Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Prozeßrechentechnik, Automation und Robotik
Prof. Dr.-Ing. R. Dillmann

Javabasierte 3D-Objekterkennung aus Stereobildern

Studienarbeit
von

Felix Schill

Oktober 2001

Betreuer: Dipl.-Ing. T. Asfour

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Umwelterfassung ist eine der wichtigsten Grundlagen autonomer Serviceroboter. Gerade humanoide Roboter, die sich in einer für Menschen entworfenen, sich ständig verändernden Umgebung zurechtfinden müssen, sind auf eine flexible und vielseitige sensorische Analyse ihres Arbeitsumfeldes angewiesen. Ein solcher Roboter sollte Objekte lokalisieren und, falls bekannt, wiedererkennen und klassifizieren können. Unbekannte Objekte sollten möglichst einfach einlernbar sein.

Beim Menschen ist das Auge das leistungsfähigste und vielseitigste Sinnesorgan; ebenso liefert die maschinelle Bildauswertung mehr Informationen über die Umwelt als alle anderen Sensoren eines autonomen Roboters. Die Komplexität des menschlichen Sehapparates wird aber noch lange unerreicht bleiben. Bisher bekannte Bildauswertungsverfahren sind meist sehr speziell und auf einen bestimmten Einsatzzweck zugeschnitten.

Am Forschungszentrum für Informatik der Universität Karlsruhe (FZI) wird der humanoide Roboter ARMAR entwickelt. Um dem Ziel eines autonomen und menschenähnlichen Roboters näher zu kommen, wird ein flexibles und leistungsfähiges Bildauswertungssystem benötigt, das einfach erweiterbar ist. Bekannte Verfahren und neue Ansätze müssen deshalb auf ihre Tauglichkeit getestet, angepasst und entsprechend kombiniert werden.

1.2 Aufgabenstellung

Ziel der Arbeit ist es, ein Bildauswertungssystem zu entwerfen, das im Sichtfeld des Roboters vorhandene Objekte erkennen und ihre Position und Ausrichtung im Raum ermitteln kann. Hierzu steht ein am Kopf des Roboters montiertes Stereokamera-System zur Verfügung.

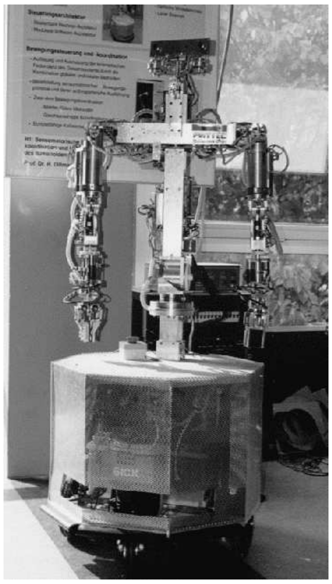

Abbildung 1.1: Der humanoide Roboter ARMAR

Dazu soll zunächst eine Testumgebung entwickelt werden, in der verschiedene Algorithmen, Filter und Datenformate schnell implementiert, integriert und getestet werden können. Parameter sollen zur Laufzeit einstellbar sein, und Zwischenergebnisse sollen auf Benutzerwunsch angezeigt werden.

Diese Testumgebung soll zur Implementierung und zum Testen der notwendigen Module verwendet werden, um abschließend ein Gesamtsystem zusammenzustellen.

1.3 Aufbau der Arbeit

In **Kapitel 2** wird die im Rahmen dieser Arbeit entwickelte Testumgebung beschrieben. Für weiterführende Arbeiten finden sich dort wichtige Hinweise zur Implementierung und Struktur des Systems.

Kapitel 3 befasst sich mit den einzelnen Bestandteilen des entwickelten Bildverarbeitungssystems. Die notwendigen Algorithmen werden hier erläutert und die gewonnenen Ergebnisse zusammengefasst. Der Aufbau dieses Kapitels richtet sich nach dem Datenfluß durch das System von den Kameras bis zum Ergebnis.

Abschließend werden in **Kapitel 4** verschiedene Anwendungsszenarien und dazu notwendige Erweiterungen beschrieben. Ein Teil des Kapitels beschäftigt sich mit Verbesserungsmöglichkeiten des Verfahrens. Das Kapitel schließt mit einer Zusammenstellung weiterführender Arbeiten.

Kapitel 2

Die modulare Testumgebung "FilterGUI"

2.1 Motivation

Ein Bildverarbeitungssystem besteht meist aus einer Vielzahl voneinander relativ unabhängiger Module. Diese Module (im Folgenden auch als "Filter" bezeichnet) besitzen Ein- und Ausgänge, und sind meist vom Benutzer durch Parameter konfigurierbar. Zwischen den Filtern werden Bilddaten, aber auch andere Datenstrukturen (Parametersätze, 3D-Daten, Modellbeschreibungen, ...) ausgetauscht.

Bei der Entwicklung eines Bildverarbeitungssystems werden ständig neue Filter hinzugefügt oder bereits vorhandene Teile weiterentwickelt bzw. ersetzt. Dabei müssen die Neuentwicklungen ständig getestet werden, sowohl im alleinigen Betrieb mit geeigneten Testdaten, als auch im Zusammenspiel mit dem bestehenden Gesamtsystem. Ergebnisse und Zwischenergebnisse müssen geeignet angezeigt und ausgewertet werden können. Um Tests zu erleichtern, sollten alle Änderungen an relevanten Parametern ohne Modifikationen am Quellcode mit anschliessendem Neukompilieren möglich sein. Ohne geeignete Hilfsmittel und Werkzeuge kann das zu einem enormen, aber letztendlich unproduktiven Programmieraufwand führen.

Die am FZI zur Robotersteuerung entwickelte Software MCA2 ermöglicht das einfache Erstellen, Testen und Kombinieren von Steuermodulen. MCA2 verfügt über umfangreiche Diagnose- und Administrationswerkzeuge und ist netzwerkfähig. Die aktuelle Version sieht aber nur die Verarbeitung einzelner Sensor- und Steuerwerte vor, eine Verarbeitung komplexer Datenstrukturen und großer Datenmengen wie Bilddaten wird von den Werkzeugen noch nicht unterstützt. Für die Entwicklung eines Bildverarbeitungs-Systems ist MCA2 in dieser Version ungeeignet.

Aus diesen Überlegungen entstand die Idee, eine objektorientierte, modulare und leicht erweiterbare Testumgebung für Bildverarbeituns-Filter zu entwickeln.

2.2 Anforderungen

Um die Implementierung eines Filters zu vereinfachen, sollte eine Klasse vorgegeben sein, die möglichst allgemein gehaltene Schnittstellen für Ein- und Ausgänge sowie Parameter bereitstellt. Die Implementierung sollte sich auf folgende zwei Schritte beschränken:

1. Festlegung der Anzahl von Ein- und Ausgängen, Anzahl und Typ der Parameter
2. Implementierung des datenverarbeitenden Algorithmus

Ebenso einfach sollte es sein, ein neues Datenformat zu implementieren. Die Testumgebung sollte diesbezüglich so wenig Einschränkungen wie möglich machen.

Um das Testen zu erleichtern, sollte eine grafische Benutzeroberfläche folgende Funktionen bieten:

- Anordnung von Filtern zu einem System
- Festlegung des Datenflusses zwischen den Filtern
- Speichern und Laden von Filtersystemen
- Komfortable Einstellmöglichkeit der Parameter der einzelnen Filter
- Geeignete Darstellung der Ein- und Ausgabedaten jedes Filters
- Starten der Ausführung des Systems

2.3 Realisierung

2.3.1 Vorüberlegungen

Aus den Anforderungen geht klar die Unterteilung in die Testumgebung als Verwaltungs- und Ausführungseinheit auf der einen Seite und die grafische Benutzeroberfläche (im folgenden auch GUI [1] genannt) auf der anderen Seite hervor.

Da der Datenfluss in einem System flexibel und jederzeit veränderbar sein soll, ist es sinnvoll, Filter mit speziellen Datenkanälen zu verbinden. Ein Filter hat demzufolge Ein- und Ausgangskanäle, wobei die Ausgangskanäle mit den Eingangskanälen anderer Filter verbunden werden können. Diese Kanäle kümmern sich auch um den Datentransport.

[1] GUI: engl.: Graphical User Interface

Um die Struktur der Filter und Datenkanäle möglichst einfach zu halten, ist es sinnvoll, die graphische Benutzeroberfläche und die Testumgebung so unabhängig wie möglich zu gestalten. Das heißt einerseits, dass bei neuen Datentypen oder Filtern keine Änderungen an der GUI nötig sind und trotzdem die volle Funktionalität stets zur Verfügung steht. Andererseits sollten auf der Seite der Testumgebung (also der Filter und Datenkanäle) möglichst wenige grafische Bestandteile enthalten sein. Um ersteres erfüllen zu können, ist es sinnvoll, jedes Datenobjekt mit einer Methode zu versehen, die dieses geeignet grafisch darstellen kann. Diese Methode muss für jeden Datentyp neu implementiert werden. Dadurch entstehen an der GUI keine Änderungen. Alle anderen grafischen Elemente werden komplett in den Benutzeroberflächen-Teil verlagert.

2.3.2 Wahl der Programmiersprache

Da das Ziel eine möglichst schnelle und einfache Entwicklung der Filter ist, bietet sich Java an. Die vereinfachte Speicherverwaltung mit Garbage-Collector ist sehr nützlich, da laut den Anforderungen der Datenfluss zur Laufzeit veränderbar sein soll. Auch die Plattform-Unabhängigkeit von Java ist ein großer Vorteil.

Der Nachteil an Java ist die geringe Geschwindigkeit, die bei der Bildverarbeitung eine enorme Bedeutung hat. Wenn man auf die Plattformunabhängigkeit verzichtet, kann man aber mit dem Java Native Interface (JNI) geschwindigkeitskritische Filter in C oder C++ schreiben und integrieren. Da der Schwerpunkt bei der Entwicklung von Bildverarbeitungssystemen mit vereinfachtem Testen liegt, stellt die etwas geringere Geschwindigkeit hier kein großes Problem dar. Nachdem die Test- und Entwicklungsphase abgeschlossen ist, können die einzelnen Filter nacheinander nach C++ portiert und mit JNI integriert und getestet werden. Nachdem alle involvierten Filter portiert worden sind, ist es relativ einfach, das komplette System in ein C++-Programm umzusetzen und zu optimieren.

2.3.3 Grundlegende Klassen

Die Klasse "ProcessData"

Alle Datenobjekte, die verarbeitet werden sollen, müssen einer Unterklasse von `ProcessData` angehören. Bezüglich des Inhalts oder der Datenstruktur werden keine Vorgaben gemacht.

Um den Inhalt mit der grafischen Benutzeroberfläche betrachten zu können, muss die Methode `public Component getViewContent()` überschrieben werden, die ein AWT-Component-Objekt mit der grafischen Darstellung der Daten zurückgibt. Dieses Objekt kann natürlich auch Menüs, Schaltflächen oder Listener beinhalten, um die Darstellung nach Benutzerwunsch anpassen zu können.

Um den Inhalt eines Datenobjektes zuordnen zu können, muss auch die Methode `public String toString()` überschrieben werden, so daß sie eine kurze Beschreibung der Daten liefert.

Falls benötigt, können auch die beiden Methoden `void save(String filename)` und `void load(String filename)` überschrieben werden, um die Daten abspeichern und wieder einlesen zu können.

Die Klasse “Channel”

Diese Klasse enthält ein Datenobjekt vom Typ `ProcessData\verb` und kümmert sich um dessen Transport. Hierzu enthält ein Channel Informationen über den Zielfilter und den dortigen Dateneingang, die mit der grafischen Benutzeroberfläche bearbeitet werden können.

Zur besseren Übersichtlichkeit kann einem Channel eine Beschreibung gegeben werden. Sie ist in dem String `description` enthalten und sollte für jeden Channel entsprechen gesetzt werden.

Für den Datentransport stehen die Methoden `public void deliver()` bzw. `public void deliver(ProcessData data)` zur Verfügung.

Die Klasse “Filter”

Diese Klasse bildet die Oberklasse für alle zu integrierenden Filtermodule. Ein Filter hat Ein- und Ausgänge vom Typ `Channel`, die in den Feldern `inputChannel[]` und `outputChannel[]` enthalten sind. Daneben enthält die Klasse Filter ein Feld `param[]` vom Typ `FilterParameter[]`, in dem eventuell benötigte Parameter verwaltet werden.

Die Verarbeitung der Daten findet in der Methode `public void filterAction()` statt. Diese Methode wird entweder auf Benutzerwunsch aufgerufen, oder wenn sich bei der automatischen Ausführung ein Eingang des Filters geändert hat. Die Methode `public void changedParameters()` wird aufgerufen, nachdem der Benutzer etwas an den Parametern geändert hat.

2.3.4 Die Testumgebung “FilterManager”

Diese Klasse enthält eine Auflistung aller verfügbaren Filter (neue Filter müssen deshalb hier eingetragen werden) und eine Systembeschreibung, bestehend aus einer Liste der benutzten Filterobjekte, wobei die Verbindungen unter den Filtern als Verweise in den Ausgabekanälen (`outputChannel`) der einzelnen Filterobjekte gespeichert sind.

Dazu existieren diverse Methoden zum Hinzufügen und Entfernen von Filtern und Verbindungen, zum Speichern und Laden des Systems und zum Ausführen.

Bei der Ausführung des enthaltenen Systems wird ein Thread gestartet, der folgende zwei Schritte wiederholt:

1. Ermitteln aller Filter, deren Eingänge sich geändert haben
2. Aufruf der Methode `filterAction` aller in Schritt 1 ermittelten Filter

Der `FilterManager` kann direkt von der Kommandozeile gestartet werden, wobei als Argument ein Systemlayout angegeben werden muss, wird aber auch von der Benutzeroberfläche instanziiert. Es wurde auch ein Filter implementiert, der einen `FilterManager` enthält und ein geladenes Subsystem in ein anderes System integrieren kann. Eine ausführliche Beschreibung hiervon folgt später im Abschnitt 2.3.7.

2.3.5 Die grafische Benutzeroberfläche (GUI)

Aufbau

Der wichtigste Bestandteil der GUI ist die Arbeitsfläche [2]. Hier können mit der Maus Filter zu einem System hinzugefügt und in der zweidimensionalen Fläche frei positioniert werden, Verbindungen können per *"drag and drop"* hergestellt oder verändert werden. Daneben gibt es Menüs zur Dateiverwaltung und Ansicht sowie Schaltflächen für die verschiedenen Bearbeitungsmodi und zur Ausführung.

Die GUI enthält ein Objekt des Typs `FilterManager`, das sie auf der Arbeitsfläche visualisiert. Filter werden als weiße Kreise mit Beschriftung dargestellt, die Verbindungen (`Channels`) zwischen den Filtern werden durch Pfeile symbolisiert, wobei die Pfeilrichtung dem Datenfluß entspricht. Falls der entsprechende `Channel` keine Daten enthält, ist er hellgrau-bläulich gefärbt, ansonsten schwarz. Filter, die gerade ausgeführt werden, sind grün ausgefüllt.

