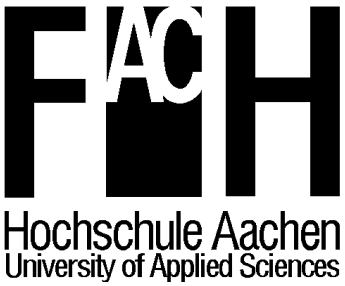

Fachhochschule Aachen

Fachbereich
Maschinenbau und Mechatronik

Mechatronik

Diplomarbeit

Ein Kamerasystem zur Fusion von 3D-Abstandsinformationen mit digitalen Farbbildern

Kai Pervölz

Matr.-Nr.: 171126

Referent : Prof. Dr. Klaus-Peter Kämper
Korreferent : Dr.-Ing. Hartmut Surmann

In Zusammenarbeit mit

Fraunhofer-Institut für
Autonome Intelligente Systeme (AIS)
Schloß Birlinghoven
53754 Sankt Augustin

Erklärung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, im August 2003

Diese Arbeit wurde am Fraunhofer-Institut für Autonome Intelligente Systeme in Zusammenarbeit mit dem Fachbereich Maschinenbau und Mechatronik der Fachhochschule Aachen angefertigt.

Danksagung

Herrn Prof. Dr. Klaus-Peter Kämper möchte ich für die Betreuung dieser Diplomarbeit danken.

Herzlich danken möchte ich auch Herrn Dr.-Ing. Hartmut Surmann für die Betreuung dieser Arbeit und seine große Bereitschaft meine Fragen zu beantworten. Durch ihn habe ich sehr interessante Einblicke in den Bereich der autonomen mobilen Robotik erhalten. Desweiteren möchte ich ihm danken für die bereitwillige Übernahme des Korreferates.

Mein herzlicher Dank gilt auch Dipl. Inform. Andreas Nüchter, der diese Arbeit mit wertvollen Anregungen und Diskussionen begleitet hat und stets zur Beantwortung meiner Fragen bereit war.

Ein weiterer Dank gilt meinen Eltern, die mir dieses Studium erst ermöglicht haben und deren Unterstützung ich mir stets sicher sein konnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Ein Bedürfnis des Menschen ist es, seine Umgebung oder Teile daraus so realistisch wie möglich darzustellen. Einerseits wird dadurch die Gegenwart für zukünftige Generationen dokumentiert und weitergegeben, andererseits werden Informationen, Wissen und Erfahrungen veranschaulicht. Die Art der Darstellung dieser Informationen hängt immer von den zur Verfügung stehenden Mitteln ab. Dies beginnt bei den Höhlenmalereien der Steinzeit über die Erstellung von Skizzen und Zeichnungen, die Fotographie und das Filmen bis hin zur Darstellung mit Hilfe moderner Computer-Medien. Die letzteren bieten die Möglichkeit, verschiedene Formen der Wahrnehmung des Menschen gleichzeitig anzusprechen z.B. durch Geräusche, bewegte Bilder und Text. Dadurch wird es dem Einzelnen erleichtert, die Darstellung der Informationen zu verstehen.

Nicht zuletzt durch die Computerspiele-Industrie mit immer realistischeren Szenarien und der Filmindustrie mit am Computer entstehenden „special-Effects“ wurde die Technologie zur Darstellung von 3D-Szenarien am Computer in den letzten Jahren sowohl im Software- als auch im Hardware-Bereich sehr weit vorangetrieben. Durch Technologien wie Head-Mounted-Displays, 3D-Simulatoren mit Surround-Sound und Positionssensoren ist es möglich, dem Benutzer nicht mehr nur eine sehr realistische Darstellung zu bieten, sondern ihn vielmehr Teil der virtuellen Szene werden zu lassen.

Um eine reale Szene in dieser Weise virtuell darstellen zu können, ist es zunächst einmal notwendig, Daten wie Enfernungen, Farben und Helligkeit der darzustellenden Szene zu erfassen. Dies kann im Wesentlichen auf zwei verschiedene Arten geschehen. Die erste ist die manuelle Erfassung der Daten und anschließende Digitalisierung mit Hilfe von CAD-Software und entsprechend geschultem Personal. Der Aufwand für die Modellierung ist dabei um ca. 3-10 mal höher als die eigentliche Datenerfassung. Die zweite Methode ist die automatische Erfassung aller Daten mit Hilfe von autonomen mobilen Robotern und unterschiedlichen Sensoren, sowie die automatische dreidimensionale Modellierung. Diese Methode benötigt sehr komplexe Hard- und Software, ist aber mit geringem Personalaufwand möglich. [10]

Das autonome Robotersystem KURT3D mit seinem 3D-Laserscanner ermöglicht die autonome Erfassung der 3D-Abstandsinformationen einer beliebigen Szene. Um diese Daten als realistische virtuelle Szene darzustellen, müssen sie mit Texturen „beklebt“ werden. Ziel dieser Arbeit ist daher die Entwicklung und Realisierung eines Kamerasystems zur Erfassung der hierfür notwendigen Bilddaten und die Implementierung verschiedener Softwaremodule zur automatischen Fusion der Bilddaten mit den 3D-Abstandsinformationen des 3D-Laserscanners. Die Vorgehensweise bei der Realisierung des Kamerasystems und der Implementierung der Softwaremodule wird in den nachfolgenden Kapiteln dargestellt.

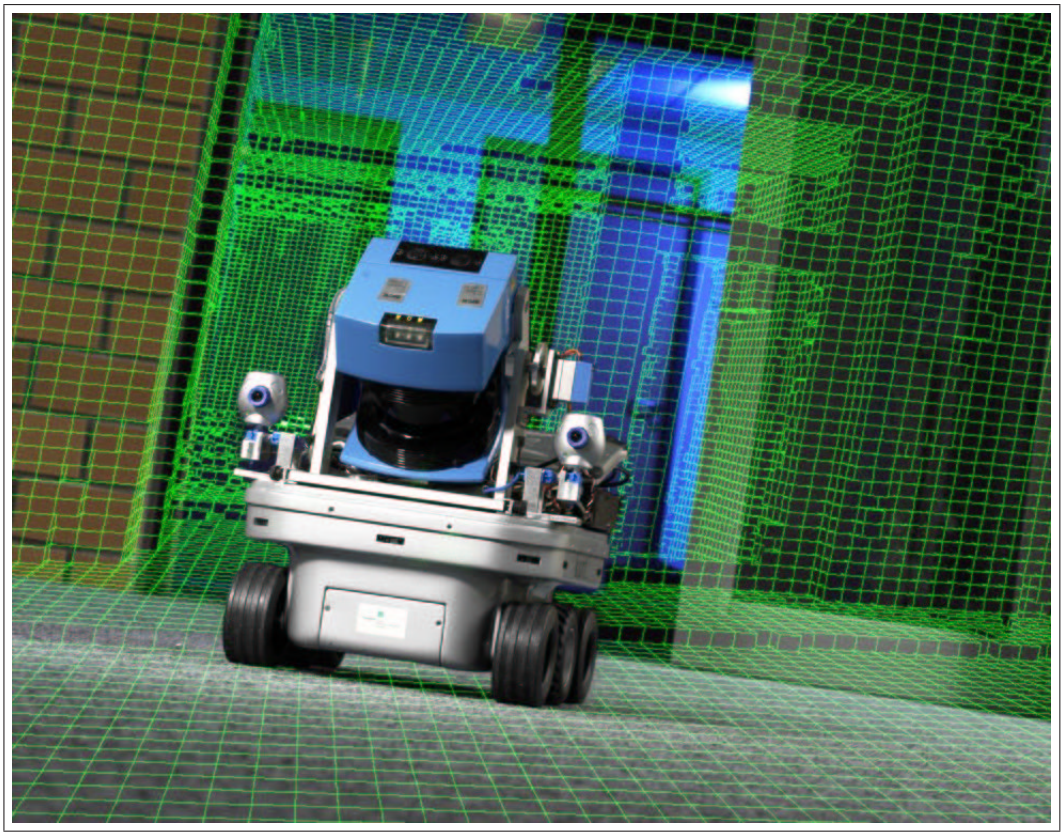

Abbildung 1.1: Robotersystem KURT3D

1.1 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in 7 Kapitel:

Kapitel 1 ist die vorliegende Einleitung.

Kapitel 2 beschreibt den autonomen mobilen Roboter KURT3D und stellt verschiedene mögliche Konzepte der Integration eines Kamerasystems in ein Robotersystem vor.

Kapitel 3 stellt die Entwicklung eines Kamerasystems zur Gewinnung von Texturdaten vor. Zunächst werden die an das System zu stellenden Anforderungen dargestellt und anschließend deren Umsetzung in Konzeption, Konstruktion, Fertigung und Integration gezeigt.

Kapitel 4 beschäftigt sich mit der Ansteuerung der Schwenk-Nick-Vorrichtung des Kamerasystems. Das Betriebssystem Real-Time Linux und seine Grundfunktionen werden vorgestellt, sowie das zur Ansteuerung des Kamerasystems verwendete Modul `rt_servo` erläutert. Anschließend wird noch eine alternative Möglichkeit der Ansteuerung mittels zusätzlichem Mikrokontroller und Kommunikation über die serielle Schnittstelle aufgezeigt.

Kapitel 5 erläutert die zur Fusion der Texturdaten des Kamerasystems mit den 3D-Abstandsinformationen des Laserscanners notwendigen Grundlagen der Photogrammetrie. Desweiteren werden verschiedene Verfahren zur Kalibrierung von Kamerasystemen vorgestellt.

Kapitel 6 beschreibt das Verfahren zur Fusion der digitalen Farbbilder des Kamerasystems mit den 3D-Abstandsinformationen des Laserscanners. Sowohl das Verfahren zur Kamerakalibrierung und seine Implementierung in der Klasse `camera` werden erläutert als auch die Erzeugung von Texturen mit OpenGL und ihre Umsetzung beim Rendering-Programm des Laserscanners werden beschrieben. Anhand von Beispielen wird die Erzeugung von textuierten 3D-Szenen veranschaulicht.

Kapitel 7 ist die Zusammenfassung der Arbeit. Die Ergebnisse der Arbeit werden abschließend zusammengefaßt und ein Auslick auf zukünftige Arbeiten gegeben.

Kapitel 2

Mobile Robotik

In diesem Kapitel wird die mobile Roboterplattform KURT3D näher beschrieben und Möglichkeiten zur Integration einer Kamera in ein Robotersystem vorgestellt.

2.1 Die Roboterplattform KURT3D

Die in dieser Arbeit zur Fusion mit den Kamerabildern verwendeten 3D-Umgebungsdaten stammen von dem Laserscanner der Roboterplattform KURT3D [15], einer speziellen Version der **K**anal-**U**ntersuchungs-**R**oboter-**T**estplattform KURT2 [38]. Dieser autonome mobile Roboter dient zur Entwicklung bzw. Erprobung von Sensorik und Software für autonome Fahrzeuge. Im folgenden wird KURT3D näher vorgestellt.

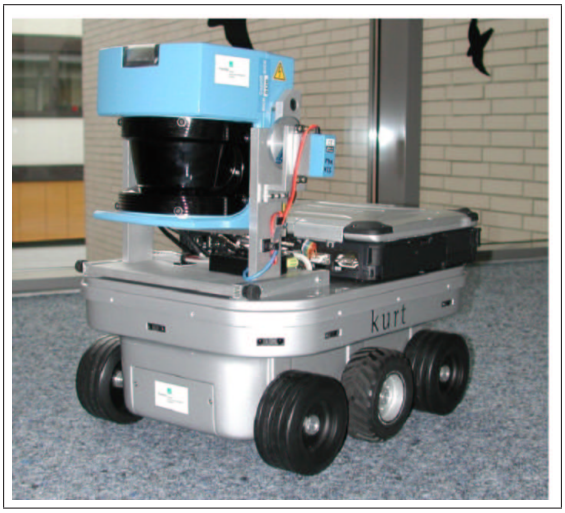

Abbildung 2.1: Die Roboterplattform KURT3D.

2.1.1 Technische Daten

KURT3D (Abbildung 2.1) ist ein 3-achsiger, autonom fahrender Roboter mit den Maßen (HxBxL) 47 cm x 26 cm x 33 cm und einem Gesamtgewicht von 22,6 kg. Durch die Anordnung der 6 Räder entspricht das Fahrverhalten dem eines Kettenfahrzeuges. Um Reibungsverluste bei Kurvenfahrten zu minimieren, sind nur die beiden mittleren Räder mit Profilrillen versehen. Die Steuerung des Roboters ist auf zwei Ebenen verteilt, wobei ein 16-Bit CMOS Mikrokontroller (Phytec Minimodul C167) [2] mit Flash Speicher die untere Ebene darstellt und für die Ansteuerung der zwei 90 Watt Maxon-Motoren und das Auslesen der HP-Winkelencoder der Motoren zuständig ist [38]. Durch die Verwendung von 1:14 Getrieben ist eine maximale Geschwindigkeit von 5,4m/s möglich, wobei die Winkelencoder 4266 Werte pro Radumdrehung liefern. Ein „toughbook“ der Firma Panasonic mit einem 600Mhz Pentium-3 Prozessor, 384MB RAM und Real-Time Linux als Betriebssystem, stellt die obere Ebene dar [15, 38]. Diese übernimmt die globale Planung bzw. Steuerung des Roboters und ist auch für die Auswertung des 3D-Laserscanners (vgl. Kapitel 2.1.2) zuständig. Die gesamte Spannungsversorgung des Robotersystems besteht aus 52 NiMH Zellen. Davon 28 Zellen für den KURT3D, 20 Zellen für den Laserscanner und 4 Zellen für die Spannungsversorgung des Servomotors. Die maximale Betriebsdauer des Systems ist begrenzt durch Akkukapazität des Panasonic „toughbooks“ und beträgt zur Zeit ca. 4 Stunden [15].

Motoren	2 x 90W Maxon
Spannungsversorgung	NiMH Akkus (1,2V pro Zelle)
Höchstgeschwindigkeit	5,4m/s
Hauptsensor	3D-Laserscanner, Winkelencoder
Höhe	47cm
Breite	26cm
Länge	33cm
Gesamtgewicht	22,6kg

Tabelle 2.1: Technische Daten KURT3D.

2.1.2 3D-Laserscanner

Der von KURT3D verwendete Laserscanner (vgl. Abbildung 2.2) wurde speziell für den Einsatz in der mobilen Robotik entwickelt und ist in der Lage, die Umgebung des Roboters dreidimensional zu erfassen. Ein aus der Sicherheitstechnik stammender Standard 2D-Laserscanner (Sick LMS) wurde so modifiziert, daß er mit Hilfe eines Servomotors um die horizontale Achse gedreht werden kann. Dieser zusätzliche Freiheitsgrad ermöglicht es, in verschiedenen Ebenen 2D-Laserscans durchzuführen, welche dann zu einem dreidimensionalen Bild der Umgebung zusammengefügt werden können. Die Auflösung in horizontaler Richtung wird durch den 2D-Laserscanner bestimmt und liegt bei maximal 721 Punkten bei 180°. In vertikaler Richtung

wird sie durch den Servomotor bestimmt und liegt bei 420 Linienscans bei 130°. Die Genauigkeit der Tiefeninformationen hängt von dem verwendeten 2D-Laserscanner (hier: Sick LMS, 1cm) ab. Über ein serielles Interface werden die Daten des Scanners direkt an das „toughbook“ übertragen. Zur Analyse der Daten stehen verschiedene Software-Module zur Verfügung und bereits während des Scans werden Online-Algorithmen zur Linien- und Flächenerkennung eingesetzt. Ausführlichere Beschreibungen der Hard- und Software des 3D-Laserscanners finden sich in [36, 35, 30].

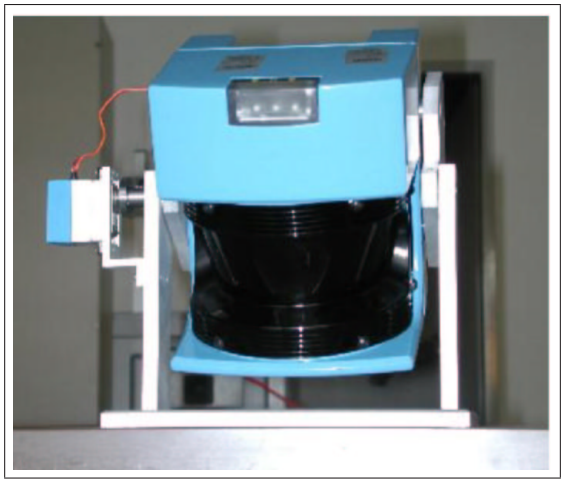

Abbildung 2.2: Der 3D-Laserscanner.

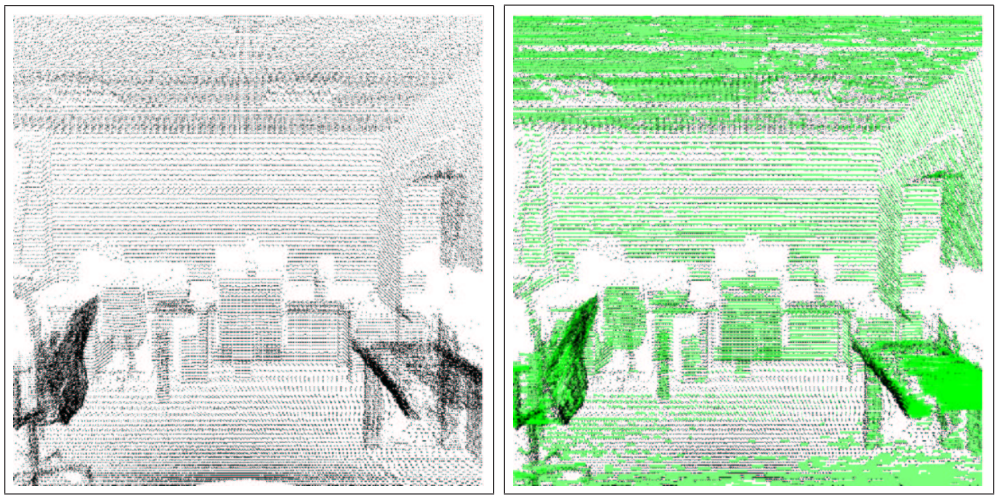

Abbildung 2.3: Meßpunkte (linkes Bild) und grün gekennzeichnete detektierte Linien eines Laserscans.

2.2 Kamerasysteme in der mobilen Robotik

Der aktuelle Stand der Technik bietet verschiedene Möglichkeiten ein Kamerasystem in einen mobilen Roboter mit PC-Steuerung zu integrieren. Im folgenden werden einige dieser Möglichkeiten beschrieben.

Framegrabber-Karten

Diese Variante ist die klassische Methode um Kamerabilder mit einem PC zu verarbeiten. Normale Videokameras liefern ein analoges Videosignal, welches in dieser Form nicht direkt von einem PC verarbeitet werden kann. Framegrabber-Karten sind zusätzliche Steckkarten, die in der Lage sind, diese analogen Signale zu digitalisieren und dem PC zur Verfügung zu stellen. Somit ist es möglich, fast jede herkömmliche Videokamera an einen PC anzuschliessen.

Firewire Kameras

Nach der Einführung von seriellen Bus-Standards mit hoher Übertragungsrate in der PC-Technik wurden Kameras entwickelt, die direkt digitale Signale zur Verfügung stellen und ohne zusätzliche Hardware mit dem PC verbunden werden können. Einer dieser Standards ist der IEEE-1394 oder auch Firewire genannt. Wegen seiner hohen Übertragungsrate (100-400 Mbits/sek.) wird er sehr oft für Multimedia-Anwendungen mit großen Datenmengen, wie zum Beispiel digitalen Videokameras mit großer Auflösung und hoher Bildrate, eingesetzt.

