-Basis vorteilhaft, da innerhalb des zugehörigen Plugins sämtliche Aspekte des Zugriffs auf Konfiguration und Daten eines Systems und auch die Nutzerinteraktion selbst kontrollierbar sind.

Grundlegendes Modell der Anwendungsentwicklung unter Verwendung pluginbasierter Frameworks ist die Inversion of Control (IoC, Steuerungsumkehr) [Fowler 2004]. Dieses Umsetzungsparadigma formalisiert die Übergabe des Kontrollflusses von der Rahmenapplikation an eingebettete Erweiterungen, die dadurch eigene Funktionalität im Kontext der Gesamtapplikation ausführen. Mayer et al. [2003] beschreibt das Pluginkonzept selbst als eigenständiges Entwurfsmuster für die Entwicklung schlanker erweiterbarer Programme. Bekannte Realisierungen des Konzepts finden sich beispielsweise im Webbrowser Firefox oder auch bei der erweiterbaren Entwicklungsumgebung Eclipse [Budinsky 2003; Gamma und Beck 2003].

Gewöhnlich realisieren Pluginarchitekturen die Erweiterung einer Applikation um Zusatzfunktionen. Wird hingegen die gesamte Funktionalität und Nutzerschnittstelle in Form von Plugins realisiert, so spricht man von reinen Pluginarchitekturen [Birsan 2005], bei denen der Applikationsrahmen lediglich aus einer Pluginengine besteht. Tracklab ist konzeptionell eine solche reine Pluginarchitektur, da sämtliche umgesetzten Nutzerszenarien als Tracklabmodule realisiert werden. Zusätzlich ist jedoch eine zentrale übergeordnete Instanz zur Verwaltung und Kontrolle der Trackingsysteme erforderlich, die nicht als Plugin implementiert werden kann. Damit umfasst die Tracklab-Rahmenapplikation zwei Hauptkomponenten: die Pluginengine zur Einbettung der aufgabenspezifischen Module und die Kontrollfunktionalität für die Trackingsysteme. Abbildung 5.1 gibt einen schematischen Überblick der Struktur.

Die Pluginengine integriert die Plugins in die Applikation und bindet sie mit ihren spezifischen GUI-Elementen in die Benutzeroberfläche ein. Weiterhin stellt sie die Kommunikationsstrukturen zwischen Plugin, Rahmenanwendung und Trackingbibliothek bereit. Die Engine kontrolliert auch Selektion und und Freischaltung geeigneter Plugins für das aktive Trackingsystem sowie ihre Initialisierung und Terminierung. Die zweite wesentliche Funktionalität des Tracklab-Applikationsrahmens ist die Verwaltung der Trackingsysteme. Dem Anwender stellt das Framework Bedienelemente zur Kontrolle der Trackingsysteme und Dialoge für die Systemkonfiguration bereit, während der Entwicker zusätzlich Zugriff auf die Systemfunktionalität über

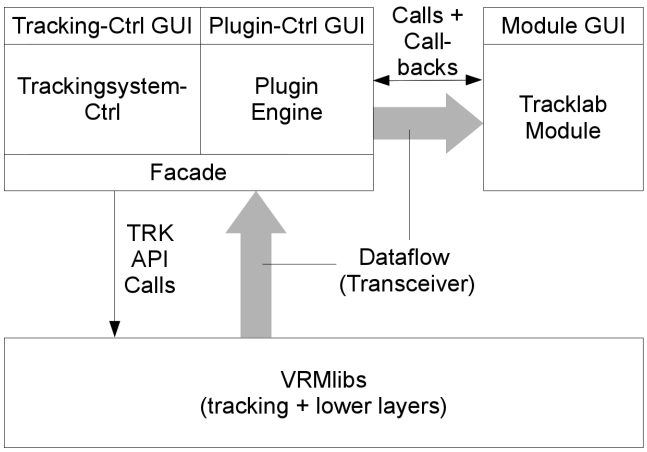


Abbildung 5.1: Übersicht der logischen Struktur des Tracklab-Applikationsframeworks

eine Softwareschnittstelle erhält. Weiterhin kapselt die Systemverwaltung die immanenten Unterschiede von Trackingsystemen mit MOSCOT-spezifischer Hardware und virtuellen bzw. externen Spezialsystemen, so dass für die Plugins ein einheitliches Gerätemodell vorliegt. Das Tracklab-Framework stellt darüber hinaus auch Infrastruktur in Form von wiederverwendbaren generischen Funktionskomponenten zur Verfügung, beispielsweise Visualisierungsmodule für verschiedene Datentypen wie Kamerabilder, 2D-Trackingdaten bzw. rekonstruierte 3D-Szenen.

Abbildung 5.2 zeigt die Elemente der graphische Benutzeroberfläche (Graphical User Interface - GUI) von Tracklab an einem Beispiel. Die Rahmenanwendung stellt Bedienelemente der Systeme sowie Datendisplays zur Visualisierung von Kameradatensätzen. Die Plugin-GUI integriert sich als zentrales Element in die Tracklab-Umgebung. So steht für jedes Szenario eine angepasste Bedienoberfläche bereit.

5.2.2 Datenfluss und Kommunikationsstruktur

Die internen Kommunikationsstrukturen werden bestimmt vom Datenfluss zwischen der Trackingbibliothek und den Tracklabmodulen. Die Rahmenapplikation bindet dabei als Zwischenschicht die Plugins an die Bibliothek an. Mit ihrer Datenflussarchitektur verfügt die Trackingbibliothek bereits über einen effektiven Mechanismus für die Weitergabe von Systemdaten und -konfigurationen (vgl. Abschnitt 4.4.8). Es ist daher naheliegend, diese Kommunikationsstruktur in Tracklab weiterzuführen und die Datenempfänger in den Plugins als Spezialisierungen von `VtrkTransceiverBase` zu realisieren. So bleiben die Trackingbibliothek und ihre interne Datenverarbeitung gekapselt vor Modifikationen innerhalb der Tracklab-Applikation; gleichzeitig

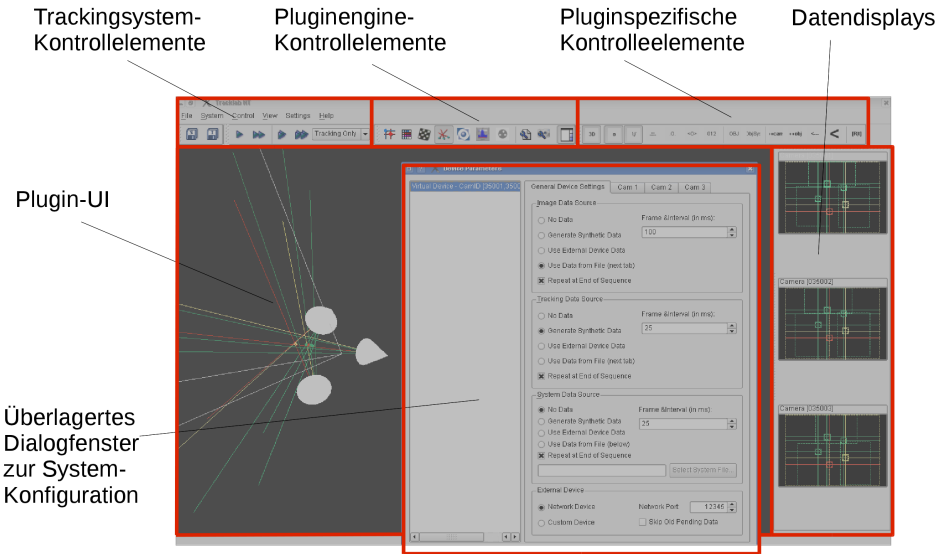


Abbildung 5.2: Schematische Darstellung der graphischen Benutzerschnittstelle von Tracklab. Der Anwendungsrahmen stellt Bedienelemente für die Trackingsysteme und die Pluginengine sowie Dialoge zur Systemkonfiguration dar. Die Plugin-GUI füllt den zentralen Bereich und integriert Bedienelemente in den Rahmen. Zusammen mit optional im Randbereich einblendbaren Datendisplays erhält so jedes Nutzungsszenario seine eigene angepasste Benutzerschnittstelle.

erlauben jedoch die Datencontainer Zugriff und Weiterverarbeitung aller Daten und Zwischenergebnisse. Ebenso ist Transparenz für Änderungen der Systemkonfiguration sichergestellt, d.h. Benachrichtigungen über Konfigurationsänderungen werden den Tracklabmodulen direkt zugänglich gemacht. Zusätzlich können neue Kernkomponenten so in einem abgetrennten Bereich außerhalb der Trackingbibliotheken entwickelt und erst nach erfolgter Verifikation in selbige integriert werden.

Für die Kommunikation in die Trackingbibliothek hinein existieren Interfacemethoden in verschiedenen grundlegenden Klassenobjekten (`VtrkManager`, `VtrkSystem` – vgl. Abschnitt 4.4). Die Rahmenapplikation Tracklab kapselt als Fassadenentwurfsmuster ([Gamma et al. 2001, S.212ff]) den Zugriff auf diese Klassen. Das Entwurfsmuster fasst den Zugriff in einer zentralen Schnittstelle zusammen und vereinheitlicht den Umgang mit verschiedenen Systemen und Geräten.

Neben der Anbindung an die VRM-Bibliotheken erfordern Initialisierung und Integration der Plugins eine zusätzliche bidirektionale Kommunikation zwischen Tracklab und seinen Modulen. Für die Kommunikation zu den Modulen hin können auf-

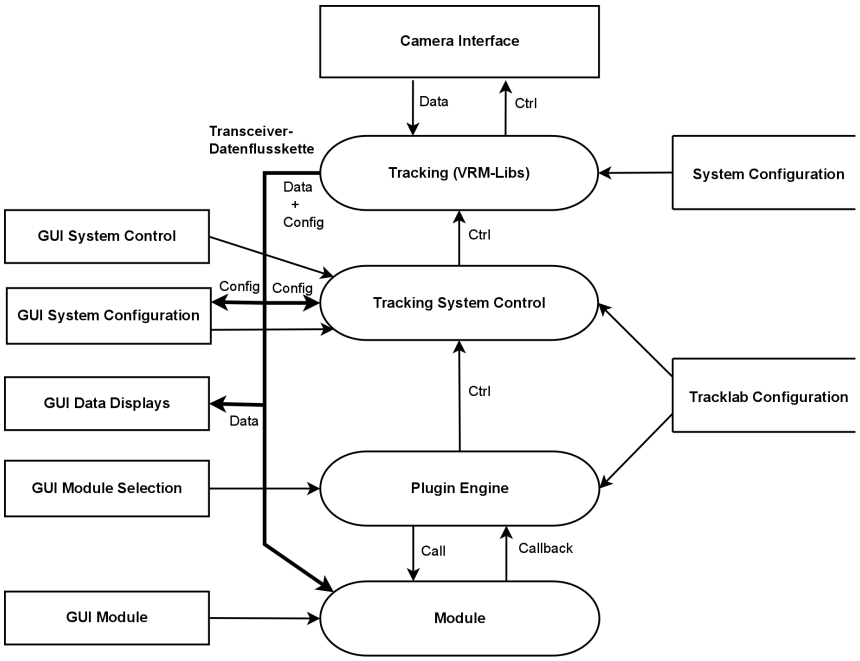


Abbildung 5.3: Detaillierte Übersicht über den Daten- und Kommunikationsfluss des Tracklab-Frameworks

grund der Aggregation der Tracklabmodule Methoden in den Plugins direkt aufgerufen werden. Für die Rückrichtung existieren Callbackmethoden in der Rahmenapplikation, die ein Signal-Slot-Entwurfsmuster [Collins 2005; Gregor 2004] implementieren.

Abbildung 5.3 gibt einen detaillierteren Überblick über die Datenfluss- und Kontrollstrukturen zwischen Tracklab, seinen Plugins und der Trackingbibliothek.

5.2.3 Pluginimplementierung

Die Implementierung basiert auf einer abstrakten Basisklasse `VtlModuleBase` (Abbildung 5.4), von der die einzelnen Tracklabmodule ableiten. Mittels polymorpher Elementfunktionen wird die Schnittstelle für Anmeldung, Initialisierung und Aktivierung der Module deklariert. Anforderungen und Fähigkeiten der Plugins werden dabei über ein Property-Objekt (`VtlModuleProperties`) festgelegt. Die interaktiven Aufrufe in Rückrichtung erfolgen wie beschrieben über Callbackfunktionen, die als Signal-Slot-Kommunikation realisiert sind [Gregor 2004] [Digia Plc 2012b]. Die Modulbasisklasse bietet den einzelnen Plugins dazu Zugriff auf die zentrale Instanz

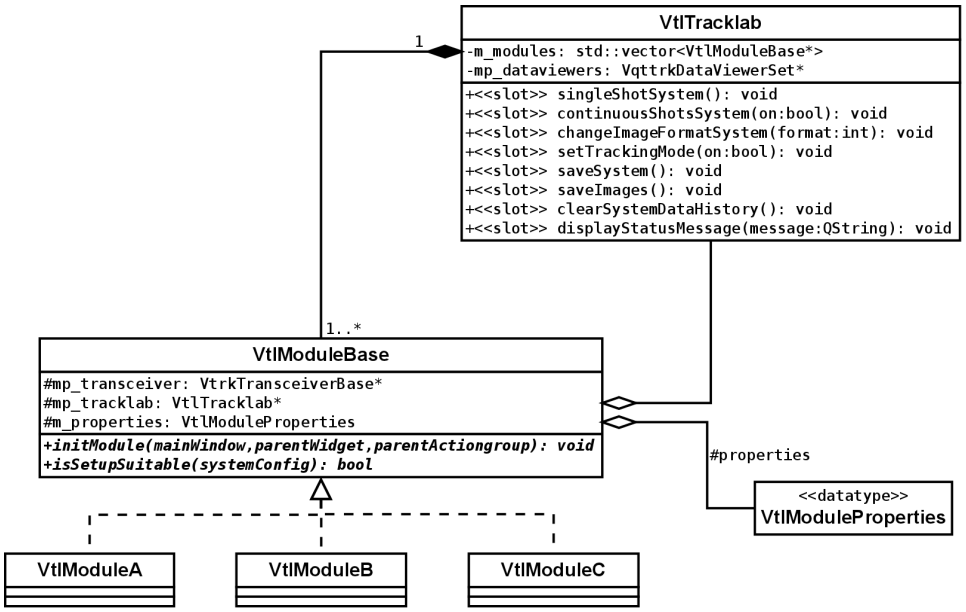


Abbildung 5.4: Ausschnitt aus dem Klassendiagramm der zentralen Tracklabklassen VtrlTracklab, VtlModuleBase und davon abgeleitete Pluginklassen sowie wichtige Elementfunktionen und Slots zur Kommunikation.

`VtlTracklab`, die die Slotfunktionen zur Kontrolle des Systemverhaltens bereitstellt. Tracklabmodule können je nach Einsatzbereich als statische Objektmodule zur Compilezeit festgelegt und gebunden oder als dynamische Bibliotheksmodule (DLLs) zur Laufzeit über ein Factory-Entwurfsmuster eingebunden werden [Gamma et al. 2001, S. 131]. Dadurch kann die Rahmenapplikation selektiv und ohne Neukompilierung um Funktionalität erweitert werden.

Das interne Design aller Plugins basiert auf einem MVC-Entwurfsmuster (Model-View-Controller [Burbeck 1992]). Es wird insbesondere in graphischen Frameworks verwendet, um eine Kapselung und strukturelle Trennung von Funktionalität und visueller Repräsentation zu gewährleisten. Dabei stellt das Modell die abstrakte Repräsentation der darzustellenden Datenstrukturen bereit. Die View-Komponente implementiert die eigentliche graphische Oberfläche des Plugins, die mit der GUI-Bibliothek Qt realisiert ist [Blanchette und Summerfield 2006]. Der Controller schließlich verbindet das abstrakte Modell mit der Darstellung im View und ermöglicht die Benutzerinteraktion. Das Plugin selbst implementiert die Interaktionslogik des Kontrollers, der das abstrakte Modell mit der Darstellung im View verbindet. Er reagiert auf Änderungen der Daten (Model) durch Aktualisieren der GUI (View), Interaktionen des Anwenders mit der GUI (View) manipulieren im Gegenzug die

Konfigurationsdaten (Model).

Als Modelle werden direkt die Datenhaltungsklassen der Trackingbibliothek eingesetzt, wodurch sich zwei Arten von Plugins unterscheiden lassen:

Konfigurationsplugins basieren auf dem Modell der `VtrkSystemConfig`-Klasse. Sie stellen die graphische Schnittstelle für die in Abschnitt 3.2 beschriebenen Abläufe bei der Konfiguration des Systems dar. Dazu zählen Modifikation der Definitionen von Kameras, Markern und Objekten, Einstellung von Farbparametern, Beleuchtung etc. Charakteristisch ist die interaktive, nutzergesteuerte Betriebsweise dieser Module, die keine Echtzeitfähigkeit und damit auch keine Datenverarbeitung und -darstellung mit der Trackingdatenrate erfordert.

Trackingplugins verwenden die Datenklassen `VtrkCameraData` bzw. `VtrkSystemData` als Modell. Sie bilden die graphische Schnittstelle für die Kontrolle der im Rest von Kapitel 3 beschriebenen Datenverarbeitungsvorgänge. Hierzu zählen Befehle an das System, Datenvisualisierung von Kamerabildern und Markerpositionen, Auswertung und Verifikation der Daten. Diese Module arbeiten während des Trackingvorgangs und bearbeiten und visualisieren die Daten in Echtzeit mit der Datenrate des Trackingvorgangs. Es findet abgesehen von Start und Stop des Trackingbetriebs nur wenig Interaktion des Benutzers statt.[4]

5.3 Beispiele für Tracklabmodule

Im Folgenden werden exemplarisch einige Plugins beschrieben, die im Rahmen verschiedener Trackinganwendungen entwickelt wurden. Sie sollen die Bandbreite an Plugins veranschaulichen, die mit der vorgestellten Architektur realisierbar sind.

Plugins für Standardaufgaben

Eine Reihe von Modulen deckt Standardaufgaben ab, die bei verschiedenen Trackingsystemen und -szenarien immer wieder in derselben Form auftreten. Sie entsprechen den Nutzungsszenarien der Klasse der Anwender aus Abschnitt 5.1.1. Als Beispiele werden Plugins zur Kalibration, zur Markerkonfiguration, zur Visualisierung der rekonstruierten 3D-Szene und zur statistischen Datenauswertung dargestellt.

[4] Komplexe Konfigurationsplugins können als Hybridmodule auch Kombinationen beider Funktionalitäten vereinen, beispielsweise bei automatisierten Einmessvorgängen für Markerfarbwerte, Belichtungszeit oder Kalibration. Diese Betriebsmodi zeichnen sich durch einen Wechsel zwischen Tracking- und Konfigurationsbetrieb aus.

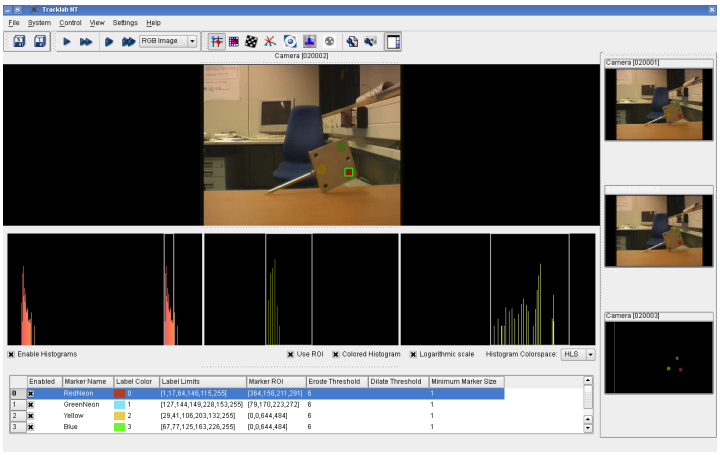


Abbildung 5.5: Tracklabplugin Farbmarkerkonfiguration

Abbildung 5.5 zeigt ein Modul zur manuellen Farbmarkerkonfiguration. Der Anwender erhält Zugriff auf die zugrundeliegenden Markerdefinitionen in der Systemkonfiguration, mit Farbraumgrenzen, Schwellwerten für Erosion und Dilatation, minimale Markergröße für Rekonstruktion usw. Zur definierten Einstellung der Farbgrenzen können Histogramme über frei selektierbare Bereiche des aktuellen Kamerabildes in verschiedenen Farbräumen berechnet und visualisiert werden. Das Modul verwendet einen generischen 2D-Datenviewer mit Erweiterungen zur Bereichsselektion.

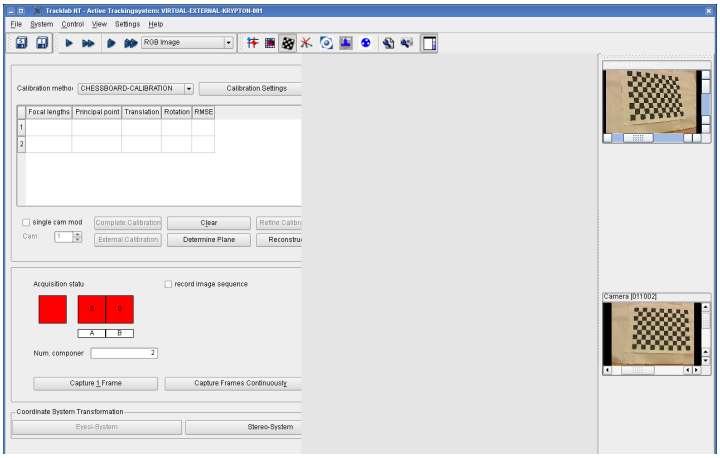


Abbildung 5.6: Tracklabplugin Kalibration

Abbildung 5.6 zeigt ein Modul für Kamerakalibrierung (eine Weiterentwicklung des Moduls aus [Handel 2004]). Es führt in einem interaktiven Prozess durch die interne

und externe Kalibrierung von Einzelkameras und Systemen (vgl. Abschnitt 3.2.1). Kalibrierungsoptionen sind über Dialoge einstellbar. Kontrolle über die Bildakquisition ist durch die Livebilder der Kameradatendisplays neben dem Plugin gegeben. Nach dem Optimierungsprozess wird das Ergebnis angezeigt und in einem in einem eingebetteten 3D-Szenendisplay visualisiert.

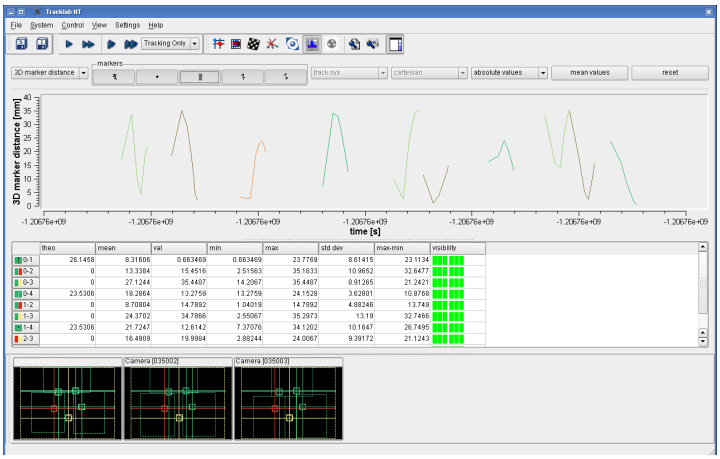


Abbildung 5.7: Tracklabplugin Datenaufnahme

Abbildung 5.7 zeigt ein Plugin zur Datenauswertung. Es werden verschiedene Parameter (Position, Pixelanzahl, Ausdehnung) der erfassten Marker aufgenommen und ihre zeitliche Veränderung visualisiert.

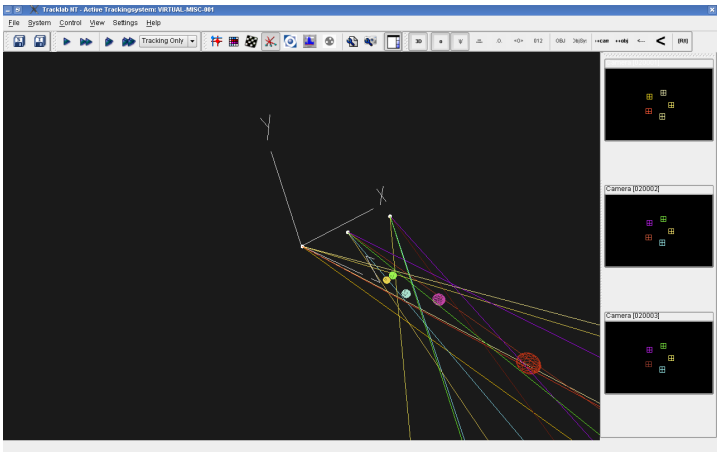


Abbildung 5.8: Tracklabplugin Rekonstruktion

Abbildung 5.8 zeigt ein 3D-Visualisierungsmodul mit einer rekonstruierte Szene des Trackingvolumens mit Kamerapositionen, Markern und Objekten zeigt. Es ermöglicht die Onlinekontrolle des Rekonstruktionsprozesses durch verschiedene Einblendungen und Darstellungsmodi. So können über die in Tracklab integrierte Werkzeugleiste Sehstrahlen, Objekte, Kameraöffnungswinkel, Rekonstruktionsfehler, etc. eingeblendet werden. Beobachtung der Szene aus den Kamerapositionen, aber auch freie Bewegung durch die Szene ermöglicht Detektion von Markerverdeckungen, unzureichenden Kameraerfassungsbereichen etc.

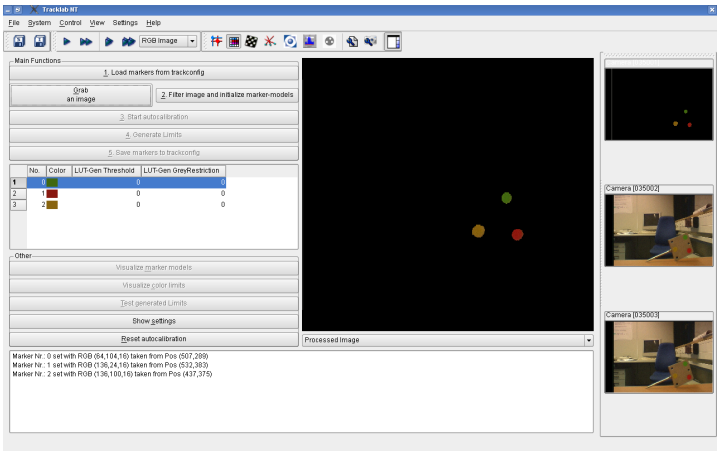


Abbildung 5.9: Tracklabplugin zur semi-automatisierten Konfiguration der Markerfarben

Abbildung 5.9 zeigt ein Modul zur semiautomatisierten Farbmarkerkonfiguration [Diederich 2005]. Es können Bildsequenzen mit variabler Beleuchtung, Markersegmentierung etc. aufgenommen werden. Der Anwender setzt Startwerte für Farbraum und Markerbereiche. Eine anschließende Optimierung über die Bildsequenz bestimmt die passenden Farbgrenzen der Marker (vgl. Abschnitt 3.2.3). In einer 3D-Darstellung des Farbraums werden die generierten Farbbereiche der Marker visualisiert.

Applikationsspezifische Plugins

Applikationsspezifische Plugins konzentrieren sich auf besondere Aspekte bestimmter Trackinganwendungen. Sie unterstützen bei Entwurf und Test neuer Methoden für Trackingbibliothek und Applikationen. Zu dieser Gruppe zählen auch dedizierte Kalibrationsplugins für spezielle Trackingsysteme. Damit entsprechen sie den Nutzungsszenarien der Klasse der Anwendungs- und Kernentwickler aus Abschnitt 5.1.1. Beispiele für solche Plugins werden in Kapitel 6 bei den jeweiligen Applikationen beschrieben – so das Kalibrationsmodul für den Augenoperationssimulator EYESI (Abschnitt 6.1), das Entwicklungsmodul für das Deformationstracking eines flexiblen Au-

geninterfaces (Abschnitt 6.2) und eine Testumgebung für ein Laparoskopietracking (Abschnitt 6.6).

5.4 Tracklab und MOSCOT als Entwicklungskit für Trackinglösungen

Die Pluginarchitektur von Tracklab bildet zusammen mit den grundlegenden Datenklassen aus der VRM-Trackingbibliothek eine definierte Programmierschnittstelle (API) für die Entwicklung von Trackinganwendungen. Während die Klassenstruktur der Trackingbibliothek die eigentliche Datenverarbeitungsfunktionalität bereitstellt, bilden die Tracklabrahmenanwendung und Modulbasisklasse die Basis zur Entwicklung angepasster graphischer Benutzerschnittstellen für Konfiguration und Betrieb der Systeme. Darüberhinaus bietet der Tracklabrahmen gewissermassen eine Sandbox-Umgebung für Entwicklung und Test von Anwendungen aber auch neuen Kernkomponenten außerhalb der eigentlichen Trackingbibliotheken.

Zusammen mit den Hardwarekomponenten kann das Gesamtsystem aus MOSCOT und Tracklab als ein Tracking-Development-Kit mit Hard- und Softwareintegration betrachtet werden. Es kombiniert die Möglichkeiten des Baukastenkonzepts von MOSCOT-Hardware und Software mit einer Anwendungs- und Entwicklungsumgebung, ermöglicht dem Entwickler direkten Systemzugriff und vereinfacht die Entwicklung von Hard- und Softwarekomponenten.