Entsprechend den Benutzereingaben wird das System im FilterManager über die dort vorhandene Schnittstelle angepasst. Die Visualisierung wird bei jeder Änderung aktualisiert; die Struktur und Funktionsweise des Systems ist somit jederzeit klar ersichtlich. Durch die grafische Darstellung der Ausführungsreihenfolge wird das Testen sehr erleichtert.

Für jedes im System vorhandene Filterobjekt kann eine Dialogbox [3] geöffnet werden. Dort werden die Ein- und Ausgänge aufgelistet. Alle im Filter definierten Parameter werden mit Bezeichnung und Wert angezeigt und können verändert werden. Zudem kann mit den

[2] Klassen 'GraphPlane' und 'GraphPanel'
[3] Klasse 'FilterEditDialog'

Schaltflächen *"View"* der Inhalt der Ein- und Ausgänge angezeigt werden. Hierbei wird das zurückgegebene Objekt der oben erwähnten Methode `getViewContent()` aus der Klasse `ProcessData` in einem eigenen Fenster dargestellt.

Es spielt somit für die GUI grundsätzlich keine Rolle, welche Daten ein `Channel` enthält; bei der Entwicklung neuer Datentypen und Filter muss nichts an der GUI geändert werden.

Bedienung

Zur Bearbeitung des Filtersystems gibt es drei Bearbeitungsmodi:

Edit: Wenn in diesem Modus auf einen Filter in der Arbeitsfläche geklickt wird, öffnet sich die oben erwähnte Filter-Dialogbox.

Add Filter: Dieser Modus erlaubt das Hinzufügen und Positionieren von Filtern. In einem zusätzlichen Fenster kann aus einer Liste mit allen zur Verfügung stehenden Filtern der gewünschte ausgewählt werden. Durch einen Mausklick auf eine freie Stelle der Arbeitsfläche wird der gewählte Filter dort in das System eingefügt.

Durch einen Mausklick auf einen bereits in der Arbeitsfläche vorhandenen Filter und anschließendes Bewegen des Mauszeigers bei gedrückt gehaltener Maustaste kann dieser Filter neu positioniert werden. Um Überdeckungen in der Darstellung zu vermeiden, werden andere Filter gegebenenfalls zur Seite geschoben.

Ist jedoch die Schaltfläche *"Delete"* zusätzlich aktiviert, wird der als nächstes angeklickte Filter gelöscht.

Connect: Um zwischen den Filtern Channels einzufügen, wählt man diesen Bearbeitungsmodus. Durch einen Mausklick auf einen Filter werden in einer Liste am rechten Rand die vorhandenen Ausgänge angezeigt. Nachdem man dort den gewünschten Ausgang selektiert hat, klickt man wieder auf den zuvor gewählten Filter, zieht bei gedrückter Maustaste den Mauszeiger über den Zielfilter und lässt dort die Taste los. Dem gewählten Ausgangskanal wird daraufhin die Adresse des Zielfilters zugewiesen. Nun muß nur noch in der erscheinenden Dialogbox der gewünschte Eingangskanal des Zielfilters gewählt werden.

Möchte man eine Verbindung löschen, wählt man wie eben beschrieben den Filter und den entsprechenden Ausgangskanal, aktiviert die Schaltfläche *"Delete"* und klickt nochmals auf den Filter.

Die Schaltfläche *"Optimizer"* aktiviert bzw. deaktiviert einen Thread, der die Filteranordnung entzerrt, um eine übersichtlichere Darstellung zu erreichen. Die beschriebenen Bearbeitungsmodi bleiben davon unbeeinflusst.

Rechts oben befinden sich Schaltflächen zum Vergrößern und Verkleinern der Darstellung sowie zum Verändern des Ausschnitts der Arbeitsfläche. Im Menü *"View"* kann die Darstellung an die Größe des Systems angepasst werden. Das Menü *"File"* enthält Befehle zum Laden und Speichern von Systemen.

2.3.6 Implementierungshinweise zur Filterentwicklung

Um einen neuen Filter zu implementieren, erzeugt man eine neue Klasse, die von *"Filter"* erbt. Im Konstruktor wählt man die gewünschte (eindeutige) Bezeichnung sowie die Anzahl der Ein- und Ausgänge des Filters, indem man den Konstruktor der Oberklasse mit den Parametern `(String description, int noOfInputs, int noOfOutputs)` aufruft. Falls man Parameter benötigt, weist man der Klassenvariablen `param[]` ein Feld vom Typ `FilterParameter` der gewünschten Größe zu und initialisiert die einzelnen Parameter mit der entsprechenden Bezeichnung und dem gewünschten Standardwert.

Anschließend überschreibt man die Methode `public void filterAction()`, in der man die Verarbeitung der Daten implementiert. Folgende Vorgehensweise ist empfehlenswert:

- Einlesen der Eingangsdaten und Konvertieren in die gewünschte Klasse
- Überprüfung der Korrektheit der Daten
- Verarbeitung
- Ausliefern der Ergebnisse in die Ausgabekanäle
- Rücksetzen der Eingänge

Beispiel eines Filters:

```
class FExample extends Filter {
    public FExample() {
        // 2 Eingaenge, ein Ausgang
        super("Beispielfilter", 2,1);

        // Ein- und Ausgaenge benennen
        inputChannel[0].description="input1";
        inputChannel[1].description="input2";
        outputChannel[0].description="output1";

        // Zwei Parameter
        param=new FilterParameter[2];
        param[0]=new FilterParameter(FilterParameter.TEXT,
```

```
                                    "Text-Parameter");
        param[0].change(``Default'');
        param[1]=new FilterParameter(FilterParameter.VALUE,
                                    "Double-Parameter");
        param[1].change(1.0);
    }

    /** Hier geschieht die Datenverarbeitung */

    public void filterAction() {

        // Eingaenge einlesen und konvertieren
        ImageData input1=(ImageData)inputChannel[0].getData();
        ImageData input2=(ImageData)inputChannel[0].getData();

        // Ueberpruefung der Eingangsdaten
        if ((input1!=null)&&(input2!=null)) {

            // Beginn der Verarbeitung
            ProcessData output= process(input1, input2);

            // Ergebnis ausliefern
            outputChannel[0].deliver(result);
        }
        // Eingaenge zuruecksetzen
        clearInputs();
    }
}
```

2.3.7 Spezielle Filter

Ab einer gewissen Komplexität ist es sinnvoll, Systeme beim Entwurf in Subsysteme zu unterteilen und diese dann zu einem Gesamtsystem zusammenzufügen. Das sorgt nicht nur für bessere Übersichtlichkeit, sondern ermöglicht auch das einfache Ersetzen von Subsystemen durch verbesserte Versionen.

Hierzu stehen spezielle Filter zur Verfügung:

Filter GroupInput

Dieser Filter stellt eine Quelle für ein Subsystem zur Verfügung. Er hat einen Ausgang und kann mit einer Beschreibung versehen werden.

Filter GroupOutput

Als Gegenstück zu GroupInput entspricht dieser Filter einer Senke mit einem Eingang.

Filter SubSystem

Der Filter SubSystem enthält einen FilterManager (Abschnitt 2.3.4). Der einzige Parameter dieses Filters ist der Dateiname eines gespeicherten Subsystems. In der Grundform hat der Filter SubSystem keine Ein- und Ausgänge. Die angegebene Datei wird zunächst in den internen FilterManager geladen und nach GroupInput- und GroupOutput-Filtern durchsucht. Für jeden enthaltenen GroupInput wird ein Eingang mit der entsprechenden Bezeichnung erzeugt, entsprechend ergibt jeder GroupOutput einen Ausgang. Bei der Ausführung werden die Daten aus den Channels automatisch in das Subsystem weitergeleitet und die Ausgänge nach aussen verbunden.

Die GUI stellt im Menü *"View"* eine Funktion *"Show SubSystem"* zur Verfügung, mit der ein neues Fenster mit dem FilterManager des gerade selektierten SubSystem-Filters geöffnet werden kann.

SubSysteme können beliebig kaskadiert werden, es dürfen aber keine zyklischen Abhängigkeiten erzeugt werden.

Kapitel 3

3d-Objekterkennung für ARMAR

3.1 Vorüberlegungen

3.1.1 Anforderungen

Der humanoide Roboter ARMAR soll in einer für Menschen entworfenen Umwelt zurechtkommen und mit Menschen oder anderen Servicerobotern interagieren können. Anders als bei Montagerobotern in der Industrie, die ihren Arbeitsbereich exakt kennen und nicht mit unvorhersehbaren Veränderungen rechnen müssen, steht hier nur unzureichendes a-priori-Wissen über die Umwelt zur Verfügung Damit der Roboter autonom agieren kann, muss die Sensorik den gesamten Arbeitsbereich erfassen, Objekte klassifizieren und potentielle Hindernisse erkennen können.

Dieses Kapitel beschreibt die Realisierung einer stereobildbasierten Objekterkennung. Das System soll bekannte Objekte wiedererkennen können und Daten über Lage und Orientierung im Raum liefern. Daneben ist es erstrebenswert, auch Informationen über unbekannte Objekte gewinnen zu können, um mit dem gleichen Verfahren die Objektdatenbank erweitern zu können.

3.1.2 Vorhandene Hardware

Der humanoide Roboter ARMAR besitzt einen Kamerakopf mit drei Freiheitsgraden: Neigen, Schwenken und Schielen. Der Antrieb erfolgt durch Gleichstrom-Getriebemotoren, für die Winkelerfassung sind an den Motorachsen Relativ-Schrittgeber angebracht. Die Winkelregelung der Achsen erfolgt auf einem C167-Microcontroller von Siemens. Die gesamte Ansteuerung ist bereits implementiert und erfolgt über die im FZI entwickelte Open-Source-Softwarearchitektur *"MCA2"*. Das Modul zur Ansteuerung erwartet einen Punkt im dreidimenensionalen Raum; daraufhin wird der Kopf so ausgerichtet, dass beide Kameras diesen Punkt fixieren.

Für die Bilderfassung werden zwei CCD-Farbkameras der Firma Phytec GmbH vom Typ AK-035-1 eingesetzt. Für die Digitalisierung kommen zwei handelsübliche PCI-Grabberkarten mit dem BTTV-Chipsatz zum Einsatz.

Technische Daten der CCD-Kamera AK-035-1	
Bildsensor:	1/3“ CCD Farbsensor (Sony ICX-059CK)
TV-Signalnorm:	PAL
Pixelauflösung:	752(H) x 582(V)
Objektiv	Fixfocus f=5,6mm F3,0
Videoausgang:	VBS (1Vpp, 75 Ohm), Y/C (S-Video)
Aufnahmesystem:	2:1 interlace
Synchronisation:	Intern oder Extern (VS/VBS, Sync, HD/VD o. Line Lock)
Gamma-Korrektur:	0,45
Mindestlichtstärke:	Standard 0.5 Lux (bei AGC=ON +39dB, F1.4)
Signal-Rausch-Abstand:	>50dB (AGC=OFF)
AGC:	39dB max. (Analog 24dB + Digital 15dB), konfigurierbar
Shutter:	AEC=ON: 1/50 - 1/30.000 sec / konfigurierbar
Weißabgleich:	ATW=ON: 2600°K - 9000°K/ATW=OFF/konfigurierbar
Betriebsspannung:	+6 bis +15V DC, 1,8 W bei 12V
Abmessungen:	42(L) x 42(B) x 34(H) mm
Gewicht:	ca. 35 g

3.1.3 Montage der Kameras

Eingehende Tests mit dem vorhandenen Kopf und den Kameras ergaben, dass der dritte Freiheitsgrad, das Schielen, mehr Nachteile als Vorteile bringt.

Der Mensch benötigt diesen zusätzlichen Freiheitsgrad hauptsächlich aufgrund der unterschiedlich verteilten Auflösung der Netzhaut: die maximale Sehschärfe und Farbauflösung wird nur im sogenannten “Gelben Fleck” erreicht, der im Durchstosspunkt der optischen Achse auf der Netzhaut liegt. Die Augen werden deshalb immer auf das interessierende Objekt ausgerichtet [1].

Die verwendeten CCD-Kameras haben aber über den gesamten Bildbereich die gleiche Orts- und Farbauflösung; auch die Bildschärfe ist überall ausreichend. Durch das Weitwinkelobjektiv sind auch nahe gelegene Objekte noch in beiden Kameras zu sehen.

Die Präzision der Mechanik erreicht nicht die Spaltenauflösung der Kameras, so dass eine exakte Tiefenschätzung erschwert wird; zudem ist die Kalibrierung bei zueinander beweglichen Kameras ungleich aufwändiger. Aus diesem Grund wurden beide Kameras parallel zueinander auf einer Trägerplatte starr fixiert. Diese Einheit passt exakt auf den vorhandenen Kopf. Dadurch waren keine Modifikationen am Roboter und an der Steuersoftware notwendig.

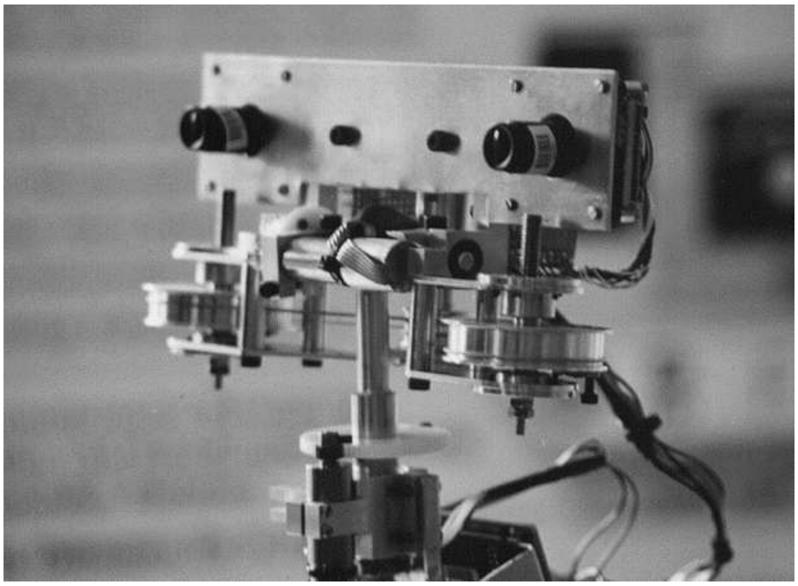

Abbildung 3.1: Der Kopf des humanoiden Roboters ARMAR

3.1.4 Prinzipielle Struktur des Bildverarbeitungssystems

Da das System auch in unbekannter Umgebung zurechtkommen soll, sind modellgestützte Bildverarbeitungsverfahren ungeeignet. Eine Lösung des Problems ist die Kombination einer Stereobild-Tiefenschätzung mit einer Objekterkennung auf Basis der gewonnenen dreidimensionalen Punkte.