USB Kameras

Beim **U**niversal **S**erial **B**us (siehe Anhang A.1) handelt es sich ebenfalls um einen seriellen Bus mit hoher Übertragungsrate. Der USB 1.1 Standard bietet eine Geschwindigkeit von 1,5 bis 12Mbits/sek. und wird vor allem für Videokameras mit kleiner und mittlerer Übertragungsrate oder digitale Fotokameras verwendet. Mit der Einführung des USB 2.0 Standards wurde die Übertragungsrate auf 480Mbits/sek. erhöht und somit der Einsatz von USB auch bei digitalen Videogeräten mit großen Datenmengen ermöglicht.

	Framegrabber-Karten	Firewire Kameras	USB 1.1 Kameras
Videoqualität	hoch - sehr hoch	hoch	mittel
Einzelbildqualität	hoch	hoch	hoch
Schnittstelle	PCI, AGP, PCMCIA	IEEE-1394	USB
Eigenschaften	hohe Bildqualität etablierte Technik höhere Kosten extra Hardware notwendig	keine extra Hardware notwendig gute Videoqualität IEEE-1394 Schnittstelle erst bei neueren PC'S Standard	geringe Kosten USB Schnittstelle ist Standard bei PC's keine extra Hardware notwendig geringe Datenübertragung

Tabelle 2.2: Gegenüberstellung der verschiedenen Schnittstellen.

Kapitel 3

Konstruktion und Integration des Kamerasystems

Dieses Kapitel beschäftigt sich mit der Konzeption und Realisierung der Hardware für ein robotergestütztes Kamerasystem. Die an das System gestellten Anforderungen werden dargestellt, sowie deren Umsetzung in Planung, Konstruktion und Fertigung gezeigt. Anschließend werden die zur Integration notwendigen Erweiterungen der auf dem Robotersystem KURT3D vorhandenen Elektronik beschrieben.

3.1 Entwurf eines geeigneten Kamerasystems

Zunächst muß eine geeignete Kamera für das System gewählt werden. Die folgende Aufzählung zeigt die bei der Wahl einer Kamera für einen mobilen Roboter entscheidenden Kriterien:

- maximale Auflösung
- Bildrate
- Schnittstelle (USB, Firewire etc.)
- Betriebssystem-Unterstützung
- optischer Zoom (ja, nein)
- Autofocus (ja, nein)
- maximale und durchschnittliche Stromaufnahme
- Gewicht, Abmaße
- Preis

Unter Beachtung der zur Verfügung stehenden Schnittstellen ist eine USB-Webcam aus dem PC-Zubehör eine gute und kostengünstige Lösung. Hierbei muß allerdings darauf geachtet werden, daß der von der Kamera verwendete Chipsatz auch von dem hier verwendeten Linux Betriebssystem (Real-Time Linux auf SuSE 8.0 Basis) unterstützt wird. Ein weit verbreiteter und ab

der Kernelversion 2.2.16 unterstützter Chipsatz ist der OV511 [3]. Dieser wird auch von der schließlich für dieses Projekt ausgewählten Webcam, eine TerraCAM USB Pro (Abbildung 3.1) der Firma TerraTec [18], benutzt. Die maximale Auflösung der Kamera beträgt 640x480 Pixel und die Bildrate 7 Bilder/Sekunde. Sie verfügt über einen manuellen Focus und besitzt keinen optischen Zoom.

Abbildung 3.1: TerraCam USB Pro der Firma Terratec.

Um den durch den Öffnungswinkel der Kamera eingeschränkten Bildbereich zu vergrößern und so eine flexiblere Erfassung der Umgebung zu ermöglichen, ohne die Position des Roboters verändern zu müssen, soll das System um eine Schwenk-Nick-Vorrichtung erweitert werden. Als Antrieb für die Schwenk- und Nick-Achse bieten sich Modellbau-Servos an. Tabelle 3.1 zeigt die technischen Daten der „Micro-Maxx“ Servos der Firma Volz [39]. Mit ihren geringen Abmaßen, einer Stellkraft von 48 Ncm und einem Gewicht von 19g pro Servo erfüllen sie alle Andorderungen für dieses Projekt. Das verwendete Metallgetriebe gewährleistet eine hohe Wiederholgenauigkeit der Servostellung.

Getriebe	Metallgetriebe
Lager	Kugellager
Gewicht	19g
Maße	33 x 34 x 16,5mm
Spannung	4,8 / 6V
Drehmoment	40 / 48 Ncm
Stellzeit(40°)	0,12 / 0,09s
Stellweg	2 x 45°

Tabelle 3.1: Technische Daten des Modellbauservos Micro-Maxx der Firma Volz [39].

3.1.1 Anforderungen an die Schwenk-Nick-Vorrichtung

Durch die speziellen Bedingungen in der mobilen Robotik werden verschiedene Anforderungen an jedes verwendete Gerät gestellt:

- geringe Stromaufnahme
- geringe Verlustleistung
- geringes Gewicht
- hohe Zuverlässigkeit
- geringe Wartungsintervalle

Beim späteren Einsatz des Roboters mit Kamerasystem und 3D-Laserscanner werden verschiedene Situationen die Kooperation der beiden Sensoren und die Verknüpfung ihrer Daten erfordern. Aus diesem Grund müssen sie in der Lage sein, einen möglichst identischen Bereich der Umgebung zu erfassen. Daraus ergibt sich für die Schwenk-Nick-Vorrichtung, unter Beachtung des Öffnungswinkels der Kamera [18], ein Schwenk- und Nickwinkel von 90°.

3.2 Konstruktion und Fertigung

Für die Konstruktion der mechanischen Hardware wird das 3D CAD Tool „Solidworks“ [33] (Abbildung 3.2) verwendet. Durch dreidimensionale Bewegungssimulationen können eventuelle Konstruktionsfehler bereits vor der Fertigung erkannt und korrigiert werden. Hierzu werden Funktionsmodelle der Webcam und der Servos erstellt und später für die Simulation in die Baugruppen eingebunden.

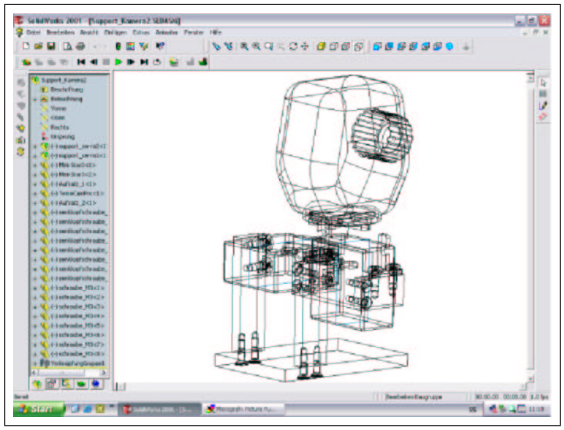

Abbildung 3.2: Screenshot des 3D CAD Tools „Solidworks“.

Die Schwenk-Nick-Vorrichtung selber besteht aus 2 Halterungen für die Servos, wobei die erste Halterung aus der Bodenplatte des gesamten Systems und einer Horizontalhalterung für

den Nick-Servo besteht. An diesem Servo wird die Halterung des vertikal montierten Schwenk-Servos befestigt. Die Kamera wird direkt auf dem Stellhebel des Schwenk-Servos montiert (vgl. Abbildung 3.4). Für alle Bauteile wurde die Aluminium-Legierung AlMg4,5Mn verwendet. Die Fertigung der Bauteile erfolgte mit einer Bohr-Fräsmaschine der Firma Optimum (Abbildung 3.3).

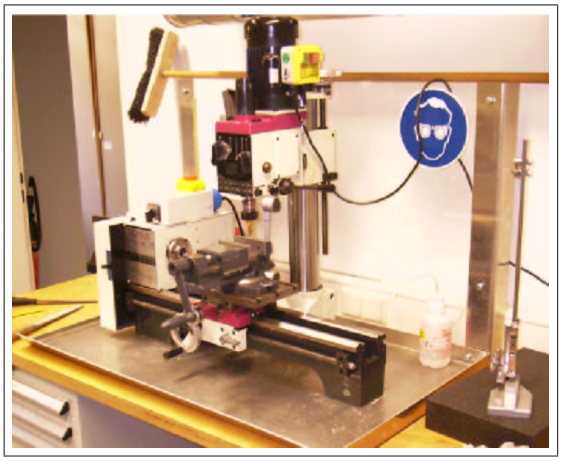

Abbildung 3.3: Die Bohr- und Fräsmaschine der Firma Optimum.

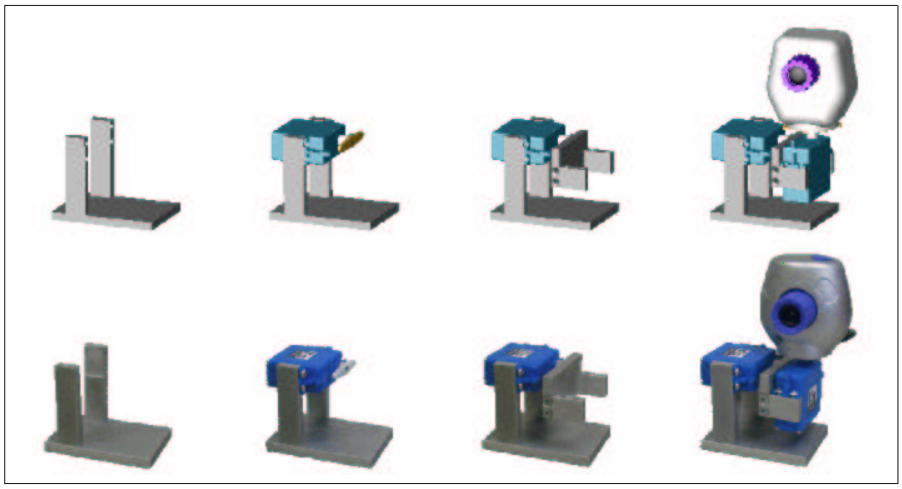

Abbildung 3.4: Vergleich zwischen den 3D „Solidworks“ Modellen (oben) und den gefertigten und montierten Bauteilen.

3.3 Aufbau der Steuerungselektronik

Um das Kamerasystem in die Steuerung von KURT3D zu integrieren, muß die Elektronik erweitert werden. Für die Steuerung der Servos werden pro Kamerasystem drei Leitungen des Parallelportes benötigt, zwei Leitungen für die PWM-Signale der Servos und eine Masseleitung. Die Spannungsversorgung des Systems soll über einen 4,8V-Akkumulator geschehen, wobei eine zusätzliche Anforderung ist, daß das System bei Bedarf per Software von der Spannungsversorgung getrennt werden kann. Dies wird durch eine TTL-Signal gesteuerte Power-Switch-Schaltung realisiert, welche direkt über den Parallelport des „toughbook“ angesteuert werden kann. Wichtig bei dieser Schaltung ist eine möglichst geringe Verlustleistung und ein kleiner Übergangswiderstand in eingeschaltetem Zustand. Zum Aufbau dieser Schaltung wird ein „BTS555“, ein Smart Highside High Current Power Switch der Firma Infineon [23], verwendet. Tabelle 3.2 zeigt die wichtigsten technischen Daten des Bauteils.

Betriebsspannung	5.0...34V
Innenwiderstand (On-State)	2.5 mOhm
Standby Strom	25μA
Schaltzeit (on)	120...600 μs
Schaltzeit (off)	50...200 μs

Tabelle 3.2: Technische Daten des Power-Switch „BTS555“.

Der Power Switch „BTS555“ schaltet bei einem „low“ Signal am Eingang „IN“ den Pin „VBB“ auf die beiden Pins „OUT1“ und „OUT2“. Um zu gewährleisten, daß das Kamerasystem bei ausgeschaltetem Roboter oder einer Störung in der Verbindung zum Parallelport ausgeschaltet ist, wird die Schaltung noch um einen NPN-Transistor und einen Widerstand erweitert. Diese sorgen dafür, daß der Eingang „IN“ des „BTS555“ nur bei einem positiven Signal am Parallelport mit „GND“ verbunden ist und durchschaltet.

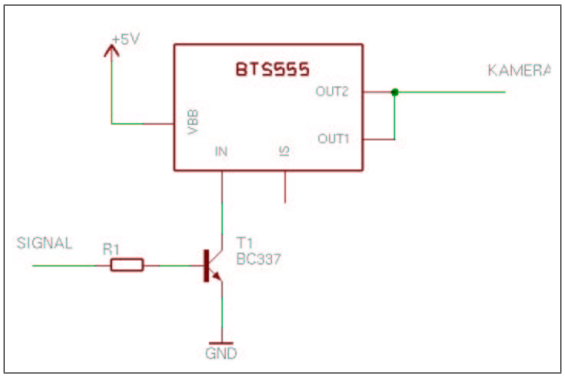

Abbildung 3.5: Schaltplan der Power-Switch-Schaltung.

Kapitel 4

Echtzeitsteuerung und Benutzer-Interface

Das folgende Kapitel beschreibt die Ansteuerung der Schwenk-Nick-Vorrichtung des Kamerasystems. Es werden sowohl das Betriebssystem Real-Time Linux und seine Grundfunktionen als auch das Benutzer-Interface zur Kamerasteuerung vorgestellt.

4.1 Real-Time Linux

Eine klare Einführung in Real-Time Linux geben H. Surmann, K. Lingemann, A. Nüchter und J. Hertzberg in „Aufbau eines 3D–Laserscanners für autonome mobile Roboter“ [36], die hier wiedergegeben werden soll. Die dort vorgestellten Module werden angepaßt und erweitert.

„Real-Time Linux ist ein Betriebssystem, bei dem ein kleiner Echtzeitkernel und der Linux-Kernel nebeneinander existieren. Es soll erreicht werden, daß die hochentwickelten Aufgaben eines Standardbetriebssystems für Anwendungen zur Verfügung gestellt werden, ohne auf Echtzeitaufgaben mit definierten, kleinen Latenzen zu verzichten. Bisher waren Echtzeitbetriebssysteme kleine, einfache Programme, die nur wenige Bibliotheken dem Anwender zur Verfügung gestellt haben. Mittlerweile ist es aber notwendig geworden, daß auch Echtzeitbetriebssysteme umfangreiche Netzwerk-Bibliotheken, graphische Benutzeroberflächen, Datei- und Datenmanipulationsroutinen zur Verfügung stellen müssen. Eine Möglichkeit ist es, diese Features zu den existierenden Echtzeitbetriebssystemen hinzuzufügen. Dies wurde beispielsweise bei den Betriebssystemen VXworks und QNX gemacht. Eine andere Herangehensweise ist, einen bestehenden Betriebssystemkernel dahingehend zu modifizieren, daß dieser Echtzeitaufgaben bewältigt. Ein solcher Ansatz wird von den Entwicklern von RT-IX (Modcomp) verfolgt. RT-IX ist eine UNIX System V Implementation, die auch harte Echtzeitaufgaben bewältigen kann [25]. Real-Time Linux basiert auf einer dritten Herangehensweise. Ein kleiner Echtzeitkernel läßt ein

Nicht-Echtzeitbetriebssystem als einen Prozeß mit niedrigster Priorität laufen. Dabei wird eine virtuelle Maschine für das eingebettete Betriebssystem benutzt. Real-Time Linux behandelt alle Interrupts zuerst und gibt diese dann an den Linux-Betriebssystem-Prozeß weiter, wenn diese Ereignisse nicht von Echtzeit-Prozessen behandelt werden. Durch einen solchen Aufbau sind minimale Änderungen am Linux-Kernel nötig, da der Interrupt-Handler auf den Echtzeitkernel abgestimmt werden muß. Der Real-Time Linux-Kernel selbst ist ein ladbares Modul und kann bei Bedarf eingefügt und entfernt werden. Dieser Kernel stellt hauptsächlich die Schicht zwischen dem Linux-Kernel und der Hardware-Interruptebene dar. Real-Time Linux mutet anfänglich sehr spartanisch an, denn es liegt der Gedanke zugrunde, daß die Prinzipien von harten Real-Time Anwendungen in der Regel nicht vereinbar sind mit komplexer Synchronisation, dynamischer Resourcenverwaltung und allem anderen, was zeitliche Mehrkosten verursacht. Daher werden in der Standardkonfiguration nur sehr einfache Konstrukte zur Implementierung von Prozessen zur Verfügung gestellt, beispielsweise ein einfacher fixed priority scheduler, nur statisch allozierter Speicher und kein Schutz des Adressraums. Dennoch kann man mit den gegebenen Mitteln sehr effizient arbeiten, da alle Nicht-Echtzeitkomponenten unter Linux laufen und somit dortige Bibliotheken benutzen können. Real-Time Programme bzw. Prozesse werden als ladbare Module eingebunden, was das Echtzeitbetriebssystem erweiterbar und einfach zu modifizieren macht. Die Programme werden mit den Standard-Linux-Werkzeugen erzeugt (GNU C compiler). Oftmals übernehmen andere, Nicht-Real-Time Programme, die weitere Datenverarbeitung. Da Real-Time Linux keinerlei Mechanismen zum Schutz gegen Überladung der Hardware besitzt, ist es möglich, daß die Real-Time Prozesse die CPU blockieren. In diesem Fall bekommt der Linux-Kernel nicht die Chance, eine Zeitscheibe zu bekommen, da er die niedrigste Priorität hat. Da Real-Time Linux OPENSOURCE ist und da die Module von Real-Time Linux ladbar und damit jederzeit austauschbar sind, fand eine breite Entwicklung von Zusatzmodulen statt. Als Zusatzmodule stehen zur Zeit alternative Scheduler (z.B. rate-monotonic scheduler), ein Semaphore Modul, IPCs (interprocess communication Mechanismen) und etliche Real-Time device driver zur Verfügung. Real-Time und Linux-User Prozesse kommunizieren über lock-freie Queues (FIFOs) und shared memory. Anwendungsprogrammierern stellen sich die FIFOs als standard character devices dar, auf die mittels POSIX `read/write/open/ioctl` zugegriffen wird. Es hat sich gezeigt, daß Real-Time Linux den Anforderungen eines Echtzeitbetriebssystems sehr nahekommt. Die Worst-Case interrupt latency auf einem 486/33MHz PC liegt unter 30μs ($30 \cdot 10^{-6}$s) und somit nahe dem Hardwarelimit [20].“ [36]

4.1.1 API - die Grundfunktionen

„Im folgenden werden die wichtigsten in der Scanner-Software verwendeten Real-Time Interfacefunktionen vorgestellt. Dabei wird die neue POSIX-konforme API V2 verwendet.

Real-Time Scheduler

Die Aufrufe zur Prozeß-Verwaltung sind ab der Version 2 dem Standard `pthread` angeglichen. Für einen Real-Time-Prozeß steht folgende Struktur zur Verfügung:

```
struct rtl_thread_struct {
        int *stack;     /* hardcoded */
        int uses_fp;
        enum rtl_task_states state;
        int *stack_bottom;
        rtl_sched_param sched_param;
        struct rtl_thread_struct *next;
        RTL_FPU_CONTEXT fpu_regs;
        int cpu;
        hrtime_t resume_time;
        hrtime_t period;
        void *retval;
        int pending_signals;
        struct tq_struct free_task;
        void *user[4];
        int errno_val;
        int pad [64];
};

typedef struct rtl_thread_struct RTL_THREAD_STRUCT;
typedef RTL_THREAD_STRUCT *pthread_t;
```

Es genügt daher, einen Thread wie folgt zu definieren

```
pthread_t REALTIME_TASK;
```

Eine Funktion *(Prozeß)* kann nun diesem Thread zugeordnet und gestartet werden:

```
int pthread_create(pthread_t  *thread,
                   pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   (void *arg));
```

Nun muß der Prozeß periodisch gemacht werden. Dies geschieht mittels folgender nicht-posix konformer (erkennbar an dem Postfix `_np`) Funktion:

```
int pthread_make_periodic_np(pthread_t  thread,
                             hrtime_t start_time,
                             hrtime_t period);
```

Innerhalb des Prozesses lassen sich die Parameter des Threads (wie beispielsweise die Periode `REALTIME_TASK->period`) beeinflussen. Der Prozeß läßt sich bis zu dieser Periode mittels der Funktion

```
int pthread_wait_np(void);
```

explizit schlafen legen, die Steuerung wird an andere Prozesse zurückgeben. Prozesse mit der Priorität 1 haben die höchste Priorität, der normale Linux-Kernel wird mit der niedrigsten betrieben.

