Institut für Prozeßrechentechnik, Automation
und Robotik
Prof. Dr.-Ing. H. Wörn
Prof. Dr.-Ing. R. Dillmann
Universität Karlsruhe
Fakultät für Informatik

Entwurf und Implementierung eines farbbildbasierten Tracking-Systems zur Führung eines mobilen Roboters

Studienarbeit
von

cand.inform. Michael Aschke

01. Januar 2000 – 31. März 2000

Referent : Prof. Dr.-Ing. R. Dillmann
Betreuer : Dipl.-Inform. P. Steinhaus

Ich versichere hiermit, die vorliegende Studienarbeit selbständig und ohne fremde Hilfe angefertigt zu haben.

Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 30. März 2000

Michael Aschke

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung und Aufgabenstellung

1.1 Einführung

Seitdem der Mensch Maschinen baut und benutzt, war der Benutzer stets derjenige, der sich an die Bedienkonzepte der Maschinen anpassen musste. Zwar haben sich die Bedienkonzepte in Richtung einer menschenfreundlichen Bedienung entwickelt, nutzen aber immer noch nicht annähernd die menschenmöglichen Ein- und Ausgabemodi wie Gestik, Haptik, Mimik und Sprache. So ist das Bedienen der Maschinen meist noch auf das Drücken von Knöpfen oder vergleichbare Eingabemöglichkeiten reduziert, während der Mensch in vielen Fällen intuitiv andere Bedienungsformen wie Spracheingabe oder deiktische[1] Gesten nutzen würde. Im Bereich der Robotik gibt es unterschiedliche Bedienmodelle für verschiedene Roboterklassen, doch können diese größtenteils nicht als menschenfreundlich bezeichnet werden.

Dies kann bei Industrierobotern[2] durch verschiedene Arten der Programmierung geschehen. Zum einen gibt es die explizite Programmierung durch Spezialisten, in der jedem Gelenk des Roboters mitgeteilt wird, um wieviel Grad es sich drehen soll oder um wieviel ein Werkzeug bewegt werden soll. Eine weitere Möglichkeit ist das Teach-In oder die Fernsteuerung, bei dem der Roboter auf bestimmte Art und Weise in verschiedene Positionen gefahren wird, welche er sich "merkt" und dann später auf gleiche Art und Weise automatisch ansteuern soll, um dort ebenfalls durch Fernbedienung eingegebene Manipulationen durchzuführen. Ein ähnlicher Ansatz ist das Master-Slave Verfahren, bei dem ein kleines Modell eines Roboters durch den Menschen dazu gebracht wird, eine Aufgabe virtuell zu lösen und der eigentliche Roboter synchron zu dem Modell verfährt und somit die Aufgabe löst.

Aber auch bei mobilen oder autonomen Robotern erfolgt die Eingabe neuer Lösungen noch meist über den Leitstand oder einen am Roboter angeschlossenen Joystick. Dies alles zeugt nicht gerade von einem menschenfreundlichen Mensch-

[1] demonstrative, hinweisende

[2] Oder allgemeiner: bei Manipulatoren

Maschine-Dialog. Gerade das ist das angestrebte Ziel der Wissenschaft: Die menschenfreundliche Bedienung von Maschinen. Dazu gehört auch, die multimodalen Fähigkeiten des Menschen besser auszunutzen. Warum soll ein Roboter nicht wirklich verstehen können, was man ihm sagt? Wieso soll ein Roboter nicht sehen können, was er zu tun hat? Gerade die Möglichkeit einer multimodalen Steuerung eines Roboters machen den Mensch-Maschine-Dialog einfacher, da sich die Maschine auf die Bedienmöglichkeiten des Menschen konzentriert und somit neue Aufgaben auch effizienter und kosteneffektiver in Angriff genommen werden können. Jetzt soll sich die Technik an den Menschen anpassen.

1.2 Aufgabenstellung

Ziel dieser Aufgabe war es, ein Tracking-System als Baustein eines Software-Baukastensystems für Service-Roboter zu entwerfen. Es soll beispielhaft auf einem mobilen Roboter implementiert werden und eine einfache, aber sichere Führung des Roboters durch den Menschen ermöglichen. Damit es auf vielen Systemen eingesetzt werden kann, darf es nur geringe Hardware-Ressourcen in Anspruch nehmen, d.h. es sollte mit einer einzigen Farbkamera auskommen.

Die Aufgabe gliederte sich in folgende Teilaufgaben:

- Aufbau der Bildverarbeitung
- Auswahl des zu trackenden Objektes
- Entwurf und Implementierung der Bildverarbeitungsmethoden zum Tracken
- Entwurf des Reglers zur Robotersteuerung
- Berücksichtigung der Benutzersicherheit

Der konzipierte Lösungsweg ist in Abbildung 1.1 zu sehen. Zuerst wird ein Bild aus einem Ringbuffer (siehe Kapitel 4.1.1) eingelesen. Auf diese Daten wird ein Hautsegmentierungsalgorithmus angewendet. Nach der Hautsegmentierung wird versucht, das segmentierte Objekt in der Bildmitte zu halten. Hierzu soll ein Schwerpunkt berechnet und dessen physikalische Attribute erfasst werden. Der Schwerpunkt wird genutzt, um die Regelung des Roboters durchzuführen.

1.3 Gliederung

Die Arbeit gliedert sich in folgende Kapitel:

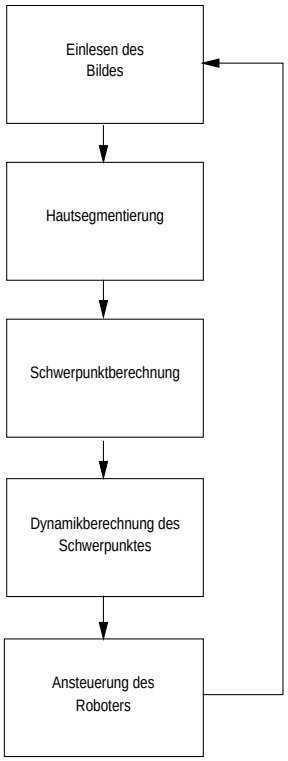

Abbildung 1.1: Überblick der Teilschritte zur Realisierung einer Folgefahrt

Kapitel 1: Einführung und Aufgabenstellung – Hier wird eine kleine Einführung in die Thematik dieser Studienarbeit gegeben und eine Motivation aufgeführt, warum es sich lohnt diese Aufgabe zu lösen. Desweiteren wird auf die Aufgabenstellung eingegangen und die Gliederung der Arbeit aufgezeigt.

Kapitel 2: Bestehendes System – Dieses Kapitel beschäftigt sich mit dem schon bestehenden System. Es werden Komponenten beschrieben, welche in den Lösungsweg mit einbezogen wurden. Auch wird der mobile Roboter MORTIMER (siehe Abbildung 1.2) und seine Komponenten beschrieben, auf welchem beispielhaft die Tracking-Software implementiert wurde.

Kapitel 3: Architektur des Trackings – Um dem Leser einen Eindruck zu geben, wie die entwickelte Software funktioniert und auf was geachtet werden muss, wird in diesem Kapitel ein Überblick über die Architektur des Lösungsweges gegeben, bei dem bewusst auf Details verzichtet wird. Damit soll erreicht werden, dass man sich schnell über das Wesentliche informieren kann, ohne von störenden Details belästigt zu werden. Außerdem werden zu einigen Bereichen

Abbildung 1.2: Der mobile Roboter MORTIMER

wichtige Grundlagen erläutert.

Kapitel 4: Realisierung der Teilkomponenten – Dieses Kapitel bildet den inhaltlichen Schwerpunkt der Studienarbeit. In ihm werden nähere Details zur Realisierung des konzipierten Lösungsweges behandelt.

Kapitel 5: Experimente – Die Experimente dienen dazu, die Qualität der entwickelten Software zu testen und eventuelle Fehlerquellen einzugrenzen und zu analysieren. Dazu wird ein Experiment konzipiert, nach welchem das erwartete Verhalten des Roboters mit dem wirklichen Verhalten verglichen wird.

Kapitel 6: Zusammenfassung und Ausblick – Schließlich wird das bisher Erarbeitete zusammengefasst und kritisch betrachtet. Am Ende wird ein Ausblick auf potentielle Entwicklungen, welche auf der entwickelten Software basieren können, gegeben.

Kapitel 2

Bestehendes System

In diesem Kapitel wird kurz auf die für diese Arbeit relevanten Hard- und Softwaremodule des Roboter-Baukastensystems eingegangen.

2.1 Systemarchitektur - Client/Server – Konzept

Die Client/Server-Architektur für die Roboterbaukastensysteme ist in Abbildung 2.1 zu sehen. Dabei läuft auf jedem Roboter eine Server-Software, mit welcher der Leitstand kommunizieren kann. Dabei kann man die ganze Architektur in drei Ebenen aufteilen: die Kontrollebene, auf der sich die Leitstände befinden; die Kommunikationsebene, welche hier nicht explizit dargestellt ist und die Ausführungsebene, auf welcher sich die Roboterbaukastensysteme befinden. Hierbei stehen auf der Kontrollebene mehrere Leitstände zur Verfügung. Hier seien die Leitstände MARS und MobVis erwähnt, welche in Kapitel 2.3.1 bzw. 2.3.2 näher erläutert werden. Hauptunterschied der beiden Clients ist, dass MARS gleichzeitig Verbindungen zu mehreren Robotern unterstützt, während MobVis nur die Möglichkeit einer zeitgleichen Verbindung bietet.