Kapitel 6

Anwendungen

Zu Beginn dieser Arbeit wurde eine Reihe von Anforderungen an den MOSCOT-Systembaukasten formuliert. Das Hauptaugenmerk liegt dabei auf den nicht-funktionalen Anforderungen, die in ihrer Gesamtheit eine einfache Adaptabilität an neuartige Aufgabenstellungen gewährleisten sollen. Zur Veranschaulichung, wie das Konzept von MOSCOT diese Anpassung an verschiedene Anwendungsfälle realisiert, werden im Folgenden einige trackingbezogene Projekte aus der VIPA-Arbeitsgruppe vorgestellt, bei denen das MOSCOT-System zum Einsatz kam. Ausgehend von den projektspezifischen Anforderungen an das optische Tracking wird die Adaption der Hard- und Softwarekomponenten sowie der Systemarchitektur an die jeweilige Aufgabenstellung gezeigt. Dabei wird auf die Darstellung von Komponenten und Architektur in den Kapiteln 3 und 4 aufgebaut und lediglich anwendungsspezifische Modifikationen werden ausführlicher dargestellt.

6.1 Instrumententracking für den Augenoperationssimulator EYESI

Die Erfassung der Benutzerinteraktion im Augenoperationssimulator EYESI stellt den Ausgangspunkt der Arbeiten im Bereich optischen Trackings innerhalb der Arbeitsgruppe VIPA dar [Ruf 2000]. Mit der Evolution des MOSCOT-Trackingsystems aus diesen Anfängen heraus musste auch die EYESI-Trackingrealisierung kontinuierlich den Veränderungen des zugrundeliegenden Frameworks angepasst werden. Diese Arbeiten wurden im wesentlichen vom Projektpartner VRmagic und den dortigen Entwicklern geleistet. Da das optische Tracking der Benutzerschnittstelle von EYESI quasi als Referenzanwendung für den MOSCOT-Systembaukasten fungiert und sehr gut grundlegende Konzepte illustriert, wird es hier in seinem aktuellen Entwicklungsstand beschrieben.

Projektbeschreibung

Der opthalmochirurgische Simulator EYESI [Schill et al. 1999; Wagner et al. 2002] dient als Trainings- und Ausbildungsgerät für verschiedene Arten von chirurgischen Eingriffen im menschlichen Auge. Aufbau und Funktionsweise wurden bereits in der Einführung in Abschnitt 1.2 beschrieben. Das optische Trackingsystem erfasst die Interaktion des Arztes mit dem beweglichen Augenphantom und den Instrumenten und stellt die rekonstruierten Positionsdaten dem Simulator als Eingangswerte bereit.

Anforderungen an das Tracking

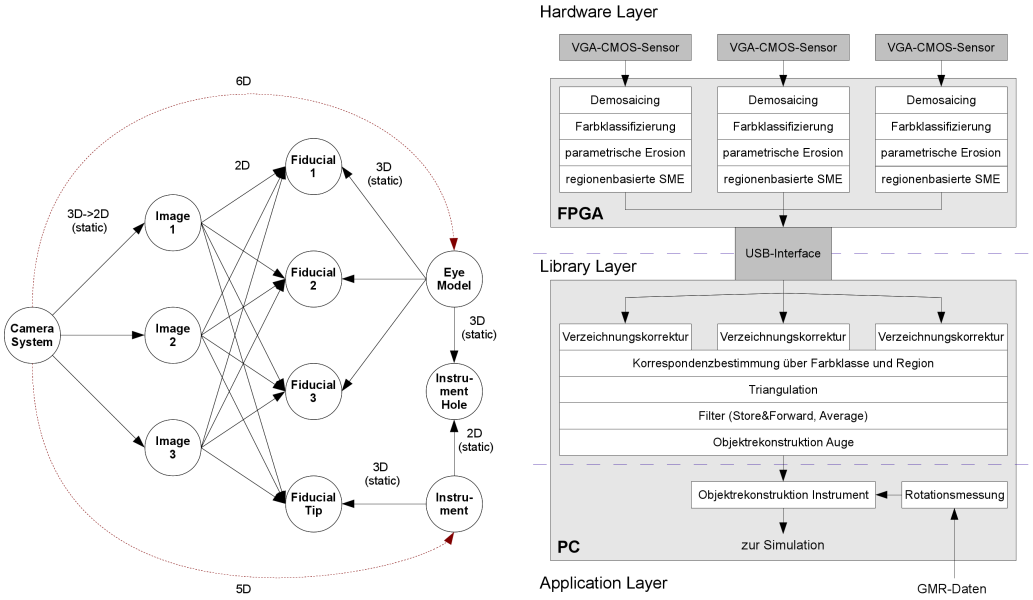
Wesentliche Charakteristika des Trackingaufbaus sind durch das Operationsszenario vorgegeben. So sind das zu erfassende Volumen und die zu trackenden Objekte ungewöhnlich klein (Augendurchmesser ca. 2,5cm, entsprechend einem Volumen von ca. 8 cm^3, Instrumentkörper: 0,9mm Durchmesser, ca. 10–15mm Eindringtiefe innerhalb des Auges). Die geringe Größe stellt das Haupthindernis für den Einsatz konventioneller Marker und Kamerasysteme dar.

Abbildung 6.1(a) zeigt die räumlichen Relationen anhand des SRG (Spatial-Relationship-Graphen). Gesuchte Größen des Trackingprozesses sind Position und Lage von Augenphantom und Instrumenten. Die beiden Instrumente haben nadelförmige Gestalt und werden über je ein definiertes Einstichloch in das Augphantom eingeführt, so dass nur die Instrumentspitze zuverlässig von den Kameras erfasst werden kann. Im Unterschied zum üblichen 6DOF-Tracking des Augenphantoms mit drei Objektmarkern kann die Instrumentenpose damit nur über einen Marker an der Instrumentspitze und das definierte Einstichloch bestimmt werden, so dass ein 5DOF-Instrumententracking vorliegt.[1]

Für den interaktiven Eindruck der Simulation ist eine niedrige Gesamtlatenz notwendig. Mit den bereits in Abschnitt 2.2.7 angeführten Abschätzungen von Wagner [2003] ergibt sich für den Simulator EYESI eine Trackinglatenz im Bereich von ca. 20ms, die nicht überschritten werden sollte. Als über den Benutzer rückgekoppeltes interaktives System muss für die Bewegungserfassung des Simulators keine hohe absolute Genauigkeit angestrebt werden. Essentiell sind vielmehr Stetigkeit und Auflösungsvermögen, d.h. kleine lokale Bewegungen der Instrumente müssen direkt und kontinuierlich abgebildet werden. Die Positionierungsgenauigkeit der Bewegung eines Mikrochirurgen wird von Riviere und Khosla [1999] mit 0,1mm angegeben, der Tremor der Hand liegt im Frequenzbereich von 8–10Hz. Diese Bewegungen müssen für eine überzeugende VR-Simulation abgebildet werden.

[1] Bei nicht rotationssymmetrischen Instrumenten (z.B. Pinzetten) wird der fehlende Freiheitsgrad der Rotation um die Längsachse über eine zusätzliche Sensorik (statisches gerichtetes Magnetfeld und GMR-Sensor im Instrumentengriff) bestimmt.

Etwaige auftretende Störungen in der Datenerfassung sollen durch den Einsatz von Filterstufen kompensiert werden. Weiterhin ist eine redundante Auslegung der Kameras erforderlich, da durch den gleichzeitigen Einsatz von zwei Instrumenten im Auge Markerverdeckungen auftreten können. Eine weitere wichtige Randbedingung ist ein niedriger Ressourcenbedarf, da die Trackingrekonstruktion auf der Host-CPU parallel zu Simulation und Visualisierung ausgeführt wird.



(a) Spatial-Relationship-Graph

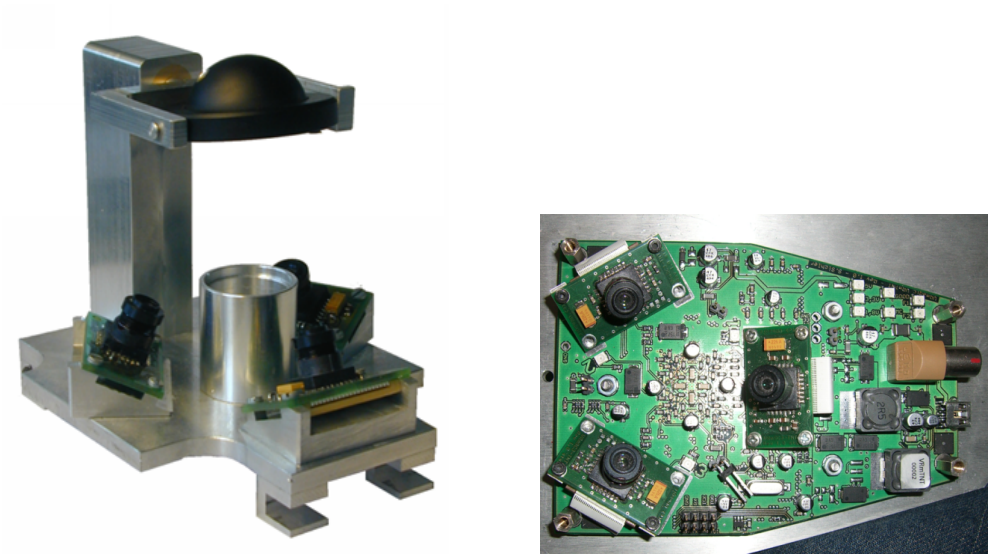
(b) Darstellung der realisierten Datenflussarchitektur

Abbildung 6.1: EYESI: SRG der Trackingkonfiguration und Adaption der Datenflussarchitektur

Realisierung

Das für den Simulator EYESI realisierte Konzept basiert auf einem Farbmarkertracking mit miniaturisierten VGA-CMOS-Kameras des Typs Hynix HV7131D, die das Augenphantom von der Unterseite her erfassen (Abbildung 6.2(a)). Passive Farbmarker bieten sich an, da sie unkompliziert in Form von Farbmarkierungen an Instrumenten und Auge anzubringen sind und im Augeninterface kontrollierte Beleuchtungsbedingungen herrschen. Genauigkeitseinbußen, die durch unpräzisen Farbauftrag bzw. die bekannte Problematik der Farbkonstanz entstehen, sind durch den interaktiven Charakter dieser VR-Anwendung tolerierbar. Eine Kodierung mit verschiedenen Farben ermöglicht darüber hinaus die Unterscheidung einzelner Instru-

mente. Die Farbe der drei Augenmarker kann identisch sein, da durch den eingeschränkten Bewegungsraum des Auges in seiner Aufhängung jedem Marker definierte Bildregionen zugeordnet sind und er damit eindeutig identifizierbar ist (vgl. die bildregionenbezogene Merkmalsbestimmung in Abschnitt 3.3.7). Drei Kameras in einer symmetrischen 120°-Anordnung kommen zum Einsatz, um gegenseitige Verdeckungen der Instrumente zu kompensieren. Eine speziell entwickelte Platine integriert die drei miniaturisierten VGA-Sensorplatinen mit einem FPGA-basierten Bildvorverarbeitungsmodul und einem USB-Kommunikationsmodul und schafft einen kompakten integrierten Aufbau der Trackinghardware (Abbildung 6.2(b) EYESI-Trackingplatine).



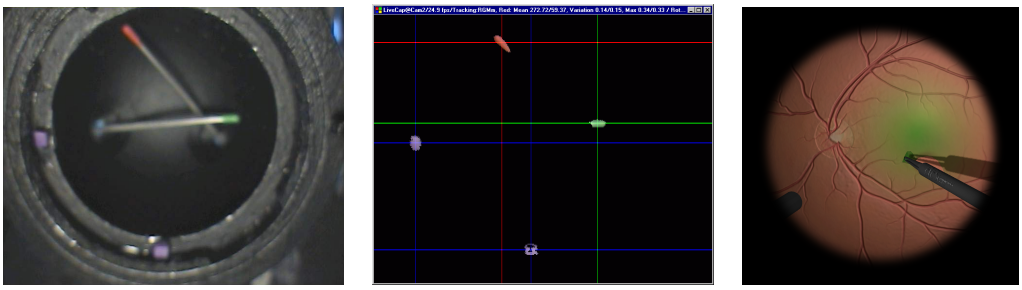
(a) EYESI-Trackingaufbau (b) EYESI-Trackingplatine

Abbildung 6.2: Künstliches Auge und die Trackingplatine mit den drei Kameras des EYESI-Intrumenten- und Augentrackings

Abbildung 6.1(b) zeigt die Datenflussarchitektur von MOSCOT für die Umsetzung des optischen Trackings von EYESI. Durch die kontrollierte Umgebungsbeleuchtung beschränken sich die Bildverarbeitungsmodule zur Markerdetektion im FPGA auf eine Farbklassifizierung mit anschliessender parametrischer Erosion sowie Merkmalsextraktion der resultierenden Farbbereiche, jeweils in der für den Marker vordefinierten Bildregion. Das FPGA integriert die parallelen Verarbeitungsketten für alle drei Kameras und sendet die erfassten 2D-Markerdaten über die USB-Schnittstelle an die Trackingbibliothek im Simulationsrechner.

Die Rekonstruktion kann durch die eindeutige Markeridentifikation auf aufwendige

Korrespondenzanalyseverfahren verzichten und über direkte Triangulation die Position von Augenmarkern bzw. Instrumentenspitze bestimmen. Kurze Störungen in den erfassten Daten durch Rauschen, Übertragungsfehler etc. werden durch Filterung der rekonstruierten 3D-Markerkoordinaten weitestgehend unterdrückt. Im Verlauf der Entwicklung zeigte sich, dass einfache Filter hinreichende Ergebnisse erzielen, so dass auf aufwendigere Kalmanfilter verzichtet werden kann. Die Lichtquellenpositionen werden bei der Operation über einen Store & Forward-Filter geführt, der bei Datenausfall die letzte Position wiedergibt. Die anderen Instrumente erfahren eine gleitende Durchschnittsbildung durch einen Mittelwert-Filter.



(a) Kamerasicht in das EYESI-Augeninterface

(b) Detektierte Marker

(c) Gerenderte Ansicht von Auge und Instrumenten im Simulator

Abbildung 6.3: Kamerasicht und gerenderte Sicht auf Auge und Instrumente im Simulator EYESI

Aus den gefilterten Positionen der Augen- und Instrumentmarker errechnen sich die Posen von Augeninterface und Instrumenten. Einen Sonderfall stellt die Bestimmung der Ausrichtung der Instrumente dar, da sich die Einstichlöcher mit dem Auge mitbewegen. Unter Verwendung der während des Kalibrierens eingemessenen Position des Einstichlochs am Auge und der getrackten Instrumentenspitze und Augenpose kann die Instrumentenausrichtung bestimmt werden. Die softwareseitige Realisierung erfolgt über ein dediziertes Datenverarbeitungsmodul zur Instrumentenrekonstruktion, das den Standardrekonstruktionsmodulen für Marker- und Objektrekonstruktion in der Transceiverkette nachgeschaltet ist.

Abbildung 6.3(a) zeigt den Blick der Trackingkameras in das EYESI-Augenphantom, Abbildung 6.3(b) die detektierten Marker und Abbildung 6.3(c) eine gerenderte Ansicht von Instrumenten und Auge im simulierten Operationsmikroskops.[2]

[2]Während die Trackingkameras von unten in die als Augenphantom verwendete Metallhalbkugel blicken, gibt die gerenderte Rekonstruktion von EYESI-Intrumenten und Auge den Blick von oben über ein simuliertes Operationsmikroskop wieder.

Zur Kalibrierung des EYESI-Trackingsystems wurden bei der VRmagic GmbH spezialisierte Plugins für die Konfigurationsumgebung Tracklab entwickelt, die das Kalibrieren der Kameras, die Konfiguration der Farbmarker und die Registrierung des Auges inklusive Augenmarkern und Einstichlöchern vereinfachen und automatisieren.

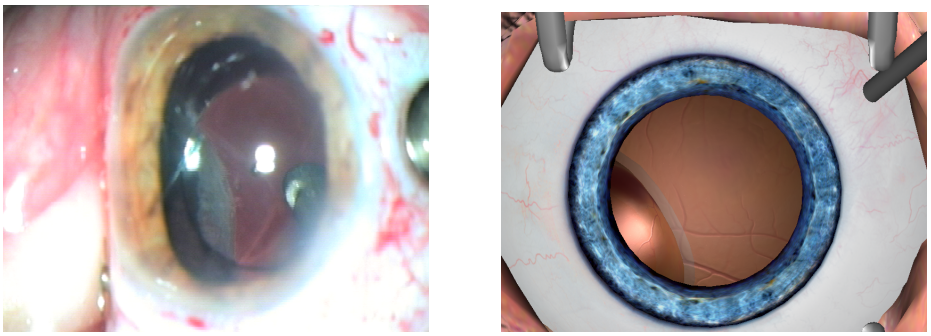
Ergebnisse und Diskussion

Geschwindigkeit, Auflösung und Latenz des vorgestellten Trackingaufbau liegen innerhalb der geforderten Bereiche für den EYESI-Simulator. Die Bildwiederholfrequenz der Kameras im Trackingmodus beträgt ca. 48Hz und ihre Auslesezeit 21ms. Die reine Verarbeitungslatenz bleibt unter 3ms. Die Auflösung des EYESI-Trackings wurde mit $30\mu m$ gemessen [Schill 2001, Kap. 6.2.1]. Augenchirurgen bestätigen die gute Immersion der simulierten Intervention und die quasi reale Interaktion mit Auge und Instrumenten. Das MOSCOT-System ermöglicht somit die Realisierung eines massgeschneiderten Trackingsetups für dieses spezielle Simulatorszenario, bei gleichzeitig ressourceneffizienter Umsetzung. Durch die Anpassbarkeit des Software-Frontends Tracklab konnte weiterhin ein massgeschneidertes Softwarefrontend für einfache Kalibration und Konfiguration sowie Test und Kontrolle des EYESI-Trackingaufbaus außerhalb der eigentlichen Simulatoranwendung bereitgestellt werden.

6.2 Deformationstracking eines flexiblen Augenphantoms für EYESI

Projektbeschreibung

Im Verlauf mancher ophthalmochirurgischer Eingriffe dellt der Chirurg den Augapfel seitlich ein, um durch das Stereomikroskop einen erweiterten Bereich der Netzhaut einsehen zu können (Abbildung 6.4). Ein solcher Eingriff soll im Simulator EYESI nachgebildet werden. Wird die Eindellungseinstellung über ein Softwaremenu realisiert, führt dies einerseits zu einer Unterbrechung des Operationsflusses, andererseits wird auch nicht die erforderliche Koordination beider Hände trainiert. Ein Simulator, der nach größtmöglichem Realismus der virtuellen Operation strebt, sollte daher eine manuelle Deformationsinteraktion mit dem Augeninterface implementieren. Als Vorarbeiten für eine mögliche Erweiterung des Simulators EYESI wurde die Machbarkeit einer solchen Deformationserfassung mittels optischen Trackings untersucht. Beier [2006] beschreibt die Details der Adaption des MOSCOT-Systems an diese Aufgabenstellung.



(a) reale Operation (b) simulierte Operation

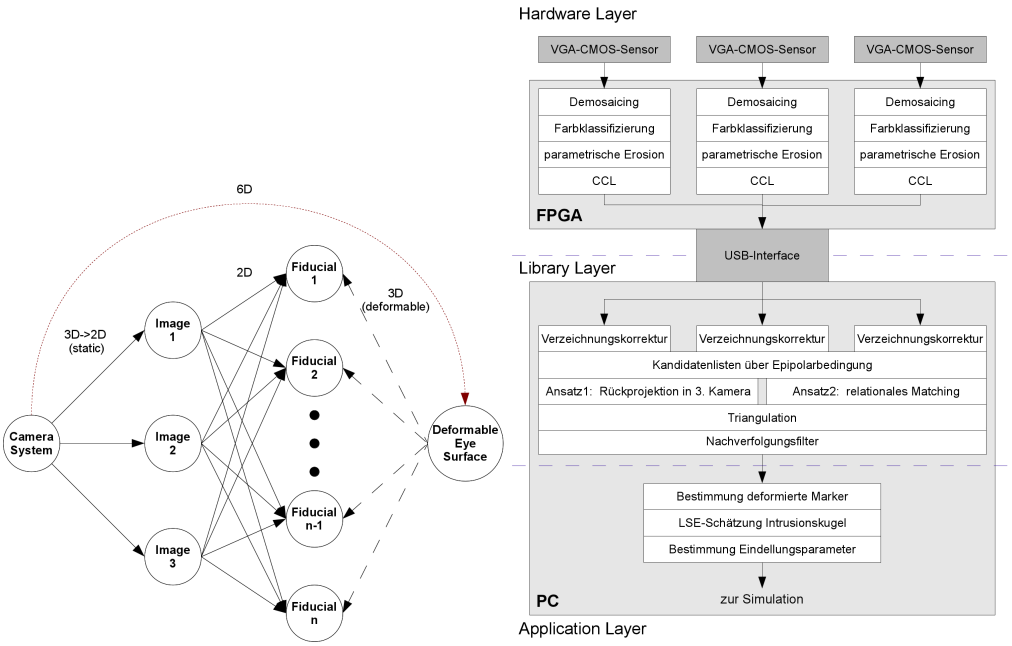
Abbildung 6.4: Eindellung des Augapfels während eines ophthalmo-chirurgischen Eingriffs, reale und simulierte Ansicht

Anforderungen an das Tracking

Das Trackingsystem soll Position und Tiefe der Eindellung erfassen. Für diese Machbarkeitsstudie wurde auf ein Instrumententracking verzichtet, und nur die Deformation des Auges erfasst. Abbildung 6.5(a) zeigt die einfachen räumlichen Relationen anhand des Spatial-Relationship-Graphen. Zentraler Punkt der Realisierung ist die Lösung des Korrespondenzproblems für die nicht-starre Markeranordnung, da für eine hinreichende Deformationserfassung mit einer größeren Zahl von Markern gearbeitet werden muss. Hieraus ergibt sich direkt eine weitere Anforderung nach kleinen Markern, die leicht auf der Kugelinnenfläche fixierbar sein müssen.

Die absolute Erfassungsgenauigkeit des Deformationstrackings spielt bei EYESI durch die Interaktivität des Deformationsprozesses nur eine untergeordnete Rolle und auch die Erfassungslatenz ist aufgrund der tendenziell langsamen Bewegung beim Eindellvorgang unkritisch. Genauigkeiten von ca. 1mm und Latenzen von 100ms sind als ausreichend anzusehen. Wichtig sind bei dieser Anwendung jedoch wieder geringe Rechenzeitanforderungen für die Rekonstruktion im Hostrechner des Simulators.

Der Aufbau des Systems soll sich am vorhandenen EYESI-Tracking orientieren, um einen einfachen Wechsel auf das weiterentwickelte deformierbare Augeninterface zu ermöglichen. Die Problemstellung dieses Trackingszenarios ist damit recht ungünstig, da die etablierten Standardverfahren zum Tracking von Einzelmarkern bzw. nichtstarren Körpern entweder einzeln schaltbare Marker oder eine größere Anzahl von Kameras einsetzen. Beides ist in diesem Anwendungsfall nicht möglich. Stroian et al. [2004] untersuchen, wie die Anordnung der Marker die Rekonstruktion von Ghost-Markern verhindern kann. Der Ansatz ist allerdings bei einer Vielzahl von Markern auf kleinem Raum nur unzureichend umsetzbar. Andere Ansätze zur Echtzeitrekon-



(a) Spatial-Relationship-Graph

(b) Darstellung der realisierten Datenflussarchitektur

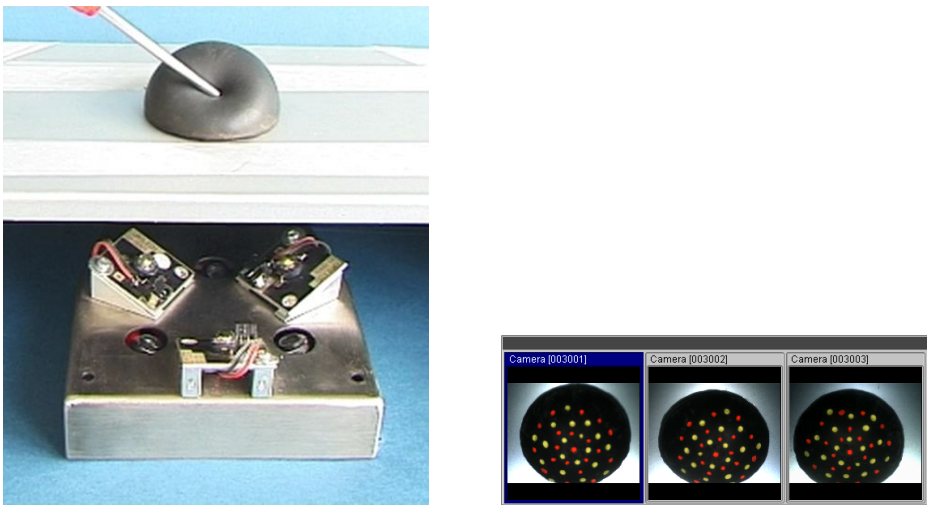
Abbildung 6.5: EYESI-Deformationstracking: SRG der Trackingkonfiguration und Adaption der Datenflussarchitektur

struktion deformierbarer Oberflächen finden sich z.B. in [Pilet et al. 2005], [Ramey et al. 2004] und [Lau et al. 2004]. Diese basieren jedoch nicht auf einem markerbasierten Ansatz, sondern auf strukturierten bzw. texturierten Oberflächen und bildbasierter Zuordnung der Korrespondenzstellen.

Realisierung

Es wird der in Abschnitt 6.1 beschriebene Trackingaufbau verwendet, jedoch die starre Metallhalbkugel des Standardauges durch eine deformierbare Gummihalbkugel ersetzt (Abbildung 6.6(a)). Als Marker bieten sich wieder kleine flächige Farbmarker an, die recht einfach mittels Dispersionsfarbe auf die Innenseite der Gummihalbkugel appliziert werden können (Abbildung 6.6(b)). Verschiedene Farben werden verwendet, um die Marker in Gruppen zu unterteilen, die die kombinatorischen Mehrdeutigkeiten bei der Korrespondenzbestimmung verringern.

Abbildung 6.5(b) zeigt die Adaption der MOSCOT-Datenflussarchitektur an die Aufgabenstellung. In der Markerdetektionsstufe erfordert die Vielzahl gleichfarbiger



(a) Prototyptracking deformierbares Augeninterface

(b) Markeranordnung auf der Kugelinnenfläche

Abbildung 6.6: Aufbau des Funktionsmusters zum Deformationstracking und Blick der Kameras auf die Markeranordnung der Kugelinnenfläche

Marker den Ersatz der ursprünglichen bildregionenbasierten Merkmalsbestimmung durch ein lokales Connected-Component-Labeling. Die Implementierung erfolgt im FPGA der EYESI-Trackinghardware, wobei die übrigen Module der Bildverarbeitungskette unverändert bleiben. Die Rekonstruktion der 3D-Markerpositionen im Hostrechner gestaltet sich deutlich aufwendiger als beim EYESI-Instrumententracking. Da die Markerbilder nicht eindeutig zuzuordnen sind und auch keine Rekonstruktion für Starrkörper anwendbar ist, müssen verschiedene Techniken bei der Korrespendenzanalyse kombiniert werden.

Zwei Ansätze wurden detaillierter untersucht [Köpfle et al. 2007]. Bei beiden Ansätzen werden im ersten Schritt mögliche Korrespondenzkandidaten für jedes Kamerapaar über die Epipolarbedingung gefiltert, für den zweiten Ansatz werden die Bilder zusätzlich paarweise rektifiziert (s. Anhang A.7). Die Korrespondenzkandidaten werden in den folgenden Schritten weiteren Plausibilitätsbedingungen unterworfen.

Die erste Variante nutzt die Erweiterung der Epipolarbedingung auf das 3-Kamera-Setup und verifiziert bzw. eliminiert nach Algorithmus 3.3 testweise rekonstruierte Korrespondenzkandidaten über Rückprojektion in die dritte Kamera (siehe auch Abbildung 3.21). Bei ungünstiger Markerkonstellation verbleiben jedoch Mehrdeutigkeiten, die zur fehlerhaften Rekonstruktion von Phantommarkern führen. Mit der Abschätzung aus Gl. (3.31) für Dreikamerasysteme ergibt sich die Anzahl der erwar-

teten Mehrdeutigkeiten für ein Bildtripel zu:

$$N_a = \frac{4(n^2 - n) \cdot \varepsilon^2}{F \cdot sin\alpha}(1 + \frac{b_{12}}{b_{13}} + \frac{b_{12}}{b_{23}}) \approx 0,4 \tag{6.1}$$

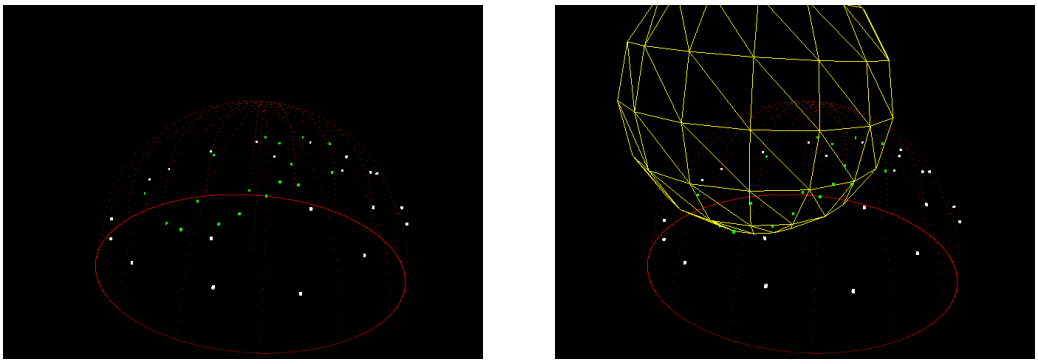
(für 20 Marker, Epsilontoleranzbereich = ±5 Pixel, 3 VGA-Kameras, Anordnung im gleichseitigen Dreieck, EYESI-Trackingplatine). Dieser Wert deckt sich mit den empirischen Ergebnissen, dass fehlrekonstruierte Marker durchaus auftreten, jedoch nicht in jedem Frame. Maas [1992] zeigt, dass die Anzahl der Mehrdeutigkeiten bei einer Dreikameraanordnung in Form eines gleichseitigen Dreiecks minimal wird. Diese Geometrie ist bei der Anordnung der EYESI-Trackingkameras bereits gegeben, so dass durch eine Änderung des Kamerasetups keine Verbesserung erreicht werden kann. Die wenigen verbleibenden fehlerkannten Marker müssen daher in einem abschliessenden Filterungsschritt detektiert und eliminiert werden.