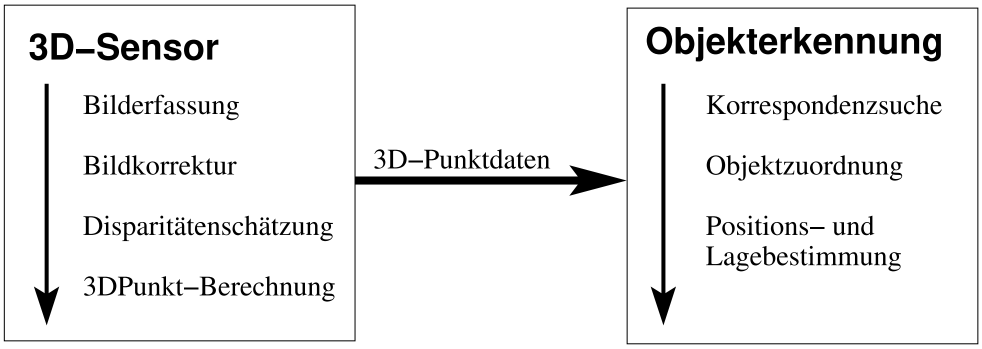

Abbildung 3.2: Die geplante Struktur des Bildverarbeitungssystems

Daraus ergibt sich eine klare Zweiteilung des Systems: Zunächst ermittelt ein "Stereobild-3D-Sensor" Oberflächenpunkte aus der Szene. Die darauffolgende Objekterkennung vergleicht die entsprechend nachbearbeiteten 3D-Punktwolken mit Referenzobjekten in einer Datenbank. Falls ein entsprechender Eintrag gefunden wird, kann die Position und Orientierung des erkannten Objektes berechnet werden.

3.2 Realisierung

Bei der Implementierung war die in Kapitel 2 beschriebene Testumgebung *"Filtergui"* von großem Nutzen. Die benötigten Algorithmen zur Bild-Vorverarbeitung und -Auswertung wurden als Filtermodule implementiert und konnten dann für sich und in zunehmend komplexeren Filter-Systemen getestet werden. Zur Fehlersuche und Optimierung war die Anzeigeoption der Zwischenergebnisse sehr hilfreich.

Während der Implementierungsphase enstanden zu Test- und Optimierungszwecken zahlreiche Filtermodule, die keinen direkten Bezug zum endgültigen Bildverarbeitungssystem haben. Der Filter `ChannelWriter` speichert beispielsweise die Daten vom Eingangskanal unter einem als Parameter gegebenen Dateinamen ab, vorausgesetzt, der Eingangsdatentyp stellt die entsprechenden Speicherfunktionen zur Verfügung. Bei den Datentypklassen `ImageData`, `PointData` und `ParameterData` wurden entsprechende Funktionen implementiert. Entsprechend stehen die Filtermodule `ImageLoader`, `PointLoader` und `ParameterLoader` zum Einlesen der gespeicherten Daten zur Verfügung.

Im Folgenden werden nun die Komponenten des implementierten Bildverarbeitungssystems genau erläutert. Die Reihenfolge der Darstellung folgt im Wesentlichen dem Datenfluss durch das System.

3.3 Kamera-Kalibrierung und Bildkorrektur

Die Kamera-Kalibrierung und Implementierung der Bildkorrektur-Module wurden im Rahmen eines Roboter-Projektpraktikums am FZI vorgenommen. Um einen groben Überblick über die Funktionsweise der Bildkorrektur zu vermitteln, wird die Vorgehensweise im folgenden kurz skizziert:

Mit Hilfe der Tsai-Kalibrierungsbibliothek wurden die Kamera-Parameter ermittelt. Hierzu wurde ein Stereo-Bildpaar von einem Referenzobjekt erstellt, das ein Raster von schwarz ausgefüllten Quadraten auf weißem ebenen Hintergrund aufweist. Mit der Grauwert-Strukturtensormethode ([6], [7]) wurden die Ecken der Quadrate ermittelt. Aus den Ecken im Bild und den tatsächlichen Maßen der Quadrate konnten die Parameter für den Kameraabstand sowie den rotatorischen und translatorischen Versatz zur zeilenparallelen Ausrichtung bestimmt werden. Ebenso wurden die Brennweite in Pixeln und ein Parameter zur Linsenverzerrung (oder auch "Kissenverzerrung") ermittelt.

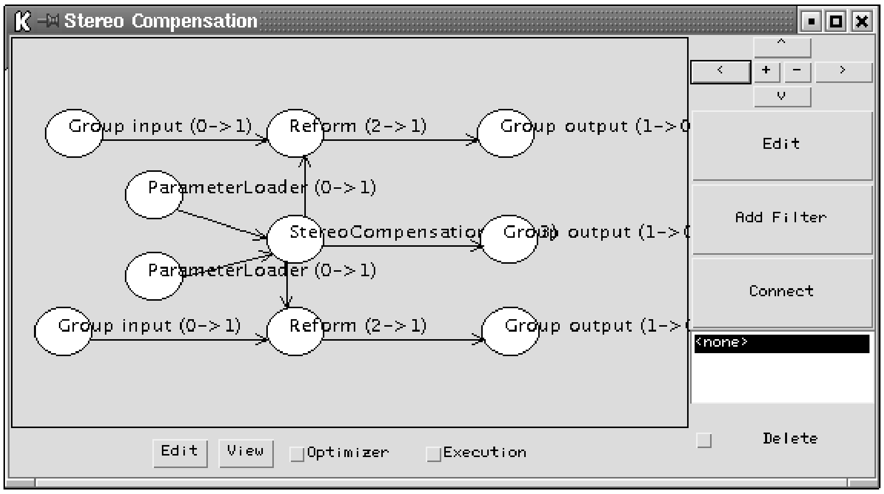

Abbildung 3.3: Das Bildkorrektur-Subsystem

Mit Hilfe dieser Parameter wird eine Abbildung vom realen auf ein virtuelles, ideales Kamerapaar ohne Linsenverzerrung berechnet. Die Achsen der virtuellen Kameras sind orthonormiert, die optischen Zentren liegen auf der x-Achse der jeweils anderen Kamera. Zudem sind die x-, y- und z-Achsen der Kameras parallel. Die Epipole (Schnittpunkt der Verbindungsgerade der optischen Zentren mit der Bildebene) beider Kameras liegen im Unendlichen; somit sind alle Epipolaren parallel zu den Pixelzeilen. Außerdem sind alle Epipolaren identisch mit den zugehörigen Epipolaren der anderen Kamera.

Aus Geschwindigkeitsgründen wird ein Feld vorberechnet, das für jedes Pixel des Ergebnisbildes die Koordinaten des entsprechenden Urbild-Pixels enthält. So werden bei linearem Aufwand außerdem Lücken im Ergebnisbild vermieden. Da sich die Kalibrierdaten während des Betriebs nicht ändern, kann so jedes aufgenommene Bild schnell entzerrt werden. Zur späteren Triangulation von 3d-Punkten werden zusätzlich die Kalibrierparameter der virtuellen Kameras ausgegeben (Basisabstand B, Brennweite (f_x, f_y) und optisches Zentrum).

3.4 Disparitätenschätzung

Um aus Stereobildern Tiefeninformationen zu erhalten, müssen korrespondierende Bildpunkte gefunden werden. Alle korrespondierenden Punkte zu einer Epipolaren im linken Bild liegen auf einer Epipolaren im rechten Bild, die durch die Stereokamera-Geometrie vorgegeben ist. Nach der oben beschriebenen Bildkorrektur kann davon ausgegangen werden, dass die Epipolaren mit den Pixelzeilen zusammenfallen. Ein unendlich weit entfernter Punkt wird in beiden Bildern auf Pixel mit den gleichen Koordinaten abgebildet. Ist die Entfernung aber endlich, wird der Punkt im rechten Bild auf der Epipolaren um einen gewissen Betrag nach links verschoben abgebildet. Ist diese Disparität bekannt, kann problemlos per Triangulation die Position des Urbildpunktes dreidimensional rekonstruiert werden.

Die große Schwierigkeit ist die schnelle und zuverlässige Bestimmung der zugehörigen Disparitäten aller Pixel eines Bildes. Der hier verfolgte Ansatz errechnet aus einem Stereobildpaar ein Tiefenbild, das für jeden Bildpunkt des linken Bildes eine Schätzung der entsprechenden Disparität enthält.

3.4.1 Erster Ansatz

Die grundsätzliche Idee ist, zu jedem Pixel im linken Bild einen entsprechend passendes Pixel im rechten Bild zu finden. Es liegt nun nahe, die "Abstände" zwischen einem Pixel im linken Bild und allen auf der zugehörigen Epipolaren liegenden Pixel zu bilden und daraus das Minimum zu wählen. Natürlich macht es keinen Sinn, einzelne Pixel zu vergleichen; ein geeignetes Abstandsmaß muss daher die Umgebung der Pixel berücksichtigen.

Definition 3.4.1 (Grauwert-Abstandsmaß für Pixel in Grauwertbildern) *Seien $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ Pixel in Grauwertbildern mit den gegebenen Koordinaten. Außerdem sei $g(x, y)$ der Grauwert des Bildes an den Koordinaten (x, y). Dann bezeichnet*

$$d(P_1, P_2) = d(x_1, y_1, x_2, y_2) := |g(x_1, y_1) - g(x_2, y_2)|$$

den Abstand der Grauwerte der Pixel P_1 und P_2.

Definition 3.4.2 (Umgebungs-Abstandsmaß für Pixel in Grauwertbildern) *Seien $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ Pixel in Grauwertbildern. Die Blockgröße des rechteckigen Testausschnitts sei gegeben durch ε_x und ε_y. Dann ist der Abstand der Testauschnitte um P_1 und P_2*

$$d_{\varepsilon_x,\varepsilon_y}(x_1, y_1, x_2, y_2) := \frac{\sqrt{\sum_{i=-\varepsilon_x}^{\varepsilon_x} \sum_{j=-\varepsilon_y}^{\varepsilon_y} d\left(x_1 + i, y_1 + j, x_2 + i, y_2 + j\right)^2}}{(2 * \varepsilon_x + 1) * (2 * \varepsilon_y + 1)}$$

Damit das Abstandsmaß unabhängig von der Größe des Testauschnitts ist, wird zur Normierung durch die Anzahl der Pixel im Testausschnitt dividiert.

Eine Schätzung der Disparität in einem Pixel (x,y) im linken Bild wird wie folgt definiert:

Definition 3.4.3 (Disparität in (x,y)) [1]

$$disparity(x, y) := \arg\min_{p \in [0..x]} \left(d_{\varepsilon_x,\varepsilon_y}\left(x, y, x - p, y\right)\right)$$

[1] $\arg\min_{p \in I}(f(p))$ stellt jenes Argument p dar, welches f(p) minimiert.

Auf diese Weise kann ein Tiefenbild berechnet werden, dass jedem Pixel (x,y) des linken Bildes die zugehörige geschätzte Disparität $disparity(x, y)$ zuordnet. Der Algorithmus wurde in dem Filtermodul *"3D Correlator"* implementiert.

Die Nachteile dieses ersten Ansatzes liegen auf der Hand: Da nur Bildbereiche der festen Blockgröße $(\varepsilon_x, \varepsilon_y)$ miteinander verglichen werden, können bei sich wiederholenden Strukturen leicht Fehlzuordnungen auftreten. Zudem ist der Berechnungsaufwand enorm, da für jedes Pixel im Mittel über die halbe Bildzeile hinweg das Minimum des Abstandes gefunden werden muss. Wird die Berechnung des Abstands der Grauwerte zweier Pixel $d(x_1, y_1, x_2, y_2)$ als atomare Operation zugrunde gelegt, beträgt der Aufwand für ein Bildpaar der Breite b und der Höhe h

$$O\left(b \cdot h\left(\frac{b}{2}\left(2\varepsilon_x + 1\right)\left(2\varepsilon_y + 1\right)\right)\right).$$

Bei konstanter Blockgröße ergibt sich ein Aufwand von $O(b^2 \cdot h)$. Die Ausführungszeit auf einem PC mit dem AMD Athlon 800 Prozessor bei einer Bildauflösung von 400x300 Pixeln betrug über zehn Minuten. Es muss also ein besseres Verfahren gefunden werden.

3.4.2 Hierarchisches Block-Matching

Um Fehlzuordnungen zu minimieren, sollte die Blockgröße $(\varepsilon_x, \varepsilon_y)$ des Abstandsmaßes nach Definition 3.4.2 möglichst groß sein. Dadurch nimmt aber die Genauigkeit der Disparitätenschätzung ab und der Berechnungsaufwand steigt stark an. Eine Lösung dieses Dilemmas ist ein hierarchisches Verfahren, das zunächst eine grobe Schätzung der Disparität großer Strukturen im Bild ermittelt und diese dann schrittweise verfeinert [5].

Aus den beiden Eingangsbildern wird zunächst eine Gaußpyramide G (siehe [3], Kap. 5.4 Mehrgitterrepräsentation, S. 143) berechnet. Die Ebenen G_n der Gaußpyramide enthalten Bilder mit einer Auflösung, die das $1/r(n)$-fache der Auflösung des Ursprungsbildes beträgt, wobei die Auflösungsreduzierung eine gaußglockengewichtete Mittelung durchführt. Die Funktion $r(n)$ ist eine Exponentialfunktion; üblicherweise gilt $r(n) := 2^n$. Die Anzahl k der notwendigen Ebenen hängt von der Auflösung der Eingangsbilder ab. Der Berechnungsaufwand und der Speicherbedarf verhalten sich logarithmisch zur Anzahl der zu berechnenden Ebenen. In Abbildung 3.4 sind die beiden Gaußpyramiden für das linke und rechte Bild grafisch dargestellt.

Auf dem entsprechend niedrig aufgelösten Bild der untersten Ebene G_k kann mit dem in Definition 3.4.1 beschriebenen Verfahren eine grobe Disparitätenschätzung schnell durchgeführt werden. Eine Blockgröße von wenigen Pixeln ist auf dieser Skala groß relativ zur Bildgröße, so dass Fehlzuordnungen unwahrscheinlicher werden. Aufgrund der geringen Auflösung ist auch der Berechnungsaufwand gering. Natürlich wird die Präzision mit der Auflösung ebenfalls reduziert. Die grobe Schätzung auf der untersten Ebene muss also sinnvoll verwendet werden, um mit wenig Aufwand die Disparitäten für die nächsthöhere Ebene der Gaußpyramide zu ermitteln.