2.2 Roboter/Server – Architektur

Bei der Betrachtung der Roboter/Server-Architektur wird zwischen der Hardware-Architektur und der Software-Architektur differenziert. Der Hardware-Architektur liegt das Baukastenprinzip zu Grunde, um durch die so entstehende Funktionsmodularisierung eine möglichst hohe Flexibilität zu erreichen.

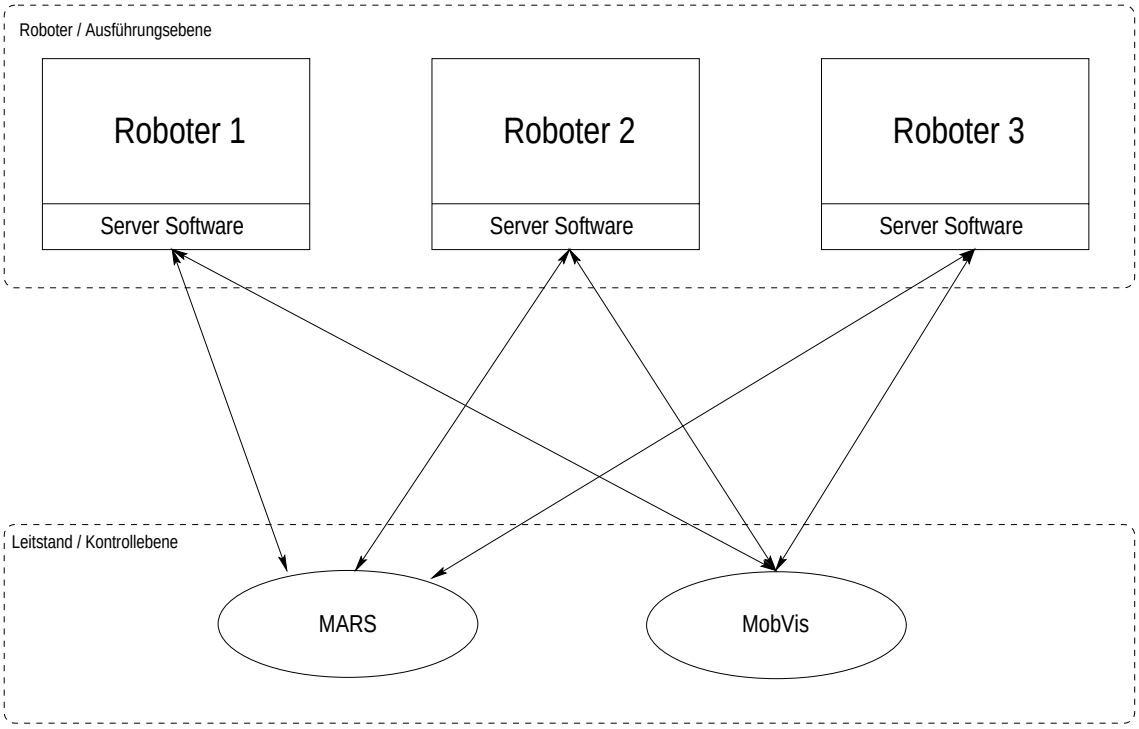

Abbildung 2.1: Das Client/Server-Konzept, auf dem diese Arbeit basiert.

2.2.1 Hardware-Architektur

Das Roboterbaukastensystem am Beispiel von MORTIMER

Das Systemkonzept von MORTIMER kann in drei Ebenen unterteilt werden. Auf Ebene 1 befindet sich die mobile Plattform, welche sich aus der Mechanik, dem Antriebskonzept und der Motorregelung zusammensetzt. Die Ebene 2 stellt die sensorbasierte Navigation dar, in welcher die Bewegungssteuerung und die Positionsbestimmung realisiert sind. Schließlich besteht die Ebene 3 aus der Planung & Bedienschnittstelle. Im Begriff der mobilen Plattform soll auch die Interpolation elementarer Bewegungen enthalten sein.

Der Antrieb – Mobiler Untersatz und Motorsteuerung

Als Antriebskonzept wurde eine Differentiallenkung realisiert. Zwei von einander unabhängige angetriebene Räder benötigen auf Grund der Beschränkung auf zwei Antriebsräder einen geringen mechanischen und steuerungstechnischen Aufwand bei gleichzeitig hoher Beweglichkeit. Kurvenbewegungen des Roboters werden durch unterschiedliche, eventuell auch gegenläufige ("Drehen auf der Stelle") Drehzahlen der

Antriebsräder ermöglicht.

Die Motorsteuerung wurde zusammen mit der amerikanischen Firma TRC entwickelt. Bei dieser Entwicklung konnten sicherheitsrelevante Komponenten eingebaut werden. Die Motorgeschwindigkeit wird von einem PC-Board durch eine Kaskadenregelung geregelt.

Das Rechnersystem

Aus Preisgründen wurden PC-Komponenten einer VMEbus-Lösung vorgezogen. Als Betriebssystem wurde VxWorks verwendet, um Echtzeitanforderungen zu genügen. Die Vernetzung der PC-Komponenten ist heterogen. Neben seriellen und parallelen Punkt-zu-Punkt–Verbindungen sind ein Statusbus, welcher den Zugriff jeder Komponente auf wichtige Statusinformationen in Echtzeit ermöglicht, als auch eine Ethernetverbindung, über welche Navigations- und Sensorrechner kommunizieren, realisiert.

Als Sensorrechner befindet sich auf MORTIMER ein 200MHz-Pentium, welcher unter dem Echtzeitbetriebssystem VxWorks betrieben wird.

Der Laserscanner

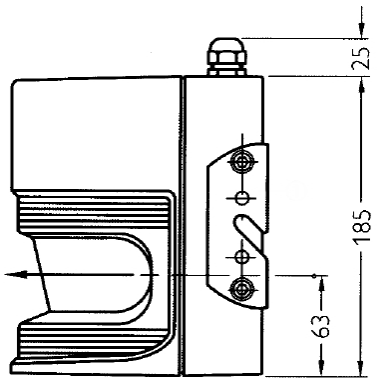

Abbildung 2.2: Der Laserscanner PLS 100 (*Seitenansicht*)

Bei dem Laserscanner handelt es sich um einen PLS 100 der Firma SICK (siehe Abbildung 2.2). Er verfügt über eine Reichweite von bis zu 50 Metern, wobei der zuverlässige Messbereich ca. 4m beträgt. Sein "Sichtfeld" d.h. der maximale Messbereich beträgt 180°, welchen er mit einem Winkelabstand von 0.5° abtastet, so dass 360 Werte abrufbar sind. Die Ansprechzeit für den PLS liegt bei über 80ms. Bei einer Entfernung von 4m verfügt der Laserscanner über eine Auflösung von 70mm.

Bildverarbeitung

Bei der hier verwendeten Kamera handelt es sich um eine CCD-Farbkamera, welche RGB-Daten liefert. Als Framegrabber wird ein Matrox Meteor eingesetzt. Die Bilder werden mit 24bit Farbtiefe und 320x240 Pixel digitalisiert.

Für eine detailliertere Beschreibung des mobilen Roboters MORTIMER wird auf den Projektbericht [Gra96] verwiesen, welcher als Grundlage dieser Beschreibung diente.

2.2.2 Software-Architektur

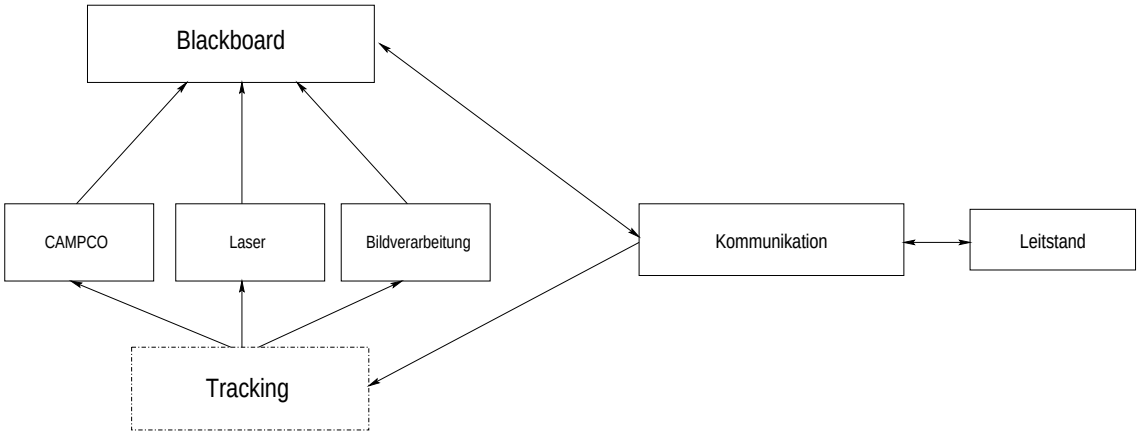

Abbildung 2.3: Überblick über die existierende (und geplante) Software-Architektur

Bei der in Abbildung 2.3 dargestellten Software-Architektur sind die schon existierenden Komponenten und deren Zusammenspiel zu erkennen. Der CAMPCO (siehe Kapitel 2.2.2), sowie der Laserscanner und ein Kamerasystem mit Framegrabber für die Bildbearbeitung existierten bereits. Sie alle stehen in Verbindung mit dem "Blackboard". Das Blackboard setzt sich aus allen globalen Variablen der einzelnen Komponenten zusammen und stellt den Hauptbezugspunkt, auf dem Variablen ausgelesen und gesetzt werden können, der Kommunikation zum Leitstand dar. Diese vorhandene Struktur soll auch das Tracking nutzen. Es kann durch einen Schalter eingeschaltet werden, nutzt den Laser, um Kollisionen zu vermeiden, den CAMPCO, um der Roboterplattform Fahrbefehle mitzuteilen und schließlich auch eine um eigene Komponenten erweiterte Bildbearbeitung, um zu wissen, ob sich ein zu trackendes Objekt im Bildbereich befindet. Eine nähere Beschreibung zum Trackingvorgang ist in Kapitel 3 und 4 zu finden.