Das zweite untersuchte Verfahren gründet auf einem relationalen Matchingansatz (vgl. Abschnitt 3.4.4). Zhang et al. [1995] beschreiben dieses Verfahren, das auf der Grundannahme basiert, dass um korrespondierende Punkte eine ähnliche Umgebung von Markern vorzufinden ist. Diese Ähnlichkeit wird über die Distanz zu benachbarten Punkten bestimmt. Mittels einer Relaxierung werden die Zuordnungen dann für den besten Schätzwert über alle Marker optimiert. Die Zuordnungen mit dem höchsten Schätzwert werden für die Rekonstruktion herangezogen. Auch bei diesem Ansatz entstehen jedoch fehlrekonstruierte Marker, die herauszufiltern sind.

Bei beiden Ansätzen erfolgt die Filterung fehlrekonstruierter Marker durch Nachverfolgung aller rekonstruierten Punkte. Da fehlerkannte Marker keine stetige Bewegung vollführen, sondern vielmehr im Trackingvolumen "springen", können sie identifiziert werden. Hierzu wird eine maximal zulässige 3D-Bewegungsdistanz Δx_{max} von Punkten in aufeinanderfolgenden Frames definiert. Nur Marker, die sich innerhalb dieses Abstands um einen Punkt aus dem vorangeganen Frame befinden, werden als gültig angesehen.

Aus den rekonstruierten 3D-Punkten ist schließlich die Deformation des Auges zu bestimmen.[3] Sie wird bei EYESI vereinfacht als zweite Kugel modelliert, die die Augenkugel von außen eindellt. Das Deformationstracking muss demnach die Kugelparameter dieser Intrusionskugel sowie den tiefsten Kontaktpunkt bereitstellen, d.h. die Polarkoordinaten der Kontaktstelle und ihre Eindringtiefe. Hierzu werden über einen Distanzschwellwert die von der bekannten unverformten Augenkugel abweichenden Markerpositionen bestimmt. Aus mindestens 4 deformierten Markern können daraufhin die Parameter der Intrusionskugel bestimmt werden (Abbildung 6.7).

[3]Bei diesem Funktionsprototypen wird nur die Deformation und keine Pose des Augenphantoms bestimmt. Diese kann später analog zum bisherigen Trackingaufbau von EYESI erfasst werden, indem drei zusätzliche Marker an der (starren) Aufhängung des Augenphantoms angebracht werden.



(a) rekonstruierte Marker mit Augenkugel (deformierte Marker in grün)

(b) rekonstruierte Eindellungskugel

Abbildung 6.7: 3D-Darstellung der rekonstruierten Marker, der Augenkugel und der Eindellungskugel

Pratt [1987] beschreibt hierzu ein Verfahren zur Kleinste-Quadrate-Schätzung für algebraische Flächen mit einer Spezialisierung für Kugeln. Die Robustheit gegenüber Ausreißern kann durch die Anwendung des RANSAC-Verfahrens auf die Kugelschätzung erhöht werden. Der Schnittpunkt der Verbindungsgerade zwischen den Kugelmittelpunkten mit der Oberfläche der Intrusionskugel definiert schließlich die Kontaktstelle.

Die Softwarerealisierung erfolgt über ein spezielles Transceivermodul, das in der Datenverarbeitungskette den Standardrekonstruktionsmodulen nachgeschaltet wird. Es hat Zugriff auf die rekonstruierten Markerkoordinaten und führt die Deformationsrekonstruktion aus. Abbildung 6.8 zeigt die Benutzeroberfläche des speziellen Tracklab-Plugins, das für Entwicklung und Tests des Augendeformationstrackings geschaffen wurde.

Ergebnisse und Diskussion

Untersuchungen mit dem Prototypen zeigen die Machbarkeit eines Deformationstrackings für den Simulator EYESI, obwohl die Problemstellung ungünstig für Standardlösungen zum Tracking nichtstarrer Körper ist. Durch angepasste Marker- und Oberflächenrekonstruktion können fehlrekonstruierte Marker ausreichend gut erkannt und gefiltert werden. In einem Setup mit zwei Codierungsgruppen von jeweils ungefähr 20 gleichfarbigen Markern können mit den vorgestellten Verfahren ca. 95% der Marker korrekt rekonstruiert werden.

Messungen an dem Prototypen ergaben eine Genauigkeit der Deformationsbestimmung im Bereich von ca. 1mm. Die gemessene Verarbeitungslatenz für den ersten

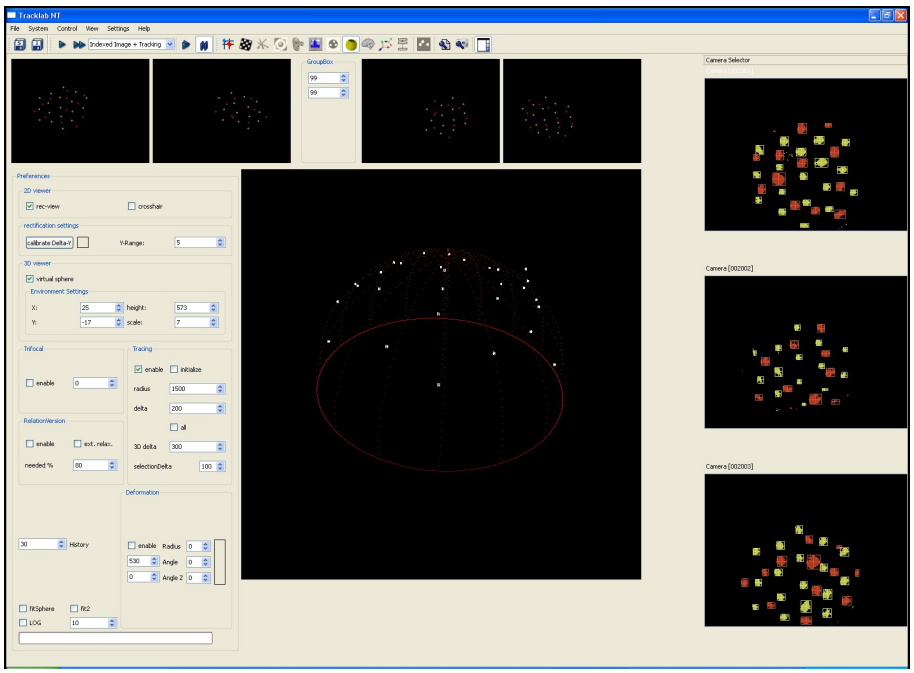


Abbildung 6.8: Tracklab-Plugin für die Entwicklung des Deformationstracking

Ansatz beträgt 2ms, für den zweiten Ansatz 9ms [Köpfle et al. 2007] (ohne Auslesezeit der Kameras). Der effizientere erste Ansatz benötigt jedoch drei Kameraansichten zur sicheren Rekonstruktion, so dass die Fehlerkennungsrate bei Verdeckungen durch Instrumente voraussichtlich steigen wird. Der zweite Ansatz ist hier robuster.

Für die weitere Integration in den Simulator ist das Deformationstracking mit dem normalen Instrumententracking zu kombinieren und in einen gemeinsamen Aufbau zu überführen. Durch das gleichartige Trackingkonzept beider Implementierungen ist dies für einen zukünftigen Simulator unkompliziert realisierbar. Dabei zeigen sich die Vorzüge der Farbkodierung, da die Instrumentmarker anhand der Farbe identifizierbar sind und so das komplexere Deformationstracking nicht beeinflussen können. Weitere Untersuchungen mit dem integrierten Trackingaufbau müssen zeigen, welcher der beiden beschriebenen Ansätze unter den diametralen Kriterien Robustheit und Ressourcenbedarf für den Simulator geeigneter ist. Die verwendete Deformationsmodellierung lässt sich direkt auf andere Flächen zweiter Ordnung übertragen, für komplexere Formen sind jedoch aufwendigere Verfahren zur Rekonstruktion der deformierten Fläche aus den rekonstruierten Markerpunkten notwendig.

6.3 Schnelles latenzarmes Tracking für einen handgehaltenen Chirurgieroboter

Projektbeschreibung

Das Operationsrobotersystem ITD (Intelligent Tool Drive) [Pott et al. 2003] ist Entwicklungsgegenstand einer Kooperation des ICM mit dem Lehrstuhl für Automation, dem Klinikum Mannheim sowie zwei mittelständischen Unternehmen. Es basiert auf einer handgehaltenen selbststabilisierenden Maschine, die den Chirurgen als aktiv korrigierendes Werkzeug bei der intraoperativen Knochenbearbeitung unterstützt (Abbildung 6.9). Hierzu werden mittels umfangreicher Sensorik die Positionen von Patient und Werkzeug relativ zueinander erfasst und in allen 6 Freiheitsgraden aktiv ausgeregelt [Pott 2007]. Dies geschieht derart, dass Abweichungen von der präoperativ geplanten Trajektorie aufgrund von Patientenbewegung oder Handtremor des Chirurgen ausgeglichen werden. Die VIPA-Gruppe entwickelt im Rahmen der Kooperation das optische Tracking zur Positionserfassung von Patient und Roboter. Während des Eingriffs werden Knochen und Roboter durch Marker signalisiert und von stationären Kameras erfasst.

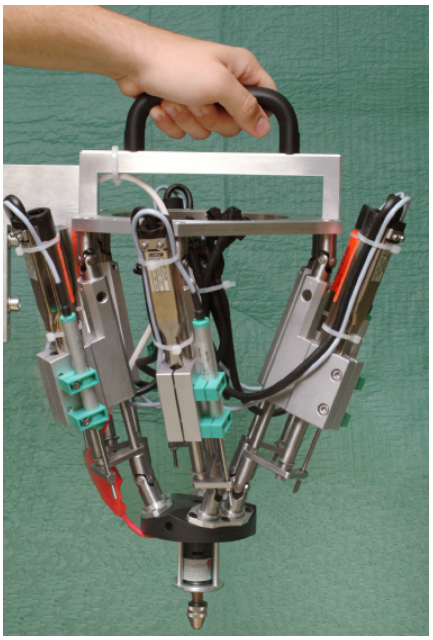


Abbildung 6.9: Funktionsmuster des Operationsroboters ITD

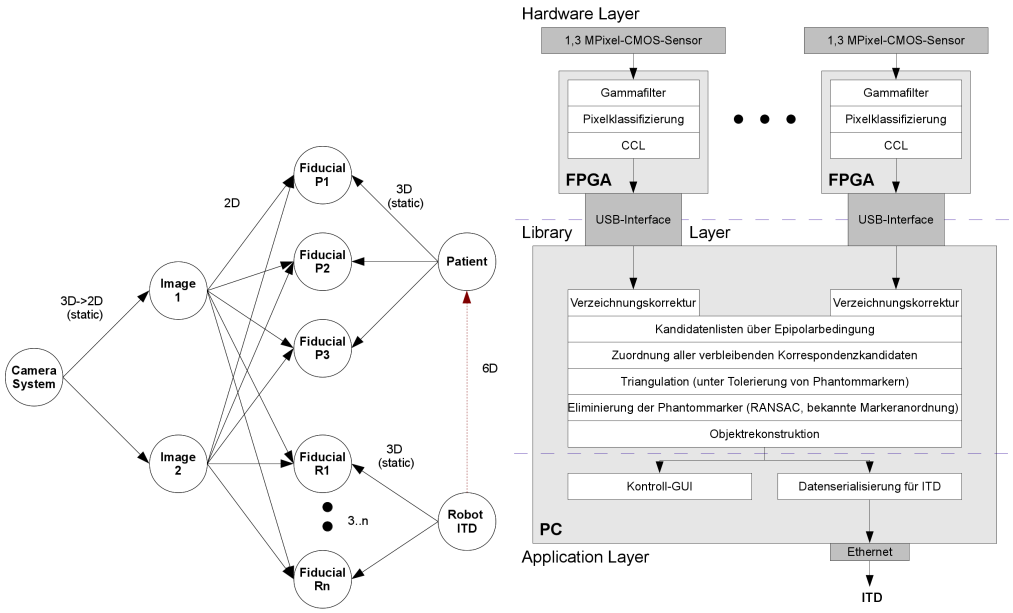
Anforderungen an das Tracking

Abbildung 6.10(a) zeigt den Spatial-Relationship-Graphen für die Erfassung der räumlichen Relativposition von Roboter und Knochen. Hauptanforderungen an dieses eher konventionelle Trackingszenario sind hohe Genauigkeit, hohe Bildaufnahmerate und eine sehr geringe Latenz. Letztere ist insbesondere notwendig, um ein gegenphasiges Aufschwingen des Roboterregelkreises durch verzögerte Detektion von Bewegungsänderungen zu vermeiden. Laut Halliday et al. [1999] liegt der Tremor von Hand und Unterarm im Frequenzbereich von 8–12 Hz. Auf dieser Grundlage wurden in Zusammenarbeit mit Regelungstechnikern am Lehrstuhl für Automation folgende Randbedingungen für das optische Tracking des Roboters definiert [Wagner et al. 2004]: Abtastfrequenz von mindestens 100–150Hz (im wesentlichen durch die geforderte Kameraausleselatenz von max 10ms bedingt – die zu beobachtenden Frequenzen sind deutlich niedriger), Genauigkeit im Submillimeterbereich (Zielvorgabe 0,25mm) sowie Gesamtlatenz von unter 20ms für die vollständige Prozesskette von Bildaufnahme bis zur Bereitstellung der 3D-Daten. Knappe et al. [2003] und Kuo et al. [2004] präsentieren ähnliche Ansätze der Kontrolle eines chirurgischen Roboters durch ein optisches Trackingsystem. Als handgehaltenes System stellt der Roboter ITD jedoch höhere Anforderungen an die Lageregelung und damit an die Latenz der optischen Positionsbestimmung.

Realisierung

In Voruntersuchungen wurde die Machbarkeit des Konzepts und das Zusammenspiel der Systemkomponenten (optisches Tracking, Roboter und Positions- und Lageregelung) demonstriert. Dieser erste Versuchsaufbau wurde noch mit Analogvideokameras und am Roboter angebrachten Farbmarkern durchgeführt (vgl. die Beschreibung in [Köpfle et al. 2004b] und Abbildung 6.11). Im Unterschied zur üblichen Registrierung passiver Objekte muss der Roboter als Objekt mit aktiven Freiheitsgraden zusätzlich eingemessen werden. Dabei ist das interne Koordinatensystem der Bewegungsachsen des Roboters auf das Objektkoordinatensystem des Trackingsystems abzugleichen. Im Rahmen der Testläufe wurde hierfür ein spezialisiertes Plugin für die Konfigurationsumgebung Tracklab entwickelt, das die interaktive Registrierung von Tracking- und Roboterkoordinatensystemen erlaubt.

Für die finale Auslegung des Systems mit den erforderlichen hohen Genauigkeiten wird eine robustere Markersignalisierung mittels IR-Licht gewählt. Übliche Einschränkungen von aktiven Infrarotmarkern bei Plazierung und Verkabelung sind aufgrund der Größe des Roboters und der ohnehin notwendigen Kabelanbindung unkritisch. Das System unterstützt aktive LED-Marker oder passive retroreflektierende Marken mit IR-Illumination an den Kameras. Für die Erfüllung der Auflösungs- und Geschwindigkeitsanforderungen kommen intelligente Kameras mit schnellen hoch-



(a) Spatial-Relationship-Graph

(b) Darstellung der realisierten Datenflussarchitektur

Abbildung 6.10: ITD: SRG der Trackingkonfiguration und Adaption der Datenflussarchitektur

auflösenden CMOS-Bildsensoren des Typs Micron MT9M413 [Micron Technology Inc. 2004] zum Einsatz (vgl. die Beschreibung und Abbildung in Abschnitt 4.3.4). Die Sensoren erzeugen bei einer Auflösung von 1280x1024 Pixeln und den geforderten 150Hz Bildwiederholfrequenz einen Datenstrom von ca. 190MB/s, der parallel über 10 Pixeldatenbusse ausgegeben wird. Zur lokalen Weiterverarbeitung der Bilddaten in den Kameras dient das in Abschnitt 4.3.2 beschriebene Bildverarbeitungsmodul aus [Hegner 2005]. Das FPGA serialisiert den Pixeldatenstrom und führt nach der Bildvorverarbeitung ein Connected-Component-Labeling zur Markerdetektion aus. Damit kann der Datenstrom auf wenige 100 KB/s reduziert werden, so dass die USB-Kommunikationslatenz minimiert wird. Die 3D-Rekonstruktion im Hostrechner erfolgt nach Filterung über Epipolargeometrie mittels eines Starrkörpermatchings auf die bekannten Markeranordnungen von Roboter und Knochentarget mit jeweils drei Markern.[4] Abbildung 6.10(b) zeigt die Auslegung der MOSCOT-Datenflussarchitektur für diesen Anwendungsfall.

[4]Zusätzliche Marker zur redundanten Objekterfassung sind in den nächsten Entwicklungsstufen vorgesehen.

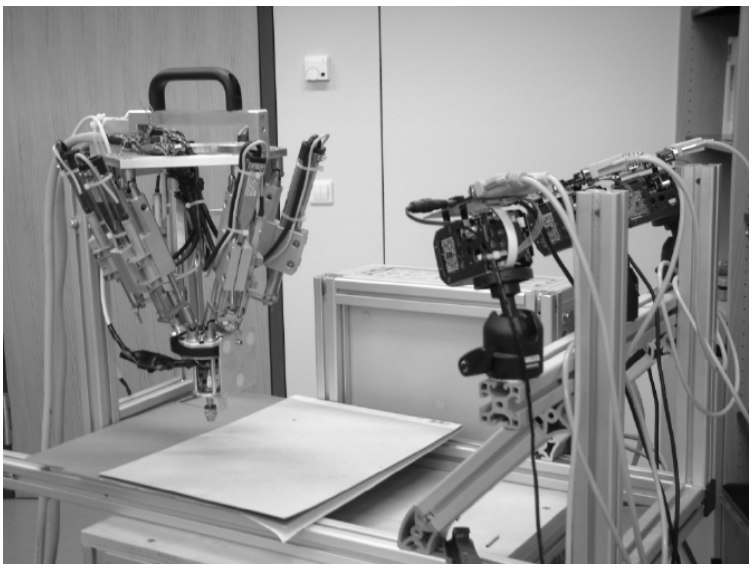


Abbildung 6.11: Teststand für Versuche mit dem Funktionsmuster des ITD-Gesamtsystems

Die Anbindung des Trackingsystems an die Roboterkontrolle und das zugehörige Regelungssystem erfolgt über eine Ethernetverbindung. Die Kommunikationslatenz von Fast-Ethernet kann bei direkter Punkt-zu-Punkt-Verbindung unter 1ms gehalten werden [Welsh et al. 1996]. Eine Kalmanfilterung der Trackingdaten zur Bewegungsprädiktion und Verdeckung der Verarbeitungslatenzen ist sinnvoll, wird jedoch im ITD-Setup nicht innerhalb des Trackingsystems sondern durch das Regelungssystem des Roboters selbst ausgeführt. Es fusioniert die Positionsdaten des Trackingsystems mit weiteren Sensordaten von Inertialsensoren und Motorencodern und kann durch eine Modellierung des Gesamtsystems eine bessere Basis für die Filterung bereitstellen.

Ergebnisse und Diskussion

Die Voruntersuchungen zur Integration des Gesamtsystems ITD wurden wie erläutert mit einem Farbmarkertracking mit PAL-Analogvideokameras durchgeführt [Pott et al. 2004]. Versuchsbohrungen zeigten eine durchschnittliche Abweichung des Gesamtsystems von 3mm für den Bohreintrittspunkt und einen Tiefenfehler von 1mm. Die Genauigkeit des Trackingteils allein lag durch das Farbmarkersetup und die Analogkameras mit einem gemessenen RMS-Fehler von 1–1.5mm (je nach Markerfarbwert) erwartungsgemäß noch deutlich über den angestrebten Werten. Latenzmessungen für das Tracking ergaben Werte um 35ms, wovon ca. 25ms auf kameraspezifische Latenzen für Belichtungs- und Auslesezeit entfallen. Damit erreicht die reine Daten-

verarbeitungslatenz mit unter 10ms bereits den angestrebten Bereich [Köpfle et al. 2004a].

Erste Resultate der Aufrüstung des ITD-Trackings mit den beschriebenen 1,3-Megapixel-Hochgeschwindigkeitskameras zeigen die erwarteten Verbesserungen. Durch die leistungsfähigeren Kamerasensoren und Bildverarbeitungskomponenten ergibt sich ein verbessertes Zeitverhalten des Systems. Die Auslesezeit der Kameras beträgt bei 33Mhz Pixeltakt $t_{readout} = 1280 \cdot 1024\text{Pixel}/10\text{Pixelpipelines}/33\text{Mhz} \approx 4\text{ms}$. Die Verarbeitungszeit der Bildverarbeitungspipeline ergibt sich bei einer Arbeitsfrequenz des Connected-Component-Labeling von ca. 80 Mhz zu $t_{processing} = 1280 \cdot 1024\text{Pixel}/80\text{Mhz} \approx 16\text{ms}$, was die Kamerafrequenz aktuell noch auf ca. 60Hz beschränkt. Um die volle Kamerabildrate verarbeiten zu können, wird daher eine parallele Auslegung der Bildverarbeitungspipelines im FPGA notwendig. Das Bild wird dafür in senkrechte Bereiche aufgeteilt, so dass jede Pipeline einen Bildbereich verarbeiten kann. Teile von Markern, die über einer Bereichsgrenze liegen, werden von beiden beteiligten Pipelines erkannt. Diese Markerfragmente werden vor der Rekonstruktion im Hostrechner wieder zusammengefügt. Mit einer Umsetzung auf drei parallele Pipelines sollen dann Frequenzen im geforderten Bereich von 150–180Hz erreicht werden. Die 3D-Rekonstruktion im Hostrechner erfordert weniger als 1ms, zusammen mit den Kommunikationslatenzen von USB und Fast Ethernet ergibt sich eine Latenzzeit ab Bildvorverarbeitung von weniger als 4ms. Die Gesamtlatenz wird damit im wesentlichen durch Belichtungszeit und Bildauslese- bzw. -verarbeitungszeit dominiert. Bei einer Belichtungszeit von ca. 5ms bleibt die Gesamtlatenz des Systems aktuell unter 25ms. Mit der Parallelisierung der Bildverarbeitungspipelines im FPGA ist eine Reduktion dieses Werts auf ca. 15ms zu erwarten. Die Genauigkeit des Systems wurde in ersten Messungen mit verschiedenen Markertypen zu ca. 0,3–0,35mm (RMSE) bestimmt [Handel 2009, Kap. 6.3]. Hier müssen Versuche mit dem Gesamtsystem zeigen, ob dieser Wert ausreichend für die Roboterregelung ist, oder weitere Maßnahmen zur Verbesserung der Genauigkeit (z.B. verbesserter mechanischer Aufbau, Temperaturkompensation, Einsatz anderer Markertypen) ergriffen werden müssen.

6.4 Lokales Zusatztracking für den Chirurgieroboter ITD

Projektbeschreibung

In einer Studie wurde eine Erweiterung des ITD-Trackings untersucht, mit dem Ziel Auflösung und Genauigkeit der Rotationsbestimmung des Roboters zu verbessern. Dazu soll das im letzten Abschnitt vorgestellte Trackingsetup mit einem Inside-Out-Ansatz kombiniert werden, bei dem eine zusätzliche, lokale Kamera auf dem

Roboter selbst die Marker am Patientenphantom erfasst (siehe Abbildung 6.12). Da der Aufwand für die Entwicklung eines solchen Zusatztrackings recht hoch ist, sollte durch eine vorausgehende Systemsimulation innerhalb der Entwicklungsumgebung Tracklab geklärt werden, inwiefern ein solcher Ansatz zielführend ist.

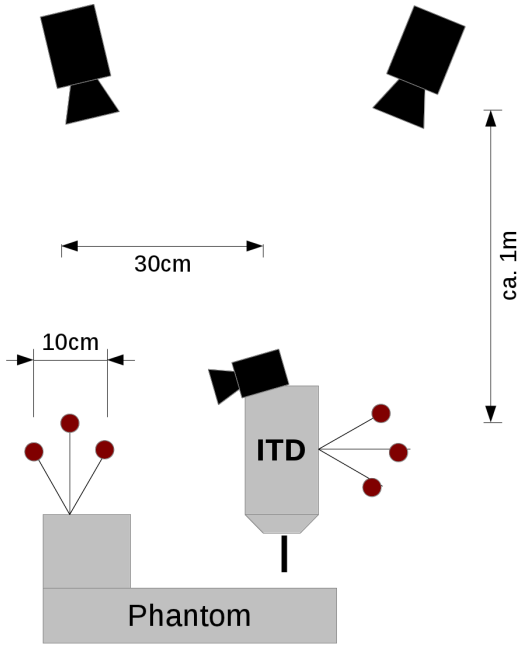
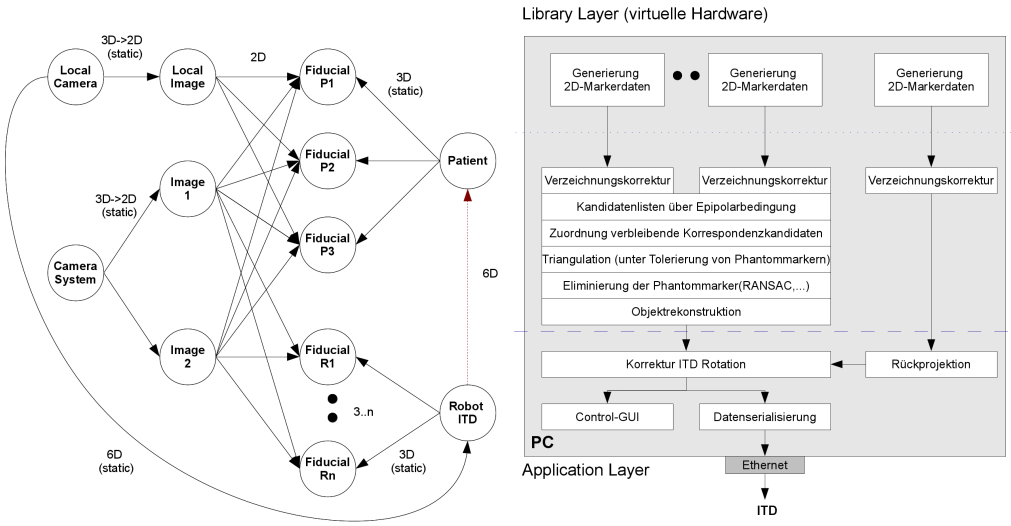


Abbildung 6.12: Lokales ITD-Tracking: Geometrische Konfiguration des Zusatztrackings

Anforderungen an das Tracking

Für eine einfache Realisierung sollen der bisherige ITD-Trackingaufbau sowie Software und Konfiguration möglichst wenig verändert werden. Es sind dieselben Marker am Knochen zu verwenden, die parallel von den Hauptkameras und der Zusatzkamera erfasst werden. Das Zeitverhalten soll sich an den Hauptkameras orientieren, so dass eine einfache Synchronisation gegeben ist. Für die spätere Realisierung des Zusatztrackings ist auf einen kleinen und leichten Aufbau zu achten, der unkompliziert am Roboter integriert werden kann. Abbildung 6.13(a) zeigt den erweiterten Spatial-Relationship-Graphen. Ähnliche Ansätze der Erweiterung eines externen Outside-In-Trackings durch ein lokales Inside-Out-Tracking für eine verbesserte Rotationsgenauigkeit wurden bisher insbesondere im Zusammenhang mit dem Tracking von Kopfbewegungen vorgestellt (z.B. von [Hoff und Golden 1999], [Hoff und Vincent 2000] und [Sielhorst et al. 2004]).



