COMROS

Basis-Dokumentation

Universität Stuttgart, IPVR
Lehrstuhl Praktische Informatik - Bildverstehen, Prof. Levi

Zusammengestellt von Bräunl
Mit Beiträgen von Bayer • Gerl • Mamier • Muscholl
Rausch • Sommerau • Vogt • Will

Vorwort

COMROS steht für Cooperative Mobile Robot Systems Stuttgart. Wir beschäftigen uns mit der Entwicklung autonomer Systeme auf der Basis der mobilen Roboter „Robuter“ von Robosoft, Bayonne, Frankreich.

Der vorliegende Text ist eine Zusammenstellung der Basis-Routinen für die Robotersteuerung und Bildverarbeitung von vier verschiedenen Rechnersystemen aus: IBM-PC Pentium unter Linux, Sun SPARCstation, Sun mit ELTEC-VectEx VME-Subsystem und MasPar MP-1216 (massiv parallel).

Die einzelnen Kapitel dokumentieren die Implementierungen von Basisoperationen und sollen neuen Studenten und Mitarbeitern den Einstieg in die Robotersteuerung an unserem Lehrstuhl erleichtern.

Als aktuelles Robotik-Informationssystem für Mitarbeiter und Studenten dient das www-basierte RIS, das unter folgender Adresse erreicht werden kann:

http://vasarely/roboter/ris/ris.html

bzw. als File:

file://localhost/usr/local/bv/robot/ris/ris.html

Dort finden sich aktuelle Hinweise über Veranstaltungen, Probleme und Lösungen, abgeschlossene, bzw. aktuelle Studien- und Diplomarbeiten, zu vergebende Themen, usw. Auch auf die Protokolle der Robotik-Teilgruppen (Architektur, Bildverstehen, Ultraschall, Neuro und Wartung) kann hier zugegriffen werden.

August 1995

Paul Levi
Thomas Bräunl

Allgemeines

Abb. 0.1 zeigt den Basisaufbau des Robotiklabors. Bilder und Daten werden über getrennte Funkstrecken übertragen. Bilddaten werden über eine analoge uni-direktionale Videofunkstrecke gesendet, während die Datenverbindung digital bi-direktional arbeitet.

Die derzeitigen Motorola Prozessor-Boards der Fahrzeuge erlauben nur eine RS-232 Kommunikation mit 9.600 Baud. Trotzdem wurden bereits die bestehenden RS-232 Funkmodems durch Funk-Ethernet ersetzt, da ein gleichzeitiger Betrieb mehrerer Modempaare auf Grund von gegenseitiger Störungen nicht möglich war. Die Umsetzung zwischen Ethernet und RS-232 wird derzeit von speziellen Umsetzern (*Comserver*) auf dem Fahrzeug übernommen, die Prozessorkarten sollen jedoch in absehbarer Zeit auf neuere Modelle mit Ethernet-Interface aufgerüstet werden.

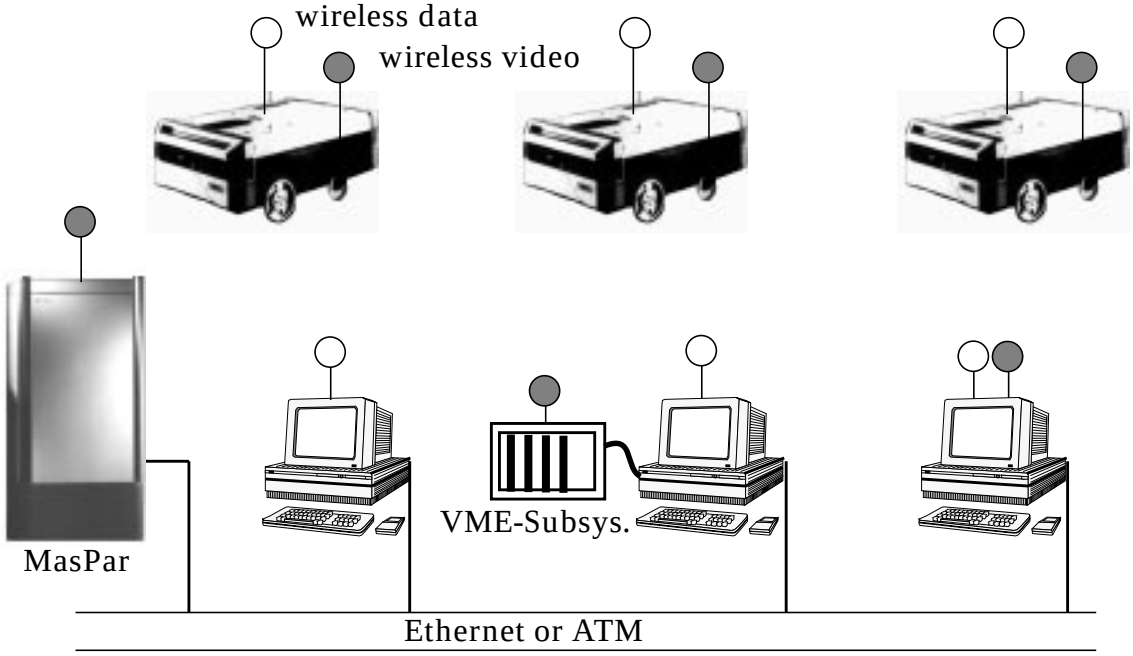


Abbildung 0.1: Hardware-Konfiguration

Eine weitere Option, die derzeit realisiert wird, ist die Integration eines Pentium-PCs unter Betriebssystem Linux direkt auf dem Fahrzeug, um so zwar mit geringerer Rechenleistung, jedoch vollkommen autonom agieren zu können.

Von M. Vogt wurde ein einfaches Interface unter Verwendung des Tools FORMS erstellt, daß eine „Fernsteuerung“ von mobilem Roboter, bzw. Greifarm / Stereokopf ermöglicht. Dieses Tool heißt `xremroc`, als Abkürzung für x-remote-robot-control.

Eine erste Version findet sich unter

`/usr/local/bv/robot/bin/SUNMP/XRRC0-1/xremroc`

Beim Aufruf ist als einziger Parameter einer der drei Roboternamen anzugeben. Dabei wird die Datei `/usr/local/bv/robot/etc/system.RoIrc` gelesen, die die Zuordnung zwischen Fahrzeugen, Comservern und ttys definiert. Sollte im eigenen HOME Verzeichnis eine Datei `.RoIrc` existieren, so werden diese Daten von dort gelesen.

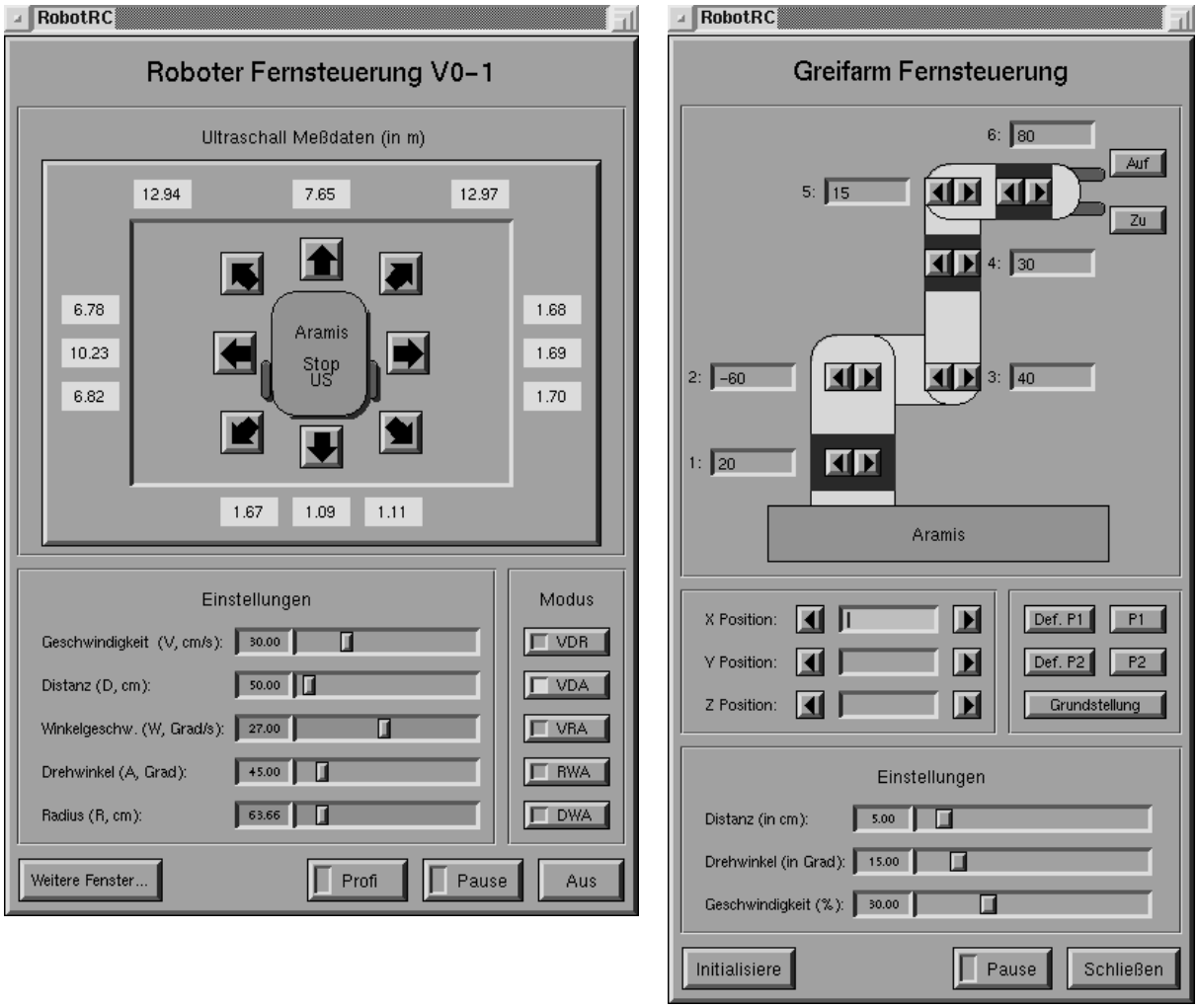


Abbildung 0.2: xremroc

Ein remote-Aufruf außerhalb der Workstation-Konsolen der Roboter (*robosun1-3*) ist möglich, jedoch ist hierbei aus Sicherheitsgründen die Steuerung eines Fahrzeugs unterbunden. Für die Fahrt stehen acht Pfeiltasten zur Verfügung, die wahlweise gerade, gebogen oder auf der Stelle gedrehte Bewegungen ausführen. Die Schieber im unteren Bereich des Fensters definieren die Bewegung genauer.

Beim Greifarm lassen sich neben der direkten Ansteuerung der Achsen auch bestimmte Armpositionen speichern und wieder abrufen. Da keine Kollisionsvermeidung stattfindet, muß hier besonders vorsichtig agiert werden! Die Umrechnung von kartesischen Koordinaten und die Bedienung der Greifhand sind derzeit noch nicht implementiert.

Horst Stolz entwickelte in seiner Diplomarbeit unter Leitung von Thomas Bräunl das Roboter-Simulationssystem MOBS (Abb. 0.3). Das System ermöglicht die Erstellung einer 3D-Umgebung und die gleichzeitige Simulation beliebig vieler Roboter. Simuliert werden die Grundbefehle der „Robuter“, wobei die gleichen ASCII-Steuersequenzen übertragen werden. Roboter-Steuerungsprogramme können ohne erneute Übersetzung sowohl für die Steuerung realer Roboter wir auch für die Steuerung eines Roboters in der Simulation eingesetzt werden. Simuliert werden die Ultraschall-

sensoren, Odometrie sowie das Kamerabild aus dem Blickwinkel eines Roboters (über die Inventor-Bibliothek der SGI).

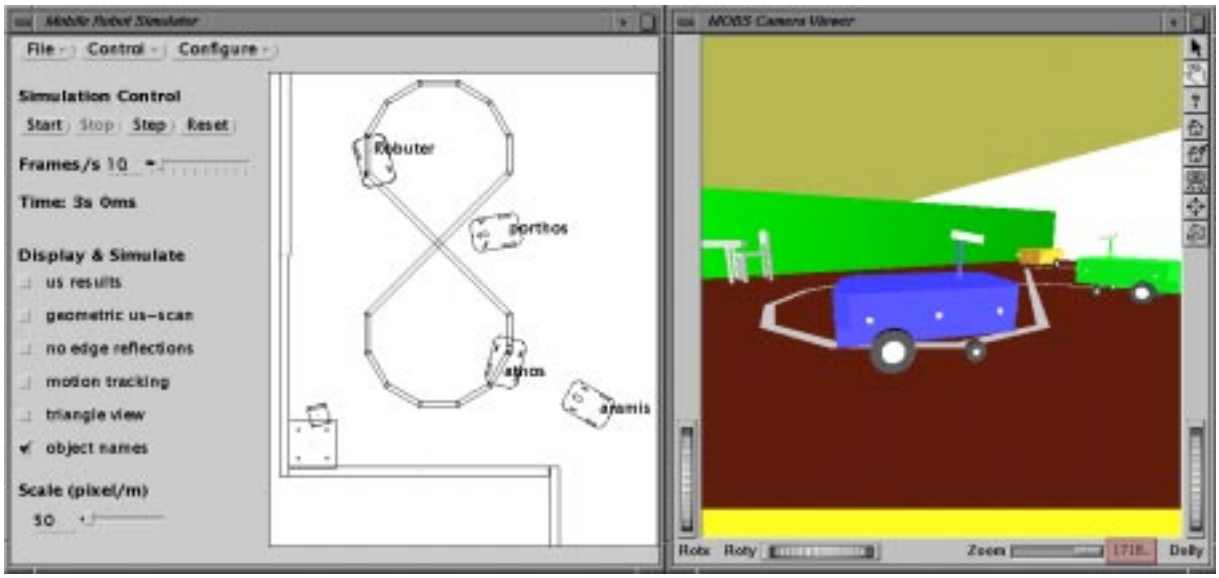


Abbildung 0.3: Roboter-Simulator

Inhaltsverzeichnis

Kapitel 1

Projektverwaltung

Matthias Muscholl, Marco Sommerau

Wichtig für die Entstehung des Roboterprojektes ist es, daß jeder Softwareentwickler nicht nur einheitliche Schnittstellen innerhalb seiner Software anbietet, sondern daß bei der Softwareentwicklung einheitliche Strukturen verwendet werden. Dieses Kapitel handelt von der Strukturierung der Verzeichnisse des Roboterprojektes, der Verzeichnisse der Teilprojekte, in denen die Software für einzelne Module entwickelt wird, sowie von den Projektverwaltungsdateien, die einen möglichst einheitlichen, hoffentlich einfachen und fehlerfreien Zugriff auf Teile des Projektes ermöglichen – also: was wann wo steht, oder wie man was wo ablegt[1].

Das Roboterprojekt wird in Teilprojekte gegliedert, die jeweils eine mindestens drei Zeichen lange Abkürzung erhalten. Im folgenden wird die Abkürzung TPR als Stellvertreter für beliebige Teilprojektabkürzungen verwendet. (TPR ist selbst ein generisches Teilprojekt, das die hier beschriebene Struktur definiert.)

Ziele für die entworfene Strukturierung waren:

1. Programme entwickeln zu können, die auf verschiedenen Rechnersysteme ausführbar sind. Es soll ein und derselbe Code auf unterschiedlichen Architekturen und Betriebssystemen übersetzbar sein, wobei architekturabhängiger Code durch bedingte Compile-Anweisungen getrennt wird. Die Architekturen die unterstützt werden sind Sun4, Sun4 Solaris, Sun4 Solaris Multiprozessor, HP, Ultrix (Maspar), SGI und Linux (in Kürze auch Maspar mit OSF/1).
2. Da es sich bei dem Roboterprojekt um ein sehr dynamisches, weitestgehend im Experimentierstadium befindliches Projekt handelt, ist die Verwendung eines Versionsverwaltungssystems unumgänglich. Nach Ablauf verschiedener Studien zu

1. Die Projektverwaltung wurde aufbauend auf einer Verzeichnisstruktur entwickelt, die in der Abteilung Verteilte Systeme (VS) des IPVR verwendet wird.

Systemen wie SCCS, RCS, CVS entschieden wir uns für CVS. Einen ersten Überblick bietet hierzu das Kapitel ‘Verwendung von CVS’ von Michael Vogt.

3. Eine weitere Anforderung bestand darin, einen einheitlichen Umgang mit dem Projekt und seinen Teilen anzubieten, um eine möglichst komfortable Einbindung der eigenen Code-Dateien, und die sichere Verwendung von Libraries anderer Teilprojekte zu garantieren.
   Teilprojekte geben nach Entwicklungsfortschritt Libraries und Programme zur Verwendung in anderen Teilprojekten frei. Meist geht die Entwicklung jedoch weiter, so daß gleichzeitig mehrere Versionen eines Teilprojekts verwendet werden.
   Das Versionsverwaltungssystem bietet nun die Basis, um jedem Nutzer diejenige Version zugänglich zu machen, auf die er aufbaut. Zu einer veröffentlichten Version gehören Include-Dateien und Manpages, sowie Libraries und Programme. Die Include-Dateien und Manpages werden in der benötigten Version lokal für den nutzenden Entwickler aus dem CVS-Repository extrahiert. Versionen die nicht mehr benötigt werden, kann der Nutzer aus dem lokalen Verzeichnis löschen.
   Die Libraries und Programme werden nicht im CVS-Repository gehalten, sondern beim Installieren im Veröffentlichungsbereich des Projektes dauerhaft abgelegt.
   Welche Version extrahiert und hinzugebunden wird, kann der Entwickler durch Angabe in einer einzigen Makefile-Variablen definieren. Global definierte Makefile-Targets ermöglichen dann das automatische Extrahieren von angegebenen Versionen.

Der Abschnitt "1.2 Die Verzeichnisstruktur der privaten Sicht auf ein Teilprojekt" auf Seite 18 beschreibt die Verzeichnisstruktur in der jeder Entwickler seine Programme schreibt.

1.1 Die Verzeichnisstruktur des Projektes

Unter `/usr/local/bv` befindet sich das Verzeichnis `robot` (Abb. 1.1):

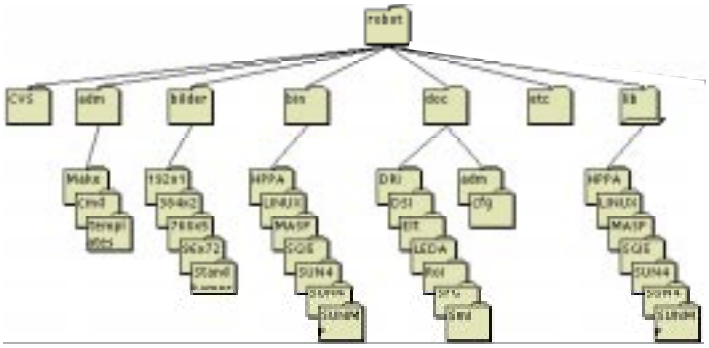


Abbildung 1.1: Das HOME-Verzeichnis des Robot-Projektes.

Hier befindet sich das Home-Verzeichnis des Roboterprojektes. In den Dateien „`LABORORDNUNG`“ und „`PROBLEME`“ finden sich aktuelle Informationen über die Organisation des Forschungsbetriebes.

Die in den Teilprojekten entwickelte Software wird im Sourcecode in einem Repository der Versionsverwaltung CVS gespeichert. Vom Software-Entwickler getestete und freigegebene Libraries oder Programme werden in Installations-Verzeichnissen anderen Teilprojekten zur Verfügung gestellt.

Wir unterscheiden zwei Bereiche: den Verwaltungsbereich (`CVS`, `adm`) und den Veröffentlichungsbereich (`bin`, `doc`, `lib`).

1.1.1 Der Verwaltungsbereich

In diesem Bereich sind Dateien abgelegt, die für eine einheitliche Administration des Projektes notwendig sind. Auf diese Dateien wird fast ausschließlich automatisiert zugegriffen.

Auf den `CVS`-Unterbaum wird mit Kommandos des Versionsverwaltungssystems gelesen oder geschrieben. Es darf auf ihn nicht unter Zuhilfenahme normaler Unix-Kommandos zugegriffen werden! Jegliches Verändern hat Auswirkungen über das eigene Teilprojekt hinaus.

Im `adm`-Verzeichnis befinden sich neben den Verzeichnissen `MakeSupport`, `cmd` und `templates` folgende Dateien (Abb. 1.2):

*Abbildung 1.2: Administrationsverzeichnis (**adm**)*

1. **`AbbreviationList`** enthält zu jedem Teilprojekt eine Zeile, in der die Teilprojekt-Abkürzung, die Teilprojektbezeichnung, die Namen des Entwicklungsteams sowie der Name des Teamleiters (meist der Betreuer) eingetragen ist. Diese Daten werden beim Erzeugen eines Teilprojektes durch das Shellscript CreateNewTPR automatisch auf dem laufenden gehalten. Es wird sichergestellt, daß keine zwei Teilprojekte die gleiche Abkürzung erhalten.
2. Aus Kompatibilitätsgründen zu früheren Versionen der Verzeichnisstruktur befindet sich im Verzeichnis `robot/adm` noch eine Datei **`cshrc`**, die bisher von privaten `.cshrc` jedes Entwicklers eingebunden wurde. Environment-Variablen werden nun vom abteilungsweiten `/usr/local/bv/rc/cshrc` gesetzt, wenn im privaten `.cshrc` „`set ROB_USER`“ und „`set PVM_USER`“ angegeben wurde. Folgende Variablen sind dadurch gesetzt:
   a. `CVSROOT` der Pfad, unter dem das Repository des Versionsverwaltungssystem gespeichert ist,
   b. `BV_ARCH` die Architektur des Systems auf dem man sich augenblicklich befindet,
   c. `PROJ_TOPDIR` das Home-Verzeichnis des Roboter-Projektes,
   d. und für `PROJECTABBREVIATION` steht CoMRoS,

e. was für `PROJECTNAME='Cooperative Mobile Robot Systems Stuttgart'` steht.

Weiterhin existieren symbolischen Links, die aus Kompatibilitätsgründen auf globale Makefiles verweisen.:

1. `Makefile.Sniff.BasedOnProjects` auf `MakeSupport/Global.Toplevel.Targets.make`
2. `Makefile.Sniff.install` und
3. `Makefile.install` auf `MakeSupport/Global.SrcLevel.Targets.make`

Im Unterverzeichnis **`adm/MakeSupport`** befinden sich globale Makefiles. Hier werden Targets definiert, die eine einheitliche und sichere Integration der einzelnen Teilprojekte ermöglichen (Abb. 1.3):

*Abbildung 1.3: Dateien mit globalen Makefile-Regeln (**adm/MakeSupport**).*

1. **`Global.Makefile.Admin.make`** enthält die Konfigurationsinformation für die folgenden, globalen Makefiles.
2. **`Global.SrcLevel.Targets.make`** enthält die projekteinheitlichen Targets für das automatische Generieren von Source-Dateien (`template`), zur Berechnung der Abhängigkeiten (`depend`) der Source-Dateien, für das Installieren fertiger Softwareversionen (`install`) und das Generieren von Manpages (`man`). Die Datei wird von den teilprojektbezogenen Makefiles der `TPR/src` Verzeichnisse aufgerufen.
3. **`Global.Toplevel.Targets.make`** enthält die Projekt-einheitlichen Targets für das Bereitstellen von Versionen, auf denen Teilprojekte basieren (`get`, siehe Abschnitt "1.2.1 Generische Makefiles" auf Seite 19). Ferner besitzt es Targets für `all` und `install`, die in die Makefiles der `TPR/src` Verzeichnisse verzweigen. Die Datei wird von Makefiles der Teilprojektebene eingebunden.

Im Unterverzeichnis **`adm/templates`** befinden sich folgende Schablonen. Sie werden bei der Generierung von Source-Dateien durch das in der Datei `adm/MakeSupport/Global.SrcLevel.Targets.make` definierten Target `template` verwendet (Abb. 1.4):

*Abbildung 1.4: Templates zur Source-Dateien Generierung (**adm/templates**).*

1. **`Abbreviationlist.empty`** enthält die Schablone für eine leere Abbreviationlist.
2. **`TPR.get.sh`** enthält die Schablone für ein Shell-Skript das automatisch abgearbeitet wird, wenn dieses Teilprojekt als Teil eines übergeordneten Teilprojekts ausgecheckt wird (verwendet bei Target: `get`).
3. **`template.h`** enthält die Schablone für Header von C Sourcen.
4. **`template.c`** enthält die Schablone für C Sourcen.
5. **`template.c.main`** enthält die Schablone für ein `main()` in C.
6. **`template.H`** enthält die Schablone für Header von C++ Sourcen.
7. **`template.C`** enthält die Schablone für C++ Sourcen.
8. **`template.C.main`** enthält die Schablone für ein `main()` in C++.

Die global definierten Targets rufen Shell-Skripte auf, die teilweise interaktiv bedient werden. Sie sind im Unterverzeichnis **`adm/cmd/make*`** abgelegt. Dort befinden sich weiterhin folgende Shell-Skripte (Abb. 1.5):

Abbildung 1.5: Die bereitgestellten Skripte

1. **`CreateNewTPR`** (Shell-Skript) erzeugt im aktuellen `PROJ_DEVELOPDIR` ein neues Teilprojekt. Die Variable `PROJ_DEVELOPDIR` sollte im privaten `.cshrc` gesetzt sein(z.B. `setenv PROJ_DEVELOPDIR $HOME/SA`). Zur Erzeugung eines neuen Teilprojekts werden Informationen wie die Teilprojektbezeichnung, die Teilprojekt-Abkürzung, die Namen des Entwicklungsteams sowie der Name des Teamleiters (meist der Betreuer) interaktiv abgefragt. Dieses Skript stellt daraufhin ein neues, bereits konfiguriertes Teilprojekt zur Verfügung (siehe Abschnitt "1.3.2 Erzeugen eines neuen Teilprojektes" auf Seite 24).
2. **`UpdateTPR`** (Shell-Skript) dient dazu, Teilprojekte die auf einer älteren Version des generischen Teilprojekts `TPR` basieren, an die neue Struktur anzupassen. Dazu wird interaktiv die Abkürzung des anzupassenden Teilprojekts abgefragt um die automatisierbaren Änderungen wie Hinzufügen von Verzeichnissen und Dateien vornehmen zu können (siehe Abschnitt "1.3.7 Aktualisieren eines Teilprojekts" auf Seite 30).
3. **`ChmodTPR`** (Shell-Skript) erlaubt die Festlegung der Zugriffsberechtigung von Teilprojekten im CVS Repository. Die Teilprojektbezeichnung und die gewünschte Berechtigung werden interaktiv abgefragt (siehe Abschnitt "1.4 CVS-Repositories für weitere Projekte" auf Seite 34).

4. **ChangeMakefileVariable** (Shell-Skript) ermöglicht es gezielt Makefilevariablen zu ändern, ohne daß der Aufruf des Editors nötig ist. Die Syntax ist:
   `ChangeMakefileVariable filename [variable value] ...`
5. **check** (Shell-Skript) überprüft, ob die für einen Übersetzungslauf notwendigen Include-Dateien des Makefiles `TPR/src/Makefile` aktualisiert sind und erzeugt diese neu, falls sie fehlen, oder veraltet sind (Aufruf aus dem Makefile heraus).
6. **configure.OSE.scripts** (Shell-Skript) ändert die Shell-Skripte des Tools `classinfo` des C++ Programmpakets `OSE` für die Verwendung in `/usr/local/bv/cmd` ab.
7. **make.install** (Shell-Skript) installiert die Libraries und Programme der aktuelle Version eines Teilprojekts im Veröffentlichungsbereich (Target: `install`).
8. **make.deinstall** (Shell-Skript) entfernt die Libraries und Programme der aktuelle Version eines Teilprojekts aus dem Veröffentlichungsbereich.
9. **make.depend** (Shell-Skript) erzeugt eine Abhängigkeitsliste aller `.c` und `.C`-Dateien, für Lex- und Yacc-Dateien sowie SNNS-Netzbeschreibungsdateien. Es werden alle durch ein `#include` eingebundene Dateien aufgeführt, wobei Systemincludes ausgenommen sind. Die Unterscheidung ist wie folgt: `"TPR/userInclude.h"` bzw. `<systemInclude.h>`. Das Makefile wird selbst mit in die Abhängigkeitsliste aufgenommen (Target: `depend`).
10. **make.dependency** (Shell-Skript) und **make.Sniff.dependency** (Shell-Skript) rufen jeweils `make.depend` mit erweiterter Parameterleiste auf und sind nur aus Kompatibilitätsgründen noch vorhanden.
11. **make.template** (Shell-Skript) erzeugt aus den generischen Templates eine neue Source-Datei, dessen Dateikopf bereits teilprojektspezifisch konfiguriert ist. Dazu wird interaktiv der Name der zu generierenden Datei abgefragt. Für `*.[cC]` Dateien kann angegeben werden, ob der Rumpf einer `main()` Funktion ebenfalls generiert werden soll und für `*.[hH]` Dateien wird abgefragt, ob es sich um einen Schnittstellen-Header handelt, und somit im `TPR/include/TPR`-Verzeichnis angelegt werden soll (Target: `template`).
12. **make.get.modifyProj.awk** (awk-Skript) paßt die in SNiFF+ Projektbeschreibungsdateien vorhandenen Pfade an versionsabhängigen Pfade von Unter-Teilprojekten an (verwendet bei Target: `get`).
13. **make.get.subprojects** (Shell-Skript) durchläuft rekursiv alle Unter-Teilprojekte, erstellt dabei eine Liste der Unter-Teilprojekte und exportiert die Include-Dateien und Manpages der im `PROJ_DEVELOPDIR` fehlenden Unter-Teilprojekte (verwendet bei Target: `get`).

1.1.2 Der Veröffentlichungsbereich

In diesem Bereich befinden sich die Verzeichnisse für die Ablage von fertiggestellten Versionen (Abb. 1.6). Hierhinein werden voll ausgetestete, stabile Libraries und Programme abgelegt. Von vorhandenen Dateien kann man ausgehen, daß sie während der Laufzeit des Gesamtprojektes bestehen bleiben.

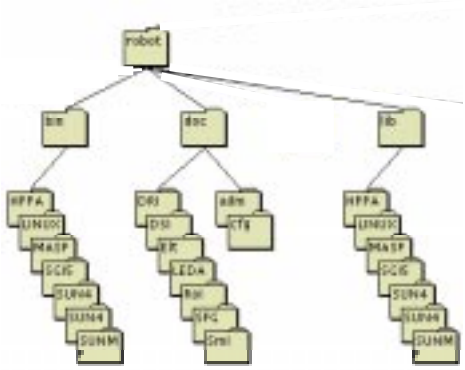


Abbildung 1.6: Der Veröffentlichungsbereich

1. **doc**: Zu jedem Teilprojekt wird eine Dokumentation (Usermanual und / oder Ausarbeitung der Arbeit) im Unterverzeichnis `$(PROJECTTAG)` abgelegt. Der Name muß das Präfix `$(PROJECTTAG).$(PROJECTVERSION)` besitzen und sollte mit dem Suffix die Dateiart (`.ps` oder `.dvi` oder `.fm`) spezifizieren. Grundsätzlich sollte darauf geachtet werden, daß ein Dateiformat gewählt wird, das möglichst platzsparend ist, aber ohne zusätzliche uncompress-Verfahren gelesen werden kann.
2. **bin**: Lauffähige Programme sollten ein ausführliches Usage angeben, wenn man sie mit `a.out -h` aufruft.
3. **lib**: Libraries sind vorübersetzte Code-Files und bilden eine Einheit mit ihren include-Dateien, die Prototypen für Funktionen und Datentypen spezifizieren. Die include-Dateien sind in der Versionsverwaltung gespeichert. Jedes Teilprojekt, das auf andere aufbaut holt sich automatisch die von ihm benötigten Versionen der Library-spezifischen include-Dateien aus dem Repository.

Aufgrund der unterschiedlichen Betriebssysteme und Rechnerarchitekturen wird in den Unterverzeichnissen `bin` und `lib` eine Strukturierung in architekturspezifische Unterverzeichnisse vorgenommen, hier `HPPA`, `LINUX`, `MASPAR` (Ultrix), `SGI5`, `SUN4`, `SUN4SOL2` und `SUNMP`. Das zu einem Rechner gehörige Architekturkürzel wird automatisch beim Öffnen einer Shell gesetzt.

1.1.3 Sonstige Verzeichnisse

1. **bilder**:
   Es wurde unter `/usr/local/bv/robot` ein Verzeichnis `bilder` angelegt, in das alle möglichen Roboterbilder gelegt werden können (und sollen), um somit für alle einen einfachen Zugriff auf Bilder von unseren Fahrzeugen zu ermöglichen. Das Unterverzeichnis habe ich entsprechend den möglichen Bildgröße des SFG [Sun Frame Grabber] unterteilt. Eine weitere Unterteilung dieses Verzeichnisses erfolgt dann nach Format. Bisher sind `ppm` und `tif` vorgesehen.

   Als Basis habe ich meine Bilder zur Verfügung gestellt. Sie sind von der Größe 192x144 im `ppm`-Format und stehen folglich unter `/usr/local/bv/robot/bilder/192x144/ppm/`.

2. **`etc`**:
   Im Verzeichnis `etc` sind Konfigurationsfiles abgelegt, die für den Betrieb der Roboterfahrzeuge erforderlich sind (`system.*`). Ferner finden sich in diesem Verzeichnis Beispiele für persönliche Konfigurationsfiles, die für den Einsatz von PVM gebraucht werden [`pvm_hosts`, `rhosts`].

1.2 Die Verzeichnisstruktur der privaten Sicht auf ein Teilprojekt

Wie bereits beschrieben, werden die Codedateien durch das Versionsverwaltungssystem CVS gespeichert. Die Benutzung sieht wie folgt aus: Sobald ein Teilprojekt begonnen wird, richtet der Betreuer die zugehörige Verzeichnisstruktur ein. Sie wird im Home-Verzeichnis des Softwareentwicklers angelegt und entsprechend dem Teilprojekt konfiguriert. Gleichzeitig wird diese Struktur im CVS-Repository eingetragen. Der Softwareentwickler kann nun entweder das Versionsverwaltungssystem nutzen und einzelne Abschnitte seiner Arbeit in das Repository speichern, oder bis zum Ende seiner Arbeit seine Programme entwickeln und erst zum Schluß den Source-Code im Repository ablegen. CVS ermöglicht es, daß mehrere Entwickler gleichzeitig an den selben Dateien editieren, und ihre Änderungen über das Repository austauschen. Genaueres siehe "1.5 Richtlinien für Multi-EntwicklerInnen-Teilprojekte" auf Seite 38.

Ein Teilprojekt besteht aus mehreren Unterverzeichnissen, deren Aufgabe darin besteht, verschiedene Typen von Dateien aufzunehmen (Abb. 1.7):

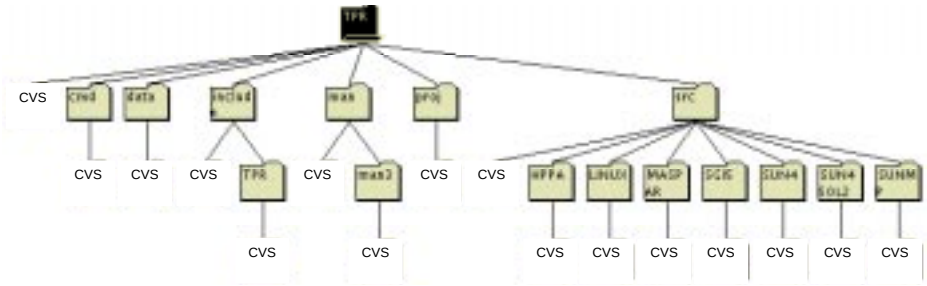


Abbildung 1.7: Die private Sicht auf ein Teilprojekt

1. **`src`**: Innerhalb dieses Verzeichnisses werden die Code-Dateien entwickelt. C-Dateien haben die Endung `.c` und `.h`, C++-Dateien die Endung `.C` und `.H`.
   Zu jeder Architektur / Betriebssystem existieren Unterverzeichnisse. Dorthinein werden alle erzeugten oder architekturabhängigen Dateien abgelegt. Die erzeugten Dateien sind `.o`-Dateien, die Datei `.Make.dependencies.rej` mit den Makefile Abhängigkeiten, alle ausführbaren Programme und Libraries.
   Die Datei `CONFIG.make` enthält schließlich die Definition der architekturabhängigen Makefile Variablen
2. **`include`**: Innerhalb dieses Verzeichnis-Baumes (im Unterverzeichnis `TPR`) werden diejenigen include-Dateien abgelegt, die andere Teilprojekte zur Benutzung von Teilprojekt-Libraries benötigen. Benutzt der Entwickler SNiFF+, so werden im

include-Verzeichnis selbst Projektbeschreibungsdateien abgelegt (siehe Erklärung zu dem Verzeichnis `proj`).

3. **`man`**: Zu den in Libraries zur Verfügung gestellten Funktionen, Variablen sowie C++ Klassen sind Beschreibungen in Form von Manpages zu erzeugen. Dazu werden Hilfsmittel bereitgestellt, die aus formatgerechten Kommentaren in den include-Dateien entsprechende Manpages generieren, und sie im `man3`-Unterverzeichnis ablegen (Target: `man`).
4. **`proj`**: Dieses Verzeichnis dient zur Aufnahme von SNiFF+ Projektbeschreibungsdateien, die das gesamte Teilprojekt umfassen. Für diejenigen die SNiFF+ verwenden wurde folgende Konvention entworfen: die Codefiles einer zur Verfügung gestellten Library werden in zwei Projektbeschreibungsdateien zusammengefaßt: Einerseits die include-Dateien die die sichtbare Schnittstelle der Library darstellen, andererseits alle restlichen Code-Dateien, die die Library implementieren. Dieses zweite SNiFF+ Projekt enthält dann das SNiFF+ Projekt der include-Files als Unterprojekt (Tabelle 1.1):

Namen der SNiFF+ Projekte / Unterprojekte	Verzeichnis	Sourcen
`libTPR.proj`	`TPR/proj`	`TPR/src/*.[hHcC]`
`libTPR.Interface.proj`	`TPR/include`	`TPR/include/TPR/*.[hH]`

Tabelle 1.1: Namenskonvention für SNiFF+ Projektbeschreibungsdateien.

Ein Teilprojekt kann prinzipiell aus beliebig vielen SNiFF+ Projekten bestehen.

5. **`cmd`**: Shell, awk, perl, ... -Skripte, für die Verwaltung oder Aufrufe im Teilprojekt.
6. **`data`**: Testszenarien und andere Dateien, die in keines der anderen Verzeichnisse gehören (z.B. `pvm_hostfile`).

1.2.1 Generische Makefiles

Aufgrund der unterschiedlichen Funktionalitäten von make, die von den einzelnen Betriebssystemen bereitgestellt werden, wird im Roboterprojekt einheitlich Gnumake verwendet (Aufruf: `gmake`).