Der CAMPCO

Der CAMPCO[1] ist ein Modul, welches lokal auf dem Roboter läuft. Es regelt die Position und Orientierung der Roboterbasis des MORTIMER. Wie in [Hub98] genauer beschrieben, sieht die CAMPCO-Struktur wie folgt aus:

1. Kommunikation: Steuerung des CAMPCO über die TCP/IP-Socketverbindung sowie Datentransfer zum und vom Motorregler.
2. Globale Variablen: Speicherung der aktuellen und geplanten Daten von Position, Ausrichtung, Geschwindigkeiten und Beschleunigungen.
3. Positionsregler: Er gibt Sollgeschwindigkeiten und -Beschleunigungen vor, um eine bestimmte Position bzw. Zielausrichtung zu erreichen (Statemachine).
4. Gelenkregler: Er regelt die Geschwindigkeiten und Beschleunigungen auf ihren Soll-Wert (PI-Regler).

Die Struktur des CAMPCO wird in Abbildung 2.4 verdeutlicht.

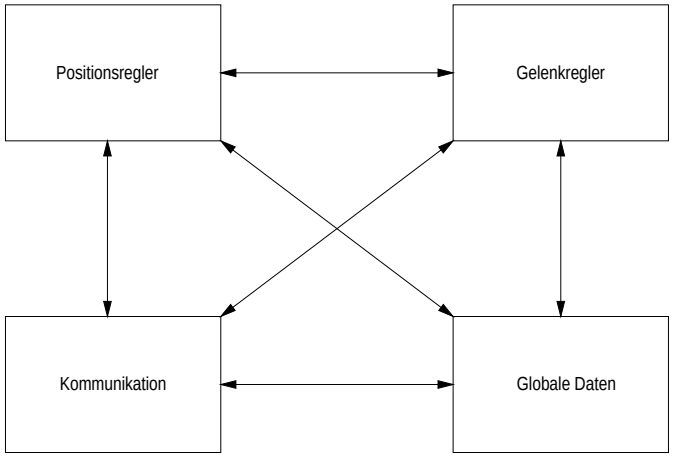

Abbildung 2.4: Die Komponenten des CAMPCO

2.3 Clients/Leitstände

2.3.1 MARS

Das Navigationssystem MARS[2] ist am Institut für Prozessrechentechnik und Robotik (IPR) an der Universität Karlsruhe entwickelt worden. Es besteht aus einer

[1] **CA**scaded **M**ulti **P**urpose **CO**ntroler
[2] **M**ulti **A**gent **R**obot **S**ystem

grafischen Benutzerschnittstelle (GUI), an welcher man sich per Menü einen der verfügbaren Roboter auswählen kann. Auf der grafischen Oberfläche von MARS kann zudem die Umgebung des Roboters wahlweise in 2D oder 3D dargestellt werden und auf Wunsch noch der aktuelle Laserscan (in 2D) hinein geblendet werden. Außerdem kann man Befehle an den Roboter über das Navigationssystem MARS schicken, welcher der Roboter ausführen soll. Dazu kommuniziert MARS über eine Funkverbindung mit dem CAMPCO-Modul des Zielroboters.

2.3.2 MobVis

Das Projekt MobVis (*Mobile Vision*) hatte zum Ziel, im Rahmen des KORINNA-Projektes den Einsatz von optischen Kameras auf Service-Robotern möglich zu machen. Die durch die Kameras gewonnen Umweltinformationen sollen zur Kollisionsvermeidung und Wegplanung genutzt werden. Dabei sollen die Bildverarbeitungseinheiten als unabhängige Module ansprechbar sein, um diese später auf verschiedenen Roboter-Plattformen (MORTIMER, VIPER, ...) einsetzen zu können. Die Software für MobVis wurde auf MORTIMER getestet und auch für MORTIMER entwickelt. Aufgrund der Spezifikation des Sensorrechners musste ein Linux-Treiber für den Meteor Framegrabber auf VxWorks portiert werden. Es existiert eine Software-Umgebung, mit der man die Bildverarbeitungssysteme auf mobilen Robotern steuern kann. Sie besteht im wesentlichen aus einem Client-/Server-System, wobei sich der Server auf der jeweiligen mobilen Plattform befindet.
Nähere Informationen kann man unter [Ste98] finden.

2.3.3 Entwicklungsumgebung – Der Tornado-Launcher

Der Launcher (siehe Abbildung 2.5) ist eine grafische Benutzerschnittstelle, mit der man auf dem Roboter einzelne Programme oder sogar nur einzelne Funktionen eines Programms, welches manuell oder automatisch geladen wurde, ausführen kann. Zu Testzwecken kann man sich auch auf einer virtuellen Konsole (siehe Abbildung 2.6) Status- oder Debuginformationen ausgeben lassen, sofern das Programm oder die Funktion solche Informationen ausgibt. Um Module auf dem Roboter zu starten wird eine WindShell (siehe Abbildung 2.7) geöffnet, in welcher man dann die Anweisungen zum Laden der gewünschten Module eingibt. Schließlich kann manuell die gewünschte Funktion gestartet werden. Der Launcher war grundlegend wichtig für die Evaluierung der Software. Bei Abschluss dieser Arbeit muss er aber nicht mehr gestartet werden, da die Tracking-Software im Kernel eincompiliert sein wird.

Abbildung 2.5: Der Tornado-Launcher

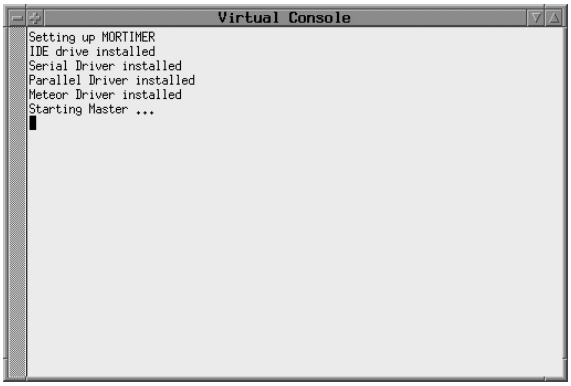

Abbildung 2.6: Die Virtual Console des Tornado-Launchers

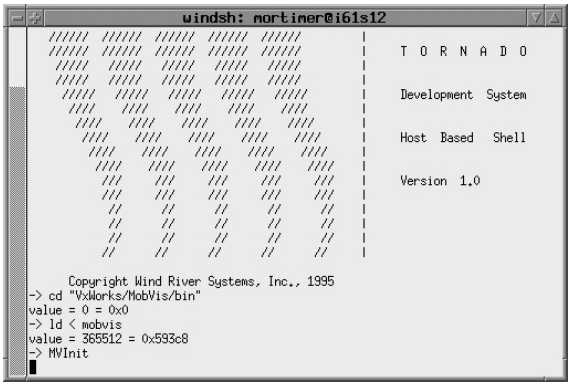

Abbildung 2.7: Die WindShell des Tornado-Launchers

Kapitel 3

Architektur des Trackings

In diesem Kapitel werden die Teilschritte des Trackings aufgezählt und grob erklärt. Es soll dazu dienen, einen Überblick über die Architektur des Trackings zu gewinnen. Eine detaillierte Beschreibung ist im nächsten Kapitel zu finden.

3.1 Initialisierung

Bei der Initialisierung des Trackingmoduls wird ein Prozess gestartet, welcher fortwährend die Kamera ausliest und der Trackingsoftware ein aktuelles Bild zur Verfügung stellt. Außerdem wird der Kippschalter, welcher sich an der Front der Roboterplattform befindet abgefragt und bei entsprechender Stellung der Trackingprozess gestartet.