Zeit-Verwaltung

Für Zeiten in Real-Time Linux wird die Datenstruktur `hrtime_t` verwendet. Dies ist ein 64-Bit Integer. Die aktuelle Auflösung der Zeit (in Prozessorticks) durch diese Variable ist hardwareabhängig, es wird aber garantiert, daß die Auflösungsgenauigkeit unter 1μs (Mikrosekunde) liegt. Das Symbol `HRTIME_INFINITY` repräsentiert den maximalen Wert und wird niemals erreicht. Das Symbol `HRTICKS_PER_SEC` ist definiert als die Ticks pro Sekunde. Die Funktion

```
hrtime_t gethrtime(void);
```

liefert die aktuelle Zeit.

FIFO-Handling

Für Linux-Prozesse stehen die Real-Time FIFOs als character devices `/dev/rtf0`, `/dev/rtf1` etc. zur Verfügung. Der Real-Time Prozeß muß die FIFOs explizit erzeugen und ihnen eine Größe geben. Dies geschieht mit der Funktion

```
int rtf_create(unsigned int fifo, int size);
```

Innerhalb der Real-Time Applikation kann mit den Funktionen

```
int rtf_put(unsigned int fifo, char *buf, int count);
int rtf_get(unsigned int fifo, char *buf, int count);
```

auf die FIFOs zugegriffen werden. Eine wichtige Funktion ist der Handler, der immer dann aufgerufen wird, wenn ein Linux-User-Programm auf einen FIFO zugreift. Ein solcher Handler wird mit folgender Funktion mit dem FIFO verbunden:

```
int rtf_create_handler(unsigned int fifo, int (* handler)(unsigned int fifo));
```

“ [36]

4.2 Servos ansteuern mit Real-Time Linux

Servomotoren kommen ursprünglich aus dem Bereich des Modellbaus, werden aber wegen ihrer Robustheit und Zuverlässigkeit immer öfter auch für industrielle Anwendungen eingesetzt. Die hier verwendeten Servos der Firma Volz werden über ein TTL-kompatibles pulsweiten-moduliertes Signal (PWM) angesteuert. Dabei handelt es sich um ein periodisches Rechteck-Signal dessen Highimpulslänge die Stellung des Servos bestimmt, d.h. 0,8ms=links, 1,5ms=Neutralstellung, 2,2ms rechts. Damit die Servoelektronik das Signal auswerten kann, muß die Periodendauer zwischen 15ms und 30ms liegen, ideal sind 20ms [39] (vgl. Abbildung 4.1).

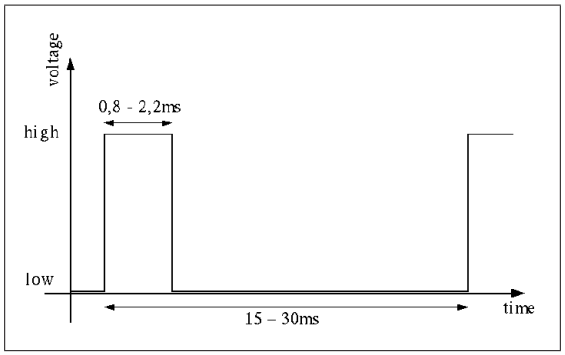

Abbildung 4.1: Zeitlicher Verlauf eines pulsweiten-modulierten Signals. Die Länge des High-Impulses bestimmt die Stellung des Servos.

Die Abbildung 4.2 veranschaulicht den Anschluß eines Servos an den Parallelport. Die Spannungsversorgung muß extern erfolgen und die Masse der Spannungsversorgung muß unbedingt mit der Masse der Parallelports verbunden sein, um ein gemeinsames Massepotential zu gewährleisten. Bis zu 12 Servos können auf diese Weise über den Parallelport angesteuert werden [25].

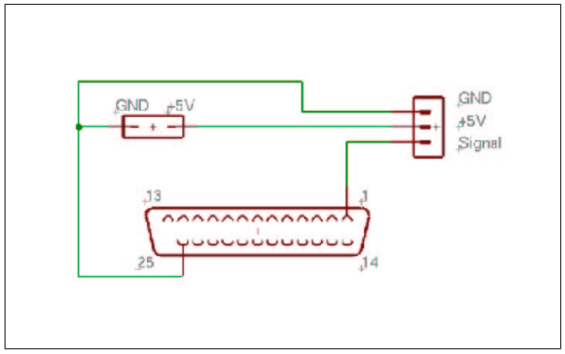

Abbildung 4.2: Anschluß eines Servomotors an den Parallelport eines PC's.

4.2.1 Das Real-Time Modul `rt_servo`

Das bisher verwendete Real-Time Modul `rt_servo` steuert den Servo des 3D-Laserscanners und generiert 3 High-Signale zum Einschalten verschiedener Power-Switch-Schaltungen. In diesem Modul wird ein periodischer Prozeß gestartet, der im wesentlichen aus einer Endlosschleife besteht. In dieser wird zunächst das Bitmuster für die 3 High-Signale und das Low-Signal des Servos in das Register des Parallelports geschrieben und dadurch die entsprechenden TTL-Signale erzeugt. Anschließend wird der Prozeß für die aktuell eingestellte Periodendauer schlafen gelegt, d.h. die Kontrolle wird an andere Prozesse abgegeben. Eine neue Periodendauer für den Prozeß wird festgelegt. Sobald die Zeit abgelaufen ist, wird die Kontrolle wieder an den periodischen Prozess übergeben. Am Parallelport wird das High-Signal des Servos angelegt und der Prozeß für die neu eingestellte Periodendauer wieder schlafen gelegt, diese Zeit bestimmt die Stellung des Servos. Zusätzlich besteht die Möglichkeit, den Servomotor kontinuierlich zu drehen und das Ende der Drehung an das aufrufende Programm zurück zu melden. Die Kommunikation mit dem RT-Modul erfolgt über 2 Real-Time FIFOs. Über den ersten FIFO (`/dev/rtf0`) können Anwendungsprogramme Anweisungen an das Modul senden. Dazu wird folgende Struktur verwendet:

```
struct my_msg_struct{
   char option;
   hrtime_t data1;
   hrtime_t data2;
   hrtime_t time;
};
```

Wird als `option` der Buchstabe 'N' gesendet, wird die in `data1` übermittelte Zeit als Stellung für den Servo festgelegt. Bei der Option 'T' wird eine Bewegung des Servomotors von der Position `data1` zur Position `data2` generiert, wobei das Signal jeweils um `time` Ticks erhöht bzw.

erniedrigt wird. Diese Bewegung wird von dem Modul selbständig generiert und somit kann das aufrufende Programm bereits andere Aufgaben ausführen. Das Ende der Drehung wird durch den zweiten FIFO (`/dev/rtf1`) mitgeteilt. Sobald die Drehung beendet ist, wird von dem Real-Time Modul ein 'D' in den FIFO geschrieben und kann vom Anwendungsprogramm ausgelesen werden.

4.2.2 Modifikation des Real-Time Moduls

Die Steuerung des Kamerasystems soll ebenfalls vom Real-Time Modul `rt_servo` übernommen werden, wodurch einige Modifikationen notwendig sind. Die Anzahl der erzeugten PWM Signale musste auf 3 erhöht und ein während der Laufzeit des Modules abschaltbares High-Signal für den Power-Switch generiert werden. Da die drei Servosignale verschiedene Werte annehmen müssen, ist es notwendig, den Algorithmus zur Erzeugung der Signale zu modifizieren. In der folgenden Schleife werden jetzt die PWM-Signale nacheinander erzeugt:

```
for(servo=0;servo<NUM_SERVO;servo++){

   if(servo==NUM_SERVO-1){
      servo_task->period = servo_data[0].length;
   }else {
       servo_task->period = servo_data[servo+1].length;
    }

   outb(servo_data[servo].mask1on, PORT);
   outb(servo_data[servo].mask2, PORT+2);
   pthread_wait_np();

}
```

Zu Beginn wird die Periodendauer des Real-Time Tasks auf die Länge des beim nächsten Schleifendurchlaufs zu erzeugenden Signales gesetzt. Anschließend werden die Register des Parallelportes mit den entsprechenden Bitmustern gefüllt, um das erste PWM-Signal zu erzeugen und der Prozess wird für die alte Periodendauer schlafen gelegt. Danach wird durch ein neues Bitmuster das erste Signal wieder gelöscht und das nächste erzeugt. Zum Schluß wird noch ein 15ms Low-Signal für alle Servos erzeugt, um eine Periodendauer zwischen 17,4 und 21,6ms, je nach Länge der einzelnen Signale, zu erhalten. Um das Signal des Power-Switch abschalten zu können, musste die `servo_data` Struktur geändert werden. Es mußten verschiedene Bitmuster abgelegt werden, mit und ohne Power-Switch-Signal, und eine Option eingefügt, die bestimmt, welches verwendet wird. Anschließend wurde die Servoschleife erweitert, so daß die Power-Switch Option abgefragt und das entsprechende Bitmuster geladen wird. Die Struktur `msg_struct`, zur

Kommunikation zwischen Anwendungsprogramm und Real-Time Modul, wurde um die Variable servo erweitert:

```
struct my_msg_struct{
   char option;
   int servo;
   hrtime_t data1;
   hrtime_t data2;
   hrtime_t time;
};
```

Dies ermöglicht den Anwendungsprogrammen, die Werte der richtigen `servodata` Struktur zu ändern. Zusätzlich wurden zwei weitere Optionen in die Auswertung von `msg_struct` eingefügt, eine zum Einschalten des Power-Switch-Signales und die andere zum Ausschalten. Der Algorithmus zur Erzeugung von kontinuierlichen Bewegungen wurde dahingehend angepasst, daß jeweils die Daten des richtigen Servos eingelesen und ausgewertet werden.

4.3 Benutzer-Interface

Für die Steuerung des Kamerasystemes ist ein Benutzer-Interface nötig, das über die beiden Real-Time FIFOs mit dem Modul rt_servo kommuniziert. Auf der einen Seite muß dieses eine Schnittstelle zu anderen auf der Roboterplattform laufenden Programmen sein und auf der anderen Seite eine möglichst intuitive Schnittstelle für den Menschen.

4.3.1 Kommandozeilenprogramme

Die Softwareschnittstelle stellen zwei Kommandozeilenprogramme dar, die beim Aufruf übergebene Argumente auswerten. Nach dem Aufruf wird eine Instanz der Klasse `t_Servo` mit den entsprechenden Argumenten erzeugt. Die verschiedenen Methoden der Instanz übernehmen dann das Senden von `msg_struct` Nachrichten an das Modul `rt_servo`. Mit `setServo` kann man einem bestimmten Servo eine Stellung zuweisen. Dies geschieht durch die Übergabe von zwei Parametern, zuerst die Stellung des Servos (0..255) und dann die zu dem gewünschten Servo gehörende Nummer, d.h. Scannerservo = 0, Schwenkservo = 1 und Nickservo = 2. Die Kompatibilität zur bereits bestehenden Laserscanner-Software ist durch den default Wert '0' des Servos gewährleistet. Mit den Optionen „on“ und „off“ kann der Power-Switch geschaltet werden. Für die kontinuierlichen Bewegungen wurde das Programm `moveServo` entwickelt. Hier werden als Argumente Startstellung, Endstellung, Dauer des Bewegung (in 1/10sec) und der gewünschte Servo übergeben. Auch hier ist der default Wert des Servos '0'. Diese Programme ermöglichen es zwar das System zu steuern, aber die Bedienung ist nicht sehr intuitiv. Aus diesem Grund wurde zusätzlich ein grafisches Benutzer-Interface entwickelt.

4.3.2 Grafische Oberfläche mit JAVA

Zur Realisierung der grafischen Oberfläche wurde die Sprache JAVA verwendet.

JBuilder

Die von der Firma Borland entwickelte integrierte Entwicklungsumgebung „JBuilder“ [8] ermöglicht die effiziente Entwicklung von Software mit grafischen Oberflächen in der Sprache JAVA. Zur grafischen Gestaltung stehen eine Vielzahl von Bibliotheken zur Verfügung.

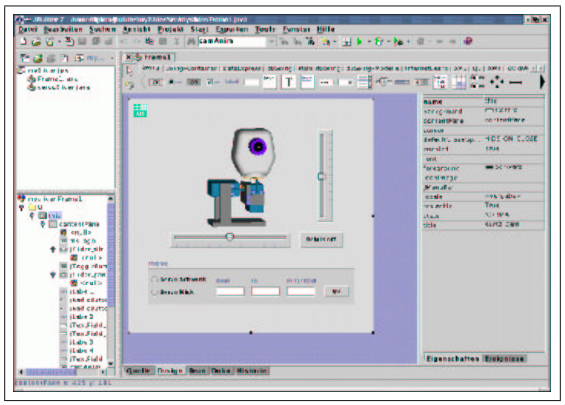

Abbildung 4.3: Oberfläche des JBuilder 7.0

Grafische Oberfläche

Abbildung 4.4 zeigt die grafische Oberfläche des Benutzer-Interfaces. Über die beiden Schieberegler kann die entsprechende Achse (Schwenk, Nick) einzeln bewegt werden. Zum besseren Verständnis der Steuerung bewegt sich das Modell des Kamerasystemes in der Bildmitte entsprechend der Reglerstellung. Über den Button „Relais Off“ kann die Spannungsversorgung der Servos an- bzw. abgeschaltet werden. Im unteren Teil können kontinuierliche Bewegungen der Servos erzeugt werden. Auch hier ist die Bedienung sehr intuitiv, es werden die gewünschte Achse, der Startpunkt, der Endpunkt und die Dauer der Bewegung angegeben.

Die Kommunikation mit dem Real-Time Modul zur Umsetzung der Befehle geschieht über die beiden Kommandozeilenprogramme `setServo` und `moveServo`. Die Aufrufe der Programme werden mit `Runtime.getRuntime().exec()` vorgenommen. Das Programm `setServo` wird immer dann gestartet, wenn einer der Schieberegler bewegt wurde. Das folgende Listing zeigt die Implementierung des Programmaufrufes mit JAVA:

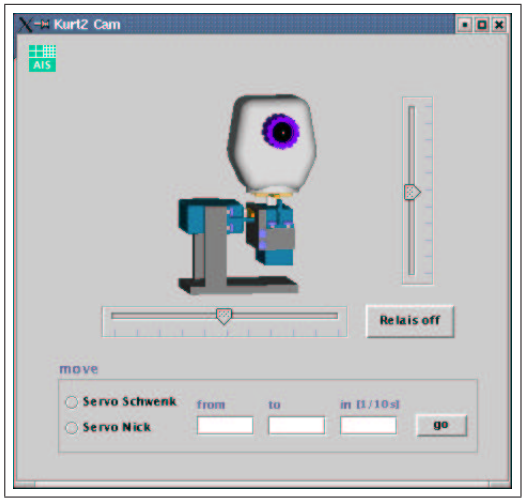

Abbildung 4.4: Das Grafische Benutzer-Interface zur Kamerasteuerung.

```
void jSlider_tilt_stateChanged(ChangeEvent e) {
  try{  Runtime.getRuntime().exec("'/home/diplom/rt_modules/bin/setServo "'+
          Integer.toString(jSlider_tilt.getValue())+"' 2"');
  } catch (Exception ee) { System.out.println("'SetServo-Error: "' + ee);}
  change_camlogo();
}
```

Kontinuierliche Bewegungen werden durch den Aufruf des Programmes `moveServo` erzeugt:

```
void jButton_go_actionPerformed(ActionEvent e) {
  if (jRadioButton_schwenk.isSelected()){
    try{  Runtime.getRuntime().exec("'/home/diplom/rt_modules/bin/moveServo "'+
            jTextField_from.getText()+"' "'+jTextField_to.getText()+"' "'+
              jTextField_time.getText()+"' 1"');
        java.lang.String string = jTextField_to.getText();
    }catch (Exception ee) { System.out.println("'SetServo-Error: "' + ee);}
  } else if (jRadioButton_nick.isSelected()){
           try{  Runtime.getRuntime().exec("'/home/diplom/rt_modules/bin/moveServo "'+
         jTextField_from.getText()+"' "'+ jTextField_to.getText()+"' "'+
                     jTextField_time.getText()+"' 2"');
           }catch (Exception ee)
         { System.out.println("'SetServo-Error: "' + ee);}
         };
}
```

4.4 Alternative Ansteuerung der Servomotoren

Ein Nachteil der in Kapitel 4.2 vorgestellten Methode zur Ansteuerung der Servomotoren über den Parallelport ist die unbedingte Verwendung eines Echtzeit-Betriebssystemes, in diesem Fall Real-Time Linux. Die Installation eines Linux-Betriebssystems mit Real-Time Kernel ist komplexer als die eines Standard-Betriebssystems und auch die Administration erfordert größere Erfahrung auf diesem Gebiet. Durch die unbedingt notwendige Echtzeitfähigkeit des Betriebssystemes ist eine Verwendung eines der sehr weit verbreiteten Windows-Betriebssysteme nicht möglich. Zusätzlich wurde während der Programmierarbeiten an den Real-Time Modulen festgestellt, daß es bei identischen Installationen auf verschiedenen Systemen zu unterschiedlichen und teilweise gravierenden Timing-Problemen kommt. Dabei können Verzögerungen von bis zu 0,6ms auftreten, wodurch die Erzeugung von sauberen PWM-Signalen unmöglich wird. Abbildung 4.5 zeigt die Ergebnisse von Messungen zur Timing-Genauigkeit auf zwei unterschiedlichen Systemen. Bei dieser Messung wird ein periodischer Prozeß (Periodendauer = 1000μs) erzeugt und über einen gewissen Zeitraum die maximale und minimale Abweichung im Timing ermittelt. Auch die geringe Anzahl von großen Timing-Fehlern beim ersten System (vgl. Abbildung 4.5, links) kann zu undefinierten Zuständen des Robotersystems führen.

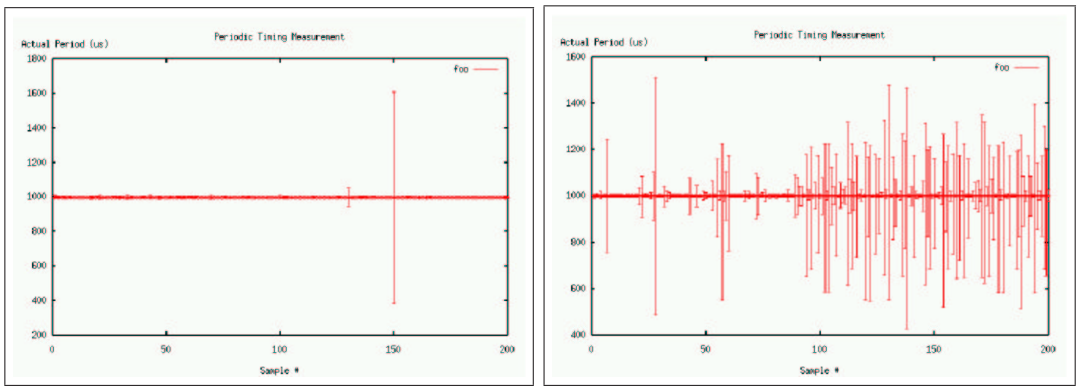

Abbildung 4.5: Ergebnisse einer Timing-Genauigkeits Messung auf zwei verschiedenen Systemen mit Real-Time Linux als Betriebssystem. Die maximale und minimale Abweichung eines periodischen Prozesses mit der Periodendauer 1000μs wird dargestellt.