Schnelle Disparitätenschätzung mit a-priori-Wissen

Ist ein niedrig aufgelöstes Tiefenbild der untergeordneten Ebene G_{n+1} bereits bekannt, muss für die verfeinerte Berechnung der Disparitäten in der Ebene G_n im rechten Bild nicht die gesamte Epipolare nach dem Minimum abgesucht werden. Vielmehr kann aufgrund des a-priori-Wissens aus G_{n+1} ein Intervall $I = [z - \varepsilon, z + \varepsilon]$ angegeben werden, in dem das Minimum zu erwarten ist. Das Zentrum z des Intervalls ergibt sich direkt aus der Disparität der gleichen Stelle im Bild der Ebene $n+1$, wobei mit dem Auflösungsverhältnis $a_n = \frac{b_n}{b_{n+1}}$ zwischen den Ebenen n und $n+1$ skaliert werden muss:

$$z_{G_n}(x, y) = a \cdot disparity_{G_{n+1}}(x/a_n, y)$$

Die Intervall-Ausdehnung ε hängt ebenfalls von a_n ab, es müssen aber unter Umständen die Disparitäten der benachbarten Pixel in G_{n+1} berücksichtigt werden. Entspricht die Pixelposition in G_n ganzzahligen Koordinaten in G_{n+1}, ist ε gegeben durch $\varepsilon_{G_n}(x, y) = a_n$. Falls die entsprechenden Koordinaten in G_{n+1} jedoch nicht ganzzahlig sind, kann die Disparität Werte zwischen den Disparitäten der links bzw. rechts davon liegenden ganzzahligen Bildpositionen annehmen. Die Intervallausdehnung ε berechnet sich dann wie folgt:

$$\varepsilon_{G_n}(x, y) = a_n \cdot \left| disparity_{G_{n+1}}(\lfloor x/a_n \rfloor, y) - disparity_{G_{n+1}}(\lceil x/a_n \rceil, y) \right|$$

Mit diesem Intervall I kann nun bei bekanntem Tiefenbild aus G_{n+1} eine schnelle Disparitätenschätzung für G_n berechnet werden:

Definition 3.4.4 (Disparität in (x,y) mit a-priori-Wissen) *Sei das Suchintervall I gegeben durch*

$$I := [z_{G_n}(x, y) - \varepsilon_{G_n}(x, y), z_{G_n}(x, y) + \varepsilon_{G_n}(x, y)]$$

mit z, ε wie oben beschrieben. Dann ist

$$disparity_{G_n}(x, y) := \arg\min_{p \in I} \left(d_{\varepsilon_x, \varepsilon_y}(x, y, x - p, y) \right)$$

Der Aufwand kann hier nicht mehr exakt bestimmt werden, da er durch $\varepsilon_{G_n}(x, y)$ linear vom Gradienten des G_{n+1}-Tiefenbilds in x-Richtung abhängt. Dieser Gradient nimmt aber nur an Objektkanten große Werte an, die einen geringen Teil des Gesamtbildes ausmachen. Schätzt man daher $\varepsilon_{G_n}(x, y)$ nach oben durch einem konstanten Wert ab, ergibt sich im O-Kalkül ein Aufwand von $O(b \cdot h)$.

Gesamtaufwand

Um das voll aufgelöste Tiefenbild für G_0 zu erhalten, wird also zunächst für die unterste Stufe G_k der Gauß-Pyramide wie in Definition 3.4.3 beschrieben ein Tiefenbild $T_k(G_k)$

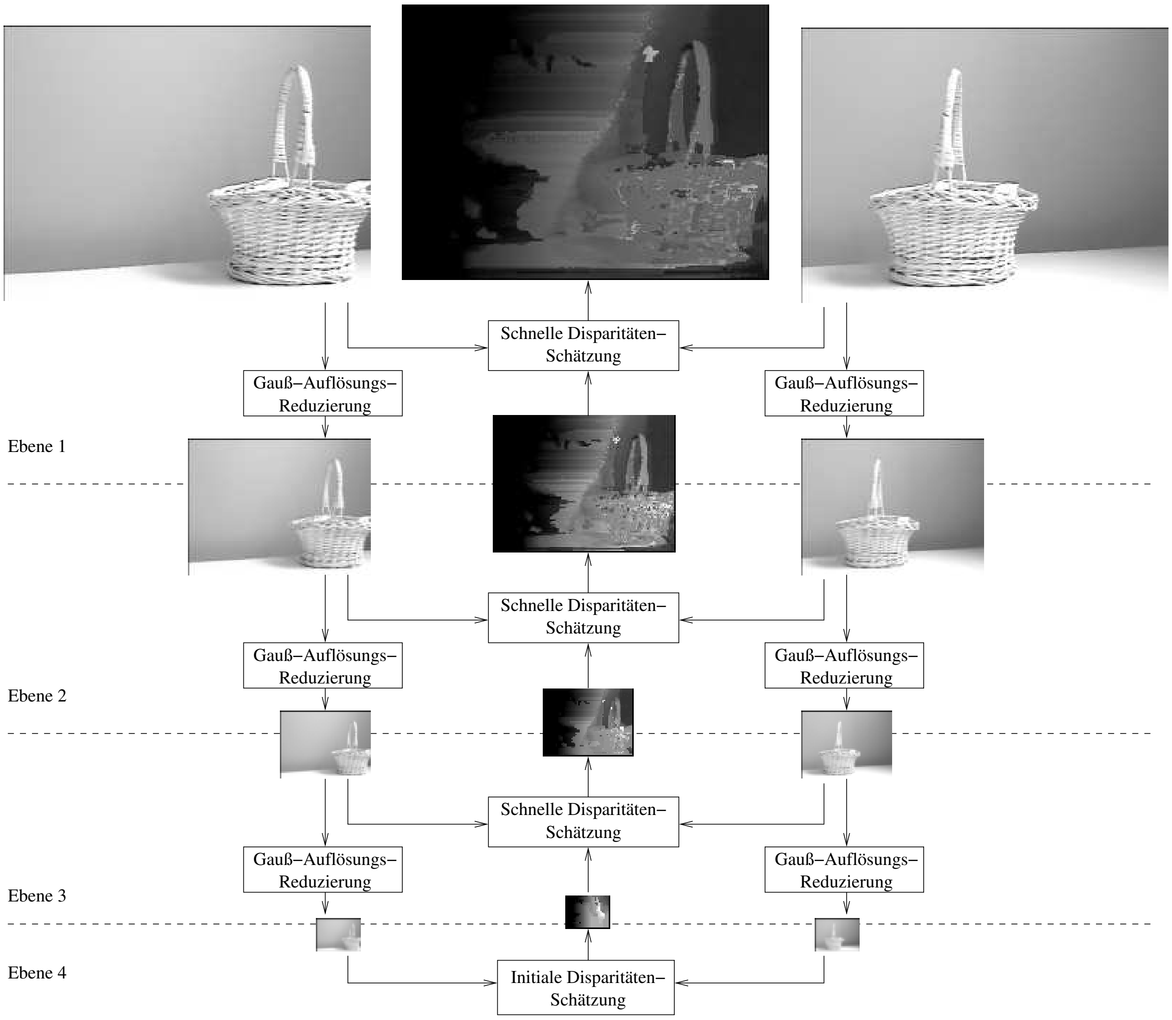

Abbildung 3.4: Datenfluß im Disparitätenschätzer

berechnet. Die schnelle Disparitätenschätzung (Definition 3.4.4) berechnet anhand dieses Tiefenbilds und den entsprechend höher aufgelösten Bildern der Ebene G_{k-1} das Tiefenbild $T_{k-1}(G_{k-1})$ und ermöglicht somit den Schritt zur nächsthöheren Stufe der Pyramide. Dieser Schritt wird bis zur obersten Ebene iterativ wiederholt.

Bei einer Pyramide mit k Ebenen muss daher einmal $disparity(x, y)$ nach 3.4.3 mit dem Aufwand $O(b_k^2 \cdot h_k)$ und $(k-1)$-mal $disparity_{G_n}(x, y)$ mit dem Aufwand $O(b_n \cdot h_n)$ berechnet werden. Für die Bildbreiten und -höhen gilt $b_n = b/r(n)$ und $h_n = h/r(n)$. Dazu kommt noch der Berechnungsaufwand für die Gauß-Pyramide. Ist $r(n)$ eine Exponentialfunktion, beträgt er $O(b \cdot h \cdot \log k)$.

Der Gesamtaufwand ist daher

$$O\left(b \cdot h \log k + \left(\frac{b}{r(k)}\right)^2 \frac{h}{r(k)} + \sum_{n=0}^{k-1} \frac{b \cdot h}{r(n)^2}\right) =$$

$$O\left(b \cdot h \left(\log k + \frac{b}{r(k)^3} + \sum_{n=0}^{k-1} \frac{1}{r(n)^2}\right)\right)$$

Da die Bildgröße auf der untersten Ebene deutlich größer als die Blockgröße $(\varepsilon_x, \varepsilon_y)$ sein muss, ist die Anzahl der Ebenen k begrenzt. Wählt man k in Abhängigkeit der Eingangsauflösung so, dass die Bildgröße $(b_k, h_k) = (b/r(k), h/r(k))$ der untersten Ebene annähernd konstant bleibt, wird für $r(n) = 2^n$ der Term $b/r(k)^3$ sehr klein. Der Summenterm $\sum_{n=0}^{k-1} \frac{1}{2^{2n}}$ hat die obere Schranke $\frac{3}{2}$ und kann daher vernachlässigt werden. Der Aufwand ist somit annähernd $O(b \cdot h)$.

3.4.3 Qualitätsmaß für die Disparitätenschätzung

Wie leicht einzusehen ist, hängt die Verlässlichkeit und Genauigkeit der Schätzung sehr stark von der Struktur des Stereobildpaars bzw. der aufgenommenen Szene ab. Große strukturarme Flächen erlauben ebensowenig eine verlässliche Schätzung der Disparität wie sich über den ganzen Bildbereich wiederholende Strukturen oder stark unterschiedliche Verdeckungen im linken und rechten Bild. Es ist daher ratsam, nicht nur eine Schätzung der Disparität durchzuführen, sondern für jeden Schätzwert ein Maß der Qualität der Schätzung anzugeben.

Der Schätzwert für die Disparität wurde als das Argument für den minimalen Umgebungs-Abstand in einem gegebenen Suchintervall angegeben (Definitionen 3.4.3 und 3.4.4). Soll nun die Disparität an einer Stelle im Raum mit wenig Struktur angegeben werden, wird sich dieser der Umgebungs-Abstand für die tatsächliche Disparität nicht stark von den Abständen anderer Disparitäten unterscheiden - Fehlschätzungen durch ungleiche Belichtung der Bilder, Reflexionen oder andere Störungen sind hier also besonders wahrscheinlich. Ist das Minimum des Umgebungsabstand einer bestimmmten Disparität aber deutlich kleiner als die Abstände anderer Disparitäten, spricht das für eine gute Schätzung, da diese Bildstelle offensichtlich in beiden Teilbildern eine recht einzigartige und gut wiederzufindende Struktur hat.

Als Qualitätsmaß für die Schätzung der Disparität an einer bestimmten Bildstelle kann daher der Quotient aus dem Mittelwert aller Abstände über das gesamte Suchintervall und dem minimalen Abstand verwendet werden:

Definition 3.4.5 (Qualitätsmaß zur Disparitätenschätzung) *Sei I das Suchintervall an der Stelle (x, y) (vgl. Definition 3.4.4). Ein Qualitätsmaß $q(x, y)$ für die Dispa-*

ritätenschätzung disparity(x, y) *ist*

$$q(x,y) := \frac{\sum_{i \in I} d_{(\varepsilon_x,\varepsilon_y)}(x,y,x-i,y)}{|I| \cdot d_{(\varepsilon_x,\varepsilon_y)}(x,y,x-disparity(x,y),y)}$$

Die Abbildung 3.5 zeigt eine Visualisierung des Qualitätsmaßes für die Disparitätenschätzung. dargestellt. Es ist deutlich erkennbar, dass die Qualität strukturierter Bildteile hoch eingestuft wird, während einfarbigen Flächen nur geringe Qualität zugesprochen wird.

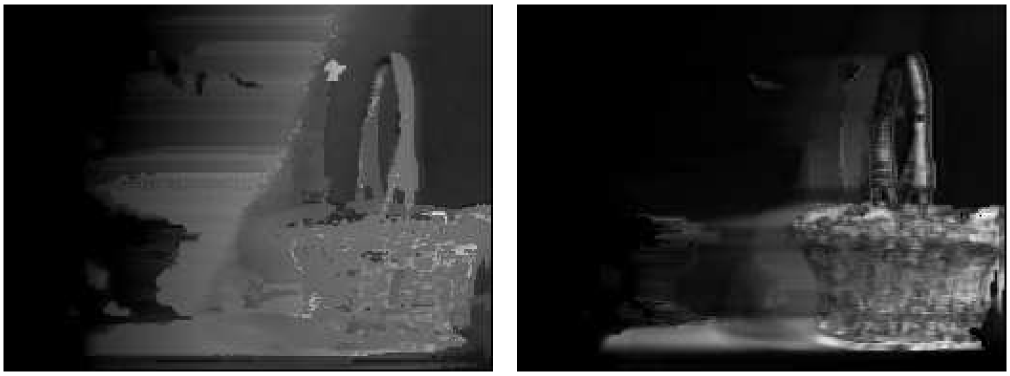

Abbildung 3.5: Tiefenbild und zugehöriges Qualitätsbild eines Korbs. Große Werte für das Qualitätsmaß erscheinen hell.

3.4.4 Implementierung des hierarchischen Blockmatchings

Für dieses Verfahren wurden drei Filtermodule für die Testumgebung *FilterGUI* implementiert: Den Filter `Gauß-Auflösungsreduzierung` (Quelldatei "FGaussReduce.java")zur Berechnung der Gauß-Pyramide, den initialen Disparitätenschätzer `3D-Korrelierung` (Quelldatei "F3DCorrelator.java") und den schnellen Disparitätenschätzer `3D-Korrelierung mit Tiefenvorgabe` (Quelldatei "F3DPyramidCorrelator.java"). Aus Geschwindigkeitsgründen wurden die Kernmethoden auch in C implementiert und mit JNI, dem *Java Native Interface*, eingebunden. Die C-Methoden befinden sich in der zugehörigen C-Bibliothek "imageprocessing".

Der Filter `Gauß-Auflösungsreduzierung` hat einen Eingang und einen Ausgang vom Typ `ImageData`, als Parameter können der Reduzierungsfaktor r die Größe der Gauß-Filtermatrix angegeben werden. Dabei wird im Bilddatensatz der Skalierungsfaktor zum Originalbild mitgespeichert, um die Kalibrierungsdaten korrekt umrechnen zu können.

Die beiden Filter zur Disparitätenschätzung haben jeweils einen Eingang für das linke und rechte Bild des Stereobildpaars. Die Blockgröße in x- und y-Richtung für das Umgebungs-Abstandsmaß sowie die Schrittweite der Abtastung können als Parameter eingestellt werden. Der schnelle Disparitätenschätzer hat zusätzlich einen Eingang für das Tiefenbild der untergeordneten Ebene; dieser Eingang ebenfalls vom Typ `ImageData`. Beide Filter haben jeweils einen Ausgang für ein Tiefenbild (*"height"*, entspricht der Disparität) und ein zugehöriges "Qualitätsbild" *"confidence"*, das für jeden Bildpunkt des Tiefenbilds ein Qualitätsmaß der Schätzung enthält.

Mit diesen drei Filtern können beliebig viele Ebenen erzeugt werden. Eine Ebene enthält zwei Gauß-Auflösungsreduzierungsfilter und einen schnellen Disparitätenschätzer mit Tiefenvorgabe, und sie erhält von oben das linke und rechte Bild sowie von unten ein niedriger aufgelöstes Tiefenbild.

Zunächst wird das Stereo-Bildpaar mit dem Auflösungsreduzierungsfilter verkleinert und nach unten an die nächste Ebene weitergereicht, bis am unteren Ende der initiale Disparitätenschätzer ein stark verkleinertes Stereobildpaar erhält. Dieser berechnet daraus ein Tiefenbild gleicher Größe, das er an die nächsthöhere Ebene weiterreicht. Diese Ebene berechnet aus den Stereobildern, die hier in der nächsthöheren Auflösung vorliegen, und dem Tiefenbild ein größeres Tiefenbild, das wiederum an die nächsthöhere Ebene weitergereicht wird (vgl. Abbildungen 3.4 und 3.6).