3.2 Hautsegmentierung

Um eine Folgefahrt überhaupt erst zu ermöglichen, ist es notwendig, sich auf bestimmte Bildmerkmale zu konzentrieren. Hierbei war zuerst die Frage nach leichter und benutzerfreundlicher Bedienung wichtig. Neben anfänglichen Ideen wie z.B. Kugeln bestimmter Farbe oder Handschuhe bestimmter Farbe hat sich die Führung des Roboters basierend auf einer Hautsegmentierung als im Vergleich zu den anderen Ansätzen menschenfreundlichste Führungsform durchgesetzt, weil sie zudem noch relativ lichtunabhängig ist. Bei der hier implementierten Lösung werden also Bereiche mit Hautfarbe als die zu segmentierenden Bildmerkmale betrachtet.
Die für die in dieser Studienarbeit verwendete Hautsegmentierung grundlegende Veröffentlichung [YLW98] basiert auf einem stochastischen Hautfarben-Modell und deren Adaptierung. Auf diese Publikation wird in Kapitel 4.1.1 näher eingegangen.
In Abbildung 3.1 sind grob die Hauptschritte zur Hautsegmentierung aufgezeigt. Zuerst wird das aktuelle Bild eingelesen

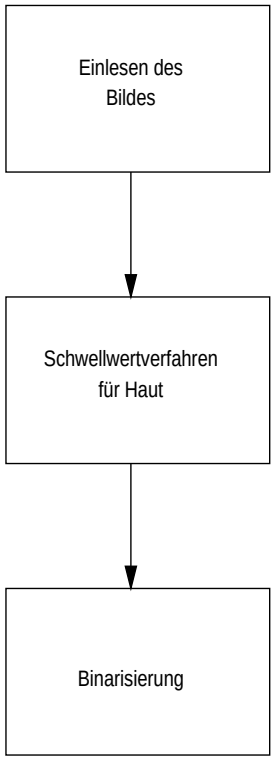

Abbildung 3.1: Verfahren der Hautsegmentierung im Überblick

3.3 Bildnachbearbeitung

Da die Ergebnisse der Hautsegmentierung leider noch nicht befriedigend genug waren, ist eine Bildnachbearbeitung notwendig. Um das Ergebnis noch weiter zu optimieren wurden weitere Mechanismen implementiert, welche fälschlicherweise segmentierte Bereiche eliminieren. Außerdem wurde das Ergebnis auf das Wesentliche reduziert, um Fehlschätzungen und daraus resultierende Fehlinformationen für den Roboter zu vermeiden. Wie in Abbildung 3.2 zu sehen ist, besteht die Bildnachbearbeitung grob gesehen aus einer Aufbereitung der Bilddaten für die Schwerpunktsberechnung. Dabei bedeutet der Opening-Operator, dass eine Maske über das Bild fährt, welche ein "Pixelrauschen" (z.B. Einzelpixel in der Nähe einer größeren Gruppe von Pixeln) um größere Bereiche eliminiert und auch schmale Randbereiche der größeren Bereiche abschneidet. Der Closing-Operator hat den Effekt, dass eng bei einander liegende Bereiche verschmelzen. Eine genauere Beschreibung befindet sich in [Jäh97], oder im nächsten Kapitel.

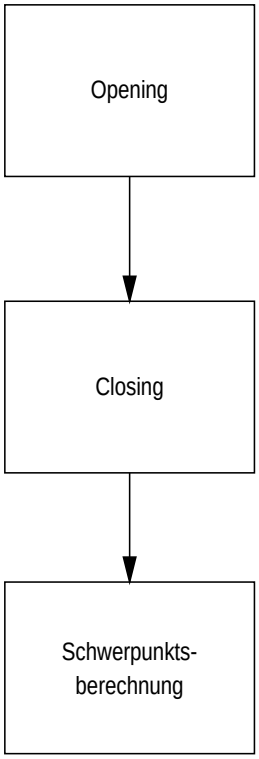

Abbildung 3.2: Struktur der Bildnachbearbeitung

3.4 Regler

Abbildung 3.3 zeigt den Wirkungsplan eines PID-Reglers. Dabei bedeutet "PID", dass das Ausgangssignal aus drei Anteilen besteht, welche das Eingangssignal jeweils anders verarbeiten: Dem Proportionalitätsanteil (P), dem Integrationsanteil (I) und dem Differentialanteil (D). Bei der Realisierung des Reglers für die Folgefahrt wurde ein PI-Regler vorgesehen.
In Abbildung 3.4 ist zu sehen, wie der Regelkreis (hier noch ohne Anbindung an die Robotersteuerung) aufgebaut ist. Nachdem das ursprüngliche Bild nachbearbeitet wurde und nun auf das Wesentliche reduziert wurde kann damit begonnen werden, den Schwerpunkt der segmentierten Bereiche zu berechnen, um den Roboter anzuweisen, diesen berechneten Schwerpunkt in der Bildmitte zu halten.

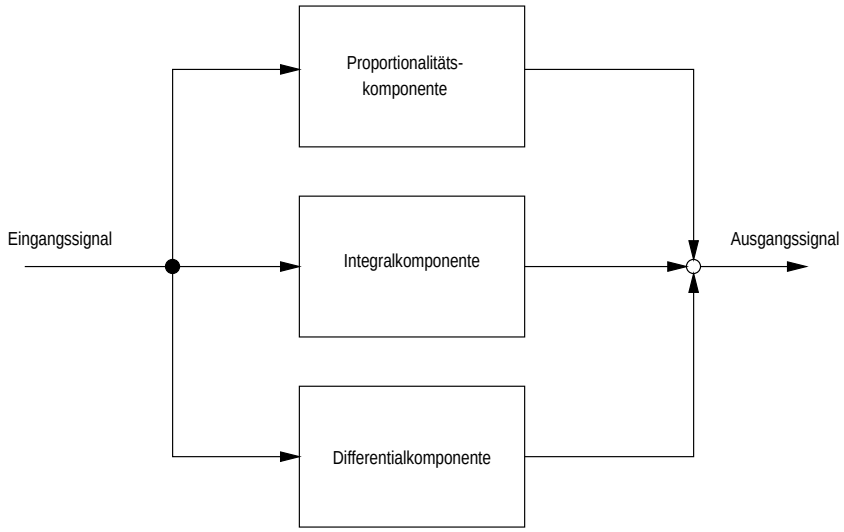

Abbildung 3.3: Wirkungsplan eines PID-Reglers

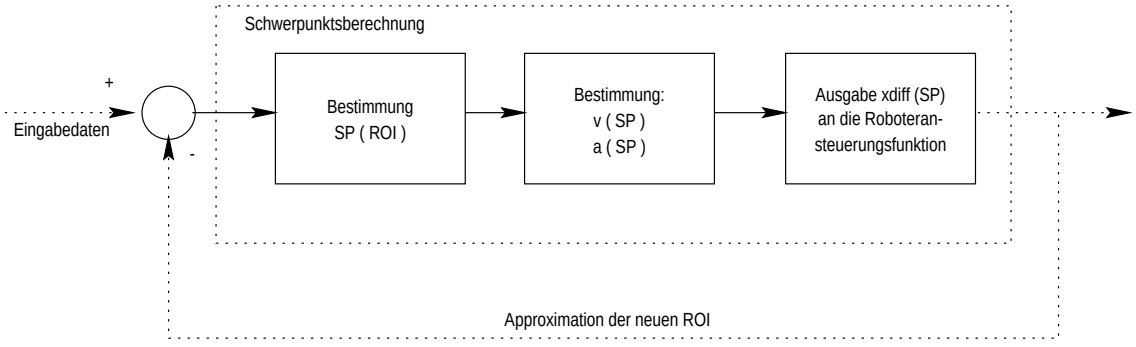

Abbildung 3.4: Innerer Regelkreis bei Übergabe des Schwerpunktes an die Roboteransteuerungsfunktion

Kapitel 4

Realisierung der Teilkomponenten

In diesem Kapitel erfolgt eine detaillierte Beschreibung der einzelnen Teilschritte und deren Verzahnung.

4.1 Aquirierung

Zuerst bestand das Problem, den Prozess des Auslesens der Kameras schlank zu gestalten, um möglichst viel Rechenleistung für das eigentliche Auswerten der Bilder übrig zu lassen. Gelöst wurde dieses Problem durch die Verwendung eines Ringpuffers, wie er im nächsten Abschnitt beschrieben wird.

4.1.1 Ringpuffer

Für das Auslesen der Kamera (Grabben) wurde ein eigener Prozess generiert, welcher in einer Endlosschleife einen Ringpuffer, bestehend aus drei Pufferspeichern, füllt. Wie man in Abbildung 4.1-a sehen kann, wird das jeweils aktuelle Bild eingelesen und reihum in einen Puffer geschrieben. Aus diesem "Topf" bedient sich die Trackingsoftware, wenn sie für eine neue Iteration ein neues Bild benötigt. In Abbildung 4.1-b wird durch einfaches Sperren des von der Software gerade ausgelesenen Puffers verhindert, dass während des Lesevorgangs durch Überschreiben durch den Grabprozess inkonsistente und damit unbrauchbare Daten entstehen. Durch die beiden getrennten Prozesse ergibt sich zudem noch eine höhere Leistungsfähigkeit, da während des Einlesens neuer Bilder der parallele Prozess stets ein Bild auswerten kann.