Eine Alternative zur bisherigen Ansteuerung ist die Nutzung einer Zusatzhardware, die über eine serielle Schnittstelle angesteuert wird. Die Servoplatine „S10“ (vgl. Abbildung 4.6) der Firma Animatronik.de [4] bietet die Möglichkeit bis zu acht Servos zu steuern. Sie kann direkt auf den seriellen Port eines PC's aufgesteckt werden und benötigt eine externe Spannungsversorgung von 4,8 bis 6V. Hierüber werden auch die Servomotoren versorgt. Die Steuerung der Servomotoren geschieht durch das Senden von kodierten Zeichenketten über die serielle Verbindung. In solch einer Zeichenkette wird neben dem zu steuernden Servomotor und der gewünschten Position

auch noch ein Kontroll-Bit mit der Länge der Zeichenkette übermittelt, um zu verhindern, daß bei Übertragungsfehlern nicht gewollte Positionen angefahren werden. Die Auflösung der Servostellung liegt bei 550 Schritten, was einer PWM-Signal Auflösung von 3μs entspricht.

Um diese Hardware nutzen zu können, wurde das Modul `rt_servo` durch das Modul `servo` ersetzt. Am folgenden Beispiel der `set()` Funktion wird die Arbeitsweise des `servo` Moduls veranschaulicht:

```
int t_Servo::set(const int& value){

 //creating string
  sprintf(tmp_string,"S%d,%d#", servonum, value);
  sprintf(set_string,"%d", (strlen(tmp_string)-1));
  strcat(set_string, tmp_string);

 //write string to serial device until command is successful executed
 //or max. time delay is reached
  do{
    write(fd_RS232, set_string, strlen(set_string));
    wait = 0.0;
   //wait for answer from servo modul
    do {
      Get_mtime_diff(0);
      res = read(fd_RS232,buf,2);
      for (int i=0; i<res;i++) {
        if (buf[i]=='#') stop = true;
      }
      wait +=  Get_mtime_diff(0);
    } while (stop==false && wait < 5.0);
    stop = false;
    send++;
  } while(strncmp(buf, "O#", 2) && send < 10);
  if (strncmp(buf, "O#", 2))    return -1;
  return 0;
 }
```

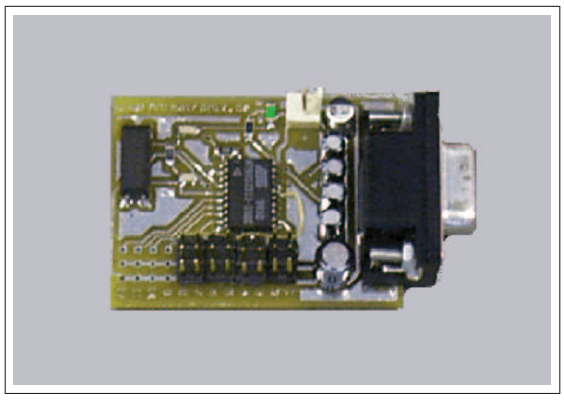

Abbildung 4.6: Die Servoplatine S10.

Kapitel 5

3D-Rekonstruktion und Verfahren zur Kamerakalibrierung

Bei der Fusion von 3D-Abstandsinformationen mit digitalen Farbbildern werden den Abstandsinformationen die entsprechenden Texturen zugewiesen. Im folgenden Kapitel werden die hierzu notwendigen Grundlagen der Photogrammetrie beschrieben, sowie einige Verfahren zur Kalibrierung von digitalen Kameras vorgestellt.

5.1 Digitale Photogrammetrie

Die digitale Photogrammetrie befasst sich mit der Gewinnung und Verarbeitung von Informationen über Objekte mit Hilfe von digitalen Bildern. Schwerpunktmäßig mit der Bestimmung von Form, Größe und Lage von Objekten im Raum [16, 11]. Da durch die Informationen des Laserscanners bereits die 3D-Abstandsinformationen vorliegen, ist dies nicht Ziel des zu entwickelnden Verfahrens, sondern viel mehr sollen den Abstandsinformationen die entsprechenden Texturen zugeordnet werden. Die dazu notwendigen Überlegungen sind aber in beiden Fällen sehr ähnlich. Um die geometrischen Beziehungen zwischen den Objekten im Raum und ihren digitalen Bilddaten mathematisch beschreiben zu können, müssen folgende Fragestellungen gelöst werden [14, 13]:

- Welche geometrischen Beziehungen bestanden während der Aufnahme zwischen der Kamera und den abgebildeten Objekten?
  Eine genaue Messung der Positionen von Kamera und den abgebildeten Objekten ist im allgemeinen aufwendig. Durch die 3D-Abstandsinformationen des Laserscanners können aber Lage und Orientierung der abgebildeten Objekte bestimmt werden. Zusammen mit Informationen über die Größe eines oder mehrerer der abgebildeten Objekte können alle geometrischen Beziehungen berechnet werden. Diese Beziehungen werden später durch die

extrinsischen Kamera-Parameter dargestellt.

- Wie ist der Zusammenhang zwischen Punkten in den Bilddaten und der Richtung des Raumstrahles zu dem verursachenden Objekt?
  Bei der Abbildung eines Objektes durch eine reale Linse kommt es zu einer Krümmung der Verbindungslinie von Objekt, Linse und Abbildung. Sie ergibt sich aus dem internen Aufbau der Kamera und der Beschaffenheit des Objektivs. Sowohl die relative Lage von Objektiv und Bildsensor als auch die optischen Eigenschaften der verwendeten Linsen sind hier von Bedeutung. Diese Eigenschaften der Kamera werden später durch die intrinsischen Kamera-Parameter dargestellt [14, 13].

Im weiteren wird zunächst der Abbildungsvorgang einer digitalen Kamera betrachtet, welcher sich später in zwei Abschnitte aufteilen läßt. Diese können dann auf die eben gestellten Fragen zurückgeführt werden.

5.2 Abbildungsvorgang einer digitalen Kamera

Im allgemeinen kann der Aufbau einer Digitalkamera durch einen rechteckigen CCD-Sensor und eine Bikonvexlinse beschrieben werden. Dadurch stellt sich der zu modellierende Abbildungsvorgang als eine Transformation eines Punktes $P = [X_o, Y_o, Z_o]^T$ aus dem $\mathbb{R}^3$ Raum in die Ebene des CCD-Sensors auf den Punkt $P' = [u_c, v_c]^T$ dar. Für diese Modellierung werden insgesamt 3 Koordinatensysteme (vgl. Abbildung 5.1) benötigt [14]:

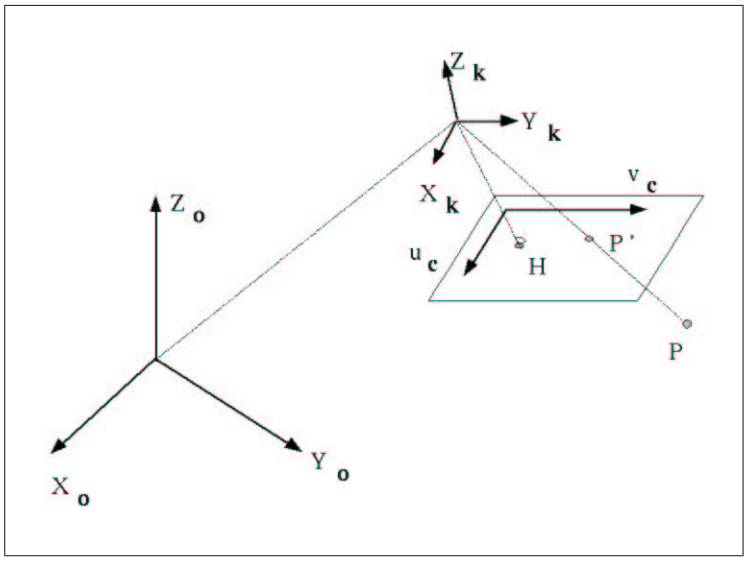

Abbildung 5.1: Geometrie des Abbildungsvorgangs einer digitalen Kamera. Objektkoordinatensystem S_o (X_o, Y_o, Z_o), Kamerakoordinatensystem S_k (X_k, Y_k, Z_k), Ebene des CCD-Sensors S_c (u_c, v_c), Hauptpunkt P, Objektpunkt P und Bildpunkt P' [14].

1. das Objektkoordinatensystem S_o
2. das Kamerakoordinatensystem S_k
3. das Koordinatensystem des CCD-Sensors S_c

Die Transformation vom Objektkoordinatensystem zum Koordinatensystem des CCD-Sensors läßt sich in 4 Schritten modellieren (vgl. Abbildung 5.2). Zunächst die Transformation des Objektpunktes P von S_o nach S_k. Im zweiten Schritt dann die ideale Projektion des Raumpunktes P in die Bildebene zum idealen Punkt $\bar{P}$. Dieser Punkt $\bar{P}$ wird dann vom Kamerakoordinatensystem S_k in das Koordinatensystem des CCD-Sensors S_c transformiert. Im vierten Schritt werden schließlich die Abbildungsfehler durch die Linse modelliert, um den Bildpunkt P' zu erhalten [14].

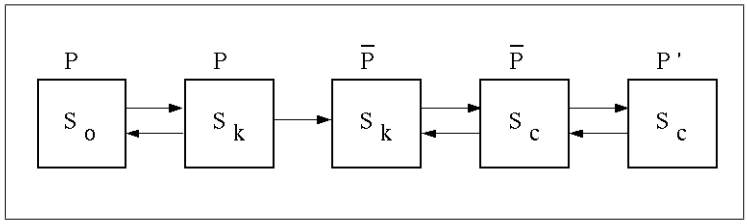

Abbildung 5.2: Schema des Abbildungsvorgangs einer digitalen Kamera [14].

Die für diese vier Schritte notwendigen Parameter werden in extrinsische und intrinsische Parameter unterteilt. Die extrinsischen Parameter beschreiben die Transformation vom S_o nach S_k und beantworten somit die Frage nach den geometrischen Beziehungen zwischen der Kamera und den abgebildeten Objekten. Die intrinsischen Parameter beschreiben sowohl die Transformation von S_k nach S_c als auch die Korrektur der Abbildungsfehler der Linse. Durch sie kann die Krümmung des Raumstrahles zwischen dem Bildpunkt und dem Objektpunkt beschrieben werden [14, 24, 40].

5.2.1 Extrinsische Parameter

Die Transformation vom System S_o nach S_k kann durch eine Translation t und eine Rotation R vollständig beschrieben werden.

$$P_k = [Rt]P_o \tag{5.1}$$

Zunächst wird die Translation des Systems S_o in das System S_k durchgeführt, welche durch den Vektor $t = (x, y, z)^T$ beschrieben wird. Dessen Parameter stellen den Ursprung des Systems S_k in Objektkoordinaten dar. Anschließend wird das System S_o in das System S_k rotiert [14]. Diese Rotation R kann durch drei Euler-Winkel $\theta_x, \theta_y, \theta_z$, die einer Drehung um die X, Y und

Z-Achse des Systems S_k entsprechen, ausgedrückt werden. Im folgenden wird für R eine 3×3 Rotationsmatrix verwendet, die durch

$$R = R_x R_y R_z \tag{5.2}$$

berechnet wurde, wobei

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}$$

ist und θ_x den Winkel der Rotation um die X-Achse bezeichnet. Die Matrizen R_y, R_z sind analog definiert:

$$R_y = \begin{pmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{pmatrix}, \quad R_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Damit ergibt sich die Rotationsmatrix als

$$\mathbf{R} = \begin{pmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & -\sin\theta_y \\ -\sin\theta_x \sin\theta_y \cos\theta_z + \cos\theta_x \sin\theta_z & \sin\theta_x \sin\theta_y \sin\theta_z + \cos\theta_x \cos\theta_z & -\sin\theta_x \cos\theta_y \\ \cos\theta_x \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z & -\cos\theta_x \sin\theta_y \sin\theta_z + \sin\theta_x \cos\theta_z & \cos\theta_x \cos\theta_y \end{pmatrix}.$$

Hierbei ist die Reihenfolge der Multiplikationen in (5.2) von entscheidender Bedeutung. Das Ergebnis einer Rotation hängt im allgemeinen davon ab, um welche Achse zuerst rotiert wird [30, 32]. In Anhang A.2 wird die mathematische Darstellung von Rotationen und Translationen im $\mathbb{R}^3$ Raum ausführlicher beschrieben.

5.2.2 Intrinsische Parameter

Die intrinsischen Parameter beschreiben die Transformation des Punktes P vom Kamerakoordinatensystem S_k in das Koordinatensystem des CCD-Sensors S_c zum Punkt P'. Praktisch bedeutet dieser Schritt die Abbildung durch die Linse der Kamera auf den CCD-Sensor.

Abbildung 5.3 zeigt das Abbildungsverhalten einer realen Linse auf einen CCD-Sensor. Der Brennpunkt F liegt auf der optischen Achse im Abstand f von der Hauptebene H_2. Die Schnittpunkte der Hauptebenen H_1 und H_2 mit der optischen Achse bilden die Punkte K_1 und K_2.

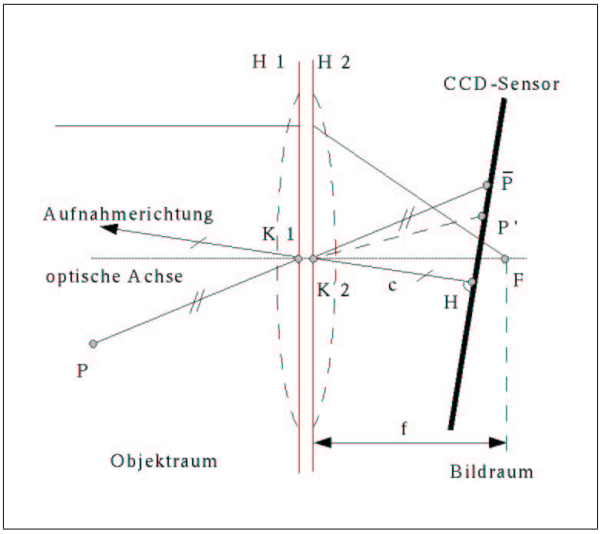

Abbildung 5.3: Geometrische Zusammenhänge bei der Abbildung durch eine reale Bikonvexlinse [14, 13].

Der Fußpunkt des Lots von K_2 auf den CCD-Sensor bildet den Hauptpunkt H. Die Richtung HK_2 wird als Aufnahmerichtung bezeichnet. Bei einer idealen Linse, ohne Abbildungsfehler, würde der Punkt P auf den Punkt $\bar{P}$ abgebildet und der Abbildungsstrahl hätte im Objektraum und im Bildraum die gleiche Richtung, wäre aber durch die Punkte K_1 und K_2 geringfügig versetzt. Durch eine reale Linse wird der Punkt P aber auf den Punkt P' abgebildet [14].

Ideale Kamera

Bei der Modellierung wird zunächst von einer idealen Abbildung ausgegangen, so daß radiale und tangentiale Abbildungsfehler der Linse (Distortion) vernachlässigt werden können. Diese ideale Abbildung kann durch

$$\bar{P} = AP \tag{5.3}$$

beschrieben werden (mit A als 3×3 Projektionsmatrix). Im allgemeinen ist die Lage der Bildfläche (hier: der CCD-Sensor) zum Objektraum vernachlässigbar, wodurch die Hauptebenen H_1 und H_2 in eine Ebene geschoben werden können [14]. Dies führt dazu, daß alle Abbildungsstrahlen Geraden sind und durch den Punkt $K_{neu} = K_1 = K_2$ gehen. Dieser Punkt bildet den Ursprung des Kamerakoordinatensystems S_k und wird im folgenden Projektionszentrum genannt und mit O bezeichnet. Zur Vereinfachung der Abbildungsbeziehungen werden die Achsen X_k und Y_k parallel zur Bildebene gelegt und bilden mit der senkrecht auf der Bildebene stehenden Achse Z_k ein kartesisches Koordinatensystem. Der Abstand zwischen der Bildebene und dem Projektionszentrum ist die Kamerakonstante und wird mit c bezeichnet. Im ersten Schritt der

Modellierung wird der Ursprung des Koordiniatensystems des CCD-Sensors (u_c, v_c, w_c) in den Hauptpunkt H gelegt, wodurch die optische Achse und die Aufnahmerichtung zusammenfallen.

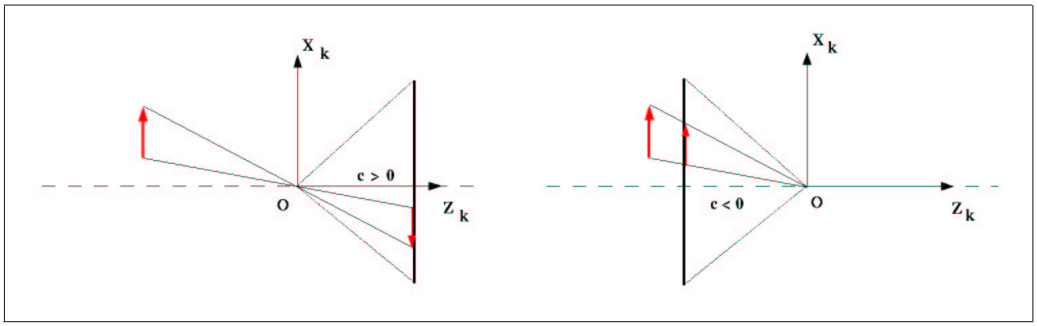

Abbildung 5.4: Ideale Abbildung einer digitalen Kamera. Das Objekt erscheint um 180° gedreht und an der Mittelsenkrechten gespiegelt (links). Durch die Wahl einer negativen Kamerakonstanten c erscheint das Objekt richtig herum (rechts) [14, 29].

Die vereinfachte ideale Abbildung läßt das Objekt um 180° gedreht und an der Mittelsenkrechten gespiegelt erscheinen (vgl. Abbildung 5.4). Zur besseren Veranschaulichung wird eine negative Kamerakonstante c gewählt, wodurch das Objekt richtig herum erscheint [14, 29]. Durch Anwendung des Strahlensatzes erhalten wir die Koordinaten

$$ {}^{\bar{P}}u_c = -c\frac{{}^{P}X_k}{{}^{P}Z_k} \qquad {}^{\bar{P}}v_c = -c\frac{{}^{P}Y_k}{{}^{P}Z_k} \qquad {}^{\bar{P}}w_c = -c\frac{{}^{P}Z_k}{{}^{P}Z_k} \tag{5.4} $$

für den Punkt $\bar{P}$ als Funktion der Kamerakoordinaten. In homogenen Koordinaten (vgl. Anhang A.3) läßt sich 5.4 als

$$ \begin{pmatrix} {}^{\bar{P}}u_c \\ {}^{\bar{P}}v_c \\ {}^{\bar{P}}w_c \\ {}^{\bar{P}}T \end{pmatrix} = \begin{pmatrix} -c & 0 & 0 & 0 \\ 0 & -c & 0 & 0 \\ 0 & 0 & -c & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} {}^{P}X_k \\ {}^{P}Y_k \\ {}^{P}Z_k \\ 1 \end{pmatrix} \tag{5.5} $$

ausdrücken. Da in der Bildebene die ${}^{\bar{P}}w_c$-Koordinate nicht benötigt wird, kann die dritte Zeile vernachlässigt werden und es ergibt sich für die Projektionsmatrix

$$ A_1 = \begin{pmatrix} -c & 0 & 0 \\ 0 & -c & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.6} $$

Im zweiten Schritt der Modellierung der idealen Abbildung wird nun davon ausgegangen, daß der Hauptpunkt H und somit der Ursprung des Systems S_c nicht auf der optischen Achse liegt, sondern leicht verschoben ist (vgl. Abbildung 5.3, 5.6). Desweiteren werden die, durch den strukturellen Aufbau eines CCD-Sensors (vgl. Abbildung 5.5) bedingten, leichten Unterschiede im Auflösungsvermögen zwischen der horizontalen und vertikalen Richtung berücksichtigt. Die Scherung s (vgl. Abbildung 5.6) der beiden Achsen eines CCD-Sensors ist vernachlässigbar gering und wird daher bei der Modellierung nicht berücksichtigt [14]. Abbildung 5.6 zeigt die Lage der Systeme S_k und S_c für das neue Modell.