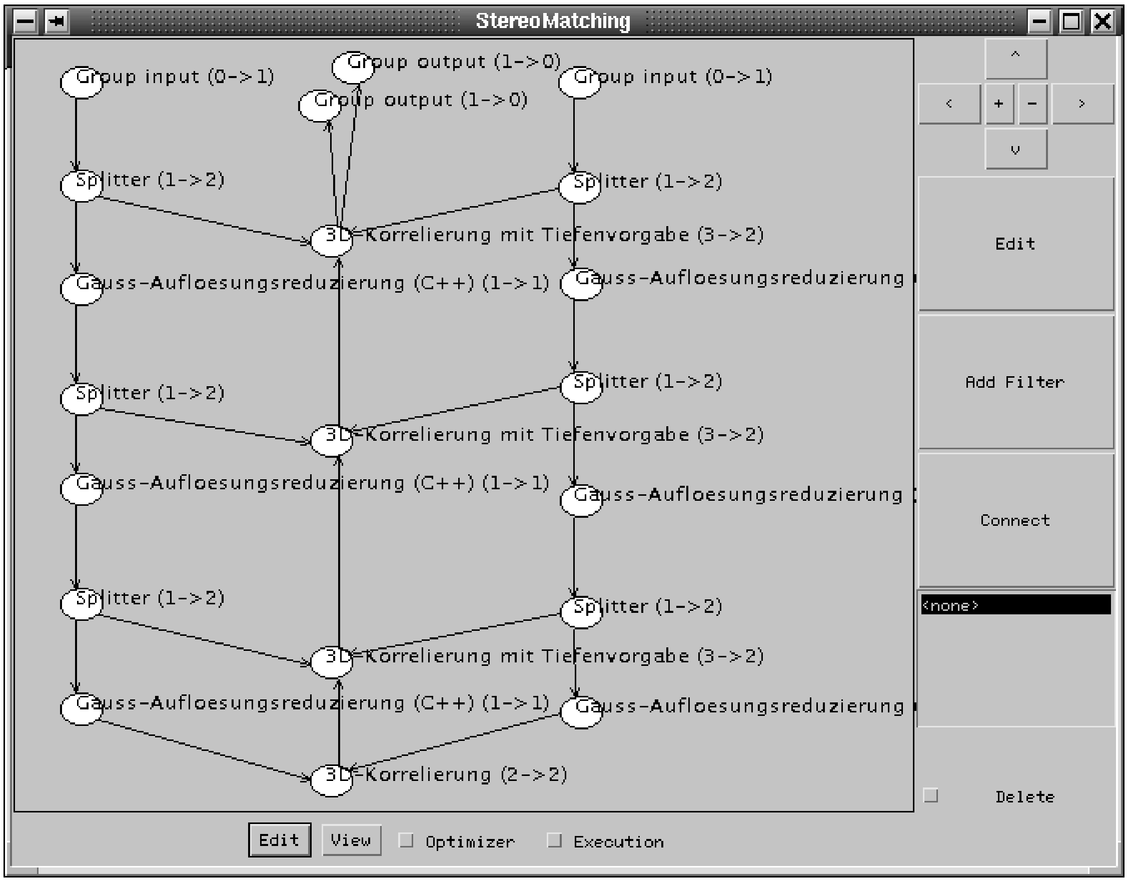

Abbildung 3.6: Disparitätenschätzer als FilterGUI-System

3.4.5 Erkenntnisse aus der Implementierung

Erste Tests mit der Implementierung der Disparitätenschätzung ergaben brauchbare Ergebnisse, bestätigten aber leider auch den sehr hohen Berechnungsaufwand. Wie erwartet reduzierte sich die Ausführungszeit bei der hierarchischen Disparitätenschätzung (3.4.2) von mehreren Minuten auf einige Sekunden (abhängig von der Bildgröße). Auf einem AMD Athlon-Prozessor mit 800 Mhz Taktrate wurden aber immer noch 7-8 Sekunden bei einer Auflösung von 384x287 Pixeln benötigt. Durch weitere Optimierungen läßt sich dieser Wert verbessern, allerdings sind aufgrund der enormen Datenmengen keine Geschwindigkeitssteigerungen um Größenordungen zu erwarten. Eine direkte Einbindung dieses Verfahrens in den Regelprozeß ist derzeit also noch nicht möglich. Lediglich mit Spezialhardware ist eine Zykluszeit von Sekundenbruchteilen vorstellbar. Es erschließen sich dennoch vielfältige Anwendungsmöglichkeiten, auf die im nächsten Kapitel eingegangen wird.

Wie sich schnell herausstellte, ist die Qualität des errechneten Tiefenbildes stark von der Belichtung der beiden Einzelbilder abhängig. Um ein gutes Ergebnis zu bekommen, müssen Objekte in beiden Bildern gleich hell abgebildet werden. Die verwendeten Kameras sind per Software konfigurierbar bezüglich Belichtungszeit (Shutter), Blende, Weißabgleich, Bildverstärkung und Helligkeitskorrektur. Die automatische Belichtungseinstellung erwies sich jedoch als problematisch, da bei den zwangläufig verschiedenen Bildausschnitten die automatisch ermittelten Belichtungsparameter beider Kameras voneinander abweichen. Die Deaktivierung der Belichtungsautomatik unter Verwendung fester Parameter stellt keine Lösung dar, da bei subjektiv nur wenig helleren oder dunkleren Verhältnissen die Bilder stark über- bzw. unterbelichtet und somit unbrauchbar werden.

Um trotz automatischer Einstellung gleiche Helligkeiten in beiden Bildern zu erreichen, wurde bei beiden Kameras die automatische Wahl der Belichtungszeit (Shutter) aktiviert, die analoge und digitale Bildverstärkung (AGC, Automatic Gain Control) wurde jedoch abgeschaltet. Der Shutter beherrscht nur diskrete, klar abgestufte Belichtungszeiten. Dadurch wählen beide Kameras trotz leichter Unterschiede der Bildausschnitte die gleiche Belichtungsstufe, ohne dass durch die stufenlose Nachregelung des AGC ungleiche Parameter zustande kommen. Natürlich gibt es Grenzfälle, in denen unterschiedliche Belichtungen zustande kommen; mit der vorhandenen Hardware läßt sich das aber nur schwer verhindern. Denkbar wäre ein Stereokamerapaar mit gemeinsamer und kalibrierter Belichtungsautomatik, oder eine Softwarelösung, bei der die Kameras über die vorhandene Schnittstelle angesteuert werden können.

Verfahrensbedingt liefert der hier verfolgte Ansatz im Gegensatz zu kantenbasierten Verfahren besonders gute Ergebnisse bei stark texturierten Oberflächen und gewölbten Flächen. Probleme ergeben sich bei großen einfarbigen Flächen und bei Reflexionen. Natürlich lässt sich die Qualität unter Verwendung von strukturiertem Licht stark verbessern. Damit eine effektive Verbesserung erreicht wird, muss das strukturierte Licht heller als das vorhandene Umgebungslicht sein. Der Einsatz eines Systems mit strukturiertem Licht auf einem humanoiden Roboter kommt somit nicht in Frage, da bei normalen Beleuchtungsverhältnissen eine sehr helle Strukturlichtquelle verwendet werden müsste. Die Blendwirkung und der hohe Stromverbrauch sind der Akzptanz und Verwendbarkeit abträglich.

Die Ergebnisse der Disparitätenschätzung sind schwer zu bewerten, da die geschätzte Disparität geeignet mit dem ermittelten internen Qualitätsmaß verknüpft werden muss. Es ist sinnvoller, eine Qualitätsbewertung später auf Basis der ermittelten 3D-Daten durchzuführen.

3.5 3D-Punktextraktion

Der nächste Schritt ist die Berechnung von Punkten im 3D-Raum aus den Disparitäten. Die gewonnenen Punkte müssen in einer geeigneten Datenstruktur abgelegt und geglätted werden, um Störungen zu minimieren.

3.5.1 Triangulation

Aus dem gewonnenen Tiefenbild können mit Triangulation unter Verwendung der Kalibrierparameter Punkte im dreidimensionalen Raum bestimmt werden. Da die Werte für die Disparitäten im Tiefenbild sich auf das linke Bild beziehen, entspricht das Koordinatensystem dem der linken virtuellen Kamera (Abbildung 3.7).

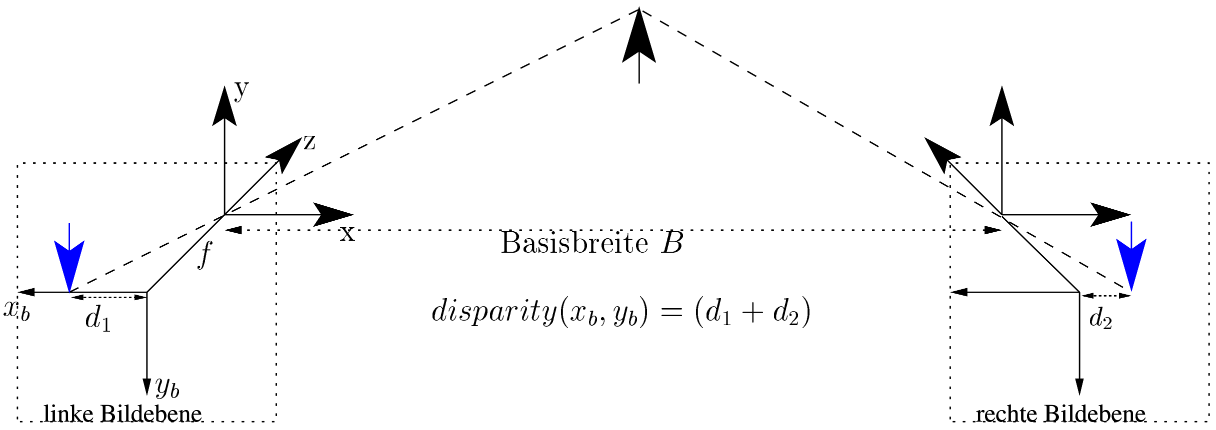

Abbildung 3.7: Punkt-Triangulation

Wird ein Punkt $P = (x, y, z)$ im Raum auf den Bildpunkt $P_l(x_b, y_b, 1)$ im linken Kamerabild abgebildet, kann mit Hilfe der geschätzten Disparität der zugehörige Bildpunkt P_r im rechten Kamerabild ermittelt werden. Da durch die in Abschnitt 3.3 beschriebene Bildkorrektur von einer idealen Lochkamera ausgegangen werden kann, ist P der Schnittpunkt zweier Geraden, die jeweils durch den Brennpunkt und den entsprechenden Bildpunkt der linken und rechten Kamera gegeben sind.

Die Rekonstruktion von $P = (x, y, z)$ aus der Disparität im Punkt P_l und der Basisbreite B sowie der Brennweite f berechnet sich daher folgendermaßen:

$$P = \frac{B \cdot f}{disparity(x_b, y_b)} P_l$$

Die Triangulation wurde in dem Filtermodul `3D-Punkte-Extraktion` (Quelldatei "F3DExtractor.java") implementiert. Um Fehlmessungen zu reduzieren, wird das Qualitätsmaß *"confidence"* für jeden einzelnen Punkt mitberücksichtigt. Das Tiefenbild wird mit einem per Parameter wählbaren Raster abgetastet, wobei pro Rastereinheit der Punkt mit maximalem Qualitätsmaß verwendet wird. Liegt das Qualitätsmaß eines Punktes unter einem wählbaren Schwellwert, wird der Punkt verworfen. Um eine spätere Filterung der Punkte zu vereinfachen, werden zudem für jeden Punkt die unmittelbaren Nachbarpunkte mitgespeichert.

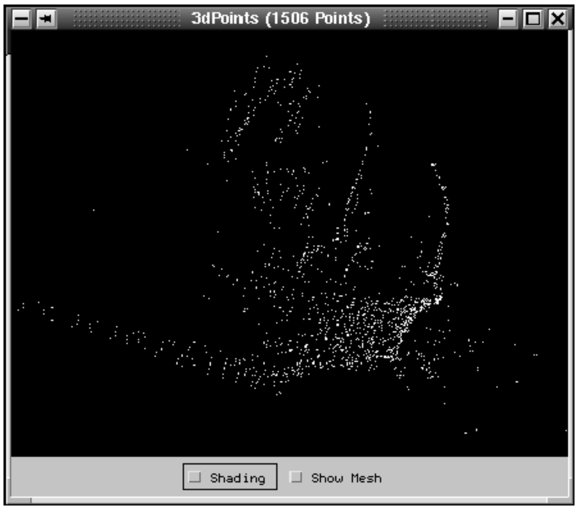

Abbildung 3.8: Die extrahierten Punkte eines Korbs

3.5.2 Repräsentation der 3D-Punkte

Ein 3D-Punkt ist definiert durch seine 3D-Koordinaten $p = (x, y, z)$. Im Rahmen der späteren Verarbeitung wird die Orientierung der Objektoberfläche in einem Punkt benötigt. Ein 3D-Punkt wird daher repräsentiert durch ein Tupel (p, n), wobei der Normalenvektor n die Tangentialebene im Punkt an der Objektoberfläche beschreibt. Diese Tupel werden als *"Orientierte Punkte"* bezeichnet [4].

Die Implementierung eines 3D-Punktes (Klasse "Point3D.java") enthält neben den Koordinaten x, y, und z auch die Orientierung der Objektoberfläche in dem Punkt als Normalenvektor (dx, dy, dz). In dieser Klasse ist außerdem eine Auflistung der benachbarten

Punkte vorgesehen, daneben existiert ein Zeiger auf einen korrespondierenden Punkt. Das Qualitätsmaß und der Grauwert des entsprechenden Bildpunktes sind ebenfalls enthalten. Daneben existieren Methoden zum Addieren, Multiplizieren, Verschieben und Rotieren von Punkten sowie zur Abstands- und Spin-Abstandsberechnung.

Eine Menge von 3D-Punkten wird in der Klasse `PointData` verwaltet, die eine Unterklasse von `ProcessData` ist. Die Punkte werden sowohl in einer Liste als auch in einem Octree (Baum mit maximal acht Abwärtskanten pro Knoten) gespeichert. Hinzugefügte Punkte werden rekursiv in den bestehenden Octree einsortiert, wobei ein Knoten entweder einen einzigen Punkt oder einen Unterbaum enthält. Die Knoten des Baums werden Octanten genannt (Quelldatei "Octant.java"). Liegt der Punkt außerhalb des bestehenden Octrees, wird der Baum automatisch aufsteigend erweitert, bis der Punkt enthalten ist. Für die Octree-Verwaltung ist zusätzlich ein Zeiger auf den zugehörigen Octree-Knoten vorhanden.

Durch den Octree und die Verweise auf die nächsten Nachbarn ist ein schneller und einfacher Zugriff auf die Umgebung eines Punktes möglich, was bei späteren Filter- und Glättungsalgorithmen benötigt wird.