Wir unterscheiden drei Abstraktionsebenen in einem Teilprojekt, denen entsprechende Makefiles zugeordnet sind: die Teilprojektebene (`TPR`), die Ebene der Code-Dateien (`TPR/src`) und die architekturabhängige Ebene (`TPR/src/$(BV_ARCH)`). In der Teilprojektebene befindet sich ein Makefile, daß die teilprojektspezifischen Variablen definiert (`Makefile.project.part.defines`). Dieses wird von allen anderen Makefiles eingebunden.
In der Architekturebene befindet sich das Konfigurationsfile, in dem architekturabhängige Variablen gesetzt werden (`CONFIG.make`), sowie die erzeugten Abhängigkeitsbeschreibungen (`.Make.dependencies.rej`).
Auf der Ebene der Code-Dateien ist das Makefile abgelegt, in dem die *Targets* für das Erzeugen der Libraries und der Programme vom Entwickler angegeben werden und

stellt das sog. Arbeits-Makefile dar. Es bindet ein globales Makefile ein, in dem *Targets* projekteinheitlich spezifiziert sind. Ein Makefile, daß der Entwickler kaum verwenden wird, welches aber für die spätere Weiterverwendbarkeit wichtig ist, befindet sich auf der Teilprojektebene. Es ermöglicht ohne Kenntnisse der internen Projektstruktur Programmversionen nachträglich zu übersetzen und zu installieren.

Im folgenden werden die einzelnen Teile detailliert aufgeführt:

1. **`TPR/Makefile.project.part.defines`**: Diese Datei wird von allen Makefiles eingebunden und enthält folgende änderbaren teilprojektspezifischen Variablen:

   a. `BASESON`: Spezifiziert all jene Teilprojekte, auf die das Teilprojekt aufbaut, und die aus dem gleichen CVS-Repository stammen. Der Inhalt dieser Variablen wird verwendet um include-Pfade und Linkpfade zu bestimmen. Teilprojekte werden in Form von `<project-tag><project-version>` angegeben (z.B: `BASESON = TPR Sub2-1`). Der Inhalt dieser Variablen dient dem automatischen Extrahieren der verwendeten Teilprojektversionen aus dem CVS-Repository. Der zu den Unter-Teilprojekten gehörende Schnittstellenbereich `TPRx-y/include/TPR` und `TPRx-y/man` wird in die Ebene des `PROJ_DEVELOPDIR` exportiert. (Sobald sie nicht mehr benötigt werden können sie z. B. mit `/bin/rm -rf SUB3-0` gelöscht werden).
   Falls einige der in `BASESON` angegebenen Teilprojekte nur intern zur Erzeugung von lauffähigen Programmen und nicht zur allgemeinen Verwendung der Teilprojekt-Libraries notwendig sind, sollten diese mit `[...]` geklammert werden. Dadurch wird das unnötige Exportieren der geklammerten Teilprojekte bei der Verwendung dieses Teilprojekts vermieden (z.B: `BASESON = TPR Sub2-1 [TST1-5 SUB3-0]`).

   b. `ADDITIONAL_BASESON`: Mit Hilfe dieser Variablen können auch Teilprojekte aus anderen CVS-Repositories (die für andere Projekte eingerichtet wurden) von diesem Teilprojekt verwendet werden. Solche Teilprojekte können hier mit
   ```
   <CVS repository>:<lib directory>:\
   <project tag><project version>
   ```
   angegeben werden (z.B.: `ADDITIONAL_BASESON = /otherproject/CVS:/otherproject/lib:OTPR1-1`). Pfade, Schnittstellenbereiche und die Klammerung intern benötigter Teilprojekte werden wie die von BASESON behandelt.

   c. `PROJECTVERSION`: Enthält die Versionsnummer, an der gerade gearbeitet wird. Anstelle des üblichen Trennsymbols '`.`' wird '`-`' verwendet! (z.B. `1-0`, `2-4-1-2`)

   Die folgenden Variablen werden in dieser Datei definiert, sollen aber vom Entwickler nicht geändert werden:

   d. `PROJECTTAG`: Mindestens drei Zeichen lange Abkürzung des Projektes. Sie ist eindeutig innerhalb des gesamten Roboterprojektes und wird als Verzeichnisname, als Modulname (CVS) und als erster Teil von Versions-Tags verwendet. Sie wird beim Erzeugen eines Teilprojektes angegeben und ist ab dann fest (siehe dort).
   Zukünftige Teilprojekte müssen alle global sichtbaren Bezeichner (globale Va-

riablen, Typdefinitionen, Klassendefinitionen, Prozedurnamen, ...) mit dem Präfix „<PROJECTTAG>_“ versehen, um Probleme beim Linken zu vermeiden.

e. `ADM_VERSION`: Gibt die Version der hier beschriebenen Administrationsumgebung an mit der dieses Teilprojekt erstellt wurde.

f. `INSTDIR`: Ist identisch mit `PROJ_TOPDIR`.

g. `INSTLIBDIR`: Gibt den architekturabhängigen Pfad zur Installation der erzeugten Libraries an.

h. `INSTBINDIR`: Gibt den architekturabhängigen Pfad zur Installation der erzeugten Programme an.

i. `MAKE`: Gibt das für Make zu verwendende Kommando an (hier `gmake`).

j. `PVM_TOPDIR`: Der Pfad der das `include`- und `lib`-Verzeichnis von PVM enthält.

k. `TPR_DEVELOPDIR`: Home-Verzeichnis dieses Teilprojekts, wobei sich der Pfad aus `PROJ_DEVELOPDIR` und `TPR` zusammensetzt.

2. **`TPR/Makefile`**: Dieses Makefile verzweigt in das `src`-Verzeichnis und ruft dort wieder Make auf.
Die Philosophie dahinter ist die, daß man sich eine private Kopie eines Teilprojektes geben lassen kann, `gmake` ***aufruft und es werden die Libraries, Programs, etc. erzeugt, ohne daß man genaueres über das Teilprojekt und dessen Struktur wissen muß***.
Für teilprojektübergreifende *Targets* wird ein globales Makefile eingebunden.

3. **`TPR/src/Makefile`**: Dies ist das eigentliche Arbeitsmakefile. Wie schon vorher genannt, werden architekturabhängigen Konfigurationsdateien eingebunden. Die Abhängigkeiten der `.o`-Files von `.c`- und `.h`-Dateien werden durch den Aufruf von `gmake depend` erzeugt und in architekturspezifische Dependency-Files gespeichert. *Es empfielt sich nach Änderung von include-Abhängigkeiten die Dependency-Files neu zu erzeugen.* Sie werden in das Arbeitsmakefile eingebunden und bewirken ein Compilieren derjenigen Sourcen, die geändert wurden, bzw. von solchen abhängen.
Folgende Eintragungen sind jedoch vom Entwickler selbst vorzunehmen:

   a. Regeln zum Linken compilierter Codefiles zu ausführbaren Programmen

   b. Regeln zum Erstellen von static oder shared Libraries

   c. Eintragen aller zur Veröffentlichung gedachten Programme in `PROGRAMS`

d. Eintragen aller zur Veröffentlichung gedachten statischen Libraries in `STATICLIBRARIES` (vgl. Tabelle 1.2)

Teilprojekt enthält	
nur eine Library	mehrere Libraries
libTPR.a	libTPR_<name1>.a libTPR_<name2>.a ...

Tabelle 1.2: Namenskonventionen für erstellte Libraries.

e. Eintragen aller zur Veröffentlichung gedachten shared Libraries in `DYNAMICLIBRARIES` (vgl. Tabelle 1.2) Wer shared Libraries erzeugen möchte, sollte die Hinweise in `$PROJ_TOPDIR/doc/adm/Shared.Libraries.txt` lesen.

4. **`TPR/src/$(BV_ARCH)/CONFIG.make`**: Architekturabhängige Konfiguration von Pfaden und Variablen.
   Weiterhin können hier folgende Eintragungen vom Entwickler selbst vorgenommen werden:
   a. Eintragen aller zur Veröffentlichung gedachten architekturspezifischen Programme in `ARCH_PROGRAMS`
   b. Eintragen aller nicht zur Veröffentlichung gedachten architekturspezifischen Testprogramme in `ARCH_TESTS`
   c. Eintragen aller zur Veröffentlichung gedachten architekturspezifischen statischen Libraries in `ARCH_STATICLIBS`
   d. Eintragen aller zur Veröffentlichung gedachten architekturspezifischen shared Libraries in `ARCH_DYNAMICLIBS`

1.3 Die Projektverwaltung in der Praxis

1.3.1 Einrichten der Umgebung

Um die vorhandene Projektverwaltung nutzen zu können und ein komfortables Arbeiten zu ermöglichen sind für jeden Entwickler ein Reihe von Eintragungen in den Dateien `~/.cshrc` bzw. `~/.emacs` erforderlich.

Damit die Umgebungsvariablen und Pfade korrekt gesetzt werden, müssen in der Datei `~/.cshrc` des Entwicklers mindestens die in Prog. 1.1, 4-10 aufgeführten Schalter gesetzt werden. Außerdem ist es dringend erforderlich, die Variablen `PROJ_DEVELOPDIR` zu setzen (Prog. 1.1, 12). Um die Änderungen in der aktuellen Shell wirksam zu machen sollte nach dem Abspeichern ein `source ~/.cshrc` ausgeführt werden.

Programm 1.1: Eintragungen in der Datei ~/.cshrc des Entwicklers.

```
      [...]
   1  #
   2  # uncomment to use  this software
   3  #
      [...]
   4  set PVM_USER
   5  set BV_USER
      [...]
   6  #       DO NOT DELETE NEXT LINE
   7  source /usr/local/rc/cshrc
      [...]
   8  set ROB_USER
   9  set SNIFF201
  10  source /usr/local/bv/rc/cshrc
  11
  12  setenv PROJ_DEVELOPDIR  $HOME/SA
      [...]
```

Für das komfortablere Arbeiten mit CVS ist es empfehlenswert das emacs Erweiterungspaket pcl-cvs zu verwenden. Dazu müssen in der Datei `~/.emacs` des Entwicklers die in Prog. 1.2, 1-6 aufgeführten Zeilen eingefügt werden. Damit die Änderungen wirksam werden, sollte ein eventuell laufender emacs verlassen und neu gestartet werden. Das Starten des Programmpakets erfolgt dann durch die Eingabe von `M-x cvs-update` im laufenden emacs. Eine ausführliche Beschreibung der Funktionsweise dieses Programmpakets ist über das emacs Menü Help, Untermenü Info unter dem Punkt Pcl-cvs zu bekommen

Wer bei der Verwendung von SNiFF+ statt des eingebauten Editors weiterhin emacs verwendet werden möchte, kann dies durch den Eintrag der in Prog. 1.2, 8ff aufgeführten Zeilen konfigurieren. Die genaue Anbindung kann in der Dokumentation zu SNiFF+ unter `$SNIFF_DIR/doc/UsersRefPart1.ps.gz`, nachgelesen werden..

Programm 1.2: Eintragungen in der Datei ~/.emacs des Entwicklers.

```
      [...]
   1  ;;; pcl-cvs-startup.el,v 1.2 1992/04/07 20:49:17 berliner Exp
   2  (autoload ‚cvs-update „pcl-cvs“
   3            „Run a ‚cvs update‘ in the current working directory.
Feed the
   4  output to a *cvs* buffer and run cvs-mode on it.
   5  If optional prefix argument LOCAL is non-nil, ‚cvs update -l‘ is
run.“
   6            t)
   7
   8  ; --- SNiFF+ ---
   9  (load-library „$SNIFF_DIR/config/sniff-mode“)
      [...]
```

1.3.2 Erzeugen eines neuen Teilprojektes

Das Erzeugen eines neuen Teilprojekts erfolgt weitgehend automatisiert. Der Entwickler, oder der Betreuer führt in einer Shell des Entwicklers das Kommando `CreateNewTPR` aus. Zu beachten ist, daß man sich in dem Verzeichnis befindet, unter dem die Softwareentwicklung stattfinden soll (also `$PROJ_DEVELOPDIR`). Dort hinein wird dann der Verzeichnisunterbaum angelegt, der das Teilprojekt enthalten wird.

Da alle Entwickler im Roboter-Projekt der Unix-Gruppe `bvrobot` angehören, sind die Schreibrechte auf das CVS-Repository auf diese Gruppe beschränkt. Falls die Gruppe nicht aktiv ist, muß `newgrp bvrobot` aufgerufen werden.

Ein korrektes Protokoll sieht dann folgendermaßen aus:

```
robosun3:[SA] >/usr/local/bv/robot/adm/cmd/CreateNewTPR

This script prompts you for project part name, abbreviation, developers
and project part leader that is required by the organization of the
CoMRoS-Project: Cooperative Mobile Robot Systems Stuttgart.


Current settings of the necessary environment:
        PROJ_TOPDIR     /usr/local/bv/robot
        CVSROOT         /usr/local/bv/robot/CVS
        PROJ_DEVELOPDIR /home/<account>/SA

Current working directory:
        /home/<account>/SA
Do you want to continue [yes]? yes

These are the existing project parts:
```

man erhält nun eine Liste der schon vergebenen Namen, hier ein Ausschnitt:

```
#Abbr.  Name            students                        leader

TPR     Teilprojekt Template                            mmuschol
RoI     RS232 Robot Interface   mmuschol                mmuschol
Elt     Eltec Interface sommerau                sommerau
SFG     Sun Frame Grabber Software      mamier          mamier
DRI     Distributed Robots Interface    rausch          rausch
Fea     Flaschen erkennen und ansteuern goerzisn                gerl
DSI     Distributed SpaceMouse Interface        jnkarau         rausch
MGI     Manipulator Gripper Interface   rausch          rausch
OpG     Optimales Greifen       filipph,gerl            gerl
MobS    Mobile Robot Simulator  stolz,braunl            braunl
NUM     Navigation mit Ultraschall (Modellbildung)      loethe          rausch
NNF     neural network object following clemengo                zell
CRE     Automatisches Einparken msoberdo                zell
DCI     Device Controller Interface     loethe          rausch
```

[...]

```
Please answer the following questions with care. The information
is needed to simplify the administration of the whole project.

Abbreviation of your project part (e.g. ‚Bsp‘) []? NEU
```

```
Full name of your project part (or ‚none‘) []? IHR NEUES PROJECT

Accounts of software developers (e.g. ‚roy,lee‘) []? ENTWICKLER

Account of project part leader (e.g. ‚braunl‘) []? BETREUER
```

nun werden eine Reihe von Operationen durchgeführt:

i. Die Abkürzung des Teilprojektnamens wird reserviert.
ii. Die generische Verzeichnisstruktur eines Teilprojekts wird angelegt.
iii. Die Dateien werden hinsichtlich dem neuen Teilprojekt konfiguriert.
iv. Das Teilprojekt wird im Repository angelegt.

1.3.3 Verwenden schon vorhandener Teilprojekte

Einer der wesentlichen Aspekte dieser Verwaltungsstruktur ist die Möglichkeit zur einfachen Einbindung schon vorhandener Teilprojekte in ein neues Teilprojekt. Dazu müssen in der Datei `TPR/Makefile.project.part.defines`, wie schon in "1.2.1 Generische Makefiles" auf Seite 19, 1.a) und b) beschrieben, die Makefile-Variablen `BASESON` (Prog. 1.3, 7) und `ADDITIONAL_BASESON` (Prog. 1.3, 18ff) entsprechend gesetzt werden.

Programm 1.3: Eintrag von Unter-Teilprojekten im TPR/Makefile.project.part.defines.

```
     [...]
  1  ####################################
  2  # names and versions of other projects
  3  # on which this one bases on
  4  # e.g. TPR1-0
  5  # TPR only is needed by default
  6  ####################################
  7  BASESON       = TPR SUBTPR2-1 [SUB1-0]
  8
  9  ####################################
 10  # names and Versions of other projects from other CVS repositories
 11  # and projects on which this one bases on
 12  # syntax:
 13  # <CVS repository>:<lib directory>:<project tag><project version>
 14  # e.g. /otherproject/CVS:/otherproject/lib:OTPR1-1
 15  #
 16  # architecture substitution applies as usual
 17  ####################################
 18  ADDITIONAL_BASESON = \
 19  /home/mueller/CVS:/home/mueller/lib:MUELL1-5 \
 20  [/home/maier/CVS:/home/maier/lib:/MAI2-5]
     [...]
```

Nachdem alle notwendigen Unter-Teilprojekte angegeben und abgespeichert sind, wird automatisch beim nächsten Aufruf von `gmake` sichergestellt, daß im aktuellen `PROJ_DEVELOPDIR` die Includes und Manpages aller angegebener Unter-Teilprojekte vorhanden sind. Falls dies nicht der Fall sein sollte, werden diese aus dem jeweiligen

CVS-Repository exportiert. Damit bei der Verwendung des neuerstellten Teilprojekts durch andere Teilprojekte nur die für die Libraries notwendigen Unter-Teilprojekte exportiert werden, sollte von der Möglichkeit Gebrauch gemacht werden, Teilprojekte die nur zur Erstellung teilprojekt-interner Programme benötigt werden durch [] auszuklammern.

Teilprojekte werden rekursiv nach weiteren benötigten Teilprojekten durchsucht. Es sind also nur die Unter-Teilprojekte anzugeben, auf die direkt aufgebaut wird. In der Datei `TPR/src/.Make.subprojects.rej` werden alle ermittelten Abhängigkeiten abgelegt. Nach einer Änderung von `BASESON` oder `ADDITIONAL_BASESON` werden die Abhängigkeiten beim nächsten Aufruf von `gmake` automatisch neu ermittelt

1.3.4 Anpassen der generischen Makefiles

Um die eigenen Programme und Libraries in den generischen Makefiles einzubauen muß im Regelfall nur das Makefile `TPR/src/Makefile` editiert werden. Dort gibt es die Variablen `DYNAMICLIBRARIES`, `STATICLIBRARIES` und `PROGRAMS` (siehe auch "1.2.1 Generische Makefiles" auf Seite 19, 3.). Diese Variablen enthalten je ein Liste der zu diesem Teilprojekt gehörenden Softwarekomponenten. Für jede dort angegebene Komponente muß eine Regel existieren.

Um eine Library mit Namen `libTPR.a` zu erstellen muß also der Name (Namenskonventionen siehe Tabelle 1.2) in der Variablen `STATICLIBRARIES` angegeben werden (Prog. 1.4, 2), die zur Library gehörenden Objektdateien in der Variablen `LIBRARY1_OBJECTS` (Prog. 1.4, 5ff) aufgeführt und schließlich eine Regel für die Generierung formuliert werden (Prog. 1.4, 12ff).

Programm 1.4: Eintragungen im TPR/src/Makefile zur Erstellung einer Library.

```
      [...]
   1  STATICLIBRARIES  = $(ARCH_STATICLIBS) \
   2                     $(BV_ARCH)/libTPR.a
   3  ...
   4  # -- build libraries ------------------------------------------------
   5  LIBRARY1_OBJECTS = $(BV_ARCH)/complex.o \
   6                     $(BV_ARCH)/simple.o \
   7                     $(BV_ARCH)/array.o \
   8                     $(BV_ARCH)/scalar.o \
   9                     $(BV_ARCH)/calculate.o
  10
  11  # -- build a static library
  12  $(BV_ARCH)/libTPR.a:    $(LIBRARY1_OBJECTS)
  13          $(RM) $@
  14          $(STATIC_LIBRARY) $@ $(LIBRARY1_OBJECTS)
  15          $(RANLIB) $@
  16          @echo
      [...]
```

Ähnlich sehen die Eintragungen für ein Programm `test` aus, das die schon erwähnte Library benutzt. Dazu muß der Programmname in der Variablen `TESTS` (`PROGRAMS` falls es sich um ein zu installierendes Programm handelt) angegeben (Prog. 1.5, 2) und zwei Regeln zur Generierung des Programms formuliert werden (Prog. 1.5, 6ff),

wobei die erste der beiden nur zu Vereinfachung des gmake Aufrufs dient: statt `gmake $BV_ARCH/test` genügt ein `gmake test`.

Programm 1.5: Eintragungen im TPR/src/Makefile zur Erstellung eines Programms

```
     [...]
  1  TESTS            = $(ARCH_PROGRAMS) \
  2                     $(BV_ARCH)/test
  3
  4  # -- build programs ------------------------------------------------
  5  # -- build a single program
  6  test: $(BV_ARCH)/test
  7
  8  $(BV_ARCH)/test: $(BV_ARCH)/test.o $(STATICLIBRARIES)
  9          $(CCC) -o $@ $(LDEFINES) \
 10          $(BV_ARCH)/test.o \
 11          $(LIBPATH) -lTPR -lm $(LIBS)
 12          @echo
     [...]
```

Wie schon in Prog. 1.4, 1 und Prog. 1.5, 1 zu erkennen ist, gibt es noch die Möglichkeit Programme bzw. Libraries nur auf bestimmten Architekturen zu erstellen. Dazu müssen die Namen der jeweiligen Komponenten in den Variablen `ARCH_DYNAMICLIBS`, `ARCH_STATICLIBS`, `ARCH_TESTS` bzw. `ARCH_PROGRAMS` im `CONFIG.make` der gewünschten Architektur eingetragen werden. Die erforderlichen Regeln sind wie schon bei den auf allen Architekturen vorhandenen Komponenten im Makefile `TPR/src/Makefile` anzugeben.

1.3.5 Benutzen von SNiFF+

SNiFF+ ist ein Werkzeug, das für die Entwicklung von C++ Programmen eine komfortable Entwicklungsumgebung mit verschiedenen Browsern bereitstellt (Die Entwicklung von reinen C Programmen wird ebenfalls unterstützt).

Dieses Werkzeug verwendet ebenfalls den Begriff Projekt, jedoch in einem anderen Zusammenhang. Hier wird unter einem Projekt eine Menge von Dateien gesehen, die alle in demselben Verzeichnis liegen müssen. Diese Definition des Projekts ist also nicht mit dem seither verwendeten Begriff Teilprojekt vereinbar, da die Header der Schnittstellen zu vorhandenen Libraries nicht im Pfad `TPR/src`, sondern im Pfad `TPR/include/TPR` verwaltet werden. Die Lösung des Problems liegt darin, daß SNiFF+ Projekte ihrerseits wieder Projekte enthalten können und damit beliebig schachtelbar sind.

Die im Roboterprojekt verwendete Regelung ist wie schon in "1.2 Die Verzeichnisstruktur der privaten Sicht auf ein Teilprojekt" auf Seite 18, Punkt 4. erwähnt so, daß die Schnittstellen-Header einer Library aus dem Verzeichnis `TPR/include/TPR` in einem SNiFF+ Projekt zusammengefaßt werden und die Projektbeschreibungsdatei im Verzeichnis `TPR/include` abgelegt wird. Der Name dieser Datei ergibt sich aus den Namenskonventionen von Tabelle 1.1 und Tabelle 1.2. SNiFF+ wird durch den Aufruf `sniff` gestartet. Um ein Projekt für die Schnittstellen-Header anzulegen müssen folgende Schritte durchgeführt werden:

1. Anwahl Menüpunkt **Project** -> **New Project** im Hauptfenster.
2. Mit Dateiauswahlbox in das Verzeichnis **TPR/include/TPR** springen und den Button **Select** anklicken.
3. Eintragungen im Fenster **Attributes of a New Project**:
   a. `Project Directory` **TPR/include/TPR**
   b. `Project File Name` **libTPR.Interface**
   c. `Project File Extension` **proj**
   d. `Destination of Project File(s)` **..**
   e. `Project Type` **Relative Project**

   und **OK** anklicken.
4. Durch Anwahl des Menüpunkts **Project** -> **Add/Remove Files** im **Project Editor** kann nachträglich die Auswahl der zum Projekt gehörigen Dateien geändert werden (z.B. bei Unterteilung in mehrere Libraries).

Weiterhin werden die zu dieser Library gehörenden Implementierungsdateien aus dem Verzeichnis `TPR/src` zu einem zweiten SNiFF+ Projekt zusammengefaßt und im Verzeichnis `TPR/proj` abgelegt, wobei das zuvor erzeugte SNiFF+ Projekt der Schnittstelle der Library als Unterprojekt hinzugenommen wird:

1. Anwahl Menüpunkt **Project** -> **New Project** im Hauptfenster.
2. Mit Dateiauswahlbox in das Verzeichnis **TPR/src** springen und den Button **Select** anklicken.
3. Eintragungen im Fenster **Attributes of a New Project**:
   a. `Project Directory` **TPR/src**
   b. `Project File Name` **libTPR**
   c. `Project File Extension` **proj**
   d. `Destination of Project File(s)` **../proj**
   e. `Project Type` **Relative Project**

   und **OK** anklicken.
4. Die Auswahl der zum Projekt gehörigen Dateien kann wie schon beschrieben geändert werden.
5. Durch Anwahl des Menüpunkts **Project** -> **Add Subproject** im Project Editor kann das zugehörige Projekt **libTPR.Interface.shared** aus dem Verzeichnis **TPR/include** hinzugenommen werden.

Für alle Programme, wobei jedes einzelne wieder ein SNiFF+ Projekt sein sollte, muß nur das entsprechende Projekt der Library als Unterprojekt angeben werden damit alle notwendigen Sourcen im Browser verfügbar sind.

Falls für Programme im Teilprojekt weitere Teilprojekte benutzt werden und die Entwickler dieser Teilprojekte ebenfalls SNiFF+ verwendet haben, sind die Projektbeschreibungsdateien der Schnittstelle in den jeweiligen include-Pfaden vorhanden. Diese können also einfach vom eigenen SNiFF+ Projekt als Unterprojekt eingebunden werden.

Weitere Informationen zu SNiFF+ sind im Verzeichnis `$SNIFF_DIR/doc` zu finden.

1.3.6 Installieren von getesteter und stabiler Software

Das Installieren erfolgt mit Hilfe des Makefiles. Zu beachten ist, daß man bereits in `TPR/Makefile.project.part.defines` die Variable `PROJECTVERSION` aktualisiert hat. Es werden alle in den Variablen `PROGRAMS`, `STATICLIBRARIES` und `DYNAMICLIBRARIES` angegebenen Komponenten installiert.

Zur durchgängigen Unterstützung von SNiFF+ ist es notwendig, daß für jedes Teilprojekt zumindest eine SNiFF+ Projektbeschreibungsdatei für jede der darin vorhandenen Libraries existiert! (Kurzanleitung siehe "1.3.5 Benutzen von SNiFF+" auf Seite 27)

Man ruft in einer Shell der entsprechenden Architektur `gmake install` auf:

```
matisse:[src] >gmake install


you are now installing a new version of your software.
1. verify that the version number SmI1-1 is correct
   or change PROJECTVERSION in
   /home/mmuscholl/ROBO/inWork/SmI/Makefile.project.part.defines
2. continue or rerun make install

3. run cvs commit in /home/mmuscholl/ROBO/inWork/SmI
4. run cvs tag SmI1-1 in
   /home/mmuscholl/ROBO/inWork/SmI


Do you want to continue [no]: yes
shared libraries successfully installed

installing SUN4/libmouse.a in /usr/local/bv/robot/lib/SUN4/libSmI1-1mouse.a
static libraries successfully installed

installing SUN4/mousetest in /usr/local/bv/robot/bin/SUN4/SmI1-1mousetest
programs successfully installed

you have successfully installed a new version of your software.
Keep in mind that you have to add generated Files like Manpages
and copied includes into the CVS repository (see below).

Please do not forget to complete point 3 and 4 above, like:

cd /home/mmuscholl/ROBO/inWork/SmI/man/man3; cvs add *.3
cd /home/mmuscholl/ROBO/inWork/SmI; cvs commit; cvs tag SmI1-1
```

Die als letztes ausgedruckten Unix-Kommandos sind Ausgaben des Installskiptes. Mit Cut-and-Paste kann man die Kommandos direkt in der Shell auszuführen.

Nachdem alle angegebenen Kommandos ausgeführt sind ist die erstellte Software dieses Teilprojekts allgemein verfügbar und kann wie in "1.3.3 Verwenden schon vorhandener Teilprojekte" auf Seite 25 beschrieben exportiert und von anderen Teilprojekten verwendet werden.

Sollte man direkt nach dem Installieren doch noch einen Bug finden, so können mit `gmake deinstall` die veröffentlichten Dateien gelöscht werden. Nach dem Bug-fix und einem `gmake install` muß `cvs tag` abermals ausgeführt werden.

Falls zur Verwendung eines installierten Teilprojekts außer dem Exportieren zusätzliche Schritte erforderlich sind (z.B. Anlegen von Links), gibt es ab der Version adm2-2 des Verwaltungsbereichs die Möglichkeit im Verzeichnis `TPR/cmd` ein Shell-Skript `TPR.get.sh` abzulegen. Dieses Shell-Skript wird, falls vorhanden, nach dem Exportieren der gewünschten Teilprojekt-Version durch den Aufruf `gmake get` eines übergeordneten Teilprojekts automatisch ausgeführt. Da dieser Fall eher die Ausnahme sein wird, ist dieses Shell-Skript nicht als Default in jedem erstellten Teilprojekt vorhanden, sondern kann bei Bedarf hinzugefügt werden (Template: `adm/templates/TPR.get.sh`).

1.3.7 Aktualisieren eines Teilprojekts

Falls ein älteres Teilprojekt die im Lauf der Zeit erweiterte Funktionalität ebenfalls verwenden möchte, muß die Struktur des Teilprojekts aktualisiert werden. Ob es sich bei einem Teilprojekt um eine ältere Version handelt kann durch den Aufruf von `gmake version` im Verzeichnis `TPR/src` festgestellt werden (die aktuelle Version ist `adm2-2`). Die Aktualisierung des Teilprojekts startet man mit dem Aufruf des Kommandos `UpdateTPR`.

Ein korrektes Protokoll sieht dann folgendermaßen aus:

```
robosun3:[SA] >/usr/local/bv/robot/adm/cmd/UpdateTPR

This script prompts you for the project abbreviation of the
project to be updated.


Current settings of the necessary environment:
        PROJ_TOPDIR     /usr/local/bv/robot
        CVSROOT         /usr/local/bv/robot/CVS
        PROJ_DEVELOPDIR /home/<account>/SA

Current working directory:
        /home/<account>/SA
Do you want to continue [yes]? yes


These are the existing project parts:
```

man erhält nun eine Liste der schon vergebenen Namen, hier ein Ausschnitt:

```
#Abbr.  Name            students                        leader

TPR     Teilprojekt Template                            mmuschol
RoI     RS232 Robot Interface   mmuschol                mmuschol
Elt     Eltec Interface sommerau                sommerau
SFG     Sun Frame Grabber Software      mamier          mamier
DRI     Distributed Robots Interface    rausch          rausch
Fea     Flaschen erkennen und ansteuern goerzisn                gerl
DSI     Distributed SpaceMouse Interface        jnkarau         rausch
MGI     Manipulator Gripper Interface   rausch          rausch
OpG     Optimales Greifen       filipph,gerl            gerl
MobS    Mobile Robot Simulator  stolz,braunl            braunl
NUM     Navigation mit Ultraschall (Modellbildung)      loethe          rausch
```

```
NNF     neural network object following clemengo                zell
CRE     Automatisches Einparken msoberdo                zell
DCI     Device Controller Interface     loethe          rausch
```

[...]

```
Select the project abbreviation whose source tree has to
be updated.

Abbreviation of your project part (e.g. ‚Bsp‘) []? OLD

The selected project is:
        OLD = IHR ALTES PROJEKT (ENTWICKLER:BETREUER)

Do you want to continue [yes]? yes

Checking for status of OLD in /home/<account>/SA ...
        OLD exists and is not modified.


Now the new/changed files/directories of the generic project part
since ADM_VERSION adm1-1 will be exported to $HOME/SA:
```

man erhält nun eine Liste der seit adm1-1 geänderten Dateien (diese Liste variiert natürlich je nachdem wie alt das Teilprojekt ist):

```
U TPR/include/TPR/README
U TPR/Makefile
U TPR/Makefile.project.part.defines
```

[...]

```
Now the new/changed files/directories will be processed and
copied to OLD:

Processing OLD/include/OLD/README ...
Processing OLD/Makefile ...
Processing OLD/Makefile.project.part.defines ...
```

[...]

```
Now the new directories will be added to the repository:

Adding OLD/include/OLD/ ...
Add directory /home/sommerau/CVS/OLD/include/OLD to the repository (y/n) [n]
? Directory /home/sommerau/CVS/OLD/include/OLD added to the repository


Now the new files will be added to the repository:

Adding OLD/include/OLD/README ...
cvs add: scheduling file `README‘ for addition
cvs add: use ‚cvs commit‘ to add this file permanently
```

```
Some files are not needed by default any more
and may be removed from the repository:

OLD/src/HPPA/C.Make-Dependencies
OLD/src/HPPA/c.Make-Dependencies
```

[...]

```
OLD/src/OLDmain.c
OLD/src/OLDmain.h
```

ACHTUNG: unter Umständen werden die Dateien OLDmain.[ch] in diesem Teilprojekt verwendet!

```
(use ‚cvs add <file>‘ for accidentally removed files)


Do you want to remove them ALL from the repository [no]? yes
```

*bei Angabe von **no** wird für jede einzelne Datei gefragt, ob diese Datei gelöscht werden soll*

```
cvs remove: scheduling C.Make-Dependencies for removal
cvs remove: scheduling c.Make-Dependencies for removal
cvs remove: scheduling C.Make-Dependencies for removal
```

[...]

```
cvs remove: use ‚cvs commit‘ to remove these files permanently


Files modified:

OLD/Makefile
OLD/Makefile.project.part.defines
OLD/README
OLD/src/HPPA/CONFIG.make
```

[...]

```
ATTENTION: Do not commit the modified files! These are generic files
           which first have to be merged with the last checked in
           version!
```

Nach der Ausführung dieses Skripts sind die unter `Files modified:` aufgeführten Dateien durch ihr Pendant aus dem generischen Teilprojekt überschrieben. Aus diesem Grund müssen die Inhalte der jeweils betroffenen beiden Dateien zusammengeführt werden. Dies geschieht mit der Hilfe des SNiFF+ Werkzeugs DiffMerge, indem ein spezielles SNiFF+ Projekt erstellt wird, das alle Makefiles des Teilprojekts beinhaltet:

1. Anwahl Menüpunkt **Project** -> **New Project** im Hauptfenster.
2. Mit Dateiauswahlbox in das Verzeichnis **TPR** springen und den Button **Select** anklicken.
3. Eintragungen im Fenster `Attributes of a New Project`:
   a. `View` **General**, `Project Options`:

   i. `Project Directory` **TPR**
   ii. `Project File Name` **TPR**
   iii. `Project File Extension` **shared**
   iv. `Destination of Project File(s)` **.**
   v. `Project Type` **Shared Project**

b. `View` **General**, zusätzliche Selektionen bei `New Project Options`:
   `i. Generate Subproject Tree` **selektieren**
   `ii.Remove Empty Projects` **selektieren**

c. `View` **File Types**:
   Auswahl ausschließlich des File Types **Make** durch Doppelklicks.

und **OK** anklicken.

4. Im `Project Editor` über dem unteren Fensterteil mit der Überschrift `Projects` die rechte Maustaste drücken und den Menüpunkt **Select From All Projects** anwählen.

Im `Project Editor` kann nun für jede der angezeigten Dateien nacheinander der Menüpunkt **File** -> **Show Differences...** angewählt und der folgenden Requester bestätigt werden. Das nun erscheinende Fenster stellt, falls Unterschiede vorhanden sein sollten, auf der linken Seite den Inhalt der aktuellen Datei und auf der rechten Seite den Inhalt der Datei beim letzten Einchecken dar. Für das Zusammenfügen genügt es zu wissen welche Teile aus der zuletzt eingecheckten Version übernommen werden müssen (Fehlerhafte Übernahmen können durch den Menüpunkt **Edit** -> **Undo Merge Text** zurückgenommen werden):

1. Allgemein:
   $Revision$, $Date$, $Author$ und Log übernehmen.
2. `TPR/src/*/CONFIG.make`:
   Bis auf die in 1. erwähnten Punkte nichts übernehmen, außer eventuell vorhandenen Erweiterungen des Entwicklers im Bezug auf Compileroptionen und Include- bzw. Librarypfaden.
3. `TPR/Makefile`:
   Bis auf die in 1. erwähnten Punkte nichts übernehmen, außer eventuell vorhandenen Erweiterungen des Entwicklers.
4. `TPR/Makefile.project.part.defines`:
   a. `BASESON` und
   b. `PROJECTVERSION` übernehmen, bzw. gleich erhöhen.
5. `TPR/src/Makefile`:
   Hier gibt es die gravierendsten und unübersichtlichsten Änderungen, wobei das Werkzeug DiffMerge leider nur wenig Hilfe leisten kann. Es empfiehlt sich ein häufigeres Abspeichern, da dabei ein erneutes diff ausgeführt und die Darstellung erneuert wird. Zu beachten sind folgende Punkte:
   a. Jetzt enthält die Variable `CCC` den C++ Compiler (nicht mehr `CPP`!).
   b. Namenskonvention für Libraries siehe Tabelle 1.2.

c. Regeln der Form
`<program>: $(BV_ARCH)/<program>`
müssen entweder mit einem Semikolon abgeschlossen werden, oder in der nächsten Zeile einen Tabulator haben.

Falls bei der Arbeit mit SNiFF+ etwas nicht so funktioniert wie hier beschrieben, das Programm verlassen und nochmals starten, das verschafft meistens Abhilfe. Nachdem alle Dateien angepaßt sind, kann das SNiFF+ Projekt geschlossen werden. Anschließend können die im TPR Verzeichnis gelegenen `*.shared` Dateien gelöscht werden.

Der Platz für die Schnittstellen-Header der Libraries hat sich im Lauf der Zeit mehrfach geändert. Falls sie bei dem umzustellenden Teilprojekt noch nicht im Verzeichnis `TPR/include/TPR` liegen, können diese einfach mit `mv *.[hH] $PROJ_DEVELOPDIR/TPR/include/TPR` dorthin bewegt werden. Ein direkt anschließend ausgeführtes cvs-update im emacs meldet, daß die verschobenen Include-Dateien verschwunden sind und daher aktualisiert wurden (mit `Updated` markiert). Im Gegenzug sind die Include-Dateien in ihrem neuen Pfad natürlich unbekannt (mit `Unknown` markiert). Die mit `Updated` markierten Dateien können nun vom Repository gelöscht und die mit `Unknown` markierten Include-Dateien hinzugefügt werden (Genauere Beschreibung: im cvs-buffer den Menüpunkt Help -> Describe Mode anwählen).
Die Log-Messages im Header dieser verschobenen Dateien können gelöscht werden, da die Zählung wieder bei Version 1.1 beginnt. Außerdem sollte anschließend ein `gmake depend all` aufgerufen werden, da unter Umständen bei einer Compilation nicht mehr alle Includes gefunden werden und die Sourcen dementsprechend angeglichen werden müssen (statt `#include “header.h“` nun `#include “TPR/header.h“`).

War die Entwicklung des Teilprojekts bereits abgeschlossen, d.h. die aktuelle Version ist auch installiert, dann sollte die aktualisierte Version des Teilprojekts eingecheckt und mit einer neue Versionsnummer getagt werden.

1.4 CVS-Repositories für weitere Projekte

Für diejenigen, die die Notwendigkeit sehen, ihre Sourcen unter die Verwaltung von CVS zu stellen, sei an dieser Stelle eine kurze Anleitung zur Einrichtung der Umgebung gegeben.

Es bietet sich an, die für das Roboterprojekt erstellte Projektverwaltung auch für die Sourcen-Verwaltung eigener Projekte zu verwenden, da die Möglichkeit Teilprojekte aus verschiedenen CVS-Repositories zu mischen.

Empfehlenswert ist die Überlegung, ob für das neue Projekt eine eigene Gruppen-ID eingerichtet werden soll. Dadurch können die Zugriffsrechte, insbesondere die Schreibrechte besser kontrolliert werden.