(a) Spatial-Relationship-Graph

(b) Datenflussarchitektur für Simulationssetup

Abbildung 6.13: Lokales ITD-Tracking (Simulation): SRG der Trackingkonfiguration und Adaption der Datenflussarchitektur

Realisierung

Die Nutzenanalyse des lokalen Zusatztrackings wird anhand einer Softwaresimulation durchgeführt. Das ITD-Trackingsetup wird dabei als virtuelles System innerhalb der Tracklab-Umgebung nachmodelliert. Für die Umsetzung kopiert das virtuelle Setup die Marker- und Objektdefinitionen des existierenden ITD-Trackingsetups. Die realen Kameras werden als virtuelle Geräte nachmodelliert, die 2D-Markerdaten liefern. Dabei werden die dreidimensionalen Originalpositionen der Kameras und Marker simuliert und die Kameraprojektion nach Gl. (3.1) nachvollzogen. Die projizierten Markerpositionen werden für die Simulation mit einem (willkürlich gewählten) normalverteilten Fehler mit einer Standardabweichung von 3 Pixeln beaufschlagt und dann als gewöhnliche 2D-Markerdatenlisten in die Datenverarbeitungskette eingespeist. Abbildung 6.13(b) zeigt die angepasste Datenverarbeitungsarchitektur für dieses Setup. Die Bildverarbeitungs- und Markerdetektionsfunktionen aus dem FPGA entfallen, während die Rekonstruktionsroutinen der Haupttrackingkameras unverändert bleiben. Über ein nachgeschaltetes Softwaremodul werden die Informationen aus dem Zusatztracking mit den Daten der Hauptkameras integriert. Es wird über eine Rückprojektion die Rotation der auf dem Roboter montierten Zusatzkamera in Relation zu dem am Phantom befestigen Markerstern bestimmt und damit die über die Hauptkameras gewonnene Rotationsbestimmung des Roboters verbessert.

Ergebnisse und Diskussion

Die Ergebnisse der Simulation zeigen eine relative Verbesserung der Rotationsgenauigkeit um den Faktor 2–3 beim Vergleich der Trackingergebnisse mit und ohne Zusatzkamera, je nach geometrischer Lage von Roboter und Phantom zu den Trackingkameras.[5] Diese sehr einfache Simulation berücksichtigt nur die geometrische Marker- und Kamerakonfiguration und ein sehr einfaches Fehlermodell, reale Effekte wie Vibrationen, Temperatureffekte, Ausreißer in den Messdaten werden nicht nachgestellt. Die Untersuchung zeigt jedoch, dass eine solche Erweiterung des ITD-Trackings sinnvoll ist und an einem realen Funktionsmuster weiter untersucht werden sollte. Hierzu ist eine miniaturisierte Zusatzkamera mit Bildverarbeitungshardware in den Roboteraufbau zu integrieren und mit den Hauptkameras zu synchronisieren. Da ein IR-Ringlicht auf der Zusatzkamera eine Blendung der Hauptkameras verursachen kann, ist der Einsatz von aktiven Markern notwendig. Konfiguration und Datenverarbeitungsmodul zur Integration der Zusatzkamera können dabei direkt aus dem Simulationssystem übernommen werden.

Untersuchungen an diesem realen Systemaufbau sollten die Ergebnisse der Simulation bestätigen. Darüber hinaus müssen Effekte wie das Auftreten von Vibrationen am Roboter und ihre Auswirkungen auf die Bilderfassung und das Trackingergebnis untersucht werden. Langfristig ist die Fragestellung interessant, ob ein solches lokales Tracking mit mehreren miniaturisierten Kamerakomponenten möglicherweise in Zukunft das stationäre Haupttracking nicht nur ergänzen, sondern vollständig ersetzen kann.

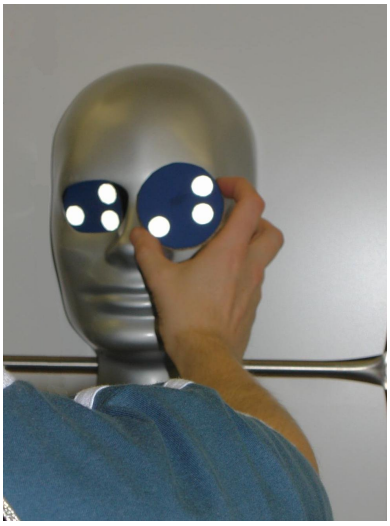
6.5 Kopf- und Handlinsentracking für einen Ophthalmoskopiesimulator

Projektbeschreibung

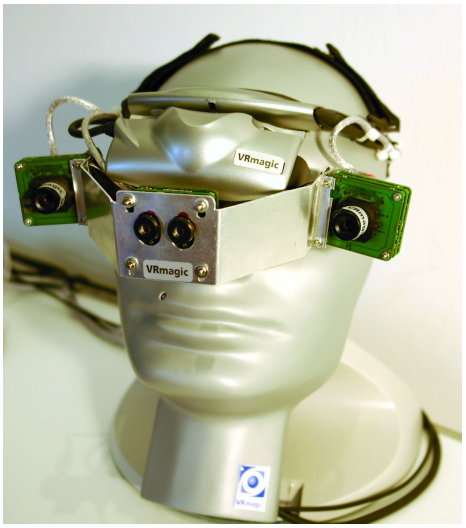
Die indirekte binokulare Ophthalmoskopie ist eine einfache kostengünstige Untersuchung des Augenhintergrundes auf krankhafte Veränderungen. Dabei blickt der Arzt durch eine handgehaltene Linse direkt in das Auge des Patienten. Die notwendigen manuellen Fertigkeiten für die Handhabung dieser Linse vor dem Auge müssen ebenso wie die diagnostischen Fertigkeiten für das sichere Erkennen unterschiedlicher Pathologien trainiert werden. Mit diesem Ziel entwickelt die VRmagic GmbH in Kooperation mit der Arbeitsgruppe VIPA einen Simulator für die indirekte Ophthalmoskopie [Schuppe et al. 2009]. Für eine optimale Immersion wird

[5] Die absoluten Werte sind aufgrund der einfachen Modellierung des Fehlers und der nicht real durchgeführten Kalibrierung der Kameras für die Praxis wenig relevant.

dem Arzt die virtuelle Untersuchungsszene über ein Head-Mounted-Display (HMD) visualisiert. Er trainiert dabei die Positionierung der Handlinse vor einem Kopfphantom sowie die Diagnostik durch Beobachtung eines simulierten Augenhintergrundes (Abbildung 6.14(a)). Die Position des HMDs relativ zum Kopfphantom ist durch ein optisches Trackingsystem zu erfassen. Zusätzlich müssen die Bewegungen der Handlinse getrackt werden. Als Voruntersuchung zur Realisierbarkeit eines solchen Simulators wurde daher die Umsetzung einer angepassten Trackinglösung auf Basis des MOSCOT-Systems evaluiert.



(a) Tracking von Kopfphantom und Handlinse



(b) Funktionsmuster des Head-Mounted-Displays mit Kameras

Abbildung 6.14: Voruntersuchungen für einen Diagnostiksimulator zur indirekten Ophthalmoskopie

Anforderungen an das Tracking

Aufgrund der geforderten hohen Rotationsauflösung für ein stabiles HMD-Bild ist dies ein typisches Trackingszenario für eine Inside-Out-Aufnahmekonfiguration. Wichtige Randbedingungen der Trackingrealisierung folgen unmittelbar aus dem Einsatz für ein Kopftracking, so z.B. ein Aufbau mit leichten miniaturisierten Kameras zur unmittelbaren Montage auf dem HMD. Ähnliche Trackingansätze bei anderen medizinischen Trainingssimulatoren beschreiben z.B. Sielhorst et al. [2004] und Sauer et al. [2002]. Abbildung 6.15(a) illustriert die räumlichen Relationen. Gesucht sind die Relativpositionen von Handlinse und Kopfphantom in Relation zu den Bildebenen des verwendeten Head-Mounted-Displays. Übliche Trackinganwendungen für

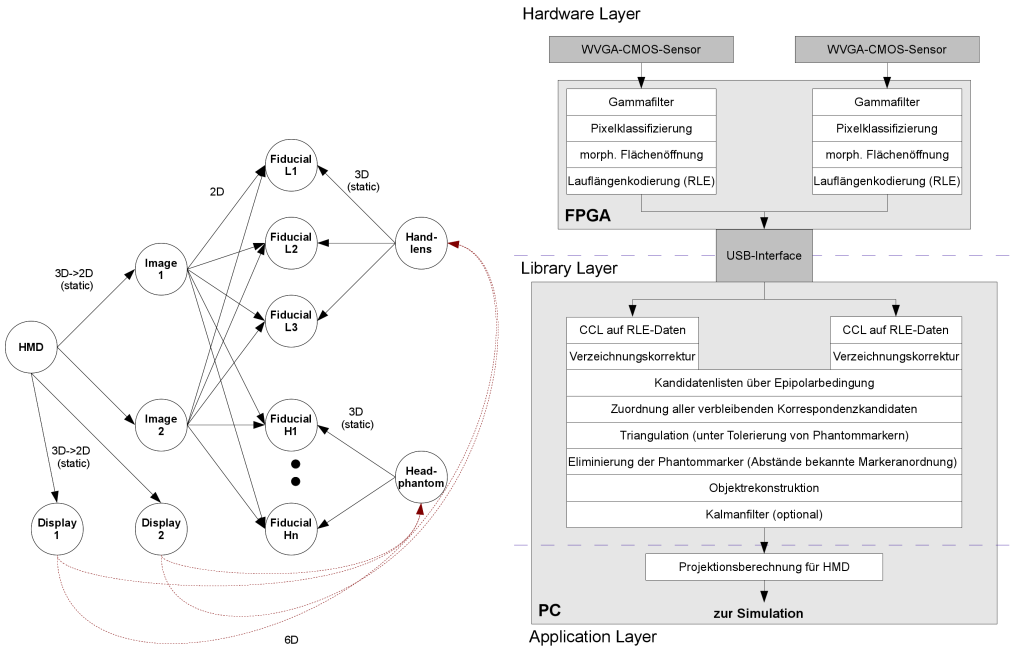
AR-Umgebungen mittels HMDs erfordern eine Latenzobergrenze von ca. 30ms [Held und Durlach 1993]. Da durch das Diagnostikszenario keine schnellen Kopf- und Handbewegungen zu erwarten sind, sind bei dieser Anwendung jedoch auch höhere Latenzzeiten tolerierbar. Ausschlaggebend ist immer die Immersion der resultierenden AR-Simulation. Forderungen nach geringem Ressourcenbedarf für den Hauptrechner des Simulators müssen auch bei diesem Anwendungsfall beachtet werden. Und schließlich ist auch eine Realisierung mit einfachen, kostengünstigen Komponenten anzustreben, da die Ophthalmoskopie eine sehr kostengünstige Untersuchungsmethode darstellt, deren Training keine hohen Investitionskosten erfordern sollte.

Realisierung

Die Implementierung erfolgt in Form zweier am Head-Mounted-Display angebrachter miniaturisierter WVGA-CMOS-Kameras vom Typ Aptina MT9V024 (Abbildung 6.14(b)).[6] Durch den offenen Aufbau des Systems mit Umgebungslichteinfluss ist eine Markersignalisierung auf Infrarotlichtbasis vorteilhaft. Das vor dem Arzt befindliche Kopfphantom und die Handlinse werden mittels retroreflektiver Marker oder aktiver LED-Marker markiert und erfasst, welche sich beim Setup des Trackingsystems nur unwesentlich unterscheiden. Der Kameraaufbau beinhaltet allerdings für retroreflektierende Marker einen IR-Beleuchtungsring. Die finale Auswahl für eine Realisierung des Simulators muss dann nach Kriterien wie Stabilität, Handhabung und finanziellem Aufwand vorgenommen werden. Abbildung 6.15(b) zeigt die Datenverarbeitungsarchitektur für diesen Einsatz. Ein integriertes FPGA-Modul ermöglicht Bildvorverarbeitung in den Kameras. Die lauflängenkodierten vorverarbeiteten Bilder werden sodann zum PC übertragen und erst dort mittels Connected-Component-Labeling die Markerstrukturen extrahiert. Da zum Zeitpunkt der Tests mit dem Funktionsmuster noch keine CCL-Implementierung für das FPGA bereitstand, wurde diese Aufteilung von hard- und softwarebasierter Datenverarbeitung vorgenommen.

Die 3D-Rekonstruktion erfordert eine Vorfilterung der Korrespondenzkandidaten über Epipolargeometrie. Die verbleibenden Markerkorrespondenzen werden rekonstruiert und durch Bestimmung der paarweisen Markerdistanzen vorsortiert. Über ein Starrkörpermatching werden anhand der bekannten Markerkonfigurationen von Handlinse und Kopfphantom etwaige Phantommarker identifiziert und herausgefiltert. Aus den übrigen Markern wird dann die absolute Orientierung der Objektposen geschätzt.

[6] Die beiden anderen Kameras am Funktionsmuster sind für eine Erweiterung auf eine AR-Simulation vorgesehen.



(a) Spatial-Relationship-Graph

(b) Darstellung der realisierten Datenflussarchitektur

Abbildung 6.15: Ophthalmoskopiesimulator: SRG der Trackingkonfiguration und Adaption der Datenflussarchitektur

Ergebnisse und Diskussion

Versuche mit dem Funktionsmuster zeigen die Realisierbarkeit des Kopftrackings für einen HMD-basierten VR-Trainingssimulator auf Basis des MOSCOT-Systems. Mittels angepasster Bilderfassungs- und -verarbeitungsmodule kann die Hardwarerealisierung auf einen für die HMD-Montage passenden Formfaktor gebracht werden. Die Genauigkeit des Systems wurde bisher nicht quantitativ vermessen, zeigt sich in Versuchen jedoch qualitativ als ausreichend für die Immersion bei einem interaktiven Simulator [Schuppe et al. 2009]. Die Gesamtlatenz des Systems ist mit ca. 100ms in dieser ersten Implementierung noch sehr hoch. Hiervon entfallen ca. 30ms auf die sehr aufwendige Raytracing-Darstellung der Linseneffekte und weitere 10–20ms sind durch Bufferingeffekte des HMDs zu erklären. Die Kameraauslesezeit beträgt 33ms (bei 30Hz Kamerafrequenz), so dass ca. 15–25ms als reine Verarbeitungslatenz des Trackings verbleiben. Die Untersuchungen zur Latenzzeit wurden beim Kooperationspartner VRmagic GmbH durchgeführt, wobei nach Aussage von Versuchspersonen die hohe Latenz nicht als hochgradig störend wahrgenommen wird

– wahrscheinlich aufgrund der langsamen Bewegungen bei der Untersuchung. Gleichwohl bleibt zu untersuchen, weshalb dieses spezielle Trackingsetup eine recht hohe Latenz zeigt, da sie bei allen anderen vorgestellten Systemen im Bereich der Kamerauslesezeit zuzüglich 3–10ms Verarbeitungszeit liegt. Neben der Klärung dieses Sachverhalts sollte vor der Systemintegration zum fertigen Produkt auch eine Erweiterung mittels Kalmanfilterung untersucht werden. Sie kann eingesetzt werden, um störendes Rauschen in den Ergebnissen der Kopfrotation zu glätten und durch Prädiktion die Trackinglatenzen teilweise zu maskieren. Eine Kombination des optischen Trackings mit im HMD integrierten Inertialmesssystemen stellt für die Zukunft eine weitere Option zur Verbesserung des Trackingergebnisses dar.

6.6 Weitere Anwendungen

Die präsentierten Anwendungen illustrieren jeweils verschiedene Aspekte des MOSCOT-Systems, so die Vorzüge von Farbmarkern bei EYESI, einfache Ansätze für eine Erweiterung auf nicht-starre Objekte beim EYESI-Deformationstracking, die Anpassung in Richtung eines hochperformanten konventionellen Trackingszenarios bei ITD, die Systemsimulation bei der Erweiterung von ITD für lokales Tracking und schließlich den mobilen Einsatz im Inside-Out-Betrieb beim Ophthalmoskop. Darüber hinaus kam das MOSCOT-System noch in einigen weiteren Anwendungen zum Einsatz, die hier jedoch nur kurz skizziert werden:

Untersuchungen zum Instrumententracking bei endoskopischen und mikrochirurgischen Simulatoren

Im Rahmen von Vorarbeiten für den Aufbau eines endoskopischen Simulators wurde die Adaption des MOSCOT-Systems für ein geeignetes Instrumententracking untersucht, das mit Originalinstrumenten arbeiten kann [Hoffmann 2006]. Im Zentrum der Untersuchungen stand der Vergleich aktiver und passiver Signalisierungstechnik, sowie die Verwendung von Farbcodierung für die eindeutige Unterscheidung und Erfassung von Instrumentposen und -zuständen (Öffnen und Schließen von Zangen und Pinzetten). Verschiedene Markerkonfigurationen auf den Instrumenten wurden untersucht. Das Setup mit aktiven Markern erwies sich als etwas robuster, bei optimierter Beleuchtung ist jedoch auch eine passive Farbmarkervariante realisierbar. Durch die Größe der Originalinstrumente ist eine aktive Illumination sowie die zugehörige Verkabelung unproblematisch (Abbildung 6.16).

Angedacht ist der Einsatz dieses Trackingaufbaus beispielsweise für einen laparoskopischen Simulator zum Training von Gastro-Banding-Eingriffen. Im Rahmen einer

verwandten Anwendung wird dieser Trackingaufbau aktuell in Richtung eines Trainingssimulators für allgemeine mikrochirurgische Fertigkeiten weiterentwickelt.

Abbildung 6.16: Voruntersuchung verschiedener Signalisierungstechniken für das Instrumententracking bei einem endoskopischen Simulator (oben aktive Farb-LEDs, unten passive Farbmarkierungen)

Schnelle Entwicklung eines HMD-Trackings für VR-Anwendungen

Im Rahmen eines Seminars zum Thema virtuelle Realitäten wurde von einer Studentengruppe das Tracking eines Head-Mounted-Displays auf Basis der MOSCOT-Systemarchitektur implementiert. Für einen einfachen Aufbau kamen Analogvideokameras in einem Outside-In-Tracking mit Farbmarkern auf dem HMD zum Einsatz. Aus vorverarbeiteten Kameradaten war die Position des HMD zu bestimmen. Die Studenten implementierten eigene Algorithmen zur Objektrekonstruktion aufsetzend auf der MOSCOT-Trackingbibliothek.

Dieser Aufbau ist dem in Abschnitt 6.5 geschilderten Inside-Out-Ansatz unterlegen, was die Auflösung und Stabilität der Rotationsbestimmung angeht. Hier steht jedoch der einfache mechanische Systemaufbau im Vordergrund sowie die schnelle Umsetzung des Projektes durch die Studenten innerhalb weniger Tage. Diese Anwendung demonstriert somit insbesondere die Eignung des MOSCOT-Konzepts für Rapid-Application-Development und die schnelle Realisierung von einfachen Trackinglösungen.

Optisches Tracking für einen neurochirurgischen Trainingssimulator

Beier [2012] beschreibt den Aufbau eines neurochirurgischen Trainingssimulators, der für das optische Tracking der Benutzerschnittstelle auf das MOSCOT-System aufbaut. Für den Eingriff sind sowohl die Instrumente in einem Kopfphantom bei

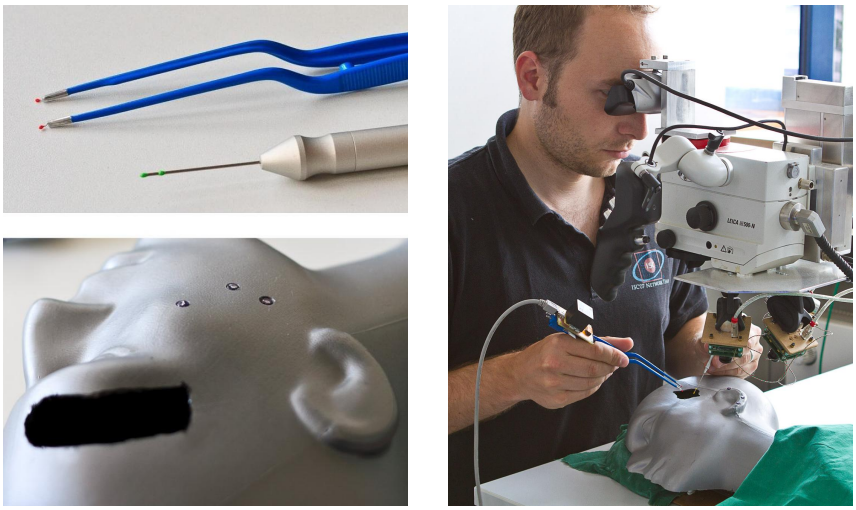


Abbildung 6.17: Tracking bei einem Simulatorprototyp für neurochirurgische Eingriffe. Man sieht die Instrumente mit passiven Farbmarkern, die Maske mit aktiven IR-LED-Markern und die Kameras des externen Trackingsystems für die Maske.

der Durchführung der manuellen chirurgischen Tätigkeiten zu erfassen, als auch die relative Lage zwischen Kopfphantom und dem zur Visualisierung eingesetzten Operationsmikroskop. Das Instrumententracking wird ähnlich der EYESI-Anwendung über ein passives Farbmarkertracking innerhalb des Kopfphantoms realisiert. Für die Rotationserfassung kommt hier ein zusätzliches Inertialmessystem auf den Instrumenten zum Einsatz. Ein zweites, am Operationsmikroskop angebrachtes Trackingsystem mit 4 Infrarotkameras erfasst die relative Lage des Kopfphantoms über aktive LED-Marker. Die Genauigkeit der Abstandsmessung des Systems wurde mit 0,3mm gemessen. Die Trackinglatenz des Systems liegt bei 24ms, wovon 16ms auf die Kameraauslesezeit und 8ms auf der Datenverarbeitung in Software entfallen. Erste Ergebnisse des Gesamtsystems zeigen, dass das komplexe Zusammenspiel mehrerer Trackingsysteme funktioniert und zusammen mit der medizinischen Simulation des Eingriffs dem Nutzer eine gute Immersion in die Trainingssituation vermittelt.

Diese Anwendung von MOSCOT vereint damit die Vorzüge der leichten Applizierbarkeit von Farbmarkern bei einem räumlich beengten, geschlossenen Setup mit der robusteren Erkennung aktiver IR-Marker in einer offenen Trackingumgebung. Des Weiteren illustriert das Instrumententracking erste Anfänge für die im Ausblick dieser Arbeit vorgeschlagene Erweiterung von MOSCOT in Richtung eines Hybridtrackings mit optischen und inertiellen Sensoren.

Kapitel 7

Zusammenfassung, Diskussion und Ausblick

Das Projekt MOSCOT – die Entwicklung eines Baukastensystems für modulares skalierbares optisches Tracking – entstand aufgrund der speziellen Anforderungen in den medizintechnischen Projekten unserer Arbeitsgruppe, die sich mit verfügbaren Trackinglösungen nur unzureichend realisieren lassen. Die Umsetzung ist daher immer auch im Kontext der Ausrichtung auf diese Anwendungen zu sehen. Hieraus resultieren besondere Aspekte des Systems wie der Einsatz einfach zu applizierender Farbmarker, die Auslegung für geringen Ressourcenbedarf und die Unterstützung für unkonventionelle Trackingobjekte wie z.B. miniaturisierte Instrumente oder deformierbare Oberflächen. Darüber hinaus besteht der Hauptbeitrag der Arbeit in der Schaffung eines flexiblen universellen und anpassbaren Gesamtsystems – einer wiederverwendbaren und erweiterbaren Datenflussarchitektur in Hardware und Software mit einer anpassbaren graphischen Entwicklungsumgebung.

Zum Abschluss der Arbeit werden im Folgenden wesentliche Aspekte des vorgestellten Systems dargestellt und diskutiert: die Systemarchitektur, besondere Eigenschaften für die medizintechnischen Anwendungsfälle und die integrierte Entwicklungsumgebung. Die Ergebnisse der Anpassung an die verschiedenen Beispielanwendungen in Projekten der Arbeitsgruppe werden diskutiert. Eine Analyse und Bewertung des Gesamtkonzepts beschließen die Rückschau. Im anschließenden Ausblick werden Perspektiven für die zukünftige Weiterentwicklung aufgezeigt.

7.1 Systemarchitektur

Das MOSCOT-Systemkonzept gewinnt seine Flexibilität, indem es von den speziellen, dem jeweiligen Trackingsetup zugrundeliegenden Verarbeitungsschritten und

Datenstrukturen abstrahiert und sie zu einer generischen modularen Datenflussarchitektur erweitert. Für sämtliche Datenverarbeitungsschritte des Trackingprozesses stellt die Systemarchitektur Hard- und Softwaremodule in einer Datenflusskette bereit. Die Anpassung des Systems an neuartige Trackingszenarien bedingt typischerweise Änderungen bei Eigenschaften wie Kamera- und Markertypen, Objektmarkierung oder Rekonstruktionsverfahren. Durch Erweiterung und Adaption bzw. Neukombination der entsprechenden Module kann die Datenflussarchitektur deutlich einfacher an solche veränderten Anforderungen angepasst werden als ein monolithisch ausgelegtes System. Für Anpassbarkeit und Erweiterbarkeit sorgt außerdem die Einführung einer generischen Datencontainerstruktur, die die Verarbeitungsergebnisse akkumuliert und so auch nachgeschalteten Modulen Zugriff auf die Historie der Daten ermöglicht. Zusammen mit der Kommunikation von Systemkonfiguration und -ereignissen über dieselben Datenflussmechanismen ist die grundlegende Softwarearchitektur damit flexibel erweiterbar und anpassbar, um auch unkonventionelle Trackingszenarien zu unterstützen. Ebenso wird damit die externe Weiterentwicklung in einer Entwicklungsumgebung wie Tracklab ermöglicht.

Ein skalierbares System mit einer Vielzahl von Kameras und Markern muss ausreichend Übertragungsbandbreite und Verarbeitungskapazität für die anfallenden Datenmengen bereitstellen. Da der wesentliche Ressourcenbedarf in den Bildverarbeitungsschritten der Markerdetektion anfällt, ist MOSCOT als verteiltes System, mit optionaler, jeweils in die Kameras integrierbarer Bildverarbeitungshardware in Form von FPGAs ausgelegt. Durch eine hardwareseitige Modularisierung können so je nach Anwendungsfall intelligente Kameras aus geeigneten Kamerasensoren, Bildverarbeitungshardware und Datenübertragungsschnittstelle kombiniert werden. Der zentrale Hostrechner des Simulators wird damit entlastet und führt nur die verbleibenden, weniger ressourcenaufwendigen Rekonstruktionsschritte durch.

Je nach konkreten Anforderungen der Bildverarbeitung und Ressourcenverfügbarkeit werden verschiedene Stufen der Datenreduktion bereits in der Hardware oder erst im Hostrechner in Software vorgenommen. Der MOSCOT-Systembaukasten ist insbesondere flexibel in der Aufteilung der Datenverarbeitungskette auf solche dualen Rechenressourcen in Form dedizierter Hardwaremodule bzw. Softwarebibliotheken für Standard-PC-Betriebssysteme. Uniforme Datenstrukturen und Konfigurationseinstellungen gewährleisten die Austauschbarkeit von Soft- und Hardware-Modulen und erleichtern die Anpassung des Systems an veränderte Gegebenheiten. Die Ausdehnung der bisher nur in der Middleware realisierten Datenflussarchitekturen auf hardwarebasierte Datenverarbeitung manifestiert damit eine zentrale Erweiterung gegenüber bestehenden Systemkonzepten.

Darüberhinaus entsteht so die Möglichkeit, Trackingkonzepte als reine Softwareimplementierung rasch zu entwerfen und zu testen und erst danach in Hardware umzu-

setzen. In eine ähnliche Richtung zielt die Integration von Konzepten zur Simulation von Trackingvorgängen. So ermöglichen simulierte – virtuelle – Trackingsysteme die Aufzeichnung und beliebige Wiedergabe von Trackingdatensätzen verschiedener Datenverarbeitungsstufen zur Fehlersuche und Verifikation. Bei der Neuentwicklung hilft die komplette synthetische Generierung solcher Daten, so dass Untersuchungen zu Marker- oder Kameraanordnung, Einsatz verschiedener Rekonstruktionsverfahren etc. schon vor der tatsächlichen Realisierung vorgenommen werden können.

Die Datenverarbeitungsarchitektur der Hard- und Softwaremodule von MOSCOT ist auf Latenzvermeidung ausgelegt. Die Bildverarbeitungshardware arbeitet dabei direkt auf dem Bilddatenstrom der Kamerasensoren. Damit entfallen Verzögerungen beim Einlesen und Umkopieren der Bilder im Host-PC. Weiterhin ermöglichen FPGAs die massive Parallelisierung und das Pipelining einfacher Bildverarbeitungsoperationen, so dass resultierende Latenzzeiten im wesentlichen nur durch die eingesetzten Kamerasensoren (Belichtungs- und Bildauslesezeit) limitiert sind. Gleichermassen implementiert auch die Software der Trackingbibliothek auf allen Ebenen eine strikte Zero-Copy-Policy, die das zeitaufwendige Umkopieren von Daten vermeidet. Damit kann der gewöhnlich mit einer modularen universellen Systemarchitektur einhergehende Effizienzverlust beschränkt werden, und ein flexibler gleichwohl schlanker Rahmen mit niedrigen Latenzzeiten bereitgestellt werden.

7.2 Besonderheiten der Auslegung für medizinische Simulatoren

Die Auslegung der MOSCOT-Systeme für den Einsatz in medizinischen Simulatoren führt zu einigen Besonderheiten, die sich bei gängigen Trackingsystemen nicht vorfinden.