3.5.3 Filterung und Glättung von 3D-Punktwolken

Für die später beschriebene Objekterkennung werden orientierte Punkte benötigt, die möglichst gleichmäßig auf den Objektoberflächen verteilt sind. Die aus Tiefenbildern gewonnenen 3D-Punktwolken sind jedoch stark verrauscht, was eine stabile Bestimmung der Oberflächennormalen erschwert. Zudem liegen die Punkte mit zunehmender Entfernung zur Kamera weiter auseinander.

Filterung mit einem Octree

Der im Folgenden beschriebene Algorithmus basiert auf einem Octree und erzeugt eine Punktwolke mit den gewünschten Eigenschaften. Als Parameter wird die gewünschte minimale Kantenlänge k_{min} für Octanten (Würfel, die einem Octree-Knoten entsprechen) vorgegeben, die zugleich die mittlere Distanz zwischen Oberflächenpunkten vorgibt.

Von der Wurzel des Octree wird eine Tiefensuche durchgeführt, bis die minimale Kantenlänge k_{min} im Knoten K_{min} erreicht ist. Aus allen Punkten im Unterbaum mit Wurzel K_{min} wird der Schwerpunkt errechnet, wobei die Punkte nach ihrem Qualitätsmaß gewichtet werden. Dieser Schwerpunkt wird zur Ergebnismenge hinzugefügt. Um einen Oberflächen-Normalenvektor bestimmen zu können, werden für jeden der acht Unterbäume von K_{min} nach gleichem Verfahren Schwerpunkte gebildet, falls sie eine gewisse minimale Anzahl an Punkten enthalten. Ergeben sich mindestens drei Schwerpunkte, kann ein Normalenvektor bestimmt werden, ansonsten wird der Punkt verworfen.

Der Hauptvorteil dieses Algorithmus ist seine Geschwindigkeit. Sowohl der Knoten K_{min} als auch die 3D-Punkte seiner Unterbäume werden per Tiefensuche ermittelt. Der Aufwand einer erschöpfenden Tiefensuche ist bei n Punkten $O(n \log n)$. Leider ist das Ergebnis nicht optimal.

Die 3D-Punkte sind nicht gleichverteilt im Raum. Abgesehen von Rauscheinflüssen liegen sie auf den Objektoberflächen. Durch die Würfelform der Octanten und deren Ausrichtung im Raum hängt der mittlere Abstand der 3D-Punkte im Ergebnis von der Lage der Oberfläche im Raum ab. Liegt sie diagonal zu den Octanten, ist der Abstand entsprechend der Flächen- oder Raumdiagonalen der Octanten größer. Der Algorithmus berücksichtigt auch keine Nachbarschaftsbeziehungen oder bekannte Eigenschaften von Punkten aus Tiefenbildern. Da sich bei der Stereoskopie aus Blickrichtung der Kamera keine ermittelten Punkte überdecken können, sind Ausreißer in Tiefenrichtung leicht erkennbar und können berücksichtigt werden.

Dennoch konnten in der Praxis mit diesem Algorithmus bei starker Reduzierung der Punktdichte brauchbare Oberflächennormalen bestimmt werden (*"3D Octree Equalizer"*, Quelldatei "F3DOctreeEqualizer.java").

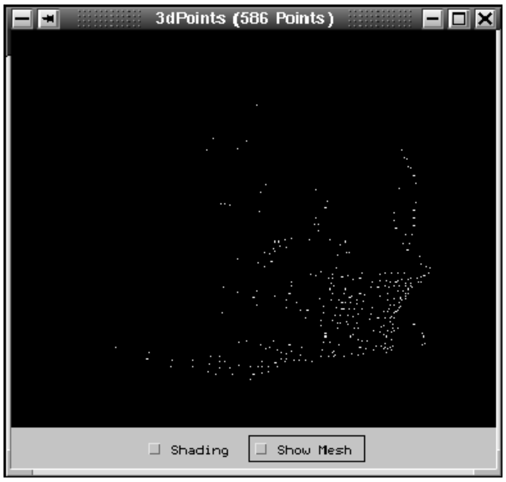

Abbildung 3.9: Die Punkte eines Korbs nach Glättung mit dem Octree-Verfahren

Umgebungsbezogene Filterung

Da die 3D-Punkte aus einem Tiefenbild stammen, ist es einfach, zu jedem Punkt Referenzen auf benachbarte Punkte mitzuspeichern. Durch Verfolgung der Punktreferenzen kann schnell eine Menge U von Punkten aufgebaut werden, die innerhalb eines maximalen Abstands zu einem bestimmten Punkt liegt.

Aus der Ursprungsmenge G_0 kann eine Punktmenge G' mit einem gegebenen mittleren Punktabstand d_m berechnet werden, indem man die Umgebungsmenge U_p für jeden Punkt $p \in G$, der weiter als d_m von allen Punkten aus G' entfernt ist, berechnet. Der Schwerpunkt von U bildet einen neuen Punkt, der zu G' hinzugefügt wird. Bei der Berechnung des Schwerpunkts werden alle Punkte entsprechend ihrem Abstands zu p mit einer Gauß-Funktion gewichtet.

Da der initiale Mittelpunkt p jeweils den Mindestabstand d_m zu allen anderen Punkten hat, ergibt sich im Mittel ein Punktabstand von d_m. Durch die Gaußgewichtete Schwerpunktbildung werden Rauscheinflüsse reduziert. Zusätzlich können Punkte ignoriert werden, falls die Menge naheliegender Punkte U_p zu wenige Elemente enthält.

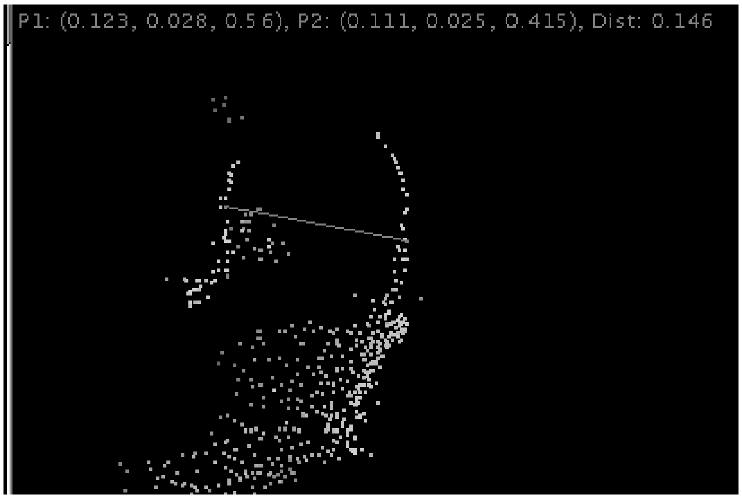

Abbildung 3.10: Orientierte Punkte eines Korbs nach umgebungsbezogener Filterung und Vermessung des Henkeldurchmessers.

3.5.4 Bestimmung von Oberflächennormalen

Ist die Punktwolke geeignet geglättet und sind Gruppen nebeneinanderliegender Oberflächenpunkte bekannt, können Oberflächen-Normalenvektoren aus dem Mittelwert der Kreuzprodukte der Verbindungsvektoren der Punkte bestimmt werden. Um ein stabileres Ergebnis zu erhalten, ist es sinnvoll, den Schwerpunkt einer Punktegruppe als neuen orientierten Punkt zu verwenden und zur Normalenbestimmung die Kreuzprodukte jeweils zweier Verbindungsvektoren vom Schwerpunkt zu den Gruppenpunkten zu bilden. Da die Punkte aus einem Tiefenbild stammen, können keine von der Kamera abgewandten Normalenvektoren existieren; gegebenenfalls müssen die Ergebnissvektoren der Kreuzprodukte invertiert werden. Der Mittelwert ergibt nach anschließender Normierung eine initiale Schätzung der Oberflächennormalen im Schwerpunkt. Durch die Eigenschaften des Kreuzprodukts werden weiter auseinanderliegende Punkte stärker gewichtet, wodurch Restrauschen weniger Einfluß auf das Ergebnis hat.

Gegebenenfalls kann die initiale Schätzung durch ein iteratives Verfahren verfeinert werden, allerdings auf Kosten der Ausführungszeit. Versuche haben jedoch gezeigt, dass die initialen Werte ausreichend genau für die Weiterverarbeitung sind (Abbildung 3.10).

3.6 3D-Objekterkennung

Das bisher beschriebene System erzeugt aus Stereobildern Punkte im dreidimensionalen Raum. Diese Punkte erlauben Rückschlüsse auf mögliche Hindernisse für den Roboter, auch können nach einer dreidimensionalen Segmentierung Aussagen über Objektlage und Ausdehnung gemacht werden. Um die bisherigen Ergebnisse in Anbetracht der Anforderungen an einen humanoiden Serviceroboter sinnvoll nutzen zu können, werden spezifischere Informationen über Objekttyp, Position und Orientierung benötigt. Hierzu ist es notwendig, Korrespondenzen zwischen erfassten Punkten und Objektpunkten in einer Datenbank zu finden. Ein interessantes Verfahren zur Bestimmung von Punktkorrespondenzen wird in [4] vorgestellt, das im folgenden erläutert wird.

3.6.1 Bestimmung von Punktkorrespondenzen mit SpinImages

Dieses Verfahren beruht auf einer translations- und rotationsunabhängigen Abbildung von Paaren orientierter Punkte nach R^2.

Definition 3.6.1 (Spin-Abbildung für orientierte Punkte [4]) *Ein orientierter Punkt $\psi := (p, n)$ sei gegeben durch seinen Ortsvektor p und den Normalenvektor n der Tangentialebene zur Objektoberfläche in p. Die Spin-Abbildung $S_\psi : R^3 \longrightarrow R^2$ ist gegeben durch*

$$S_\psi(x) \longmapsto (\alpha, \beta) = \left(\sqrt{\|x-p\|^2 - (n \cdot (x-p))^2}, n \cdot (x-p) \right)$$

Ist der Punkt x' die orthogonale Projektion von x auf die Tangentialebene in ψ , entspricht α dem (immer positiven) Abstand $d(p, x')$ und β der Differenz $x - x'$, die sowohl positiv als auch negativ sein kann.

Offensichtlich ist diese Abbildung invariant gegenüber Rotationen um n. Da $S_\psi(x)$ nur von von der relativen Lage von x zu ψ abhängt, ist die Abbildung sowohl translations- als auch rotationsinvariant.

Wählt man aus den Oberflächenpunkten eines Objekts einen orientierten Punkt ψ, erhält man nach Abbildung aller anderen Oberflächenpunkte mit der Spin-Abbildung eine für das Objekt und ψ charakteristische Punkteverteilung in R^2, die unabhängig von Lage und Orientierung des Objekts im Raum ist. Um den Vergleich solcher 2D-Punkteverteilungen zu vereinfachen, kann man die Punkthäufigkeiten in definierten Flächen bestimmen. Idealerweise teilt man die Ebene in rechteckige Felder fester Größe auf und akkumuliert die

Abbildung 3.11: Beispiel eines SpinImages

hineinfallenden Punkte. Man erhält ein zweidimensionales Histogramm, das SpinImage genannt wird.

Ein Maß für die Ähnlichkeit zweier SpinImages ist ihre lineare Abhängigkeit - sie sind gleichwertig, wenn sie sich bis auf einen konstanten Faktor nicht unterscheiden. Der direkte Vergleich von Häufigkeiten ist nicht empfehlenswert, da sie stark von der Auflösung der Aufnahme und dem mittleren Abstand und Anzahl der verwendeten Oberflächenpunkte abhängt. Betrachtet man die Häufigkeiten einander entsprechender Felder in P und Q als Punkte in R^2, kann man mit der Least-Square-Methode eine am besten passende Gerade durch diese Punkte legen. Die Qualität der Korrelation läßt sich anhand des folgenden Korrelationskoeffizienten angeben, der sich aus der quadratischen Abweichung von den Daten und dieser Geraden berechnet:

$$R(P,Q) = \frac{N\sum p_i q_i - \sum p_i \sum q_i}{\sqrt{\left(N\sum p_i^2 - \left(\sum p_i\right)^2\right)\left(N\sum q_i^2 - \left(\sum q_i\right)^2\right)}}$$

Der Wert von R bewegt sich zwischen -1 (keine Korrelation) und 1 (vollständige Korrelation).

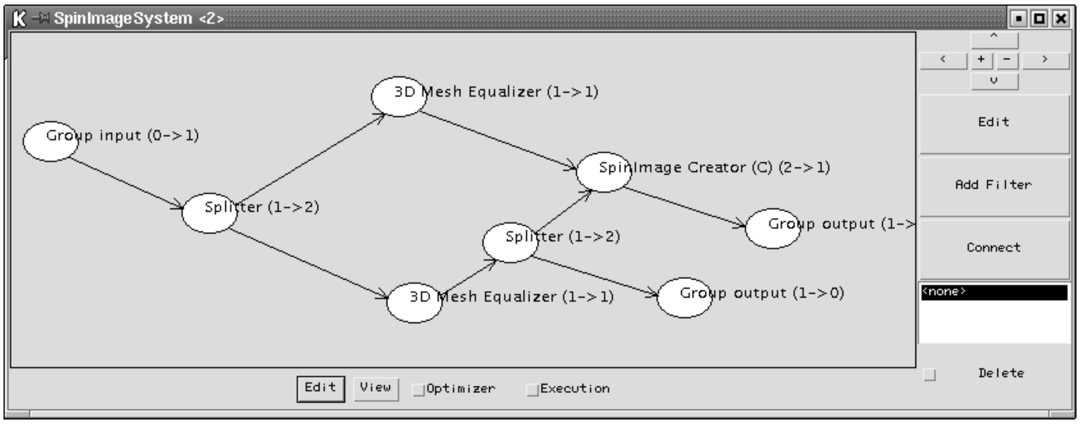

Abbildung 3.12: Das Subsystem zur Punktfilterung und SpinImage-Berechnung

Existiert nun eine Datenbank mit Mengen orientierter Punkte von bekannten Objekten, können die entsprechenden Mengen von SpinImages offline im Voraus berechnet werden. Jedes aus der Szene erzeugte SpinImage wird mit allen SpinImages der Datenbank verglichen.

Korrespondierende Punkte aus Szene und Modell werden einen maximalen Korrelationskoeffizienten aufweisen, daher findet man den korrespondierenden Punkt in der Datenbank durch Maximumsuche. Falls das beobachtete Objekt noch nicht in der Datenbank vorhanden ist, verhindert ein Schwellwert Fehlzuordnungen.

Aus den Punktkorrespondenzen können vielfältige Informationen gewonnen werden. Zum einen können leicht die in der Szene gefundenen Objekte anhand der Anzahl vorhandener Punktkorrespondenzen angegeben werden, zusätzlich können aber auch Lage und Orientierung der Objekte bestimmt werden.