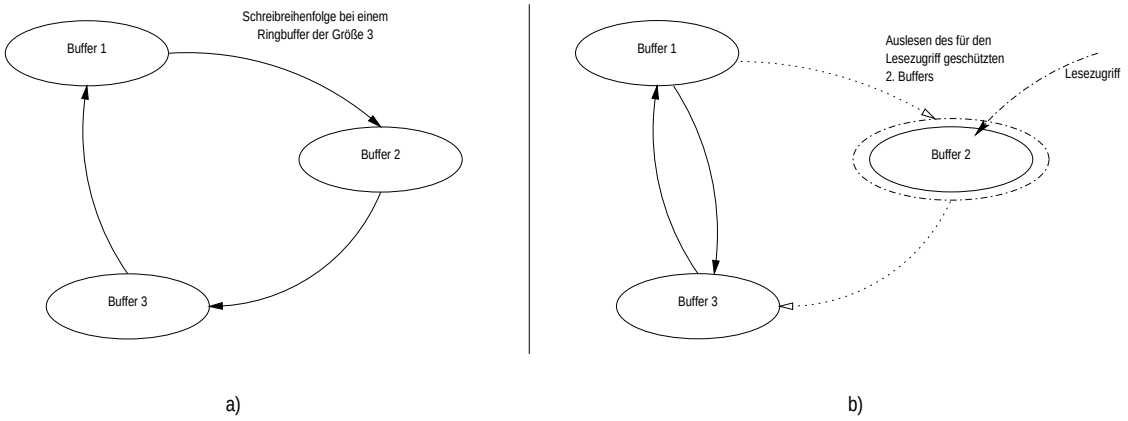

Abbildung 4.1: Verwendete Ringpufferstruktur: **(a)** während des Grabbens und **(b)** während eines Lesezugriffs

4.2 Hautsegmentierung

Das Verwenden von Farbe als Merkmal des Trackings stößt auf mehrere Probleme: Zuerst einmal hängt die Farbrepräsentation von Haut von mehreren Faktoren, wie z.B. dem Umgebungslicht, etc. ab. Hinzu kommt die Tatsache, dass verschiedene Kameras auch verschiedene Farbwerte selbst für dasselbe Objekt (z.B. eine Hand, oder ein Gesicht). Auch haben verschiedene Personen verschiedene Hautfarben. Allerdings machen sich z.B. letztere Merkmale nicht unbedingt in verschiedenen Farbwerten bemerkbar, sondern eher in verschiedenen Farbintensitäten. Das Schema der Hautsegmentierung wird in Abbildung 4.2 dargestellt, wobei hier schon zwei Schritte verschmolzen sind: Die Hautsegmentierung und die Binarisierung (siehe Kapitel 4.3).

4.2.1 Farbmittelwerte

Man hat festgestellt, dass sich die Farbwerte von Hautfarben verschiedener Menschen nur in einem relativ kleinen Bereich des Farbraums befinden. Wenn man sich, wie in [YLW98] detailliert beschrieben und wie in Abbildung 4.3-b zu sehen, den Farbraum ansieht und darin die Verteilung der Hautfarben, so stellt man fest, dass man sich "nur" auf einen relativ kleinen Bereich des Farbraums konzentrieren muss. In Abbildung 4.3-a sieht man Hautfarbenvorkommen von 48 Gesichtern, welche zufällig aus einer Datenbank von 1000 menschlichen Gesichtern ausgewählt wurden. Hierbei wurden die Mittelwerte und Varianzen festgestellt, welche in Tabelle 4.1 dargestellt sind.

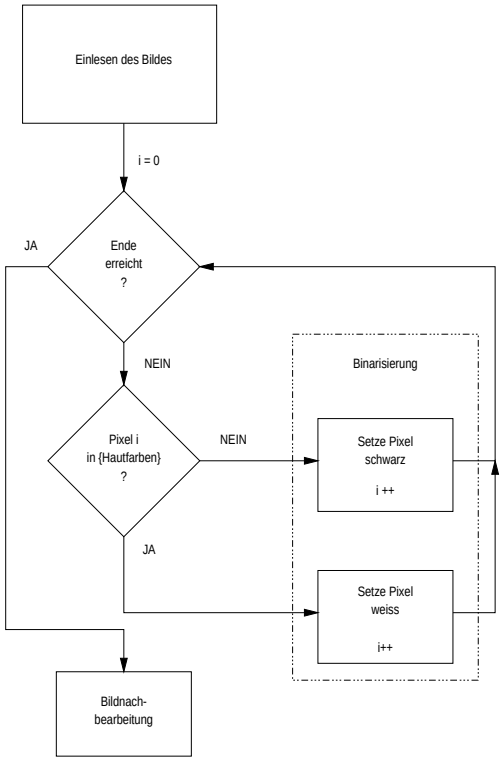

Abbildung 4.2: Schema der Hautsegmentierung mit anschließender Binarisierung

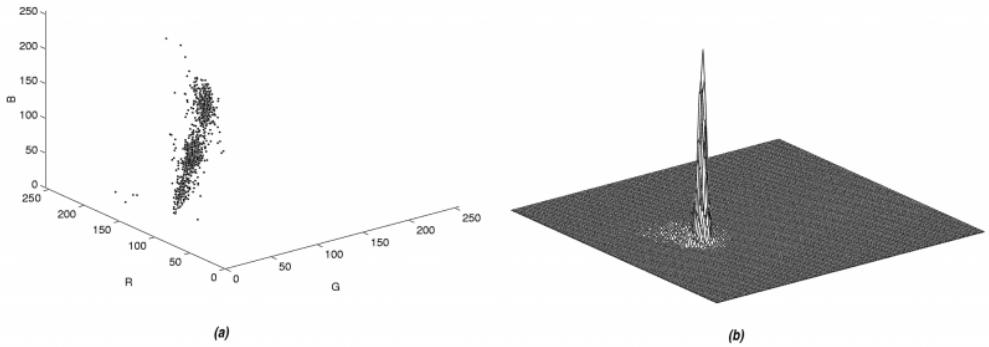

Abbildung 4.3: Hautfarbenmodellierung nach [YLW98] **(a)** Auftreten von Hautfarbereignissen im RGB-Farbraum – **(b)** Farbverteilung der Hautfarben in Farbraum

Mittelwerte	**Varianzen**
$m_R = 234.29$	$\sigma_R = 26.77$
$m_G = 185.72$	$\sigma_G = 30.41$
$m_B = 151.11$	$\sigma_B = 25.68$

Tabelle 4.1: Mittelwerte und Varianzen für die Hautsegmentierung im RGB-Raum

Mittelwerte	**Varianzen**
$m_r = 104.22$	$\sigma_r = 4.93$
$m_g = 81.59$	$\sigma_g = 3.89$

Tabelle 4.2: Mittelwerte und Varianzen für die Hautsegmentierung im normalisierten Farbraum

4.2.2 Normalisierte Farbwerte

Die Hautfarbensegmentierung im RGB-Raum hat den Nachteil, dass sie abhängig von den gegebenen Lichtverhältnissen ist. Dieses Problem kann man wesentlich verringern, indem man mit normalisierten Farbwerten arbeitet. Dabei berechnet sich ein normalisierter Farbwert wie folgt (am Beispiel von Rot):

$$r = 255 \cdot \left(\frac{R}{R + G + B} \right) \tag{4.1}$$

Der Faktor 255 aus (4.1) wurde hinzugefügt, da bei der eigentlichen Formel zur Berechnung von normalisierten Farbwerten $r \in [0..1]$ gilt. Hieraus ergeben sich dann folgende Mittelwerte und Varianzen für den normalisierten Farbraum, welche Tabelle 4.2 zu entnehmen sind. Aufgrund der Eigenschaften des normalisierten Farbraums genügen hier zwei Farbwerte (hier R und G) aus, um eine Farbe genau zu spezifizieren. In Abbildung 4.4 kann man drei Beispiele für die Hautsegmentierung in RGB-Kamerabildern sehen. Dabei werden auch schon Ergebnisse der in den nächsten beiden Teilkapiteln behandelten zusätzlichen Verfahren erkennbar.