1.4.1 Einrichten der Umgebung

Der erste Schritt ist das Anlegen eines privaten CVS-Repositories durch das Ausführen folgender Kommandos (falls keine eigene Gruppen-ID verwendet wird entfallen die Punkte 2. und 3.):

1. `mkdir <myProj>` (`$PROJ_TOPDIR` Verzeichnis des eigenen Projektes im `$HOME`)
2. `chgrp <ourGroup> <myProj>`
3. `chmod g+s <myProj>`
4. `unsetenv CVSROOT`
5. `cvsinit`

```
The CVSROOT environment variable is not set.
You should choose a location for your source repository
that can be shared by many developers.  It also helps to
place the source repository on a file system that has
plenty of free space.

Please enter the full path for your CVSROOT source repository:
/home/<account>/<myProj>/CVS
[...]
```

Der nächste Schritt beinhaltet das Umsetzen der Umgebungsvariablen `CVSROOT`, `PROJ_TOPDIR` und `PROJ_DEVELOPDIR`, die von der Projektverwaltung benötigt werden. Es hat sich bewährt, dieses Umsetzen in der Datei `~/.cshrc` vorzunehmen, indem die in Prog. 1.6 aufgeführten Zeilen hinzugefügt werden. Durch den Aufruf `newgrp bvrobot` werden dann automatisch die Variablen richtig gesetzt..

Programm 1.6: Zusätzliche Eintragungen in der Datei ~/.cshrc des Entwicklers.

```
   1  setenv MYGID `id | awk ,{i=index($2,“(„); l=length($2); print
substr($2,i+1,l-i-1)}‘`
   2  switch ($MYGID)
   3      case <ourGroup>:
   4           setenv PROJ_TOPDIR $HOME/<myProj>
   5           setenv PROJ_DEVELOPDIR  $HOME/MYGROUPWORK
   6           setenv CVSROOT $PROJ_TOPDIR/CVS
   7           breaksw
   8      case bvrobot:
   9           umask 002
  10           setenv PROJ_TOPDIR /usr/local/bv/robot
  11           setenv PROJ_DEVELOPDIR  $HOME/ROBOWORK
  12           setenv CVSROOT $PROJ_TOPDIR/CVS
  13           set prompt = „%M:[%.] ROBOT>“
  14           breaksw
  15  endsw
```

1.4.2 Einrichten des Verwaltungsbereichs

Für die Projektverwaltung muß im Verzeichnis `$PROJ_TOPDIR` unter anderem das Verzeichnis `adm` für den Verwaltungsbereich angelegt werden ("1.1.1 Der Verwaltungsbereich" auf Seite 13).

Die zum Verwaltungsbereich gehörenden Dateien und Verzeichnisse werden in einem eigenen Teilprojekt (`adm`) unter CVS verwaltet. Da sich dieser Bereich gelegentlich ändert, ist es von Vorteil einen symbolischen Link vom eigenen `adm`-Verzeichnis auf das `adm`-Verzeichnis des Roboterprojekts zu legen. Dadurch ist sichergestellt, daß immer die neueste und weitgehend getestete Version verwendet wird. Da jedoch im `adm`-Verzeichnis auch die Datei `Abbreviationlist` liegt, die sämtliche Module des eigenen CVS-Repositories enthalten soll, kann dieser Link nicht für die oberste Ebene des `adm`-Verzeichnisses angelegt werden.

Die einfachste Möglichkeit das Gewünschte zu erreichen besteht in der Ausführung der folgenden Kommandofolge:

```
cd $PROJ_DEVELOPDIR
mkdir adm
cd adm
ln -s /usr/local/bv/robot/adm/* .
```

Der ebenfalls entstandene Link `Abbreviationlist` muß durch eine Kopie der Datei `adm/templates/Abbreviationlist.empty` ersetzt werden. Dadurch ist das neue CVS-Repository auch für die Projektverwaltung völlig leer, wenn man von einigen reservierten Namen absieht.

Einer der reservierten Namen ist das generische Teilprojekt TPR. Um im eigenen CVS-Repository neue Teilprojekte anlegen zu können muß dieses Teilprojekt dort ebenfalls vorhanden sein. Dies wird durch das Anlegen eines symbolischen Links im Verzeichnis `$CVSROOT` auf das Verzeichnis `/usr/local/bv/robot/CVS/TPR` erreicht. Ein angenehmer Nebeneffekt dieses Links ist, daß ohne eigenes Zutun immer die aktuellste Version des generischen Teilprojekts verwendet wird.

1.4.3 Einrichten des Veröffentlichungsbereichs

Für die Projektverwaltung fehlen im Verzeichnis `$PROJ_TOPDIR` nur noch die für die Installation notwendigen Unterverzeichnisse ("1.1.2 Der Veröffentlichungsbereich" auf Seite 16). Diese werden durch das Ausführen der folgenden Kommandos angelegt:

```
mkdir bin lib
cd bin
mkdir HPPA LINUX MASPAR SGI5 SUN4 SUN4SOL2 SUNMP
cd ../lib
mkdir HPPA LINUX MASPAR SGI5 SUN4 SUN4SOL2 SUNMP
```

1.4.4 Zugriffsberechtigung

Die im neuen CVS-Repository verwalteten Sourcen sollen im Regelfall nicht von jedermann verwendet werden können. Aus diesem Grund ist es möglich die Zugriffs-

rechte für jedes einzelne Teilprojekt durch den Aufruf des Shell-Skripts ChmodTPR zu ändern.

Ein korrektes Protokoll sieht dann folgendermaßen aus:

```
robosun3:[SA] >/usr/local/bv/robot/adm/cmd/ChmodTPR

This script prompts you for the project abbreviation of the
project to change permissions.


Current settings of the necessary environment:
        PROJ_TOPDIR     /usr/local/bv/robot
        CVSROOT         /usr/local/bv/robot/CVS

Do you want to continue [yes]? yes


These are the existing project parts:
```

man erhält nun eine Liste der schon vergebenen Namen, hier ein Ausschnitt:

```
#Abbr.  Name            students                        leader

TPR     Teilprojekt Template                            mmuschol
RoI     RS232 Robot Interface   mmuschol                mmuschol
Elt     Eltec Interface sommerau                sommerau
SFG     Sun Frame Grabber Software      mamier          mamier
DRI     Distributed Robots Interface    rausch          rausch
Fea     Flaschen erkennen und ansteuern goerzisn                gerl
DSI     Distributed SpaceMouse Interface        jnkarau         rausch
MGI     Manipulator Gripper Interface   rausch          rausch
OpG     Optimales Greifen       filipph,gerl            gerl
MobS    Mobile Robot Simulator  stolz,braunl            braunl
NUM     Navigation mit Ultraschall (Modellbildung)     loethe          rausch
NNF     neural network object following clemengo                zell
CRE     Automatisches Einparken msoberdo                zell
DCI     Device Controller Interface     loethe          rausch
```

[...]

```
Select the project abbreviation which permissions of the source
tree have to be changed.

Abbreviation of your project part (e.g. ‚Bsp‘) []? CHG

The selected project is:
        CHG = IHR PROJEKT (ENTWICKLER:BETREUER)

Who else shall have permissions (()user, (g)roup, (o)thers&group) []? g



The permissions of project
        CHG = IHR PROJEKT
will be changed:
        chmod  ug=r  CHG
```

```
Do you want to continue [yes]? yes
```

[...]

1.5 Richtlinien für Multi-EntwicklerInnen-Teilprojekte

folgendes steht ebenso in /usr/local/bv/info/cvs.faq:

```
2D.2   If I work with multiple modules, should I check them all out and
       commit them occasionally?  Is it OK to leave modules checked out?

       The simple answers are "Yes."

       There is no reason to remove working directories, other than to
       save disk space.  As long as you have committed the files you
       choose to make public, your working directory is just like any
       other directory.

       CVS doesn't care whether you leave modules checked out or not.
       The advantage of leaving them checked out is that you can quickly
       visit them to make and commit changes.


 committing a file?  Is there a "cvs-mode" for Emacs?

       See Section 4F.1


4F.1   How do I use CVS under Emacs?  Is there an Emacs cvs-mode?

       The pcl-cvs package distributed with CVS 1.3 is an emacs package
       that helps with the update/commit process.  When you are ready to
       update, you use the 'cvs-update' command within emacs.  This
       executes "update" and fills a cvs-mode buffer with a line for each
       file that changed.  The most helpful features are: descriptive
       words for what happened (i.e. Merged or Conflict rather than 'U'),
       single keys bound to diffs and commits, and the ability to mark
       arbitrary groups of files, possibly from different directories,
       for commit as a whole.

       All the developers in my group that use emacs find pcl-cvs a much
       friendlier and more helpful way to update/commit than raw cvs.
       One vi user even converted to emacs just to use pcl-cvs.

                                Contributed by Jeffrey M Loomis


2D.7   How does conflict resolution work?  What *really* happens if two
       of us change the same file?

       While editing files, there is no conflict.  You are working on
       separate virtual branches of development contained in your working
       directories.  When one of you decides to commit the file, the
       other may not commit the same file until "update" has merged the
       two together.

       Say you both check out rev 1.2 of <file>.  Your coworker commits
       revision 1.3.  When you try to commit your file, CVS says:

               cvs commit: Up-to-date check failed for `<file>'

       You must merge your coworker's changes into your working file by
       typing:
```

```
        cvs update <file>
```

which will produce the output described in 2B.6.

After you resolve any overlaps caused by the merging process, you may then commit the file.

Yes, the first one who commits can cause the other some work.

Yes, between the time you execute "update" and "commit", someone else may have committed a later revision of <file>. You will have to execute "update" again to merge the new work before committing. Most organizations don't have this problem. If you do, you might consider splitting the file.

3O.4 So "tag" labels a bunch of files. What do you use a Tag for?

You use it to "checkout" the labeled collection of files as a single object, referring to it by name.

Anywhere a revision number can be used a Tag can be used. In fact tags are more useful because they draw a line through a collection of files, marking a development milestone.

The way to think about a Tag is as a curve drawn through a matrix of filename vs. revision number. Consider this:

Say we have 5 files (in some arbitrary modules, some may be in 2 or more modules by name, some may be in 2 or more modules because of the Repository tree structure) with the following revisions:

```
        file1   file2   file3   file4   file5
        1.1     1.1     1.1     1.1  /--1.1*      <-*-  <tag>
        1.2*-   1.2     1.2    -1.2*-
        1.3  \- 1.3*-   1.3   / 1.3
        1.4           \  1.4  /  1.4
                       \-1.5*-   1.5
                         1.6
```

At some time in the past, the '*' versions were tagged. Think of the <tag> as a handle attached to the curve drawn through the tagged revisions. When you pull on the handle, you get all the tagged revisions. Another way to look at it is that you draw a straight line through the set of revisions you care about and shuffle the other revisions accordingly. Like this:

```
        file1   file2   file3   file4   file5

                        1.1
                        1.2
                1.1     1.3                  _
        1.1     1.2     1.4     1.1           /
        1.2*----1.3*----1.5*----1.2*----1.1  (--- <-- Look here
        1.3             1.6     1.3           \_
        1.4                     1.4
                                1.5
```

I find that using these visual aids, it is much easier to understand what a <tag> is and what it is useful for.

4C.2 Why (or when) would I want to create a branch?

Remember that you can think of your working directory as a "branch for one". You can consider yourself to be on a branch all the time because you can work without interfering with others

until your project (big or small) is done.

The four major situations when should create a branch are when:

1. You expect to take a long enough time or make a large enough set of changes that the merging process will be difficult.

2. You want to be able to "commit" and "tag" your work repeatedly without affecting others.

   If you ever think you need Source Control for your own work, but don't want your changes to affect others, create a private branch. (Put your username in the branch tag, to make it obvious that it is private.)

3. *You need to share code among a group of developers, but not the whole development organization working on the files.*

   Rather than trying to share a working directory, you can move onto a branch and share your work with others by "committing" your work onto the branch. Developers not working on the branch won't see your work unless they switch to your branch or explicitly merge your branch into theirs.

4. You need to make minor changes to a released system.

   Normally a "release" is labeled by a branch tag, allowing later work on the released files. If the release is labeled by a non-branch tag, it is easy to add a branch tag to a previously tagged module with the "rtag" command. If the release is not tagged, you made a mistake. Recovery requires identifying all revisions involved in the release and adding a tag to them.

4C.3 How do I create and checkout a branch?
4C.4 Once created, how do I manage a branch?
4C.5 Are there any extra issues in managing multiple branches?
4C.6 How do I merge a whole branch back into the trunk?

Kapitel 2

Verwendung von CVS

Michael Vogt

Sämtliche Projekte, die sich mit der Steuerung der Roboterfahrzeuge befassen, sollen bezüglich der Software-Entwicklung durch CVS unterstützt werden. Hierdurch soll paralleles Arbeiten mehrerer Entwicklergruppen auf jeweils stabiler Software erreicht werden. Die nachfolgenden Abschnitte geben eine kurze Einführung in CVS und in die Verwendung von CVS in Roboterprojekten.

2.1 Was ist CVS

CVS steht für „Concurrent Versions System“. Es ist ein System zur Entwicklung und Verwaltung von Quelltexten aller Art (Programme, Dokumentation, usw.). CVS setzt auf dem System RCS (Revision Control System) auf, welches das GNU-Pendant zum bekannten SCCS (Source Code Control System) ist, das auf fast allen Unix-Plattformen verbreitet ist.

CVS bietet folgende Möglichkeiten:

1. Verwaltung von Dateien und Verzeichnissen

   Ein Teilprojekt im Roboterprojekt besteht typischerweise aus einem Verzeichnisbaum mit fest vorgegebener Architektur. Ein solches Teilprojekt wird unter CVS als *Modul* bezeichnet. (Im weiteren Verlauf dieses Textes ist mit *Modul* immer ein solches Teilprojekt bzw. ein Verzeichnisbaum gemeint).

   Alle Module, die durch CVS verwaltet werden, befinden sich in einem speziellen Verzeichnis (Repository) und sollten normalerweise nicht angefaßt und keinesfalls von Hand verändert werden.

2. Unterschiedliche Sichten

   Jeder Entwickler, der an einem Modul arbeitet, verfügt über seine private Sicht auf dieses Modul. Die private Sicht ist eine äquivalente Verzeichnisstruktur im privaten Home-Verzeichnis des Entwicklers. Unterschiedliche Entwickler können unterschiedliche Sichten auf ein und dasselbe Modul besitzen (z.B. unterschiedliche Versionen).

3. Verschiedene Entwicklungszweige

   Durch die unterschiedlichen Sichten ergibt sich sofort das Konzept von unterschiedlichen Entwicklungszweigen. Diese Zweige können durch CVS wieder vereinigt werden, oder aber als vollwertige Abzweigungen weitergeführt werden.

Weitere Dokumentation zu CVS, die über diese Kurzübersicht hinausgeht, befindet sich an folgenden Stellen:

1. Manual Pages zu cvs und rcs
2. Tutorial, Frequently Asked Questions, Postscript Texte unter `/usr/local/bv/info`
3. Per Emacs über `M-x info m cvs RETURN`
4. Lokale Newsgruppe `inf.ml.cvs-info` (Kopie der Mail-Liste zu CVS)

2.2 Voraussetzungen für die Nutzung von CVS

Um CVS verwenden zu können, muß die Shell Environment-Variable `CVSROOT` auf den Pfadnamen des Repository (s.o.) gesetzt werden. In unserem Fall ist dies das Verzeichnis `/usr/local/bv/robot/CVS`. Folgendes Kommando müßte ausgeführt werden:

```
setenv CVSROOT /usr/local/bv/robot/CVS
```

Ein Weg, dies automatisch auszuführen, ist eine entsprechende Eintragung in der privaten `.cshrc` Datei oder ein automatisches Importieren aller Roboter relevanten Erweiterungen von .cshrc durch

```
set ROB_USER
set PVM_USER
source /usr/local/bv/rc/cshrc
```

Diese Kommandos aollten am besten gleich vom privaten `.cshrc` aus ausgeführt werden. Neueinsteiger sollten die angedeutetet automatische Methode anwenden. Hierdurch werden auch noch einige andere wichtige Variablen gesetzt.

Durch das Setzen der Variable `CVSROOT` wird automatisch das Zielverzeichnis für sämtliche CVS Kommandos bestimmt. Wer gerne weitere, andere Projekte (nicht Roboter) mit CVS verwalten möchte, muß entprechend ein privates Repository aufbauen und die Variable entsprechend setzen. Dies wird hier aber nicht weiter besprochen.

Eine weitere notwendige Voraussetzung ist die Zugehörigkeit zu einer bestimmten Gruppe (hier: `bvrobot`). Betreuer von Studien- und Diplomarbeiten müssen bei der Beantragung eines Accounts auf diese Gruppe hinweisen. Vor dem Arbeiten mit dem COMROS CVS System sollte grundsätzlich das Kommando

```
newgrp bvrobot
```

ausgeführt werden. Ausschließlich diejenigen Benutzer, die das Schreibrecht für diese Gruppe haben, können die Module, die von CVS unter dem oben angegebenen Pfad verwaltet werden, manipulieren.

Während der Arbeit mit CVS werden vom Entwickler immer wieder Kommentare zur Änderungsgeschichte der privaten Sicht verlangt (z.B. beim Erzeugen neuer Datein oder beim Freigeben einer neuen Version). Diese Kommentare werden durch einen Editor aufgenommen, der von CVS gestartet wird. Falls die Shell Environment Variable `EDITOR` gesetzt ist, so wird der dort angegebene Editor verwendet. Ist diese Variable nicht gesetzt, so wird der Standard-Editor `vi` aufgerufen.

2.3 Grundlegende Kommandos von CVS

Die hier vorgestellten Kommandos stellen nur einen sehr kleinen Teil der großen Funktionalität der CVS Kommandos dar. Zur weiteren Information sei ausdrücklich auf die angegebene Dokumentation verwiesen.

Jedes CVS Kommando erlaubt grundsätzlich die Option `-H`, die eine kurze online-Hilfe des entsprechenden Kommandos anzeigt und weiter keine Funktion ausführt.

Jedes CVS Kommando wird grundsätzlich in einem privaten Verzeichnis ausgeführt, nämlich dort, wo sich die private Sicht des zu bearbeitenden Teilprojektes (Moduls) befindet, bzw. wo diese private Sicht entstehen soll. Es ist normalerweise nie notwendig und **sollte unbedingt unterlassen werden**, direkt Dateien des Repository zu lesen oder zu schreiben. Letzteres könnte fatale Folgen für die gesamte Projektorganisation nach sich ziehen.

2.3.1 Erzeugen einer privaten Sicht

Um irgendeine der nachfolgend noch beschriebenen Operationen auszuführen zu können, muß eine private Sicht auf ein Modul bestehen. Durch das Kommando

`cvs checkout` *module*

wird eine private Sicht des Verzeichnisbaums *module* im aktuellen Verzeichnis angelegt. Bei dieser einfachen Form der Anwendung des checkout-Befehls wird immer die neueste verfügbare Version erstellt.

Durch die Ausführung des Befehls entsteht eine komplette Verzeichnisstruktur im aktuellen Verzeichnis. Sämtliche Dateien dieser Struktur sind ausschließlich für den Auftraggeber schreibbar. Eine weitere Anmeldung, um nun tatsächlich Änderungen vorzunehmen, ist nicht notwendig.

Innerhalb der entstandenen Verzeichnisstruktur ist in jedem Unterverzeichnis ein neues Verzeichnis mit dem Namen `CVS` entstanden. Hier legt das CVS System während der weiteren Bearbeitung wichtige Informationen ab. Unter keinen Umständen dürfen Inhalte dieser `CVS` Verzeichnisse verändert werden.

2.3.2 Hinzufügen von Dateien

Sollen in einem Modul weitere Dateien unter die Verwaltung von CVS gestellt werden (weitere C-Sourcen, Dokumentation, Skripte, usw.) so reicht es nicht aus, diese Dateien einfach nur zu erzeugen. CVS ignoriert bei einem späteren Freigeben des Moduls nämlich alle Dateien, die nicht schon früher von CVS verwaltet wurden. (Hier wird allerdings eine ausführliche Warnmeldung ausgegeben). Dieses Verhalten ist sinnvoll, da z.B. Binärdateien und Libraries nicht mit CVS verwaltet werden sollen, obwohl sie typischerweise während der Entwicklung im Verzeichnisbaum des Moduls entstehen.

Sobald eine neue Datei angeleget wurde, kann sie mit dem Kommando

`cvs add` *file*

unter die Verwaltung von CVS gestellt werden. Die Datei *file* muß sich innerhalb der privaten Sicht des Moduls im aktuellen Verzeichnis befinden.

2.3.3 Löschen von Dateien

Ähnlich wie das Hinzufügen von Dateien muß auch das Löschen von Dateien explizit dem CVS System bekanntgegeben werden. Dies gilt aber nur für Dateien, die bisher bereits von CVS verwaltet wurden. Zum Löschen wird die Datei zunächst mit dem Unix Kommando `rm` entfernt und anschließend mit dem CVS Komando

`cvs remove` *file*

aus der Verwaltung von CVS gestrichen. Die Lebensgeschichte der gelöschten Datei bleibt dabei jedoch unter der Verwaltung von CVS und es ist später weiterhin möglich, alte Versionen dieser Datei zu extrahieren.

2.3.4 Überprüfen der privaten Sicht

Während der Weiterentwicklung eines Moduls auf der privaten Sicht können gleichzeitig andere Entwickler ebenfalls dieses Modul weiterentwickeln und ihre Änderungen evtl. bereits in das Repository zurückgestellt haben (s.u.). Es besteht also grundsätzlich jederzeit die Möglichkeit, daß die private Sicht nicht mehr mit der Spitze des entsprechenden Entwicklungszweigs übereinstimmt.

Um diese Unterschiede festzustellen und auch um die eigenen Änderungen mit der urprünglich erzeugten privaten Sicht zu vergleichen, geht man zunächst in das Wurzelverzeichnis des Moduls. Dort kann man sich mit dem Befehl

`cvs diff` [*files* ...]

sämtliche Änderungen aller Dateien des Moduls anzeigen lassen, oder bestimmte Dateien herausgreifen.

In den meisten Fällen reicht es jedoch aus, nicht ausdücklich alle Unterschiede aufgelistet zu erhalten (wie dies bei `diff` üblich ist), sondern eine kurze Status Information über den aktuellen Zustand der Dateien des Moduls reicht aus. Diese Status Information kann auch dazu verwendet werden, den möglichen Effekt eines `update` Befehls (siehe nächster Abschnitt) abzuschätzen. Der Status Befehl lautet:

`cvs status` [*files* ...]

2.3.5 Private Sicht auf den neusten Stand bringen

Bevor die private Sicht eines Moduls als fertige Version in das Repository zurückgestellt wird, muß man sich entscheiden, ob hierdurch ein neuer Entwicklungszweig eingeleitet werden soll, oder ob evtl. zwischenzeitlich von anderen Entwicklern durchgeführte und freigegebene Änderungen an diesem Modul übernommen werden sollen.

Die (halb)automatische Übernahme von anderen Änderungen (*Merge*) geschieht mit dem Befehl

`cvs update` [*files* ...]

Hierdurch werden die angegebenen Dateien oder aber der gesamte Verzeichnisbaum auf die neuste Version angepaßt. Für das Ergebnis eines `update` Befehls auf einer bestimmten Datei gibt es sechs unterschiedliche Fälle, die durch eine speziellen Buchstaben als Statusmeldung zusammen mit dem Dateinamen ausgegeben werden:

- `U` *file*
  file im Repository hatte einen neueren Inhalt als die private Kopie. Die private Kopie war unverändert gegenüber dem urprünglichen `checkout`. Sie wurde durch die neue Version ersetzt.
- `A` *file*
  file ist bisher noch nicht im Repository enthalten, wurde aber durch einen `cvs add` Befehl bereits angemeldet
- `R` *file*
  file wurde aus der privaten Sicht per `cvs remove` Befehl bereits gelöscht, ist momentan aber noch im Repository enthalten
- `M` *file*
  file in der privaten Sicht ist gegenüber dem Repository verändert. Die Änderungen werden bei der Freigabe in dieser Form übernommen. Eventuelle weitere Änderungen, die zwischenzeitlich von anderen Entwicklern freigegeben wurden, wurden erfolgreich in die private Kopie der Datei eingebaut (*Merge*)
- `C` *file*
  Beim Versuch eine Merge-Operation auszuführen ist ein Konflikt aufgetreten. Als Ergebnis enthält *file* nun die Ausgabe des Kommandos `rcsmerge` (siehe Manual Page). Die urprüngliche private Kopie wurde unter dem Namen `.#`*file*`.`*version* abgelegt.
- `?` *file*
  Die angegebene Datei befindet sich in der privaten Sicht aber nicht im Repository. Eventuell wurde eine `add` Kommando für diese Datei bisher vergessen oder aber es handlet sich um eine Datei, die bewußt nicht unter der Verwaltung von CVS steht.

2.3.6 Eigene Änderungen der Allgemeinheit zur Verfügung stellen

Die endgültige Freigabe der Änderungen der privaten Sicht, also das Freigeben einer neuen Version bzw. die Übertragung der Dateien in das Repository geschieht mit dem Kommando

`cvs commit` [*files* ...]

Hierdurch wird entweder das gesamte Modul freigegeben (beim Aufruf ohne *file* Argumente) oder nur bestimmte Dateien. Weitere Einzelheiten, wie z.B. die Erzeugung eines neuen Entwicklungszweigs oder die Zuweisung einer expliziten Marke für diese Version, sind der weiteren Dokumentation von CVS zu entnehmen.

2.3.7 Eigene Änderungen aufgeben bzw. Bearbeitung abbrechen

Nach der Freigabe einer privaten Sicht durch `commit`, soll eventuell die private Sicht aus dem privaten Verzeichnis entfernt werden. Dies kann durch einen gezielten Unix `rm` Befehl geschehen, dessen Anwendung allerdings nicht empfehlenswert ist. Besser ist es, den durch CVS bereitgestellten Befehl zu verwenden, da hierbei nochmals die Konsistenz der privaten Sicht mit der aktuellen freigegebenen Sicht überprüft wird.

Ein anderer Fall, der eintreten kann, ist, daß die Änderungen auf der privaten Sicht nicht freigegeben werden sollen (es soll also kein commit ausgeführt werden), aber trotzdem dauerhaft entfernt werden sollen.

In beiden Fällen sollte der Befehl

`cvs release [-d]` *module*

angewendet werden. Hierzu muss zunächst das private Verzeichnis aufgesucht werden, in dem sich das Wurzelverzeichnis von *module*, also die Wurzel der privaten Sicht befindet. Durch die Option `-d` wird nach erfolgreicher Überprüfung und nochmaliger Rückfrage der entsprechende Verzeichnisbaum gelöscht. Ohne diese Option werden keine Dateien gelöscht. Jedoch wird in diesem Fall im Repository vermerkt, daß der Entwickler widerruflich bekanntgegeben hat, daß er an diesem Modul keine Änderungen mehr vornimmt. Er behält jedoch eine private Kopie zurück.

2.3.8 Änderungsgeschichte ansehen

Die Änderungsgeschichte von Modulen, das checkout, commit und release usw. wird in einer globalen History-Datei protokolliert. Diese Datei kann durch das Kommando

`cvs history`

ausgewertet werden. Es bestehen viele möglichen Optionen, die alle im entprechenden Manual beschrieben sind.

2.3.9 *emacs* Interface zu CVS

Benutzer des Editors *emacs* können durch die Ergänzung der Zeile

```
(autoload ‘cvs-update “pcl-cvs“ “Run CVS update“ t)
```

in ihrem `.emacs` File den Befehl

```
M-x cvs-update
```

verwenden, der viele Möglichkeiten von CVS in einem *emacs* Buffer mit ansprechender Bedienung bereitstellt. Näheres erfährt man in der vorhandenen online-Dokumentation.

2.4 Einrichten eines neuen Roboterteilprojektes

Das folgende Unterkapitel beschreibt, wie ein neues Teilprojekt mit CVS angelegt werden kann. Da in der Zwischenzeit sehr leistungsfähige Skripte erstellt wurden, die dies automatisch ausführen und da sich außerdem inzwischen die Struktur der Verzeichnisse verändert hat, sollten die nachfolgend beschriebenen Schritte nicht von Hand ausgeführt werden. Sie sind trotzdem in der Dokumentation zu CVS enthalten, um noch einmal die beschriebenen Elementarbefehle, die natürlich zum Teil weiterhin direkt ausgeführt werden müssen, im Zusammenhang zu demonstrieren.

Am Beispiel der Erzeugung eines neuen Teilprojektes soll nun exemplarisch die Verwendung von CVS demonstriert werden. Hierbei werden auch zwei Befehle verwendet, die bisher nicht besprochen wurden, die aber für den täglichen Gebrauch von untergeordneter Bedeutung sind.

Für die Erzeugung eines neuen Teilprojektes steht eine generische Projektschablone zur Verfügung, die ebenfalls unter CVS verwaltet ist und bei Bedarf erweitert wird. Das hier beschriebene Beispiel durchläuft die folgenden Schritte:

1. Es wird eine Kopie der generischen Projektschablone in einem privaten Verzeichnis angelegt
2. Der Name des Teilprojektes (Modulname) wird festgelegt. Alle entsprechenden Dateien in der privaten Kopie werden entspreched anderer Richtlinien (siehe Kapitel Projektverwaltung) modifiziert.
3. Das neue Modul wird in die Moduldatenbank des Repository von CVS eingetragen. Diese Moduldatenbank ist selbst auch durch CVS verwaltet. Entsprechend sind einige CVS Funktionen aufzurufen.
4. Das neue Teilprojekt wird erstmalig unter die Kontrolle von CVS gestellt.
5. Die private Kopie des neuen Teilprojekts wird per Unix `rm` gelöscht. Hier darf noch nicht das `release` Kommando von CVS verwendet werden, da bisher ja auch noch kein `checkout` stattgefunden hat.
6. Mit `checkout` wird nun eine neue Arbeitsversion des Teilprojektes generiert.
7. Ab jetzt können alle gewünschten Arbeiten, die im letzten Unterkapitel beschrieben wurden, durchgeführt werden.

Es folgt nun die ausführliche Beschreibung der einzelnen Punkte, die in dieser Form jederzeit nachvollzogen werden können. Alle Arbeiten finden im privaten Home-Verzeichnis `$HOME` statt. Das neue Teilprojekt erhält den exemplarischen Namen `Bsp`.

2.4.1 Generisches Projekt erzeugen

Zunächst wird die Voraussetzung für das korrekte Funktionieren von CVS durch Setzen von CVSROOT und EDITOR sichergestellt:

```
matisse:[~] >cd $HOME
/home/vogt
matisse:[~] >setenv CVSROOT /usr/local/bv/robot/CVS
matisse:[~] >setenv EDITOR emacs
```

Nun wird mit dem export Kommando ein generischer Teilprojektbaum im privaten Verzeichnis angelegt. Da hierzu eine Versionsnummer oder eine Zeitangabe notwendig ist, wurde exemplarisch date aufgerufen:

```
matisse:[~] >date
Thu Apr 7 15:07:23 MET DST 1994
matisse:[~] >cvs export -D '15:07:23' TPR
cvs export: Updating TPR
U TPR/Makefile
cvs export: Updating TPR/cmd
cvs export: Updating TPR/data
cvs export: Updating TPR/include
cvs export: Updating TPR/man
cvs export: Updating TPR/src
U TPR/src/Makefile
cvs export: Updating TPR/src/HPPA
U TPR/src/HPPA/CONFIG.make
cvs export: Updating TPR/src/MASPAR
U TPR/src/MASPAR/CONFIG.make
cvs export: Updating TPR/src/SUN4
U TPR/src/SUN4/CONFIG.make
cvs export: Updating TPR/src/SUN4SOL2
U TPR/src/SUN4SOL2/CONFIG.make
```

Es ist nun eine Verzeichnisstruktur mit Namen TPR entstanden:

```
matisse:[~] >ls -ld TPR
drwxr-xr-x 3 vogt    512 Apr  7 15:08 TPR
```

2.4.2 Anpassung an das neue Projekt

Das neue Projekt soll den Namen Bsp erhalten. Hierzu wird einfach mit dem Unix Befehl mv der Name des Verzeichnisses verändert:

```
matisse:[~] >mv TPR Bsp
matisse:[~] >ls -ld Bsp
drwxr-xr-x 3 vogt    512 Apr  7 15:08 Bsp
```

Weiterhin müssen nun verschiedene Dateien innerhalb des Verzeichnisbaums an die eigenen Wünsche angepaßt werden. Da diese Änderungen nicht mit CVS zusammenhängen, sind sie an anderer Stelle beschrieben (Kapitel Projektverwaltung).

2.4.3 Eintragung in die Moduldatenbank

Das neue Modul muß nun in die Moduldatenbank von CVS eingetragen werden. Da die Moduldatenbank selbst von CVS verwaltet wird, erfolgen einige CVS Aufrufe, die genau in dieser Form wiederholt werden müssen. Zunächst wird wieder sichergestellt, dass man sich im privaten Home-Verzeichnis befindet:

```
matisse:[~] >cd $HOME
/home/vogt
matisse:[~] >cvs checkout modules
U modules/modules
matisse:[~] >ls -ld modules
drwxr-xr-x 3 vogt    512 Apr  7 15:31 modules
matisse:[~] >cd modules
/home/vogt/modules
matisse:[modules] >ls -l
total 3
drwxr-xr-x 2 vogt    512 Apr  7 15:31 CVS
-rw-r--r-- 1 vogt   1349 Mar 25 17:47 modules
matisse:[modules] >emacs modules
```

In der Datei `modules` wird die letzte Zeile eingefügt, so daß das Ende der Datei etwa folgendermaßen aussieht:

```
# Add other modules here...
TPR TPR
Bsp Bsp
```

Anschließend kann z.B. die Wirkung des `status` Befehls demonstriert werden:

```
matisse:[modules] >cvs status
cvs status: Examining .
===================================================================
File: modules Status: Locally Modified

Version: 1.2 Fri Mar 25 17:47:26 1994
 RCS Version: 1.2 /usr/local/bv/robot/CVS/CVSROOT/modules,v
 Sticky Tag:     (none)
 Sticky Date:    (none)
 Sticky Options: (none)
```

Um die Änderung wirksam zu machen wird das Modul `modules` wieder freigegeben. Vorher wird noch ein `update` ausgeführt:

```
matisse:[modules] >cvs update
cvs update: Updating .
M modules
matisse:[modules] >cvs commit
cvs commit: Examining .
cvs commit: Committing .
Checking in modules;
/usr/local/bv/robot/CVS/CVSROOT/modules,v <-- modules
new revision: 1.3; previous revision: 1.2
```

```
done
cvs commit: Executing 'mkmodules /usr/local/bv/robot/CVS/CVS-
ROOT'
```

Während der Bearbeitung von `commit` wurde nun wiederum ein Editor gestartet, um die Änderung an der Moduldatenbank zu kommentieren. Dies geschieht automatisch und ist hier nicht dargestellt. Zum Abschluß wird die private Sicht auf das Modul `modules` wieder entfernt:

```
matisse:[modules] >cd ..
/home/vogt
matisse:[~] >cvs release -d modules
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module 'modules':
y
```

2.4.4 Bereitstellen des neuen Teilprojektes

Das angepaßte Teilprojekt `Bsp` ist bisher nur als Modulname bekanntgegeben. Das Projekt selbst ist aber noch nicht unter die Kontroll von CVS gestellt. Dies wird im nächsten Schritt durchgeführt. Absolut wichtig ist hierbei, daß zuvor das Wurzelverzeichnis des neuen Projektes aufgesucht wird, da der verwendete `import` Befehl alle Dateien des aktuellen Verzeichnis rekursiv als neues Projekt einspielt:

```
matisse:[~] >cd Bsp
/home/vogt/Bsp
matisse:[Bsp] >cvs import Bsp VOGT START
N Bsp/Makefile
cvs import: Importing /usr/local/bv/robot/CVS/Bsp/src
N Bsp/src/Makefile
cvs import: Importing /usr/local/bv/robot/CVS/Bsp/src/HPPA
N Bsp/src/HPPA/CONFIG.make
cvs import: Importing /usr/local/bv/robot/CVS/Bsp/src/MASPAR
N Bsp/src/MASPAR/CONFIG.make
cvs import: Importing /usr/local/bv/robot/CVS/Bsp/src/SUN4
N Bsp/src/SUN4/CONFIG.make
cvs import: Importing /usr/local/bv/robot/CVS/Bsp/src/SUN4SOL2
N Bsp/src/SUN4SOL2/CONFIG.make

No conflicts created by this import
```

Der `import` Befehl hat drei Parameter. Der erste Parameter ist der Modulname (`Bsp`). Der zweite und dritte Parameter sind zwei Marken, die für den Erzeuger des Moduls (`VOGT`) und für einen symbolischen Release-Namen (`START`) stehen.

2.4.5 Entfernen der Urversion des neuen Projektes

Bevor irgendwelche weiteren Änderungen gemacht werden, wird nun die private Urversion des Projektes entfernt. Wird dies nicht gemacht, so kann es passieren, daß

CVS später bei der Verwaltung des neuen Moduls im eigenen Verzeichnis Probleme bekommt, da dieses Verzeichnis nicht durch ein `checkout` eingerichtet wurde:

```
matisse:[Bsp] >cd ..
/home/vogt
matisse:[~] >rm -r Bsp
```

2.4.6 Erstellen einer privaten Sicht

Durch ein `checkout` wird nun eine private Sicht des Moduls `Bsp` erstellt:

```
matisse:[~] >cvs checkout Bsp
cvs checkout: Updating Bsp
U Bsp/Makefile
cvs checkout: Updating Bsp/src
U Bsp/src/Makefile
cvs checkout: Updating Bsp/src/HPPA
U Bsp/src/HPPA/CONFIG.make
cvs checkout: Updating Bsp/src/MASPAR
U Bsp/src/MASPAR/CONFIG.make
cvs checkout: Updating Bsp/src/SUN4
U Bsp/src/SUN4/CONFIG.make
cvs checkout: Updating Bsp/src/SUN4SOL2
U Bsp/src/SUN4SOL2/CONFIG.make
matisse:[~] >cd Bsp
/home/vogt/Bsp
matisse:[Bsp] >ls -l
total 5
drwxr-xr-x 2 vogt    512 Apr  7 16:09 CVS
-rw-r--r-- 1 vogt   3004 Apr  7 15:52 Makefile
drwxr-xr-x 7 vogt    512 Apr  7 16:10 src
```

Auffällig an dieser privaten Sicht ist, daß nun in jedem Verzeichnis des erzeugten Verzeichnisbaums ein Verzeichnis mit Namen CVS steht. Hier werden zukünftige Änderungen protokolliert.

2.4.7 Bearbeitung

Die Bearbeitung erfolgt mit den üblichen Unix Tools. Am Rande kann hier noch die Wirkung eines `history` Befehls demonstriert werden, der hier nun die sehr kurze Geschichte des Projektes `Bsp` aufzeigt:

```
matisse:[Bsp] >cvs history -m Bsp
O 04/07 16:09 vogt Bsp =Bsp= ~/*
```

Die Ausgabe dieses Befehls beschreibt, daß `vogt` das Modul `Bsp` am 4.7. um 16:09 Uhr per `checkout` in sein Home-Verzeichnis kopiert hat.

2.5 Literatur

[1] Manual zu CVS, online Dokumentation, erreichbar mit `man cvs`

[2] CVS Infoseiten, online Dokumentation, erreichbar im *emacs* mit `M-x info m cvs`

[3] PCL-CVS Infoseiten, online Dokumentation zum *emacs* Frontend zu CVS, erreichbar im *emacs* mit `M-x info m pcl-cvs`

Kapitel 3

RoI (Robot Interface)

Alexander Rausch

RoI stellt die unterste Schnittstellenebene für den programmgesteuerten Betrieb der ROBOSOFT - Fahrzeuge dar. Die Befehlsübergabe und das Auswerten der Roboterantwort erfolgen über das Versenden von Zeichenketten, die dem ALBATROS - Befehlssatz entsprechen. Diese Beschreibung ist ab **Version 3-1** gültig.