3.6.2 Gruppierung geometrisch konsistenter Punktkorrespondenzen

Um zuverlässig die durch die korrespondierenden Punkte beschriebene Transformation bestimmen zu können, müssen die Punktkorrespondenzen geometrisch konsistent sein. Da Fehlzuordnungen nicht auszuschließen sind und sich auch mehrere Objekte in der Szene befinden können, müssen inkonsistente Korrespondenzen entfernt werden und konsistente Partitionen gefunden werden. Als Konsistenzkriterium kann der Abstand von Punkten verwendet werden. Da aber orientierte Punkte vorliegen, erhält man eine restriktivere Bedingung unter zusätzlicher Verwendung der Oberflächennormalen. Es liegt nahe, hierfür die Spin-Abbildung zu verwenden. Demnach sind zwei Punktkorrespondenzen (s_1, m_1) und (s_2, m_2) konsistent, falls

$$\|S_{m_1}(m_2) - S_{s_1}(s_2)\| < k$$

und

$$\|S_{m_2}(m_1) - S_{s_2}(s_1)\| < k$$

erfüllt sind. Der Schwellwert k sollte ein- bis zweimal so groß wie der mittlere Abstand der verwendeten orientierten Modellpunkte gewählt werden.

Folgender Algorithmus ermittelt eine Partitionierung der korrespondierenden Punkte in geometrisch konsistente Gruppen:

- Gegeben sei eine Menge $K = (m_i, s_i)$ von korrespondierenden Punkten. Zunächst wird diese Menge absteigend nach der Qualität der Korrespondenz (Ähnlichkeit der SpinImages) sortiert.

- Nun wird die Menge G_1 aller Korrespondenzen bestimmt, die zur besten Korrespondenz geometrisch konsistent sind. Falls G_1 Paare von Korrespondenzen enthält, die inkonsistent sind, wird die Korrespondenz mit der niedrigeren Übereinstimmung entfernt. Im nächsten Schritt wird der Vorgang mit der Restmenge $K \backslash G_1$ wiederholt, bis alle Korrespondenzen einer Gruppe zugeordnet wurden.

Da für die Bestimmung einer starren euklidischen Transformation mindestens drei korrespondierende Punkte benötigt, können alle Gruppen mit weniger als drei Elementen ignoriert werden. Da Fehlzuordnungen und Fehlmessungen mit großer Wahrscheinlichkeit untereinander inkonsistent sind, werden sie somit stark reduziert.

3.6.3 Bestimmung der Szene-Modell-Transformation

Da eine starre euklidische Transformation (in diesem Fall eine Rotation mit anschließender Translation) in R^3 durch drei linear unabhängige korrespondierende Punkte eindeutig bestimmt ist, sollen zunächst nur drei Punktkorrespondenzen $K_i = (m_i, s_i)$ mit i=1,2,3 betrachtet werden, die geometrisch konsistent sind.

Um eine initiale Berechnung der Rotation R zu bestimmen, kann zur Veranschaulichung o.B.d.A. angenommen werden, dass m_1 und s_1 im Ursprung liegen. Das kann durch eine entsprechende Translation erreicht werden.

Zunächst wird eine Rotation R_0 um die y- und z-Achse bestimmt, die $\vec{s} = (x, y, z) = s_2 - s_1$ in die x-Achse überführt. Die Rotationswinkel β (um die y-Achse) und γ (z-Achse) können trigonometrisch mit der Arcustangens-Funktion bestimmt werden: [2]

$$\beta = \tan^{-1}\left(\frac{z}{x}\right), \qquad \gamma = \tan^{-1}\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$

Zur einfacheren Berechnung der gesuchten Rotation werden alle Szenen- und Modellpunkte mit R_0 transformiert.

Aufgrund der geometrischen Konsistenz gibt es eine Rotation R_1, die den Vektor $m_2 - m_1$ in den Vektor $s_2 - s_1$ überführt. Da $s_2 - s_1$ auf der x-Achse liegt, kann diese Rotation analog zu R_0 bestimmt werden. Transformiert man alle Szenenpunkte mit R_1, stimmen bereits m_1 und s_1 sowie m_2 und s_2 überein. Die dritte Korrespondenz hat jetzt nur noch einen rotatorischen Freiheitsgrad; die abschließende Rotation R_2 um die x-Achse kann leicht bestimmt werden. Die gesuchte Rotation R erhält man durch Hintereinanderausführen von R_1 und R_2, bzw. durch Matrixmultiplikation der entsprechenden Rotationsmatrizen: $R = R_2 \cdot R_1$. Um das Ergebnis zu verbessern, kann der Vorgang zur Mittelung mit anderen Korrespondenztripeln wiederholt werden.

Ist die Rotation bekannt, läßt sich die entsprechende Translation t aus der Menge der Korrespondenzen K leicht berechnen:

$$t = \sum_{i \epsilon K} s_i - R * \sum_{i \epsilon K} m_i$$

[2] Die beschriebene Winkelberechnung mit $tan^{-1}(z/x)$ ist für $x = 0$ nicht definiert und berücksichtigt die Vorzeichen von x und z nicht. Die Programmiersprachen C und Java stellen die Methode atan2 zur Verfügung, die diese Probleme behebt.

Diese initiale Schätzung kann als Startwert für einen Least-Square-Schätzer dienen, der den folgenden Ausdruck minimiert:

$$E_T = \sum_{i \epsilon K} \|s_i - (Rm_i + t)\|$$

Eine Lösung in geschlossener Form wird von Horn in [2] beschrieben.

3.7 Abschließende Bewertung

Nachdem alle notwendigen Komponenten vorhanden waren, konnte das geplante Bildverarbeitungssystem (vgl. 3.2) zusammengesetzt werden. Die Abbildungen der verwendeten Subsysteme sind bei der Vorstellung der entsprechenden Verfahren zu finden (Abb. 3.3, 3.6, 3.12). Die Module arbeiten ordungsgamäß zusammen, der gesamte Ablauf von den Kamerabildern bis zu Punktkorrespondenzen zur Datenbank mit anschließender Transformationsschätzung konnte realisiert werden.

3.7.1 Genauigkeit der 3D-Punktdaten

Die Präzision der 3D-Daten hängt von den Kameras, der verwendeten Auflösung und nicht zuletzt von der Struktur der Szene ab. Die verwendeten Kameras (vgl. 3.1.2) ermöglichen rechnerisch mit einer Brennweite von 910 Pixeln und einem Basisabstand von 15 cm bei einem Meter Entfernung eine Auflösung von 1mm lateral und 1cm in Tiefenrichtung. Bei einem Objektabstand von 0,5m verbessert sich die Tiefenauflösung auf 3,6mm, die laterale Auflösung halbiert sich auf 0,5mm.

In der Praxis werden diese Werte aufgrund von Messrauschen und geschwindigkeitsbedingter Auflösungsreduzierung nicht erreicht. Bei der Objekterkennung kommt es jedoch nicht so sehr auf die Vermessung von Details an, sondern vielmehr auf die Form des gesamten Objektes. Die beschriebenen Verfahren zur Punktfilterung reduzieren durch Mittelung die Detailtreue, gleichzeitig wird aber gerade in Tiefenrichtung durch Interpolation die Auflösung virtuell erhöht.

Mit halber Bildauflösung kann bei einem Objektabstand theoretisch eine Auflösung von 7,2mm erreicht werden. Da aber die Auflösung parallel zur Bildebene mit 1mm deutlich höher ist, kann aus dem Überschuss an Punkten eine geglättete Punktwolke mit 5mm mittlerem Punktabstand berechnet werden. Natürlich geht dabei Information verloren, durch die Interpolation werden aber Messfehler zum Teil ausgeglichen. Strukturen, die deutlich größer als die Auflösung sind, werden besser angenähert.

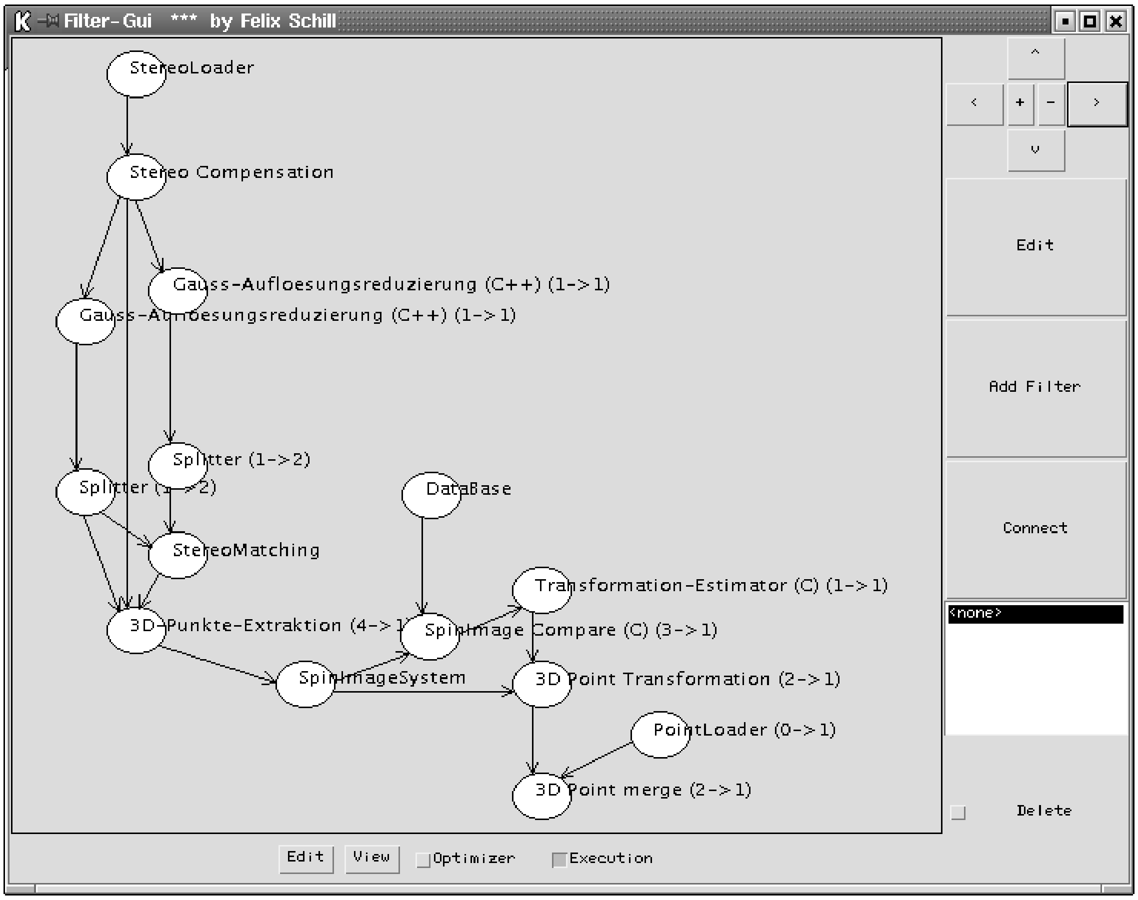

Abbildung 3.13: Das komplette Bildverarbeitungssystem

Eine exakte Überprüfung der 3D-Punkte stellte sich als problematisch heraus, da kein kalibrierter 3D-Scanner zur Verfügung stand. Eine manuelle Vermessung bestimmter Objektpunkte bestätigte eine Auflösung im Bereich von 5mm bei einem Objektabstand von 0,5m.

Da keine dreidimensionale vollständige Abtastung der Testobjekte zur Verfügung stand, mussten die Referenzdaten mit dem Bildverarbeitungssystem erstellt werden. Dazu wurden die Objekte vor einem einfarbigen Hintergrund aufgenommen und mit hoher Auflösung 3D-Punkte berechnet. Die extrahierten Punkte wurden nach sorgfältiger Glättung und Bestimmung der Oberflächennormalen in eine Datei abgespeichert.

Eine so erzeugte Referenzdatei zeigt das Objekt nur von einer Seite. Nach zu großer Drehung des Objekts konnten verständlicherweise kaum noch Korrespondenzen gefunden werden. Bei kleineren Rotationen oder Translationen wurden aber ausreichend viele korrekte Korrespondenzen gefunden.

Es zeigte sich, dass die Qualität der Korrespondenzen sehr stark von der Genauigkeit der

Oberflächennormalen abhängt. Kleine Abweichungen durch messbedingte Störungen oder zu geringe Auflösung führen schnell zu Fehlzuordungen. Durch das geometrische Gruppieren können sie zwar aussortiert werden, allerdings stehen dann meist nicht mehr genügend viele Korrespondenzen zur Transformationsschätzung zur Verfügung.

3.7.2 Eignung des Verfahrens

Besonders gut eignet sich das vorgestellte System für stark texturierte, asymmetrisch geformte Objekte. Das hierarchische Stereomatching benötigt für gute Ergebnisse eine charakteristische Oberflächenstruktur. Dadurch hebt sich das Verfahren von kantenbasierten Ansätzen ab, bei denen texturbedingt extrahierte Kanten die Objektzuordnung erschweren.

Andererseits eignet sich das StereoMatching nicht so gut, wo kantenbasierte Verfahren klare Vorteile haben: bei großen, einfarbigen Flächen.

Die Objekterkennung mit SpinImages hängt hauptsächlich von der Qualität der 3D-Punkte ab. Einzig rotationssymmetrische Objekte bereiten Schwierigkeiten, da die rotationsunabhängigen SpinImages nicht mehr einem bestimmten Oberflächenpunkt zuzuordnen sind.

Aufgrund der langen Ausführungszeiten kommen zeitkritische Anwendungen momentan nicht in Betracht. Testläufe ergaben stark schwankende Zeiten, abhängig von der Szenenstruktur und der Größe der Datenbank. Auf einem Athlon-Prozessor mit 800Mhz ergeben sich Zeiten zwischen 25 und 50 Sekunden.

Um eine flexible Bildverarbeitung zu erhalten, erscheint eine Kombination mit anderen Verfahren sinnvoll; so können die Vor- und Nachteile einzelner Verfahren geschickt ausgenutzt werden. In Frage kommende Anwendungen werden im nächsten Kapitel beschrieben.

Kapitel 4

Anwendungen und Erweiterungen

Das vorgestellte Bildverarbeitungssystem kombiniert einen passiven 3D-Sensor (Stereobildauswertung) mit einer robusten 3D-Objekterkennung. Neben einer Objektklassifizierung können Aussagen über Lage und Orientierung bekannter Objekte gemacht werden. Auch unbekannte Objekte können berücksichtigt werden, da eine dreidimensionale Abtastung berechnet wird. Dieses Kapitel widmet sich den sich ergebenden Anwendungen.

4.1 Anwendungsszenarien

4.1.1 Szenenanalyse und Objekterkennung

Um Manipulationsaufgaben mit einem humanoiden Roboter bewerkstelligen zu können, sind genaue Informationen über Objekte in der Szene nötig. Erreicht der Roboter einen Arbeitsplatz, sollte er zunächst eine umfassende Szenenanalyse durchführen.

Zur Verwaltung bekannter Objekte wird eine einfache Datenbank benötigt. Ein Objekt wird beschrieben durch eine Menge orientierter Punkte und weiteren Objektinformationen wie Name, Greifpunkte oder sonstige Merkmale. Für alle Objekte in der Datenbank können offline SpinImages berechnet werden, wie in Kapitel 3 beschrieben wurde.