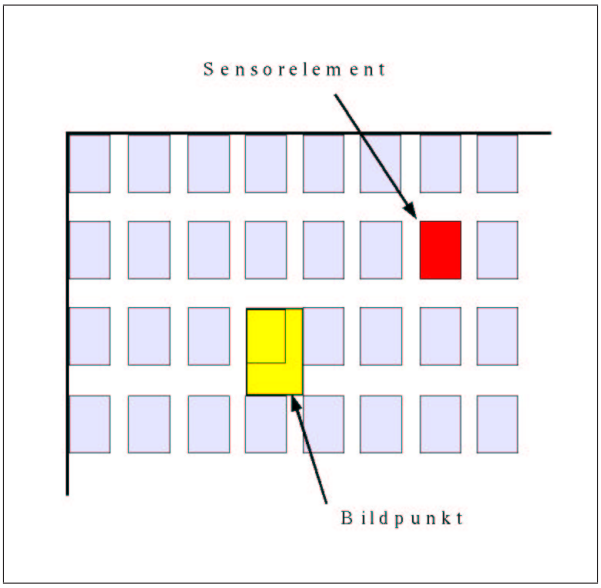

Abbildung 5.5: Aufbau eines CCD-Sensors mit unterschiedlichem Auflösungsvermögen in horizontaler und vertikaler Richtung [14].

Die Transformation wird beschrieben durch eine Verschiebung des Koordinatensystems S_k in den Hauptpunkt H. Die unterschiedliche Skalierung der Achsen wird durch den Faktor $1 + m$ berücksichtigt. Somit ergeben sich im zweiten Schritt die Koordinaten

$$\bar{P}u_c = {}^{P}X_k + u_0 \tag{5.7}$$
$$\bar{P}v_c = (1+m)^{P}Y_k + v_0 \tag{5.8}$$

für den Punkt $\bar{P}$ (vgl. Abbildung 5.6). Als Projektionsmatrix erhalten wir

$$A_2 = \begin{pmatrix} 1 & 0 & u_0 \\ 0 & 1+m & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.9}$$

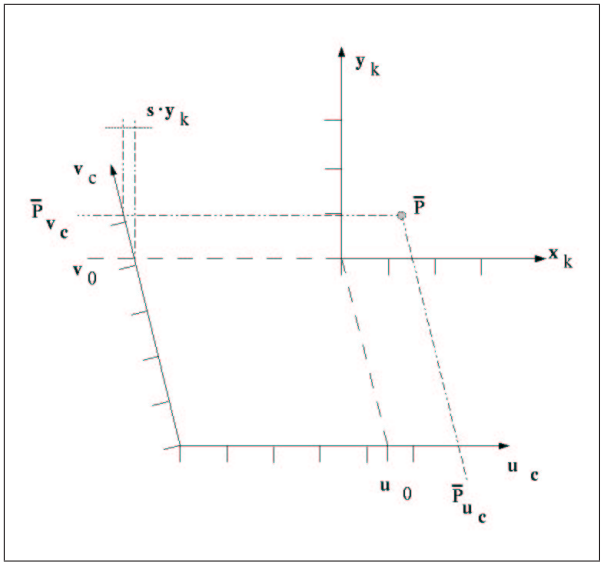

Abbildung 5.6: Lage der Koordinatensysteme S_o und S_k zueinander. Die Scherung s ist bei der Abbildung auf einem CCD-Sensor vernachlässigbar [14, 43].

Die Zusammenfassung der Projektionsmatritzen A_1 und A_2 bringt die endgültige Projektionsmatrix

$$A = A_2 A_1 = \begin{pmatrix} 1 & 0 & u_0 \\ 0 & 1+m & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -c & 0 & 0 \\ 0 & -c & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -c & 0 & u_0 \\ 0 & -c(1+m) & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.10}$$

der idealen Kamera. Die Koeffizienten u_0 und v_0 werden auch als principal Point bezeichnet.

Abbildungsfehler der Linse

Bisher wurden durch die Linse hervorgerufene Abbildungsfehler vernachlässigt. Bei der realen Abbildung müssen diese aber ebenfalls berücksichtigt werden. Bei einer digitalen Kamera sind radiale und tangentiale Verzerrungen dominierend [40, 19]. Sie werden durch kleine Unterschiede der Focuslänge in verschiedenen Bereichen der Linse hervorgerufen [13].

Der Effekt der radialen Verzerrung wird in Abbildung 5.7 sichtbar. In Abhängigkeit der verwendeten Linse tritt eine kissen- oder fassförmige Verzerrung auf. Beschreiben läßt sich diese Verzerrung durch ein Polynom ungerader Ordnung folgender Form [29]:

$$\tilde{r} = r + k_1 r^3 + k_2 r^5 + k_3 r^7 + ... \tag{5.11}$$

mit $r^2 = x^2 + y^2$. Roger Y. Tsai [37] und Zhengyou Zhang [40] haben die Auswirkungen radialer Verzerrungen untersucht und dabei festgestellt, daß für eine hinreichend genaue Modellierung eines Kamerasystems die ersten beiden Koeffizienten des Polynoms ausreichend sind [29]. Die Verwendung weiterer Terme würde die Genauigkeit nicht erhöhen und unter Umständen zu numerischen Instabilitäten bei der Berechnung der Parameter führen [37].

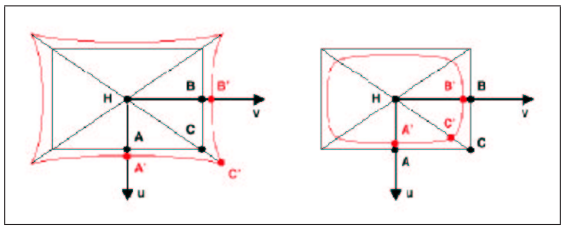

Abbildung 5.7: Auswirkung radialer Verzerrung. In Abhängigkeit der verwendeten Linse treten kissenförmige (links) oder fassförmige (rechts) Verzerrungen auf [29].

Im Vergleich zur radialen Verzerrung ist der Einfluß der tangentialen Verzerrung (vgl. Abbildung 5.8) sehr gering. Obwohl sowohl Tsai als auch Zhang diese Parameter in ihren Verfahren außer Betracht lassen, sollen sie hier trotzdem beschrieben werden. Die tangentiale Verzerrung kann nicht mehr alleine als Funktion des Bildradius beschrieben werden. Zusätzlich werden die Verzeichnungskonstanten p_1 und p_2 direkt mit den Koordinaten des betrachteten Bildpunktes verknüpft [29]:

$$\tilde{x} = x + x(2p_1xy + p_2(r^2 + 2x^2)) \tag{5.12}$$

$$\tilde{y} = y + y(2p_2xy + p_1(r^2 + 2y^2)) \tag{5.13}$$

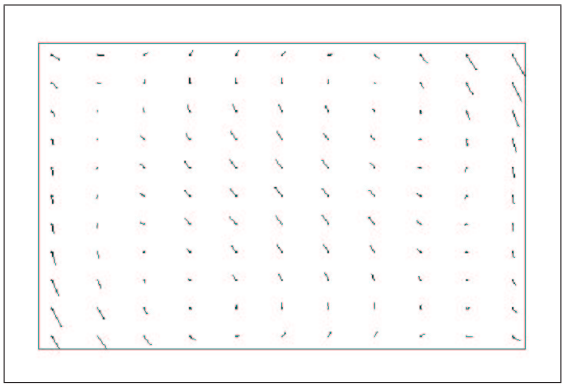

Abbildung 5.8: Auswirkung tangentialer Verzerrung [19].

Somit werden für die Modellierung des Abbildungsvorganges einer digitalen Kamera folgende Parameter benötigt:

1. Translationsvektor t [3]
2. Rotationsmatrix R [3×3]
3. Projektionsmatrix A [3×3]
4. radiale Verzerrung k [2]
5. tangentiale Verzerrung p [2]

Zur Bestimmung dieser Parameter müssen die verwendeten Kameras kalibriert werden.

5.3 Verfahren zur Kamerakalibrierung

Im Bereich der Photogrammetrie und in letzter Zeit auch im Bereich der Computer-Vision wurden eine Vielzahl von Verfahren zur Kamerakalibrierung entwickelt und veröffentlicht. Einige davon sollen in den nächsten Abschnitten vorgestellt werden.

5.3.1 Photogrammetrische Kalibrierung

Hierbei geschieht das Kalibrieren der Kamera durch die Abbildung eines Objektes (vgl. Abbildung 5.9), dessen dreidimensionalen Abmessungen sehr genau bekannt sind. In den meisten Fällen handelt es sich bei dem Objekt um zwei oder drei orthogonal angeordnete plane Flächen mit einem aufgedruckten Muster [42]. Diese Methode erlaubt eine sehr präzise Bestimmung der Parameter, erfordert aber auch sehr hohe Präzision bei der Fertigung bzw. Vermessung des Kalibrierobjektes und einen sehr aufwendigen Versuchsaufbau [40, 37, 41].

5.3.2 Selbstkalibrierung

Verfahren, die nach dem Prinzip der Selbstkalibrierung arbeiten, benötigen kein Kalibrierobjekt. Durch das Bewegen der Kamera in einer statischen Szene ergeben sich Zusammenhänge zwischen bestimmten Bildpunkten, die es ermöglichen, die nötigen Parameter zu bestimmen. Die Aufnahme einer statischen Szene aus drei verschiedenen Positionen ermöglicht die Bestimmung der intrinsischen und extrinsischen Parameter. Bei diesem Verfahren ist es notwendig, viele der Parameter abzuschätzen, was die mathematische Lösung sehr komplex werden läßt [40, 41, 42].

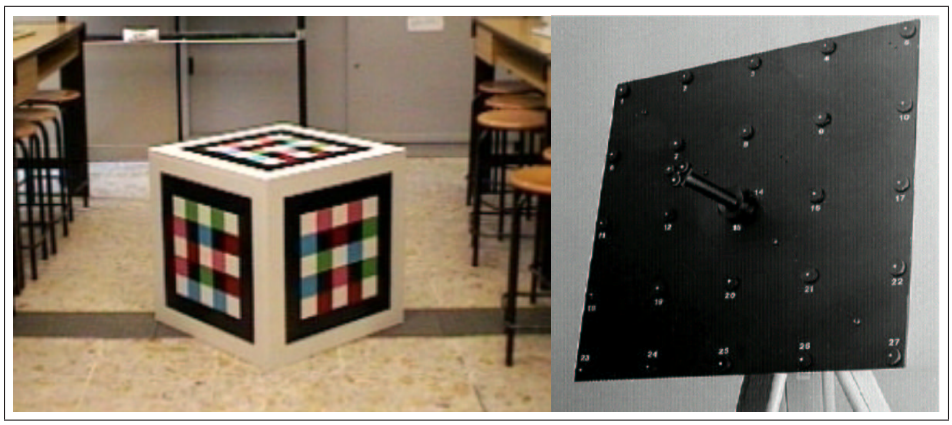

Abbildung 5.9: Kalibrierobjekte für photogrammetrische Kalibrierung [19, 6].

5.3.3 Kalibrierung nach Zhang

Eine weitere Methode zur Kamerakalibrierung wird von Zhengyou Zhang in [40, 41] beschrieben. Dieses Verfahren benutzt ein planares Schachbrettmuster (vgl. Abbildung 5.10) als Kalibrierobjekt, wobei die Berührpunkte der einzelnen Schachfelder als Kalibriermarken genutzt werden. Zur Bestimmung der intrinsischen und extrinsischen Parameter sind mindestens zwei Bilder des Musters mit unterschiedlichen Orientierungen nötig, wobei diese Orientierungen nicht bekannt sein müssen. Im folgenden wird dieses Verfahren beschrieben.

Abbildung 5.10: Schachbrettmuster zur Kamerakalibrierung nach Zhang [40, 41]. Die zur Kalibrierung verwendeten Berührpunkte des Musters (Kalibriermarken) sind grün gekennzeichnet.

Die Bestimmung der Kameraparameter mit dem Verfahren nach Zhang läßt sich in 3 Schritte unterteilen:

1. Abschätzung der Homographie H, einer allgemeinen Abbildung im Raum
2. Bestimmung der intrinsischen und extrinsischen Parameter aus H
3. Modellierung der Distortion und Optimierung der Gesamtlösung

Abschätzung der Homographie H

Es kann angenommen werden, daß das Schachbrettmuster in der $Z = 0$ Ebene liegt, ohne die Allgemeingültigkeit der Gleichungen aufzuheben. Die i-te Spalte der Rotationsmatrix R wird mit r_i bezeichnet. Somit ergibt sich für den Bildpunkt $P' = [u, v, 1]^T$ mit dem zugehörigen Objektpunkt $P = [X, Y, Z, 1]^T$ folgende Gleichung:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A[Rt] \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = A[r_1 r_2 r_3 t] \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = A[r_1 r_2 t] \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}. \tag{5.14}$$

Für die Beziehung zwischen P und P' ergibt sich die Homographie H

$$sP' = HP \qquad mit \qquad H = A[r_1 r_2 t]. \tag{5.15}$$

Mit den Punkten P_{ij} und P'_{ij} als i-te Punkte im Bild j wird ein überbestimmtes Gleichungssystem mit $i \times j$ Gleichungen aufgestellt. Dieses Gleichungssystem wird dann zunächst reduziert und in geschlossener Form gelöst. Da alle Punkt-Koordinaten fehlerbehaftet sind aber nur ein Teil dieser Punkte zur Lösung des Gleichungssystem verwedet wurden, muß die gefundene Homographie anschließend optimiert werden. Dies geschieht durch die Minimierung folgender Fehlerfunktion über alle Punkte P'_{ij}:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \| P'_{ij} - \hat{P}'(H, P_i) \|^2 \tag{5.16}$$

mit $\hat{P}'(H, P_i)$ als Projektion des Punktes P_i in Bild j nach Gleichung (5.15). Die Minimierung erfolgt mit dem Levenberg-Marquardt Algorithmus [27].

Bestimmung der intrinsischen und extrinsischen Parameter aus H

Der zweite Schritt besteht aus zwei Teilschritten. Zuerst werden die Homographie H und die Kameramatrix A in ein Gleichungssystem überführt, mit dem die intrinsischen Parameter geschätzt werden. Aus diesen geschätzten Parametern der Kameramatrix und der zuvor geschätzten Homographie lassen sich dann die extrinsischen Parameter berechnen. Anschließend ist es auch hier notwendig, die gefundenen Parameter durch Minimierung der Fehlerfunktion zu optimieren. Für die Fehlerfunktion ergibt sich hier:

$$\sum_{j=1}^{n}\sum_{i=1}^{m} \|P'_{ij} - \hat{P}'(A, R_j, t_j, P_i)\|^2 \tag{5.17}$$

Als Startwerte für die Optimierung werden die geschätzten Parameter verwendet.

Modellierung der Distortion und Optimierung der Gesamtlösung

Die radiale Distortion wird im Verfahren nach Zhang mit Hilfe der ersten beiden Koeffizienten modelliert (vgl. Kapitel 5.2.2). Die sich daraus ergebende Gesamtlösung wird durch die Minimierung folgender Funktion optimiert

$$\sum_{j=1}^{n}\sum_{i=1}^{m} \|P'_{ij} - \hat{P}'(A, k_1, k_2, R_j, t_j, P_i)\|^2 \tag{5.18}$$

wobei $\hat{P}'(A, k_1, k_2, R_j, t_j, P_i)$ die Projektion des Punktes P_i in Bild j nach Gleichung (5.15) ist, gefolgt von der Distortion mit den beiden Koeffizienten k_1 und k_2. Als Startwerte für A und $\{R_j, t_j | j = 1, \ldots, n\}$ werden die geschätzten und optimierten Werte aus Schritt zwei verwendet. Die Parameter k_1 und k_2 werden zu Beginn der Optimierung gleich 0 gesetzt [40, 41].

5.3.4 Weitere Verfahren zur Kamerakalibrierung

Es existieren noch weitere Verfahren zur Kamerakalibrierung wie zum Beispiel die Kalibrierung durch reine Rotation der Kamera [22] oder die Kalibrierung mit Hilfe von Fluchtpunkten [7]. In dieser Arbeit sollen sie aber nicht näher betrachtet werden.

Kapitel 6

Implementierung – Kamerakalibrierung und Datenfusion

Dieses Kapitel beschreibt das Verfahren zur Fusion der digitalen Farbbilder des Kamerasystems mit den 3D-Abstandsinformationen des Laserscanners. Das Verfahren zur Kamerakalibrierung und seine Implementierung in der Klasse `camera` werden erläutert. Die Erzeugung von Texturen mit OpenGL wird beschrieben und die Umsetzung beim Rendering-Programm des Laserscanners, welches für die Darstellung der Abstandsinformationen auf dem Bildschirm zuständig ist, beschrieben. Anhand von Beispielen wird die Erzeugung von textuierten 3D-Szenen veranschaulicht.

6.1 Verfahren zur Fusion der digitalen Farbbilder mit den 3D-Abstandsinformationen

Zur Gewinnung der digitalen Bildinformationen wurden 2 Kamerasysteme gefertigt und neben dem Laserscanner an der Vorderseite des Roboters montiert (vgl. Abbildung 6.1). Um den gesamten Bereich eines Laserscans abdecken zu können, muß jede Kamera sechs verschiedene Bilder der Umgebung aufnehmen. Diese werden als PPM-Dateien im Datenverzeichniss des jeweiligen Laserscans gespeichert. Für die Fusion der Bilddaten mit den Daten des Laserscanners ist es zunächst notwendig, die Kameras zu kalibrieren. Das hierzu gewählte Verfahren wird im nachfolgenden Abschnitt beschrieben.

Abbildung 6.1: Das Robotersystem KURT3D mit den montierten Kamerasystemen.

6.1.1 Kamerakalibrierung

Als Grundlage dient das von Zhengyou Zhang in „A Flexible New Technique for Camera Calibration“ [40] beschriebene Verfahren zur Kamerakalibrierung, dessen theoretischen und mathematischen Grundlagen bereits in Kapitel 5.3.3 beschrieben wurden. Als Kalibrierobjekt wird ein, im DIN-A1 Format gedrucktes und auf eine 8mm starke Holzplatte aufgeklebtes, Schachbrettmuster (vgl. Abbildung 6.2) verwendet.

Abbildung 6.2: Das Kalibrierobjekt zur Kamerakalibrierung. Die grün markierten Berührpunkte des Schachbrettmusters werden als Kalibriermarken bezeichnet. Die orange markierten Eckpunkte des Kalibrierobjektes werden zur Berechnung der 3D-Koordinaten der Kalibriermarken im Laserscan verwendet.

Der Ablauf der Kamerakalibrierung wird am Beispiel einer der beiden Kameras beschrieben. Zunächst werden die intrinsischen Parameter der Kamera bestimmt. Hierzu werden fünf Bilder des Kalibrierobjektes mit unterschiedlichen Orientierungen aufgenommen. Dabei ist zu beachten, daß die Unterschiede in der Orientierung der einzelnen Bilder nicht nur einen translatorischen Anteil haben, da solche Bilder keine weiteren Informationen für die Kalibrierung liefern [40, 41] und zu Fehlern bei der Optimierung der intrinsischen Parameter führen können. Desweiteren sollte das Kalibrierobjekt einen möglichst großen Bereich des Bildes abdecken, da die für die intrinsischen Parameter wichtigen radialen Verzerrungen im Randbereich des Bildes den größten Einfluß haben und bei der Kalibrierung nur dann berücksichtigt werden, wenn sie sich auf die Kalibriermarken (vgl. Abbildung 5.10) auswirken. Nach der Bestimmung der intrinsischen Parameter ist es notwendig, diese zu evaluieren. Allein anhand der Werte der intrinsischen Parameter ist es schwer eine Aussage über ihre Qualität zu treffen. Eine bessere Methode zur Evaluierung dieser Parameter ist die visuelle Auswertung eines mit Hilfe der intrinsischen Parameter und einer Funktion der OpenCV-Bibliothek (Informationen hierzu finden sich in Kapitel 6.3.1) entzerrten Bildes durch den Benutzer. In Abbildung 6.3 ist zu erkennen, welche Auswirkungen falsche intrinsische Parameter beim Entzerren eines Bildes haben können. Die zugehörigen intrinsischen Parameter werden in Tabelle 6.1 aufgeführt.