3.1 Hardwareumgebung

Die Roboterfahrzeuge sind über Funk ansprechbar. RoI unterstützt den Einsatz der von ROBOSOFT gelieferten Modems und die Verwendung des MOTOROLA Funkethernets in Verbindung mit den WT-Ethernet-RS232 Umsetzern. Die aktuelle Verkabelung spiegelt sich im Systemfile `/usr/local/bv/robot/etc/system.RoIrc`. Abhängig von der Art der Funkverbindung zu einem Fahrzeugs deckt RoI folgende Anwendungsfälle ab:

- ROBOSOFT-Modem: Das Anwenderprogramm läuft auf dem Steuerrechner, an dem das Fahrzeug angebunden ist.
- Funkethernet: Das Fahrzeug kann von allen Rechnern angesprochen werden.

Es besteht die Möglichkeit, ein eigenes Konfigurationsfile `$(HOME)/.RoIrc` File anzulegen, sodaß das Systemfile nicht verwendet wird. Dies ist nur sinnvoll, wenn die Verkabelung verändert wird. **Da nur Mitarbeiter die Verkabelung ändern dürfen (in Absprache mit dem Laborleiter), erübrigt sich für Studenten das Anlegen eines eigenen Konfigurationsfiles**. In späteren Versionen wird zur Absicherung die Gruppenmitgliedschaft des Anwenders überprüft werden, um Mißbrauch zu vermeiden.

WARNUNG: Eine unsachgemäße Veränderung der Verkabelung kann unter Umständen die Beschädigung der Steuerrechner, der Funkmodems oder der Funkethernetgeräte nach sich ziehen. Veränderungen an der Hardwarekonstellation ist Studenten ausdrücklich untersagt.

3.2 Befehlssatz

Alle Befehle und Datentypen besitzen einheitlich den Präfix `RoI_`, um Schwierigkeiten beim Linken der Programme aus dem Weg zu gehen. Jeder Roboter wird über einen Handle vom Datentyp `RoI_Handle` angesprochen. Der aktuell verfügbare Funktionsvorrat besteht aus je einer Funktion zum

- Öffnen der Verbindung zum Fahrzeug
- Absetzen von Befehlen
- Schließen der Verbindung zum Fahrzeug
- Rücksetzen des Roboterfahrzeugs
- Debuggen von Programme, indem zusätzlich auf einem Logfile im /tmp/RoI.xxxxx (xxxxx = Benutzernummer im UNIX System) Debuginformation ausgedruckt wird.

Detaillierte Erläuterungen zu den Funktionen finden sich in den man-pages

3.3 Beispiel

```
#include“RoboInterface.h“
void main(int argc, char *argv[])
{
RoI_Handle robot;
char *reply;
RoI_debug(RoI_Log);
RoI_open(&robot, argv[1]);
RoI_send(robot,“MOTV ON“,&reply);
RoI_reset(robot);
RoI_send(robot,“RNOD R=ON“,&reply);
RoI_send(robot,“READ C=1,110000,000000“,&reply);
RoI_send(robot,“ODOM ON“,&reply);
RoI_send(robot,“ODOM“,&reply);
RoI_send(robot,“MOTV OF“,&reply);
RoI_close(robot);
}
```

Die einzubindende Library befindet sich im Verzeichnis

`/usr/local/bv/robot/lib/`*Rechnerarchitektur*`/RoI`*Versionsnummer*.

3.4 Abbildung der Verkabelung

Das im Verzeichnis /usr/local/bv/robot/etc abgelegte File system.RoIrc hat folgenden Aufbau:

<roboternamе> <hostname><port>

1. <roboternamе>: porthos,aramis, athos
2. <hostname>: Steuerrechner, falls das Fahrzeug ueber serielle Schnittstelle angesprochen wird. Ansonsten der Name des Ethernet-RS232-Umsetzers
3. <port>: Bei seriellen Schnittstellen mit Bezeichnung /dev/ttyx: SI_x, bei den Ethernet-RS232-Umsetzern und Ausgabeport x: WT_x

Es können Kommentarzeilen eingefügt werden, die mit # beginnen.

3.5 Verwendung der Sourcen

Teilprojektname: `RoI (Robot Interface)`
aktuelle Version: `RoI4-0`

Library: `libRoI.a`
Beschreibung: Routinen zum Ansprechen der Roboter
Architekturen: `SUN4 SUNMP MASPAR`
Includes: `#include "RoI/RobotInterface.h"`
Linkoptionen: `-lRoI`

Programme: `RoI_test`
Beschreibung: Testprogramm für die Library-Wartung
Architekturen: `SUN4 SUNMP MASPAR`

Kapitel 4

Verwendung von DRI

(Distributed Robots Interface)

Alexander Rausch

Der Ethernetanschluß auf allen Roboterfahrzeugen und eine entsprechende Erweiterung von RoI hat die DRI praktisch überflüssig gemacht. Nur bei Verwendung des Simulationssystems MOBS ist der Einsatz von DRI noch sinnvoll.

Die in dieser Beschreibung vorgestellte Roboterschnittstelle ermöglicht die Ansteuerung der vorhandenen mobilen Fahrzeuge namens „Portos“, „Athos“ und „Aramis“ auf einheitliche Weise. Diese Schnittstelle stellt eine Grundfunktionalität in der Programmiersprache C zur Verfügung, auf die Steuerprogramme für die Fahrzeuge aufsetzen können. Insbesondere werden in DRI keine anwendungsspezifischen Funktionen bereitgestellt, da diese dem Charakter der allgemein verwendbaren Schnittstelle widersprechen würden. Anwendungsspezifische Funktionalität muß somit in weiteren Projekten, die auf DRI aufbauen, bereitgestellt werden. Beispielsweise sind hier Verfahren zu nennen, die eine besondere Strategie zum Auslesen der Fahrzeugsensoren Ultraschall und Odometrie betreffen. Eine Erweiterung in dieser Hinsicht wird jedoch dann erfolgen, wenn sich herausstellt, daß die getesteten Verfahren von allgemeinem Nutzen sind.

Wesentlicher Unterschied sämtlicher Weiterentwicklungen auf höherer Ebene wird die objektorientierte Gestaltung der Schnittstelle sein. So wird Funktionalität bereitgestellt werden, die C++ - Klassen zur Sensorabfrage und Fahrzeugbewegung verwendet.

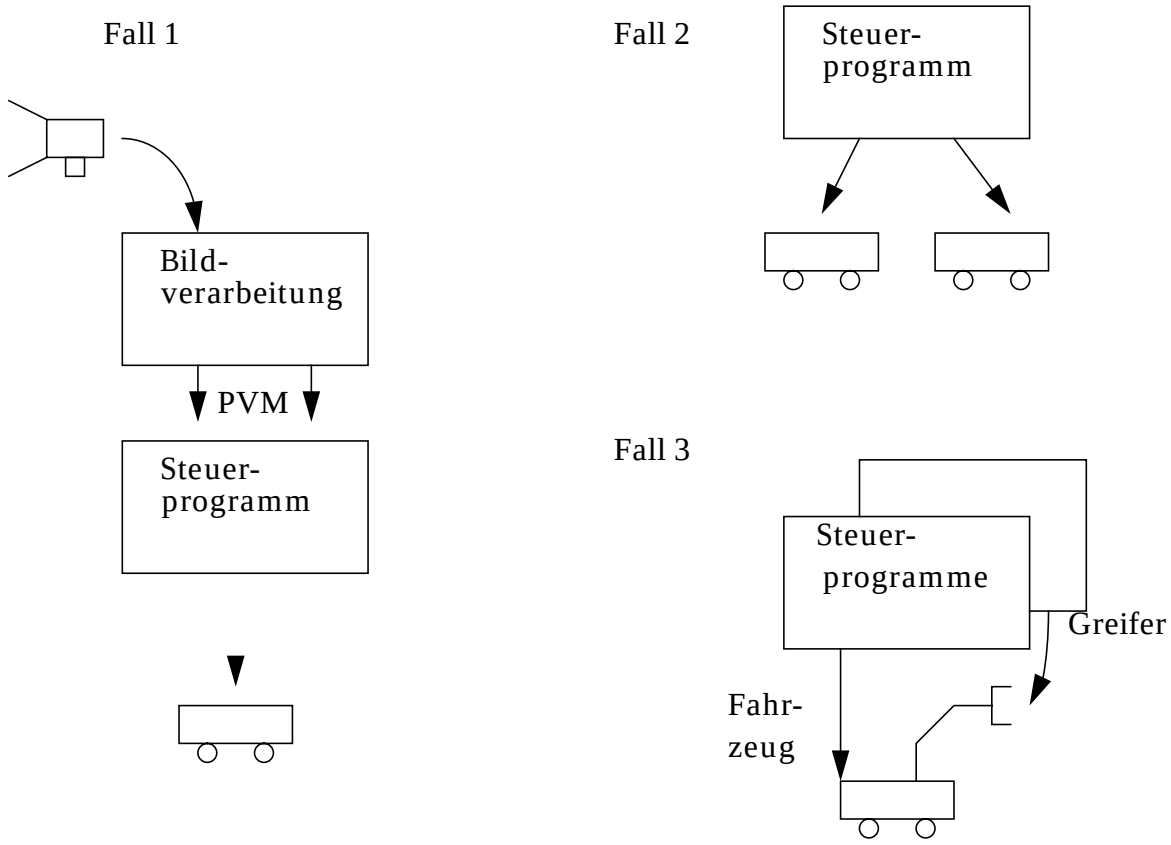
Nach wie vor wird die vorhandene C - Schnittstelle jedoch im hier vorgestellten Umfang unterstützt, um auch von der MasPar aus die Fahrzeuge steuern zu können.

4.1 Einleitung

Im Roboterlabor sind alle mobilen Fahrzeuge an je einen dedizierten Steuerrechner angeschlossen. Derselbe Sachverhalt trifft für die vorhanden 6D - Steuerkugeln zu. Die Videosignale können über ein Switchboard zur weiteren Bildverarbeitung auf die MasPar, das Eltec-Board oder die SUN geführt werden. Somit ergibt sich die im folgenden beschriebene Problematik:

1. Die feste Zuordnung der Fahrzeuge zu Steuerrechnern und die beliebige aber softwaremäßig nicht zu beeinflussende Zuordnung der Videosignale zu Bildverarbeitungsrechnern bewirkt, daß Fahrzeugsteuerrechner und Bildverarbeitungsrechner nicht identisch sein müssen. Die Verbindung zwischen Bildverarbeitungs- und Steuerrechner kann über Interprozeßkommunikation, hier mit dem Werkzeug PVM [1] geschlossen werden.
2. Die Ansteuerung eines Fahrzeuges ist ohne Einsatz von Interprozeßkommunikation nur von dem Rechner und das Fahrzeug möglich, das an dem betreffenden Steuerrechner über die serielle Schnittstelle angeschlossen ist. Die Interprozeßkommunikation ermöglicht es, einen „Serverprozeß“ auf einem entfernten Rechner zur Ansteuerung eines weiteren Fahrzeugs zu starten und die Fahrzeugsteuerbefehle über Nachrichtenaustausch weiterzuleiten.
3. Der Einsatz mehrerer Steuerprogramme zur Steuerung eines einzigen Fahrzeugs ist naturgemäß nur über den Einsatz von Interprozeßkommunikation möglich.

Nachfolgende Abbildung veranschaulicht die Problematik.



4.2 Funktionalität

4.2.1 Direkte Ansteuerung der Fahrzeuge

DRI erlaubt den direkten Zugriff auf dieTTY - Schnittstelle. Dies ist z.B. dann sinnvoll, wenn Bildverarbeitung und Ansteuerung auf demselben Rechner stattfinden. Das Programm hat dann prinzipiell folgende Struktur:

Programm 4.1: Direkte Ansteuerung der Fahrzeuge

```
1  #include“rob.h“
2  char reply[256];
3  int aramis = rob_init(”aramis“);
4  ...
5  rob_command(aramis,“MOTV ON“,reply);
6  ...
7  rob_exit(aramis);
```

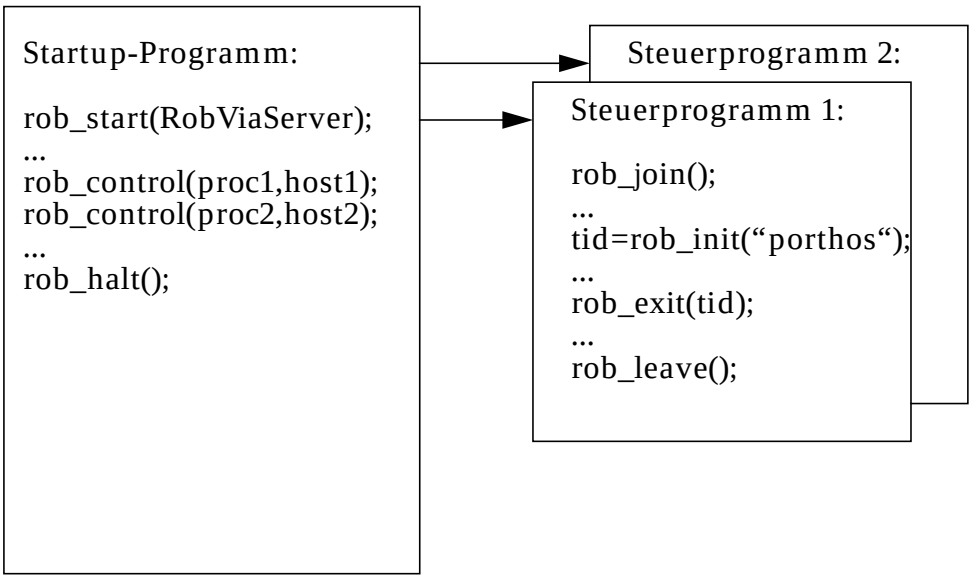
4.2.2 Indirekte Ansteuerung der Fahrzeuge

Eine indirekte Ansteuerung der Fahrzeuge unter Zuhilfenahme von Interprozeßkommunikation ist erforderlich, wenn Bildverarbeitungsrechner und Steuerrechner nicht übereinstimmen. Wird beispielsweise die MasPar zur Bildverarbeitung eingesetzt, so muß das Bildverarbeitungsprogramm mit einem Serverprozeß auf dem Fahrzeugsteuerrechner kommunizieren. Das Bildverarbeitungsprogramm hat dann folgenden strukturellen Aufbau:

Programm 4.2: Indirekte Ansteuerung der Fahrzeuge

```
1  include“rob.h“
2  char reply[256]; int aramis;
3  rob_start(RobViaServer);
4  aramis = rob_init(”aramis“);
5  ...
6  rob_command(aramis,“MOTV ON“,reply);
7  ...
8  rob_exit(aramis);
9  rob_halt();
```

4.2.3 Indirekte Ansteuerung über mehrere Steuerprogramme



In diesem Fall ist zu beachten, daß PVM vor dem Starten des Startup-Programms hochgefahren wird. Falls die Steuerprogramme bereits auf den Steuerrechnern der Roboter laufen, kann auch `rob_start(RobViaRS232)` verwendet werden. Diese Option bewirkt, daß bei `rob_init()` in den Steuerprogrammen direkt auf die tty - und somit ohne Instantiierung eines dedizierten Serverprozesses - geschrieben wird.

4.3 Voraussetzungen für den Einsatz

Der Einsatz von DRI erfordert die Einhaltung einiger weniger Voraussetzungen, die im folgenden beschrieben sind:

1. Setzen der Environmentvariablen. Dies kann z.B. durch Setzen einer entsprechenden Variable set `ROB_USER` erfolgen. Zuvor muß `PVM_USER` gesetzt werden. Das alte File `/usr/local/bv/adm/cshrc` darf nicht mehr eingesourced werden. Diese Funktion übernimmt das File `/usr/local/bv/rc/cshrc`, das unmittelbar nach `/usr/local/rc/cshrc` eingesourced werden sollte.
2. Auschecken des Schnittstelleninterfaces DRI aus CVS.
3. Bereitstellen eines `.pvm_hosts`. Ein Beispiel ist unter `/usr/local/bv/robot/etc/pvm_hosts` zu finden (Achtung: Dieses File sollte im eigenen $(HOME) unsichtbar unter dem Namen (`.pvm_hosts`) abgespeichert werden.
4. Im eigenen `$(HOME)` muß ein `.rhosts` vorhanden sein. Ansonsten ist es nicht möglich, auf den robosuns Prozesse remote zu starten. Ein Beispiel ist unter `/usr/local/bv/robot/etc/rhosts` zu finden (Achtung: Dieses File sollte im eigenen `$(HOME)` unsichtbar unter dem Namen (`.rhosts`) abgespeichert werden.

5. Standardmäßig wird die unter `/usr/local/bv/robot/adm/etc/system.robotrc` beschriebene Roboterkonfiguration verwendet. Falls eine andere Konfiguration gewünscht wird, muß im `$(HOME)` ein `.robotrc` vorhanden sein.
6. Die Libraries müssen mit

- `-L/usr/local/bv/robot/lib/$(BV_ARCH)/DRI2-0 -lDRI` und
- `-L/usr/local/bv/pvm3 -lpvm3`

hinzugebunden werden. Das Ausckecken sollte die entsprechenden Pfade automatisch setzen und die richtigen libraries finden (vgl. Kapitel Projektverwaltung).

4.4 Arbeitsweise

4.4.1 Direkte Ansteuerung der TTY - Schnittstelle

Falls eine Interprozeßkommunikation nicht benötigt wird, werden die mittels der Funktionen `rob_init(), rob_command()` und `rob_exit()` ausgelösten Aktionen direkt and die serielle Schnittstelle des Arbeitsplatzrechners weitergereicht.

4.4.2 Indirekte Ansteuerung der TTY - Schnittstelle

Falls die Anwendung den Einsatz der Interprozeßkommunikation erfordert, wird ein Systemprozeß gestartet, der für die Steuerung des Prozeßsystems erforderlich ist. Dieser Prozeß (genannt: administrator) verwaltet die Zugriffe auf die tty.

1. Zentraler Verwaltungsprozeß: Dieser Prozeß sequentialisiert alle Aufträge, die das Öffnen und Schließen einer seriellen Schnittstelle betreffen. Ein Programm, das eine Schnittstelle öffnen möchte, meldet sich hierzu beim sog. „Administrator“ an und erhält die Adresse des dedizierten TTY-Roboterserverprozesses, falls dieser schon instanziiert wurde. Falls der Roboterserverprozeß noch nicht läuft, wird er zuvor instanziiert. Eine Aufforderung zum Schließen der Schnittstelle bewirkt erst dann das Herunterfahren des Roboterserverprozesses, falls der Prozeß, der den Auftrag zum Schließen gab, der letzte Prozeß ist, der noch Zugriff auf den Roboterserverprozeß hat. Ansonsten wird lediglich der zugreifende Anwenderprozeß aus der Liste der auf den Roboterserverprozeß zugreifenden Prozesse ausgetragen. Beim Absetzen von Fahrkommandos, Sensorabfragen, usw. kommuniziert das Anwendungsprogramm direkt mit dem Roboterserverprozeß.
2. Der Messageserver wird nicht mehr benoetigt, da DRI auf den Einsatz von XPVM vorbereitet ist. Der testweise Einsatz von XPVM erfolgte bereits, eine abteilungsweite Installation kann jedoch aufgrund eines PVM-Bugs erst mit der nächsten major Release von PVM3.4.x erfolgen. XPVM beinhaltet bereits die Funktionalität des zuvor implementierten Messageservers.

4.4.3 Ansteuerung des Simulators

Die Simulationsumgebung mobs kann verwendet werden, indem beim Aufruf des Roboterprozeßsystems die Option RobViaSimulator verwendet wird, also `rob_start(RobViaSimulator)`. Es ist erforderlich zuvor PVM und den Simulator selbst zu starten. DRI haengt sich dann in das laufende PVM ein.

4.5 Funktionsvorrat

Zur Programmierung der Schnittstelle stehen Funktionen zur Verfügung, die die komplette Funktionalität der mobilen Fahrzeuge erschließen, jedoch noch nicht auf die spezifischen Eigenheiten der Kommandos, wie Fahrbefehle, Ultraschallsensorik, Odometrie, usw. Rücksicht nehmen. Um auch von der MasPar aus direkt Fahrbefehle absetzen zu können, ist die vorliegende Implementierung in C gehalten. Weitergehende Implementierungen werden im objektorientierten Sinne Klassen bereitstellen, die einen Bezug zu gewissen Befehlsgruppen herstellen. Die Robotersteuerungskommandos reichen den Befehl als ASCII - Zeichenkette zum mobilen Fahrzeug durch. Für eine ausführliche Dokumentation der bereitgestellten Funktionen sei auf die man - pages verwiesen. Im folgenden werden die Befehle stichpunktartig aufgeführt. Als Referenz dienen die man-pages, die beim Ausckecken automatisch verfuegbar sind.

4.5.1 Systembefehle

```
int rob_start(int ttyflag); /* startet das Prozeßsystem */
int rob_halt(); /* hält das Prozeßsystem an */
int rob_join();  /* klinkt den Prozess in ein laufendes System ein */
int rob_leave(); /* klinkt den Prozess aus einem laufenden System aus */
```

4.5.2 TTY-Befehle

```
int rob_init(char* robname); /* Öffnet die Verbindung zum Fahrzeug */
int rob_exit(int robtid); /* Schließt die Verbindung zum Fahrzeug */
```

4.5.3 Programmentwicklungsbefehle

Der Programmier hat für Zwecke der Programmentwicklung die Möglichkeit die TTY-Ausgabe abzuklemmen und / oder den aktuell abgesetzten Befehl textuell angezeigt zu bekommen (im File /tmp/pvml.xxxxx):

```
/* Folgende Routine erlaubt die textuelle Ausgabe des zum Fahrzeug gesendeten
Kommandos und / oder das Abklemmen der TTY */
int rob_setopt(int what, int val);

/* Folgende Routine liest die gewünschte Option aus */
```

```
int rob_getopt(int what);
```

4.5.4 Robotersteuerungsbefehle:

1. `rob_command(int robtid,char* command, char* reply)` - synchrones Absetzen eines Robotersteuerungsbefehls und Abwarten der Antwort.
2. `rob_send_command(int robtid, char* command)` - Absetzen eines Robotersteuerungsbefehls, wobei auf eine Antwort des Roboterserverprozesses nicht gewartet wird.
3. `rob_get_reply(int robtid, char* reply)` - Auslesen der Antwort des Roboters zum letzten abgesetzten Robotersteuerungsbefeh.
4. `rob_send_command_confirm(int robtid, char* command)` - Absetzen eines Robotersteuerungsbefehls, wobei auf ein Echo des Roboterserverprozesses gewartet wird. Die Antwort des Roboters wird jedoch nicht abgewartet.
5. `rob_send_command_buffer(int robtid, char* command)` - Absetzen eines Robotersteuerungsbefehls, wobei auf das Echo des Roboterserverprozesses auf den vorigen `rob_send_command_buffer(..)` gewartet wird.
6. `rob_send_command_clearbuffer(int robtid)` - Dieser Befehl ist erforderlich um nach dem Absetzen einer Sequenz von `rob_send_command_buffer(..)` - Befehlen das vom Roboterserverprozeß gesendete letze Echo korrekt zu verarbeiten.

4.5.5 Ausdrucken von Nachrichten

Das Ausdrucken von Nachrichten erfolgt über einheitliche Funktionen. Die Verwendung von printf(...) sollte vermieden werden, um den Nachrichtenfluß auch im Hinblick auf spätere Erweiterungen über die folgenden Funktionen zu kanalisieren:

```
void rob_print(char* message);
void rob_error(char* message);
```

4.6 Deklarationen und Fehlercodes

4.6.1 Deklarationen und Fehlercodes

Folgende Deklarationen und Fehlercodes wurden bislang im include-File vorgesehen und sollten in Anwendungsprogrammen verwendet werden. Eine auf jeden Fall gültige Auflistung kann dem File `DRI/include/rob.h` entnommen werden:

Programm 4.3: DRI - Deklarationen und Fehlercodes

```
   1  /* general constants */
   2  #define ROBOCMDLENGTH      256
   3  #define ROBOMSGLENGTH      256
   4  #define MAXEXECLENGTH      20
   5  #define MAXHOSTNAMELENGTH  256
   6
   7  /* general tty behavior */
   8
   9  #define Activated 1
  10  #define NotActivated 0
  11
  12  #define RobDebugDefault    Activated    /* no debugging */
  13  #define RobTTYDefault      NotActivated /* tty I-O */
  14
  15  /* for rob_pvm_start */
  16
  17  #define RobViaRS232        402  /* all rob_commands are routed
to RS232 immediaely*/
  18
  19  #define RobViaServer       401  /* all rob_commands are routed
to a robotserver*/
  20
  21  #define RobViaSimulator 403 /* all commands are routed to simu
lator
  22
  23  /* for rob_setopt and rob_getopt */
  24
  25  #define RobToTid 305 /* set options */
  26  #define RobGetOpt          304  /* set options */
  27  #define RobSetOpt          303  /* set options */
  28  #define RobDebug           302  /* debug option */
  29  #define RobTTY             301  /* tty option */
  30
  31  /* librob error codes */
  32
  33  #define RobOk              0    /* okay */
  34  #define RobPvmFail         -1   /* pvm not started / stopped */
  35  #define RobAdmFail         -2   /* administration process not
started / not cleaned up
correctly */
  36  #define RobEnvFail         -3   /* environment variables for robot
servers missng */
  37  #define RobFileMissing     -4   /* environment variables for robot
servers missng */
  38  #define RobHostMismatch    -5   /* direkt tty only: actual host
does not match erver
host */
  39  #define RobCtrlFail         -6  /* control process not started */
  40  #define RobNoRobot         -7   /* no such robot name */
  41  #define RobOptFail         -8   /* no such option */
  42  #define RobNoServers       -9   /* no server processes running */
  43
```

4.7 Ausblick

Die vorgestellte Schnittstelle steht unter fortdauernder Entwicklung. Soweit möglich wird eine Aufwärtskompatibilität angestrebt. In zukünfigten Releases sollen folgende Punkte eingearbeitet werden:

- Beseitigung aufgetretener Bugs.
- Schedulingalgorithmus zur Verwaltung mehrerer auf denselben Roboterserverprozeß zugreifender Prozesse.
- C++ - Aufsatz für Bewegungsbefehle der MOTV-Gruppe, Ultraschall, Odometrie
- Mit der nächsten PVM3.4.x Release wird XPVM hoffentlich fehlerfrei funktionieren. Mit XPVM steht dann eine graphisch ansprechende Monitoringumgebung zur Verfügung.

4.8 Verwendung der Sourcen

Teilprojektname: `DRI (Distributed Robots Interface)`
aktuelle Version: `DRI2-0`

Library: `libDRI.a`
Beschreibung: Routinen zum Ansprechen der realen Roboter
Architekturen: `SUN4 SUN4SOL2 SUNMP`
Includes: `#include "DRI/rob.h"`
Linkoptionen: `-lDRI`

Programme: `test1, test2, test3, test4`
Beschreibung: Programme für die Library-Wartung
Architekturen: `SUN4 SUNMP MASPAR`

4.9 Literatur

[1] Al Geist et.al. *PVM 3.0 User's Guide and Reference Manual*, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1993

Kapitel 5

6-D-Maus

Matthias Muscholl

Die Space Mouse™ ist eine Eingabeeinheit, die die Steuerung von graphischen Objekten oder die Positionierung von Effektoren in 6 Freiheitsgraden erlaubt. Sie versetzt den Benutzer in die Lage, Objekte in den drei translatorischen und den drei rotatorischen Bewegungsrichtungen mit einem Handgriff zu führen.

5.1 Pinbelegung und Adapterkabel

Die Space Mouse wird mit einem 9 polige Stecker geliefert, die an die RS232 Schnittstelle des Rechners angeschlossen wird. Die Pinbelegung ist wie folgt:

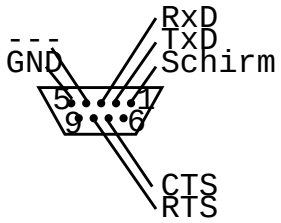


Abbildung 5.1: Space Mouse Pinbelegung des 9-poligen RS232-Steckers

SUN bietet an einer RS232-Buchse(A/B) eine Verschaltung von zwei RS232-Schnittstellen. Die Pinbelegung sieht für A wie folgt aus:

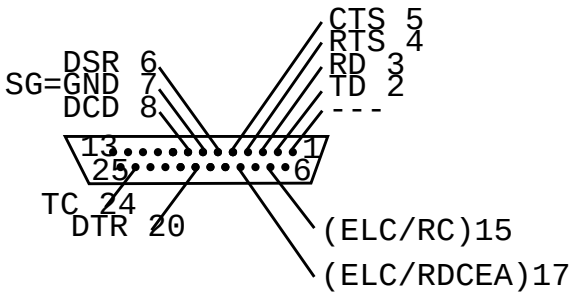


Abbildung 5.2: SUN Pinbelegung der 25-poligen RS232-Buchse für ttyA

Für den Anschluß an eine RS232-Schnittstelle(A) von SUN ergibt sich die folgende Verschaltung:

Pinbelegung Space Mouse	Pinbelegung SUN ttyA	Farbcodierung
1 Schirm	---	
2 TxD	3 RD	braun
3 RxD	2 TD	rot
5 GND	7 SG	blau
7 CTS	4 RTS	gelb
8 RTS	5 CTS	grün

Tabelle 5.1: Verkabelung für den Anschluß an SUN ttyA

5.2 Koordinaten und Einstellungen der Space Mouse

Die Space Mouse war ursprünglich für die graphische Steuerungen von 3-D-Applikationen gedacht. Daher ist das Koordinatensystem der Maus der des Bildschirmes angepaßt[1]. Für unsere Anwendung definieren wir die Koordinaten entsprechend den üblichen Weltkoordinaten um, so wie sie in Abb. 5.3 dargestellt sind.

5.2.1 Das Koordinatensystem

Das Koordinatensystem ist rechtwinklig, die x-Achse zeigt nach rechts, die y-Achse nach hinten und die z-Achse nach oben. Die Rotation um die x-Achse bezeichnen wir mit a, die um die y-Achse mit b und die um die z-Achse mit c.

1. Die Achsen sind wie folgt definiert: x-Achse nach rechts, y-Achse nach oben, z-Achse nach vorne.

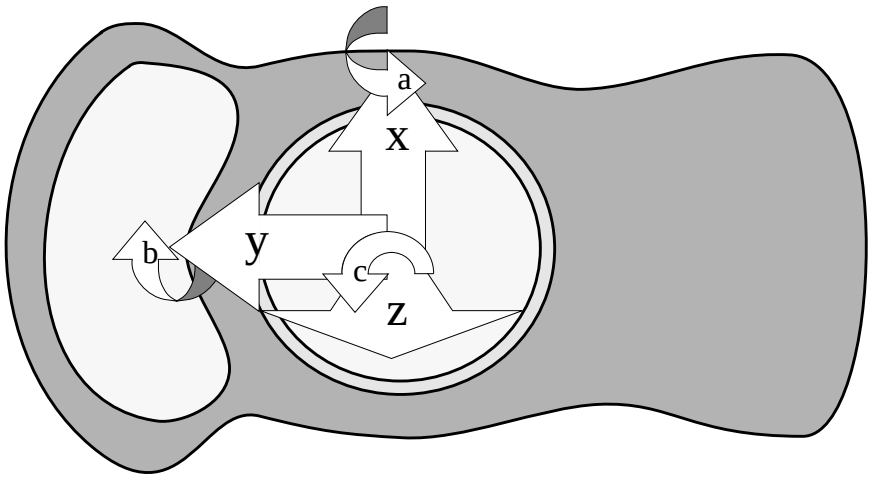


Abbildung 5.3: Koordinatensystem der Space Mouse

5.2.2 Die Steuerparameter

Die Space Mouse hat folgende Einstellmöglichkeiten, die fast vollständig auch über das Tastaturfeld der Maus (drücken zweier Tasten gleichzeitig) vorgenommen werden können:

Abkürzung	Erklärung	Tasten
Trans	Setzt alle translatorischen Komponenten auf 0. [True, False]	* 1
Rot	Setzt alle rotatorischen Komponenten auf 0. [True, False]	* 2
Dom	Setzt alle Komponenten auf 0, bis auf die betragsmäßig größte [True, False]	* 3
Zeroing	Eicht den Nullpunkt der Space Mouse auf die augenblickliche Auslenkung	* 4
Sens Trans	Stellt die Empfindlichkeit der translatorische Auslenkung ein. Jeder Tastendruck zählt als Inkrement. [0 … 15]	* 5
Sens Rot	Stellt die Empfindlichkeit der rotatorische Auslenkung ein. Jeder Tastendruck zählt als Inkrement. [0 … 15]	* 6
Nullradius	Stellt Schwellwert, ab der eine Auslenkung wahrgenommen wird. Jeder Tastendruck zählt als Inkrement. [0 … 15]	* 7
Set Default	Die Empfindlichkeit der translatorischen und rotatorischen Komponenten wird auf 0 gestellt. Der Schwellwert wird auf den Wert 8 gesetzt.	* 8

Tabelle 5.2: Kontrollparameter der Space Mouse

Abkürzung	Erklärung	Tasten
Data Rate	Einstellen der maximalen und minimalen Periodenzeit (siehe 5.2.3). {60, 80, …, 320} [ms]	
Beep	Das interne Pietzoelement kann für bestimmte Zeitspannen ertönen. {32, 64, 125, 250, 500, 1000, 1500, 2000} [ms]	

Tabelle 5.2: Kontrollparameter der Space Mouse

5.2.3 Das Kommunikationskonzept

Die Space Mouse schickt asynchron Datenpakete an den Rechner, die folgende Informationen transportieren: Auslenkungen der Kappe in den 6 Dimensionen, Drücken einer Kombination von Tasten, Loslassen einer Kombination von Tasten, Rückmeldung von vom Benutzer vorgenommenen Änderungen der Steuerparameter bzw. Fehlermeldung bei unbekannten Kommandos.

Datenpakete werden dann übersendet, wenn

1. wenn Auslenkungswerte ungleich 0 und
   a. wenn die maximale Periodenzeit überschritten ist, nach der spontan ein Datenpaket gesendet wird, oder
   b. sobald die minimale Periodenzeit schon verstrichen ist und Datenpakete angefordert wurden (Pollen).
2. oder der Auslenkungswerte ist null, aber vorher sind nur Datenpakete mit Werten ungleich 0 übersendet worden.

Mithilfe der maximalen und minimalen Periodenzeit kann die Granularität der zeitlichen Abtastung eingestellt werden:

1. feine Granularität
   $P_{min} = P_{max} = 60$ ms
2. grobe Granularität
   $P_{min} = P_{max} = 320$ ms
3. grobe Granularität mit zwischenzeitig feinerer Granularität beim Pollen
   $P_{min} = 60$ ms, $P_{max} = 320$ ms

5.3 Schnittstelle zum Anwendungsprogramm

5.3.1 Verbindungsaufbau und -abbau zur Space Mouse

Das Initialisieren der RS232-Schnittstelle und die Konfiguration der Space Mouse übernimmt die Funktion smConfig() die den Deskriptor auf das TTY liefert. . Die Funktion öffnet das TTY, konfiguriert die RS232-Schnittstelle und setzt die Space Mouse auf die in Programm 5.1.angegebenen Werte. Der Rückgabeparameter ist der Deskriptor auf das TTY.

Programm 5.1: Prototype der Initialisierungsfunktion smConfig()

```
int smConfig(void)
   1  { …
   2  td = open("/dev/ttya", O_RDWR);
   3  …
   4  smCntrl(td, smPeriodMaxMin, 60, 60);
   5  smCntrl(td, smSensityTransRot, 0, 0);
   6  smCntrl(td, smNullRadiusTo, 8);
   7  smCntrl(td, smZeroing);
   8  smCntrl(td, smBeepDuration, 64);
   9  smCntrl(td, smRotOnTransOn);
  10  return(td);
  11  }
```

Mit der Prozedur smClose() wird die Verbindung zur Space Mouse wieder beendet. Die SpaceMouse wird veranlaßt keine Werte mehr zu senden (siehe Programm 5.2).

Programm 5.2: Prototype der Abmeldeprozedur smClose()

```
void smClose(int td)
   1  {
   2    smCntrl(td, smRotOffTransOff);
   3    close(td);
   4  }
```

5.3.2 Steuerung der Space Mouse Funktionen

Die Space Mouse wird mit der in Programm 5.3 aufgeführten Funktion gesteuert. Der Rückgabeparameter ist -1, falls die aufgerufene Option nicht verfügbar ist. In Tabelle 5.3 sind die implementierten Optionen aufgeführt

Optionen	Parameter	Bedeutung
smZeroing		siehe Zeroing (Tabelle 5.2)
smRotOnTransOn		empfindlich auf Rot / Trans
smRotDomTransOff		empfindlich auf betragsgrößte Rot
smRotOffTransOn		empfindlich nur auf Trans
smRotOffTransDom		empfindlich auf betragsgrößte Trans
smRotDomTransDom		empfindlich auf betragsgrößte Rot / Trans
smRotOffTransOff		unempfindlich auf Rot / Trans
smBeepDuration	int duration	siehe Beep (Tabelle 5.2)
smPeriodMaxMin	int max, int min	siehe Data Rate (Tabelle 5.2)
smNullRadiusTo	int radius	siehe Nullradius (Tabelle 5.2)

Tabelle 5.3: Optionen der Stererfunktion smCntrl()

Optionen	Parameter	Bedeutung
smSensityTransRot	int trans, int rot	siehe Sens Trans / Sens Rot (Tabelle 5.2)

Tabelle 5.3: Optionen der Sterer funktion smCntrl()

Programm 5.3: Prototype der Steuerfunktion smCntrl()

```
int smCntrl(int td, int option, ...)
   1  { …
   2  }
```

5.3.3 Datenverkehr mit der Space Mouse

Die Werte der Space Mouse werden mit der Funktion smDataRequest() abgerufen.

Programm 5.4: Prototype der Prozedur smDataRequest()

```
void smDataRequest(int td, int polling, smDataConfirm *reply);
   1  {…
   2  }
```

Der erste Parameter übergibt den Deskriptor, der zweite bestimmt, ob anstatt alle Daten weiterzugeben nur die aktuelle Auslenkung der Kappe ausgelesen werden soll (polling $\in$ {TRUE, FALSE} siehe dazu 5.2.3) und der dritte Parameter ist der Rückgabeparameter. Der Rückgabeparameter ist vom Typ smDataConfirm (siehe Tabelle 5.4 und Programm 5.5).

smReplyType <var>.tag ==	Erklärung	Zugehörige Variantentupel
smKeyboard	Eine oder mehrere Tasten wurden gedrückt. k1-ks sind Bool-Werte, no gibt die Anzahl der gleichzeitig gedrückten Tasten wieder.	<var>.msg.button
smData	Die Kappe wurde bewegt. Das Koordinatensystem entspricht dem der Abb. 5.3	<var>.msg.move

Tabelle 5.4: Aufzaehlungsvarianten des Typs smDataConfirm

smReplyType <var>.tag ==	Erklärung	Zugehörige Variantentupel
smUserChange-dConfiguration	Der Benutzer hat kritische Konfigurations-änderungen durch Drücken von Tasten-kombinationen (siehe auch smKeyboard) vorgenommen. Zu diesen zaehlen: a. Sens Trans b. Sens Rot c. Nullradius d. Set Default Ist dies unzulässig, so kann das Programm die Änderungen geeignet überschreiben.	
smCmdError	Ein Fehler in einem Kommando ist erkannt worden. In unknown wird zeichenweise das Kommando an das Programm übergeben.	<var>.msg
smFrameError	Es trat ein Datenübertragungsfehler auf.	