So unterstützt MOSCOT neben der gebräuchlichen Markerdetektion in Graustufenbildern auch die Verwendung von farbcodierten Markern und Farbkameras. Es werden Datenstrukturen und Algorithmen zur Verfügung gestellt, die eine einfache Konfiguration und effiziente Datenverarbeitung von Farbmarkern unterstützen. Passive Farbmarker erweisen sich in der Anwendung in Simulatoren mit geschlossenem Trackingaufbau als effektive und kostengünstige Signalisierungstechnik. Die kontrollierte Umgebungsbeleuchtung erlaubt eine ressourceneffiziente stabile Echtzeitdetektion der Marker. Punktförmige Farbmarker lassen sich unkompliziert durch Farbauftrag auf verschiedensten Oberflächen realisieren. Dies gilt insbesondere bei Konfigurationen, die für herkömmliche Markierungstechniken schlecht zugänglich sind – z.B. das Tracking von Originalinstrumenten, sehr kleine Trackingvolumina oder

deformierbare Oberflächen. Farbe dient zusätzlich als einfache Codierung zur Identifizierung von Objekten und zur Komplexitätsreduktion des Korrespondenzproblems. Farbmarker sind jedoch auch anfällig für Farbwertverschiebungen unter variablen Beleuchtungsbedingungen (Problem der Farbkonstanz), was zu einer schlechteren Erkennung der relevanten Merkmale und damit zu Genauigkeitseinbußen führen kann. Durch geeignete Parametrisierung der Markerfarbregionen kann dieser Effekt in Simulatorumgebungen kontrolliert werden. Dabei werden farbwertbasierte Farbräume genutzt, die eine robustere Kompensation von Beleuchtungsänderungen erlauben. Die Implementierung der Farbklassifizierung über Lookuptabellen erlaubt eine einfache schwellenbasierte Definition von Farbregionen, aber auch eine gute zukünftige Erweiterbarkeit für komplexer aufgebaute Farbmodelle.

Ebenfalls durch eine spezielle Problemstellung medizinischer Simulatoren veranlasst, entstanden Ansätze für das Tracking einfacher Oberflächendeformationen, da in der Anatomie vielfach deformierbare Objekte (z.B. Gewebe) vorliegen. Hier kann das häufig in optischen Trackingsystemen eingesetzte Starrkörpermatching zur Identifikation falscher Markerkorrespondenzen nicht angewandt werden. Andere Lösungen wie eine sequentielle Signalisierung oder der Einsatz einer größeren Zahl von Kameras wie in Motion-Capture-Systemen scheidet häufig schon aufgrund des Installationsaufwandes und des eingeschränkten Raumangebots aus. Gleichzeitig wird eine Lösung benötigt, die eine robuste Rekonstruktion innerhalb der notwendigen Latenzzeit für interaktive Anwendungen sicherstellt. MOSCOT realisiert verschiedene einfache Ansätze auf Basis von trifokaler Rekonstruktion, relationalem Matching und Nachverfolgung von Markern, die je nach Randbedingungen der jeweiligen Anwendung einzeln oder in Kombination eingesetzt werden können. Die entwickelten Verfahren sind ausreichend schnell für Anwendungen mit langsamen Verformungsbewegungen (ca. 10ms zusätzliche Berechnungszeit für die Rekonstruktion), bei Verwendung von lediglich drei Kameras und passiven Markern, und zeigen eine Robustheit, die hoch genug für die Erfassung einfacher Deformationen in den Benutzerschnittstellen von Simulatoren ist.

Eine weiterer Aspekt der Auslegung für unterschiedlichste medizinische Einsatzbereiche, ist die Austauschbarkeit der Bilderfassungshardware. Je nach Anforderungen des jeweiligen Trackingszenarios können so massgeschneiderte Kamerasensoren in das System integriert werden, mit Anpassungsmöglichkeiten hinsichtlich so unterschiedlicher Eigenschaften wie Baugröße, Gewicht, Auflösung, Geschwindigkeit, Preis und Abbildungsqualität.

Im Unterschied zu gängigen optischen Trackingsystemen ist MOSCOT auch nicht auf den Einsatz eines dedizierten Host-Rechners angewiesen. Vielmehr können die Softwaremodule der Systemarchitektur mit der Anwendung integriert werden und gemeinsam mit dieser im PC des Simulators ablaufen. Die optionale Verlagerung der

rechenintensiven Bildverarbeitungsschritte in die vorgelagerten Hardwareeinheiten erlaubt diese Integration, ohne größere Nachteile durch hohe Ressourcenanforderungen an den Simulationsrechner in Kauf nehmen zu müssen.

7.3 Konfigurations- und Entwicklungsumgebung

Mit der Konfigurations- und Entwicklungsumgebung Tracklab wurde ein zentrales Software-Frontend für Anwender und Entwickler des MOSCOT-Systems geschaffen. Tracklab vermeidet die Segmentierung der Systemsoftware in Einzelanwendungen für Konfiguration, Kalibration und Trackingbetrieb. Das führt zu einem effizienteren Workflow für die Nutzer, da alle für den Betrieb relevanten Interaktionen mit dem System innerhalb einer zentralen Anwendung durchgeführt werden können. Ein pluginbasiertes Framework ermöglicht dabei die Anpassung von Funktionalität und graphischer Benutzeroberfläche an spezialisierte Trackingsysteme und Aufgabenstellungen.

Um Entwicklern innerhalb der Tracklab–Umgebung direkten Zugriff auf relevante Daten und Parameter der tieferen Verarbeitungsstrukturen zu geben, wird die Datenflussarchitektur der Trackingbibliotheken in die Entwicklungsumgebung hinein fortgeführt. Die Weiterverarbeitung rekonstruierter Daten und die Umsetzung komplexerer Rekonstruktionsalgorithmen wird so ermöglicht, bei gleichzeitiger Kapselung der Kernkomponenten. Weitere Faktoren für die schnelle Entwicklung von Tracking-lösungen sind wiederverwendbare Module für die graphische Benutzeroberfläche und die direkte Unterstützung der dualen Hard- und Softwarearchitektur. Simulation, Test und Verifikation von Trackinganwendungen werden auch durch die gleichartige Unterstützung für virtuelle wie reale Systeme und die unmittelbare Visualisierung der Daten unterstützt. Erfahrungen mit studentischen Projekten zeigen, dass bereits nach kurzer Einarbeitungszeit auch komplexere Projekte umsetzbar sind. Da Entwickler und Anwender mit demselben Tool arbeiten, entstehen schnelle Entwicklungszyklen innerhalb derer Neuentwicklungen dem Anwender zur Verfügung stehen. Je nach Einsatzzweck kann die Tracklab-Rahmenanwendung auch so angepasst werden, dass sie dem Nutzer direkt als Frontend einer speziell angepassten Trackinganwendung dient.

Grundsätzlich erweist sich das Konzept eines zentralen Frontends mit pluginbasierten Erweiterungsmöglichkeiten als ein sehr zweckmäßiges Werkzeug für Anwender und auch Entwickler, und erleichtert sowohl die Zugänglichkeit des Systems als auch die Entwicklung neuer Anwendungen.

7.4 Anwendungen

Das vom MOSCOT-Projekt bereitgestellte Baukastensystem wurde in den Projekten der Arbeitsgruppe erfolgreich an verschiedenartige Trackingproblemstellungen angepasst:

Ausgangspunkt und initiale Anwendung des Systems war das Benutzerinterface des opthalmochirurgischen Simulators EYESI. Die Bewegungserfassung von Instrumenten und künstlichem Auge stellt spezielle Anforderungen an das optische Tracking, die mit gängigen Systemen nicht realisierbar sind. Der MOSCOT-Systembaukasten unterstützt eine angepasste Lösung mit Farbmarkern zur Instrumentmarkierung und -codierung auf engstem Raum und Bilddatenverarbeitung in Hardware mit niedrigen Ressourcenanforderungen, die die Integration in den zentralen Simulations-PC zulässt. Genauigkeit und Geschwindigkeit der Bewegungserfassung sind dabei ausreichend, die Immersion der interaktiven Simulatoranwendung zu gewährleisten. Weiterentwicklung und Optimierung des EYESI-Tracking erfordern regelmäßig auch Erweiterungen der MOSCOT-Architektur, der kommerzielle Hintergrund aber immer auch eine Rückwartskompatibilität bei Veränderungen am Gesamtframework.

Für eine mögliche Erweiterung von EYESI entstand im Rahmen dieser Arbeit ein Funktionsprototyp für das Tracking der Oberflächendeformationen eines Auges. Auch hier ist kein System bekannt, welches ein solches Setup innerhalb des geforderten kleinen Trackingvolumens unterstützt. Wichtig ist darüber hinaus die Vereinbarkeit mit dem etablierten EYESI-Instrumententracking. Essentiell für die Umsetzung dieses Deformationstrackings ist die Lösung des Korrespondenzproblems mit ausreichender Stabilität unter den gegebenen zeitlichen Randbedingungen für eine interaktive Anwendung. Passive Farbmarker bieten dabei neben der einfachen Applizierbarkeit eine Komplexitätsbeschränkung des Korrespondenzproblems durch ihre intrinsische Codierung. Verschiedene Verfahren auf Basis von trifokaler Geometrie, relationalem Matching und einer Markernachverfolgung wurden implementiert. Die RAD-Unterstützung von MOSCOT und Tracklab erwies sich als vorteilhaft für die schnelle Implementierung verschiedener Ansätze und ihren direkten Vergleich hinsichtlich zeitlicher Rahmenbedingungen, Stabilität und Genauigkeit der Deformationsrekonstruktion. Aufgrund der Ergebnisse wird eine Kombination der entwickelten Verfahren favorisiert für den Einsatz in zukünftigen Versionen des Simulators. Die vorliegenden Ergebnisse können auf weitere Simulatoren übertragen werden, so z.B. für die Deformationserfassung der Bauchdecke bei simulierten endoskopischen Eingriffen im Bauchraum.

Eine weitere Anwendung erfolgte im Rahmen des Projektes ITD, der Positionsregelung eines handgehaltenen chirurgischen Roboters. Ziel des Projektes ist die schnelle Nachregelung der Werkzeugpositionierung durch den Roboter entlang einer präope-

rativ vorgegebenen Trajektorie und damit eine Korrektur möglicher Positionierungsfehler des Chirurgen. Zu diesem Zweck musste das MOSCOT-System mittels schneller hochauflösender Kamerasensoren hinsichtlich erhöhter Geschwindigkeit und Genauigkeit weiterentwickelt werden. Von der Trackinggeometrie her ein konventionelles Setup (Outside-In-Anordnung bei mittlerem Trackingvolumen und Objektgröße) stellt diese Anwendung erweiterte Anforderungen an Skalierbarkeit, Verschattungsrobustheit und Latenzverhalten der MOSCOT-Architektur. Eine erste Implementierung auf Basis eines einfachen Farbmarkertrackings zeigte die prinzipielle Funktion des Gesamtkonzepts aus Tracking und aktiv geregeltem Roboter, erreichte jedoch erwartungsgemäß noch nicht die erforderliche Genauigkeit und Geschwindigkeit. Eine Eigenentwicklung intelligenter Kameras mit schnellem hochauflösenden Kamerasensor und erweiterter Bildverarbeitungshardware bietet mittlerweile die Voraussetzung für die geforderten geringen Latenzzeiten (15ms inkl. Belichtungszeit) und hohen Bildwiederholraten(150Hz). Durch die Markersignalisierung auf Infrarotbasis zeigt sich die Genauigkeit ebenfalls stark verbessert (ca. 0,35mm), erreicht jedoch noch nicht ganz die geforderten Werte (0,25mm). Ihre Optimierung bleibt damit Aufgabe weiterer Entwicklungsarbeiten bei Kamerakalibration und -montage. Das ITD-Tracking zeigt, dass MOSCOT durchaus auch die Anwendungsgebiete spezialisierter konventioneller Trackingsysteme abdecken kann. Zum Erreichen sehr hoher Genauigkeiten (besser als ca. 1/1000 der Dimension des Trackingvolumens) muss allerdings wie bei kommerziellen Systemen substantieller Aufwand in Entwicklungs- und Optimierungszyklen von mechanischem Aufbau und Systemkalibration investiert werden.

Als alternativer Ansatz zur Verbesserung der Rotationsauflösung des ITD-Trackings wurde die Ergänzung mit einer direkt auf dem Roboter montierten Zusatzkamera als Inside-Out-Konfiguration untersucht. Da im Vorfeld nicht absehbar war, ob das Resultat den Aufwand von Entwicklung und Integration einer solchen miniaturisierten Kamerahardware rechtfertigt, wurde ein solches Setup vorab als virtuelles Trackingsystem in der Tracklab-Umgebung simuliert. Das Konzept virtueller Trackingsysteme erlaubt die Nachbildung des realen ITD-Trackingsetups unter identischen Randbedingungen von Kamerapositionierung, -ausrichtung, -auflösung etc. In dieses Setup wird eine zusätzliche virtuelle Kamera integriert, so dass die Rotationsrekonstruktion durch deren zusätzlichen Informationen ergänzt werden kann. Die MOSCOT-Architektur unterstützt die Simulation und Umsetzung solcher heterogener Systemsetups mit geringem Aufwand. Die Ergebnisse der Simulation zeigen, dass sich mit diesem Ansatz die Genauigkeit der Rotationsbestimmung deutlich erhöhen lässt (in der simulierten Konfiguration um einen Faktor von ca. 2–3). Sie müssen nun bei Tests mit einer realen Zusatzkamera auf dem Operationsroboter bestätigt werden.

Als weitere Adaption an ein medizinisches Simulatorprojekt wurde bei der VRmagic GmbH auf Basis des MOSCOT-Systems ein Funktionsmuster für das optische

Tracking eines Ophthalmoskopie-Simulators entwickelt. Diese Anwendung erfordert das Tracking der Kopfposition für ein Head-Mounted-Display sowie einer handgehaltenen Linse vor einem Patientenphantom. Da für ein Head-Mounted-Display die Rotationsauflösung eine zentrale Rolle spielt, bietet sich für die Umsetzung ein Inside-Out-Szenario an, bei dem die Trackingkameras in das Head-Mounted-Display integriert sind. Die Herausforderungen für dieses Setup liegen damit zum einen in der Konstruktion einer spezialisierten kleinen leichten Stereokamerahardware, zum anderen sind für den Simulatoreinsatz wiederum geringe Ressourcenanforderungen im Simulations-PC bedeutsam. Ergebnisse mit dem Funktionsmuster zeigten die Realisierbarkeit dieser Anwendung mit dem MOSCOT-System. Die noch recht hohen Latenzzeiten (ca. 25ms reine Verarbeitungslatenz) müssen untersucht und vermindert werden. Tests mit Anwendern zeigen aber interessanterweise eine weniger große Beeinträchtigung der Immersion als anhand der Messwerte zu erwarten wäre – wahrscheinlich aufgrund der langsamen abzubildenen Bewegungen bei der Untersuchung.

Neben diesen ausführlich beschriebenen Projekten kam das MOSCOT-System erfolgreich bei verschiedenen weiteren Trackingaufgabenstellungen in der Arbeitsgruppe zum Einsatz, die jeweils einzelne Aspekte des Konzepts verdeutlichen. So konnten beim Einsatz in einem Simulatorprototypen für endoskopische, mikrochirurigische Eingriffe die Vorzüge von Farbmarkern für einfache Markierung und Identifikation verschiedenartiger Instrumente ausgenutzt werden. Ein studentisches Seminar zum Thema Virtuelle Realität zeigte, dass mit dem Gesamtpaket aus der Entwicklungsumgebung Tracklab und den Hard- und Sofwarekomponenten der MOSCOT-Architektur bereits nach kurzer Entwicklungszeit problembezogene angepasste Trackinglösungen realisierbar sind. Und an einem weiteren Prototypen für neurochirurgische Eingriffe konnte die Kombination und Erweiterbarkeit von Farbmarker- und Infrarottracking mit einem Inertialmesssystem demonstriert werden.

7.5 Analyse und Schlussbetrachtung

Das im Rahmen dieser Arbeit entwickelte Gesamtsystem bildet eine ausgereifte Basis, auf die zukünftige Trackinganwendungen in der VIPA-Gruppe und bei der VRmagic GmbH aufbauen können. Das System ist universell einsetzbar und erweiterbar; durch das offene modulare Systemkonzept ist es an Anwendungen mit unterschiedlichsten Anforderungen anpassbar. Die Abstraktion und Generalisierung der Datenverarbeitungsschritte vermeidet eine Festlegung hinsichtlich Art und Anzahl von Markern und Kameras bzw. Signalisierungs- und Rekonstruktionsmethoden. Die Skalierbarkeit ermöglicht Redundanz und damit erhöhte Robustheit bei schwierigeren Einsatzbedingungen. Gleichzeitig ist die Architektur schlank genug gehalten, um Anforderungen nach niedrigen Latenzzeiten zu unterstützen. Die Austauschbarkeit

von Hard- und Softwaremodulen, die Transparenz von Systemsetup und Datenverarbeitung gegenüber der Implementierung zusätzlicher Verarbeitungsschritte sowie die enge Integration mit einer dedizierten Entwicklungsumgebung erlauben gute Zugänglichkeit für Entwicklung, Test und Verifikation.

Die Bandbreite der realisierten Anwendungen zeigt, dass das Ziel einer universellen Auslegung des MOSCOT-Systems in großen Teilen erreicht werden konnte. In einfachen Fällen kann eine Trackingaufgabe unmittelbar unter Verwendung von Standardkameras zusammen mit vorhandenen Softwarekomponenten für Systemsetup und Trackingbetrieb gelöst werden. Am anderen Ende des Anwendungsspektrums unterstützt MOSCOT auch unkonventionelle Trackingszenarien, welche speziell angepasste Marker- und Kameratypen und dedizierte Detektions- und Rekonstruktionsverfahren erfordern (z.B. das Augendeformationstracking für EYESI). Solche Anpassungen sind möglich ohne grundlegende Änderungen am Framework durch bloße Erweiterungen der Datenverarbeitungskette - meist sogar ausserhalb der eigentlichen Trackingbibliothek. Der Zeitbedarf für die Neuentwicklung spezialisierter Trackinglösungen kann durch die gute Adaptabilität sowie die beschriebene enge Integration mit der Entwicklungsumgebung Tracklab deutlich begrenzt werden. Mit Ausnahme des Basistrackings für den Roboter ITD wären die präsentierten Lösungen mit verfügbaren Systemen nicht oder nur unter Einschränkungen und mit hohem Anpassungsaufwand realisierbar gewesen.

Der Einsatz von FPGAs in der Bildverarbeitungshardware führt zu niedrigen Latenzzeiten, hohen Datenbandbreiten und großer Flexibilität beim Anschluss der Sensor-ICs. Andererseits verlängern sich die Entwicklungszeiten, da FPGAs aufwendiger zu programmieren sind. Zumindest bei Kamerasensoren mit geringer Bandbreite, die nicht auf FPGA-Eigenschaften wie parallele Datenverarbeitung und Pipelining angewiesen sind, können Entwicklungsziele schneller mit DSPs erreicht werden. Für zukünftige Entwicklungen können DSP-basierte Bildverarbeitungskomponenten das Anwendungsspektrum von MOSCOT damit zusätzlich erweitern. Auch das vermehrte Aufkommen von Mehrkern-Prozessoren und universell einsetzbaren Grafikprozessoren (GPUs) seit Beginn dieser Arbeit bietet neue Optionen bei der Aufteilung der Verarbeitungsressourcen, da die verfügbaren Datenverarbeitungsbandbreiten im Host-PC deutlich angestiegen sind. Ein Flaschenhals der Datenübermittlung bleibt jedoch die Datenanbindung hochperformanter Systeme an den PC selbst bei Einsatz aktuellster Schnittstellentechnik (vgl. Abschnitt 4.1.3), so dass auf die Datenreduktion durch intelligente Kameras weiterhin nicht verzichtet werden kann.

Ganz allgemein kann die flexible Hard- und Softwareauslegung von MOSCOT solchen neuen Gegebenheiten jedoch Rechnung tragen, indem sie beispielsweise die teilweise Verlagerung von Bildverarbeitungsoperationen aus dedizierter Hardware

(FPGA) in moderne GPU-Einheiten unterstützt.[1] Genauso können komplexere Verarbeitungsoperationen aus dem Host-PC in die Kameras verlagert werden, falls dort durch den Einsatz von FPGAs mit integrierten DSP-Einheiten oder CPU-Kernen in herkömmlichen Programmiersprachen zu programmierende Verarbeitungsressourcen bereitstehen.

Bei vielen Simulatoranwendungen mit geschlossener Benutzerschnittstelle stellen sich Farbmarker als adäquate Methode der Objektsignalisierung heraus. Ihre immanent geringere Genauigkeit erweist sich durch die nicht auf absolute Genauigkeit angewiesenen Anwendungen als unkritisch und die Vorzüge wie Codierung, Applizierbarkeit und ressourcenschonende Detektion überwiegen. Bei offenen Trackingsetups und Anwendungen mit höheren Genauigkeitsanforderungen sind infrarotbasierte Signalisierungstechniken vorzuziehen. Aktive und retroreflektierende passive Marker unterscheiden sich dabei für die geplanten MOSCOT-Anwendungen nur wenig hinsichtlich Genauigkeit und Stabilität. Entscheidend sind Randbedingungen des mechanischen Aufbaus sowie der Kamerapositionierung. Die generische Unterstützung für farbige und monochrome Markersignalisierung in der Systemarchitektur gewährt deren einfache Austauschbarkeit.

Die mit MOSCOT realisierten Trackinganwendungen erreichen Latenzzeiten im Bereich der jeweiligen Kameraauslesezeit zuzüglich 3–10ms, je nach Komplexität der Datenverarbeitung auf der Softwareseite. Die Latenzen der hardwarebasierten Datenverarbeitung selbst sind vernachlässigbar. Damit eignet sich das System gut für den angestrebten Einsatz in interaktiven VR-Simulatoren. Die absolute Genauigkeit ist für Anwendungen mit sehr hohen Anforderung (besser als etwa 1/1000 der Dimension des Trackingvolumens) noch nicht ausreichend. Dieser Aspekt ist jedoch unabhängig von grundsätzlichen Designkonzepten der Systemarchitektur. Vielmehr muss hier ähnlich wie bei kommerziellen Systemen für das Erzielen hoher Genauigkeiten ein erhöhter Optimierungsaufwand bei einzelnen Aspekten des Systems wie mechanischem Aufbau, Temperaturstabilität, Kamerakalibration etc. betrieben werden. Durch den modularen Systemaufbau können bei der weiteren Entwicklung einzelne, die Genauigkeit stark beeinflussende Komponenten oder Verfahren wie Kamerasensoren oder Kalibrationsverfahren ausgetauscht bzw. optimiert werden. Für die Gewährleistung der Immersion in den medizinischen Simulatoren der Arbeitsgruppe erweist sich die Genauigkeit in jedem Falle als ausreichend. Das System soll auch durchaus nicht mit Spezialtrackingsystemen (beispielsweise für Karosseriemessung in der Autoindustrie, oder Motion-Capture in der Filmindustrie) in deren jeweiligen Einsatzgebieten konkurrieren. MOSCOT als Systembaukasten soll vielmehr eine breite Einsetzbarkeit

[1] Handel [2009] untersucht die Möglichkeiten von GPU-basierter Bildverarbeitung für MOSCOT. Die Ergebnisse legen nahe, dass GPU eine gute Ergänzung für komplexere Bildverarbeitungsoperationen im Host-PC sein können, jedoch kein Ersatz für die Datenreduktion durch spezialisierte Hardware direkt in den Kameras.

und Anpassbarkeit für unterschiedlichste Trackingaufgaben auch außerhalb der Medizintechnik aufweisen - insbesondere bei unkonventionellen Setups, die mit gängigen Systemen nicht oder nur stark eingeschränkt realisierbar sind.

7.6 Ausblick

Die weitere Entwicklung des MOSCOT-Systems konzentriert sich auf Optimierungen des bestehenden Systems und die Erweiterung seines Anwendungsspektrums.

Ein Optimierungsschwerpunkt liegt auf der Erhöhung der Genauigkeit, eine Notwendigkeit insbesondere für die höheren Anforderungen bei Anwendungen im Operationsumfeld und im Bereich Augmented Reality. Neben konstruktiven Maßnahmen bei Gehäusen und Objektiven betrifft dies vor allem den Kalibrierungsprozess. So können Einflüsse wie Temperaturänderungen in der Aufwärmphase vermessen und während des Betriebs ausgeglichen werden. Die Ergebnisse der parallelen Dissertation von Handel [2009] müssen hierzu in die MOSCOT-Architektur eingegliedert werden. Die Möglichkeit einer dynamischen Rekalibrierung über Bündelblockausgleichung und ihre Auswirkung auf Genauigkeit und Echtzeitfähigkeit sollten ebenfalls untersucht werden.

Eine weitere Fortentwicklung des Systems betrifft die Unterstützung von Trackingszenarien mit einer großen Zahl an Objekten und Markern. Während Architektur und Datenverarbeitungsressourcen bereits skalierbar ausgelegt sind, müssen bei Markerdetektion und Objektrekonstruktion erweiterte Verfahren implementiert werden. Hierzu gehören Bildverarbeitungsverfahren zur Trennung sich gegenseitig verdeckender Marker und Methoden zur Lösung der 3D-Posenschätzung für große Punktwolken von rekonstruierten Markern.

Zur Erweiterung der Bandbreite möglicher Trackinganwendungen ist die Unterstützung zusätzlicher Markertypen in den hardwarebasierten Bildverarbeitungsmodulen wünschenswert. So kann der Einsatz komplexerer kantenbasierter Markertypen Vorteile bei bestimmten Anwendungsbereichen bieten. Vielversprechend erscheint hierfür auch die Kombination einfacher Segmentierungsmethoden im FPGA mit mathematisch aufwendigerer Merkmalsbestimmung in GPUs. Eine Erweiterung der Bildverarbeitungshardware auf digitalen Signalprozessoren ist ebenfalls sinnvoll, da diese die Implementierung einer Echtzeitdetektion von kantenbasierten Markern erleichtern. Die Kombination von FPGA- und DSP-basierter Bildverarbeitung werden bei der VRmagic GmbH und in weiteren Projekten der Arbeitsgruppe untersucht.

Zur Erschließung neuer Einsatzbereiche müssen auch zusätzliche Hardware-Plattformen für die Ausführung der Marker- und Objektrekonstruktion bereitgestellt werden.

So ist beispielsweise die Umsetzung auf ein eingebettetes Echtzeitsystem Voraussetzung für den Einsatz eines medizinischen Robotersystems im Operationssaal. Eine mögliche Entwicklungsrichtung ist auch die direkte Implementierung der Rekonstruktionsalgorithmen in intelligenten Kamerasystemen mit integrierten Prozessoreinheiten, so dass für diesen Anwendungsfall auf einen separaten Hostrechner verzichtet werden kann.

Den Ausgangspunkt für alle Weiterentwicklungen des MOSCOT-Systems stellen jedoch immer die Anwendungen dar. So müssen einige der in Kapitel 6 erläuterten Projekte zusammen mit den beteiligten Industriepartnern zur Produktreife weiterentwickelt werden. Neue Trainingsszenarien, die anders als endoskopische Operationen ohne genau definierte Zugangspunkte auskommen müssen, erfordern eine freie Beweglichkeit der Instrumente im Raum und führen damit zusätzliche Randbedingungen für das Tracking ein. Simulatoren für neurochirurgische Eingriffe und allgemeines mikrochirurgisches Training werden als erste solche Anwendungen entwickelt.

Über das optische Tracking hinaus zielt die Fortentwicklung des MOSCOT-Systems auf eine generalisierte Bibliothek für Eingabegeräte. Die Basis der bestehenden Datenflussarchitektur kann zur Integration der Sensordaten von weiteren Messwertaufnehmern erweitert werden. Von besonderem Stellenwert ist die Sensorfusion, da absolute und relative Genauigkeiten verschiedener Sensortypen sowie ihre Messlatenzen sehr unterschiedlich sind. Vielversprechend erscheint hier insbesondere eine Kombination von Intertialmesssystemen mit optischem Tracking. Die unmittelbare Erfassung der translatorischen und rotatorischen Beschleunigungen durch die Inertialsensoren und der Abgleich ihrer Relativdrift mittels einer absoluten optischen Positionsbestimmung verbindet elegant die Vorteile beider Messprinzipien und erscheint derzeit als beste Lösung für eine schnelle, direkte und präzise Positionsbestimmung im Raum. Die vielversprechenden Aussichten einer solchen Hybridlösung werden auch dadurch unterstrichen, dass sie genau jenes Zusammenspiel von Messprinzipien darstellt, welches die Evolution in der Kombination von Gleichgewichts- und Gesichtssinn zur Lösung des Problems der Lagekontrolle hervorgebracht hat.

Anhang A

Mathematische Grundlagen

In diesem Anhang werden die Grundlagen für die mathematische Beschreibung des Abbildungs- und Rekonstruktionsprozesses optischer Trackingsysteme zusammengefasst. Für eine detaillierte Beschreibung sei auf die Ausführungen in [Faugeras 1993] und [Hartley und Zisserman 2000] verwiesen.

A.1 Projektive Geometrie

Die projektive Geometrie ist eine Erweiterung der altbekannten euklidischen Geometrie. Sie ermöglicht die Darstellung von zusätzlichen Transformationen (z.B. Skalierung, Scherung), die über die euklidischen Isometrien Rotation und Translation hinausgehen. Insbesondere stellt sie ein Modell für die perspektivische Projektion bereit, die beim Abbildungsprozess von Kameras auftritt. Daraus erklärt sich die Bedeutung der projektiven Geometrie für viele Bereiche der Computer Vision, so z.B. bei Kamerakalibration, Zentralprojektion und Stereorekonstruktion.