Abbildung 6.3: Die Auswirkungen falscher intrinsischer Parameter beim Entzerren werden gezeigt. Das linke Bild wurde mit den falschen und das rechte Bild mit den richtigen Parametern entzerrt.

	linkes Bild	rechtes Bild
Distortion	-0.4306 -1.9363 0.0066 0.0076	-0.4911 1.6565 0.0026 -0.0047
Kamera-Matrix	843.12 0 480.67 0 857.20 448.68 0 0 1	800.72 0 332.80 0 799.22 208.95 0 0 1

Tabelle 6.1: Die zu Abbildung 6.3 gehörenden intrinsischen Parameter

Im nächsten Schritt werden die extrinsischen Parameter bestimmt. Da diese von Lage und Orientierung der Kamera zu den abgebildeten Objekten abhängen, müssen sie für jede der sechs Kamerapositionen einzeln bestimmt werden. Hierzu wird das Kalibrierobjekt jeweils so platziert, daß es sowohl von der Kamera als auch vom Laserscanner gut erfaßt werden kann. Dann wird ein Bild der Szene aufgenommen und anschließend ein 3D-Laserscan gemacht. Dabei ist zu beachten, daß die Position des Kalibrierobjektes und die des Roboters zwischen der Aufnahme des Bildes und der des Laserscans unverändert bleibt. Zur Bestimmung der extrinsischen Parameter werden nun die exakten 3D-Koordinaten der Kalibriermarken mit Hilfe des Laserscans bestimmt. Da bei dem Laserscan nur die äußeren Konturen und nicht das Muster des Kalibrierobjektes erfaßt werden, müssen die Positionen der Kalibriermarken mit Hilfe der Eckpunkte des Kalibrierobjektes (vgl. Abbildung 6.2) berechnet werden. Aus diesem Grund muß darauf geachtet werden, daß diese für den Laserscan genügend Abstand zu dahinter- oder danebenliegenden Objekten haben. Die Bestimmung der Eckpunktkoordinaten geschieht automatisch mit Hilfe eines ICP-Algorithmus der bereits für das Registrieren, das Darstellen verschiedener Scans in einem gemeinsamen Koordinatensystem, verwendet und in [30] ausführlicher beschrieben wird. Als Startwert werden dem Algorithmus die Koordinaten eines in der Mitte des Kalibrierobjektes liegenden Punktes übergeben, woraufhin dieser eine rechteckige Fläche in der Größe des Kalibrierobjektes mit den Daten des Laserscans vergleicht. Die Koordinaten der Eckpunkte des Kalibrierobjektes werden dann automatisch berechnet und ausgegeben. Abbildung 6.4 zeigt eine typische Szene zur Bestimmung der extrinsischen Parameter und die Visualisierung des Scans mit der registrierten Fläche.

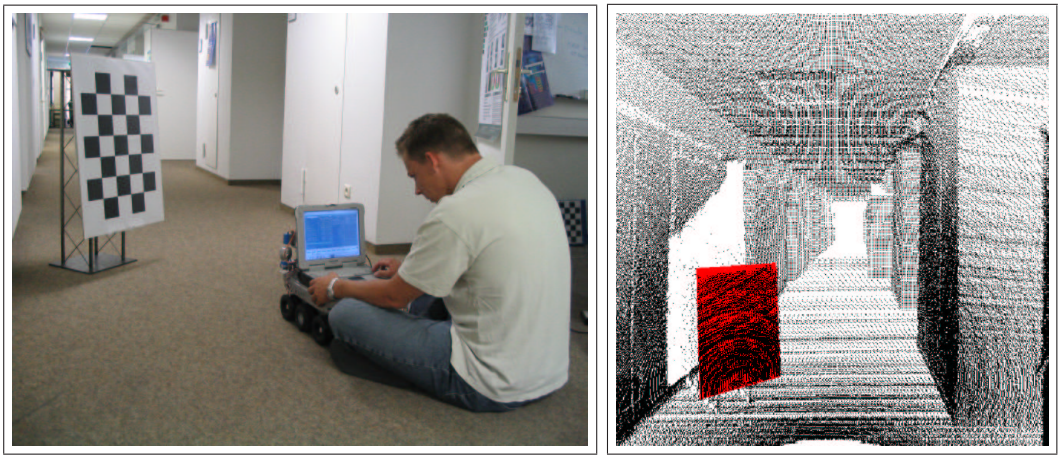

Abbildung 6.4: Das linke Bild zeigt eine typische Kalibrierszene mit Kalibrierobjekt und KURT3D. Im rechten Bild ist die rote, auf das Kalibrierobjekt gematchte Fläche, zu erkennen.

Stabilität der intrinsischen Parameter

Wie schon beschrieben, werden die intrinsischen Parameter durch den internen Aufbau der Kamera und die Beschaffenheit der Linse bestimmt. Bei den hier verwendeten Kameras wird die Bildschärfe durch ein drehbares Objektiv, welches den Abstand zwischen Linse und CCD-Sensor variiert, eingestellt. Jede Bewegung des Objektivs führt somit zu einer Veränderung des internen Aufbaus der Kamera. Um eine Neukalibrierung der intrinsischen Parameter nach einer ungewollten Bewegung des Kameraobjektives zu umgehen, wurde die Schärfe des Kamerabildes manuell auf das in 4m Entfernung positionierte Kalibrierobjekt eingestellt und die Position des Objektives an der Kamera markiert. Somit ist es möglich, das Objektiv jederzeit auf die kalibrierte Position zurück zu stellen, wodurch eine erneute Kalibrierung der intrinsischen Parameter unnötig wird. Zur Evaluierung dieses Verfahrens wurde eine Testreihe aufgenommen, bei der zunächst die intrinsischen Parameter für die Kalibrierposition des Objektivs bestimmt wurden. Anschließend wurde das Objektiv mehrmals verdreht und dann zurück auf die Kalibrierposition gestellt. Dann wurden die Parameter erneut bestimmt. Dieser Vorgang wurde noch einmal wiederholt. Die sich daraus ergebenden Parameter werden in folgender Tabelle gezeigt:

	Parameter 1	**Parameter 2**	**Parameter 3**
Focuslänge	781.41 782.62	785.76 785.26	788.93 789.02
principal Point	312.01 188.12	322.75 188.24	324.69 192.76
Distortion	-0.3622 0.1818 -0.0009 -0.0016	-0.3609 0.1188 0.0010 -0.0013	-0.3585 0.1200 0.0007 -0.0010

Tabelle 6.2: Gegenüberstellung der intrinsischen Parameter zur Evaluierung der Stabilität. Die Abweichungen der Parameter sind minimal. Sie können als stabil betrachtet werden.

Die Ergebnisse zeigen, daß die Werte nur minimal voneinander abweichen. Damit können die intrinsischen Parameter als stabil angenommen werden, wenn die Position des Objektives manuell auf die Kalibrierposition eingestellt wird.

Genauigkeit der extrinsischen Parameter

Die Genauigkeit der extrinsischen Parameter kann von verschiedenen Faktoren beeinflußt werden. Zum einen ist das die Wiederholgenauigkeit der Schwenk-Nick-Vorrichtung, also die Wiederholgenauigkeit der Servomotoren und zum anderen ist das die Genauigkeit, mit der die Lage des Kalibrierobjektes im Raum für die entsprechende Kameraposition bestimmt werden kann.

Die verwendeten Servomotoren haben laut Hersteller ein Auflösungsvermögen von 3μs. Ihre Ansteuerung erfolgt mit maximaler Auflösung, also mit 3μs pro Schritt, bei einer maximalen Ungenauigkeit des Signals von 0,1μs ergibt sich eine maximale Positionier-Ungenauigkeit von 1 Schritt, was einem Winkel von 0,19° entspricht. Eine weitere Ungenauigkeit entsteht durch das interne Lagerspiel des Servomotors. Diese Ungenauigkeiten können zu Positionierfehlern führen, wodurch es passieren kann, daß ein während eines Laserscans aufgenommenes Bild nicht exakt zu den extrinsischen Parametern dieser Kameraposition paßt.

Die Lage des Kalibrierobjektes wird mit Hilfe des 3D-Laserscans bestimmt. Durch die endliche Auflösung des Laserscanners kommt es beim scannen des Kalibrierobjektes zu Ungenauigkeiten an den Rändern des Objektes. Dies führt dazu, daß die Maße des Kalibrierobjektes im Laserscan von den wirklichen Maßen abweichen. Durch das Matchen einer Fläche in der Größe des Kalibrierobjektes mit den 3D-Laserdaten kann es nun passieren, daß die Lage sowohl in horizontaler als auch in vertikaler Richtung von der realen Lage abweicht. Dadurch werden ungenaue extrinsische Parameter bestimmt, welche die Texturen später verschoben auf den 3D-Abstandsinformationen abbilden können.

Durch die Bestimmung der intrinsischen und extrinsischen Parameter ist es nun mit Hilfe des in Kapitel 5.1 aufgestellten mathematischen Modells möglich, von einem dreidimensionalen Punkt im Raum auf seine Koordinaten in der Abbildung zu schließen. Dabei muß dem Punkt allerdings die Kameraposition, die ihn abbildet, zugeordnet werden können. Nur so können die passenden extrinsischen Parameter und die richtigen Bildinformationen verwendet werden. Im nächsten Abschnitt wird diese Zuordnung erläutert.

6.1.2 Bestimmung der richtigen Texturen

Um einem Punkt im Raum die richtigen Bildinformationen zuordnen zu können, werden zunächst für jede der zwölf Kamerapositionen die Bildkoordinaten des Punktes berechnet. Wurde der Raumpunkt durch die entsprechende Kameraposition nicht erfaßt, liegen die Bildkoordinaten außerhalb des Bildes, also außerhalb des Bereiches von 0 bis 640 bzw. 480. Wie in Abbildung 6.5 zu erkennen ist, würde der rot markierte Punkt nur für die Kameraposition A1 gültige Bildkoordinaten liefern und somit dieser Kameraposition zugeordnet. Durch Überschneidungen an den Bildrändern kommt es beim grün markierten Punkt allerdings dazu, daß die Positionen A3 und B3 gültige Bildkoordinaten liefern. In diesem Fall wird der Abstand des jeweiligen Bildpunktes zu seiner Bildmitte berechnet und die Bildinformationen verwendet, deren Abstand zur Mitte geringer ist.

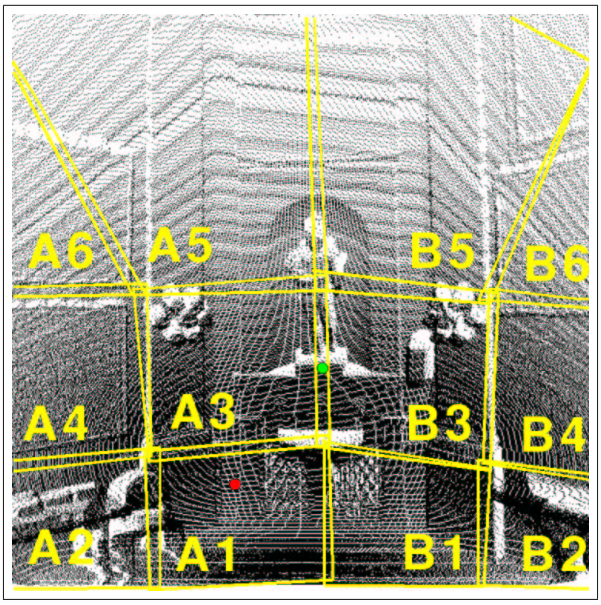

Abbildung 6.5: Das Bild zeigt einen 3D-Laserscan. Die gelb umrandeten Flächen zeigen die Lage der Bilddaten der verschiedenen Kamerapositionen im Laserscan. (A1-A6 = linke Kamera; B1-B6 = rechte Kamera)

Somit sind alle Informationen vorhanden, um die 3D-Abstandsinformationen des Laserscanners mit den Bilddaten zu fusionieren.

6.1.3 Texturen mit OpenGL

Die Darstellung der 3D-Abstandsinformationen auf dem Bildschirm geschieht mit einem auf der offenen Grafikbibliothek `OpenGL` basierenden Rendering-Programm. Eine detaillierte Beschreibung dieser Software findet sich in [30, 36]. In diesem Rendering-Programm stehen verschiedene Anzeige-Modi zur Verfügung, die in drei Kategorien unterteilt werden können (vgl. Abbildung 6.6):

- Anzeige aller Meßdaten als Punktwolke
- Anzeige einer reduzierten Anzahl von Meßdaten als Rechtecke
- Anzeige der Meßdaten als Octalbaum.

Für die realistische Darstellung einer textuierten 3D-Szene ist es sinnvoll, einen Anzeige-Modus zu wählen, der die Szene als möglichst dichte Fläche darstellt. Hierzu eignet sich die Darstellung als Octalbaum. Wie in Abbildung 6.6 zu erkennen ist, wird hierbei die Szene als geschlossene Aneinanderreihung von Würfeln dargestellt.

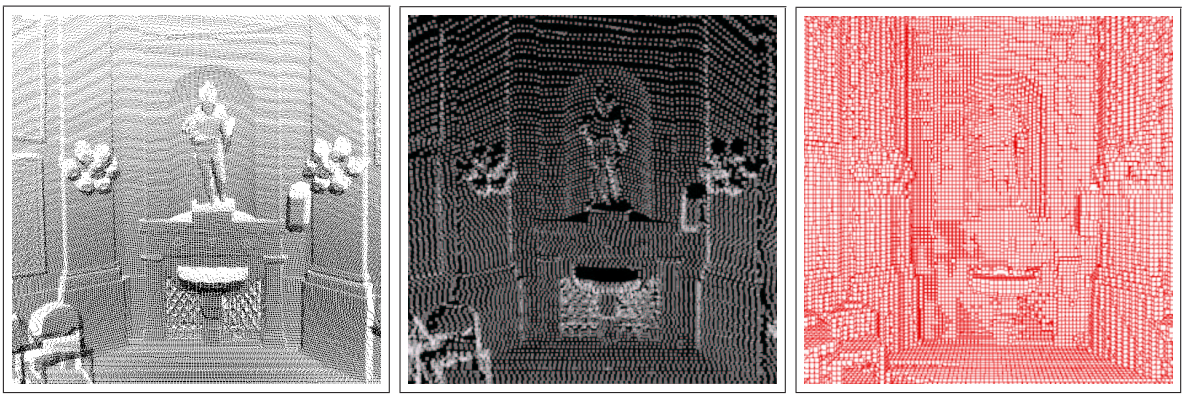

Abbildung 6.6: Die verschiedenen Anzeige-Modi der Rendering-Software. Links die Punktwolke, in der Mitte die reduzierten Punkte als Rechtecke und rechts die Darstellung als Octalbaum.

Die Graphikbibliothek `OpenGL` bietet die Möglichkeit, dargestellte Flächen mit Texturdaten zu „bekleben“, wodurch die realistische Darstellung einer 3D-Szene möglich wird. Die Erzeugung und Darstellung von Texturen mit OpenGL läßt sich in vier Schritten beschreiben:

1. Erzeugung eines Texturobjektes und Spezifizierung einer bestimmten Textur für dieses Objekt
2. Festlegung der Art der Textuierung
3. Aktivierung der Textur
4. Zeichnen der 3D-Szene mit Textur.

Im nächsten Abschnitt werden die einzelnen Schritte und deren Umsetzung im Rendering-Programm beschrieben [9].

Erzeugung eines Texturobjektes und Spezifizierung der Textur

Für die textuierte Darstellung eines Laserscans werden die Bildinformationen aus zwölf verschiedenen digitalen Bildern verwendet. OpenGL erlaubt es, mehrere Texturobjekte zu erstellen und diese dann verschiedenen Bereichen einer Szene zuzuordnen. Ein zweidimensionales Texturobjekt wird durch die OpenGL-Funktion

```
void glTexImage2D(GLenum target, GLint level, GLint internalFormat,
                        GLsizei width, GLsizei height, GLint border,
                    GLenum format, GLenum type, const GLvoid *pixels);
```

erzeugt. Die einzelnen Argumente und ihre Bedeutung werden im folgenden beschrieben:

GLenum target	Es können die zwei Konstanten `GL_TEXTURE_2D` oder `GL_PROXY_TEXTURE_2D` verwendet werden. Im ersten Fall wird die Textur in jedem Fall übergeben und im zweiten wird zunächst geprüft, ob noch genügend Speicher für eine Textur entsprechender Größe vorhanden ist.
GLint level	Dieses Argument wird nur benötigt, wenn verschiedene Auflösungen einer Textur verwendet werden. Ansonsten wird es auf 0 gesetzt.

GLint internalFormat	Die Bilddaten eines 2D-Texturobjektes werden normalerweise in einem zweidimensionalen Feld gespeichert, in welchem für jeden Bildpunkt ein R-, G-, B- und A-Wert hinterlegt wird. Ein solcher Bildpunkt wird auch als Texel bezeichnet. Die R-, G- und B-Werte beschreiben den Rot-, Grün- bzw. Blau-Anteil des Texels. Mit A wird der Alpha-Wert, also die Transparenz des Texels bestimmt. Das Argument `internalFormat` beschreibt, in welcher Form die einzelnen Werte bei der Darstellung der Textur berücksichtigt werden.
GLsizei width, height	Sie beschreiben die Größe des Feldes mit den Bilddaten der Textur. Sie müssen immer ein ganzes Vielfaches von 2 sein, also z.B. 256 × 256 oder 512 × 1024. Für den Fall, daß ein Rahmen um die Textur gezeichnet werden soll, muß dessen Breite noch addiert werden.
GLint border	Um die Textur kann ein Rahmen gezeichnet werden, dessen Breite in Pixeln hier übergeben wird. Wenn kein Rahmen gezeichnet werden soll, wird `border` gleich 0 gesetzt.
GLenum format, type	Mit `format` wird das Format der Bilddaten übergeben. Für den Fall, daß für die Texturen Bildinfomrationen einer Kamera verwendet werden, handelt es sich dabei um Daten im RGB- oder RGBA-Format. Das Argument `type` gibt an, in welchem Format die einzelnen RGB- oder RGBA-Werte im Texturdaten-Feld abgelegt sind.
GLvoid *pixels	Hier wird der Zeiger auf das Feld mit den Bildinformationen der Textur übergeben. Dadurch wird bestimmt, welche Bilddaten welchem Texturobjekt zugeordnet werden.