Tabelle 5.4: Aufzaehlungsvarianten des Typs smDataConfirm

5.3.4 Programmtemplate für die Verwendung der Space Mouse

Im eigenen Programm kann man den Code aus Programm 5.6 verwenden, um die Space Mouse abzufragen.

5.4 Verwendung der Sourcen

Teilprojektname: `SmI` (Space Mouse Interface)
aktuelle Version: `SmI2-0`
bei nutzenden TPR: Makefilevariable `BASESON` um die aktuelle Version erweitern

Library: `libSmI.a`
Beschreibung: Routinen zum Ansprechen der Space Mouse
Architekturen: `SUN4 SUN4SOL2 SUNMP`
Includes: `#include "SmI/SpaceMouseInterface.h"`
Linkoptionen: -lSmI

Programme: `mousetest`
Beschreibung: Gibt die von der Space Mouse übertragen Daten auf der Shell aus. Beendet wird es mit ctrl-c.
Architekturen: `SUN4 SUNMP`

Programm 5.5: Datentyp der von der Space Mouse verschickten Werte

```
typedef enum { smKeyboard, smData, smUserChangedConfiguration,
smCmdError, smFrameError } smReplyType;

typedef union {

  struct {           /* tag == smKeyboard */
    unsigned k1 : 1; /* Taste gedrueckt, dann k. == TRUE */
    unsigned k2 : 1;
    unsigned k3 : 1;
    unsigned k4 : 1;
    unsigned k5 : 1;
    unsigned k6 : 1;
    unsigned k7 : 1;
    unsigned k8 : 1;
    unsigned ks : 1;
    unsigned no : 4; /* Anzahl gedrueckter Tasten [0..9] */
} button;

  struct {           /* tag == smData */
    int x,y,z,a,b,c; /* enthaelt transl. und rotat. Werte */
  } move;
                     /* tag == smCmdError */
  char unknown;      /* unverstandenes Kommando (zeichenweise) */

} smReplyMessage;

typedef struct {
  smReplyType tag;
  smReplyMessage msg;
} smDataConfirm;
```

5.5 Literatur

[1] NN. *Space Mouse Software Interface Benutzerhandbuch*, Space Control Gesellschaft für 3D Systeme, 82216 Malching, 1994

Programm 5.6: Programmtemplate

```
 1  #include “SpaceMouseInterface.h”
 2  …
 3  … funct(…)
 4  {
 5     int td; /* Deskriptor der Schnittstelle zur Space Mouse */
 6     smDataConfirm mouse;
 7  …
 8     td = smConfig();
 9     …
10     while (!<Ende Bedingung>) {
11        smDataRequest(td, FALSE, &mouse);
12        switch(mouse.tag) {
13        case smKeyboard: …; break;
14        case smData:     …; break;
15        case smCmdError: …; break;
16        case smFrameError: …; break;
17        default:          …; break;
18        }
19     }
20     …
21     smClose(td);
22  }
```

Kapitel 6

Bildformate

Michael Vogt, Harald Bayer, Susanne Gerl

Für die im Roboterprojekt anfallenden Bilder und Verarbeitungsroutinen sollen regelmäßig die gleichen internen Formate verwendet werden, um eine möglichst hohe Wiederverwendbarkeit der einzelnen Programme zu garantieren. Zur Diskussion stehen mehrere Formate, die zum Teil durch die Hardware (Sun, MasPar, Eltec) und zum anderen Teil durch vorhandene Software (Horus, Khoros, pbmplus) in Betracht kommen. Nachfolgend erfolgt eine Zusammenstellung der bisher vorhandenen Formate und eine Empfehlung für ein integriertes Format, das möglichst vielen Anforderungen gerecht wird.

6.1 Hardware Formate

6.1.1 Sun XIL Framegrabber

Der Sun Framegrabber liefert in Zusammenarbeit mit der XIL Library die Bilder in Form einer speziellen Speicherstruktur mit Namen XilMemoryStorage. Der Aufbau ist in Programm 6.1 verdeutlicht.

Wie man erkennen kann, handelt es sich um eine Union, die vier verschiedene Fälle (also Bildformate) abdeckt. Leider enthält der Datentyp keinen Hinweis darauf, welches Format tatsächlich vorliegt. Der einzige Hinweis hierauf ergibt sich aus der Initialisierung des Framegrabbers und der Aufrufreihenfolge und Definition verschiedener Zwischenbilder, die der Endbenutzer jedoch nicht sieht. Die bereits vorhandenen Routinen zum Lesen eines Bildes (schwarz weiß oder farbig) verwenden beide das Byte Format, welches in Programm 6.2 definiert ist.

Programm 6.1: Definition von XilMemoryStorage

```
1  typedef union __XilMemoryStorage {
2     XilMemoryStorageBit   bit;
3     XilMemoryStorageByte  byte;
4     XilMemoryStorageShort shrt;
5     XilMemoryStorageFloat flt;
6  } XilMemoryStorage;
```

Programm 6.2: Definition von XilMemoryStorageByte

```
1  typedef struct __XilMemoryStorageByte {
2     Xil_unsigned8* data;
3  /* pointer to the first byte of the image */
4     unsigned long scanline_stride;
5  /* the number of bytes between scanlines */
6     unsigned int pixel_stride;
7  /* the number of bytes between pixels */
8  } XilMemoryStorageByte;
```

Die Breite und Höhe des Bildes geht nicht aus dem Format hervor, sondern wird bei der Initialisierung des Framegrabbers festgelegt und muß an anderer Stelle gespeichert werden.

Nach unserer bisherigen Erfahrung liegen die Bilddaten folgendermaßen vor:

1. Grauwertbilder:
   Xil_unsigned8* ist ein Zeiger auf einen Speicherbereich mit #Zeilen mal #Spalten Bytes. Jedes Byte enthält einen Grauwert zwischen 0 (schwarz) und 255 (weiß). Die Anordnung ist Zeilenweise von oben nach unten und innerhalb der Zeilen von links nach rechts. Das Feld scanline_stride gibt die Anzahl der Bytes pro Zeile an, steht also de facto für die Bildbreite. Der Zugriff auf ein Pixel an der Position (x,y) kann z.B. erfolgen über folgenden Pseudocode:

Programm 6.3: Zugriff auf XIL Grauwertbilder

```
1  grey = data[x+y*scanline_stride]
```

2. Farbbilder:
   Wie bei schwarz weiß Bildern gibt Xil_unsigned8* einen Zeiger auf den Bildbereich an. Dieser Bereich enthält #Zeilen mal #Spalten mal drei Bytes. Die zeilenweise Anordnung ist analog. Hier werden jedoch für jedes Pixel drei aufeinanderfolgende Bytes belegt, und zwar in der Reihenfolge blau, grün, rot. Der Zugriff auf die RGB Werte eines Pixels (x,y) kann z.B. erfolgen durch:

Programm 6.4: Zugriff auf XIL Farbbilder

```
1  rot = data[3*(x+y*scanline_stride)+2]
2  gruen= data[3*(x+y*scanline_stride)+1]
3  blau= data[3*(x+y*scanline_stride)]
```

Der Framegrabber von Sun bietet außer den Bildern noch weitere Information an. Z.B. ist es möglich, eine Bildnummer und einen Timestamp auszulesen.

Weitere Informationen über XIL Bilder und die Benutzung des Sun Framegrabbers befinden sich im Kapitel über den Sun Framegrabber. Hier ist die C Datenstruktur SFG_image definiert, die XIL Bilder als Grundlage enthält.

6.1.2 Eltec Kantenfinder

Der Eltec Kantenfinder liefert keine Bilder sondern bereits extrahierte Kanten. Grundsätzlich steht als Information ein „Kantenbild“ zur Verfügung, wobei folgende Definitionen gelten:

- Eine (gerade) Kante ist definiert durch ihren Startpunkt (x,y Position) eine Richtungsangabe (Winkel) und eine Länge.
- Eine Kontur ist eine Liste von (geraden) Kanten, die miteinander verkettet sind.
- Ein Kantenbild ist eine Liste von Konturen

Die genaue Kodierung sowie die Zugriffsmethoden sind Implementierungsabhängig und liegen noch nicht endgültig fest. Es besteht theoretisch die Möglichkeit, die Kantenbilder in einem Format darzustellen, das mit einem Horus oder Khoros Format identisch ist.

Marco Sommerau ist hier der richtige Ansprechpartner.

6.2 Software Formate

6.2.1 Horus

Einführung

HORUS ist ein Bildverarbeitungstool, welches an der Technischen Universität München entwickelt wurde [1], [2]. Es stellt mehr als 600 Bildverarbeitungsroutinen zur Verfügung, die in C oder C++ Programmen eingebunden werden können. Diese ermöglichen eine einfache und schnelle Implementierung komplexer Routinen zur Bildauswertung, die sowohl Aufgaben in der Low-Level-Bildverarbeitung, als auch in der höheren Bildverarbeitung lösen.

In HORUS wurde für Regionen und Bilder der Überbegriff Bildobjekt (`ObjType`) eingeführt. Eine Region besteht aus einer Menge von Koordinaten in der Bildebene. Eine solche Region muß durchaus nicht zusammenhängend sein und kann ohne weiteres auch Löcher enthalten. Regionen können auch größer als das aktuelle Bildformat sein. Intern werden Regionen durch Lauflängenkodierung realisiert.

Bilder bestehen aus mindestens einer Bildmatrix zusammen mit einer Region, die angibt, an welchen Punkten die Matrix definierte Werte enthält. Außerdem unterstützt HORUS mehrkanalige Bilder. Jedes Bild kann bis zu m Kanäle enthalten, wobei die Zahl n bei der Initialisierung des Systems mit `init_horus(...,n)` festgelegt wird. Zu einer Bildkoordinate existiert hier also nicht nur ein Grauwert, sondern ein ganzer Vektor von bis zu n Grauwerten. (Sofern die entsprechenden Bildpunkte zum Defini-

tionsbereich des Bildes gehört). Anschaulich könnte man vielleicht auch von einem Stapel von Bildern sprechen anstelle eines Einzelbildes. Damit lassen sich zum Beispiel RGB Bilder oder Voxelbilder darstellen.

HORUS / C stellt für Bildobjekte (Bilder + Regionen) den Datentype `ObjType` zur Verfügung. Dahinter verbirgt sich ein Surrogat der HORUS Datenbank, in der die Bildobjekte abgelegt sind. Eingabebildobjekte werden per value an die HORUS Prozeduren übergeben und Ausgabe-Bildobjekte mittels des &-Operators per reference. Variablen dieses Typus können sowohl ein einzelnes Bildobjekt, als auch ganze Tupel von Bildobjekten enthalten. Ein Einzelobjekt wird hierbei wie ein Tupel der Länge eins behandelt.

HORUS wurde bereits so modifiziert, daß man von HORUS aus die Framegrabber auf den Robosuns öffnen, Bilder grabben und schließen kann.

Momentan wird gerade an der Schnittstelle ELTEC / HORUS gearbeitet.

Im Folgenden werden einige wichtige Eigenschaften von HORUS aufgeführt, die wesentlich ausführlicher auch in [1] und [2] beschrieben sind.

Aufbau eines Bildes

- Jedes Bild besteht aus:
  1. Einer oder mehreren Bildmatrizen
  2. Definitionsbereich
- Man unterscheidet zwischen Bild und Bildmatrix in HORUS:
  - Die Matrix ist ein Baustein für ein Bild, es speichert die Grauwerte
  - Aus mehreren Matrizen wird ein mehrkanaliges Bild aufgebaut.
  - Der Definitionsbereich schränkt die gültigen Koordinaten der Matrix ein.

Der Definitionsbereich eines Bildes

- Alle Grauwertoperationen werden nur im Definitionsbereich des Bildes ausgeführt
- (z.B. Filter- oder Segmentierungsoperationen)
- Der Definitionsbereich eines Bildes ist als eine Region realisiert
- Der Definitionsbereich ist nie größer als die Matrix; die Form ist beliebig.
- Der Definitionsbereich kann i.a. nur verkleinert werden (z.B `reduce_domain`).
- Maximalen Definitionsbereich durch full_domain.

Das Bildformat

- Der Ursprung eines Bildes ist immer der Punkt (0,0), und liegt somit “links oben”.
- Die x-Koordinate (`column`) läuft vom Ursprung mit wachsenden Werten nach rechts bis zum Wert Bildbreite-1.
- Die y-Koordinate (`row`) läuft vom Ursprung mit wachsenden Werten nach unten bis zum Wert Bildhöhe-1.

- Pixel können nie negative Koordinaten haben
- Bilder können nur rechteckig sein.
- Die maximale Bildkantenlänge ist 10.000.
- Jedes Bild hat ein eigenes Bildformat
- Bilder mit unterschiedlichen Format können nicht gleichzeitig bearbeitet werden (z.B. `add__`, `dyn_threshold__`).
- Das Format eines Ergebnisbildes kann sich von der des Eingabebildes unterscheiden (z.B. `image_transform__`, `zoom_image1`).
- Bildformate können mit den Prozeduren `crop_image` und `change_format` direkt modifiziert werden.
- Eine Modifikation des Definitionsbereichs hat keinen Einfluß auf das Bildformat (z.B `reduce_domain`)

Die Pixeltypen

- `byte`: 0...255, typisches Graubild.
- `int1`: -127...127, Byte mit Vorzeichen.
- int2: -32767...32767, z.B. das Ergebnis einer Konvolution.
- `int4`: -2147483647... 2147483647, z.B. 2-dimensionale Histogramme
- `real`: Gleitpunktzahl mit 4 Byte
- `complex`: Komplexe Zahl; jeder Punkt besteht aus zwei Gleitpunktzahlen vom Typ real für Real- und Imaginärteil (z.B. Ergebnis der FFT).
- `dvf`: Verschiebungsvektorfeld; jeder Punkt beschreibt einen Vektor (x,y); Darstellung durch zwei Werte vom Typ int1
- `cyclic`: 0...255, wobei 255+1=0 ist; z.B für die Darstellung des Farbwertes im hsi-Farbmodell (trans_from_rgb).
- `direction`: 0...180 Darstellung eines Winkels/2 (z.B. `sobel_dir` und `edges__`).
- Alle Pixel einer Bildmatrix sind vom gleichen Typ
- Bei mehrkanaligen Bildern können die einzelnen Matrizen von unterschiedlichen Typen sein.
- Der Pixeltyp kann z.B. durch Filter verändert werden.
- Eine Typanpassung erfolgt durch die Prozeduren `convert_image_type`, `dvf_to_int` oder `complex_to_float`.

Das mehrkanalige Bildformat

- Mehrkanalige Bilder müssen ein einheitliches Format haben (d.h. alle Bildmatrizen des Bildes haben die gleiche Kantenlänge)

Die Regionen

- Eine Region ist eine beliebige Menge von Koordinatenpunkten.
- Eine Region muß nicht zusammenhängend sein.
- Eine Variable (bzw. ein Parameter) kann mehrere Regionen enthalten.
- Die Regionen einer Variablen können sich überlappen
- Der Wertebereich von Punkten einer Region ist auf -32767...32767 beschränkt.
- Regionen werden durch Lauflängenkodierung realisiert.

Zugriff auf Regionen

- Punkte der Regionen (`fetch_coord`, `fetch_chord` etc.)
- Rand (`fetch_contour`, `fetch_polygon` etc.)
- Einzelne Punkte (`inside_region2`)

Clipping von Regionen

- Eine Region ist unabhängig vom Bildformat.
- Regionen können negative Koordinaten enthalten.
- Das Systemflag `clip_region` beschneidet Regionen auf das (aktuelle maximale) Bildformat.

6.2.2 Khoros

Khoros 1.x

Hier gab es ein Format, das jetzt Xvimage genannt wird, das das global gültige Format war. Bilder anderer Formate mußten über mitgelieferte Routinen konvertiert werden. Das Xvimage-Format war dokumentiert.

Khoros 2.0

Eine endgültige Beschreibung läßt sich noch nicht geben, da noch niemand sich die Zeit genommen hat, die sehr umfangreichen Handbücher zu lesen. Was man in Ankündigungen zwischen den Zeilen fand, interpretiere ich momentan so:

Jedes Khoros2.0-Programm kann über die Library-Routinen einer Toolbox (Khoros2.0 besteht nur aus Toolboxen, die verschiedenen Zwecken dienen) Files jedes der unterstützten Formate lesen und schreiben. Unterstützt werden momentan: ASCII, AVS, PNM, (EPS,) Raw, Sun Raster, VIFF, XBM, Xvimage, XWD und XPM. Dadurch entfällt die Notwendigkeit expliziter Konverter. Aber Khoros2.0 geht noch weiter: Es soll sogar die Notwendigkeit entfallen, an die Konversion denken zu müssen. Das geht so weit, daß weder gesagt wird, ob jedes Bild beim Lesen konvertiert wird oder intern mit allen Formaten gearbeitet werden kann (einige Postings lassen sich durchaus so verstehen, daß es sich bei den Bildverarbeitungsroutinen um einen Code auf höherer Ebene handelt, der vom Format unabhängig ist), noch wird das VIFF-Format beschrieben, da keine Notwendigkeit mehr bestünde, direkt mit diesem Format umzugehen. (HFB)

6.2.3 pbmplus Format

Sehr verbreitet ist das pbmplus Format. Für alle Architekturen gibt es das pbmplus Paket, das sowohl einfache Bildoperationen aber hauptsächlich Konvertierungsfunktionen beinhaltet. Das Dateiformat für pbmplus Bilder ist in der ASCII Form rechnerunabhängig. Das schnellere und kleinere binäre Format ist hingegen immer an eine bestimmte Architektur (Sun, Dec, ...) gebunden, wobei keine Möglichkeit der Architekturprüfung besteht.

Das Speicherformat von pbmplus Bildern hängt davon ab, welches der drei möglichen Formate (Binärbilder, Grauwertbilder, Farbbilder) vorliegt. Hier werden nur Grauwert- und Farbbilder vorgestellt:

1. Grauwertbilder (pgm)
   Grauwertbilder sind als zweidimensionale C Felder gespeichert (eine andere Beschreibung ist: Es sind eindimensionale Felder von Zeigern auf eindimensionale Felder von Pixelwerten). Die C Deklaration eines pgm Bildes lautet:

Programm 6.5: #include für pgm-Bilder

```
1  #include <pgm.h>
2  gray **bild;
```

   Die Größe des Feldes, also die Bildgröße, ist im Format nicht enthalten und muß wiederum beim Laden bzw. Erzeugen des Bildes festgelegt und gespeichert werden. Die Elemente des Feldes sind vom Typ gray, der i.allg. einem unsigned char entspricht und einen Wertebereich von 0 (schwarz) bis 255 (weiß) bzw. maxgray hat. Der Zugriff auf ein Pixel (x,y) erfolgt einfach über eine doppelte Indizierung:

Programm 6.6: Zugriff auf pgm-Pixel

```
1  grey = bild[y][x];
```

2. Farbbilder (ppm)
   Farbbilder sind ebenso wie Grauwertbilder als zweidimensionale Felder gespeichert. Der Basistyp ist hier jedoch „pixel“, was eine Struktur aus drei Werten vom Typ „pixval“ für RGB ist. Die entsprechende C Deklaration für ein Farbbild lautet:

Programm 6.7: #include für ppm-Bilder

```
1  #include <ppm.h>
2  pixel **bild;
```

   Der Wertebereich der RGB Werte liegt i.allg. ebenfalls zwischen 0 und 255 (maxval). Der Zugriff auf die einzelnen Farbkomponenten erfolgt durch vorhandene Zugriffsmacros für das Lesen und Schreiben von Pixelwerten (siehe Programm 6.8).

Programm 6.8: Zugriff auf ppm-Pixel

```
1  pixval rot,gruen,blau;
2  /* Lesen: */
3  rot = PPM_GETR(bild[y][x]);
4  gruen= PPM_GETG(bild[y][x]);
5  blau= PPM_GETB(bild[y][x]);
6  /* Schreiben: */
7  PPM_ASSIGN(bild[y][x], rot, gruen, blau);
```

Weiter Informationen zu den pbmplus Formaten und zu vorhanden Routinen zum Lesen und Schreiben und Allocieren von Bildern sind leicht den entsprechenden man pages zu entnehmen (libpgm, libppm).

6.3 Weitere Gesichtspunkte

Für eine möglichst zukunftssichere Architektur wird es nötig sein, weitere Gesichtspunkte in Betracht zu ziehen. Wesentliche Punkte sind z.B.:

1. Timestamp
   Um eine Fusion mit anderen Sensorsignalen zu erreichen ist es notwendig, jedes Bild mit einer Zeitmarke zu versehen, die den Zeitpunkt seiner Aufnahme angibt.
2. Stereo Bildpaare
   Durch den Stereomischer für den Stereokopf werden zwei getrennte Bilder in Halbbilder eines einzigen Bildes zusammengefaßt. Ein effizienter Zugriff kann durch „geschickte“ Indizierung in die Bilddaten erfolgen, ohne die Bilder vorher zu trennen. Vorhanden Routinen aus pbmplus, Horus oder Khoros können aber nicht direkt angewendet werden, ohne die Bilder vorher zu trennen.

6.4 Empfehlung für ein allgemeines Bildformat

Das pbmplus Format ist sehr leicht zu benutzen und weit verbreitet. Das XIL Format ist andererseits ein Hardwareformat, das sehr schnellen Zugriff erlaubt. Keines der Formate enthält ausreichende Angaben über Bildgröße oder Zeitpunkt der Aufnahme. Auch Stereobildpaare werden nicht berücksichtigt. Es wird daher ein Format vorgeschlagen, das möglichst vielen Anforderungen gerecht wird (siehe Programm 6.9).

Die Bedeutung der Felder ergibt sich aus den Kommentaren. Durch den Inhalt des Feldes „type“ wird die Gültigkeit und Form des Zugriffs auf die Bilddaten festgelegt.

Eine Diskussion hierüber ist notwendig. Eventuell könnten man auch die Unterscheidung zwischen Mono- und Stereobildern durch eine Union erzielen. Außerdem könnten es sinnvoll sein, ein Stereobild basierend auf Halbbildern zu definieren, um der ursprünglich vorliegenden Form gerecht zu werden.

Programm 6.9: Vorschlag für ein allgemeines Bildformat

```
 1  typedef enum __image_type
 2      { MONO_GRAY, STEREO_GRAY, MONO_COLOR, STEREO_COLOR } image_type;
 3
 4  struct image
 5  {
 6      image_type type;      /* type of image */
 7      int x;                /* width of the picture */
 8      int y;                /* height of the picture */
 9      struct timeval *tp;   /* timestamp */
10      union
11      {
12          gray **gray_data;    /* gray scale data */
13          pixel **color_data;  /* color data */
14      } mono_left;
15      union
16      {
17          gray **gray_data;    /* gray scale data */
18          pixel **color_data;  /* color data */
19      } right;
20  };
```

6.5 Literatur

[1] Wolfgang Eckstein: *Horus-Referenzmanual*, Technische Universität München, Institut für Informatik, 1995.

[2] Wolfgang Eckstein: HORUS/C* Benutzerhandbuch, Technische Universität München, Institut für Informatik, 1994.

[3] Manual zum PNM System, online Dokumentation, Einstieg über `man libpnm` oder `man pnm`

Kapitel 7

Benutzung des Maspar Framegrabbers

Thilo Will

In der Maspar befindet sich ein Framegrabber. Diesen kann man mit Funktionen in libcfg-GetFrame nutzen. D.h. man kann schwarz-weiss Bilder vom Framegrabber auf das DPU transportieren. Es werden Bildgrössen deren Kantenlaengen ein vielfaches von 128 sind, unterstützt. Die Kanten duerfen nicht grösser als 512 sein. Im folgenden wird zunächst besprochen, in welcher Weise die Bilddaten dabei auf der DPU abgelegt werden, danach wird auf die Funktionen eingegangen.

7.1 Virtualisierung der Bilddaten

Die Funktionen aus libcfg1-2GetFrame dienen dazu Bilder aus dem Framegrabber auf der DPU abzulegen. Wie werden nun die Daten eines Bildes auf die einzelnen PE's verteilt? Sei

```
unsigned char bild[nx][ny];
```

ein Bild, und

```
plural unsigned char *plural_bild;
```

ein Pointer auf das entsprechende Bild auf der DPU. Es gilt dann folgende Zuordnung:

```
bild[x][y] = iproc[x%nxproc][y%nyproc].plural_bild[x/nxproc + (nx/nxproc)*(y/nyproc]);
```

Dies bezeichnet man auch als *Two-Dimensional Cut-and-Stack Virtualization.*

Insbesondere folgt daraus für Bilder der Größe nxproc x nyproc:

```
bild[x][y]=iproc[x][y].plural_bild[0];
```

7.2 cfgInit

```
#include <GetFrame.h>
void cfgInit(void);
```

Diese Funktion initialisiert den Framegrabber. Sie muß vor dem ersten Aufruf von cfgGetFrame aufgerufen werden.

7.3 cfgGetFrame

```
#include <GetFrame.h>
void cfgGetFrame(
          plural unsigned char *image,
          int ix,
          int iy,
          int xoff,
          int yoff,
          int xsk,
          int ysk
     );
```

Diese Funktion transportiert ein Bild vom Framegrabber auf die DPU. Es kann dabei auch ein Auschnitt des Bildes im Buffer des Framegrabbers gelesen werden. Dieser Auschnitt kann beliebig positioniert werden. Ausserdem kann der Ausschitt, beveor er auf der DPU abgelegt wird, um ganzahlige Faktoren in Breite und Länge verkleinert werden.

`image` zeigt auf das plural Feld wo die Bilddaten hingeschrieben werden .

`ix, iy` geben an wie Groß das Bild auf der DPU sein soll. Es gilt

Breite = `ix * nxproc`

Hoehe = `iy * nyproc`

Die Parameter `ix, iy` duerfen die Werte 1...4 annehmen.

`xoff,yoff` bestimmen die Postion des Bildausschnittes des Bildes im Framebuffer, welcher auf die DPU kopiert werden soll. `xoff,yoff` sind dabei die Koordinaten des linken oberen Pixels des Bildausschnittes im Gesamtbild.

`xsk, ysk` sind die Verkeinerungsfaktoren zwischen dem Bild im Framebuffer und dem Bild auf der DPU in x bzw. y-Richtung.

7.4 cfgGetHalfFrame

```
#include <GetFrame.h>
void cfgGetHalfFrame(
          plural unsigned char *image_even,
          plural unsigned char *image_odd,
          int ix,
          int iy,
          int xoff_even,
          int xoff_odd,
          int yoff_even,
          int yoff_odd,
          int xsk,
          int ysk
      );
```

cfgGetHalfframe ermöglicht das Lesen von Bildern die mit dem Stereobildmischer zusammengemischt wurden. Der Stereobildmischer liefert als output ein Bild dessen gerade Zeilen aus dem Halbbild des einen Inputskanales bestehen, und die ungeraden aus einem Halbild des anderen. Mit cgfGetHalfFrame kann man diese Bilder entmischt lesen. Die Bedeutng der Parameter ist anolog der von cfgGetFrame, mit dem Unterschied:

- Es gibt jetzt zwei Zieladressen, `image_even` und `image_odd,` fuer das Bild in den geraden bzw. ungeraden Zeilen.
- Für die beiden Bilder kann man unterschiedliche offset Werte angeben.

7.5 Geschwindigkeit

Der Transport der Bilddaten vom Framegrabber zur DPU ist langsam. Die Geschwindigkeit, hängt stark von den Parametern der GetFrame Funktionen ab. Den grössten Einfluss haben iy und ysk. Die Parameter für Bildgrösse ix und Skalierung xsk in x-Richtung haben weniger Einfluss. Der offset in y-Richtung hat keinen Einfluß.

Kapitel 8

ELTEC-VectEx

Marco Sommerau

Das ELTEC-VectEx ist eine Spezialhardware zur Bildverarbeitung, die Konturen aus einem Grauwertbild in Echtzeit (Videofrequenz: 50 Hz) extrahiert und durch Polygonzüge approximiert (vektorisiert).

Die zum System gehörende Hardware besteht aus drei VME-Bus Karten, die in einem 19“-Gehäuse untergebracht sind. Die Karten können nur von robosun1 aus über einen SBus VME-Bus Adapter konfiguriert und ausgelesen werden.

Da die von der Hardware gelieferten Daten zur direkten Weiterverarbeitung ungeeignet sind, müssen diese erst durch einige Vorverarbeitungsschritte aufbereitet werden.

Die notwendige Software wurde unter Verwendung von Teilen der GNU C++ Bibliothek (libg++) komplett in C++ erstellt. Diese Dokumentation beschreibt die Struktur und Funktionsweise der Software in der Version **Elt2-1**.

Diese Software besteht zum einen aus einer Bibliothek zur komfortablen Bedienung der Hardware (**libElt_boards.a**) und zum anderen aus einer Bibliothek die losgelöst von der Hardware die für die Bilddaten notwendigen Vorverarbeitungsschritte bereitstellt und Hilfestellung für eine weitere Bildauswertung gibt (**libElt_misc.a**).

8.1 Konfiguration der Hardware

Als Eingabe erhält das VME-Subsystem ein analoges Videosignal. Dieses wird zuerst digitalisiert, dann über den ELTEC-spezifischen Video-Bus (VI-Bus) von Verarbeitungseinheit zu Verarbeitungseinheit weitergereicht, bis schließlich über VME-Bus die extrahierten Daten in Form von Vektorpunkten ausgelesen werden können. Jede der beteiligten drei Karten kann über VME-Bus konfiguriert werden.

8.1.1 SBus VME-Bus Adapter (PT-SBS915)

Um die Hardware von einer SPARCstation 10 aus ansprechen zu können, wurde ein SBus VME-Bus Adapter installiert [1]. Die Adapter-Karte steckt momentan auf dem SBus-Slot 2 der robosun1 im Roboter-Labor. Die VME-Bus Karten können über die Treiber-Dateien

- `/dev/ptvme/a16d16` für 16-Bit Adresse, 16-Bit Daten
- `/dev/ptvme/a16d32` für 16-Bit Adresse, 32-Bit Daten
- `/dev/ptvme/a24d16` für 24-Bit Adresse, 16-Bit Daten
- `/dev/ptvme/a24d32` für 24-Bit Adresse, 32-Bit Daten
- `/dev/ptvme/a32d16` für 32-Bit Adresse, 16-Bit Daten
- `/dev/ptvme/a32d32` für 32-Bit Adresse, 32-Bit Daten

direkt angesprochen werden. Es wird dabei der VME-Speicherbereich in den SUN-Speicher gemappt.

Diese Lösung ist leider nur für ein geringes Datenaufkommen wie etwa nach der Bildvorverarbeitung durch die Spezialhardware geeignet, da diese Schnittstelle relativ langsam ist.

8.1.2 Image Processing Port (IPP)

Die erste der drei Karten ist ein 8-Bit Graustufen Framegrabber, der das eingehende analoge Videosignal in ein Grauwertbild digitalisiert [2]. Die Position und Größe des zu digitalisierende Bildausschnitts innerhalb des Vollbildes kann dabei relativ frei gewählt werden. Zur Konfiguration des IPP existiert die Klasse `eltecIPP`, die alle notwendigen Methoden zur Manipulation der Hardware bereitstellt.

Die Beschreibung der Schnittstelle dieser Klasse wie auch die der folgenden Klassen zur direkten Beeinflußung einzelner Karten sind nicht wichtig für das Verständnis und sind deshalb in dieser Dokumentation nicht enthalten.

8.1.3 Thinedge Processor (THIN)

Über den 16-Bit breiten VI-Bus erhält die THIN-Karte das Grauwertbild vom IPP. In diesem Bild werden mittels eines 8x8 Filters Kanten detektiert und verdünnt [3]. Die Funktionsweise entspricht prinzipiell der des Sobel-Operators (3x3). Der Filter besteht aus zwei 8x8 Matrizen, die relativ frei programmiert werden können. Es stehen fabrikmäßig drei Filter als Beispiele zur Auswahl. Eine zweite Möglichkeit zur Parametrisierung ergibt sich aus der Festlegung eines Schwellwertes, ab dem eine Kante überhaupt extrahiert werden soll. Das Ergebnis dieses Verarbeitungsschrittes ist ein Binärbild das schon die extrahierten Kanten enthält und ein Gradientenbild, jeweils in der Größe des Originalbildes. Die Konfiguration der THIN-Karte erfolgt mit den Methoden der Klasse `eltecThin`.

8.1.4 Vector Processor (VECT)

Die letzte der drei Karten erhält ebenfalls wieder über den VI-Bus das Ergebnis des vorhergehenden Verarbeitungsschrittes um aus den Pixeln der Grauwertkanten Konturen in Form von Polygonzügen zu approximieren, d.h. zu vektorisieren [4]. Die Genauigkeit der Approximation kann über eine sogenannte Winkeltoleranz beeinflußt werden. Anhand des Binär- und Gradientenbildes werden für die einzelnen Grauwertkanten Konturpunkte erzeugt wenn eine neue Kante beginnt, oder die Richtungsänderung seit dem letzten Konturpunkt dieser Kante die zuvor festgelegt Winkeltoleranz überschreitet. Der Polygonzug, im weiteren mit Kontur bezeichnet, besteht dabei mindestens aus zwei Konturpunkten. Die einzelnen Konturpunkte werden aus bestimmten Hardware-Registern ausgelesen und haben die in Prog. 8.1 beschriebene Datenstruktur.

Programm 8.1: Die von der Hardware vorgegebene Datenstruktur.

```
1  typedef struct {
2    unsigned short x;        /* column adress of contourpoint */
3    unsigned short y;        /* row    adress of contourpoint */
4    unsigned short angle;    /* direction of contour */
5    unsigned short stendfl;  /* start/end flag of contour */
6    unsigned short cnr;      /* contour number */
7    unsigned short nstendfl; /* start/end flag of neighbour contour */
8    unsigned short ncnr;     /* contour number of a neigbour */
9  } VECTPOINT;
```

Durch die Arbeitsweise des Systems das ein Grauwertbild von links oben nach rechts unten verarbeitet ergibt sich das Problem, daß die zu einer Kontur gehörenden Konturpunkte im allgemeinen nicht fortlaufend aus den Registern ausgelesen werden können. D.h. nach Beendigung der Vektorisierung des Bildes existiert eine Menge von unzusammenhängenden Konturpunkten, die zuerst anhand ihrer Konturnummer einander zugeordnet werden müssen. Zur Konfiguration der VECT-Karte sind in der Klasse `eltecVect` die notwendigen Methoden vorhanden.

8.1.5 Handhabung des Gesamtsystems

Die Hierarchie der hardware-relevanten Klassen ist in Abb. 8.1 dargestellt, wobei Oberklassen auch im Bild über den Unterklassen liegen.

Für die erfolgreiche Benutzung des Eltec-Systems genügt im Normalfall die Verwendung der Schnittstelle der Klasse `eltecSystem`, die alle Methoden zur einfachen Handhabung des Systems bereitstellt (siehe Prog. 8.2).

Der Konstruktor wie auch die Methoden dieser Klasse erlauben die Einstellung folgender Systemparameter:

1. Modus: (default: Mono-Noninterlaced)
   Das System kann momentan in den drei verschiedenen Modi Mono-Noninterlaced (`MONO_NON`), Mono-Interlaced (`MONO_INT`) und Stereo (`STEREO`) betrieben werden. Die schnellste Verarbeitung wird durch den Modus Mono-Noninterlaced erreicht.

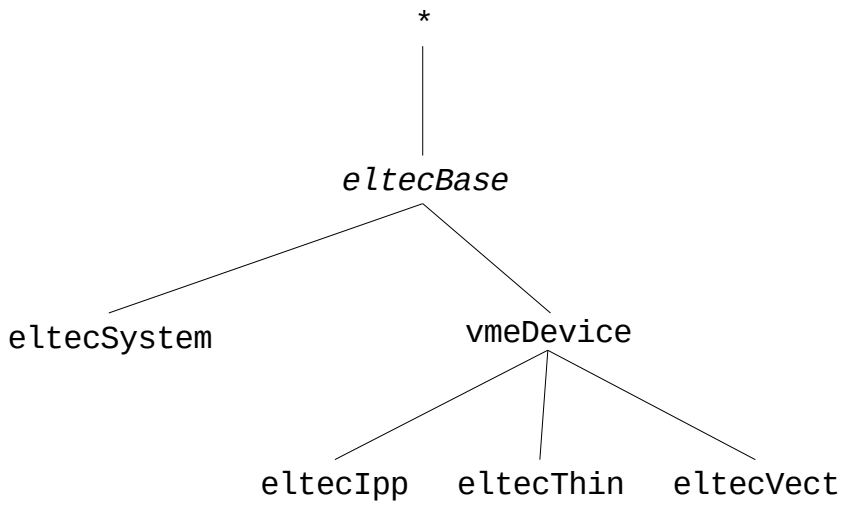


Abbildung 8.1: Die Klassenhierarchie zur Handhabung der Hardware.

Programm 8.2: Die öffentliche Schnittstelle der Klasse `eltecSystem`.

```
 1  class eltecSystem:public eltecBase
 2  {
 3    public:
 4      // enumerations
 5      const enum selmode { MONO_NON = 0,      // mono noninterlaced
 6                           MONO_INT = 1,      // mono interlaced
 7                           STEREO = 2    };   // stereo
 8
 9      // instance variables
10      eltecVect vect;   // Eltec-VECT Contour Vectorizer
11      eltecThin thin;   // Eltec-THIN Contour Thinner
12      eltecIpp  ipp;    // Eltec-IPP Framegrabber
13
14      // constructors
15      eltecSystem(const u_short imgXoffset,
16                  const u_short imgYoffset,
17                  const u_short imgWidth,
18                  const u_short imgHeight);
19
20      // methods
21      void setMode(const selmode mode);
22      void load_thresh(const u_char thresh);
23      void load_filter(const eltecConvolver& conv);
24      void load_filter(const char file[]);
25      void set_angle(const char angle);
26  };
```

2. Schwellwert: (default 20)
   Die Methode `load_thresh()` ermöglicht die Einstellung des Schwellwertes ab dem eine Kante durch den Grauwertunterschied im Originalbild extrahiert werden soll. Der Wertebereich dieses Parameters stammt aus dem Intervall [0, 255].

3. Filter: (default: siehe Abb. 8.2 oder Datei `Elt/data/conv0.ini`)
   Es können prinzipiell beliebige Filter zur Kantenextraktion verwendet werden. Durch die Methode `load_filter()` kann über die Klasse `eltecConvolver` ein neuer Filter dem System mitgeteilt werden. Diese Klasse ermöglicht es unter anderem in Dateiform gespeicherte Filter einzulesen. Im Verzeichnis `data` sind in den Dateien `conv[012].ini` drei verschiedene Filter abgelegt, die aber auch über die Variablen `conv[012]` vom Typ `eltecConvolver` verfügbar sind.

0	0	-2	-4	-4	-2	0	0
0	-5	-20	-25	-25	-20	-5	0
-2	-20	-56	-60	-60	-56	-20	-2
-4	-25	-60	-61	-61	-60	-25	-4
4	25	60	61	61	60	25	4
2	20	56	60	60	56	20	2
0	5	20	25	25	20	5	0
0	0	2	4	4	2	0	0

0	0	-2	-4	4	2	0	0
0	-5	-20	-25	25	20	5	0
-2	-20	-56	-60	60	56	20	2
-4	-25	-60	-61	61	60	25	4
-4	-25	-60	-61	61	60	25	4
-2	-20	-56	-60	60	56	20	2
0	-5	-20	-25	25	20	5	0
0	0	-2	-4	4	2	0	0

Abbildung 8.2: Die horizontale und vertikale Komponente des Default-Filters.