A.2 Homogene Koordinaten

Eine fundamentale Rolle bei der algebraischen Darstellung der projektiven Geometrie spielt das Konzept homogener Koordinaten. Dabei wird jedem Punkt des euklidischen Raums zu den kartesischen Koordinaten eine zusätzliche Koordinate hinzugefügt. Da die absolute Skalierung nicht interessiert, kann auf diese letzte Koordinate normiert werden, d.h. alle Punkte $(wx, wy, wz, w)^T$ für $w \neq 0$ entsprechen dem Punkt $(x, y, z, 1)^T$. Der Fall $w = 0$ führt bei der Normierung zu keiner gültigen Lösung im euklidischen Raum. Er beschreibt vielmehr den Punkt in Richtung $(x, y, z)^T$ im Unendlichen. Damit unterstützt die projektive Geometrie die bei der perspektivischen Abbildung auftretenden Fluchtpunkte als Schnittpunkte von parallelen Geraden im Unendlichen. Neben der Darstellung dieser speziellen Punkte liegt

die Bedeutung homogener Koordinaten in Computer Vision und Robotik darin, dass durch den Übergang in einen höher-dimensionalen Raum inhomogene Abbildungsgleichungen homogenisiert werden, d.h. dass auch Translationen und die perspektische Projektion als lineare Abbildung dargestellt und damit mittels einer einzigen Matrizenmultiplikation berechnet werden können.

So lässt sich das folgende Beispiel einer Rotation mit der Rotationsmatrix $\mathbf{R}$ mit anschließender Translation um den Vektor $\mathbf{t}$

$$\mathbf{p}' = \mathbf{R}\mathbf{p} + \mathbf{t} \tag{A.1}$$

bei Übergang zu homogenen Koordinaten durch eine einzige Multiplikation mit der Matrix $\mathbf{H}$ ausdrücken:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{A.2}$$

mit

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{A.3}$$

Mit ausgeschriebenen homogenen Koordinaten ergibt sich:[1]

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11}x + r_{12}y + t_1 \\ r21x + r_{22}y + t_2 \\ 1 \end{pmatrix} \tag{A.4}$$

A.3 Kameramodell und Abbildungsgleichungen

Der Abbildungsprozess einer Kamera stellt eine Projektion aus dem 3-dimensionalen projektiven Raum (der Welt) auf eine 2-dimensionale projektive Ebene (die Bildebene) dar.

Die bekannten Abbildungsgleichungen für eine Zentralprojektion mit der Brennweite f folgen aus dem Strahlensatz (vgl. Abbildung A.1):

$$\frac{y}{f} = \frac{Y}{Z} \Rightarrow y = f\frac{Y}{Z}, \text{ analog } x = f\frac{X}{Z}. \tag{A.5}$$

Die perspektische Abbildung des Weltpunktes $(X, Y, Z)^T$ auf den Punkt $(x, y)^T$ der Bildebene kann damit wie folgt geschrieben werden:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix}. \tag{A.6}$$

[1] Das Beispiel lässt sich analog vom zweidimensionalen auf den dreidimensionalen Anwendungsfall erweitern.

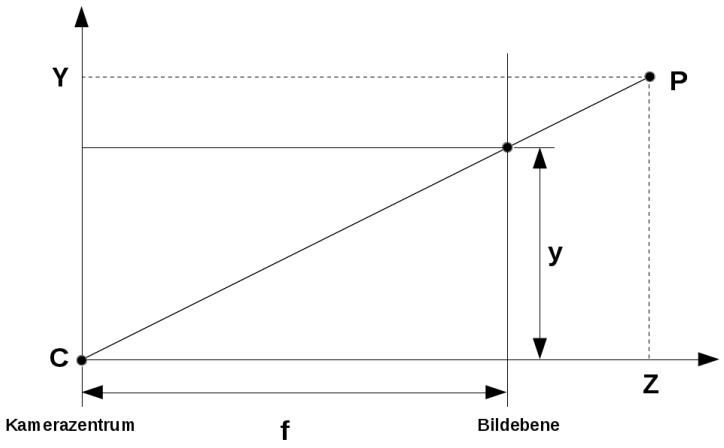


Abbildung A.1: Zentralprojektion mit Brennweite f

Beim Übergang zu homogenen Koordinaten lassen sich die Projektionsgleichungen in linearer Form darstellen:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{A.7}$$

Die inhomogenen Koordinaten ergeben sich damit wieder bei Normierung auf $\tilde{w} = 1$ zu

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix}. \tag{A.8}$$

In der praktischen Anwendung erweist es sich als sinnvoll, bei der Kameraprojektion nach Gl. (A.7) mit dem durch die Bildpixel des Kamerasensors vorgegebenen Koordinatensystem zu arbeiten. Da Pixelkoordinaten gewöhnlich von der linken oberen Ecke des Bildes aus gemessen werden, verschiebt sich der Hauptpunkt (der Durchstoßpunkt der optischen Achse mit der Bildebene) aus dem Ursprung des Bildkoordinatensystems zum Punkt $(p_x, p_y)^T$. Zusätzlich muss im Falle nicht-quadratischer Abmessungen der Kamerasensorpixel die Brennweite in x- und y-Achsenrichtung getrennt berücksichtigt werden: $f_x = m_x f$ bzw. $f_y = m_y f$, wobei m_x und m_y die Anzahl der Pixel pro Längeneinheit in jeder Koordinatenrichtung angeben. Schließlich wird noch eine etwaige Verkippung der beiden Koordinatenachsen gegeneinander, die sogenannte Bildschiefe s, im erweiterten Modell berücksichtigt, wobei dieser Wert bei aktuellen industriell gefertigten Kamerasensoren praktisch bei $s = 0$ (90°-Winkel der

Achsen des Bildkoordinatensystems) liegt. Unter Berücksichtigung aller zusätzlichen Parameter erweitert sich Gl. (A.7) damit auf

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{bmatrix} f_x & s & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{A.9}$$

Diese 3x3-Matrix beinhaltet alle internen Parameter des Lochkameramodells und wird als Kameramatrix $\mathbf{K}$ bezeichnet:

$$\mathbf{K} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{A.10}$$

Für eine vollständige Darstellung der Abbildung einer Lochkamera muss noch ihre äußere Orientierung, d.h. ihre Position und Orientierung innerhalb des Weltkoordinatensystems, berücksichtigt werden. Hierzu werden die externen Kameraparameter, eine Rotationsmatrix $\mathbf{R}$ und ein Translationsvektor $\mathbf{t}$, eingeführt. Sie können in Gl. (A.9) integriert werden und liefern dann die vollständige Projektionsgleichung des Lochkameramodells

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{bmatrix} f_x & s & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} [\mathbf{R}|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{A.11}$$

bzw. in Matrix-Vektor-Schreibweise als

$$\mathbf{x} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X} = \mathbf{P}\mathbf{X}. \tag{A.12}$$

Die resultierende homogene 3x4-Matrix $\mathbf{P}$ beinhaltet sämtliche Parameter des Lochkameramodells und wird als Projektionsmatrix $\mathbf{P}$ bezeichnet. Sie ist bis auf einen Skalierungsfaktor bestimmt und besitzt damit 11 Freiheitsgrade: 3 aus der Rotation, 3 aus der Translation und 5 aus den internen Parametern der Kameramatrix $\mathbf{K}$.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{A.13}$$

A.4 Linsenmodellierung

Als rein lineare Abbildung berücksichtigt die Projektion mit dem Lochkameramodell keine nicht-linearen Verzeichungseffekte, die jedoch beim Einsatz realer Linsen entstehen. Für erhöhte Genauigkeitsanforderungen müssen zumindest radiale

(tonnen/kissenförmige), gegebenenfalls auch tangentiale Verzeichnungen der Linsenabbildung modelliert werden. Dies geschieht über eine Verzerrung der x- und y-Koordinaten in der Bildebene im Anschluss an die eigentliche (als ideal angenommene) perspektivische Projektion.

$$x_d = x_u + \Delta x_{rad}(x_u, y_u) + \Delta x_{tan}(x_u, y_u) \tag{A.14}$$
$$y_d = y_u + \Delta y_{rad}(x_u, y_u) + \Delta y_{tan}(x_u, y_u) \tag{A.15}$$

$(x_d, y_d)^T$ bezeichnen dabei die verzeichneten Bildkoordinaten, $(x_u, y_u)^T$ die ursprünglichen, unverzeichneten.

Die Verzerrungsberechnung wird dabei auf den Bildhauptpunkt $(p_x, p_y)^T$ als Ursprung bezogen und arbeitet mit normalisierten idealen Bildkoordinaten $(\bar{x}_u, \bar{y}_u)^T$:

$$\bar{x}_u = (x_u - p_x)/f_x \tag{A.16}$$
$$\bar{y}_u = (y_u - p_y)/f_y \tag{A.17}$$

Die radiale Verzerrung kann dann nach folgender Gleichung durch die Koeffizienten k_1 und k_2 modelliert werden[2]:

$$\Delta x_{rad}(x_u, y_u) = (x_u - p_x)[k_1(\bar{x}_u^2 + \bar{y}_u^2) + k_2(\bar{x}_u^2 + \bar{y}_u^2)^2] \tag{A.18}$$
$$\Delta y_{rad}(x_u, y_u) = (y_u - p_y)[k_1(\bar{x}_u^2 + \bar{y}_u^2) + k_2(\bar{x}_u^2 + \bar{y}_u^2)^2] \tag{A.19}$$

Analog kann die tangentiale Verzerrung durch die Koeffizienten t_1 und t_2 modelliert werden:

$$\Delta x_{tan}(x_u, y_u) = 2t_1\bar{x}_u\bar{y}_u + t_2((\bar{x}_u^2 + \bar{y}_u^2) + 2\bar{x}_u^2) \tag{A.20}$$
$$\Delta y_{tan}(x_u, y_u) = t_1((\bar{x}_u^2 + \bar{y}_u^2) + 2\bar{y}_u^2) + 2t_2\bar{x}_u\bar{y}_u \tag{A.21}$$

Die Verzeichnungskoeffizienten werden zusammen mit den internen Kameraparametern während der Kamerakalibrierung bestimmt. Bei der Kalibrierung realer Linsenkameras zeigt sich, dass die Effekte radialer Verzerrung dominant sind und die tangentialen Koeffizienten meist nahe bei Null liegen. Mit den bekannten Verzeichnungskoeffizenten können Kamerabilder vor der 3D-Rekonstruktion entzerrt werden, so dass für die weiteren Rekonstruktionsschritte wieder mit dem linearen Lochkameramodell gearbeitet werden kann.

[2]Die Reihenentwicklung kann durch weitere Koeffizenten erweitert werden, dies ist jedoch bei optischen Trackinganwendungen gewöhnlich nicht notwendig.

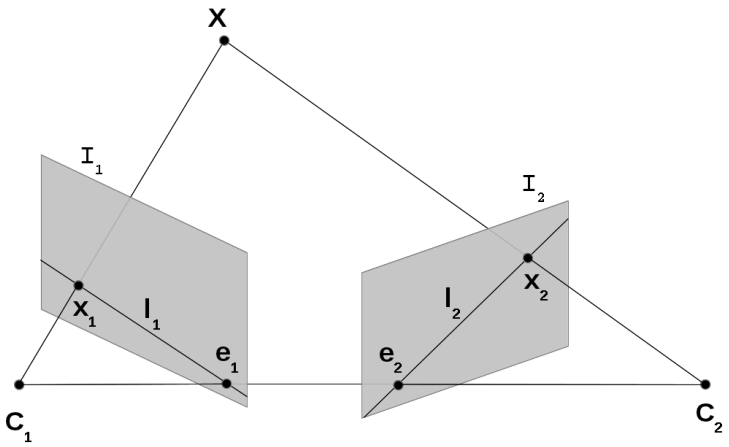


Abbildung A.2: Epipolargeometrie einer Stereoaufnahme.

A.5 Epipolargeometrie

Abbildung A.2 zeigt die Abbildung eines Punktes $\mathbf{X}$ bei einer Stereoaufnahme. Die von den beiden Kamerazentren zum Objektpunkt führenden Projektionsstrahlen spannen zusammen eine Ebene auf, die sog. Epipolarebene. Ihre Schnittlinien $\mathbf{l}_1$ und $\mathbf{l}_2$ mit den Kamerabildebenen werden als Epipolarlinien bezeichnet. Eine alternative Sichtweise ist die Beschreibung der Epipolarlinie als Abbildung des Projektionsstrahls auf die jeweils andere Kamerabildebene. Die Abbildung des Kamerazentrums d.h. der Durchstoßpunkt der Kamerabasis mit den Bildebenen wird als Epipol bezeichnet. Die Bedeutung der Epipolargeometrie für die Rekonstruktion liegt in der Einschränkung des Suchraums für die Stereokorrespondenzbestimmung.

A.6 Fundamentalmatrix und Essentielle Matrix

Die algebraische Repräsentation der Epipolargeometrie stellt die Fundamentalmatrix $\mathbf{F}$ dar. Sie beschreibt die relative Orientierung zwischen beiden Kameras. $\mathbf{F}$ ist eine homogene Matrix mit Rang 2 mit 7 Freiheitsgraden (Herleitung und weitere Eigenschaften können der umfangreichen Literatur zum Thema entnommen werden, so z.B. von Longuet-Higgins [1981], Hartley [1992]; Hartley und Zisserman [2000] und Faugeras [1993]). Mit ihrer Hilfe kann die grundlegende Gleichung für die zwei Bildpunkte eines Weltpunktes im Stereofall angegeben werden:

$$\mathbf{x}_1^T \mathbf{F} \mathbf{x}_2 = 0 \tag{A.22}$$

Die Berechnung der Fundamentalmatrix kann bei kalibrierten Kameras $\mathbf{P}_1 = \mathbf{K}_1[\mathbf{I}|\mathbf{0}]$ und $\mathbf{P}_2 = \mathbf{K}_2[\mathbf{R}|\mathbf{t}]$ aus den Projektionsmatrizen erfolgen:

$$\mathbf{F} = \mathbf{K}_2^{-T}[\mathbf{t}]_x\mathbf{R}\mathbf{K}_1^{-1} \tag{A.23}$$

Die Fundamentalmatrix ermöglicht die Berechnung der Epipolarlinien zu Bildpunkten der jeweils anderen Kamera und damit die Einschränkung des Suchraums bei der Korrespondenzanalyse:

$$\mathbf{l}_2 = \mathbf{F}\mathbf{x}_1 \text{ bzw. } \mathbf{l}_1 = \mathbf{F}^T\mathbf{x}_2 \tag{A.24}$$

Für den Spezialfall kalibrierter Kameras und dem Vorliegen normalisierter Koordinaten $\hat{\mathbf{x}}_\mathbf{1}$, $\hat{\mathbf{x}}_\mathbf{2}$ wird die Fundamentalmatrix $\mathbf{F}$ zur Essentiellen Matrix $\mathbf{E}$ [Longuet-Higgins 1981].

$$\hat{\mathbf{x}}_\mathbf{1}^T\mathbf{E}\hat{\mathbf{x}}_\mathbf{2} = 0 \tag{A.25}$$

Die Essentielle Matrix repräsentiert damit also die räumliche relative Orientierung zweier normalisierter Kameras und errechnet sich aus der äußeren Orientierung. Die Anzahl der Freiheitsgrade reduziert sich auf 5 (Translation und Rotation der äußeren Orientierung mit jeweils 3 DOF reduziert um einen Freiheitsgrad für die unbekannte Skalierung):

$$\mathbf{E} = [\mathbf{t}]_x\mathbf{R} \tag{A.26}$$

Die Beziehung zur Fundamentalmatrix ist über folgende Gleichung gegeben:

$$\mathbf{E} = \mathbf{K}_2^T\mathbf{F}\mathbf{K}_1 \tag{A.27}$$

A.7 Rektifizierung

Die Rektifizierung vereinfacht die Anwendung von Stereokorrespondenzverfahren, indem sie die beiden Kameraansichten in eine definierte, für umfangreiche Berechnungen entlang der Epipolarlinien günstige Anordnung transformiert. Diese wird als Stereonormalfall bezeichnet und beschreibt eine achsenparallele Ausrichtung der Kamerabildebenen nebeneinander in einer Ebene [Hartley und Zisserman 2000; Luhmann 2010]. Die Epipolarlinien werden so zu horizontalen Linien mit identischem y-Koordinatenwert in beiden Bildern.

Ein weit verbreitetes Verfahren zur Rektifzierung von Stereobildern ist das Verfahren nach Fusiello et al. [2000]. Es errechnet eine Rotation um beide Kamerazentren, so dass die Fokusebenen koplanar werden und die Basislinie beinhalten. Dadurch wandern die Epipole ins Unendliche und die Epipolarlinien werden parallel ausgerichtet. Die Kamerazentren selbst bleiben unverändert und Rotation und interne Parameter werden für die neuen Projektionsmatrizen beider Kameras identisch gewählt, so dass man zwei quasi gleiche Kameras erhält, die sich nur durch ihre Verschiebung entlang der x-Achse unterscheiden.

Aus den bekannten alten Projektionsmatrizen $\mathbf{P}_o$ der kalibrierten Kameras

$$\mathbf{P}_o = \mathbf{K}_o[\mathbf{R}_o|\mathbf{t}] = \mathbf{K}_o[\mathbf{R}_o| - \mathbf{R}_o\mathbf{c}] = [\mathbf{Q_o}|-\mathbf{Q}_o\mathbf{c}] \tag{A.28}$$

und den nach obigen Angaben erzeugten neuen Projektionsmatrizen des Stereonormalfalls

$$\mathbf{P}_n = \mathbf{K}_n[\mathbf{R}_n| - \mathbf{R}_n\mathbf{c}] = [\mathbf{Q}_n| - \mathbf{Q}_n\mathbf{c}] \tag{A.29}$$

kann dann die Transformation für die Rektifizierung errechnet werden (jeweils für jede Kamera):

$$\mathbf{T} = \mathbf{Q}_n\mathbf{Q}_o^{-1}. \tag{A.30}$$

Mit dieser Transformationsmatrix werden sodann die Bilder (oder Markerkoordinaten) in die rektifizierte Darstellung überführt.

A.8 Kalmanfilter

Der Kalmanfilter [Kalman 1960] stellt ein weit verbreitetes Standardverfahren zur Filterung aller Arten von linearen Systemen dar. Einführungen in die Thematik finden sich in [Welch und Bishop 1995; Foxlin et al. 2002; Gelb 1999].

Das lineare System wird über eine Reihe von nicht direkt bestimmbaren Zustandsvariablen beschrieben sowie über ein normalverteiltes Rauschen von System und Messprozess. Der Kalmanfilter liefert eine Schätzung für die unbekannten Zustandsvariablen anhand der Reihe der rauschbehafteten Messwerte. Diese fortlaufende Schätzung wird jeweils mittels der aktuellen Messdaten korrigiert.

Mit dem Zustandsvektor $\mathbf{x}_k$ und dem Messwertvektor $\mathbf{z}_k$ (jeweils zum diskreten Zeitpunkt k) wird über folgende Gleichungen die Zustandsfortschreibung bzw. der Messvorgang des Systems modelliert:

$$\mathbf{x}_{k+1} = \mathbf{\Phi}_k \mathbf{x}_k + \mathbf{w}_k \tag{A.31}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{A.32}$$

$\mathbf{\Phi}$ bezeichnet die Zustandsübergangsmatrix und $\mathbf{H}$ die Messmatrix. Die unabhängigen Zufallsvariablen $\mathbf{w}$ bzw. $\mathbf{v}$ beschreiben das Rauschverhalten von System bzw. Messprozess, und werden als normalverteiltes weißes Rauschen modelliert, mit den Kovarianzmatrizen $E[\mathbf{w}_k \mathbf{w}_k^T] = \mathbf{Q}_k$ und $E[\mathbf{v}_k \mathbf{v}_k^T] = \mathbf{R}_k$. Der Kalmanfilter liefert nun fortlaufend Schätzungen des Systemzustands $\hat{\mathbf{x}}_k$, und versucht den Fehler $\mathbf{e}_k$ zum tatsächlichen Systemzustand zu minimieren. Hierzu bedient er sich der Kovarianz des Schätzfehlers $\mathbf{P}_k = E[\mathbf{e}_k \mathbf{e}_k^T]$.

Die eigentliche Operation des Filters unterteilt sich in zwei einander abwechselnde Phasen, die Schätzung des neuen Zustands samt Messwert aus dem bisherigen (Prädiktionsphase, *time update*) und die Aktualisierung der Zustandsschätzung anhand eines neuen Messwerts (Korrekturphase, *measurement update*).

Während der Prädiktionsphase werden der Systemzustand und die Kovarianz des Schätzfehlers fortgeschrieben:

$$\hat{\mathbf{x}}_{k+1} = \mathbf{\Phi}_k \hat{\mathbf{x}}_k \tag{A.33}$$

$$\mathbf{P}_{k+1} = \mathbf{\Phi}_k \mathbf{P}_k \mathbf{\Phi}_k^T + \mathbf{Q}_k \tag{A.34}$$

Während der anschließenden Korrekturphase werden die Zustands- und Kovarianzschätzung mittels des aktuellen Messwerts aktualisiert:

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k) \tag{A.35}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k \tag{A.36}$$

$\mathbf{K}_k$ ist die Kalman Matrix. Sie wird zu Beginn der Korrekturphase nach folgender Vorschrift aktualisiert, die sicherstellt, dass der Filter dem Systemzustand mit minimalem Fehler folgt [Welch und Bishop 1995]:

$$\mathbf{K}_k = \mathbf{P}_k \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{A.37}$$

Anhang B

Komplettsysteme und Softwarebibliotheken für optisches Tracking

In diesem Anhang wird detaillierter auf einige der in Kapitel 2 vorgestellten Lösungen für optische Trackingaufgaben eingegangen, wie sie zu Beginn der Entwicklung des MOSCOT-Systems verfügbar waren.

B.1 Beispiele für kommerzielle optische Trackingsysteme

In Abschnitt 2.3 wurden die kommerziell verfügbaren Komplettsysteme in drei Klassen eingeteilt. Zu jeder Klasse werden nachfolgend einige typische Beispielsysteme vorgestellt, die insbesondere auch im Bereich der Medizintechnik zum Einsatz kommen. Der interne Aufbau der Systeme sowie ihre Hard- und Softwarearchitektur sind in der Regel nicht offengelegt und können daher nicht näher beschreiben werden. Für die technischen Daten wurde aus Mangel an aktuellen unabhängigen Vergleichen auf Angaben der Hersteller bzw. auf die Systemübersicht in Buaes [2005] zurückgegriffen.[1]

Trinokulare Systeme mit Zeilenkameras

Diese Systeme arbeiten mit aktiven LED-Markern und drei in einem Gehäuse integrierten CCD-Zeilenkameras im Infrarotbereich des optischen Spektrums. Die erhältlichen Systeme unterscheiden sich im wesentlichen in der erzielbaren Genauigkeit, Geschwindigkeit und erfasstem Trackingvolumen, und als direkte Folge hiervon

[1] Stand der Informationen ca. 2005

	NDI Optotrac	Krypton K400	easyTrack500
Kameratechnik	CCD-Zeilensensor	CCD-Zeilensensor	CCD-Zeilensensor
Trackingvolumen	1,5mx1,5mx1,5–3m	ca. 1,5mx1,5mx4m	$500mm^3$
Genauigkeit	0,15 mm	0,06–0,09mm	0,2mm
räumliche Auflösung	0,01 mm	0,002 mm	0,1mm
zeitliche Auflösung	3000 LED/s	3000 LED/s	300 LED/s
min. Latenz (abh. von Markerzahl)	2ms	1ms	10ms
max. Markerzahl	256	21	12
max. Framerate bei 1/4/20 Objekten	1000/250/50	1000/250/–	100/25/–
min. Latenz bei 1/4/20 Objekten	2/4/60 ms	1/4/– ms	10/40/– ms

Tabelle B.1: Spezifikationen einiger ausgewählter Systeme mit trinokularem Setup

in Abmessungen und Preis. Tabelle B.1 fasst die wesentlichen Daten einiger Systeme zusammen:

Das **Optotrak**-System der Firma **NDI** (Northern Digital Inc.)[2] ist ein weitverbreitetes Standardsystem, das insbesondere auch für medizinisches Tracking bei Diagnose und Intervention eingesetzt wird. Eine ähnliche, kostengünstige Alternative hierzu stellt das **easyTrack**-System der Schweizer Firma **atracsys**[3] dar, welches ebenfalls für die erwähnten medizinischen Anwendungsgebiete ausgelegt ist. Die Unternehmen **Metris** (mittlerweile in Nikon Metrology aufgegangen)[4] liefert mit der **K-Serie** eine Reihe von Koordinatenmesssystemen, die mit einer optionalen Dynamikoption ebenfalls für Trackingzwecke verwendet werden können. Diese Systeme sind am oberen Ende des Leistungsspektrums optischer Trackingsysteme angesiedelt und werden häufig als Referenzsystem verwendet.

Binokulare Systeme mit Flächenkameras

Diese Systeme arbeiten mit zwei in einem gemeinsamen Gehäuse integrierten Flächenkameras, je nach Sensortyp im infraroten oder im sichtbaren Bereich des optischen Spektrums. Das Messkonzept wird mit aktiven oder passiven Markern realisiert, bei manchen Systemen auch im Mischbetrieb. Tabelle B.2 fasst die wesentlichen Daten einiger Systeme zusammen:

Das **Polaris**-System der Firma **Northern Digital Inc.** ist ein weit verbreitetes optisches Trackingsystem im medizinischen Bereich. Es wird in allen Arten von medizinischen Anwendungen verwendet, unter anderem bei Diagnose, computernavi-

[2] http://www.ndigital.com
[3] http://www.atracsys.com
[4] http://www.nikonmetrology.com

	MicronTracker2 S60	Microntracker2 H40	NDI Polaris
Kameratechnik	2x CCD 1024x768@20Hz	unbekannt	unbekannt
Trackingvolumen	ca. 0,7m × 0,85m × 1,15m	1,2m × 0,9m × 1,2m	1m × 1m × 2,5m
räumliche Auflösung	0,05–0,2mm	0,05–0,2mm	unbekannt
zeitliche Auflösung	30Hz	15Hz	60/30Hz
Latenz	45ms	85ms	unbekannt
Genauigkeit	0,25mm RMS	0,25mm RMS	0,35mm RMS
max. Markerzahl	100	100	unbekannt

Tabelle B.2: Spezifikationen einiger ausgewählter Systeme mit binokularem Setup

gierten Operationen oder Roboterüberwachung. Es verwendet Infrarotkameras und unterstützt sowohl passive retroreflektierende Marker, die von Ringlichtern um die Kameras illuminiert werden, als auch aktive IR-LED-Marker.

Im Unterschied dazu basiert das **MicronTracker2**-System von **Claron Technology**[5] auf der Erkennung passiver Schwarz-Weiss-Marker sogenannter Xpoints (in der Photogrammetrie als linienhafte Zielmarken eingesetzt) basiert. Neben den kleinen räumlichen Abmessungen sind die kostengünstigen robusten Marker zu erwähnen, sowie der Verzicht auf aktive Beleuchtung, sodass das System mit Umgebungslicht auskommt.

Systeme aus Einzelkameramodulen

Diese Systeme arbeiten mit einer variablen Anzahl an separaten Kameramodulen. Aufgrund der Rekonstruktionsgeometrie werden Flächenkameras verwendet. Die einzelnen Kameramodule arbeiten mit CCD-Sensoren im Infrarotbereich und sind mit IR-Ringleuchten ausgestattet. Zur Objektsignalisierung werden retroreflektierende kreis- bzw. kugelförmige Marker eingesetzt. Zur Reduktion der erforderlichen Datenbandbreite wird meistens mit intelligenten Kameramodulen mit integrierter Bildverarbeitungshardware gearbeitet.

Im Unterschied zu den bisher vorgestellten monolithischen Systemen, müssen diese Systeme aufgrund der variablen Erfassungsgeometrie vom Anwender kalibriert werden. Ebenso ist ein deutlich höherer Aufwand für den Anschluss aller Kameramodule an den zentralen Rekonstruktionsrechner zu betreiben. Dafür erlauben diese Systeme jedoch eine deutlich größere Flexibilität. Angaben zur Genauigkeit sind bei modularen Systemen prinzipbedingt mit einer hohen Variabilität behaftet, da hier sehr stark der Designfaktor des Trackingszenarios, also die Marker- und Kameraanordnung eingeht.

[5] http://www.clarontech.com

	ARTtrack1	Vicon Tracker MX	Smart Motion Capture
Kameratechnik	CCD 658x496@60Hz	CMOS 2352x1728	CCD 640x480@120Hz
Trackingvolumen	ca. 3m × 3m × 10m	4m × 4m × 3m	4m × 3m × 2m
räumliche Auflösung	0,06mm	0,2mm	unbekannt
zeitliche Auflösung	60Hz	160Hz	120Hz
Latenz	40ms	3–10ms	10ms
Genauigkeit	0,4mm RMS	0,1mm RMS	1mm
max. Markerzahl	20	150	unbekannt

Tabelle B.3: Spezifikationen einiger ausgewählter modularer Systeme

Das **ARTtrack1**-System der **Advanced Realtime Tracking GmbH**[6] ist ein modulares System für Outside-In-Tracking von retroreflektierenden Markern. Die intelligenten Kameramodule verfügen über eine IR-Illumination und werden hier über Ethernet an einen zentralen PC angebunden. Zur Vor-Ort-Kalibrierung wird ein Referenzrahmen eingesetzt. Es steht eine Softwarebibliothek bereit (DTrack), die eine Einbindung des Systems in eigene Projekte ermöglicht.