Bei einer Szenenanalyse müssen anhand der online berechneten SpinImages aus der Stereobildauswertung Korrespondenzen zu Punkten in der Datenbank bestimmt und geometrisch konsistent gruppiert werden. Bei der Gruppierung können bereits die Objektzuordnungen aus der Datenbank berücksichtigt werden, somit ist die Zuordnung der Gruppen zu Objektbeschreibungen bereits getan. Wird nun wie bereits beschrieben die Transformation zwischen Szene und Modell bestimmt, ist die Objektlage und Orientierung relativ zur Repräsentation in der Datenbank bekannt. Diese Informationen können in ein abstraktes Umweltmodell eingefügt werden. Nicht klassifizierbare Punkte können in diesem Umweltmodell als Hindernisse vermerkt werden.

Sind zu einem Objekt Greifpunkte oder sogar eine Auswahl an Greiftrajektorien gespeichert, können diese anhand der ermittelten Rotations- und Translations-Parameter in den Szenenraum tranformiert und ausgeführt werden.

4.1.2 Einlernen unbekannter Objekte

Eine Objektdatenbank manuell zu erstellen ist sehr mühsam und zeitaufwändig. Zudem ist zusätzliche Hardware wie 3D-Scanner und CAD-Software erforderlich. Das hier vorgestellte Verfahren kann diese Tätigkeit weitgehend automatisieren.

Einzulernende Objekte sollten vor einem möglichst strukturlosen Hintergrund aufgenommen werden. Anhand des Qualitätsmaßes kann eine einfache Vorfilterung durchgeführt werden. Eine anschließende Segmentierung der 3D-Punkte erfasst somit zuverlässig die Punkte des einzigen sichtbaren Objektes.

Werden keine Punktkorrespondenzen zu einem in der Datenbank vorhandenen Objekt gefunden, kann von einem unbekannten Objekt ausgegangen werden. Die gefilterten orientierten Punkte werden zunächst als unvollständiges, unbekanntes Objekt in der Datenbank abgelegt. Wird das gleiche Objekt später aus einer geringfügig anderen Perspektive erfasst, können Korrespondenzen zu den unvollständigen Punktdaten gefunden werden. Anhand der aus den Punktkorrespondenzen ermittelten Transformation ist es möglich, die neugewonnenen Punkte mit den bereits vorhandenen zu verschmelzen, um so eine umfangreichere Beschreibung des Objekts zu erhalten. Da der Erfolg stark von der Richtigkeit der Transformationsparameter abhängt, muß die Übereinstimmung der gemeinsamen Punkte sichergestellt werden. Nur bei ausreichender Sicherheit kann eine Verschmelzung der Punkte durchgeführt werden. Die Überprüfung der Punktkorrespondenzen kann z.B. mit dem "Iterative Closest Point Verification"-Verfahren ([4]) vorgenommen werden.

4.1.3 Gesichtserkennung

Eine Anforderung speziell an humanoide Roboter ist die Interaktion mit Menschen. Dazu gehört neben sprachlicher Kommunikation auch Blickkontakt.

Wird ein prototypisches Gesichtsmodell als nicht zu hoch aufgelöste Punktwolke in die Datenbank eingefügt, können wie oben beschrieben auch Gesichter in der Szene gefunden werden. Aus den Translationsparametern lassen sich leicht Koordinaten für die Ansteuerung des Kopfes ermitteln, um Blickkontakt herzustellen. Da zudem auch die Rotation bestimmt wird, kann auch erkannt werden, ob der Blickkontakt erwidert wird.

Derartige Erkenntnisse lassen sich hervorragend mit einer Spracherkennung kombinieren, indem man eine Übereinstimmung zwischen der Richtung des akustischen Signals und einem zum Roboter gerichteten Gesicht sucht. Zum einen kann damit die richtungsabhängige Vorfilterung der Spracherkennung erweitert werden, es sind aber auch Rückschlüsse über den Gesprächspartner des Gegenübers möglich, da nicht alle Sprachbefehle für den Roboter bestimmt sind. Besteht kein Blickkontakt, kann ein eventuell erkannter Sprachbefehl ignoriert werden.

Die Bildausschnitte mit dem erkannten Gesicht können auch anderweitig weiterverarbeitet werden, z.B. von einer Personenidentifizierungs-Software oder einem Mimikerkenner.

4.2 Geschwindigkeitssteigerung

Der Hauptnachteil des Verfahrens ist die lange Zykluszeit. Eine Integration in den Regelkreis der Kinematik zur Positionsverifikation und Kollisionsvermeidung ist momentan unmöglich, da hierfür eine Auswertung in Sekundenbruchteilen notwendig wäre. Demgegenüber ist die Berechnungsdauer von bis zu zehn Sekunden viel zu lang. Für die in voriger Sektion beschriebenen Anwendungen ist diese Zeit tragbar, eine schnellere Ausführung wäre dennoch wünschenswert.

4.2.1 Parallelisierbarkeit

Ist der Einsatz eines Multiprozessorsystems oder eines Clusters sinnvoll? In der aktuellen Implementierung bringt der Einsatz eines Parallelsystems keinen Vorteil, es bleibt aber die Frage, ob eine Parallelisierung des Verfahrens möglich ist.

Da das gesamte System modular aufgebaut ist, könnten verschiedene Stufen der Verarbeitungspipeline auf mehrere Prozessoren verteilt werden. Die einzelnen Stufen sind in Subsystemen zusammengefasst, zu deren Ausführung ein eigener Thread gestartet wird. Da in diesem Fall die Berechungen auf mehreren Prozessoren gleichzeitig vorgenommen wird, kann tatsächlich der Datendurchsatz gesteigert und die Zykluszeit reduziert werden. Zur Objektverfolgung kann ein solcher Ansatz deutliche Vorteile bringen, da die bewegungsbedingten Unterschiede aufeinanderfolgender Bilder kleiner werden. Auch eine Ausführung auf einem Cluster von mehreren Rechnern ist denkbar. Es muss darauf geachtet werden, dass der Kommunikationsaufwand nicht zu groß wird. Trennt man beispielsweise die Stereobildauswertung von der 3D-Objekterkennung, müssen nur die extrahierten und gefilterten 3D-Punkte über die Netzverbindung geschickt werden, die wesentlich weniger speicherintensiv sind als voll aufgelöste Kamera- oder Tiefenbilder.

Allerdings wird die Latenzzeit,also die Verarbeitungszeit eines einzelnen Bildpaars, nicht verkürzt, Da die einzelnen Berechnungsschritte nicht schneller werden und ein Bildpaar die gesamte Pipeline durchlaufen muss, entsteht hier kein Vorteil. Im Falle eines Clusters kommen sogar noch Verluste durch die relativ langsame Kommunikation hinzu.

Am zeitaufwändigsten ist die Disparitätenschätzung der Stereobildauswertung. Hier kann aber mit vertretbarem Aufwand die Last auf mehrere Prozessoren verteilt werden. Da zur

Verarbeitung jeweils nur einander entsprechende Zeilen samt Umgebung aus den beiden Bildern benötigt werden, kann das Eingangsbildpaar in waagrechte Streifen zerlegt werden. Diese Streifen können unabhängig voneinander auf verschiedenen Prozessoren verarbeitet werden; das gesamte resultierende Tiefenbild kann aus den Einzelergebnissen problemlos zusammengesetzt werden. Da zu jedem Bildpunkt die Umgebung benötigt wird, muss bei der Aufteilung der Eingangsdaten eine entsprechende Überlappung der Bildstreifen berücksichtigt werden. Der Berechnungsaufwand erhöht sich insgesamt dadurch aber nicht. Auf einem Shared-Memory Multiprozessorsystem steht der gesamte Eingangsdatensatz jedem Prozessor ohne Overhead zur Verfügung, wodurch das Verfahren noch vereinfacht wird.

Eine parallele Stereobildverarbeitung hat also folgende Struktur:

Bildentzerrung und Berechnung der Gauß-Pyramiden: Stehen zwei Prozessoren zur Verfügung, kann die Berechnung der beiden Gauß-Pyramiden einfach verteilt werden. Sollen mehr als zwei Prozessoren verwendet werden, kann die Berechnung verschiedener Bildbereiche auf weitere Prozessoren verteilt werden.

Disparitätenschätzung: Jeder der zur Verfügung stehenden Prozessoren kann aufgrund der zur Verfügung stehenden Gauß-Pyramiden einen horizontalen Streifen des Bildes bearbeiten. Der Berechnungsaufwand verteilt sich gleichmäßig auf die Prozessoren. Dabei kann von einem linearen Speedup ausgegangen werden. Da die Schreib- und Lesebereiche der Prozessoren im Eingangsdatensatz und im Ergebnisbild disjunkt sind, können alle Prozessoren ohne Kollisionen auf den gleichen Daten arbeiten. Da mehrere Prozessoren insgesamt über mehr Cache-Speicher verfügen, kann sogar superlinearer Speedup erreicht werden.

3D-Punktextraktion und Punktfilterung: Eine Parallelisierung ist hier schwierig, dieser Teil ist aber auch nicht so zeitkritisch wie die Disparitätenschätzung.

Berechnung und Vergleich der SpinImages: Die Berechnung eines SpinImages ist völlig unabhängig von anderen Berechnungsprozessen und muss für jeden orientierten Punkt durchgeführt werden. Vergleiche von SpinImages sind ebenfalls eigenständige Aufgaben. Somit kann die Last problemlos auf mehrere Prozessoren verteilt werden.

Parallelisierung ist also eine effiziente Möglichkeit zur Verbesserung der Latenzzeit und des Datendurchsatzes. Dem erhöhten Platz- und Stromverbrauch eines Multiprozessorsystems kann aber auf vielen Systemen nicht genügt werden. Denkbar wäre eine spezialisierte Hardwarelösung für die Disparitätenschätzung.

4.2.2 Beschleunigung durch Anpassung an die Szene

Die Ausführungsgeschwindigkeit hängt quadratisch von der Anzahl der Pixel im zu verarbeitenden Bildausschnitt ab. Durch Reduzierung der Bildgröße kann eine enorme Beschleunigung der Verarbeitung erreicht werden. Auf Kosten der Präzision kann man die

Auflösung der Kamerabilder reduzieren. Für weiter entfernte Objekte stehen dann aber unter Umständen nicht mehr genügend 3D-Punkte für eine zuverlässige Erkennung zur Verfügung.

Da üblicherweise nur wenige Objekte im Arbeitsraum des Roboters von Interesse sind und diese nur einen kleinen Teil der Kamerabilder ausfüllen, kann die Bildgröße drastisch reduziert werden, ohne an Auflösung einzubüßen. Wurde einmal eine komplette Szenenanalyse durchgeführt, kann zur Verfolgung eines einzelnen Objekts ein Bildbereich bestimmt werden, in dem es mit großer Wahrscheinlichkeit zu finden ist. Die Bewegung des Objekts wird dabei mit einer Kalman-Filterung berücksichtigt.

Um die initiale Szenenanalyse zu verkürzen, kann hier mit verringerter Auflösung gearbeitet werden. Interessierende Objekte in der Szene können, falls nötig, anschließend mit entsprechend höherer Auflösung nochmals abgetastet werden.

Wird ein bereits erkanntes Objekt im Raum verfolgt, kann die Suche nach Punktkorrespondenzen auf den Datensatz des Objekts beschränkt werden. Daraus ergibt sich eine weitere Zeitersparnis.

4.3 Weiterführende Arbeiten

Das erstellte Bildverarbeitungssystem ist eine Grundlage für eine Vielzahl von Arbeiten im am humanoiden Roboter ARMAR. Durch die Möglichkeit der Umwelterkennung erschließen sich neue Aspekte auf dem Weg zum autonomen Agieren und zur Interaktion mit Menschen.

4.3.1 Integration in das Robotersystem

Um die Ergebnisse aus der Bildverarbeitung mit der Ansteuerung des Roboters zu verknüpfen, muss eine geeignete Schnittstelle geschaffen werden. Auf der Steuerungsseite steht das modulare MCA2-System bereit. Das Bildverarbeitungssystem basiert auf der Java-Umgebung *“FilterGUI”* aus Kapitel 2. Der einfachste Weg für eine Verbindung ist ein Filtermodul für die FilterGUI, das intern eine TCP/IP-Verbindung zu einer MCA2-Schnittstelle herstellt. Entweder wird das MCA2-Protokoll für Modulgruppen implementiert, oder auf MCA2-Seite wird eine entsprechende Gegenstelle entworfen, die sich als MCA2-Modul einbinden lässt.

Die Aufgabenplanung und Trajektoriengenerierung wird sinnvollerweise in der *“FilterGUI”* entwickelt, da hier alle Ergebnisse aus der Bildverarbeitung vorliegen und Kanäle für komplexe Datentypen zur Verfügung stehen. Die berechneten Koordinaten für Arme, Kopf und Plattform des Roboters können dann problemlos über die Schnittstelle an MCA2 übergeben und dort ausgeführt werden. Ebenso können Sensordaten oder Statusinformationen über die Schnittstelle zurückgegeben werden.

Es ist geplant, zwei PC-Systeme auf ARMAR zu installieren: ein PC-104-System für die Steuerung der Microcontroller und die Berechnung der Kinematik, und einen schnellen PC ausschließlich zur Bildverarbeitung. Die PCs werden mit 100Mbit Ethernet verbunden, der PC-104 erhält zusätzlich eine WaveLAN-Funknetzkarte zur externen Anbindung und übernimmt eine Gateway-Funktion für den anderen PC.

4.3.2 Sensorfusion von Bildverarbeitung und Laserscanner

In der Plattform des Systems befindet sich zur Navigation und Kollisionsvermeidung ein 2D-Laserscanner der Firma Sick. Hindernisse wie Tische oder Bürostühle können damit nicht komplett erkannt werden, da in der Abtastung des Scanners nur die Tischbeine oder die Mittelsäule des Stuhls sichtbar sind. Um den Umfang des Objekts zu erfassen und bei der Kollisionsvermeidung berücksichtigen zu können, sollten potentielle Hindernisse mit Hilfe der Bildverarbeitung überprüft und klassifiziert werden.

Literaturverzeichnis

[1] M. Gazzaniga (Ed.). *The Cognitive Neurosciences*. MIT Press, Cambridge, 1995.

[2] B. K. P. Horn. Closed form solution of absolute orientation using unit quaternions. *J. Optical Soc. Amer.*, 4(4):629–642, 1987.

[3] Bernd Jähne. *Digitale Bildverarbeitung, 4. Auflage*. Springer Verlag, 1997.

[4] Andrew E. Johnson and Martial Herbert. Recognizing objects by matching oriented points. Technical Report CMU-RI-TR-96-04, Robotics Institute Carnegie Mellon Unitversity, 1996.

[5] Andreas Koschan, Volker Rodehorst, and Kathrin Spiller. Color stereo vision using hierarchical block matching and active color illumination. In *Proc. 13th. Int. Conf. on Pattern Recognition ICPR'96, Vol. 1*, pages 835–839, 1996.

[6] H.-H. Nagel. *Bildfolgenauswertung, Materialien zur Vorlesung*. IAKS Universität Karlsruhe, 2001.

[7] H.-H. Nagel. *Kognitive Systeme, Materialien zur Vorlesung*. IAKS Universität Karlsruhe, 2001.