4. Winkeltoleranz: (default: 14)
   Eine weitere Möglichkeit die Hardware zu beeinflußen ist durch die Methode `set_angle()` gegeben. Gültige Parameterwerte sind aus Tabelle 8.1 abzulesen. Der Parameterwert -1 nimmt dabei eine Sonderstellung ein, da in diesem Fall die Grauwertkanten nicht im eigentlichen Sinn durch Polynomzüge approximiert werden, sondern Kantenpixel für Kantenpixel ausgelesen werden können. Diese Möglichkeit ist nur der Vollständigkeit halber aufgeführt, da sie in der Praxis ein viel zu großes Datenaufkommen verursacht.

Parameterwert	-1	0	2	4	6	8	10	12	14
Winkeltoleranz [^{o}]	0.0	1.4	4.2	7.0	9.8	12.6	15.4	18.2	21.0

Tabelle 8.1: Die Zuordnung von Parameterwerten zu Winkeltoleranzen.

5. Position und Größe des Bildausschnittes: (default: Position 82, 50; Größe 744x262)
   Im Gegensatz zu den vorangegangenen Parametern können diese vier Parameter nur bei der Instanziierung angegeben werden. Die ersten beiden Parameter beschreiben die Koordinaten der linken oberen Ecke des zu digitalisierenden Bildausschnitts. Die beiden verbliebenen Parameter bestimmen Breite und Höhe des Bildausschnitts.

Der in 4. beschriebene Parameter ist jederzeit ohne Einschränkungen änderbar.

Eine Änderung der in 1. bis 3. aufgeführten Parameter ist zwar jederzeit möglich, beansprucht aber relativ viel Zeit, da zum Teil Lookup-Tables des Systems neu berechnet und dem System mitgeteilt werden müssen.

Die zuletzt genannten Parameter (siehe 5.) sind nur bei der Instanziierung festlegbar, da ein Großteil der weiter unten beschriebenen Software auf dieser Basis dynamische Felder anlegt.

Bei der Instanziierung einer Variablen der Klasse `eltecSystem` können einige oder auch alle der aufgezählten Parameter explizit mit Werten belegt werden um die Default-Werte zu überschreiben.

8.1.6 Tips & Tricks

- Der einzige auftretbare Fehler ist ein Überlauf des Konturpuffers. Es wird dabei eine so große Anzahl von Konturen vektorisiert, daß der interne Puffer der Hardware nicht schnell genug ausgelesen werden kann. Dieser Fehler wird direkt von der Hardware generiert und hat zur Folge, daß der gesamte Frame verloren ist. Dieser Fehler tritt häufig bei Bildstörungen auf, aber auch wenn die Hardware-Parameter ungünstig gewählt wurden.
- Die wirkungsvollste Möglichkeit die vektorisierte Datenmenge zu beeinflußen ist die Winkeltoleranz. Je kleiner die Datenmenge desto schneller die Bildverarbeitung. Allerdings muß hier ein anwendungsabhängiger Kompromiß gefunden werden zwischen der Geschwindigkeit und der Ungenauigkeit der vektorisierten Kanten.
- Der Modus Mono-Interlaced bringt in Punkto Genauigkeit der Kanten keine nennenswerten Steigerungen und das auf Kosten der doppelten Verarbeitungszeit durch die Hardware.
- Für alle Hardware-Parameter ist fröhliches Experimentieren angesagt, auch bei den Filtern.

8.2 Konturpunkte

Nach dieser kurzen Einführung in den Hardware-Teil des Gesamtsystems endlich zu den von der Hardware gelieferten Daten. Die in Prog. 8.1 vorgestellte Datenstruktur `VECTPOINT` beschreibt die Hardware-Rohdaten eines einzelnen Konturpunkts. Diese Datenstruktur wurde in eine C++ Klasse namens `ctrPoint` (siehe Prog. 8.3) umbenannt und modifiziert.

Beim Vergleich dieser Klasse mit der Struktur `VECTPOINT` fällt auf, daß die Strukturkomponenten `stendfl`, `cnr`, `nstendfl` und `ncnr` in der öffentlichen Klassen-Schnittstelle fehlen. Die darin abgelegten Daten werden nur zur Sortierung und Verknüpfung der unsortierten Menge von Konturpunkten zu zusammenhängenden Konturen benötigt und gehören damit zum privaten Teil dieser Klasse. Die verbliebenen drei Attribute dieser Klasse haben folgende Semantik:

1. `x` und
2. `y` geben die Koordinaten des Konturpunktes im Konturbild an, wobei der Ursprung des Koordinatensystems in der linken oberen Ecke liegt.
3. `angle` enthält die Richtung der extrahierten Kontur an diesem Punkt. Der Wert stammt aus dem Intervall [0, 255], d.h. die 360^o des Vollkreises (Abb. 8.3, links) werden auf dieses Intervall abgebildet (Abb. 8.3, rechts). Wichtig an dieser Stelle

Programm 8.3: Die öffentliche Schnittstelle der Klasse `ctrPoint`.

```
 1  class ctrPoint:public eltecBase
 2  {
 3    public:
 4      // Instance Variables:
 5      short  x;       // column adress of contourpoint
 6      short  y;       // row    adress of contourpoint
 7      u_char angle;   // direction of contour
 8
 9      // Methods:
10             u_int angle2mask(void) const;
11      static u_int angle2mask(register const char angle);
12
13      static u_int isqrt(register const u_int x);
14      static bool in(const short left,
15                     const short x,
16                     const short right);
17
18      u_short distance(register const ctrPoint& p) const;
19      double gradient(register const ctrPoint& p) const;
20
21      bool intersectX(const ctrPoint& p1,
22                      const ctrPoint& p2,
23                      const short X);
24      bool intersectY(const ctrPoint& p1,
25                      const ctrPoint& p2,
26                      const short Y);
27
28      friend ofstream& operator<<(ofstream& s, ctrPoint& x);
29      friend ifstream& operator>>(ifstream& s, ctrPoint& x);
30      friend ostream& operator<<(ostream& s, ctrPoint& x);
31  };
```

ist, daß Richtungen aus dem Teilintervall [0, 127] durch Grauwertkanten enstehen, die, im Bild von links nach rechts gesehen, einen Helligkeitsverlauf von dunkel nach hell aufweisen. Umgekehrt sind Richtungen aus dem Teilintervall [128, 255] durch Kanten mit einem Verlauf von hell nach dunkel bestimmt.

Des weiteren sind für diese Klasse einige Methoden implementiert, die eine Weiterverarbeitung der Daten vereinfachen.

1. Die Methode `angle2mask()` setzt in den 32 Bit eines Integers je nach Wert des Attributs `angle` ein bestimmtes Bit. Die genau Zuordnung kann aus dem rechten Teil der Abb. 8.3 abgelesen werden, wobei die in den Kreissegmenten angegebenen Nummern mit der zu setzenden Bit-Nummer korrespondieren.
2. Da im Zusammenhang mit Konturpunkten nur Ganzzahlen vorkommen ist hier die Methode `isqrt()` bereitgestellt. Sie erlaubt ein bedeutend schnelleres Wurzelziehen als mit der herkömmlichen Fließkommafunktion `sqrt()`.
3. Die Methode `in()` dient lediglich zur Prüfung, ob ein Wert innerhalb eines bestimmten Intervalls liegt.
4. `distance()` ermittelt den euklidschen Abstand zwischen den beteiligten Kontupunkten.

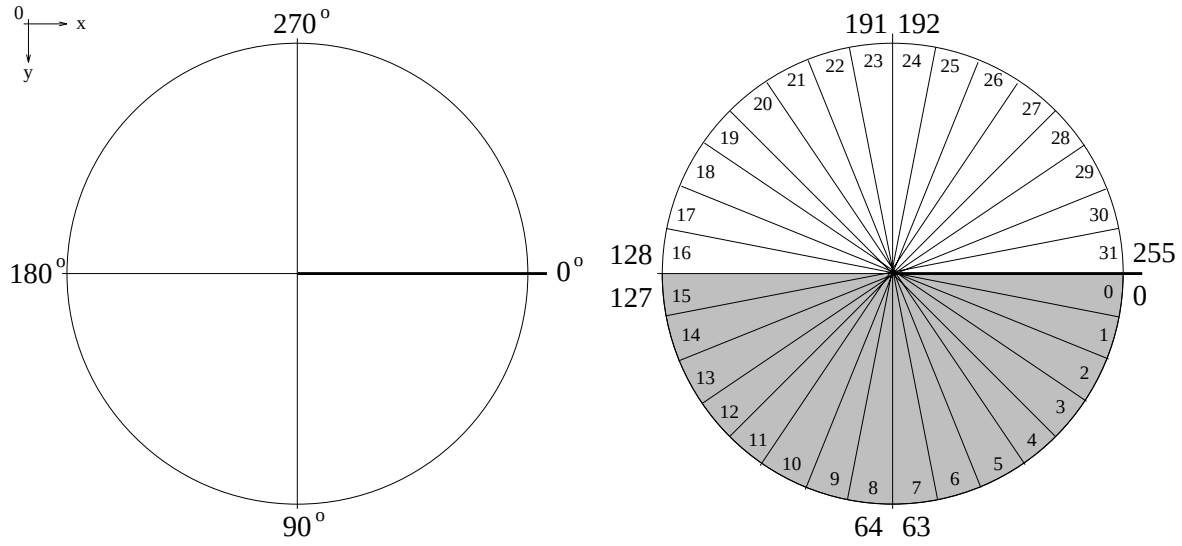


Abbildung 8.3: Die Richtung einer Kontur in einem Konturpunkt.

5. Mittels der Methode `gradient()` kann die Steigung der Geraden durch die beiden beteiligten Punkte ermittelt werden. Falls die x-Koordinaten der Konturpunkte identisch sein sollten und damit die Steigung unendlich wäre, wird der Wert `MAX_DOUBLE` zurückgegeben.
6. Der Methode `intersectX()` werden als Parameter zwei Konturpunkte und eine Koordinate `x` übergeben. Sie berechnet daraus den Schnittpunkt der Strecke zwischen den beiden Konturpunkten und einer Vertikalen an der Stelle `x`. Das Vorhandensein eines Schnittpunktes wird durch den boolschen Rückgabewert angezeigt.
7. Die Funktionsweise der Methode `intersectY()` ist äquivalent zur vorangegangenen Methode. Der einzige Unterschied ist, daß hier der Schnittpunkt mit einer Horizontalen bestimmt wird.
8. Die Operatoren << und >> dienen zur formatierten Datei und Bildschirm Ein- und Ausgabe.

8.3 Konturen

Eine Kontur besteht wie bereits erwähnt aus mindestens zwei Konturpunkten, die aus Gründen der hohen Flexibilität in einer doppelt verketteten Liste verwaltet werden. Zur Implementierung dieser Konturpunktliste wurde die Containerklasse `DLList` der GNU C++ Bibliothek verwendet. Die Klasse `ctrPointDLList` hat damit, ohne Berücksichtigung der ererbten Schnittstelle, das in Prog. 8.4 beschrieben Aussehen.

Für die Beschreibung der ererbten Klassenschnittstelle sei an dieser Stelle auf die Info-Seiten im `emacs` verwiesen. Dort gibt es eine Eintrag `Libg++` und darin einen Verweis auf `LinkList` in dem die vorhandenen Methoden von einfach und doppelt verketteten Listen beschrieben sind.

Bei der genaueren Betrachtung der Klassendeklaration von `ctrPointDLList` fällt auf, daß die Elemente der Liste nicht vom Typ `ctrPoint`, sondern vom Typ `ctrPoint *` sind. Dies hat vor allem Effizienzgründe, da etwa beim Einfügen von Elementen

Programm 8.4: Die öffentliche Schnittstelle der Klasse `ctrPointDLList`.

```
 1  class ctrPointDLList:public DLList<ctrPoint *>, public eltecBase
 2  {
 3    public:
 4      // Methods:
 5      u_short pixLength(void);
 6      u_int mask(void);
 7      void transX(register short dx);
 8      void transY(register short dy);
 9      void pre_join_reverse(register ctrPointDLList& x);
10      void pre_join(register ctrPointDLList& x);
11      void join_reverse(register ctrPointDLList& x);
12      void cut(Pix p, int dir = 1);
13      friend ostream& operator<<(ostream& s, ctrPointDLList& x);
14  };
```

immer zuerst eine Kopie des Elements erzeugt wird und diese dann in die Liste wandert. D.h. durch die Verwendung von Zeigern wird die zu kopierende Datenmenge minimiert.

Die durch die Vererbung schon vorhandenen Methoden wurden folgendermaßen ergänzt:

1. Die Methode `pixLength()` berechnet die Länge [pel] der durch die Konturpunktliste repräsentierten Kontur als Summe der euklidschen Abstände zwischen den einzelnen Konturpunkten.
2. Mit Hilfe von `mask()` werden alle vorkommenden Richtungen der Kontur in einer 32-Bit Maske codiert zurückgegeben.
3. Durch die Methode `transX()` werden alle Konturpunkte dieser Kontur um die übergebene Anzahl von Pixeln in x-Richtung verschoben.
4. Äquivalent dazu verschiebt `transY()` die Kontur in y-Richtung.
5. Die drei Methoden `pre_join_reverse()` (siehe Abb. 8.4a),
6. `pre_join()` (siehe Abb. 8.4b) und
7. `join_reverse()` (siehe Abb. 8.4c) verketten jeweils die Liste a mit der übergebenen Liste `b`.

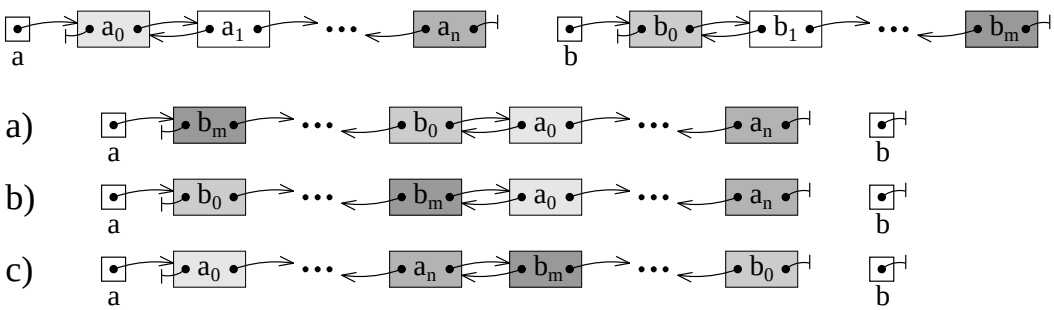


Abbildung 8.4: Verkettung von Konturlisten.

8. Durch die Anwendung der Methode `cut()` kann entweder das linke oder das rechte Ende einer Konturliste abgetrennt werden. Als Parameter wird das Listenelement angegeben von wo aus die Trennung erfolgen soll. Die Richtung in der abgetrennt wird bestimmt das Vorzeichen des zweiten Parameters: negativ entspricht links und positiv rechts vom angegebenen Listenelement.
9. Der Operator << dient zur Bildschirmausgabe einer vollständigen Konturliste.

Da für jede Kontur die Verfügbarkeit zusätzlicher Attribute interessant ist, existiert eine übergeordnete Klasse `ctrAttrib`, die diese Aufgabe erfüllt (siehe Prog. 8.5).

Programm 8.5: Die öffentliche Schnittstelle der Klasse `ctrAttrib`.

```
 1  class ctrAttrib:public eltecBase
 2  {
 3    public:
 4      // Enumerations:
 5      const enum ctrType { CTR_MONO,      // mono image
 6                           CTR_FIRST,     // first half stereo image
 7                           CTR_SECOND };  // second half stereo image
 8
 9      // Instance Variables:
10      u_short        cnr;        // id-number of contour
11      u_short        points;     // number of points
12      u_short        pixLen;     // length of contour [pel]
13      u_int          angleMask;  // anglemask containing all angles
14      ctrType        type;       // source of contour
15      ctrPointDLList pts;        // list of points
16
17      // Methods:
18      friend ostream& operator<<(ostream& s, ctrAttrib& x);
19  };
```

Im einzelnen sind folgende Attribute zu nennen:

1. Die Instanzvariable `cnr` enthält eine eindeutige Konturnummer. Sie stimmt mit der Nummer überein, die, wie in Abschnitt 8.1.4 beschrieben, für jeden einzelnen Konturpunkt in der Struktur `VECTPOINT` von der Hardware belegt wird.
2. Die Anzahl der Punkte dieser Kontur ist in `points` festgehalten.
3. Das Attribut `pixLen` beinhaltet die Länge der Kontur in Pixeln.
4. In der Variablen `angleMask` sind alle in dieser Kontur enthaltenen Richtungen in Form einer Bitmaske gespeichert.
5. Die Variable `type` enthält für den Fall daß mit Stereobildern gearbeitet wird eine Kennzeichnung aus welcher der beiden Halbbilder diese Kontur stammt. Mögliche Belegungen sind `CTR_FIRST` oder `CTR_SECOND`.
6. Auf die eigentliche Liste von Konturpunkten kann durch die Variable `pts` zugegriffen werden.
7. Der Operator << dient wieder der Ausgabe auf den Bildschirm.

Da in der weiteren Verarbeitung Mengen von Konturen zu untersuchen sind, werden diese ebenfalls in doppelt verketteten Listen verwaltet. Die entsprechende Klasse heißt `ctrAttribDLList` und ist in Prog. 8.6 abgebildet.

Programm 8.6: Die öffentliche Schnittstelle der Klasse `ctrAttribDLList`.

```
1  class ctrAttribDLList:public DLList<ctrAttrib *>, public eltecBase
2  {
3    public:
4      // Methods:
5      friend ostream& operator<<(ostream& s, ctrAttribDLList& x);
6  };
```

In dieser Klasse wurden bisher, außer einer Methode zur Ausgabe auf den Bildschirm, keine weiteren Methoden implementiert. Wie aus der Klassendeklaration zu entnehmen ist, sind die Elemente der Liste aus Effizienzgründen ebenfalls wieder nur Zeiger.

8.4 Konturdatenbanken

In den vorangegangenen Abschnitten wurden die Klassen zur Handhabung von Konturen vorgestellt. Diese müssen nun ergänzt werden durch Strukturen die die eigentlichen Daten enthalten und auf denen die Listen von Zeigern ihre Gültigkeit erhalten.

Zum Speichern der eigentlichen Konturen werden mehrere verschiedene Klassen verwendet, die jeweils eine primitive Datenbank implementieren. Primitiv deshalb, da der Benutzer nur lesenden Zugriff auf die Daten hat und im momentanen Stadium kein Mehrbenutzerbetrieb möglich ist.

Die Funktionalität der einzelnen Klassen unterscheidet sich einerseits in der Art und Weise wie die Konturdaten abgelegt werden, mit Auswirkungen auf die Arten von zulässigen Datenbankanfragen. Andererseits gibt es Unterschiede bezüglich der Art der Vorverarbeitung beim Datenbankaufbau.

8.4.1 Die Ablage der Konturdaten

Da jede der im weiteren beschriebenen Klassen von Datenbanken von der Klasse `eltecBaseDatabase` den öffentlichen Teil der Klasse erbt, genügt es die Methoden dieser Basisklasse genauer zu kennen (siehe Prog. 8.7). Unterschiede zwischen den Schnittstellen der abgeleiteten Klassen bestehen nur in den Parameterleisten der Konstruktoren.

Die vorhandenen Methoden implementieren folgende Funktionalität:

1. Durch die Methode `setGlobalMinLen()` besteht die Möglichkeit die beim Aufbau von Indizes berücksichtigten Konturen von vornherein zu filtern, indem eine Mindestlänge [pel] für Konturen vorgschrieben wird (default: 0 [pel]).
2. Eine ähnlich Möglichkeit bietet die Methode `setGlobalMask()`, durch die nur bestimmte Richtungen von Konturen beim Aufbau der Datenbankindizes berücksichtigt werden (default: `0xffffffff`).
3. Mittels der Methode `getTimeStamp()` kann der Zeitstempel, von dem die in dieser Datenbank gespeicherten Konturen stammen, abgefragt werden. Das Ergebnis sind Sekunden seit dem 1. Januar 1970 (siehe auch `man gettimeofday()`). Der Zeit-

Programm 8.7: Die öffentlich Schnittstelle der Klasse `eltecBaseDatabase`.

```
 1  class eltecBaseDatabase:public virtual eltecBase
 2  {
 3    public:
 4      // Constructors:
 5      eltecBaseDatabase(void);
 6
 7      // Destructor:
 8      ~eltecBaseDatabase(void);
 9
10      // Methods:
11      void setGlobalMinLen(const u_short len = 0);
12      void setGlobalMask(const u_int msk = 0xffffffff);
13
14      double getTimeStamp(void) const;
15      int getFrameNr(void) const;
16      int getPtsQuant(void) const;
17      int getFrame(register eltecSystem& eltec);
18      int getFrame(const char file[],       // base name of file
19                   const u_short nrLen = 4);// char length of frameNr
20
21      virtual void build(void);
22
23      int read(const char file[],          // base name of file
24               const u_short nrLen = 0,    // char length of frameNr
25               const int frame = 0,        // frameNr
26               const char ext[] = „ctr“);  // file extension
27      void write(const char file[],        // base name of file
28                 const u_short nrLen = 0,  // char length of frameNr
29                 const char ext[] = „ctr“);// file extension
30  };
```

stempel wird direkt nach dem Auslesen des ersten vektorisierten Konturpunkts bestimmt.

4. `getFrameNr()` gibt einen Integer-Wert zurück, der die Nummer das aktuellen Frames in dieser Datenbank angibt.
5. Durch die Methode `getPtsQuant()` kann die in dieser Datenbank gespeicherte Anzahl von Konturpunkten abgefragt werden.
6. Durch die Methoden `getFrame()` ist es möglich einen neuen Frame entweder von der Hardware oder von einer Datei einzulesen. Als Rückgabe dieser Methode erhält man die Anzahl der eingelesenen Konturpunkte oder -1, falls dabei ein Fehler in Form eines Überlaufs des Konturpuffers auftritt.
7. Durch den Aufruf der Methode `build()` werden die Indizes der Datenbank für die neuen Rohdaten aufgebaut.
8. Schließlich gibt es noch die Methode `read()`, durch die auf eine Datei von Kontur-Rohdaten zugegriffen werden kann.
9. Das Gegenstück zum vorhergehenden Punkt ist die Methode `write()`, die zum Schreiben von Rohdaten vorgesehen ist. Von der Hardware eingelesene Daten

können nur im Rohzustand abgespeichert werden, d.h. vor dem Aufruf der Methode `build()`.

Die von der Basisklasse abgeleiteten Klassen unterscheiden sich im wesentlichen nur in der Implementierung der unter 7. aufgeführten Methode `build()`. Je nach Art der Datenbank können, wie bereits erwähnt, Anfragen gestellt werde, um Konturen mit bestimmten Attributwerten zu erhalten:

10. Dazu haben alle Klassen die Methode `request()`, die jedoch in der Basisklasse noch nicht vorhanden ist, da diese erst in den Verfeinerungen der Unterklassen definiert wird (siehe Prog. 8.9).

8.4.2 Der Aufbau der Datenbank

Während des Einlesens der Rohdaten eines neuen Frames werden die unsortierten Konturpunkte mit identischer Konturnummer zu zusammenhängenden Konturen verknüpft. Bei dieser Verknüpfung werden Konturen die durch einen Fehler nur aus einem einzigen Konturpunkt bestehen herausgefiltert.

Beim weiteren Aufbau der Datenbank können zusätzliche, miteinander kombinierbare, Vorverarbeitungsschritte ausgeführt werden:

1. Von der Hardware werden in den Endpunkten von Konturen Verweise auf etwa vorhandene Nachbarkonturen mitgegeben. Solche Verweise erscheinen im Normalfall nur, wenn die Endpunkte der benachbarten Konturen in ihren Pixelkoordinaten direkte Nachbarn sind. Somit können diese Konturen direkt miteinander verknüpft werden.
2. Bei der Verwendung von Stereobildern sind die beiden Halbbilder in ein Vollbild gemischt, d.h. die Halbbilder müssen voneinander getrennt und die Koordinaten des zweiten Halbbilds transformiert werden.

Erst nach dem vollständigen Durchlaufen dieser Vorverarbeitungsschritte werden die in Abschnitt 8.3 aufgeführten Konturattribute berechnet. Anschließend werden die Konturen auf die Erfüllung der Mindestwerte von Attributen geprüft, die durch die in Abschnitt 8.4.1 aufgeführten Methoden `setGlobalMinLen()` und `setGlobalMask()` gesetzt worden sind.

Nachdem die Konturdaten vorverarbeitet und gefiltert sind werden die eigentliche Suchindizes aufgebaut. Dabei kann zwischen zwei miteinander kombinierbaren Möglichkeiten gewählt werden:

1. Der erste Index ist ein eindimensionales Feld, durch das alle verbliebenen Konturen referenziert werden können. Dieser Index wird im weiteren mit Linear-Index bezeichnet.
2. Um Konturen aus einem bestimmten Bildbereich abfragen zu können ohne jedesmal alle Konturpunkte der einzelnen Kontur abzuprüfen, werden die Konturen in ein Bild gezeichnet. Dieses Indexbild, oder im weiteren auch Raster-Index genannt, besteht aus einzelnen Indexpixeln, die im Normalfall eine Fläche von mehrere Pixeln des Originalbildes repräsentieren. Diese beinhalten Verweise auf alle Konturen die durch den entsprechenden Pixelbereich verlaufen. Das Größenverhältnis zwischen Pixeln im Originalbild und Indexpixeln kann in beiden Dimensionen den Anforderungen der Anwendung angepaßt werden [5]. In Abb. 8.5 sind

beispielsweise die Indexpixel in x-Richtung um den Faktor $2^2 = 4$ und in y-Richtung um den Faktor $2^1 = 2$ skaliert. Das gestrichelte Raster entspricht der Matrix des Originalbildes und das darübergelegte gröbere Raster der Matrix der Indexpixel. Auf der rechten Seite ist der Inhalt des Raster-Index für die eingezeichneten Konturen angegeben.

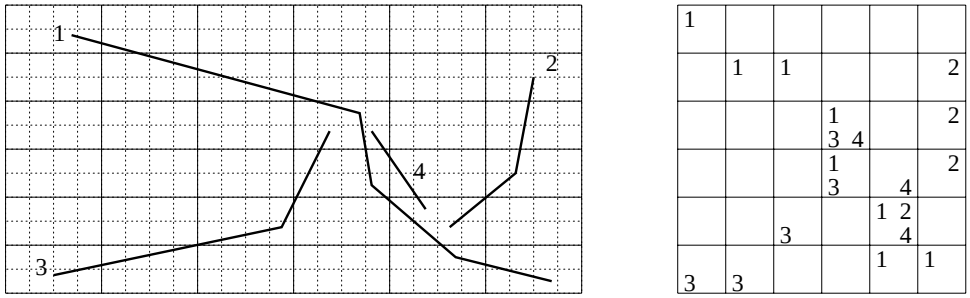


Abbildung 8.5: Die Abbildung von Konturen in den Raster-Index.

Jede Kombinationsmöglichkeit aus Vorverarbeitung und verwendeten Indizes ist eine eigene Klasse. Da zumindest ein Index vorhanden sein muß, aber nicht unbedingt eine spezielle Vorverarbeitung notwendig ist, ergeben sich die in Tabelle 8.2 aufgeführten 12 Klassen.

Klassenname	Indizes		Vorverarbeitung	
	Linear-Index	Raster-Index	Nachbar-konturen verbinden	Stereobild trennen
dbLinear	X			
dbLinear_Join	X		X	
dbLinear_Stereo	X			X
dbLinear_StereoJoin	X		X	X
dbRaster		X		
dbRaster_Join		X	X	
dbRaster_Stereo		X		X
dbRaster_StereoJoin		X	X	X
dbLinRast	X	X		
dbLinRast_Join	X	X	X	

Tabelle 8.2: Klassen von Konturdatenbanken und ihre Merkmale.

Klassenname	Indizes		Vorverarbeitung	
	Linear-Index	Raster-Index	Nachbar-konturen verbinden	Stereobild trennen
dbLinRast_Stereo	X	X		X
dbLinRast_StereoJoin	X	X	X	X

Tabelle 8.2: Klassen von Konturdatenbanken und ihre Merkmale.

Die Schnittstellen dieser Klassen sind bezüglich ihrer Methoden identisch. Unterschiede bestehen lediglich in den Parameterleisten der Konstruktoren. Klassen die das Stereobild trennen, brauchen Informationen über die Breite und Höhe des Bildes. Dieselbe Information benötigen Klassen die einen Raster-Index verwenden. Diese brauchen jedoch zusätzlich noch die Skalierungsfaktoren für den Aufbau des Raster-Indexes. Eine Parameterwert n resultiert dabei in einer Skalierung 2^n, d.h. bei einem Wert von n = 3 für die Breite ist ein Indexpixel $2^3 = 8$ Pixel breit.

8.4.3 Tips & Tricks

- Die Hardware und die Datenbanken müssen **immer** mit derselben Bildgröße arbeiten. Falls versucht wird verschiedene Werte bei der Instanziierung zu verwenden bricht das Programm mit einer Fehlermeldung ab.
- Wenn versucht wird bei der Instanziierung von Datenbanken verschiedene Skalierungsfaktoren zu verwenden wird das Programm ebenfalls sofort mit einer Fehlermeldung abgebrochen.
- Aus Geschwindigkeitsgründen sollte immer die Klasse von Datenbank verwenden, die die Minimalvoraussetzungen erfüllt, da natürlich jeder zusätzliche Verarbeitungsschritt Zeit kostet.
- Die Angabe einer minimale Konturlänge kann sehr viel Rechenzeit ersparen, allerdings hängt der Wert stark von der Anwendung ab. Falls bei der Vorverarbeitung Nachbarkonturen verbunden werden kann dieser Wert etwas höher gewählt werden.
- Bei der Skalierung des Raster-Indexes sollten jeweils Werte größer als 1 verwendet werden. Da das Ausmaß der Bilddimensionen sehr unausgeglichen ist bietet es sich außerdem an die Auflösung des Rasters verschieden zu wählen. Je gröber das Raster desto schneller der Aufbau des Raster-Indexes.

8.5 Anfragen an Konturdatenbanken

Was nützen jedoch Konturen in einem Behälter ohne die Möglichkeit sich gezielt Konturen daraus herausgeben zu lassen. Aus diesem Grund folgt nun die Beschreibung der Klassen zur Abfrage bestimmter Konturen aus einer der Datenbanken.

8.5.1 Auswahl des Bildbereichs

Eine der wichtigsten Angaben bei der Suche nach bestimmten Konturen ist die Angabe des zu durchsuchenden Bildbereichs.

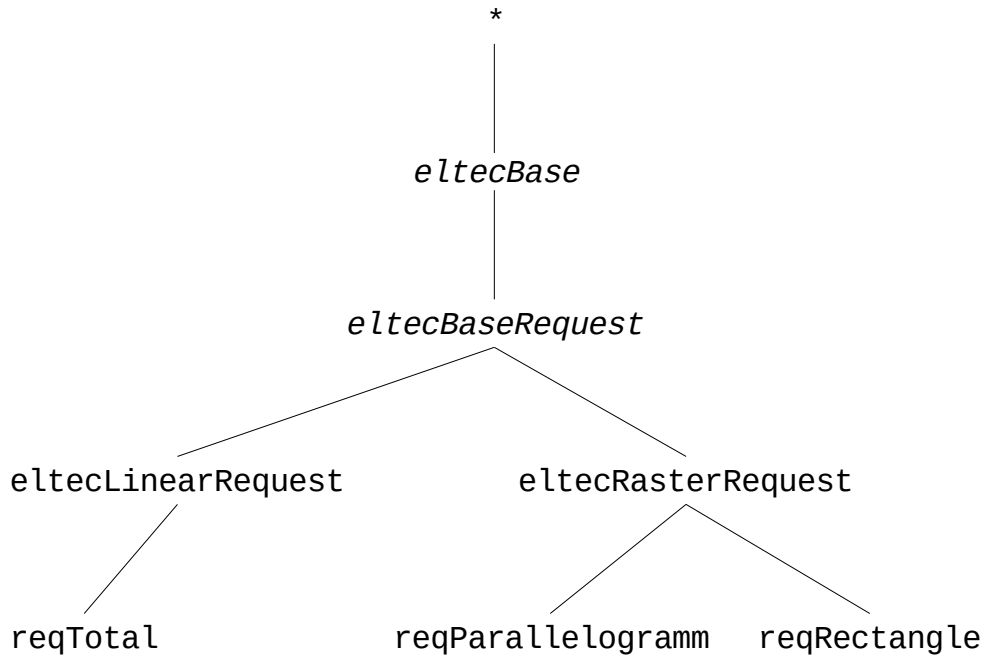


Abbildung 8.6: Die Klassenhierarchie zur Spezifikation von Bildbereichen.

Wie in Abb. 8.6 zu erkennen ist, gibt es ausgehend von der Basis-Klasse `eltecBaseRequest` für Bereichsanfragen an eine Konturdatenbank die zwei direkt abgeleiteten Klassen `eltecLinearRequest` und `eltecRasterRequest`. Die daraus abgeleiteten Klassen sind die in der Praxis zu verwendenden Klassen, mit deren Hilfe entweder innerhalb der gesamten Konturenmenge (`reqTotal`), in einem rechteckigen (`reqRectangle`) oder einem parallelogrammförmigen Bildausschnitt (`reqParallelogramm`) gesucht werden kann.

Klassen die als Oberklasse die Klasse `eltecLinearRequest` haben können nur auf eine Datenbank angewendet werden die einen Linear-Index bereitstellt. Gleiches gilt für die Klasse `eltecRasterRequest` und Datenbanken mit Raster-Index.

Eine Bereichsspezifikation einer Anfrage an eine Datenbank wird durch die Instanziierung einer der genannten Klassen erzeugt. Diese Instanz kann prinzipiell auf jede Instanz einer Datenbank angewendet werden und ist damit beliebig oft wiederverwendbar:

1. Die Klasse `reqTotal` benötigt zur Instanziierung keine weiteren Parameter.
2. Für die Instanziierung der Klasse `reqRectangle` werden die Pixel-Koordinaten der oberen linken Ecke P_1 und der unteren rechten Ecke P_2 des Rechtecksbereichs benötigt (siehe Abb. 8.7).
3. Als Parameter für die Instanziierung eine parallelogrammförmigen Bildbereichs werden wiederum die Pixelkoordinaten zweier übereinander gelegener Punkte P_1

und P_2 benötigt. Diese beschreiben die Neigung der Seiten des Parallelogramms. Zur vollständigen Beschreibung wird noch die Breite d des Parallelogramms in Pixeln benötigt (siehe Abb. 8.7).

Wie aus den Parametern von 2. und 3. zu erkennen ist können keine allgemeinen Rechtecke und Parallelogramme beschrieben werden, sondern nur solche die zwei horizontale Seiten haben.

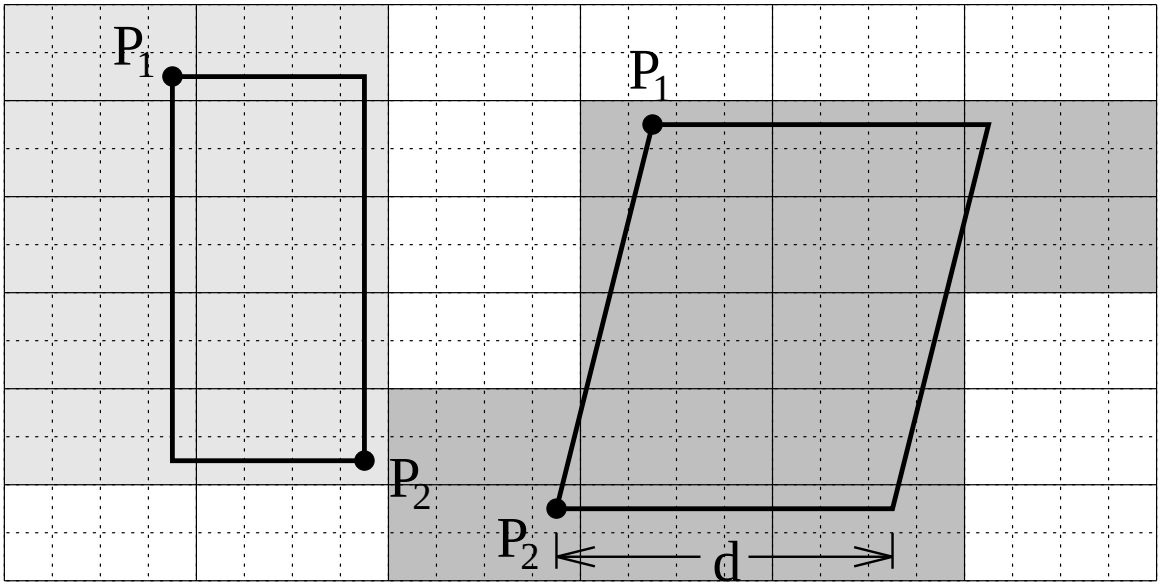


Abbildung 8.7: Spezifikation von Bildbereichen für eine Konturdatenbankanfrage.

In Abb. 8.7 ist anhand der grau unterlegten Flächen zu erkennen welche Bildbereiche durch die als Linien eingezeichneten Bereichsanfragen tatsächlich überdeckt werden.

8.5.2 Auswahl anhand von Konturattributen

Wie in Abschnitt 8.3 beschrieben besitzen Konturen eine Anzahl von Attributen. Da im Normalfall, je nach Anwendung, nur Konturen mit bestimmten Eigenschaften interessant sind, gibt es Möglichkeiten diese genau zu spezifizieren. In Abb. 8.8 sind die grundlegenden Klassen dargestellt die es erlauben entsprechende Attribute abzuprüfen.

Sinnvollerweise können die aufgeführten Klassen `chkMask`, `chkMaxLen`, `chkMinLen` und `chkStereo` beliebig miteinander kombiniert werden. Insgesamt stehen damit 16 verschiedene Klassen zur Spezifikation der zu prüfenden Konturattribute zur Verfügung (siehe Tabelle 8.3).

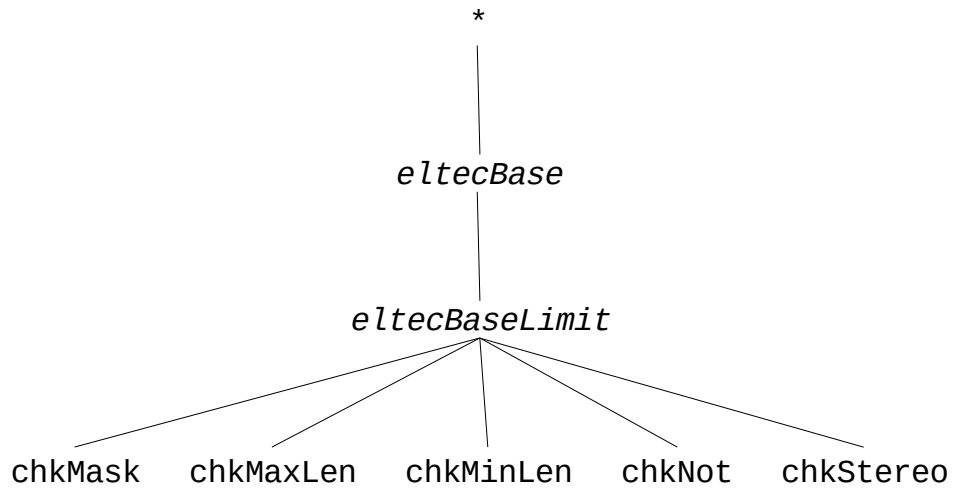


Abbildung 8.8: Klassenhierarchie zur Spezifikation von Konturattributen.