4.2.3 Probleme

Der bisher vorgestellte Algorithmus zur Segmentierung von Hautfarben hat den Nachteil, dass er einen sehr großen Bereich segmentiert, in dem sich nicht nur Hautfarben befinden. Es hat sich herausgestellt, dass zum Beispiel auch Kartons, Objekte von einem bestimmten intensiv leuchtenden Rot, Tontöpfe und Leuchtstoffröhren ebenfalls als Haut "erkannt" werden. Dieses Problem konnte bisher noch nicht vollständig gelöst werden, da z.B. beim Herausfiltern des leuchtenden Rots auch

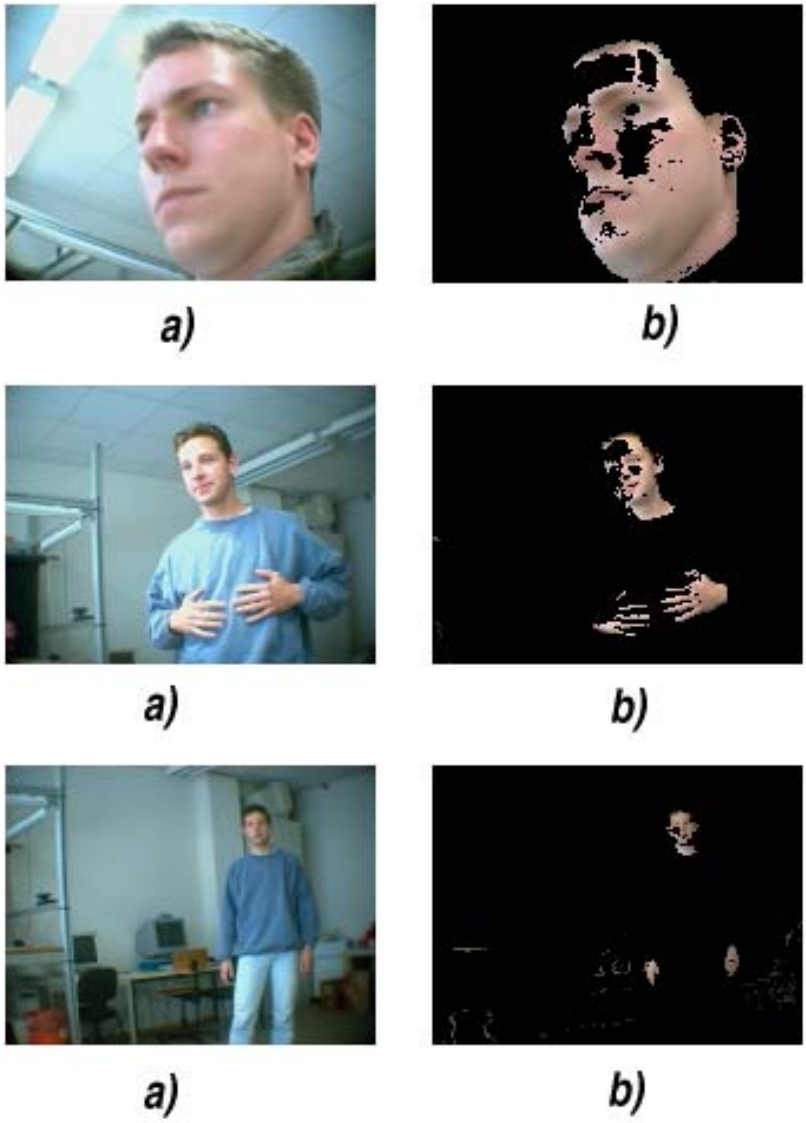

Abbildung 4.4: Beispiele für die Hautsegmentierung in RGB-Kamerabildern. Teil **(a)** repräsentiert das Original, Teil **(b)** das gleiche Bild im segmentierten Zustand.

große Bereiche von korrekt erkannter Hautfarbe verschwinden, was den Trackingprozess sehr schwierig macht.

4.2.4 Zusätzliche Schwellwerte

Um die Anzahl der Fehlsegmentierungen zuverlässig geringer zu machen wurden für Leuchtstoffröhren ein zusätzliches Schwellwertverfahren implementiert, welches alle Bereiche herausfiltert, in denen die Farbwerte von R, G und B eine bestimmte Kombination von Farbwerten überschreitet. Dies hat zur Folge, dass (wie im ersten Teil der Abbildung 4.4 zu sehen ist) eingeschaltete Leuchtstoffröhren zuverlässig herausgefiltert werden. Allerdings verschwinden ebenso zuverlässig Hautbereiche, die das Licht einer Lichtquelle reflektieren und somit ebenfalls sehr hell erscheinen. Das ist aber insofern verschmerzbar, da diese Flächen nicht so großß sind, als dass man nicht auf sie verzichten könnte. Für die anderen Fehlerkennungen sind noch keine geeigneten Kombinationen von Farbwerten gefunden worden, die einen annehmbaren Verschnitt zur Folge haben.

4.3 Binarisierung

Was mit Binarisierung gemeint ist, ist aus Abbildung 4.2 entnehmbar. Da für das Tracking ja eigentlich nur von Belang ist, wo sich ein segmentierter Bereich befindet, reicht es völlig aus, die Bildinformationen auf ein Minimum zu reduzieren. Hier wurden alle Bereiche, welche nicht der Haut zugeordnet wurden Schwarz (*nicht segmentiert*) und segmentierte Bereiche Weiß markiert. Zudem ist eine Binarisierung notwendig, um das Bild in einen für die Bildnachbearbeitung verarbeitbaren Zustand zu versetzen.

4.4 Morphologische Operatoren

Morphologische Operationen gehören zur Klasse der Nachbarschaftsoperationen, welche die Form von Objekten bearbeiten. Diese Operatoren setzen Bildpunkte in einer kleinen Nachbarschaft in Beziehung und können bei Binärbildern nur 1 (weiß) oder 0 (schwarz) als Ergebnis liefern. Hierbei werden Bildpunkte einem Objekt hinzugefügt oder entfernt. Der grobe Ablauf der Bildnachbearbeitung durch morpohologische Operatoren ist in Abbildung 4.5 zu sehen. Dabei wird das Bild binär mit einer Maske gefaltet wird. Die folgenden Erläuterungen sind an [Jäh97] angelehnt. Für alle folgenden Abschnitte und Beispiele in diesem Teilkapitel wird M als eine

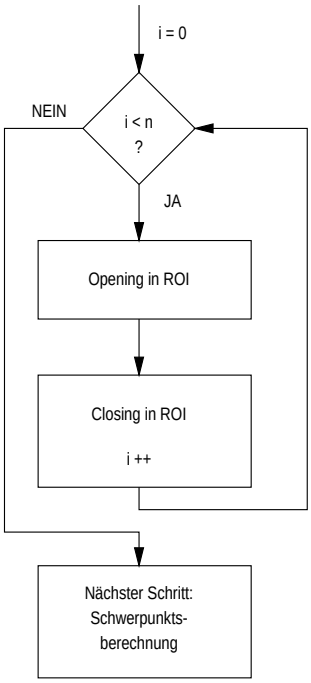

Abbildung 4.5: Ablauf der Bildnachbearbeitung durch morphologische Operatoren

3×3–Maske entsprechend (4.2) angenommen.

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.2}$$

4.4.1 Dilatation

Eine *Dilatation* ist eine binäre Faltung des Bildes mit einer Maske. Diese Operation bewirkt, dass kleine Löcher gefüllt werden und die Kontur geglättet wird. Sei $\boldsymbol{B}$ das Binärbild und $\boldsymbol{M}$ eine symmetrische $(2P+1) \times (2P+1)$–Maske, dann ist die Dilatation wie in (4.3) definiert:

$$\boldsymbol{B}'_{mn} = \bigvee_{m'=-P}^{P} \bigvee_{n'=-P}^{P} \boldsymbol{M}_{m',n'} \wedge \boldsymbol{B}_{m+m',n+n'} \tag{4.3}$$

Mit $\boldsymbol{B}'$ ist das neue Bild gemeint. Unter der Annahme, dass in dieser Maske nur Einsen stehen liefert diese Gleichung den Wert 1 (*weiß*) zurück, sobald mindestens ein weißes Pixel in ihrem Bereich liegt. Die Maske wird sukzessive über das Bild geschoben und es wird überprüft, ob sich im Bereich dieser Maske auf dem binären Bild

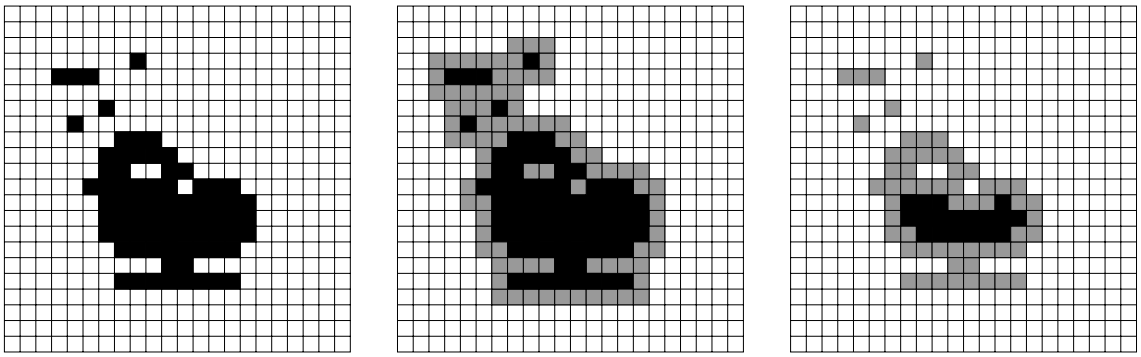

Abbildung 4.6: Beispiel einer Faltung mit einer 3×3–Matrix: **(a)** das Original, **(b)** Dilatation, **(c)** Erosion. Hinzugefügte Bildpunkte (bei der Dilatation) bzw. weggenommene Bildpunkte (bei der Erosion) sind grau dargestellt.

ein weißer Punkt befindet. Wenn das der Fall ist, wird der Bildpunkt, welcher im Zentrum der Maske liegt ebenfalls auf weiß gesetzt. Somit treten oben beschriebene Veränderungen nach einem Durchlauf auf. Gleichen Effekt hat auch ein Rangordnungfiltersoperator, der Maximumoperator, welcher in das unter dem Zentrum der Maske liegende Pixel den maximalen Wert der restlichen Pixel einträgt, welche sich im Bereich der Maske befinden.