Zur Erzeugung eines solchen Texturobjektes müssen aber zunächst die Bilddaten in ein entsprechendes zweidimensionales Feld der Form

```
unsigned char *texImage[12];

for (int i = 0; i < 12; i++){
  texImage[i] = new unsigned char[512 * 512 * 4];
}
```

eingelesen werden. Dies geschieht in der Funktion `readPPM()`, die als Argumente den Pfad des entsprechenden Bildes im Datenverzeichnis des Scans und den Zeiger auf das `texImage[i]` übergeben bekommt. Die Bilder des Kamerasystems haben eine Größe von 640×480 Pixeln. Da aber die Größe des Feldes zur Speicherung der Texturdaten immer ein ganzes Vielfaches von 2 sein muß, würde ein Feld der Größe 512×1024 benötigt und somit über 40% des für die Texturen bereitgestellten Speicherplatzes nicht verwendet. Um dies zu verhindern, werden die Bilder vor

dem Einlesen auf eine Größe von 512×512 skaliert. Dadurch gehen zwar Bildinformationen verloren, aber dies wird in Kauf genommen, da der Verlust sehr gring ist und der Bedarf an Speicher für die Texturdaten somit pro Scan um 50% gesenkt wird. Die Bilder des Kamerasystems liegen im PPM-Format und somit im RGB-Modus vor. Beim Einlesen des Bildes wird der Alphawert für jeden Texel auf den maximalen Wert gesetzt, was einer maximalen Deckung entspricht. Das folgende Listing zeigt den Einlesevorgang der Funktion `readPPM()`:

```
sizePPMimage = fread(ppmimage, sizeof(unsigned char), texImageSize, fptr);
  if (sizePPMimage <= texImageSize) {
    for (index = 0; index < sizePPMimage; index += 3) {
      y = abs((index / (w*3)) - h + 1);
      x = (index % (w*3)) / 3;
      texImage[(y*512+x)*4+0] = (unsigned char) ppmimage[index + 0];
      texImage[(y*512+x)*4+1] = (unsigned char) ppmimage[index + 1];
      texImage[(y*512+x)*4+2] = (unsigned char) ppmimage[index + 2];
      texImage[(y*512+x)*4+3] = (unsigned char) 255;
    }
  }
}
```

Somit ergibt sich für die Erzeugung der zwölf Texturobjekte folgender Aufruf:

```
for (int i = 0; i < 12; i++) {
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 512, 512, 0, GL_RGBA,
               GL_UNSIGNED_BYTE, (GLubyte*) texImage[i]);
}
```

Festlegung der Art der Textuierung

Um die Art der Textuierung festzulegen, müssen einige Parameter eingestellt und Variablen gesetzt werden. Die im Rendering-Programm gewählten Einstellungen und ihre Bedeutung werden in der folgenden Tabelle dargestellt:

Parameter	Einstellung	Beschreibung
GL_TEXTURE_ENV_MODE	GL_DECAL	Dieser Parameter bestimmt, wie die Farben der Fläche, auf welche die Textur aufgebracht werden soll, mit den Farben der Textur vermischt werden. Mit der Einstellung GL_DECAL wird die Textur auf die Fläche „geklebt“, so daß die darunterliegenden Farben keinen Einfluß mehr haben.
GL_TEXTURE_WRAP_S GL_TEXTURE_WRAP_T	GL_CLAMP	Hiermit wird festgelegt, was passiert, wenn die Fläche größer ist als die gewählte Textur. Mit GL_CLAMP wird die Textur nicht als Kachelmuster wiederholt, sondern nur einmal dargestellt.
GL_TEXTURE_MAG_FILTER GL_TEXTURE_MIN_FILTER	GL_NEAREST	Diese Parameter entsprechen dem Antialiasing-Filter aus der Bildverarbeitung. Mit der Einstellung GL_NEAREST wird ein Filter gewählt, der wenig Rechenzeit kostet.

Tabelle 6.3: Parameter und deren Bedeutung zur Art der Texturdarstellung.

Aktivierung der Textur

Im Rendering-Programm des Laserscanners können die Texturen mit der Taste `t` ein- bzw. ausgeschaltet werden. Damit `OpenGL` die Texturen bei der Darstellung einer 3D-Szene berücksichtigt, müssen sie aktiviert werden. Durch drücken der Taste `t` wird die Variable `show_texture` umgeschaltet und je nach Zustand wird die Darstellung der Texturen mit

```
if (show_texture == 1) {
  glEnable(GL_TEXTURE_2D);
  } else {
    glDisable(GL_TEXTURE_2D);
  }
}
```

aktiviert bzw. deaktiviert.

Zeichnen der 3D-Szene mit Textur

Um eine Szene als textuierten Octalbaum darstellen zu können, muß jeder Fläche eines Würfels eine Textur zugeordnet werden. Zunächst muß bestimmt werden, welches der zwölf Texturobjekte für diese Fläche verwendet werden soll. Hierzu werden, wie in Kapitel 6.1.2 beschrieben, die Bildkoordinaten für den ersten Punkt der Fläche berechnet und die passende Kameraposition und somit das Texturobjekt, bestimmt. Der Aufruf

```
glBindTexture(GL_TEXTURE_2D, texName[i]);
```

bestimmt, welches Texturobjekt im folgenden als Quelle der Texturen verwendet wird. Jede Würfelfläche des Octalbaumes wird durch seine vier Eckpunkte bestimmt und mit einer eigenen Textur „beklebt“. Hierzu müssen jedem der Eckpunkte die entsprechenden Koordinaten im Texturobjekt zugeordnet werden. Für die Zuordnung werden diese Koordinaten auf den Wert 1.0 skaliert. Wie aus Abbildung 6.7 hervorgeht, wird im Texturobjekt die linke untere Ecke einer Textur mit (0,0) angeben. Die Bilddaten einer PPM-Datei bezeichnen allerdings die linke obere Ecke mit (0,0), weshalb der y-Wert an der Mittelwaagerechten gespiegelt werden muß. Für die Skalierung der Texturkoordinaten ergibt sich:

$$x_{neu} = \frac{x}{640} \tag{6.1}$$

$$y_{neu} = -1\frac{y-480}{480}. \tag{6.2}$$

Die Berechnung der Bildkoordinaten bezieht sich auf die Originalgröße der Bilddaten, weshalb hier die Werte 640 und 480 anstatt der Texturgröße 512×512 verwendet werden. Mit

```
glTexCoord2f(TexCoord_x, TexCoord_y);
```

werden nun jedem der Eckpunkte die entsprechenden Texturkoordinaten zugewiesen.

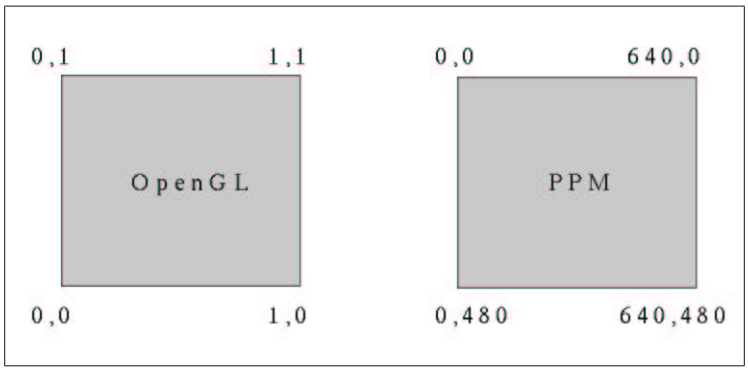

Abbildung 6.7: Bezeichnung der Koordinaten bei Texturobjekten und Bilddaten im PPM-Format.

6.2 Anwendungsbeispiele

Das in Kapitel 6.1 beschriebene Verfahren zur Fusion der digitalen Farbbilder mit den 3D-Abstandsinformationen ist in der Lage, eine virtuelle Darstellung einer realen Szene zu erzeugen. Durch die Ungenauigkeit der extrinsischen Parameter kann es zu Verschiebungen der einzelnen Bilder in der Szene kommen, so daß die Texturen nicht exakt an den richtigen Stellen eingebunden werden. Eine weitere Schwierigkeit ist die Aufnahme der digitalen Farbbilder. Normalerweise arbeitet der Linux Device-Treiber der Kameras mit einer automatischen Anpassung von Helligkeit, Farbintensität und Belichtungszeit, was bedeutet, daß vor jeder Aufnahme eines Bildes diese Parameter neu ermittelt werden. Bei dieser Art von Kamera kann dies zu teilweise starken Unterschieden in der Helligkeit und vor allem auch in der Farbgebung führen. Daraus können sich in einer Szene stark sichtbare Übergänge zwischen den einzelnen Bildern ergeben.

Abbildung 6.8: Ungenauigkeiten bei der Darstellung von textuierten 3D-Szenen. Das linke Bild zeigt den Versatz an den Bildübergängen. Im rechten Bild sind die unterschiedlichen Farbwerte einzelner Bilder zu erkennen.

6.2.1 Textuierte 3D-Szenen

Zur Erzeugung dieser Szenen wurde ein Dreiecks-Netz (triangle-mesh) verwendet, welches eine Fläche zwischen drei benachbarten Punkten im Laserscan aufspannt. Die im Bild erkennbaren schwarzen Linien bzw. Flächen entstehen dadurch, daß zwischen Punkten mit zu großen Abständen in Z-Richtung keine Fläche erzeugt wird und somit auch keine Texturen gezeigt werden können. Auf der linken Seite werden die Szenen aus der Roboterperspektive gezeigt und auf der rechten aus einer leicht erhöhten Perspektive. Sobald die Roboterperspektive verlassen wird, vergrößern sich die Flächen ohne Textur, da diese bei der Aufnahme für die Kameras im „Schatten“ lagen.

In Abbildung 6.9 wird die textuierte 3D-Szene des großen Saales im Schloß Birlinghoven gezeigt. Die schwarzen Linien der linken Bilder zeigen die wirklichen Konturen der Objekte im 3D-Laserscan. Es ist gut zu erkennen, daß z.B. die Texturen der Statue und die der Leuchter nur sehr leicht verschoben sind. Mit der Rendering-Software ist es nun möglich, diese Szene aus verschiedenen Blickwinkeln zu betrachten.

Abbildung 6.9: Textuierte 3D-Szene des großen Saales im Schloß Birlinghoven.

Die Abbildung 6.10 zeigt eine textuierte 3D-Szene des grünen Saales im Schloß Birlinghoven. Aus der Position des Roboters sind nur die Unterseiten der Tische zu erkennen, weshalb die Oberseiten der Tische in den beiden rechten Bildern nicht angezeigt werden können.

Abbildung 6.10: Textuierte 3D-Szene des grünen Saales im Schloß Birlinghoven.

Die folgende Abbildung zeigt den Gesamtüberblick der beiden eben gezeigten 3D-Szenen. Die weißen Flächen zeigen Bereiche, für die 3D-Abstandsinformationen vorhanden sind, aber keine Texturdaten existieren. Die Größe und Form dieser Flächen hängt von der Geometrie der Szene ab.

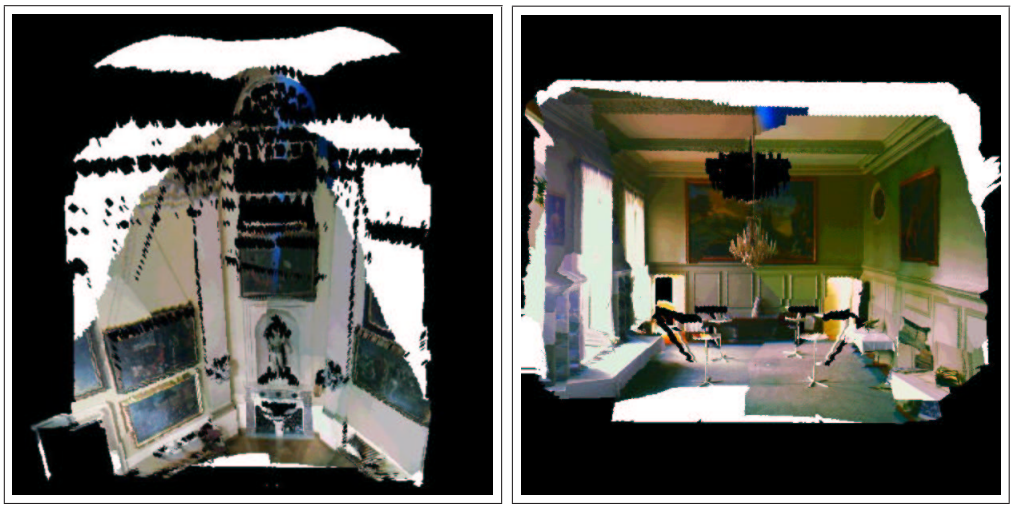

Abbildung 6.11: Gesamtüberblick der beiden textuierten Szenen. Die weißen Flächen zeigen Bereiche der Szene, für die 3D-Abstandsinformationen vorliegen, aber keine Texturdaten.

Mit der Rendering-Software ist es nicht nur möglich die erzeugten 3D-Szenen aus verschiedenen Blickwinkeln zu betrachten, sondern zusätzlich können diese Bewegungen auch animiert und in einem Film dargestellt werden. Einige dieser Filme von verschiedenen 3D-Szenen finden sich auf der beigefügten CD.

6.3 Die Klasse `Camera`

Die Implementierung des in Kapitel 6.1 beschriebenen Verfahrens zur Kamerakalibrierung erfolgte in der Klasse `Camera`. Diese Klasse basiert auf „Open-Source Computer-Vision-Library“ (OpenCV) von Intel.

6.3.1 Die OpenCV-Bibliothek

Bei der „Open-Source Computer-Vision-Library“ handelt es sich um eine freie Bibliothek für den Bereich Computer-Vision. Sie wurde sowohl für Linux-, als auch für Windows-Betriebssysteme entwickelt und erstmals im Jahr 2000 veröffentlicht. Mit dieser Bibliothek werden eine Reihe von Funktionen für Bereiche wie zum Beispiel Mensch-Maschinen-Kommunikation, Objekterkennung oder mobile Robotik zur Verfügung gestellt. Alle Funktionen liegen im Quellcode (ANSI C) vor und können frei verwendet werden. Die OpenCV-Bibliothek ist eng mit der „Image Processing Library“ (IPL) verknüpft und benutzt unter anderem das IPL-Format `IplImage` zur Darstellung von Bilddaten. Die Entwicklung zielte vor allem auf Echtzeitanwendungen im Bereich der Computer-Vision ab. Bei der Implementierung des Verfahrens zur Kamerakalibrierung werden Funktionen aus dem Bereich „3D Rekonststruction“ der OpenCV-Bibliothek verwendet [24, 34].

6.3.2 Funktionen der Klasse `Camera`

Bei der Erzeugung einer Instanz der Klasse `Camera` wird die zu kalibrierende Kamera, die Anzahl der Kamerapositionen und die Anzahl der Bilder zur Bestimmung der intrinsischen Parameter festgelegt. Mit der `public` Klassenfunktion

```
int Camera::calibrateCamera(int option){
  int currPos;
  char okay;
  int siZe = (numImages>maxPositions?numImages:maxPositions);

  imagePoints  = new CvPoint2D64d[patternCount * numImages * sizeof(CvPoint2D64d)];
  objectPoints  = new CvPoint3D64d[patternCount * siZe * sizeof(CvPoint3D64d)];

  do{
    acquireImages(0);
    intrinsicCameraParameters(0);
    std::cout << "\n\n*****************************" << std::endl;
    std::cout << " intrinsic Parameters  okay?" << std::endl;
    std::cout << "*****************************" << std::endl;
    std::cout << "(y/n): " << std::flush;
```

```
    if (validation) {
     std::cin >> okay;
     std::cin.ignore();
    } else okay = 'y';
  }while (okay != 'y');

  acquireImages(1);
  if (extrinsicParams){
    for(currPos=0; currPos<maxPositions; currPos++){
      extrinsicCameraParameters(currPos);
    }
    generateCalibrationFile();
  }

  delete(imagePoints);
  delete(objectPoints);

  return 0;
}
```

wird die Kamerakalibrierung gestartet. Zunächst wird die Funktion `acquireImages()` aufgerufen, um die Bilder zur Bestimmung der internen Parameter aufzunehmen. Nach jedem Bild wird mit der OpenCV-Funktion `cvFindChessBoardCornerGuesses()` überprüft, ob alle Kalibriermarken im Bild erkannt werden. Wenn dies nicht der Fall ist, wird ein neues Bild aufgenommen, bis die gewünschte Anzahl Bilder erreicht ist. Im Anschluß wird die Funktion

```
int Camera::intrinsicCameraParameters(int currPosition){
  _transVect   =   new double[3 * numImages];
  _rotMatr     =   new double[3 * 3 * numImages];
  _cameraMatrix =   new double[3*3];
  _distortion   =   new double[4];

  acquireObjectPoints(currPosition, "vertex.ideal.pts");
  acquireImagePoints(currPosition, 0);

  int numPoints[numImages];
  for(int currIm=0; currIm<(numImages); currIm++){
    numPoints[currIm] = patternCount;
  }
  useIntrinsicGuess = 0;
```

```
  cvCalibrateCamera_64d(  numImages, numPoints, imageSize, imagePoints, objectPoints,
 distortion, cameraMatrix, _transVect, _rotMatr, useIntrinsicGuess );

  focalLength[0] = cameraMatrix[0];
  focalLength[1] = cameraMatrix[4];
  principalPoint.x = cameraMatrix[2];
  principalPoint.y = cameraMatrix[5];

  showCorrDistortion(currPosition);

  delete(_transVect);
  delete(_rotMatr);
  delete(_cameraMatrix);
  delete(_distortion);

  return 0;
}
```

aufgerufen. Für die Bestimmung der intrinsischen Parameter ist es notwendig, die geometrische Anordnung der Kalibriermarken zu kennen. Diese wird in der Funktion `acquireObjektPoints()` aus der Datei „vertex.ideal.pts“ eingelesen und in `objectPoints[]` hinterlegt. Anschließend werden in der Funktion `acquireImagePoints()` die Bildkoordinaten der Kalibriermarken in den einzelnen Bildern bestimmt und in `imagePoints[]` hinterlegt. Zur Bestimmung dieser Bildkoordinaten werden die OpenCV-Funktionen `cvFindChessBoardCornerGuesses()` und `cvFindCornerSubPix()` verwendet. Die beiden Felder mit den Objekt- und den Bildkoordinaten werden nun zur Bestimmung der intrinsischen Parameter an die Funktion

```
cvCalibrateCamera_64d();
```

übergeben. Zur Evaluierung der gefundenen intrinsischen Parameter wird nun mit der Funktion `showCorrDistortion()` eines der Bilder unverzerrt angezeigt. Werden die Parameter durch den Benutzer bestätigt, werden sie im globalen Feld `cameraMatrix[]` hinterlegt, ansonsten beginnt die Bestimmung der intrinsischen Parameter von neuem. Als nächstes wird die erste Kameraposition angefahren und ein Bild aufgenommen. Diese Aufnahme wird so lange wiederholt, bis alle Kalibriermarken im Bild erkannt werden. Der Benutzer wird nun aufgefordert, einen Laserscan der Szene zu machen. Dieser Vorgang wird für jede Kameraposition wiederholt. Zur Bestimmung der extrinsischen Parameter wird für jede der Kamerapositionen die Funktion

```
int Camera::extrinsicCameraParameters(int currPosition){
  _transVect    =    new double[3];
  _rotVect      =    new double[3];
  _rotMatr  =    new double[3*3];
  int numPoints =    patternCount;

  sprintf(tmpString, "vertex%d.pts", (device+1));
  acquireObjectPoints(currPosition, tmpString);
  acquireImagePoints(currPosition, 1);

  cvFindExtrinsicCameraParams_64d( numPoints, imageSize, imagePoints, objectPoints,
    focalLength, principalPoint, distortion, _rotVect, _transVect);

  Rodrigues(_rotMatr, _rotVect, 0, CV_RODRIGUES_V2M);

  for (int parm=0; parm<3; parm++){
    transVects[(3*currPosition)+parm] = _transVect[parm];
  }
  for (int parm=0; parm<9; parm++){
    rotMatrs[(9*currPosition)+parm] = _rotMatr[parm];
  }

  delete(_transVect);
  delete(_rotMatr);
  delete(_rotVect);

  return 0;
}
```

aufgerufen. Hier werden zunächst die vorher mit Hilfe des Laserscans bestimmten und in einer Datei gespeicherten 3D-Koordinaten der Kalibriermarken eingelesen und in `objectPoints[]` hinterlegt. Anschließend werden die Bildkoordinaten der Kalibriermarken des entsprechenden Bildes bestimmt und in `imagePoints[]` hinterlegt. Die OpenCV-Funktion

```
cvFindExtrinsicCameraParams_64d();
```

kann nun mit den Bild- und Objektkoordinaten der Kalibriermarken und den intrinsischen Parametern die extrinsischen Parameter der aktuellen Kameraposition bestimmen. Der Translationsvektor t wird im globalen Feld `transVects[]` hinterlegt. Für die Rotation liefert die Funktion einen Rodriguez-Rotationsvektor, dessen Richtung die Achse der Rotation darstellt. Der Rotationswinkel um diese Achse wird durch die Länge des Vektors repräsentiert [17]. Mit Hilfe der

Funktion `Rodrigues()` wird dieser Vektor zu einer 3×3 Rotationsmatrix konvertiert und im globalen Feld `rotMatrs[]` hinterlegt wird.