Die Firma **Vicon**[7] stellt mit dem **Vicon Tracker** eine komplette Systemfamilie mit unterschiedlichen Kameramodulen vor, die sich in Auflösung, Geschwindigkeit und Preis unterscheiden. Es kommen jedoch immer retroreflektierende IR-Marker zum Einsatz. In der Tabelle sind die Werte für ein schnelles und hochauflösendes Kameramodul angegeben.

Das **Smart Motion Capture** System der Firma **BTS**[8] ist ein weiteres Beispiel eines modularen Trackingsystems. Die Funktionsweise basiert wie bei den anderen Systeme auf intelligenten Kameras mit IR-Beleuchtung in Outside-In-Anordnung und retroreflektierenden Markerkugeln an den Objekten.

B.2 Softwarebibliotheken für optisches Tracking

In Abschnitt 2.4 wurde bereits kurz auf einige Softwarebibliotheken eingegangen, die Lösungen für optische Trackingaufgaben bereitstellen. Im folgenden werden diese Softwarebibliotheken detaillierter beschrieben sowie weitere Softwarepakete vorgestellt, die sich zumindest in Teilaspekten mit Fragestellungen des optischen Trackings befassen.

[6] http://www.ar-tracking.com
[7] http://www.vicon.com
[8] http://www.btsbioengineering.com

Es wird eine Unterteilung der Softwarelösungen entsprechend ihrer primären Einsatzbereiche bzw. ihrer Position in der Datenverarbeitungskette der Gesamtanwendung vorgenommen: Systemsoftware ist an ein dediziertes Gerät gebunden und ermöglicht dessen Kalibration und Inbetriebnahme. Middleware abstrahiert vom Gerät und seinen spezifischen Datentypen und stellt Datenanbindung und -verteilung zur eigentlichen Anwendung hin bereit. Weitere Module und Subsysteme für optisches Tracking finden sich auch in manchen Frameworks für VR- und AR-Anwendungen. Schließlich gibt es noch Bibliotheken, die ausschließliche Bildverarbeitungsalgorithmen bereitstellen, unabhängig von Hardware- oder Systemintegration.

DTrack (Systemsoftware):

Ein Beispiel für Systemsoftware ist das Paket DTrack [ART GmbH 2009], das Softwarefrontend für Konfiguration und Betrieb der modularen Trackingsyteme der Advanced Realtime Tracking (A.R.T.) GmbH (siehe auch vorigen Abschnitt B.1). Die Software bietet eine Benutzerschnittstelle für die allgemeine Systemadministration sowie die Kalibration von Raum und Objekten. Während des Betriebes liefert sie einzelne 3DOF-Markerpositionen und auch rekonstruierte Objekte mit 6 Freiheitsgraden. Ein Monitormodus ermöglicht die Überwachung des Trackingvorgangs. Es stehen Schnittstellenmodule zur Anbindung an die Middlewarebibliohtken TrackD, VRPN und Gadgeteer/VR-Juggler bereit. TrackD unterstützt die Plattformen Windows und Linux.

TrackD (Middleware):

TrackD ist ein (mittlerweile eingestelltes) kommerzielles Produkt der Firma VRCO, das eine dünne Abstraktionsschicht als Middleware zwischen Trackinggeräten und Anwendungen darstellt. Es ist als Dämon-Prozess ausgelegt, der über diverse Treiber für kommerzielle Trackingsysteme verfügt. Der Dienst stellt den verbundenen Anwendungen damit eine plattformunabhängige Schnittstelle zur Geräteabstraktion und Netzwerktransparenz bereit. Er bietet Anbindung in den Sprachen C und C++ und unterstützt die Plattformen Windows, Linux und Mac OS X.

VRPN (Middleware):

VRPN (Virtual Reality Private Network) [Russell M. Taylor et al. 2001] ist eine Open-Source-Programmbibliothek, die viele kommerzielle Trackingsysteme unterstützt. Sie dient ähnlich wie TrackD zur Verbindung von Trackingsystemen mit Anwendungen. Sie bietet auf einem hohen Niveau Geräteabstraktion über vordefinierte Datentypen und Netzwerktransparenz für verteilte Anwendungen. Das weiter unten beschriebene OpenTracker-Projekt kann beispielsweise auf VRPN zur Geräteabstraktion aufbauen. VRPN ist in C++ realisiert und unterstützt die Plattformen Windows, Linux und Mac OS X.

Gadgeteer (Middleware):

Gadgeteer ist ein Teilprojekt des VR-Frameworks VR-Juggler (s.u.) und umfasst die Anbindung verschiedener VR-Devices, darunter auch kompletter Trackingsystemen. Es stellt über gerätespezifische Treiber eine Abstraktion der unterstützten Geräte dar und sorgt für den netzwerktransparenten Datenaustausch über genormte Datentypen. Gadgeteer ist in C++ implementiert und auf den Plattformen Windows, Linux und Mac OS X lauffähig.

VR Juggler (VR-Framework):

Das OpenSource-Projekt VRjuggler [Bierbaum et al. 2001] der Iowa State Universität stellt eine Plattform mit vielfältigen Modulen für die Entwicklung von VR-Anwendungen bereit, beispielsweise Szenegraphen, VR-Modelle, ein Konfigurationssystem sowie GUI- und Hilfsbibliotheken. Die eigentliche Anbindung von Trackingsystemen ist wie beschreiben im Subprojekt Gadgeteer zusammengefasst. Innerhalb von VR-Juggler selbst stehen allerdings Simulationsmodule für verschiedene Arten von Eingabegeräten zur Verfügung und damit die Möglichkeit, synthetische oder aufgezeichnete 3D-Daten in eine VR-Umgebung einzuspielen, ohne ein reales Trackingsystem aufzubauen. VR-Juggler ist ebenso wie Gadgeteer in C++ realisiert und unterstützt alle wichtigen Plattformen.

OpenTracker (Middleware mit AR-Trackingarchitektur):

Das OpenSource-Projekt Opentracker [Reitmayr und Schmalstieg 2005] wird von der Universiät Wien als Teil der Augmented-Reality-Umgebung Studierstube [Schmalstieg et al. 2002] entwickelt. Es stellt eine modulare, objektorientierte Architektur zur Datenverteilung bereit. Der Schwerpunkt liegt auf der Vereinfachung der Adaption der Middleware an neue Aufgabenstellungen. Hierzu wird eine generische Datenfluss-Architektur mit Graphenstruktur und Ereignisverteilung über ein Netzwerk sowie eine proprietäre XML-basierte Konfigurationssprache implementiert. Sie ermöglicht die einfache Generierung angepasster Datenflussgraphen innerhalb eines verteilten Systems. Der Fokus von OpenTracker liegt auf der Bereitstellung von vorverarbeiteten Daten für die Datenfusion in VR- und AR-Anwendungen, beispielsweise in einer Cave. Trackingsysteme werden dabei als fertige Datenquellen für 3D-Daten behandelt, es existiert keine Low-Level-Integration mit der Bildverarbeitungshard- oder -software. Opentracker ist eine C++-Architektur und unterstützt die Plattformen Windows, Linux und Mac OS X.

XVision (einfache Trackingbibliothek):

Das XVision-Projekt [Hager und Toyama 1998] der Johns-Hopkins-Universität ist eines der ältesten Open-Source-Softwareprojekte zum Thema optisches Tracking. Da

das System auf frühen Grafikworkstations entwickelt wurde, ist es auf niedrige Ressourcenanforderungen zugeschnitten. Das Projekt wird jedoch seit 2002 nicht mehr weiterentwickelt. Es stellte ein für den Nutzer schnell konfigurierbares Toolkit zur Entwicklung einfacher Trackinganwendungen bereit, wobei es sich auf das Tracking geometrischer Primitive wie Kanten, Regionen, Blobs konzentriert. XVision ist plattformunabhängig in C/C++ realisiert.

ARToolKit (AR-Framework mit Bildverarbeitungs- und Trackingsubsystem):

ARToolKit [Kato 2001] ist eine von der Universität Washington unter einer Open-Source-Lizenz entwickelte Bibliothek für Augmented-Reality-Anwendungen. Die Firma ARToolWorks bietet kommerziellen Support und Produkte für die Plattform an. Das Framework unterstützt die einfache Realisierung von AR-Projekten durch Funktionen zur Kameraanbidung, Bildverarbeitung und 3D-Darstellung. Eine Input-Library unterstützt den Anschluss verschiedener Kameras. Einfache Algorithmen zur Kamerakalibration stehen bereit. Die Software detektiert monochrome quadratische Flächenmarker mit Kodierung in sechs Freiheitsgraden relativ zur Kameraposition und ermöglicht die Überlagerung der Szene mit virtuellen 3D-Objekten. Es wird nur dieser Markertyp unterstützt, der für AR-Anwendungen zur Unterscheidung einer größerer Objektanzahl konzipiert ist. Rekonstruktionsgenauigkeit und Stabilität der Ergebnisse sind für reine Trackinganwendungen nicht ausreichend. Abgesehen vom Einlesen von Bilddaten existiert auch keine weitere Integration mit Kamera- und Bildverarbeitungshardware. ARToolKit ist hinsichtlich Ressourcenbedarf, Geschwindigkeit und Latenzzeiten auf kleinere AR-Demonstrationsanwendungen ausgelegt. Das Programmpaket ist in C implementiert und als Open-Source-Projekt für viele verschiedene Plattformen verfügbar, darunter Windows, Linux, Mac OS X aber auch eingebettete und mobile Betriebssysteme wie Windows Mobile, iOS und Android.

OpenCV (Computer-Vision-Bibliothek):

Das Projekt Open Computer Vision (kurz OpenCV) [OpenCV 2006; Bradski und Kaehler 2008] beinhaltet eine große Sammlung an Algorithmen rund um das Thema Computer Vision, darunter auch Module zu Kamerakalibration, Bildverarbeitung, Stereorekonstruktion und Tracking. Die zugehörige Programmbibliothek für die Sprachen C und C++ wird unter einer Open-Source-Lizenz zur Verfügung gestellt. Vorteile der Bibliothek sind die große Menge an enthaltenen Algorithmen zu teils auch aktuellen Forschungsgebieten der Computer Vision und die Plattformunabhängigkeit. Die Software arbeitet nur mit Bilddaten, es wird keine Hardwareanbindung zur Bilderfassung bzw. Hardwareunterstützung bei der Bildverarbeitung zur Verfügung gestellt. Es existieren Algorithmen zur Kamerakalibration, jedoch werden erst nach und nach Themen wie Tracking und 3D-Rekonstruktion in einem Unterprojekt fokussiert angegangen. Bisher liegt der Schwerpunkt der Bibliothek

bei rechenintensiven, komplexen Algorithmen wie Shape- und Gesture-Recognition, Optical-Flow, View-Morphing und weniger bei markerbasiertem Tracking für geringe Ressourcenanforderungen. OpenCV ist eine reine Sammlung von Algorithmen in der Sprache C++, es stellt kein Framework für markerbasierte Trackingaufgaben bereit. Unterstützt werden Plattformen von Windows über Linux bis zu Mac OS X.

Anhang C

Kriterien zur Auswahl geeigneter Markertypen

Die folgenden Erläuterungen sollen eine Hilfestellung geben bei der Auswahl geeigneter Markertypen für die Lösung unkonventioneller Trackingaufgaben. Es werden verschiedene Markertypen beschrieben, die bei Trackinganwendungen zum Einsatz kommen können. Mit Ausnahme von Linienmarkern und AR-Markern können die beschriebenen Markertypen direkt innerhalb des MOSCOT-Systems eingesetzt werden.

Tabelle C.1 stellt zentrale Eigenschaften der Markertypen einander gegenüber, Tabelle C.2 gibt eine (subjektive) Bewertung verschiedener, für den Einsatz relevanter Randbedingungen. Vertiefende Beschreibungen zu einzelnen Markertypen finden sich beispielsweise in [Fiala 2005; Owen et al. 2002; Zhang et al. 2002] für monochrome template-basierte Marker (AR-Marker), in [Jun et al. 1997; Cho und Neumann 1998; Kravtchenko 1999] für Farbmarker und in [Ahn und Schultes 1997; Falcon GmbH 2008] sowie [Luhmann 2010, Kap. 3.4] für Marker bei Photogrammetrieanwendungen.

LED-Marker: Leuchtdioden werden als selbstleuchtende aktive Marker eingesetzt. Zur Reduktion von Störeinflüssen dienen auf das LED-Spektrum angepasste optische Passfilter vor den Kamerasensoren. Das Gros der Anwendungen arbeitet im nahen Infrarotbereich, da Siliziumsensoren in diesem Bereich ein Empfindlichkeitssmaximum haben und IR-Licht vom Anwender nicht störend wahrgenommen wird. Farb-LEDs bieten den Vorteil einer Codierung – allerdings nur für eine geringe Zahl von Markern bzw. Markergruppen. Eine weitere Möglichkeit zur Markeridentifikation besteht durch sequentielle Schaltung der LEDs. Allerdings erhöht sich so der Schaltungsaufwand deutlich und die Erfassungsgenauigkeit bewegter Objekte nimmt ab. Als wesentliche Einschränkungen sind der Strombedarf und die resultierende Verkabelung zu nennen. Zusammen mit retroreflektierenden Marken sind LEDs die Standardmarker bei Trackingsystemen für Motion-Capture und industrielle Anwendungen.

retroreflektierende Marker: Dies sind passive, ebene oder kugelförmige Marker aus einem speziellen Material, das durch eingebettete Mikroglaskugeln das Licht direkt zur Lichtquelle reflektiert. Durch eine Ringillumination um das Kameraobjektiv wird ein mit aktiven Markern vergleichbarer Kontrast im Kamerabild erreicht. Der Einsatz erfolgt meist im nahen IR-Spektrum, ist aber ebenso im sichtbaren Bereich möglich. Problematisch ist die Empfindlichkeit des Materials gegen Verschmutzung (z.B. Fingerabdrücke etc). Dreidimensionale (kugelförmige) Marker unterliegen keinen perspektivischen Verzerrungen, allerdings erfordern sie auch erhöten Raum- und Montageaufwand. Da keine innere Struktur der Marker besteht ist keine Codierung möglich. Zusammen mit den LED-Markern sind sie weit verbreitet bei Motion-Capture, medizinischen und industriellen Anwendungen.

passive Farbmarkierungen: Solche Markierungen sind leicht applizierbar durch Farbauftrag oder bekleben mit Farbfolie. Sie zeichnen sich weiterhin durch niedrige Kosten und sehr geringen Raumbedarf aus. Gute Codierbarkeit existiert beim Einsatz weniger Marker oder über eine Gruppencodierung. Der Einsatz ist insbesondere vorteilhaft bei Anwendungen, die nur wenig Ressourcen für das Tracking aufwenden können (z.B. medizinische Simulatoren wie EYESI oder in der mobilen Robotik). Auch diese Art von Markern benötigt keine direkte Beleuchtung. Problematisch ist jedoch, dass die Farberkennung stark von der Beleuchtungskonfiguration abhängt und sich auch mit dem Reflektionswinkel leicht verändert, so dass die Genauigkeit gegenüber monochromen Markertypen eingeschränkt ist.

Photogrammetrische Marker mit innerer Kantenstrukturen: Dies sind passive, planare Marker in verschiedenen Formen wie konzentrischen Kreise, schachbrettartigen Mustern (XPoint-Marker). Sie zeigen eine höhere Robustheit gegenüber Verschmutzungen als Marker ohne innere Struktur. Allerdings ist der Raumbedarf deutlich größer und die Detektion benötigt aufwendigere Kantendetektionsverfahren.

codierte Photogrammetriemarker : Konzeptionell ähnlich den uncodierten Photogrammetriemarkern erfordern sie diesen gegenüber nochmals aufwendigere Bildverarbeitungsmethoden zur Erfassung der Codierung. Der Raumbedarf ist ebenfalls weiter erhöht. Vorteilhaft ist die durch die Codierung eindeutige Markerzuordnung und damit triviale Lösung des Korrespondenzproblems.

Linienmarker : Einige wenige Spezialanwendungen wurden vorgestellt, die auf der direkten Detektion von linienartiger Strukturen beruhen. Die Markerdetektion als auch die Rekonstruktion unterscheidet sich stark von den anderen vorgestellten Markertypen. Die Markerstruktur kann über Kantendetektionsverfahren bzw. bei Linien einer gewissen Stärke als stark elongierter Punktmarker

über Flächensegmentierung erfasst werden. Zusätzlich zur Position wird die Orientierung des Markers erfasst, so dass Linienmarker über 4 Freiheitsgrade verfügen.[1] Dieser Markertyp kommt in herkömmlichen Trackinganwendungen nicht zum Einsatz.

AR-Marker: Diese Marker besitzen eine innere Struktur, die aus mehreren ausgewiesenen Punkten und einem Template zur Codierung besteht. Damit definieren AR-Marker neben der Position auch die Orientierung und ermöglichen damit ein 6DOF-Tracking mit nur einem Marker. Außerdem ist so (bei allerdings stark verringerter Rekonstruktionsgenauigkeit) ein Tracking mit einer einzelnen Kamera möglich. Über die Codierung wird gewöhnlich dem Marker genau ein Objekt zugeordnet. Die Bildverarbeitungsanforderungen sind durch die komplexere Markerstruktur deutlich höher als bei einfacheren Markertypen, gleichzeitig ist die erreichbare Genauigkeit geringer. Dieser Markertyp wird bevorzugt bei AR-Anwendungen mit vielen zu erfassenden Objekten und nur einer Kamera eingesetzt.

[1] Wird zusätzlich die Länge der Linie erfasst, bzw. Anfangs- und Endpunkt können auch 5 Freiheitsgrade erfasst werden.

Art des Markers	Signalisierung	Schwellwert	Sensortyp	Subpixelinterpolation	innere Struktur	Codierung	2D/3D	Freiheitsgrade
IR-LEDs	aktiv	Kontrast	Zeile-Fläche	Ellipse/Zentroid	nein	ja/nein[1]	2D/3D[3]	3 DOF
Farb-LEDs	aktiv	Farbwert	Zeile-Fläche	Ellipse/Zentroid	nein	teilweise[2]	2D/3D[3]	3 DOF
retroreflektierend	retrorefl.	Kontrast	Fläche	Ellipse/Zentroid	nein	nein	2D/3D[4]	3 DOF
Farbmarkierung	passiv	Farbwert	Fläche	Ellipse/Zentroid	nein	teilweise[2]	2D/3D[4]	3 DOF
Kreuzmarker (XPoint)	passiv	Kontrast	Fläche	Kantenschnittpunkt	ja	nein	2D	3 DOF
codierte Marker	passiv	Kontrast	Fläche	Ellipse/Zentroid	ja	ja	2D	3 DOF
Linienmarker	passiv	Kontrast	Fläche	Kantendetektion	nein	nein	2D/3D	5 DOF
AR-Marker	passiv	Kontrast	Fläche	Kantenschnittpunkt	ja	ja	2D	6 DOF

[1] bei sequentieller/paralleler Aktivierung
[2] nur wenige verschiedene Codierungen durch Farbwert realisierbar, ausreichend für wenige Marker bzw. Gruppencodierung
[3] LED mit 180° Abstrahlwinkel besser als Flächenmarker, aber schlechter als Volumenmarker
[4] Ausführung als Flächenmarker (Kreis) und als Volumenmarker (Kugel) möglich

Tabelle C.1: Eigenschaften verschiedener Markertypen.

Art des Markers	Komplexität der Markerdetektion	Komplexität der Korrespondenzbestimmung	Markergröße	Genauigkeit	Robustheit bei Beleuchtungsänderungen	Robustheit gegenüber Verschmutzung	Keine Kameralichtquelle erforderlich	Keine Verkabelung erforderlich	Kosten (Hardware-Aufwand)
IR-LEDs	++	++/- -[1]	+	++	+/++	+	+	-	- -
Farb-LEDs	++	-	+	++	+	+	+	-	- -
retroreflektierend	++	- -	+	++	++	- -	-	+	-/- -[2]
Farbmarkierung	++	-	++	-	- -	-	+	+	++
Kreuzmarker (XPoint)	-	- -	-	+	+	+	+	+	-/+[3]
codierte Marker	-	++	- -	+	+	-	+	+	-/+[3]
Linienmarker	-	-	+	+	+	+	+	+	+
AR-Marker	- -	++	- -	-	-	+	+	+	++

[1] bei sequentieller/paralleler Illumination

[2] bei Verwendung von retroreflektierender Folie/Kugel

[3] bei Verwendung von retroreflektierender Folie/Schwarzweißdruck

Tabelle C.2: Bewertung verschiedener Markertypen. Für einen direkten Vergleich zwischen Markertypen sind die angegebenen Werte immer als qualitative Bewertung für einen Trackingeinsatz zu verstehen. Beispiel: Ein '-' in der Spalte Verkabelung besagt, dass eine (für das mögliche Einsatzspektrum negative) Verkabelung notwendig ist.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Abras, C., D. Maloney-Krichmar und J. Preece (2004). „User-centered design“. In: *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*.

Ahn, S. und M. Schultes (1997). „A new circular coded target for the automation of photogrammetric 3D-surface measurements“. In: *Optical 3-D measurement techniques IV*, S. 225–234.

Allison, R., L. Harris, M. Jenkin, U. Jasiobedzka und J. Zacher (2001). „Tolerance of temporal delay in virtual environments“. In: *IEEE Int. Conference on Virtual Reality* 3.

Alur, D., J. Crupi und D. Malks (2003). *Core J2EE patterns: best practices and design strategies*. Prentice Hall PTR.

Álvarez, J., T. Gevers und A. López (2010). „Learning photometric invariance for object detection“. In: *International Journal of Computer Vision* 90.1, S. 45–61.

ART GmbH, Hrsg. (2009). *System user manual ArtTrack und DTrack*. Version 2.2. ART GmbH.

Arun, K., T. Huang und S. Blostein (1987). „Least-squares fitting of two 3-D point sets“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 9.5, S. 698–700.

Austern, M. (1998). *Generic programming and the STL: using and extending the C++ Standard Template Library*. Addison-Wesley Longman Publishing Co. Inc. Boston, MA, USA.

Automated Imaging Association, Hrsg. (2004). *Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers, Version 1.1*. Automated Imaging Association, Ann Arbor, MI.

Azad, P., T. Gockel und R. Dillmann (2008). *Computer Vision: principles and practice*. Elektor Electronics Publishing.

Azarbayejani, A. und A. Pentland (1995). *Camera self-calibration from one point correspondence*. Techn. Ber. Massachusetts Institute of Technology. Perceptual Computing Section.

Bailey, D. (2011). *Design for Embedded Image Processing on FPGAs*. Wiley.

Bailey, D. und C. Johnston (2007). „Single pass connected components analysis". In: *Proceedings of Image and Vision Computing New Zealand*, S. 282–287.

Ballard, D. (1981). „Generalizing the Hough transform to detect arbitrary shapes". In: *Pattern recognition* 13.2, S. 111–122.

Bauer, M., M. Schlegel, D. Pustka, N. Navab und G. Klinker (Okt. 2006). „Predicting and Estimating the Accuracy of n-occular Optical Tracking Systems". In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*. Santa Barbara (CA), USA.

Bayer, B. (1976). *Color imaging array*. US patent 3,971,065.

Beardsley, P., A. Zisserman und D. Murray (1994). „Navigation using affine structure from motion". In: *Computer Vision—ECCV'94*, S. 85–96.

Beier, F. (2006). „Optisches Tracking von deformierbaren Objekten am Beispiel eines Interfaces fur den Operationssimulator EYESI". Diplomarbeit. Universität Mannheim.

Beier, F. (2012). „Entwicklung eines neurochirurgischen Trainingssimulators für intrakranielle Eingriffe". Diss. Ruprecht–Karls–Universität Heidelberg.

Benz, B. und B. Feddern (2012). „Ratgeber USB 3.0". In: *c't magazin für computer technik* 13, S. 108–121.

Bien, A. (2003). *J2EE Patterns-Studentenausgabe*. Pearson Education.

Bierbaum, A., C. Just, P. Hartling, K. Meinert, A. Baker und C. Cruz-Neira (2001). „VR Juggler: A Virtual Platform for Virtual Reality Application Development". In: *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*. Washington, DC, USA: IEEE Computer Society, S. 89.

Birsan, D. (2005). „On plug-ins and extensible architectures". In: *Queue* 3.2, S. 40–46.

Bishop, G., G. Welch und B. Allen (2001). „Tracking: Beyond 15 Minutes of Thought". In: *SIGGRAPH Course Pack*.

Blanchette, J. und M. Summerfield (2006). *C++ GUI programming with Qt 4*. Prentice Hall PTR.

Booch, G., R. Maksimchuk, M. Engle, B. Young, J. Conallen und K. Houston (2007). *Object-oriented analysis and design with applications*. Addison-Wesley Professional.

Bovik, A. (2009). *The essential guide to Image Processing*. Academic Pr.

Bradski, G. und A. Kaehler (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated.

Brady, P. (2006). *Interconnection Speeds Compared*. URL: `http://www.pixelbeat.org/speeds.html`.

Buaes, A. G. (2005). *A Survey on the Available Optical Tracking Systems for AR/VR Indoor Applications*. Techn. Ber. Universidade Federal Do Rio Grande Do Sul, Porto Alegre, Brazil.

Budinsky, F. (2003). *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Professional.

Burbeck, S. (1992). *Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC)*.

Buschmann, F. (1996). *Pattern-oriented software architecture: a system of patterns*. Wiley series in software design patterns. Wiley.

Cho, Y und U Neumann (1998). „Multi-ring Color Fiducial Systems for Scalable Fiducial Tracking Augmented Reality". In: *Proc of IEEE VRAIS*. Bd. 10. 6. IEEE, S. 212.

Cinquin, P., E. Bainville, C. Barbe, E. Bittar, V. Bouchard, L. Bricault, G. Champleboux, M. Chenin, L. Chevalier, Y. Delnondedieu et al. (1995). „Computer assisted medical interventions". In: *Engineering in Medicine and Biology Magazine, IEEE* 14.3, S. 254–263.

Collins, S. (2005). *A deeper look at signals and slots*. URL: `http://www.elpauer.org/stuff/a_deeper_look_at_signals_and_slots.pdf`.

Dawes, B. und D. Abrahams (2000–2007). *Boost C++ libraries*. URL: `http://www.boost.org/`.

Diederich, S. (2005). „Automatische Farbmarkereinstellung für ein Kamera-Trackingsystem“. Praktikumsbericht, Lehrstuhl Informatik V, Universität Mannheim.

Dietrich, S. und D. Walker (2005). „The evolution of real-time linux“. In: *Proc. 7th Real-Time Linux Workshop*, S. 3–4.

Digia Plc, Hrsg. (2012a). *Qt Reference Documentation*. url: `http://doc.qt.digia.com/4.7/`.

Digia Plc, Hrsg. (2012b). *Qt Reference Documentation, Signals and Slots*. url: `http://doc.qt.digia.com/4.7/signalsandslots.html`.

Dillinger, P. (2004). *Einsatz von FPGA Prozessoren als Datenreduktionshardware in der Bildverarbeitung*. Techn. Ber. Berichte des Forschungszentrums Jülich JUEL-4149. Forschungszentrum Jülich, Zentralbibliothek, Verlag.

Dold, C., M. Zaitsev, O. Speck, E. Firle, J. Hennig und G. Sakas (2006). „Advantages and limitations of prospective head motion compensation for MRI using an optical motion tracking device“. In: *Academic radiology* 13.9, S. 1093–1103.

Dold, J. (1997). „Ein hybrides photogrammetrisches Industriemesssystem höchster Genauigkeit und seine Überprüfung“. Diss. Universität der Bundeswehr.

Dorfmüller, K. (1999). „An Optical Tracking System for VRAR-Applications“. In: *Virtual Environments' 99, Proceedings of the Virtual Environments Conference & fifth Eurographics Workshop*. Citeseer.

Dorfmüller, K. und H. Wirth (1998). „Real-time hand and head tracking for virtual environments using infrared beacons“. In: *Lecture Notes in Computer Science*, S. 113–127.

Dorfmüller-Ulhaas, K. (2003). „Robust optical user motion tracking using a Kalman filter“. In: *10th ACM Symposium on Virtual Reality Software and Technology*.

Drepper, U. (2007). *What every programmer should know about memory*. Citeseer. url: `http://people.redhat.com/drepper/cpumemory.pdf`.

Eggert, D., A. Lorusso und R. Fisher (1997). „Estimating 3-D rigid body transformations: a comparison of four major algorithms“. In: *Machine Vision and Applications* 9.5, S. 272–290.

Falcon GmbH, Hrsg. (2008). *FalCon eXtra MovXact Markentypen*. Falcon GmbH. URL: `http://falcon.de/falcon/pdf/ger/datasheet/marker.pdf`.

Faugeras, O. (1993). *Three-dimensional computer vision: a geometric viewpoint*. MIT Press.

Fiala, M. (2005). „ARTag, a fiducial marker system using digital techniques“. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*. Bd. 2.

Fischler, M. und R. Bolles (1981). „Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography“. In: *Communications of the ACM* 24.6, S. 381–395.

Forsyth, D. und J. Ponce (2002). *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference.

Fowler, M. (2004). *Inversion of Control Containers and the Dependency Injection Pattern.,* URL: `http://www.martinfowler.com/articles/injection.html`.