Klassenname	Stereo-halbbild	Winkel-maske	Minimale Länge [pel]	Maximale Länge [pel]
chkNot				
chkMask		X		
chkMinLen			X	
chkMaxLen				X
chkLength			X	X
chkMaskMinLen		X	X	
chkMaskMaxLen		X		X
chkMaskLength		X	X	X
chkStereo	X			
chkStereoMask	X	X		
chkStereoMinLen	X		X	
chkStereoMaxLen	X			X
chkStereoLength	X		X	X
chkStereoMaskMinLen	X	X	X	
chkStereoMaskMaxLen	X	X		X
chkStereoMaskLength	X	X	X	X

Tabelle 8.3: Klassen zur Spezifikation von Konturattributen.

Da die Klassen eine verschiedene Anzahl von Parametern benötigen sind die Konstruktoren den Anforderungen entsprechend angepaßt. Die Reihenfolge der anzugebenden Parameter ist für alle Klassen gleich. Sie entspricht der in der Kopfzeile von Tabelle 8.3 verwendeten Reihenfolge. Falls der eine oder andere Parameter von der zu verwendenden Klasse nicht benötigt wird sind diese bei der Instanziierung der Klasse einfach wegzulassen.

Alle in Tabelle 8.3 aufgeführten Klassen haben nur die eine Methode `ok()`, die sie von der Klasse `eltecBaseLimit` erben (siehe Prog. 8.8). Diese Methode testet ob die über-

Programm 8.8: Die öffentliche Schnittstelle der Klasse `eltecBaseLimit`.

```
1  class eltecBaseLimit:public eltecBase
2  {
3    public:
4      // Methods:
5      virtual bool ok(ctrAttrib *ctr) = 0;
6  };
```

gebene Kontur den Anforderungen der bei der Instanziierung übergebenen Parameter erfüllt. Instanzen dieser Klasse können damit auch außerhalb von Datenbankanfragen zum Testen von Konturattributen auf bestimmte Werte verwendet werden.

8.5.3 Eine komplette Anfrage

Zur Bildung einer Anfrage werden also zwei Komponenten verwendet. Zum einen kann ein bestimmter Bildbereich spezifiziert werden und zum anderen können Attributwerte ausgewählt werden. Diese werden mit zwei weiteren Argumenten der Methode `request()`, die Bestandteil der Schnittstelle jeder Klasse von Datenbanken ist, übergeben (siehe Prog. 8.9).

Programm 8.9: Die Parameterleiste der Datenbank-Methode `request()`.

```
1  virtual void request(ctrAttribDLList& dat,
2                       eltecBaseRequest& req,
3                       eltecBaseLimit& lim = chkNot(),
4                       bool group = false);
```

1. Das erste Argument ist eine Variable vom Typ `ctrAttribDLList`, also einer Liste von Konturen. Das Ergebnis der Anfrage wird an das Ende dieser Liste angehängt.
2. Das zweite Argument betrifft die Spezifikation eines Bildausschnitts mittels einer Instanz der in Abschnitt 8.5.1 erläuterten Klassen.
3. Das nächste Argument ist die Instanz einer der im vorhergehenden Abschnitt aufgeführten Klassen um bestimmte Konturattribute zu testen. Als Default werden keine Attribute getestet, d.h. es wird eine Instanz der Klasse `chkNot` übergeben.
4. Das letzte Argument ist ein boolscher Wert und ermöglicht es mehrere Anfragen zu gruppieren ohne dabei in der Gesamtergebnismenge Duplikate von Konturen zu erhalten. Der Default ist `false`, d.h. jede Anfrage wird separat behandelt.

8.5.4 Tips & Tricks

- **Achtung**: Alle Daten von Konturen und Konturpunkten sind physisch nur **innerhalb** einer Datenbank vorhanden. Das Ergebnis von Anfragen sind nur Listen von Zeigern die auf Daten in der Datenbank verweisen. Änderungen von Attributwerten über diese Zeiger werden damit global in dieser Datenbank vollzogen und können nachfolgenden Anfragen und schon erhaltene Konturdaten beeinflußen!
- Wichtig beim Arbeiten mit rechteckigen oder parallelogrammförmigen Bereichsanfragen ist, daß Konturen immer in ihrer ganzen Länge zurückgegeben werden. D.h. falls eine Kontur von links oben quer über das ganze Bild nach rechts unten verläuft und ein kleiner Bereich in der Bildmitte Ziel der Anfrage ist, wird trotzdem die Kontur nicht an den Bereichsgrenzen abgeschnitten, sondern komplett zurückgegeben.
- Je feiner der Raster-Index ist desto langsamer wird natürlich auch eine Bildbereichsanfrage bearbeitet, da immer aller Index-Pixel eines angegebenen Bildbereichs durchsucht werden müssen.
- Wie in Abb. 8.7 zu erkennen ist kann der Einzugsbereich einer rechteckigen oder parallelogrammförmigen Bereichsanfrage im Endeffekt um einiges größer sein als spezifiziert. Die Genauigkeit der Abgrenzung der Bereiche hängt stark von der Körnigkeit des Rasters ab, d.h. es muß ein anwendungsabhängiger Kompromiß zwischen Geschwindigkeit und Genauigkeit gefunden werden.
- Bei der Verwendung von Richtungsmasken zur Auswahl bestimmter Konturen muß man sich bewußt sein, daß eine Kontur die Randbedingungen erfüllt wenn irgendwo eine der in der Maske angegebenen Richtungen existiert. Das bedeutet insbesonders auch daß bei einer Anfrage aus einer Kombination von Bildbereich und Maske die Maskenwerte nicht unbedingt innerhalb dieses Bereichs erfüllt sein müssen.
- Falls bei der Verarbeitung von Stereobildern nicht die Klassen `chkStereo*` verwendet werden, enthält die Ergebnismenge von Konturen immer die Konturen aus beiden Bildhälften.

8.6 Visualisierung von Konturen

Um die Arbeit mit Konturen zu erleichtern besteht die Möglichkeit diese in einem Fenster zu visualisieren. Dazu gibt es die beiden Klassen `eltecBaseX` und `Xcontours`, wobei `Xcontours` eine Unterklasse von `eltecBaseX` ist. Die Klasse `eltecBaseX` hat die in Prog. 8.10 beschriebene Klassenschnittstelle.

Bei der Instanziierung müssen die Breite und Höhe des Fensters sowie der Titel als Parameter übergeben werden. Optional können die Kommandozeilenoptionen der Anwendung ebenfalls übergeben werden, die dann die X spezifische Optionen herausfiltert und verwendet. Die Klassenschnittstelle hat folgendes Aussehen:

1. Die Methode `Rename()` zur Neubenennung des Fensters und
2. `Resize()` zur Veränderung der Fenstergröße, die damit eine nachträgliche Änderung der bei der Instanziierung festgelegten Parameter ermöglicht.
3. Die Methoden `DrawLine()` zum Zeichnen einer Linie und

Programm 8.10: Die öffentliche Schnittstelle der Klasse `eltecBaseX`.

```
 1  class eltecBaseX:public eltecBase
 2  {
 3    public:
 4      // Enumerations:
 5      const enum colour { WHITE = 0,
 6                          BLACK = 1,
 7                          RED   = 2,
 8                          GREEN = 3,
 9                          BLUE  = 4 };
10
11      // Constructors:
12      eltecBaseX(const u_int w,
13                 const u_int h,
14                 const char title[],
15                 int *argc = NULL,
16                 char **argv = NULL);
17
18      // Destructor:
19      ~eltecBaseX(void);
20
21      // Methods:
22      void Rename(const char title[]);
23      void Resize(const u_int w,
24                  const u_int h);
25
26      void DrawLine(const u_short x0,
27                    const u_short y0,
28                    const u_short x1,
29                    const u_short y1,
30                    const colour c = BLACK);
31      void DrawRectangle(const u_short x0,
32                         const u_short y0,
33                         const u_short x1,
34                         const u_short y1,
35                         const colour c = BLACK);
36      void ClearBitmap(const colour c = WHITE);
37      void ShowBitmap(void);
38      static void workOnEvents(const int doreturn = true);
39  };
```

4. `DrawRectangle()` zum Zeichnen eines Rechtecks können Ausgaben in einer bestimmten Farbe machen. Dazu sind die Farben `WHITE`, `BLACK`, `RED`, `GREEN` und `BLUE` als Konstanten deklariert. Es können mit dieser Methode nur Rechtecke eingezeichnet werden, die zwei horizontale Seiten haben (siehe Abb. 8.7).

5. Da in dieser Implementierung mit Pufferung gearbeitet wird, gibt es die Methode ClearBitmap(), die die zur Pufferung dienende Bitmap mit einer bestimmten Hintergrundfarbe füllt um den Inhalt des Fensters zu löschen.

6. Die Methode ShowBitmap() dient schließlich zum Kopieren der Bitmap in das eigentliche Fenster. Alle Ausgaben die durch die vorangegangenen drei Methoden gemacht wurden sind gepuffert, d.h. bevor eine Ausgabe im Fenster erscheint **muß** diese Methode aufgerufen werden.

7. Mit `workOnEvents()` werden die durch die vorangegangenen Methoden erzeugten X-Events abgearbeitet und damit das Ergebnis erst sichtbar. Als Parameter kann übergeben werden, ob der Kontrollfluß nach dem Abarbeiten der X-Events wieder an das Anwendungsprogramm zurückgegeben werden soll oder nicht.

Zur Ausgabe von Konturen existiert die in Prog. 8.11 beschriebene Klasse `Xcontours`.

Programm 8.11: Die öffentliche Schnittstelle der Klasse `Xcontours`.

```
 1  class Xcontours:public eltecBaseX
 2  {
 3    public:
 4      // Constructors:
 5      Xcontours(const u_int w,
 6                const u_int h,
 7                const char title[],
 8                int *argc = NULL,
 9                char **argv = NULL);
10
11      // Methods:
12      void DrawParallelogramm(const u_short x0,
13                              const u_short y0,
14                              const u_short x1,
15                              const u_short y1,
16                              const short dx,
17                              const colour c = BLACK);
18      void DrawCross(const u_short x,
19                     const u_short y,
20                     const u_short d,
21                     const colour c = BLACK);
22      void DrawContour(ctrPointDLList& ctr,
23                       const colour c = BLACK);
24      void DrawContour(ctrAttrib *ctr,
25                       const colour c = BLACK);
26      void DrawContourList(ctrAttribDLList& list,
27                           const colour c = BLACK);
28  };
```

Der Konstruktor ist identisch mit dem schon aus der Klasse `eltecBaseX` bekannten Konstruktor. Neu hinzugekommen sind die folgenden Methoden:

1. Mit `DrawParallelogramm()` kann ein Parallelogramm mit zwei horizontalen Seiten in das Fenster eingezeichnet werden (siehe Abb. 8.7).
2. Die Methode `DrawCross()` zeichnet an den übergebenen Punktkoordinaten ein Kreuz, das aus Linien der übergebenen Länge geformt wird.
3. Die beiden Methoden `DrawContour()` geben eine komplette Kontur im Fenster aus, indem sie die einzelnen Konturpunkte mit Linien verbinden.
4. Mit der Methode `DrawContourList()` können schließlich auch Listen von Konturen in einem einzigen Aufruf in ein Fenster gezeichnet werden.

Es können prinzipiell beliebig viele Instanzen der beiden vorgestellten Klassen und damit Fenster von einer Anwendung erzeugt werden. Die Ausführung der Anwendung wird abgebrochen sobald ein Tastatur-Event in einem der Fenster erscheint.

8.7 Die Bibliothek libElt_boards.a

Zur vollständigen Schnittstelle der Bibliothek `libElt_boards.a` gehören folgende Dateien und Klassen:

- eltbase.H

  Klasse `eltecBase` mit globalen Variablen und Methoden.

- sbs2vme.H

  Klasse `vmeDevice` zur allgemeinen Ansteuerung des VME-Buses.

  Klasse `sbs2vmeCfg` zur Konfiguration des SBus VME-Bus Adapters.

- ipp.H

  Klasse `eltecIpp` zur Konfiguration der IPP Karte.

- thin.H

  Klasse `eltecThin` zur Konfiguration des THIN Karte.

- vect.H

  Klasse `eltecVect` zur Konfiguration des VECT Karte.

- eltec.H

  Klasse `eltecSystem`, die je eine Instanz der Klassen `eltecIpp`, `eltecThin` und `eltecVect` zur konsistenten Konfiguration des Gesamtsystems beinhaltet.

Diese Bibliothek ist nur für die Architekturen SUN4SOL2 und SUNMP vorhanden, da sie wie bereits erwähnt nur Software zur direkten Hardwareansteuerung beinhaltet und die Hardware nur über robosun1 verfügbar ist.

8.8 Die Bibliothek libElt_misc.a

Zur vollständigen Schnittstelle der Bibliothek libElt_misc.a gehören folgende Dateien und Klassen:

- eltbase.H

  Klasse `eltecBase` mit globalen Variablen und Methoden.

- contours.H

  Klassen `ctrPoint`, `ctrPointDLList`, `ctrAttrib`, `ctrAttribDLList`.

- database.H

  Klasse `eltecBaseDatabase`, `eltecDatabase`, `eltecSubDatabase` mit Variablen und Methoden aller Datenbanken.

  Klassen `db[Linear|Raster|LinRast][_[Stereo][Join]]`.

- limitreq.H

  Klasse `eltecBaseLimit` mit Methoden aller Attributsspezifikationen.

  Klassen `chk(Not|([Stereo][Mask][MinLen|MaxLen|Length]))`.

- requestdb.H

  Klasse `eltecBaseRequest`, `eltecLinearRequest`, `eltecRasterRequest` mit Variablen und Methoden aller Bereichsspezifikationen

  Klassen `req[Total|Rectangle|Parallelogramm]`.

- Xcontours.H

  Klassen `eltecBaseX`, `Xcontours`.

- convolver.H

  Klassen `eltecConvolver`, `eltecConvolverMatrix`.

Diese Bibliothek ist zur Zeit für die Architekturen HPPA, MASPAR (Front-End), SUN4, SUN4SOL2 und SUNMP. Für LINUX und SGI5 wird die Software verfügbar sobald neuere Versionen von `gcc` und `libg++` installiert sind.

8.9 Beispiele

8.9.1 Hardware, Datenbank und Visualisierung in einem Programm

Das erste Beispiel (siehe Prog. 8.12) soll zeigen wie die Hardware initialisiert, eine Datenbank instanziiert und verschiedene einfache Anfragen gemacht werden können um anschließend die verschiedenen Ergebnisse in Fenstern zu visualisieren.

Die notwendigen Header-Dateien zur Nutzung der vorhandenen Möglichkeiten sind, wie auch in Prog. 8.12 zu sehen, folgendermaßen:

- Hardware: `Elt/eltec.H`
- Datenbanken: `Elt/database.H`
- Visualisierung: `Elt/Xcontours.H`

Die Hardware wird hier nur mit den Default-Werten initialisiert (siehe Prog. 8.12, Zeile 13). Es können natürlich jederzeit andere Werte bei der Instanziierung verwendet werden oder auch nachträglich verändert werden.

Als Datenbanktyp wird hier die Klasse `dbLinRast_Join` verwendet, d.h. es existiert ein Linear- und ein Raster-Index. Zusätzliche werden Nachbarkonturen durch die Vorverarbeitung miteinander verbunden (siehe Prog. 8.12, Zeile 16). Die verwendeten Parameter für die Skalierung des Raster-Index dürften für die meisten Anwendungen ausreichend sein.

Es werden zwei rechteckige Bereichsanfragen generiert, die die obere bzw. untere Hälfte des Bildes beinhalten und eine Anfrage über den gesamten Bildbereich (siehe Prog. 8.12, Zeile 19ff).

Als weitere Auswahlkriterien wird der Wertebereich der gültigen Attribute eingeschränkt durch Instanzen der Klassen `chkMask` und `chkMaskMinLen` (siehe Prog. 8.12, Zeile 24f). Für die Auswahl nur vertikaler Konturen sind hier beispielsweise in der Bitmaske die Bitnummern 7, 8, 23 und 24 gesetzt.

Die Fenster zur Visualisierung werden auf einmal instanziiert und anschließend je nach Inhalt umbenannt (siehe Prog. 8.12, Zeile 28).

Programm 8.12: Anwendung von Hardware, Datenbanken und Visualisierung.

```
 1  #include <iostream.h>
 2  #include <Elt/eltec.H>
 3  #include <Elt/database.H>
 4  #include <Elt/Xcontours.H>
 5
 6  int main(int argc, char *argv[])
 7  {
 8    // image - parameters
 9    int w = eltecSystem::DFLT_WIDTH;
10    int h = eltecSystem::DFLT_HEIGHT;
11
12    // initialisation of hardware
13    eltecSystem Eltec;
14
15    // contour database
16    dbLinRast_Join CtrDB(w, h, 4, 3);
17
18    // specify areas
19    reqTotal total;
20    reqRectangle upper(0, 0, w, h/2);
21    reqRectangle lower(0, h/2, w, h);
22
23    // specify attributes
24    chkMask       vertical(0x01800180), horizontal(0x80018001);
25    chkMaskMinLen light2dark_min20(0xffff0000, 20);
26
27    // open windows
28    Xcontours *win = new Xcontours[4](w, h, „Upper vertical“);
29    win[1].Rename(„Lower vertical“);
30    win[2].Rename(„Upper light-dark“);
31    win[3].Rename(„Lower horizontal“);
32
33    // mainloop
34    while (1)
35    {
36      if (CtrDB.getFrame(Eltec) >= 0)  // read contours from HW
37      {
38        register ctrAttribDLList allCtr, Ctr[4];
39
40        // build indices
41        CtrDB.build();
42
43        // get all available contours
44        CtrDB.request(allCtr, total);
45
46        // some combined requests
47        CtrDB.request(Ctr[0], upper, vertical);
48        CtrDB.request(Ctr[1], lower, vertical);
49        CtrDB.request(Ctr[2], upper, light2dark_min20);
50        CtrDB.request(Ctr[3], lower, horizontal);
51
```

Programm 8.12: Anwendung von Hardware, Datenbanken und Visualisierung.

```
52          // draw contours into windows
53          for (int i = 0; i < 4; i++)
54          {
55            win[i].ClearBitmap();
56            win[i].DrawContourList(allCtr);
57            win[i].DrawContourList(Ctr[i], eltecBaseX::GREEN);
58            win[i].ShowBitmap();
59          }
60          Xcontours::workOnEvents();
61        }
62        else
63          cerr << „Contour Buffer Overflow“ << endl;
64      }
65    }
```

In der Hauptschleife werden immer zuerst die neuen Konturen eingelesen (siehe Prog. 8.12, Zeile 36) und geprüft ob dabei ein Fehler in Form eines Konturpufferüberlaufs aufgetreten ist. Falls alles in Ordnung ist wird die Vorverarbeitung angestoßen (siehe Prog. 8.12, Zeile 41), werden Anfragen gestellt (siehe Prog. 8.12, Zeile 44ff) und deren Ergebnisse visualisiert (siehe Prog. 8.12, Zeile 53ff).

8.9.2 Aufnahme und Speicherung einer Bildsequenz auf Datei

Da die vorhandene Hardware nur für genau einen Benutzer gleichzeitig verwendbar ist, kann es von Vorteil sein eine Bildsequenz von Rohdaten aufzunehmen um unabhängig von der Hardware arbeiten zu können. Wie eine solche Aufnahme programmiert werden kann ist in Prog. 8.13 zu sehen. Aufgrund der langsamen Dateiausgabe muß, um fortlaufende Frames zu bekommen, zuerst ein Feld von Konturdatenbanken allokiert werden (siehe Prog. 8.13, Zeile 15). Dann wird das Feld vollständig mit Daten gefüllt (siehe Prog. 8.13, Zeile 18ff), bevor die Daten in Dateien geschrieben werden (siehe Prog. 8.13, Zeile 27ff). Diese Programm schreibt die Frames für die Zeitdauer einer Sekunde in die Dateien `movie.XXX.ctr` im aktuellen Arbeitsverzeichnis, wobei `XXX` die Frame-Nummer ist. Falls in Prog. 8.13, Zeile 29 als zweiter Parameter eine 2 übergeben würde, hießen die Dateien `movie.XX.ctr`, d.h. durch diesen Parameter wird die Anzahl der Stellen für die Frame-Nummer bestimmt.

8.9.3 Einlesen einer Bildsequenz von Datei

Die durch Prog. 8.13 erzeugten Dateien werden durch Prog. 8.14 eingelesen und in einem Fenster visualisiert. Da hier das Einlesen der Dateien nicht zeitkritisch ist wird kein Feld von Konturdatenbanken benötigt, sondern es werden nacheinander die Frames von Datei eingelesen (siehe Prog. 8.14, Zeile 23), die Vorverarbeitung durchgeführt (siehe Prog. 8.14, Zeile 24) und anschließend die durch eine Datenbankanfrage (siehe Prog. 8.14, Zeile 28) erhaltenen Konturen visualisiert (siehe Prog. 8.14, Zeile 31ff). In Prog. 8.14, Zeile 23 wird keine Frame-Nummer als Parameter übergeben, da diese Methode intern, bei 0 beginnend, die Frames zählt und den korrekten Nachfolgeframe von Datei lädt. Falls die explizite Angabe der Frame-Nummer notwendig sein sollte, muß auf die Methode `read()` zurückgegriffen werden. In Prog. 8.14 müßte dann die Zeile 23 durch die in Prog. 8.15: abgebildete Zeile ersetzt werden.

Programm 8.13: Aufnahme und Speicherung einer Bildsequenz auf Datei.

```
 1  #include <iostream.h>
 2  #include <Elt/eltec.H>
 3  #include <Elt/database.H>
 4
 5  int main(int argc, char *argv[])
 6  {
 7    // number of frames to record
 8    int maxframes = 25;
 9    int frame;
10
11    // initialisation of hardware
12    eltecSystem Eltec;
13
14    // contour database
15    dbLinear *CtrDB = new dbLinear[maxframes];
16
17    // read contours from HW
18    for (frame = 0; frame < maxframes; frame++)
19    {
20      CtrDB[frame].getFrame(Eltec);
21      cout << „r“;
22      cout.flush();
23    }
24    cout << endl;
25
26    // write to files
27    for (frame = 0; frame < maxframes; frame++)
28    {
29      CtrDB[frame].write(„movie“, 3);
30      cout << „w“;
31      cout.flush();
32    }
33    cout << endl;
34  }
```

8.9.4 Suche nach antiparallelen Konturen

Um etwa eine weiße Linie auf dunklem Hintergrund zu erkennen müssen antiparallele Konturen gefunden werden. Diese weiße Linie besteht im Idealfall aus zwei parallelen Konturen. Eine Kontur wird dabei durch einen dunkel-hell Kontrast erzeugt und die andere durch einen hell-dunkel Kontrast. Damit haben die beiden Konturen eine entgegengesetzte Richtung (siehe Kapitel 8.2), d.h. sie sind antiparallel [6]. Mit der in Prog. 8.16 abgebildeten Funktion können zu einer bekannten Kontur, die durch einen dunkel-hell Kontrast erzeugt wurde, antiparallele Konturen gesucht werden.

Dazu werden immer für ein Paar aufeinanderfolgender Konturpunkte der übergebenen Kontur parallelogrammförmige Bereichsanfragen generiert (siehe Prog. 8.16, Zeile 35). Zur Generierung der Bereichsanfrage werden die Koordinaten der beteiligten Konturpunkte verwendet und als Breite des Parallelogramms ein der Anwendung und der Breite der Linie angemessener Wert bestimmt. Die Menge möglicher Kandidaten wird durch die Angabe einer Richtungsmaske für diesen Bereich weiter einge-

Programm 8.14: Einlesen einer Bildsequenz von Datei.

```
 1  #include <iostream.h>
 2  #include <Elt/eltec.H>
 3  #include <Elt/database.H>
 4  #include <Elt/Xcontours.H>
 5
 6  int main(int argc, char *argv[])
 7  {
 8    // number of frames to record
 9    int maxframes = 25;
10    int frame;
11
12    // contour database
13    dbLinear CtrDB;
14
15    // visualisation
16    Xcontours win(eltecSystem::DFLT_WIDTH,
17                  eltecSystem::DFLT_HEIGHT, „movie“);
18
19    // read contours from files and visualize
20    for (frame = 0; frame < maxframes; frame++)
21    {
22      // read contours from file
23      CtrDB.getFrame(„movie“, 3);
24      CtrDB.build();
25
26      // get contours
27      ctrAttribDLList all;
28      CtrDB.request(all, reqTotal());
29
30      // visualize contours
31      win.ClearBitmap();
32      win.DrawContourList(all);
33      win.ShowBitmap();
34      Xcontours::workOnEvents();
35
36      cout << „.“;
37      cout.flush();
38    }
39    cout << endl;
40  }
```

Programm 8.15: Verwendung der Methode `read()` *beim Einlesen von Datei.*

```
1  CtrDB.read(„movie“, 3, frame);
```

schränkt. Zur Bestimmung der Richtungsmaske werden für jeden der beiden bei einer einzelnen Anfrage beteiligten Konturpunkte eine Winkelmaske bestimmt, indem die im Konturpunkt vorhandene Richtung in der Bitmaske gesetzt wird (siehe Prog. 8.16, Zeile 28). Als Toleranzbereich wird jeweils das linke und rechte Bit daneben ebenfalls gesetzt (siehe Prog. 8.16, Zeile 29). Schließlich wird die Maske um 180^0 verschoben um wirklich antiparallel Konturen zu finden (siehe Prog. 8.16, Zeile 30).

Programm 8.16: Suche nach antiparallelen Konturen.

```
 1  ctrAttribDLList searchAntiParallel(register ctrAttrib *ctr)
 2  {
 3    ctrAttribDLList result;
 4    ctrPointDLList *actPointList = &(ctr->pts);
 5
 6    register u_int AngleMask1, AngleMask2;
 7    register short x1, y1, d1, x2, y2, d2;
 8
 9    Pix i = actPointList->first();
10
11    // mask of first point, add tolerance, make antiparallel mask
12    AngleMask2 = (*actPointList)(i)->angle2mask();
13    AngleMask2 |= (AngleMask2 << 1) | (AngleMask2 >> 1);
14    AngleMask2 = (AngleMask2 << 16) | (AngleMask2 >> 16);
15    x2 = (*actPointList)(i)->x;
16    y2 = (*actPointList)(i)->y;
17    d2 = 50;
18
19    register bool firstRequest = true;
20    for (actPointList->next(i); i != NULL; actPointList->next(i))
21    {
22      AngleMask1 = AngleMask2;
23      x1         = x2;
24      y1         = y2;
25      d1         = d2;
26
27      // mask of next point, add tolerance, make antiparallel mask
28      AngleMask2 = (*actPointList)(i)->angle2mask();
29      AngleMask2 |= (AngleMask2 << 1) | (AngleMask2 >> 1);
30      AngleMask2 = (AngleMask2 << 16) | (AngleMask2 >> 16);
31      x2 = (*actPointList)(i)->x;
32      y2 = (*actPointList)(i)->y;
33      d2 = 50;
34      CtrDB.request(result,
35                    reqParallelogramm(x1, y1, x2, y2, max(d1, d2)),
36                    chkMask(AngleMask1 | AngleMask2),
37                    !firstRequest);
38      firstRequest = false;
39    }
40    return result;
41  }
```

Da im Idealfall für jeden Teilabschnitt dieselbe antiparallele Kontur gefunden wird verwendet diese Funktion die Möglichkeit Anfragen zu gruppieren, um Duplikate in der Ergebnismenge auszuschließen (siehe Prog. 8.16, Zeile 19, 37f).

Zu beachten ist, daß diese Funktion nur mögliche Kandidaten von antiparallelen Konturen liefert. Probleme gibt es sobald mehrere Kandidaten auftreten, denn welches ist die richtige Kontur, oder sind alle richtig weil die Kontur zerrissen ist?

8.10 Messungen

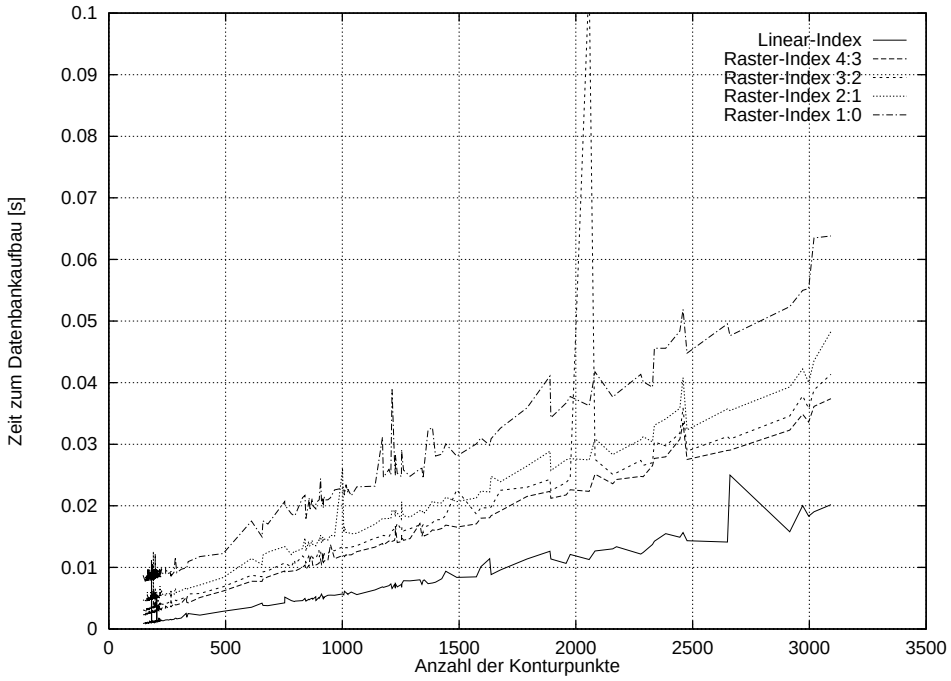


Abbildung 8.9: Zeitmessung ohne Vorverarbeitungsschritte.

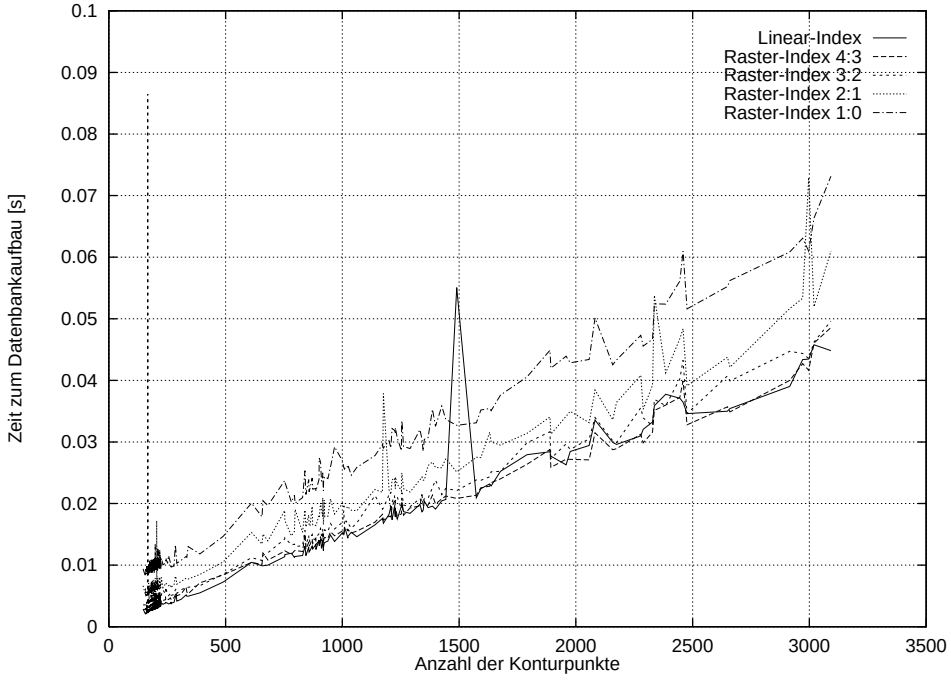


Abbildung 8.10: Zeitmessung mit Verbindung von Nachbarkonturen.

Um ein Gefühl für die Einstellung von Parametern zu bekommen sind in Abb. 8.9 und Abb. 8.10 Schaubilder aufgetragen, in denen die Zeit zum Aufbau des jeweiligen Index über der Anzahl der vektorisierten Konturpunkte aufgetragen ist. Die Messungen wurden ausschließlich auf robosun1 durchgeführt.

Die Meßergebnisse der beiden Klassen `dbLinear` und `dbRaster` sind in Abb. 8.9 abgebildet, wobei für alle Messungen dieselbe auf Datei gespeicherte Bildsequenz verwendet wurde. Für `dbRaster` wurden die Skalierungsfaktoren verschieden gewählt um eine Vergleichsmöglichkeit zwischen einigen möglichen Kombinationen zu erhalten. Die Skalierung ist aus der Kurvenbezeichnung abzulesen, wobei eine Angabe von 4:3 für eine Skalierung in x-Richtung von $2^4 = 16$ und in y-Richtung $2^3 = 8$ steht.

In Abb. 8.10 wurde wieder unter Verwendung derselben Bildsequenz eine Messung über das Zeitverhalten durchgeführt, wenn die Verbindung von Nachbarkonturen als Vorverarbeitungsschritt hinzukommt.

Wie zu erwarten war ist der Aufbau des Linear-Index etwa doppelt so schnell wie der des Raster-Index. Bei der Verwendung einer Datenbank, in der beide Indizes verwendet werden, müssen die Zeiten aus Abb. 8.9 in etwa addiert werden. Dies gilt allerdings nicht für die Zeiten aus Abb. 8.10, da sonst die Zeiten für das Zusammenfügen von Nachbarkonturen zweifach berechnet würde.

8.11 Programme

Es existieren einige Programme die unter anderem die einzelnen Karten separat konfigurieren können, bzw. Bildsequenzen speichern oder laden und anzeigen können.

Ausschließlich für die Architekturen SUN4SOL2 und SUNMP gibt es folgende Programme:

- `Eltipp`
  Konfiguration einiger Parameter der IPP-Karte. Ermöglicht außerdem das Abspeichern eines Grauwertbildes im pgm-Format.
- `Eltthin`
  Konfiguration der Parameter der THIN-Karte.
- `Eltvect`
  Konfiguration der Parameter der VECT-Karte.
- `Eltmain`
  Konfiguration des Gesamtsystems mit Visualisierung der Konturen und der Möglichkeit Bildsequenzen abzuspeichern.
- `Eltfollow`
  Anwendung der ELTEC-Hardware zur Linienverfolgung. Die Roboter können über Kommandozeilen-Option direkt angesteuert werden. Das Programm kann auch nur auf einer Bildsequenz arbeiten, sollte dann aber aus verständlichen Gründen ohne die Option zur Ansteuerung der Roboter aufgerufen werden.

Insbesondere können während des Ablaufs von `Eltmain` mit den drei zuerst genannten Programmen zur Konfiguration der Hardware Parameter verändert werden - allerdings ohne Gewähr.

Für die Architekturen HPPA, MASPAR (Front-End), SUN4, SUN4SOL2 und SUNMP gibt es das folgende Programm:

- `Eltshow`
  Einlesen und Anzeigen einer Bildsequenz.

über den genauen Aufruf aller genannten Programme kann man sich durch einen Aufruf mit der Option -h informieren.

8.12Verwendung der Sourcen

Teilprojektname: `Elt` (Eltec Interface)
aktuelle Version: `Elt2-1`
bei nutzenden TPR: Makefilevariable `BASESON` um die aktuelle Version erweitern

Library: `libElt_misc.a`
Beschreibung: Routinen zur Verarbeitung und Visualisierung von Konturen
Architekturen: `SUN4SOL2 SUNMP`
Includes: `#include "Elt/eltec.H"`
Linkoptionen: `-lElt_misc.a -lXt -lX11`

Library: `libElt_boards.a`
Beschreibung: Routinen zum Ansprechen der Eltec Hardware
Architekturen: `HPPA MASPAR(Front-End) SUN4 SUN4SOL2 SUNMP`
Includes:
```
#include "Elt/contours.H"
#include "Elt/Xcontours.H"
```
Linkoptionen: `-lElt_boards -lm -lElt_misc.a`

Programme: `Eltipp`
Beschreibung: Konfiguration IPP-Karte, Abspeichern Grauwertbilder.
Architekturen: `SUN4SOL2 SUNMP`

Programme: `Eltthin`
Beschreibung: Konfiguration THIN-Karte.
Architekturen: `SUN4SOL2 SUNMP`

Programme: `Eltvect`
Beschreibung: Konfiguration VECT-Karte.
Architekturen: `SUN4SOL2 SUNMP`

Programme: `Eltmain`
Beschreibung: Konfiguration Gesamtsystem, Abspeichern Konturbilder und Visualisierung. Beendet wird es mit ctrl-c oder Tastendruck in eines der Fenster.
Architekturen: `SUN4SOL2 SUNMP`

Programme: `Eltfollow`
Beschreibung: Linienverfolgungs-Demo. Beendet wird es mit ctrl-c.
Architekturen: `SUN4SOL2 SUNMP`

Programme: `Eltshow`
Beschreibung: Anzeigen von Konturbildern.
Architekturen: `HPPA MASPAR(Front-End) SUN4 SUN4SOL2 SUNMP`

8.13 Literatur

[1] User‘s Manual PT-SBS915 SBus-to-VMEbus Adapter, Document No 106A0183, Performance Technologies Inc., New York, 1993.

[2] Hardware Manual Image Processing Port, Rev. 2a, Eltec Elektronik, Mainz, 1991.

[3] Hardware Manual THINEDGE-Processor for Contour Matching, Rev. 1a, ELTEC Elektronik GmbH, Mainz, 1991.

[4] Hardware Manual VECTOR-Processor for Contour Matching, Rev. 1a, ELTEC Elektronik GmbH, Mainz, 1991.

[5] F. Faigle, *Entwicklung einer Echtzeitdatenbank für Bildkonturen auf einem Transputersystem zur autonomen Fahrzeugführung*, Diplomarbeit, FH Esslingen, Technische Informatik, 1992.

[6] M. Sommerau, *Echtzeitinterpretation von Verkehrsszenen anhand extrahierter Konturen*, Diplomarbeit Nr. 1036, Universität Stuttgart, Fakultät Informatik, 1993.

Kapitel 9

Die SUN Framegrabber Routinen

Michael Vogt, Günter Mamier

Dieses Kapitel beschreibt die Benutzung und die Eigenschaften der SUN-Video Framegrabber auf den Robosuns. Seit Dezember ‘94 existiert eine neuere Version der XIL Libraries. Aufgrund dieser neuen Version, die auf allen drei robosuns eingespielt wurde, ist es nun möglich, die angebotenen Funktionen wesentlich besser auszunutzen als bisher. Unter anderem gewinnt man die Möglichkeit, Zeitstempel anzulegen. Die Grabzeiten haben sich in bestimmten Fällen wesentlich verbessert.
Neben dieser Dokumentation existieren eine Reihe von Manual Pages für alle SFG spezifischen Funktionen, Variablen und Typen.

Was ist neu?

In der Version 2-1

Es besteht nun die Möglichkeit, direkt nach dem Graben eine beliebige NxMFaltung auf dem Bild durchzuführen. Außerdem wurden ein paar kleinere Fehler behoben.

In der Version 2-0

Gegenüber der 1-3 Version der SFG Routinen hat sich der Datentyp beim Graben von Bildern verändert. Alle Programme, die bisher SFG 1-3 verwenden und künftig SFG 2-x verwenden sollen (das ist sehr empfehlenswert), müssen leider minimal modifiziert werden. Ein erneutes Übersetzen und Binden mit der neuen SFG 2-0 Bibliothek reicht leider nicht aus.