4.4.2 Erosion

Die *Erosion* hat den gegenteiligen Effekt wie die Dilatation. Sie ist, um bei dem Vergleich mit den Rangordnungsfilteroperatoren zu bleiben, vergleichbar mit dem Minimumoperator, welcher in das zentral unter der Maske liegende Bildpixel das Minimum der unter der Maske liegenden Pixel einträgt. Gleichung (4.4) zeigt die Erosion in Form einer binären Faltungsoperation.

$$\boldsymbol{B}'_{mn} = \bigwedge_{m'=-P}^{P} \bigwedge_{n'=-P}^{P} \boldsymbol{M}_{m',n'} \wedge \boldsymbol{B}_{m+m',n+n'} \tag{4.4}$$

Das Ergebnis von (4.4) ist nur dann Eins, wenn sich die Maske vollständig im Objekt befindet. Objekte, die kleiner als die Maske sind, werden eliminiert. Objekte, welche nur durch eine schmale "Brücke" verbunden sind, werden getrennt. Ohne nähere Beweise, welche in [Jäh97] nachzulesen sind, sei hier noch festgestellt, dass sowohl der Erosions- wie auch der Dilatationsoperator verschiebungsinvariant, monoton,

distributiv[1] aber im allgemeinen nicht kommutativ sind.

4.4.3 Opening

Mit den elementaren Erosions- und Dilatationsoperationen kann man weitere Operatoren für die Bearbeitung von Objektformen schaffen. Die Erosion wird benutzt (wie schon in Kapitel 4.4.2 erwähnt), um kleine Objekte zu entfernen. Allerdings hat sie auch den Nachteil, dass alle im Bild verbleibenden Objekte kleiner werden (was man auch in Abbildung 4.6 sehr gut sehen kann). Wendet man anschließend aber eine Dilatation an, so kann dieser Effekt vermieden werden. Eine Erosion mit anschließender Dilatation nennt man *Opening*. (4.7) veranschaulicht dieses.
Um diese Gleichung zu verstehen werde Erosion und Dilatation noch einmal auf eine andere Art und Weise definiert. Dabei handelt es sich nur um die Betrachtung als Mengenoperationen mit Bildpunkten. Sei G die Menge aller Pixel der Bildmatrix, die nicht leer ist. M ist die Menge der Maskenpixel, welche ungleich Null sind. Mit M_ξ bezeichnet man die mit ihrem Referenzpunkt[2] zum Bildpunkt ξ verschobene Maske. Dann wird die Erosion mit

$$G \ominus M = \{\xi : M_\xi \subseteq G\} \tag{4.5}$$

und die Dilatation mit

$$G \oplus M = \{\xi : M_\xi \cap G \neq \varnothing\} \tag{4.6}$$

definiert. Mit Hilfe dieser Gleichungen kann man nun das Opening folgendermaßen formulieren:

$$G \circ M = (G \ominus M) \oplus M \tag{4.7}$$

Das Opening siebt also alle Objekte heraus, die das Strukturelement (im Implementierungsfall ein 3×3–Quadrat) in keinem Punkt vollständig enthalten, vermeidet aber die Größenreduktion aller Objekte. Somit lässt sich das Opening gut einsetzen, um Linien zu entfernen, deren Dicke geringer als das Strukturelement ist.

4.4.4 Closing

Die Dilatation vergrößert Objekte und füllt kleine Löcher und Risse oder Spalten aus. Eine der Dilatation nachfolgende Erosion kann die Vergrößerung der Objekte

[1] Die Erosion im Bezug auf die Schnittmengenoperation und die Dilatation in Bezug auf die Vereinigungsoperation

[2] Meist, wie auch bei dieser Arbeit, handelt es sich beim Referenzpunkt um das Zentrum der Maske. Dies muss aber im allgemeinen nicht zwingend so sein.

wieder rückgängig machen. Diese Kombination von Dilatation und Erosion wird *Closing* genannt:

$$G \bullet M = (G \oplus M) \ominus M \tag{4.8}$$

Folgende Beziehungen fassen die Änderung der Fläche der Objekte durch unterschiedliche Operationen zusammen:

$$G \ominus M \subseteq G \circ M \subseteq G \subseteq G \bullet M \subseteq G \oplus M \tag{4.9}$$

Opening und Closing sind idempotente Operationen, was bedeutet, dass eine zweite Anwendung einer Opening- oder Closing-Operation mit dem gleichen Strukturelement keine weitere Veränderung bewirkt. (siehe (4.10))

$$\begin{aligned} G \bullet M &= (G \bullet M) \bullet M \\ G \circ M &= (G \circ M) \circ M \end{aligned} \tag{4.10}$$

4.5 Schwerpunktsberechnung

Nachdem das Bild für die Schwerpunktsberechnung vorbereitet wurde, können nun verschiedene Eigenschaften des Schwerpunktes berechnet werden. Sei **S** die Menge aller in der *region of interest* (ROI) segmentierten Bildpunkte x. Dann berechnet sich die horizontale Position des Schwerpunktes wie folgt:

$$posX(SP) = \frac{1}{n} \sum_{i=1}^{n} posX(x_i) \quad \text{mit} \quad x_i \in \mathbf{S} \tag{4.11}$$

Die vertikale Position des Schwerpunktes berechnet sich analog. Bei (4.11) bedeutet $posX(SP)$, die X–Koordinate des Schwerpunktes SP. Das Bild wird Pixel für Pixel durchgegangen und es wird überprüft, ob der betrachtete Pixel zu der Menge der segmentierten Bildpunkte gehört. Die X–Positionen aller Pixel, bei denen das der Fall ist, werden aufaddiert und schließlich durch ihre Anzahl geteilt. Es wird allerdings nicht nur die Position des aktuellen Schwerpunktes gemessen. In der zweiten Iteration wird noch die Geschwindigkeit des Schwerpunktes gemessen. Bei der hier dargelegten Implementierung wurde für das Zeitintervall 1 (= ein Schritt) festgesetzt. Nach der zweiten Iteration wird zusätzlich noch die Beschleunigung des Schwerpunktes (natürlich jeweils auch in X– und Y–Richtung) bestimmt. Unter Zuhilfenahme dieser Informationen kann für die nächste Iteration die voraussichtliche Position des Schwerpunktes im Kamerabild berechnet werden. Kapitel 4.8 beschäftigt sich näher damit.

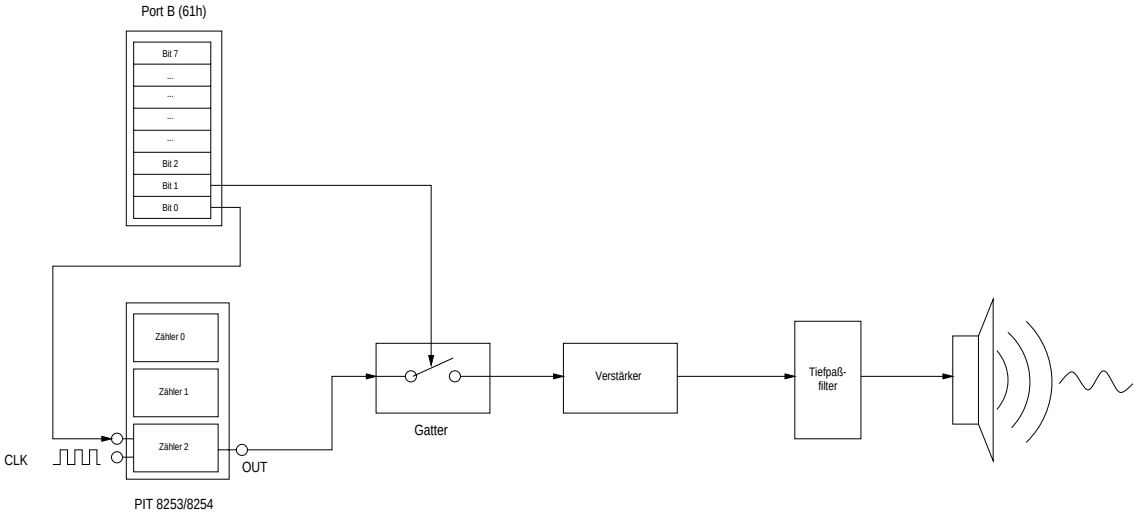

Abbildung 4.7: Blockdiagramm für die Verbindung zwischen PIT, Port B und Lautsprecher nach [Mes99]

4.6 Akustische Signale

Um eine gewisse Statusanzeige bzw. Kommunikation zwischen Roboter und Mensch während einer Folgefahrt zu ermöglichen, verfügt der Roboter über verschiedene akustische Signale, um dem vor dem Roboter stehenden Getrackten den jeweiligen Trackingstatus mitzuteilen. In Kapitel 4.6.1 wird kurz erklärt, wie der interne PC-Lautsprecher von MORTIMER angesteuert wird. Teilkapitel 4.6.2 beschäftigt sich mit dem akustischen Verhalten des Roboters während einer Folgefahrt.