Foxlin, E. et al. (2002). „Motion tracking requirements and technologies“. In: *Handbook of virtual environments: design, implementation, and applications* 8. Hrsg. von K. Stanney, S. 163–210.

Funt, B., K. Barnard und L. Martin (1998). „Is machine colour constancy good enough?“ In: *Computer Vision—ECCV'98*, S. 445–459.

Fusiello, A., E. Trucco und A. Verri (2000). „A compact algorithm for rectification of stereo pairs“. In: *Machine Vision and Applications* 12.1, S. 16–22.

Gamma, E. und K. Beck (2003). *Contributing To Eclipse: Principles, Patterns, And Plug-Ins*. Addison-Wesley Professional.

Gamma, E., R. Helm, R. Johnson und J. Vlissides (2001). *Entwurfsmuster: Elemente wiederverwendbarer opjektorientierter Software*. Addison-Wesley Longman Publishing Co. Inc. Boston, MA, USA.

Gavrila, D. (1999). „The visual analysis of human movement: A survey“. In: *Computer vision and image understanding* 73.1, S. 82–98.

Gelb, A. (1999). *Applied optimal estimation*. MIT press.

Gold, S., A. Rangarajan, C. Lu, S. Pappu und E. Mjolsness (1998). „New algorithms for 2D and 3D point matching:: pose estimation and correspondence“. In: *Pattern Recognition* 31.8, S. 1019–1031.

Golub, G. und C. Van Loan (1996). *Matrix computations*. Bd. 3. Johns Hopkins Univ Pr.

Gregor, D. (2004). *Boost.Signals*. Techn. Ber. Boost C++ Libraries. url: `http://www.boost.org/doc/html/signals.html`.

Grimm, J. (2005). „Interaktive Echtzeitmodellierung von biologischem Gewebe für Virtuelle Realitäten in der medizinischen Ausbildung“. Diss. Universität Mannheim.

Grimson, W., R. Kikinis, F. Jolesz und P. Black (1999). „Image-guided surgery“. In: *Scientific American* 280.6, S. 54–61.

Gunturk, B., J. Glotzbach, Y. Altunbasak, R. Schafer und R. Mersereau (2005). „Demosaicking: color filter array interpolation“. In: *Signal Processing Magazine, IEEE* 22.1, S. 44–54.

Hager, G. und K. Toyama (1998). „X vision: A portable substrate for real-time vision applications“. In: *Computer Vision and Image Understanding* 69.1, S. 23–37.

Halliday, D., B. Conway, S. Farmer und J. Rosenberg (1999). „Load-Independent Contributions From Motor-Unit Synchronization to Human Physiological Tremor“. In: *Journal of Neurophysiology* 82.2, S. 664–675.

Hamza-Lup, F., J. Rolland und C. Hughes (2004). „A distributed augmented reality system for medical training and simulation“. In: *Energy, simulation-training, ocean engineering and instrumentation: Research papers of the link foundation fellows* 4, S. 213–235.

Handel, H. (2004). „Development of a technique for Multi-Camera Calibration“. Diplomarbeit. Universität Mannheim.

Handel, H. (2009). „Algorithms for Building High-Accurate Optical Tracking Systems“. Diss. University of Heidelberg.

Hartley, R. (1992). „Estimation of relative camera positions for uncalibrated cameras“. In: *Computer Vision—ECCV'92*. Springer, S. 579–587.

Hartley, R. und A. Zisserman (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Hartley, R. I. und P. Sturm (1997). „Triangulation“. In: *Comput. Vis. Image Underst.* 68.2, S. 146–157.

He, L., Y. Chao und K. Suzuki (2007). „A linear-time two-scan labeling algorithm“. In: *Image Processing, 2007. ICIP 2007. IEEE International Conference on*. Bd. 5. IEEE, S. 241–245.

He, L., Y. Chao, K. Suzuki und K. Wu (2009). „Fast connected-component labeling“. In: *Pattern Recognition* 42.9, S. 1977–1987.

Hegner, J. (2005). „Development of an FPGA-based tracking camera“. Diplomarbeit. Universität Mannheim.

Held, R. und N. Durlach (1993). „Telepresence, time delay and adaptation“. In: S. 232–246.

Hennes, M. (2007). „Konkurierende Genauigkeitsmaße - Potential und Schwächen aus der Sicht des Anwenders“. In: *Allgemeine Vermessungsnachrichten* 4, S. 136–146.

Hoff, W. und T. Vincent (2000). „Analysis of Head Pose Accuracy in Augmented Reality“. In: *Visualization and Computer Graphics, IEEE Transactions on*, S. 319–334.

Hoff, W. und C. Golden (1999). „Fusion of data from head-mounted and fixed sensors“. In: *Augmented reality: placing artificial objects in real scenes: proceedings of IWAR'98*. AK Peters, Ltd., S. 167.

Hoffmann, D. (2006). „Entwicklung des optischen Trackings für einen Laparoskopie-Simulator“. Diplomarbeit. Universität Mannheim.

Holst, G. C. und T. S. Lomheim (2007). *CMOS/CCD sensors and camera systems*. JCD Publishing, SPIE.

Horn, B. (1987). „Closed-form solution of absolute orientation using unit quaternions“. In: *JOSA A* 4.4, S. 629–642.

Hu, M. (1962). „Visual pattern recognition by moment invariants“. In: *Information Theory, IRE Transactions on* 8.2, S. 179–187.

Huber, M., D. Pustka, P. Keitler, F. Echtler und G. Klinker (2007). „A system architecture for ubiquitous tracking environments“. In: *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, S. 1–4.

Ibanez, L., W. Schroeder, L. Ng, J. Cates et al. (2005). *The ITK software guide*. Kitware Inc.

IEEE, Hrsg. (2008). *IEEE Std. 1394-2008, Standard for a High-Performance Serial Bus*. IEEE.

ISO/IEC, Hrsg. (1999). *ISO 13407, Human-Centred Design Processes for Interactive Systems*. ISO/IEC.

Jablonski, M. und M. Gorgon (2004). „Handel-C implementation of classical component labelling algorithm“. In: *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*. IEEE, S. 387–393.

Jacobs, G. H. und M. P. Rowe (2004). „Evolution of vertebrate colour vision“. In: *Clinical and Experimental Optometry* 87.4-5, S. 206–216.

Jähne, B. (2005). *Digital Image Processing*. Springer.

Jain, R., R. Kasturi und B. Schunck (1995). *Machine vision*. Bd. 5. McGraw-Hill New York.

Jansen, C., F. Steinicke, K. Hinrichs, J. Vahrenhold und B. Schwald (2007). „Performance improvement for optical tracking by adapting marker arrangements“. In: *IEEE VR 2007 Workshop on Trends and Issues in Tracking for Virtual Environments*.

Johnston, C. und D. Bailey (2008). „FPGA implementation of a single pass connected components algorithm“. In: *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*. IEEE, S. 228–231.

Johnston, C., K. Gribbon und D. Bailey (2005a). „FPGA based remote object tracking for real-time control“. In: *International Conference on Sensing Technology New Zealand*, S. 66–72.

Johnston, C., D. Bailey und K. Gribbon (2005b). „Optimisation of a colour segmentation and tracking algorithm for real-time FPGA implementation“. In: *Proceedings of Image and Vision Computing New Zealand*, S. 422–427.

Jun, Y. C., J. Park und U. Neumann (1997). „Fast Color Fiducial Detection and Dynamic Workspace Extension in Video See-through Self-Tracking Augmented Reality“. In: *in Proceedings of the Fifth Pacific Conference on Computer Graphics and Applications*. IEEE Comput. Soc, S. 168–177.

Kalman, R. E. (1960). „A New Approach to Linear Filtering and Prediction Problems“. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D, S. 35–45.

Karlsson, B. (2005). *Beyond the C++ Standard Library: An Introduction to Boost*. Addison-Wesley Professional.

Kato, H. (2001). „Developing Applications with ARToolkit“. In: *SIGGRAPH 2001 Course Notes* 27.

Kato, H., M. Billinghurst und R. May (2005). *AR toolkit documentation*. url: `http://www.hitl.washington.edu/artoolkit/documentation/`.

Kawano, H., H. Shimoji, S. Yoshikawa, K. Miyatake, K. Hama und S. Nakamura (2008). „Optical testing of star sensor (I): Defocus spot measuring technique for ground-based test“. In: *Optical Review* 15 (2). 10.1007/s10043-008-0016-x, S. 110–117.

Kim, T., Y. Moon und C. Han (1999). „An efficient method of estimating edge locations with subpixel accuracy in noisy images“. In: *TENCON 99. Proceedings of the IEEE Region 10 Conference*. Bd. 1. IEEE, S. 589–592.

Kisačanin, B., S. Bhattacharya und S. Chai (2009). *Embedded computer vision*. Springer-Verlag New York Inc.

Knappe, P., I. Gross, S. Pieck, J. Wahrburg, S. Kuenzler und F. Kerschbaumer (2003). „Position control of a surgical robot by a navigation system“. In: *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings*. Bd. 4.

Köpfle, A., M. Schill, M. Schwarz, A. Pott, A. Wagner, R. Männer, E. Badreddin, P. Weiser und H. P. Scharf (Sep. 2004a). „A Modular Scalable Approach to Occlusion-Robust, Low-Latency Optical Tracking“. In: *Medical Image Computing and Computer-Assisted Intervention MICCAI*. St. Malo, France.

Köpfle, A., M. Schill, M. Rautmann, M. Schwarz, A. Pott, A. Wagner, R. Männer, E. Badreddin, P. Weiser und H. P. Scharf (März 2004b).

„Occlusion-Robust, Low-Latency Optical Tracking using a Modular Scalable System Archituecture“. In: *Medical Robotics, Navigation & Visualization MRNV*. Remagen, Germany.

Köpfle, A., F. Beier, C. Wagner und R. Männer (Feb. 2007). „Real-time marker-based tracking of a non-rigid object“. In: *Medicine Meets Virtual Reality (MMVR07)*. Palm Beach, California, USA.

Korb, W. (2007). *Technik und Physik der Chirurgischen Navigationssysteme*. URL: `http://www.iccas.de/ressource/lectures/CASMedizin/VL10-Navigation-Registrierung.pdf`.

Kraus, K. (1993). *Photogrammetry, Vol. 1: Fundamentals and Standard Processes*. Bildungsverlag Eins.

Kraus, K., J. Jansa und H. Kager (1997). *Photogrammetry, Vol. 2: Advanced Methods and Applications*. Bildungsverlag Eins.

Kravtchenko, V. (1999). „Tracking color objects in real time“. Magisterarb. The University of British Columbia.

Kuo, C., F. Huang und F. Yang (2004). „Development of active IR-based surgical marker tracking and positioning systems“. In:

Körner, O. (2003). „Entwicklung eines computergesteuerten Trainingssimulators für die Koloskopie mit aktivem Force-Feedback“. Diss. Universität Mannheim.

Lane, R. und N. Thacker (1998). „Tutorial: overview of stereo matching research“. In: *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*.

Lapeyre, J.-O. und R. Strandh (Jan. 2004). *An efficient union-find algorithm for extracting the connected components of a large-sized image*. Techn. Ber. Laboratoire Bordelais de Recherche en Informatique, Universite Bordeaux. URL: `http://citeseer.ist.psu.edu/lapeyre04efficient.html`.

Lau, W., N. Ramey, J. Corso, N. Thakor und G. Hager (2004). „Stereo-based endoscopic tracking of cardiac surface deformation“. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2004*, S. 494–501.

Li, H. und R. Hartley (2007). „The 3D-3D registration problem revisited“. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, S. 1–8.

Liu, B., D. Maier, M. Schill und R. Männer (2007). „Robust Real-time Tracking of Surgical Instruments in the Eye Surgery Simulator (EyeSi)“. In: *Fourth IASTED International Conference on Signal and Image Processing, SIP 2002*, S. 441–446.

Longuet-Higgins, H. C. (Sep. 1981). „A computer algorithm for reconstructing a scene from two projections“. In: *Nature* 293, S. 133–135.

Lucas, B., T. Kanade et al. (1981). „An iterative image registration technique with an application to stereo vision“. In: *Proceedings of the 7th international joint conference on Artificial intelligence*.

Luhmann, T. (1995). „Punktmessung in digitalen Bildern mit Subpixel-Genauigkeit“. In: *Proc. Bildverarbeitung'95-Forschen, Entwickeln, Anwenden*.

Luhmann, T. (2000). „Photogrammetrische Verfahren in der industriellen Messtechnik“. In: *Publikationen der DGPF* 9.

Luhmann, T. (2010). *Nahbereichsphotogrammetrie*. 3. Auflage. Wichmann, Heidelberg.

Maas, H. (1992). „Digitale Photogrammetrie in der dreidimensionalen Strömungsmesstechnik“. Diss. Eidgenössische Technische Hochschule Zürich.

Maas, H. (1997). *Mehrbildtechniken in der digitalen Photogrammetrie*. Institut für Geodäsie und Photogrammetrie an der Eidgenössischen Technischen Hochschule.

Mallot, H., M. von der Heyde und H. Bülthoff (1998). *Sehen und die Verarbeitung visueller Information: Eine Einführung*. Vieweg, Braunschweig.

Mayer, J., I. Melzer und F. Schweiggert (2003). „Lightweight Plug-In-Based Application Development“. In: *Objects, Components, Architectures, Services, and Applications for a Networked World: International Conference Netobjectdays, Node 2002, Erfurt, Germany, October 7-10, 2002: Revised Papers*.

McKenna, S., Y. Raja und S. Gong (1999). „Tracking colour objects using adaptive mixture models“. In: *Image and vision computing* 17.3, S. 225–231.

Meer, P., D. Mintz, A. Rosenfeld und D. Kim (1991). „Robust regression methods for computer vision: A review“. In: *International journal of computer vision* 6.1, S. 59–70.

Meyer-Baese, U. (2004). *Digital signal processing with field programmable gate arrays*. Springer Verlag.

Micron Technology Inc., Hrsg. (2004). *MT9M413 - 1.3-megapixel CMOS active-pixel digital image sensor*. Micron Technology Inc. URL: `http://download.micron.com/pdf/datasheets/imaging/mt9m413c36stc.pdf`.

Mitchell, H. und I. Newton (2002). „Medical photogrammetric measurement: overview and prospects". In: *ISPRS journal of photogrammetry and remote sensing* 56.5, S. 286–294.

Moeslund, T. und E. Granum (2001). „A survey of computer vision-based human motion capture". In: *Computer Vision and Image Understanding* 81.3, S. 231–268.

Molnar, I. (2000). *Linux Low Latency Patch for multimedia applications*.

Muehlmann, U., M. Ribo, P. Lang und A. Pinz (2004). „A new high speed CMOS camera for real-time tracking applications". In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Bd. 5. IEEE, S. 5195–5200.

Mulder, J., J. Jansen und A. van Rhijn (2003). „An affordable optical head tracking system for desktop VR/AR systems". In: *Proceedings of the workshop on Virtual environments 2003*. ACM, S. 215–223.

Newman, J., M. Wagner, T. Pintaric, A. MacWilliams, M. Bauer, G. Klinker und D. Schmalstieg (2003). „Fundamentals of ubiquitous tracking for augmented reality". In:

Newman, J., M. Wagner, M. Bauer, A. MacWilliams, T. Pintaric, D. Beyer, D. Pustka, F. Strasser, D. Schmalstieg und G. Klinker (2004). „Ubiquitous Tracking for Augmented Reality". In: *ISMAR '04: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*. Washington, DC, USA: IEEE Computer Society, S. 192–201.

Niederöst, M. und H. Maas (1997). „Entwurf und Erkennung von codierten Zielmarken". In: *Publikation der DGPF* 5, S. 55–62.

Norman, D. und S. Draper (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates Inc. Mahwah, NJ, USA.

Ohta, J. (2008). *Smart CMOS image sensors and applications*. CRC.

Okutomi, M. und T. Kanade (1993). „A multiple-baseline stereo". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 15.4, S. 353–363.

OpenCV, Hrsg. (2006). *The OpenCV Open Source Computer Vision Library*. OpenCV Developers. URL: `http://www.opencv.org`.

Owen, C., F. Xiao und P. Middlin (2002). „What is the best fiducial“. In: *The First IEEE International Augmented Reality Toolkit Workshop*, S. 98–105.

Panse, R. (2002). „Kalibrierung eines Multikamerasystem mit dynamischer Nachkalibrierung“. Diplomarbeit. Universität Mannheim.

Philips Semiconductors, Hrsg. (2000). *The I²C-bus specification*. Philips Semiconductors.

Pilet, J., V. Lepetit und P. Fua (2005). „Real-time nonrigid surface detection“. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Bd. 1. IEEE, S. 822–828.

Pintaric, T. und H. Kaufmann (2007). „Affordable Infrared-Optical Pose-Tracking for Virtual and Augmented Reality“. In: *Proceedings of Trends and Issues in Tracking for Virtual Environments Workshop (IEEE VR 2007)*. Charlotte, NC, USA.

Pintaric, T. und H. Kaufmann (2008). „A rigid-body target design methodology for optical pose-tracking systems“. In: *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*. ACM, S. 73–76.

Poggio, T. und C. Koch (1985). „Ill-posed problems in early vision: from computational theory to analogue networks“. In: *Proceedings of the Royal society of London. Series B. Biological sciences* 226.1244, S. 303–323.

Pollard, S., J. Mayhew und J. Frisby (1985). „PMF: a stereo correspondence algorithm using a disparity gradient limit.“ In: *Perception* 14.4, S. 449–70.

Pott, P., A. Wagner, A. Köpfle, E. Badreddin, R. Männer, P. Weiser, H.-P. Scharf und M. L. R. Schwarz (2004). „A handheld surgical manipulator: ITD - design and first results“. In: *CARS*, S. 1333.

Pott, P. P. (2007). „Untersuchung von Kinematiken für handgehaltene Roboter“. Diss. Üniversität Mannheim".

Pott, P., M. Schwarz, A. Köpfle, M. Schill, A. Wagner, E. Badreddin, R. Männer, P. Weiser und H. Scharf (2003). „ITD — A handheld manipulator for medical applications — Concept and design“. In: *3rd annual meeting of CAOS*, S. 290–291.

Poynton, C. (1998). „Rehabilitation of gamma“. In: *Photonics West'98 Electronic Imaging*. International Society for Optics und Photonics, S. 232–249.

Pratt, V. (1987). „Direct least-squares fitting of algebraic surfaces“. In: *ACM SIGGRAPH Computer Graphics*. Bd. 21. 4. ACM, S. 145–152.

Press, W., B. Flannery, S. Teukolsky, W. Vetterling et al. (1986). *Numerical recipes*. Bd. 547. Cambridge Univ Press.

Pustka, D., M. Huber, M. Bauer und G. Klinker (Okt. 2006). „Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems“. In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*.

Ramey, N., J. Corso, W. Lau, D. Burschka und G. Hager (2004). „Real-time 3D surface tracking and its applications“. In: *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*. IEEE, S. 34.

Reitmayr, G. und D. Schmalstieg (2001). „An open software architecture for virtual reality interaction“. In: *Proceedings of the ACM symposium on Virtual reality software and technology*, S. 47–54.

Reitmayr, G. und D. Schmalstieg (2005). „Opentracker - a flexible software design for three-dimensional interaction“. In: *Virtual Reality* 9.1, S. 79–92.

Ribo, M. (2001). *State of the Art Report on Optical Tracking*. Techn. Ber. Vienna Univ. Technol., Vienna, Austria.

Ribo, M., A. Pinz und A. Fuhrmann (2001). „A new optical tracking system for virtual and augmented realityapplications“. In: *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*. Bd. 3.

Ribo, M., M. Brandner und A. Pinz (2004). „A flexible software architecture for hybrid tracking“. In: *Journal of Robotic Systems* 21.2, S. 53–62.

Rinner, B. und W. Wolf (2008). „An introduction to distributed smart cameras“. In: *Proceedings of the IEEE* 96.10, S. 1565–1575.

Ripoll, I. (2002). „RTLinux versus RTAI“. In: *www. linux devices. com*.

Riviere, C. und P. Khosla (1999). „Microscale Tracking of Surgical Instrument Motion“. In: *Proceedings of the Second International Conference on Medical Image Computing and Computer-Assisted Intervention* 1679, S. 1080–1087.

Rosenfeld, A. und J. Pfaltz (1966). „Sequential operations in digital picture processing“. In: *Journal of the ACM (JACM)* 13.4, S. 471–494.

Ruf, T. (2000). „Entwurf, Aufbau und Evaluierung eines FPGA-basierten Farbmarker-Trackings für den Augenoperations-Simulator EyeSi2“. Diplomarbeit. Universität Mannheim.

Russell M. Taylor, I., T. C. Hudson, A. Seeger, H. Weber, J. Juliano und A. T. Helser (2001). „VRPN: a device-independent, network-transparent VR peripheral system“. In: *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM Press, S. 55–61.

Sauer, F., A. Khamene und S. Vogt (2002). „An augmented reality navigation system with a single-camera tracker: System design and needle biopsy phantom trial“. In: *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2002*, S. 116–124.

Scharstein, D. und R. Szeliski (2002). „A taxonomy and evaluation of dense two-frame stereo correspondence algorithms“. In: *International journal of computer vision* 47.1, S. 7–42.

Schill, M., C. Wagner, M. Hennen, H. Bender und R. Männer (1999). „EyeSi-A Simulator for Intra-ocular Surgery“. In: *Proceedings of the Second International Conference on Medical Image Computing and Computer-Assisted Intervention*, S. 1166–1174.

Schill, M. A. (2001). „Biomechanical Soft Tissue Modeling – Techniques, Implementation and Applications“. Diss. Universität Mannheim.

Schlegel, M. (Juli 2006). „Predicting the Accuracy of Optical Tracking Systems“. Magisterarb. TU München.

Schlessman, J., C. Chen, W. Wolf, B. Ozer, K. Fujino und K. Itoh (2006). „Hardware/software co-design of an FPGA-based embedded tracking system“. In: *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*. IEEE.

Schmalstieg, D., A. Fuhrmann, G. Hesina, Z. Szalavári, L. Encarnaçao, M. Gervautz und W. Purgathofer (2002). „The studierstube augmented reality project“. In: *Presence: Teleoperators & Virtual Environments* 11.1, S. 33–54.

Schmidt, D., M. Stal, H. Rohnert und F. Buschmann (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley Series in Software Design Patterns. Chichester [etc.] : John Wiley & Sons.

Schuppe, O., C. Wagner, F. Koch und R. Männer (2009). „EYESi ophthalmoscope-a simulator for indirect ophthalmoscopic examinations." In: *Studies in health technology and informatics* 142, S. 295.

Schwarte, R., H. Heinol, B. Buxbaum, T. Ringbeck, Z. Xu und K. Hartmann (1999). „Principles of Three-Dimensional Imaging Techniques". In: *Computer Vision and Applications* 18. Hrsg. von K. Jähne Haußecker und Geißler (1999b), S. 463–484.

Seitz, P. (2000). „Solid-State Image Sensing". In: *Computer Vision and Applications*. Hrsg. von J. und Haussecker. The Academic Press, S. 111–152.

Seitz, S., B. Curless, J. Diebel, D. Scharstein und R. Szeliski (2006). „A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Bd. 1. IEEE, S. 519–528.

Shafer, S. (1985). „Using color to separate reflection components". In: *Color Research & Application* 10.4, S. 210–218.

Shapiro, L. und G. Stockman (2001). *Computer Vision. 2001*. Prentice Hall.

Shi, J. und C. Tomasi (1994). „Good features to track". In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, S. 593–600.

Shortis, M., T. Clarke und T. Short (1994). „Comparison of some techniques for the subpixel location of discrete target images". In: *Photonics for Industrial Applications*. International Society for Optics und Photonics, S. 239–250.

Sielhorst, T., T. Obst, R. Burgkart, R. Riener und N. Navab (2004). „An augmented reality delivery simulator for medical training". In: *International Workshop on Augmented Environments for Medical Imaging-MICCAI Satellite Workshop*. Citeseer, S. 11–20.

SMProcess, Hrsg. (2011). *FxEngine Framework User Guide*. SMProcess. url: `http://www.smprocess.com`.

Snyder, W. und H. Qi (2010). *Machine vision*. Cambridge Univ Pr.

Soille, P. (2003). *Morphological Image Analysis: Principles and Applications*. 2. Aufl. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Spectrum Digital Inc., Hrsg. (2006). *TMS320C6455, Technical Reference, DSK*. Spectrum Digital Inc., Stafford, TX.

Steger, C., M. Ulrich und C. Wiedemann (2008). *Machine vision algorithms and applications*. Vch Verlagsgesellschaft Mbh.

Steinicke, F., C. Jansen, K. Hinrichs, J. Vahrenhold und B. Schwald (2007). „Generating Optimized Marker-based Rigid Bodies for Optical Tracking Systems". In: *2nd International Conference on Computer Vision Theory and Applications (VISAPP)*.

Strauss, P. S. (1993). „IRIS Inventor, a 3D graphics toolkit". In: *SIGPLAN Not.* 28.10, S. 192–200.

Stroian, G., T. Falco und J. Seuntjens (2004). „Elimination of ghost markers during dual sensor-based infrared tracking of multiple individual reflective markers". In: *Medical Physics* 31, S. 2008–2019.

Stroustrup, B. (1997). *The C++ Programming Languge*. Addison-Wesley Publishing Company.

Suzuki, K., I. Horiba und N. Sugie (2003). „Linear-time connected-component labeling based on sequential local operations". In: *Computer Vision and Image Understanding* 89.1, S. 1–23.

Triggs, B., P. McLauchlan, R. Hartley und A. Fitzgibbon (2000). „Bundle adjustment—a modern synthesis". In: *Vision algorithms: theory and practice*, S. 153–177.

Tsai, R. Y. (1987). „A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses". In: *IEEE Journal of Robotics and Automation* RA-3.4, S. 323–344.

Umeyama, S. (1991). „Least-squares estimation of transformation parameters between two point patterns". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 13.4, S. 376–380.

USB Implementers Forum, Inc., Hrsg. (2000). *Universal Serial Bus specification 2.0*. USB Implementers Forum, Inc.

Vezhnevets, V., V. Sazonov und A. Andreeva (2003). „A survey on pixel-based skin color detection techniques“. In: *Proc. Graphicon*. Bd. 3. Moscow, Russia, S. 85–92.

Vincent, L. (1994). „Morphological area openings and closings for grey-scale images“. In: *Nato Asi Series F Computer and Systems Sciences* 126, S. 197–197.

Wagner, A., E. Badreddin, P. Pott, M. Schwarz, H.-P. Scharf, A. Köpfle, R. Männer und P. Weiser (2004). „System Design and Position Control of a Handheld Surgical Robotic Device“. In: *Conf. On Mechatronics And Robotics*. Aachen, Germany, S. 1415–1420.

Wagner, C., M. Schill und R. Männer (2002). „Intraocular surgery on a virtual eye“. In: *Communications of the ACM* 45.7, S. 45–49.

Wagner, C. (2003). „Virtuelle Realitäten für die chirurgische Ausbildung: Strukturen, Algorithmen und ihre Anwendung“. Diss. Universität Mannheim.

Wang, X., J. Gao und L. Wang (2004). „A survey of subpixel object localization for image measurement“. In: *Information Acquisition, 2004. Proceedings. International Conference on*. IEEE, S. 398–401.

Welch, G. und G. Bishop (1995). „An introduction to the Kalman filter“. In: *University of North Carolina at Chapel Hill, Chapel Hill, NC* 7.1.

Welsh, M., A. Basu und T. von Eicken (1996). „Low-Latency Communication over Fast Ethernet“. In: *Proceedings of EUROPAR* 96.

Wiles, A., D. Thompson und D. Frantz (2004). „Accuracy assessment and interpretation for optical tracking systems“. In: *Medical Imaging 2004*. International Society for Optics und Photonics, S. 421–432.

Wiora, G., P. Babrou und R. Männer (2004). „Real time high speed measurement of photogrammetric targets“. In: *Pattern Recognition*, S. 562–569.

Wollnack, J. (2005). *Dreidimensional messende videometrische Messsysteme*. Techn. Ber. Technische Universität Hamburg-Harburg.

Wu, K., E. Otoo und K. Suzuki (2009). „Optimizing two-pass connected-component labeling algorithms“. In: *Pattern Analysis & Applications* 12.2, S. 117–135.

Xilinx Inc., Hrsg. (2009). *Spartan-3 FPGA Family: Complete Data Sheet*. Xilinx Inc. URL: `http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf`.

Yang, Y. (2012). „Design and Implementation of a Scalable Hardware Platform for High Speed Optical Tracking“. noch nicht veröffentlicht. Diss. University of Heidelberg.

Zhang, X., S. Fronz und N. Navab (2002). „Visual Marker Detection and Decoding in AR Systems: A Comparative Study.“ In: *ISMAR'02*, S. 97–106.

Zhang, Z., R. Deriche, O. Faugeras und Q. Luong (1995). „A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry“. In: *Artificial Intelligence* 78.1-2, S. 87–119.

Zhang, Z. (1998). *A Flexible New Technique for Camera Calibration*. Techn. Ber. Microsoft Research.

Zhang, Z. (Juli 2004). „Camera calibration with one-dimensional objects“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.7, S. 892–899.

Zhou, H. und H. Hu (2004). „A survey – human movement tracking and stroke rehabilitation“. In: *Department report CSM-420, University of Essex, England, ISSN*, S. 1744–8050.