Wenn dies für alle Kamerapositionen durchgeführt wurde, ist die Kamerakalibrierung abgeschlossen. Mit der Funktion:

```
int generateCalibrationFile();
```

werden alle Parameter in einer Datei, die von dem Rendering-Programm des Laserscanners eingelesen wird, gespeichert. Das Format der Datei lehnt sich an den XML-Standard an und wird durch den folgenden Ausschnitt veranschaulicht:

```
#intrinsic and extrinsic parameters for camera 1
#data directory: /home/camera/dat_schloss05/
<camera1>
  <cameraMatrix>
  781.87 0 298.33 0 782.83 197.06 0 0 1
  </cameraMatrix>
  <distortion>
  -0.3607 0.1822 0.0001 -0.0015
  </distortion>
  <position 0>
    <rotMatrix 0>
    0.9573 -0.0552 -0.2836 -0.0408 -0.9975 0.0562 -0.2860 -0.0422 -0.9572
    </rotMatrix>
    <transVector 0>
    18.0184 26.4213 -3.4441
    </transVector>
    <vertex 0>
    -139 80.2 -210
    -134 -3.28 -195
    -92.1 -8.46 -237
    </vertex>
  </position>
  <position 1>
    <rotMatrix 1>
      :
      :
      :
  </position>
</camera1>
```

Kapitel 7

Zusammenfassung und Ausblick

Das in dieser Arbeit aufgebaute System zur Fusion von 3D-Abstandsinformationen mit digitalen Farbbildern ist eine sehr komplexe Kombination verschiedener Disziplinen der Ingenieurwissenschaften. Angefangen bei der mechanischen Konstruktion und Fertigung, über die Integration der Elektronik, die Entwicklung von Verfahren zur 3D-Rekonstruktion und zur Erzeugung von Texturen bis hin zur Implementierung dieser Verfahren unter Verwendung verschiedener Software-Bibliotheken.

Zunächst wurde ein Kamerasystem mit einer Schwenk-Nick-Vorrichtung konstruiert und gefertigt. Als Antriebe der Achsen wurden Servomotoren verwendet, welche durch ein pulsweiten-moduliertes Signal (PWM) angesteuert werden. Zur Erzeugung dieser Signale wurden zwei alternative Systeme entwickelt. Das erste ist eine rein Software-basierte Lösung, welche nur unter der Verwendung eines Echtzeit-Betriebssystems möglich ist. Aus diesem Grund wurde Real-Time Linux verwendet. In einem speziellen echtzeitfähigen Modul wurde die Erzeugung von pulsweiten-modulierten Signalen implementiert. Die zweite realisierte Lösung ist die Verwendung eines zusätzlichen Mikrokontrollers, der die Erzeugung der PWM-Signale übernimmt und über die serielle Schnittstelle angesprochen wird. Das hierzu implementierte Modul übernimmt die Kommunikation über die serielle Schnittstelle und gleicht in der nach außen sichtbaren Funktionalität dem ersten Modul. Um die manuelle Steuerung dieses Kamerasystems zu erleichtern, wurde ein grafisches Benutzer-Interface in der Sprache JAVA realisiert. Durch ein animiertes Modell des Kamerasystems, welches die aktuelle Position der Kamera zeigt, ist die Steuerung sehr intiutiv. Bei der Integration dieses Kamerasystems in die Elektronik des Roboters KURT3D wurde noch ein elektronisches Relais eingebunden, wodurch die Steuerelektronik per Software von der Spannungsversorgung getrennt und damit abgeschaltet werden kann. Dieses Relais kann ebenfalls vom grafischen Benutzer-Interface aus bedient werden. Dieses Kamerasystem wurde in zweifacher Ausführung gefertigt und an der Vorderseite des Roboters KURT3D montiert. Durch die Schwenk-Nick-Vorrichtung war es nun möglich, Bilddaten für den gesamten Bereich vor dem Roboter aufzunehmen.

Um aus diesen Bilddaten Texturen generieren zu können, mußten jedem Punkt im Raum die entsprechenden Bildkoordinaten zugeordnet werden. Dazu war es zunächst notwendig, ein mathematisches Modell aufzustellen, welches die Abbildung eines Objektes mit dem Kamerasystem beschreibt. Hierzu wurden aus der Photogrammetrie bekannte Modelle verwendet und für diesen speziellen Fall angepaßt. Zur Bestimmung der Bildkoordinaten durch Lösung des aufgestellten mathematischen Modells wurden einige kameraspezifische Parameter benötigt. Diese werden zum Teil durch den internen Aufbau der Kamera (intrinsische Parameter) und zum Teil durch Position und Lage der Kamera (extrinsische Parameter) beeinflußt und können durch kalibrieren der Kameras bestimmt werden. Das hierzu verwendete Verfahren basiert auf dem von Zhenyou Zhang in [40, 41] beschriebene Verfahren zur Kalibrierung von Kameras. Die Implementierung dieses Verfahren erfolgte unter Verwendung der freien Bibliothek „OpenCV“.

Der gesamte Bereich eines 3D-Laserscans wird durch die Aufnahme von 2×6 Bildern von den Kamerasystemen erfasst. Durch ein speziell entwickeltes Verfahren konnte jeder Punkt des 3D-Laserscans einem dieser Bilder zugeordnet werden. Durch Lösung des mathematischen Modells zur Bestimmung der Bildkoordinaten konnten somit jedem dieser Punkte die entsprechenden Bildinformationen zugeordnet werden. Die textuierte Darstellung der fusionierten Daten als 3D-Szene erfolgte mit der freien Grafikbibliothek „OpenGL“. Dabei werden Polygonflächen auf dem Bildschirm dargestellt, deren Eckpunkten vorher die entsprechenden Bildkoordinaten zugewiesen wurden. Durch diese Zuweisung werden die entsprechenden Flächen aus den Bilddaten ausgeschnitten und auf die Polygonflächen „geklebt“. Das entwickelte System ist in der Lage, eine reale Szene vollkommen autonom zu erfassen und zu verarbeiten um sie als virtuelle 3D-Szene am Computer darstellen zu können. Bedingung hierfür ist die im voraus einmalig manuell durchgeführte Kalibrierung der Kamerasysteme.

7.1 Ausblick

Durch die Verwendung von kostengünstigen USB-Kameras ist die Qualität der erzeugten virtuellen 3D-Szenen begrenzt. Dennoch können durch einige Optimierungen noch bessere Ergebnisse erzielt werden.
Zum einen hat sich gezeigt, daß die extrinsischen Parameter nicht für alle Kamerapositionen eines Scans optimal sind. Dies kann dazu führen, daß die einzelnen Bereiche der Texturen nicht exakt mit den 3D-Abstandsinformationen übereinstimmen und leicht verschoben dargestellt werden. Durch eine Anpassung der extrinsischen Parameter nach jedem Scan anhand korrespondierender Punkte im Laserscan und im digitalen Farbbild können diese Parameter optimiert werden.
Desweiteren hat sich gezeigt, daß es bei der Aufnahme der Bilder eines Scans zu Unterschieden in der Helligkeit und vor allem in der Farbgebung zwischen einzelnen Bildern kommt. Dies kann zu sichtbaren Bildübergängen in der 3D-Szene führen. Durch das Ausschalten der automatischen Anpassung der Kamerawerte im Device-Treiber kann dies behoben werden. Diese Werte können dann für jeden Scan fest eingestellt werden. Die manuelle Bestimmung der optimalen Werte für einen Scan würde den Grad der Autonomie des Systems verringern, weshalb eine automatische Bestimmmung dieser Werte vor der Aufnahme aller Bilder eines Scans erfolgen sollte.

Anhang A

Herleitungen und Spezifikationen

A.1 USB 1.1 und USB 2.0 Spezifikationen

Der serielle Bus-Standard **U**niversal **S**eriel **B**us (USB) wurde entwickelt um Peripheriegeräte mit Echtzeitdaten, wie z.B. Sprache, mit dem PC verbinden zu können. Im Jahr 1998 wurde die Spezifikation für den USB 1.1-Standard veröffentlicht [26]. Dieser hat eine gestufte Stern-Topologie, bei der jeder Netzknoten (Hub) eine eigene Stufe darstellt. Damit ist es möglich, bis zu 127 Peripheriegeräte anzuschließen. Die Datenübertragung geschieht in einzelnen Frames, welche Daten unterschiedlicher USB-Geräte beinhalten können. Der dadurch entstehende Overhead senkt die effektive Datenübertragungsrate von 12MBit/Sekunde auf ca. 10MBit/Sekunde [31]. Über die Verbindungsleitung des USB können die Peripheriegeräte, welche in Low-Power- und High-Power-Devices unterteilt werden, mit Spannung versorgt werden. Die High-Power-Devices haben eine maximale Dauerstromaufnahme von 500mA, wozu aber im Regelfall ein Hub mit eigener Spannungsversorgung (aktiver Hub) benötigt wird. Die Dauerstromaufnahme der Low-Power-Devices ist auf 100mA begenzt. Eine Skizze mit dem Aufbau eines USB-Kabels findet sich in Abbildung A.1. Um die immer größer werdenden Datenmengen der Multimedia-Devices übertragen zu können wurde im April 2000 die Spezifikation für den USB 2.0 Standard veröffentlicht [28].Dieser ist voll abwärtskompatibel zum USB 1.1 Standard und bietet eine nominale Übertragungsrate von 480MBit/Sekunde [28]. Eine Auflistung der wichtigsten Spezifikationen beider Standards finden sich in Tabelle A.1.

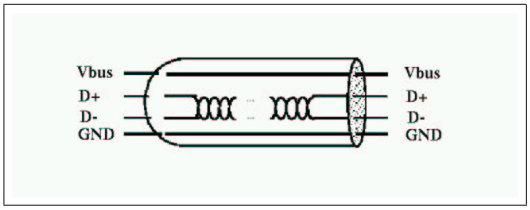

Abbildung A.1: Aufbau eines USB-Verbindungskabels. Die Leitungen D+ und D- werden zur Datenübertragung verwendet, Vbus und GND zur Spannungsversorgung. [26, 28]

	USB 1.1	**USB 2.0**
Echtzeitfähig	ja	ja
Max. Overhead pro Datenpaket	45 Bytes	173 Bytes
Datenübertragung (eff.)	12Mbit/Sekunde (10MBit)	480MBit/Sekunde (400MBit)
Buslänge	Bis 5m (zwischen 2 Geräten). Gesamtlänge bis 30m.	Bis 5m (zwischen 2 Geräten). Gesamtlänge bis 30m.
Max. Anzahl Geräte pro Bus	127	127
Max. Stromaufnahme High-Power-Device Low-Power-Device	 500mA 100mA	 500mA 100mA

Tabelle A.1: Übersicht der Spezifikationen USB 1.1. und USB 2.0 [31].

A.2 Darstellung von Rotationen im Raum

Bei der Rotation mehrerer Punkte im Raum $\mathbb{R}^3$ werden diesen Punkten neue Koordinaten im Raum zugewiesen. Dabei werden alle Punkte um einen konstanten Winkel und ein festes Drehzentrum rotiert. Die Abstände und Winkel zwischen den einzelnen Punkten bleiben dabei erhalten. Solch eine Rotation kann durch Euler-Winkel, Gibb-Vektoren, Caley-Klein-Parameter, Pauli-Spin Matrizen, Hamiltons Quaternionen und orthonormale Matrizen dargestellt werden. Die mobile Robotik verwendet jedoch meistens Euler-Winkel, Quaternionen oder Rotationsmatrizen [30]. Die Berechnung einer Rotationsmatrix mit Hilfe der Euler-Winkel wurde bereits in Kapitel 5.2.1 vorgestellt. Die Darstellung einer Rotation mit Hilfe von Quaternionen wurde von Andreas Nüchter in [30] sehr anschaulich beschrieben und wird im folgenden wiedergegeben:

Das Einheitsquaternion

„Neben der Repräsentation einer Rotation mit Hilfe der Euler Winkel wird das so genannte Einheitsquaternion (engl.: *unit quaternion*) eingesetzt [12]. Das Quaternion beschreibt eine Rotation um einen Vektor, der durch den Ursprung des Koordinatensystems geht. In Abbildung A.2 sind ein Einheitsvektor **n** und ein Winkel θ dargestellt, die die Rotation der beiden Koordinatensysteme beschreiben. Das blaue Koordinatensystem ist das Ergebnis der Drehung des schwarzen Systems um einen Winkel θ.

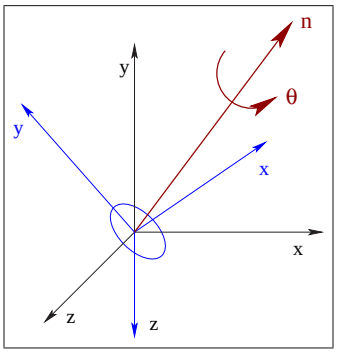

Abbildung A.2: Rotation durch das Einheitsquaternion

Das Quaternion geht auf Hamilton zurück und kann mathematisch als eine komplexe Zahl mit drei verschiedenen imaginären Anteilen behandelt werden [1, 21]:

$$\dot{q} = q_0 + \mathfrak{i}q_x + \mathfrak{j}q_y + \mathfrak{k}q_z \quad \text{mit } q_0, q_x, q_y, q_z \in \mathbb{R}.$$

Bei gegebenem Einheitsvektor $\mathbf{n} = (n_x, n_y, n_z)^T$ und Rotationswinkel θ_n lässt sich das Einheitsquaternion berechnen durch [5, 12]:

$$q_0 = \cos\frac{\theta_n}{2} \tag{A.1}$$

$$q_x = n_x \sin\frac{\theta_n}{2} \tag{A.2}$$

$$q_y = n_y \sin\frac{\theta_n}{2} \tag{A.3}$$

$$q_z = n_z \sin\frac{\theta_n}{2}. \tag{A.4}$$

Die Rotationsmatrix berechnet sich aus einem Einheitsquaternion $\dot{q}$ wie folgt:

$$\mathbf{R} = \begin{pmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y + q_z q_0) & 2(q_x q_z + q_y q_0) \\ 2(q_x q_y + q_z q_0) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_x q_0) \\ 2(q_z q_x - q_y q_0) & 2(q_z q_y + q_x q_0) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{pmatrix}. \tag{A.5}$$

“ [30]

A.3 Homogene Koordinaten

Homogene Koordinaten können erzeugt werden, indem eine zusätzliche Dimension eingeführt wird. Sie werden verwendet, um verschiedene Transformationen mit Hilfe einer einzigen Transformationsmatrix darstellen zu können. Durch diese zusätzliche Dimension bekommt der Punkt $p = (p_x, p_y, p_z)^T$ die homogenen Koordinaten $p_h = (p_x, p_y, p_z, w)^T$, wobei w ein Skalierungsfaktor ungleich Null ist. In vielen Fällen wird $w = 1$ gewählt, wodurch die Dehomogenisierung der Koordinaten einfach durch weglassen der w-Kompenenten erfolgt. Aus einer 3×3 Matrix wird in homogenen Koordinaten eine 4×4 Matrix.

Literaturverzeichnis

[1] Sir William Rowan Hamilton (1805-1865). `http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/`, 2000.

[2] PHYTEC Technologie Holding AG. `http://www.phytec.de/`, April 2003.

[3] SuSE Linux AG. `http://sdb.suse.de/de/sdb/html/usb_ov511.html`, Februar 2003.

[4] Animatronik.de. `http://www.animatronik.de/`, July 2003.

[5] P. Besl and N. McKay. A method for Registration of 3–D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239 – 256, February 1992.

[6] Andrea Bottino. Motion capture system. website, July 2003. `http://staff.polito.it/ bottino/MotionCapture/index.html`.

[7] Bruno Caprile and Vincent Torre. Using vanishing points for camera calibration. *The International Journal of Computer Vision*, 4(2):127–140, March 1990.

[8] Borland Software Corporation. `http://www.borland.de`, February 2003.

[9] Jackie Neider Tom Davis and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 1993.

[10] Joao G.M. Goncalves Vitor Sequeira Erik Wolfart Paulo Dias. *3D Reconstruction of Monuments as-built*. Cultural Heritage Networks Hypermedia, 2000.

[11] Fachwörterbuch. Benennungen und Definitionen im deutschen Vermessungswesen. Verlag des Instituts für angewandte Geodäsie, 1995. Frankfurth a.M.

[12] Matrix FAQ. Version 2, `http://skal.planet-d.net/demo/matrixfaq.htm`, 1997.

[13] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*, chapter 1. Prentice Hall, 2002.

[14] Prof. Dr.-Ing. Wolfgang Förstner. *Vorlesung Photogrammetrie I.* Universität Bonn, April 2002.

[15] Fraunhofer-Institut für Autonome Intelligente Systeme. KURT3D. `http://ais.gmd.de/ARC/kurt3D/`, April 2003.

[16] DIN Deutsches Institut für Normung e.V. Din 18716-1, Photogrammetrie und Fernerkundung, 1995.

[17] MVTec Software GmbH. Halcon reference manual, May 2003.

[18] TerraTec Electronic GmbH. `http://www.terratec.de`, Februar 2003.

[19] Dipl.-Ing. Robert Godding. Geometric calibration and orientation of digital imaging systems. Technical report, AICON 3D Systems GmbH.

[20] RT-Linux Documentation Group. RT-Linux Manual Project, July 1998.

[21] W. Hamilton. On a new Species of Imaginary Quantities connected with a theory of Quaternions. In *Proceedings of the Royal Irish Academy*, Dublin, Ireland, November 1843.

[22] Richard Hartley. *Self-calibration from multiple views with a rotating camera.* Springer-Verlag, 1994.

[23] Infineon. `http://www.infineon.de`, Februar 2003.

[24] Intel Corporation. *Open Source Computer Vision Library. Reference Manual*, 2001.

[25] Bernhard Kuhn. Zeitgenau. *Linux Magazin*, 11, November 1998.

[26] Compaq Intel Microsoft and NEC. Universal serial bus specification, revision 1.1, September 1998.

[27] J. More. *The Levenberg-Marquard algorithm, Implementation and Theory.* Springer-Verlag, 1977.

[28] Compaq Hewlett-Packard Intel Lucent Microsoft NEC and Philips. Universal serial bus specification, revision 2.0, April 2000.

[29] Marco Nef. Entwicklung eines Systems zur Simulation eines multi-Kamera-basierten Gesichtscanners. Master's thesis, ETH Zürich, 2002.

[30] A. Nüchter. *Autonome Exploration und Modellierung von 3D-Umgebungen, GMD Report 157.* GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, 2001.

[31] Michael Randt. http://www.cardbussystems.com, July 2003.

[32] Prof. Dr. rer. nat. Gisela Engeln-Müllges. *Angewandtw und Numerische Mathematik.* Fachhochschule Aachen, 1997.

[33] Solidworks. `http://www.solidworks.de`, Februar 2003.

[34] Gunther Sudra. Intel opencv. Universität Karlsruhe, July 2002.

[35] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. A 3D laser range finder for autonomous mobile robots. In *Proceedings of the of the 32nd International Symposium on Robotics (ISR '01)*, pages 153 – 158, Seoul, Korea, April 2001.

[36] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. *Aufbau eines 3D–Laserscanners für autonome mobile Roboter, GMD Report 126*. GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, March 2001.

[37] Roger Y. Tsai. A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses. *IEEE Journal of Robotics and Automation*, 1987.

[38] KTO Kommunikation und Technologietransfer Odenthal. `http://www.kurt2.de/`, April 2003.

[39] Volz. `http://www.volz-servos.com`, Februar 2003.

[40] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. Technical report, Microsoft Research, December 1998.

[41] Zhengyou Zhang. Flexible Camera Calibratin By Viewing a Plane Fram Unknown Orientations. Technical report, Microsoft Research, 1999.

[42] Zhengyou Zhang. Camera Calibration With One-Dimensional Objects. Technical report, Microsoft Research, August 2002.

[43] Christina Zimmer. Kamera-modellierung. Hauptseminar Digitale Bildverarbeitung, 2003.