Es wird dringend empfohlen, die Umstellung auf 2-x vorzunehmen, da die bisherigen Routinen zu sich verlängernden Grabzeiten führen und unter bestimmten Voraussetzungen keine aktuellen Bilder garantieren.

9.1 Initialisierung

Bevor irgendwelche Bilddaten ausgelesen werden können muß der Framegrabber initialisiert werden. Diese Initialisierung ist einmal für beliebig viele Grabs durchzuführen. Ist der Framegrabber durch einen Benutzer initialisiert, so steht er ausschließlich diesem Benutzer zur Verfügung, bis er den Framegrabber explizit wieder freigibt oder das Program terminiert (implizite Freigabe). Daraus ergibt sich die Folgerung, daß Programme den Framegrabber schließen sollten sobald der Bildaufnahmeteil durchlaufen ist. Es gab bereits öfters Beschwerden über Programme die dies unterließen.

Die Initialisierung erfolgt mit dem Befehl:

int SFG_init(scalefact, port, farbe, &breite, &höhe);

Die Parameter bedeuten:

int scalefact — Skalierungsfaktor, der die Größe der zu grabenden Bilder angibt. Der Skalierungsfaktor gibt direkt an, das wievielte Pixel in einer Zeile bzw., die wievielte Zeile digitalisiert werden soll. Es ergibt sich bei PAL Format (unser Format):

1 == 768x576 (Originalbild, PAL Format)
2 == 384x288
3 == 256x192
4 == 192x144
5 == 153x115.

Hinweis: Zeilenverschachtelte Stereobilder können nur mit ungeraden Faktoren gegrabt werden!

int port — Nummer des Eingangsports (0, 1 oder 2). Bei uns sind normalerweise immer die ports 1 belegt (über den Sony Kreuzschienenverteiler). Zusätzlich liegt manchmal an Port 0 ein S-Videosignal an (zur Zeit an robosun3 der Farb-Stereo-Mixer)

int farbe — SFG_GRAY == Graben in Graustufen,
SFG_COLOR == Graben in 24 Bit Farbe,
SFG_DITHER == Graben von geditherten Farbbildern
(SFG_DITHER ist nur zur Darstellung und nicht für die Bildverarbeitung geeignet).

int breite — Breite des zu grabenden Bildes (wird zurückgeliefert).

int höhe — Höhe des zu grabenden Bildes (wird zurückgeliefert).

Die Funktion liefert im Erfolgsfall den Wert 0 zurück, im Fehlerfall einen negativen Fehlercode: 0 bei Erfolg; -1 Fehler beim Öffnen der XIL library; -2 Fehler beim Öffnen des Ports bzw. Port belegt; -3 Ungültige Argumente.

9.2 Graben von Bildern

Zur Zeit stehen drei verschieden Routinen zum Graben von Bildern zur Verfügung. Sie können innerhalb eines SFG_init -- SFG_close Paares beliebig oft und in beliebiger Kombination aufgerufen werden. Welche zu benutzen ist, ist im jeweiligen Fall auf Grund der gegebenen Bedingungen zu entscheiden. Alle drei Funktionen liefern das Bild in der Struktur SFG_image zurück, auf die z.B. in der folgenden Art zugegriffen werden kann:

Programm 9.1: Kopieren von Bildinformation (ineffizient)

```
 1  SFG_image bild;
 2
 3  SFG_getFrameFast(&bild);
 4  for(j=0;j<h;j++){
 5      for(i=0;i<w;i++){
 6          Blue[i][j]  = (unsigned char)* bild.storage.byte.data++;
 7          Green[i][j] = (unsigned char)* bild.storage.byte.data++;
 8          Red[i][j]   = (unsigned char)* bild.storage.byte.data++;
 9      }
10  }
```

Wesentlich eleganter kann man auf den Bildinhalt zugreifen, indem man sich eine Zeigerstruktur aufbaut, die in den Bilddatenbereich hineinzeigt. Es ist vorgesehen, hierfür eine eigene Bibliothek aufzubauen. Vorerst kann man sich mit etwa folgender Konstruktion für Farbbilder behelfen:

Programm 9.2: Einfacher Zugriff auf Bildinformation

```
 1  typedef struct _uc_bgr {
 2      unsigned char b;
 3      unsigned char g;
 4      unsigned char r;
 5  } uc_bgr;
 6
 7  {
 8      SFG_image bild;
 9      uc_bgr **image;
10
11      /* Initialisierung */
12      SFG_init(scalefact, port, farbe, &breite, &hoehe);
13      image = (uc_bgr **) malloc(hoehe * sizeof(uc_bgr *));
14
15      while (verarbeitungsschleife) {
16          /* Bild graben */
17          SFG_getFrameFast(&bild);
18          for (i=0; i<hoehe; i++)
19              image[i] = ((uc_bgr *) bild.storage.byte.data)
20                         + i*breite;
21
22          /* Zugriff: image[y][x].r, image[y][x].g, image[y][x].b */
23          ....
24      }
```

Es kann so recht einfach auf jedes beliebige Pixel (RGB Werte) zugegriffen werden, ohne daß hierzu das gesamte Bild umkopiert werden muß, und ohne bei jedem Zugriff eine umständliche Berechnung der richtigen Adresse durchzuführen. Als Warnung bleibt zu erwähnen, daß sich der Bildspeicherbereich nach jedem Aufruf von SFG_getFrame... potentiell ändern kann. Es ist daher angebracht, die obige Zeigerstruktur nach jedem Grab neu aufzubauen. Es kann jedoch darauf verzichtet werden, wenn durch einen Vergleich der Basisadresse in bild.storage.byte.data sichergestellt ist, daß sich der Bereich nicht verschoben hat.

Im Kapitel über Bildformate wird der Zugriff ebenfalls beschrieben.

Gegenüber der Version 1-3 der SFG Routinen hat sich der Aufrufparameter für die nachfolgenden Grab-Funktionen geändert. Es handelt sich nun um eine Struktur, die neben den XIL-Bilddaten (früherer Parameter) weitere Daten enthält. Die Struktur hat folgenden Aufbau:

Programm 9.3: Definition von SFG_image

```
 1  typedef struct
 2  {
 3      XilMemoryStorage storage;   /* Struktur, die (unter anderem) die
 4                                     Daten des gegrabten Bildes
 5                                     enthaelt */
 6      double timestamp;           /* Grab-Zeitpunkt [s] */
 7      int width;                  /* Breite des Bildes */
 8      int height;                 /* Hoehe des Bildes */
 9      SFG_color_mode color;       /* Farbmodus des Bildes */
10      int frame_no;               /* Laufende Nummer des Bildes */
11  } SFG_image;
```

Besonders interessant sind der Zeitstempel (timestamp) und die fortlaufende Bildnummer. Der Zeitstempel gibt die Systemzeit in Sekunden seit dem 1.1.1970 (GMT) an und besitzt eine Auflösung von 1 ms. Er ist aus dem Zeitstempel abgeleitet, den der Framegrabber jedem Bild zuordnet. Leider kann dieser Zeitpunkt um bis zu 40 ms neben dem tatsächlichen Zeitpunkt liegen. Ursache hierfür sind noch nicht behobene Fehler der Software bzw. Hardware von SUN. Möglicherweise bringt Solaris 2.4 hier eine Besserung.

Die fortlaufende Bildnummer (frame_no) gibt die Nummer des anliegenden Kamerabildes an, das seit Initialisierung des Framegrabbers registriert wurde.

ACHTUNG: Dies ist nicht eine laufende Numerierung der gegrabten Bilder sondern eine Zuordnung zwischen gegrabtem Bild und Kamerasignal. Da das Graben von großen Farbbildern beispielsweise länger als 40 ms dauert, erhält man hier nicht jede mögliche Bildnummer, was auch ein Zeichen dafür ist, daß man nicht jedes einzelne Bild graben kann. Die Bildnummer wird auch weiterhin von allen SFG_getFrame... Routinen als Ergebnis zurückgeliefert.

9.2.1 Schnelles Graben

Die Routine

int SFG_getFrameFast(SFG_image *bild);

grabt Bilder möglichst schnell. Sie nutzt dabei den internen FIFO Buffer des Framegrabbers. Durch diesen Buffer ist aber nicht mehr gewährleistet, daß das gegrabte Bild den aktuellen Gegebenheiten entspricht, es kann veraltet sein. Um ein relativ aktuelles Bild zu erhalten muß deshalb evtl. mehrmals direkt hintereinander die Routine aufgerufen werden. Daher ist diese Routine nicht empfehlenswert, wenn für das rufende Program ein Bild, das mehrere Millisekunden bis Minuten alt ist nicht akzeptabel ist.

9.2.2 Sicheres Graben

Die Routine

int SFG_getFrameSave(SFG_image *bild);

nutzt den internen Buffer des Framegrabbers nicht. Sie stellt im Gegenteil sicher, daß das Bild immer neu gegrabt wird bevor es an das rufende Program gegeben wird. Dies wird erreicht, indem der FIFO Buffer des Framegrabbers mit einem speziellen Befehl geleert wird. Ein neues Bild kann erst dann wieder gegrabt werden, sobald der gesamte FIFO wieder gefüllt ist. Die Zeitdauer für diesen Vorgang hängt von der Länge des FIFO ab (siehe auch Abschnitt über weitere Parameter) und von der fest vorgegebenen Bildrate von 40 ms. Gegenüber SFG_getFrameFast nimmt man einen deutlichen Zeitnachteil in Kauf.

9.2.3 Intelligentes Graben

Die Zwischenform

int SFG_getFrame(SFG_image *bild);

versucht abzuschätzen welches Bild im Buffer des Framegrabbers der aktuellen Situation entspricht und gibt dieses Bild zurück. Dieses ist aufgrund der neuen XIL-Bibliothek und der neuen Grab Routinen auf jeden Fall aktuell (d.h.: sicher), kann aber u.U. länger dauern als SFG_getFrameSave. In der Regel sollte diese Funktion gegenüber allen anderen bevorzugt werden. Die Funktionsweise ist folgende:

Beim Aufruf von SFG_getFrame wird die aktuelle Systemzeit bestimmt. Von dieser Systemzeit wird ein tolerierbares Bildalter subtrahiert (siehe SFG_setMaxdelay) um das älteste akzeptierbare Bild festzulegen. Anschließend werden aus dem FIFO des Framegrabbers so lange Bilder entnommen, bis der jeweils mitgelieferte Zeitstempel höchstens so alt ist, wie der zuvor festgelegte Zeitpunkt. Die Geschwindigkeit dieser Routine hängt also in erster Linie davon ab, wann sie zuletzt aufgerufen wurde, d.h. wie lange die Bearbeitung des zuvor gegrabten Bildes gedauert hat. Ist die Bearbeitung sehr schnell und akzeptiert man Bilder, die z.B. maximal 40 ms alt sein dürfen, dann wird möglicherweise jedes einzelne Bild ohne Wartezeiten bearbeitet. Weitere Einflußfaktoren sind die Tiefe des FIFO und das einstellbare akzeptierte Bildalter (s.u.).

9.3 Weitere Funktionen

Um die Framegrabber Hardware optimal an das jeweilige Programm anzupassen, gibt es weitere Möglichkeiten, verschiedene Parameter zu ändern.

9.3.1 Länge des FIFO

Mit der Funktion

int SFG_setMaxbuffers(int maxbuffers);

kann die maximale Größe des Bildspeicher FIFOs eingestellt werden. Diese maximale Größe stellt eine obere Schranke dar. Die tatsächlich verwendete Anzahl hängt ab vom verfügbaren Speicherplatz auf dem Framegrabber und von der aktuellen Größe der zu grabenden Bilder. Die tatsächliche Größe soll von der Funktion zurückgegeben werden, was aber leider aufgrund eines Fehlers der XIL-Bibliothek nicht korrekt passiert. Hier wird regelmäßig der Wert 2 geliefert obwohl aus Versuchen hervorgegangen ist, daß deutlich andere Zahlen verwendet werden: Es gelten folgende Paarungen, angegeben in <Skalierungsfaktor>:<tatsächliche FIFO-Länge>, die für einen Maximalwert von 20 Speicherplätzen bestimmt wurden:

1:2, 2:5, 3:3, 4:5, 5:8, 6:12

Der Benutzer muß sich im Zweifelsfall also selbst überlegen, wieviele Speicherplätze bei einem vorgegebenen Maximalwert erzielt werden. Interessanterweise sind diese Angaben unabhängig davon, ob Farb- oder Grauwertbilder gegrabt werden. Der schlechte Wert für den Skalierungsfaktor 3 beruht offensichtlich ebenfalls auf Fehlern der Hardware oder der XIL-Bibliothek.

Als Default-Wert ist #include <xil/xil.h>nach Initialisierung des Framegrabbers die Maximalzahl von 2 FIFO Buffern eingestellt.

9.3.2 Maximales Bildalter

Mit der Funktion

void SFG_setMaxdelay(int maxdelay);

kann das maximale Alter von Bildern für die Grabroutine SFG_getFrame in Millisekunden eingestellt werden. Auf die anderen beiden Grabfunktionen hat sie keinen Einfluß. Als Default-Wert ist nach Initialisierung des Framegrabbers ein Wert von 20 ms eingestellt. Dies beruht auf der Beobachtung, daß der Fehler der Zeitstempel der Bilder in Ausnahmefällen bis zu 40 ms betragen kann. Für Anwender, die ganz sicher sein wollen, sollte hier also ein Wert von 0 ms oder sogar ein negativer Wert angegeben werden. Für Anwendungen mit wenig Bewegung im Bild reicht eventuell ein akzeptiertes Alter von bis zu 1000 ms aus. Dies muß jeder Anwender nach eigenem Ermessen selbst festlegen.

9.3.3 Automatisches Überspringen

Die Funktion

void SFG_setSkip(int skip);

bewirkt, daß der Framegrabber beim Digitalisieren grundsätzlich <skip> Bilder ausläßt. Dies hat den Vorteil, daß z.B. bei langsamen Anwendung das Zusammenspiel von maximal erlaubtem Alter und Anwendung der Funktion SFG_getFrame der Bildzugriff optimiert werden kann. Stellt man im Laufe der Anwendung anhand der Bildnummern fest, daß beispielsweise nur jedes vierte oder fünfte Bild geliefert wird, so ist es evtl. sinnvoll, des <skip> Wert auf drei zu setzen, damit nicht regelmäßig der FIFO des Framegrabbers gefüllt und unnötig leergelesen werden muß. Die Funktion kann (muß aber nicht) nach Initialisierung des Framegrabbers beliebig oft aufgerufen werden. Der eingestellte Wert bleibt jeweils bis zum nächsten Aufruf von SFG_setSkip aktiviert.

9.3.4 Automatischer Weißabgleich

Mit der Funktion

void SFG_setWB(float red_scale, float blue_scale);

kann eine automatische Weißkorrektur vorgenommen werden. Der Rot- und Blauanteil von Farbbildern wird bei jedem Graben automatisch mit den Werten red_scale und blue_scale multipliziert, falls sie ungleich 1.0 sind. Der hierdurch erzielte Wertebereich wird auf 0 bis 255 beschränkt (eventuell verliert man also relevante Information). Soll der automatische Weißabgleich ausgeschaltet werden, so müssen die Werte 1.0 angegeben werden. Nach der Initialisierung ist der automatische Weißabgleich nicht aktiv.

9.3.5 Automatische nxm Faltung

Durch die Funktion

void SFG_setConvolution(Xil_boolean active);

kann eine automatische NxM Faltung aller gegrabten Bilder aktiviert (active == TRUE) oder deaktiviert (active == FALSE) werden. Nach Initialisierung des Framegrabbers ist die Faltung deaktiviert. Der eingestellte Default-Faltungskern ist ein 3x3 Mittelwertfilter zur Glättung der Bilder. Wahlweise kann jedoch ein anderer Kern verwendet werden (z.B. Laplace), der jedoch vor Aufruf der Funktion zunächst durch

void SFG_setKernel(int width, int height, int key_x, int key_y, float *data);

erzeugt werden muß.

Durch SFG_setKernel kann der Standard-Faltungskern der SFG Routinen (Tiefpaß-Filter mit 3x3 Filtermaske) gegen einen anderen Kern ausgetauscht werden. Die Parameter width und height bestimmen die Größe des Filterkerns. key_x und key_y definieren das Zentrum des Kerns (0, 0 entspricht oben links). Beispielaufruf für die Definition eines Laplace Filters:

Programm 9.4: Aktivierung eines Laplace Filters

```
1  float laplace[9] = { 0, -1,  0,
2                       -1,  4, -1,
3                        0, -1,  0 };
4  SFG_setKernel(3, 3, 1, 1, laplace);.
```

9.4 Hilfsfunktionen

Für ein einfaches Arbeiten mit den SFG-Routinen stehen eine Reihe einfach zu bedienender Hilfsfunktionen zur Verfügung:

9.4.1 Bildinformation

Die Funktion

char *SFG_image_info(SFG_image *image);

liefert für ein gegebenes Bild einen kurzen Informations-String, der z.B. zu Protokollzwecken verwendet werden kann. Der Aufbau des Strings wird am besten an einem Beispiel klar:

 # frame [14] color captured on robosun3 by libSFG at Wed Jan 25 1995 20:16:02.864

Die Zahl in eckigen Klammern gibt die laufende Bildnummer seit der Initialisierung an. Der Zeitstempel am Ende hat eine Auflösung von Millisekunden. Alle Informationen werden aus der SFG_image Struktur abgeleitet, die beim Graben des Bildes gefüllt wird.

9.4.2 Bilder kopieren

Die Funktion

SFG_image *SFG_copy_image(SFG_image *image);

dupliziert eine gültige SFG_image Bildstruktur, allokiert Speicherbereich für die Bilddaten und kopiert effizient die Bilddaten in die neue Struktur. Sollen z.B. nacheinander mehrere Bilder aufgenommen werden, die erst nachträglich bearbeitet werden können, so ist ein Kopieren der Bilder mit dieser Funktion angebracht. Die Funktion erhält als Parameter einen Zeiger auf die bereits vorhandene Bildstruktur und liefert einen Zeiger auf eine neue Bildstruktur. (Anmerkung: die alte Bildstruktur bleibt weiterhin gültig).

9.4.3 Bilder löschen

Die Funktion

void SFG_free_image(SFG_image *image);

gibt den Speicherbereich eines Bildes, das mit SFG_copy_image dupliziert wurde, wieder frei. Nach Aufruf dieser Funktion darf nicht mehr auf die Daten dieses Bildes zugegriffen werden.

ACHTUNG: Es darf niemals eine Bildstruktur angegeben werden, die direkt von einem SFG_getFrame...() Aufruf stammt, da hierdurch die interne Speicherverwaltung des XIL unterlaufen wird (drohendes Resultat: segmentation fault).

9.4.4 Abspeichern von Bildern

Mit der Funktion

int SFG_save_pnm_file(FILE *fp, SFG_image *image);

wird ein Bild, das durch die Bildstruktur <image> übergeben wird in die Datei <fp> abgespeichert. Abhängig vom Bildtyp wird entweder ein PGM (P5) Grauwertbild oder ein PPM (P6) Farbbild geschrieben (siehe man pgm oder man ppm). Die Datei <fp> muß vor Aufruf der Funktion geöffnet werden und anschließend geschlossen werden. Die Funktion tut dies nicht selbstständig um größere Flexibilität zu gewährleisten.

9.4.5 Zugriff auf geditherte Bilder

Für den Zugriff auf geditherte Farbbilder, die sich ausschließlich zur Darstellung und nur eingeschränkt zur Bildverarbeitung eignen, wird eine spezielle Variable (SFG_ccube) exportiert, die die Farbwerte des verwendeten Farbwürfels enthält. Die Variable ist vom Typ SFG_rgb und erlaubt den Zugriff auf die RGB Komponenten der Pixel. Ein beispielhafter Zugriff würde etwa folgendermaßen aussehen:

Programm 9.5: Zugriff über color cube auf geditherte Bilder

```
1  SFG_getFrame(&image);
2  for(j=0;j<h;j++){
3      for(i=0;i<w;i++){
4              red[i][j] = SFG_ccube[image.storage.byte.data].r;
5              green[i][j] = SFG_ccube[image.storage.byte.data].g;
6              blue[i][j] = SFG_ccube[image.storage.byte.data++].b;
7      }
8  }
```

9.4.6 Zugriff auf XIL

Die SFG-Routinen basieren auf der XIL-Library von SUN. Normalerweise ist es nicht notwendig, XIL direkt anzusprechen, da alle notwendigen Operationen über SFG-Routinen nach oben weitergegeben werden. Soll aber doch einmal auf die XIL Ebene zugegriffen werden, so ist ähnlich wie bei einem Dateizugriff ein Handle notwendig. Aus diesem Grund wird die Variable SFG_xil_state vom Typ XilSystemState exportiert. Nähere Doku hierzu ist in den XIL Manual Pages (z.B. man xil_open) nachzulesen.

9.5 Schließen des Framegrabbers

Das Aufnehmen von Bildern wird durch den Befehl

void SFG_close();

abgeschlossen. Bei Programmende wird der Framegrabber automatisch geschlossen. Bei größeren Programmen sollte dieser Befehl aber immer explizit ausgeführt werden sobald keine Bilder mehr benötigt werden, da sonst andere Personen evtl. unnötig

lange von der Benutzung des Framegrabbers ausgeschlossen werden. Es ist hier aber auch zu bedenken, daß ein erneuter Aufruf von SFG_init() etwa 10 Sekunden dauert.

9.6 Messungen

Um die Zugriffszeiten beurteilen zu können, wurden folgende Messungen durchgeführt:

Grab-Zeiten in [ms] auf robosun3		Fast		(normal)		Save	
		fifo 1	fifo 2	fifo 1	fifo 2	fifo 1	fifo 2
Skalierung 2 384 x 288	s / w frei	116.7	110.6	116.5	221.2	160.1	161.1
	belastet	110.3	112.3	228.0	338.9	171.3	168.8
	farbe frei	139.9	93.9	140.0	181.9	200.2	200.2
	belastet	99.7	99.6	239.0	279.0	179.4	174.1
Skalierung 4 192 x 144	s / w frei	40.1	40.1	40.1	40.1	80.2	80.4
	belastet	26.9	27.1	66.4	83.1	83.7	83.9
	farbe frei	40.2	40.1 [40.1]	40.0	40.1 (40.1)	80.2	80.2 (80.1)
	belastet	23.8	23.8 (23.9)	63.6	67.6 (66.6)	80.9	80.7 (81.3)

Tabelle 9.1:

Durchschnittliche Bildrate bei unbelasteter Messung		fast		(normal)		save	
		fifo 1	fifo 2	fifo 1	fifo 2	fifo 1	fifo 2
Skalierung 2 384 x 288	s / w	2.9	2.8	2.9	5.5	3.9	4.0
	farbe	3.5	2.3	3.5	4.5	4.9	4.9
Skalierung 4 192 x 144	s / w	1	1	1	1	2	2
	farbe	1	1	1	1	2	2

Tabelle 9.2:

Tabelle 9.1 veranschaulicht die erzielbaren Grabzeiten: Die angegebenen Zeiten sind Mittelwerte, die durch jeweils 1.000 Aufrufe der Funktionen SFG_getFrame... erzielt wurden. Im unbelasteten Fall (frei) wurden die Aufrufe direkt hintereinandergesetzt, ohne zwischendurch irgendwelche Berechnungen durchzuführen. Im belasteten Fall (belastet) wurde die Maschine zwischen einzelnen Aufrufen im Mittel mit einer busy-wait-Schleife von 500 ms Dauer (zwischen 250 ms und 750 ms, gleichverteilt) belastet.

Die Werte in runden Klammern geben das Ergebnis für 20.000 Grab-Aufrufen wieder. In eckigen Klammern ist ein Meßwert für 100.000 Aufrufe angegeben.

In Tabelle 9.2 ist die durchschnittliche Differenz der erhaltenen Bildnummern von zwei aufeinanderfolgenden Aufrufen der SFG_getFrame... Funktionen, gemittelt über eine Meßreihe von jeweils 1000 Aufrufen.

9.7 Verwendung der Sourcen

Teilprojektname: `SFG (SUN FrameGrabber)`
aktuelle Version: `SFG2-1`
bei nutzenden TPR: Makefilevariable `BASESON` um die aktuelle Version erweitern

Library: `libSFG.a`
Beschreibung: Routinen zum Ansprechen des SUN Framegrabbers
Architekturen: `SUN4SOL2 SUNMP`
Includes:
```
#include <xil/xil.h>
#include "SFG.h"
```
Linkoptionen: `-lxil -lX11 -lSFG`

Programme: `SFGmain`
Beschreibung: Grabt Bilder und speichert sie als PNM Files.
Architekturen: `SUN4SOL2 SUNMP`

Die xil.h befinden sich in der Directory /opt/SUNWits/Graphics-sw/xil/include. Auf die SFG include und library Dateien wird am besten durch den an anderer Stelle beschriebenen CVS-Makefile-Mechanismus zugegriffen (vgl. Kapitel Projektverwaltung).

9.8 Fehlermeldungen

Kann der Framegrabber nicht geöffnet werden, weil er bereits von einem anderen Benutzer belegt wird erscheinen am Bildschirm etwa folgende Fehlermeldung falls die SFG Bibliothek mit gesetztem Flag -DSFG_DEBUG übersetzt wurde (standardmäßig nicht der Fall!):

```
XilDefaultErrorFunc:
   error category: System
     error string: SUNWRtvc: could not open SUNWRtvc device
         error id: SUNWrtvc-4
   primary error detected at location ZSUWDZDSFBUFZUZZQF226 in XIL
      object info: Device busy

usw....

Error: couldn't open SUNWrtvc device; device busy
```

Bei normaler Übersetzung erscheinen wesentlich kürzere Fehlermeldungen, die durch den SFG internen Error-Handler erzeugt werden und die gleiche Kerninformation enthalten. Ein eigener Error-Handler kann mit der XIL Funktion xil_install_error_handler() und der exportierten Variable SFG_xil_state (siehe Abschnitt 9.4.6) eingerichtet werden.

Kapitel 10

Benutzung der SUN Framegrabber mit HORUS

Susanne Gerl

HORUS ist ein Bildverarbeitungstool, welches an der Technischen Universität München entwickelt wurde [1], [2]. Es stellt mehr als 600 Bildverarbeitungsroutinen zur Verfügung, die in C oder C++ Programmen eingebunden werden können. Innerhalb von HORUS können die SUN-Framegrabber in den Robosuns geöffnet, Bilder eingelesen und verarbeitet werden.

Im Einzelnen stehen hierfür folgende Funktionen zur Verfügung
(In „hdialog“ unter „Bildeinzug“ zu finden):

1. `query_framegrabber`: „query_framegrabber“ gibt die Namen der unterstützten Framegrabber zurück.
2. `open_framegrabber`: „open_framegrabber“ öffnet den Framegrabber und legt seinen Betriebsmodus fest.
3. `info_framegrabber`: „info_framegrabber“ liefert die möglichen Betriebsmodi des aktuell geöffneten Framgrabbers.
4. `grap_image`: „grap_image“ liest ein Bild mit den in „open_framegrabber“ angegebenen Eigenschaften ein.
5. `close_framegrabber`: „close_framegrabber“ schließt den momentan geöffneten Framegrabber.
6. `get_framegrabber_lut` u. `set_framegrabber_lut`: „get_framegrabber_lut“ und „set_framegrabber_lut“ sind nicht implementiert und liefern MESS_OK zurück.

10.1 Initialisieren der Robosun-Framegrabber

Für die Initialisierung ist „open_framegrabber“ aufzurufen. Jede Robosun greift dabei auf ihren eigenen Framegrabber zu und initialisiert den Framegrabber für ein bestimmtes Bildformat. Das Bildformat legt u.a. fest, welcher Bildbereich des Framegrabbers mit welcher Auflösung eingelesen werden soll. Falls man unterschiedliche Bildformate einlesen will, muß man den Framegrabber mit dem ersten Format öffnen, grabben, schließen, und mit dem nächsten Format wieder öffnen usw.

Kurzfassung:

```
open_framegrabber(Name,FGWidth,FGHeight,Width,Height,StartLine,
StartCol,SquarePixels,Field,Bits,SpaceOrThresh,Gain,Generic,Dev
ice,Port)
```

Beschreibung der Parameter:

`Name` = „RoboFG“

Name des zu öffnen Framegrabbers,
Eingabe-Steuer-Parameter,
Datentypen: string / atomar,
Defaultwert: „VideoPix“,
Aktion: auf „RoboFG“ setzen (Groß/Kleinschreibung beachten).

`FGWidth` = -1, (768), 384, 256, 192, 154

Legt die gewünschte horizontale Auflösung des Framegrabbers fest,
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 768 in FG-Routine),
Aktion: Auf gewünschten Wert setzen.

`FGHeight` = -1, (576), 288, 192, 144, 115

Legt die gewünschte vertikale Auflösung des Framegrabbers fest,
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 576 in FG-Routine),
Aktion: Auf gewünschten Wert setzen.

Folgende Auflösungen (FGWidth x FGHeight) sind möglich:
768 x 576, 384 x 288, 256 x 192, 192 x 144, 154 x 115.

`Width` = -1, (768), 0 <= Width+StartCol <= FGWidth

Legt die Breite des gewünschten Bildausschnittes fest,
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,

Defaultwert: -1 (= 768 in FG-Routine),
Aktion: Auf gewünschten Wert setzen.

`Height` = -1, (576), 0 <= Height+StartLine <= FGHeight

Legt die Hohe des gewünschten Bildausschnittes fest,
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 576 in FG-Routine) ,
Aktion: Auf gewünschten Wert setzen.

`StartLine` = -1, (0), 0 <= StartLine <= FGWidth

Legt die Zeilennummer der oberen linken Ecke des gewünschten Bildausschnittes fest (Y-Offset),
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 0 in FG-Routine) ,
Aktion: Auf gewünschten Wert setzen.

`StartCol` = -1, (0), 0 <= StartCol <= FGHeight

Legt die Spaltennummer der oberen linken Ecke des gewünschten Bildausschnittes fest (X-Offset),
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 0 in FG-Routine) ,
Aktion: Auf gewünschten Wert setzen.

`SquarePixels` = „unchanged“, („no"),

Gibt an ob die Bildpunkte quadratisch sind oder nicht. Bisher werden nur quadratische Bildpunkte unterstützt.
Eingabe-Steuer-Parameter,
Datentypen: string / atomar,
Defaultwert: unchanged (= „no“ in FG-Routine) ,
Aktion: KEINE. Nur Bilder ohne SquarePixel möglich.

`Field` = -1, (2)

Legt fest, ob ein Halbbild (0/1) oder Vollbild (2) gegrabbt werden soll. Bisher wird nur das Grabben eines Vollbildes unterstützt.
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 2 in FG-Routine),
Aktion: KEINE. Nur Vollbilder möglich.

`Bits` = -1, (8), 24

Legt die Zahl der Bits pro Pixel fest. 8 = Grauwert oder 24 = Farbe. Binäre Bilder werden nicht unterstützt.
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 8 in FG-Routine),
Aktion: 8 für Grauwertbild, 24 für Farbbild eingeben.

`SpaceOrThresh` = „unchanged“, („rgb“), 128

Legt bei Farbbildern den gewünschten Farbraum und bei Binärbildern die Schwelle für die Binärisierung fest. Ist NICHT implementiert.
Eingabe-Steuer-Parameter,
Datentypen: string / atomar,
Defaultwert: unchanged (= „rgb“ bzw. 128 in FG-Routine)
Aktion: KEINE. Wert wird nicht verwendet.

`Gain` = -1.0, (1.0)

Verstärkungsfaktor für Video-Verstärker. Ist NICHT implementiert.
Eingabe-Steuer-Parameter,
Datentypen: real / atomar,
Defaultwert: 1.000000 (= 1.0 in FG-Routine),
Aktion: KEINE. Wert wird nicht verwendet.

`Generic` = „unchanged“, („roboframe“)

Framegrabber-spezifischer generischer Parameter. Wird für die Robosun-Framegrabber nicht verwendet.
Eingabe-Steuer-Parameter,
Datentypen: string / atomar,
Defaultwert: unchanged (= „roboframe“ in FG-Routine),
Aktion: KEINE. Wert wird nicht verwendet.

`Device` = „unchanged“, („/dev/rtvc0“)

Gibt das Device, an das der Framegrabber angeschlossen ist, an.
Eingabe-Steuer-Parameter,
Datentypen: string / atomar,
Defaultwert: unchanged (= „/dev/rtvc0“ in FG-Routine),
Aktion: KEINE. Neue Device nur eingeben, falls die default-device nicht mehr aktuell ist.

`Port` = -1, (1)

Gibt das Port des Devices, an das der Framegrabber angeschlossen ist, an.
Eingabe-Steuer-Parameter,
Datentypen: integer / atomar,
Defaultwert: -1 (= 1 in FG-Routine),
Aktion: KEINE. Neuen Port nur eingeben, falls default-port nicht mehr aktuell.

Anmerkungen:

a. Der Klammerinhalt gibt den Funktionswert an, dem der Default-Wert -1 bzw. „unchanged“ entspricht

b. Neu eingegebene Funktionswerte werden beim nächsten Aufruf dem Default-Wert zugeordnet.

c. Für die Konsistenz der Parameterwerte ist der Aufrufer verantwortlich (Die Funktionswerte, die dem Default-Wert zugeordneten sind, beachten!).

d. „hdialog“ öffnet defaultmässig ein 512x521 Bildfenster. Wenn man Width=768 und Height=576 setzt, erhält man einen Fehler, da diese Bildgröße die default-Bildgröße in „hdialog“ überschreitet. Auch das Öffnen eines größeren Fensters ändert daran nichts, da HORUS die Bildgrößen intern umrechnet.

e. Farbbilder werden von „hdialog“ nicht korrekt angezeigt.

f. Neu eingestellte Werte werden zu Default-Werten des Framegrabbers. Wird z.B. ein Farbbild eingelesen, der FG geschlossen und wieder neu geöffnet, so verbirgt sich unter dem Default-Wert -1 bei Bits nun 24 und nicht mehr 8 wie bei HORUS-Start. Dies kann zu Fehlermeldungen führen, wenn man z.B. den FG mit einer geringeren Auflösung neu öffnet und den Bildausschnitt und den Offset auf den Default-Werten (-1) läßt. Ist dieser Bildausschnitt plus Offset zu groß für die neue Auflösung, so wird der FG nicht geöffnet.

g. Width und Height geben die Größe des Bildausschnittes an, der von „grab_image“ geliefert werden soll. Der Bildausschnitt plus Offset (StartLine, StartCol) muß innerhalb des durch FGWidth und FGHeight bestimmten FG-Bildes liegen. Es gilt also:
0 <= Width + StartCol <= FGWidth und 0 <= Height + StartLine <= FGHeight

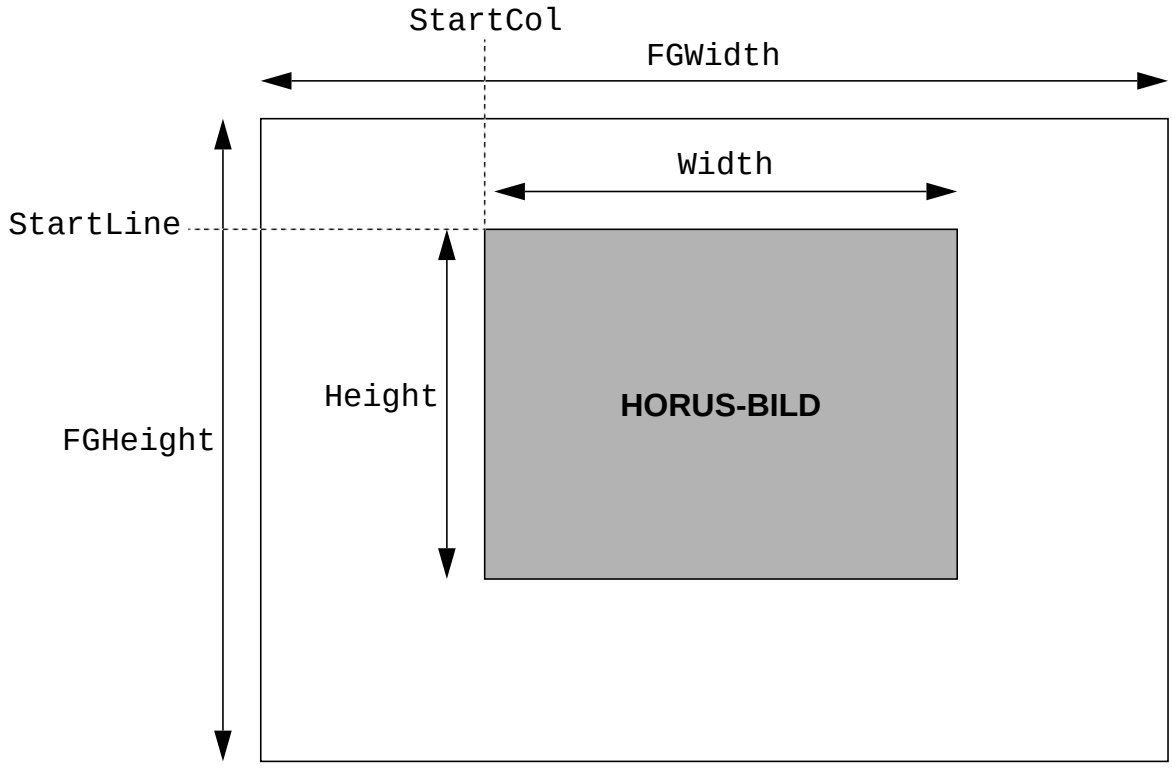


Abbildung 10.1: Wichtige Parameter bei der Initialisierung des Framegrabbers

10.1.1Verhalten

Sind die Parameterwerte korrekt und ist der gewünschte Framegrabber zum Aufrufzeitpunkt verfügbar, liefert open_framegrabber den Wert TRUE. Ansonsten wird eine Exception-Behandlung durchgeführt (siehe Exception Behandlung durch „error_text“).

Hier die C-Syntax:

```
ERR_TYPE  open_framegrabber( Name, FGWidth, FGHeight, Width,
          Height, StartLine, StartCol, SquarePixels, Field,
          Bits, SpaceOrThresh, Gain, Generic, Device, Port)
```

10.1.2Beispiele

a. Es soll ein FG-Grauwertbild 768x576 geöffnet werden und daraus ein Bild 768x576 ab Position (0,0) (Ursprung links/oben) angewählt werden:

```
open_framegrabber(„RoboFG“,-1,-1,-1,-1,-1,-1,
“unchanged“,-1,-1,“unchanged“,-1.0,“unchanged“,
“unchanged“,-1)
```

So ein Zufall, es waren gerade die Default-Werte!

b. Ein Farbbild 768x576 soll geöffnet werden und daraus ein Ausschnitt 200x100 ab Position (40,300) angewählt werden:

```
open_framegrabber(„RoboFG“,-1,-1,200,100,300,40,
“unchanged“,-1,24, „unchanged“,-1.0,“unchanged“,
“unchanged“,-1)
```

c. Ein Farbbild 154x115 soll geöffnet werden und daraus ein Ausschnitt 50x70 ab Position (10,20) angewählt werden:

```
open_framegrabber(„RoboFG“,154,115,50,70,20,10,
“unchanged“,-1,24,„unchanged“,-1.0,“unchanged“,
“unchanged“,-1)
```

10.2 Literatur

[1] Wolfgang Eckstein: *Horus-Referenzmanual*, Technische Universität München, Institut für Informatik, 1995.

[2] Wolfgang Eckstein: HORUS/C* Benutzerhandbuch, Technische Universität München, Institut für Informatik, 1994.