4.6.1 Ansteuerung des Lautsprechers

Wichtig für den Betrieb des Lautsprechers ist der Zähler 2 des Timer-Chips 8253/8254 sowie die Bits 0 und 1 von Port B des PPI-Chips 8255. Der Ausgang OUT des Zählers 2 ist im PIT 8253/8254 über ein Gatter mit dem Lautsprecher verbunden, um einen gleichmäßigen Ton erzeugen zu können. Der Tiefpaßfilter filtert alle Frequenzen heraus, welche zu hoch sind, um vom Lautsprecher wiedergegeben werden zu können. Der CLK2-Eingang des PIT ist, wie alle sonstigen Taktgeneratoren auch mit dem 1,19318 MHz-Oszillator verbunden. Hier wurde für die Erzeugung des Tons die direkte Ansteuerung des Lautsprechers über den Zähler 2 des PIT 8253/8254 benutzt, da sie für eine Tonerzeugung ohne Beteiligung der CPU sorgt. Der Prozessor wird lediglich für die Einstellung der Tonfrequenz, das Einschalten und das Ausschalten des Lautsprechers benötigt. Zur Erzeugung eines Tones muss der Timer-Chip ein Rechtecksignal erzeugen. Dafür kommt nur der Modus 3 des

PIT in Frage, da der Modus 2 nur Nadelimpulse mit einer sehr geringen Breite erzeugt. Die Membran des Lautsprechers ist zu träge, um auf die kurzen Nadelimpulse reagieren zu können, was sich dahingehend auswirken wird, dass der Lautsprecher stumm bleiben wird. Aufgrund dieser Tatsache muss der PIT durch ein Steuerwort in den Zählmodus 3 versetzt werden. Danach werden zwei Zählerbytes übergeben, von denen das erste als Teiler wirkt. In dieser Implementierung wurde der Teiler so gewählt, dass der Kammerton a (=440Hz) ausgegeben wird. Der Port B kann über die Adresse 61h angesprochen werden. Weist man dem Bit 0 dieses Ports den Wert 1 zu, so wird im Modus 3 gezählt. Das Rechtecksignal von Zähler 2 wird nur dann zum Verstärker übertragen, wenn Bit 1 gesetzt (=1) ist und somit das Gate geschlossen ist. Um den Lautsprecher schließlich wieder auszuschalten genügt es, eines der beiden Bits (0 oder 1) zu löschen. In [Mes99] ist eine noch detailliertere Beschreibung dieses Vorganges zu finden.

4.6.2 Ausgabe zustandsabhängiger akustischer Signale

In Abbildung 4.8 wird das gesamte akustische Ausgabeverhalten in Abhängigkeit vom jeweiligen Trackingzustand und verschiedener Eingaben mit Hilfe eines Mealy-Automaten dargestellt.

Erklärung der Zustände:

Idle: Wenn die Folgefahrt beim Roboter aktiviert wird (siehe 4.9) befindet sich der Roboter im Zustand Idle. Dies ist der Startzustand des Trackingvorganges. Wenn in den vorverarbeiteten Bildern in der ROI weniger als 5% der Fläche der ROI aus segmentierten Bereichen besteht, verbleibt der Roboter im Zustand *Idle*. Bei allen anderen Eingaben wechselt er in den Zustand *Tracking* und gibt einen kurzen Piepton von sich.

Tracking: Im Zustand *Tracking* spielt sich der Hauptteil der Folgefahrt ab. Hier, wie auch im Zustand *InputTrace* entstehen die Eingaben, welche später zu den Fahrbefehlen für den Roboter führen. Wenn die segmentierten Bereiche zwischen 15% und 85% bewegen, dann bleibt der Roboter im Zustand *Tracking* und gibt auch kein akustisches Signal von sich. Falls aber der gesamte segmentierte Bereich weniger als 15% oder mehr als 85% der ROI ausmacht, gibt der Roboter im ersten Fall einen kurzen Doppelpiepton aus, welcher anzeigt, dass der Roboter im Begriff ist, das zu trackende Objekt zu verlieren, und im zweiten Fall einen langen Piepton, welcher signalisiert, dass das zu trackende Objekt zu nah am Roboter bzw. vor dessen Kamera ist. In beiden Fällen wechselt er in den Zustand *Input_Trace*. Falls der gesamte segmentierte Bereich in der ROI weniger als 5% beträgt, wechselt der Roboter in den Zustand *Lost_1*.

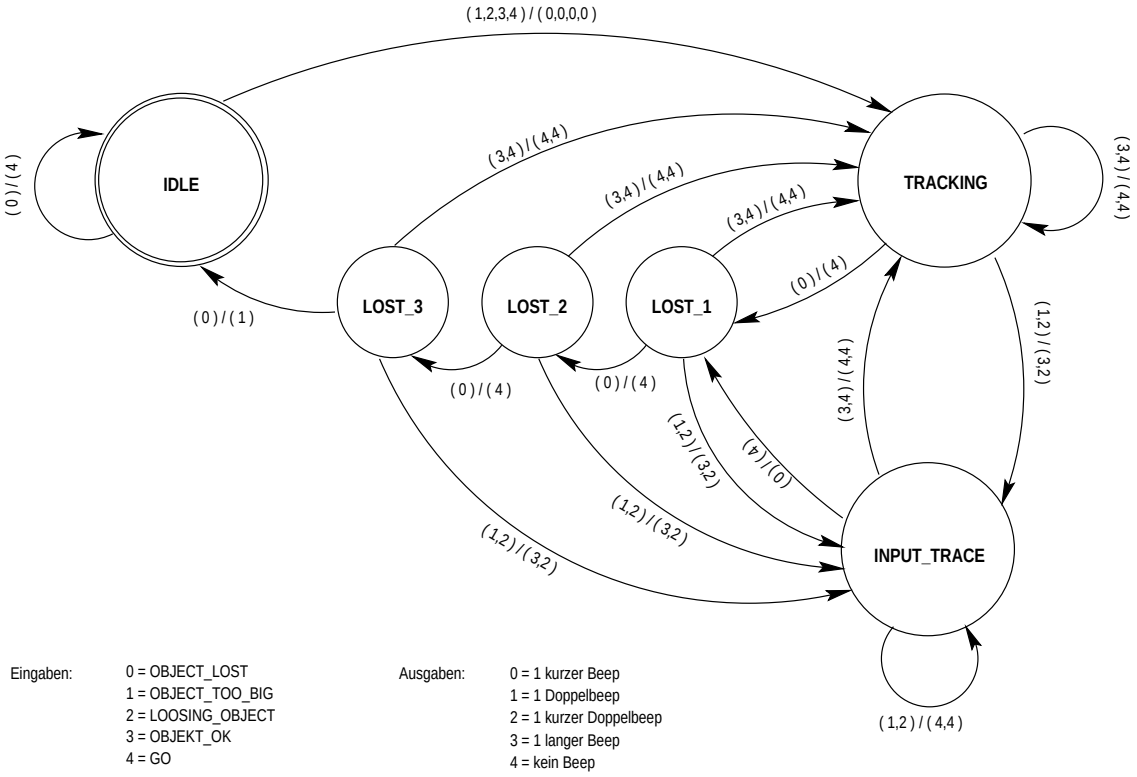

Abbildung 4.8: Mealy-Automat für das akustische Ausgabeverhalten

Input_Trace: Dieser Zustand wurde dem ursprünglichen Automaten hinzugefügt, um zu verhindern, dass unnötig viele akustische Signale ausgegeben werden. So reicht es zum Beispiel völlig aus, nur *einen* langen Piepton auszusenden, wenn das zu trackende Objekt dauernd zu nah und damit zu großß ist. Somit erklären sich auch die Zustandsübergänge, welche der Abbildung 4.8 zu entnehmen sind. Falls sich die Objektgröße wieder im normalen Bereich befindet, dann wechselt der Roboter zurück in den Zustand *Tracking*, wenn allerdings das Objekt zu klein ist ($< 5\%$ der ROI), wird in den Zustand *Lost_1* gewechselt. Dies geschieht alles aus oben beschriebenen Gründen ohne jegliche akustische Ausgabe.

Lost_1, Lost_2, Lost_3: Es kann vorkommen, dass zum Beispiel im Rahmen von mehreren Bildern die segmentierte Fläche zwischen 3% und 7% der ROI schwankt. Damit wäre einerseits manchmal das Objekt verloren, andererseits aber bloß im Begriff, verloren zu werden. Um in diesem Fall sicherzustellen, dass der Roboter nicht von Bild zu Bild zwischen Zustand *Tracking* und *Idle* hin und her pendelt, wurde die Gruppe der *Lost*-Zustände eingeführt. Auf

diese Weise wird ein Objekt erst als "verloren" anerkannt, wenn in vier aufeinanderfolgenden Bildern weniger als 5% der ROI-Fläche dem Objekt zugeordnet werden konnte. Dies wirkt sich auch auf die Vorhersage im Kamerabild aus (siehe dazu Kapitel 4.8). Falls der Roboter in einem der *Lost*-Zustände ist und das Objekt als zu groß oder im Begriff verloren zu werden, erkannt wird, geht der Roboter mit entsprechenden akustischen Signalen in den Zustand *Input_Trace* über. Er tut dies deshalb, weil damit dem Benutzer die *Lost*-Zustände völlig verborgen bleiben und er anhand der akustischen Signale meint, noch im Zustand *Tracking* zu sein. Fall das Objekt wieder als "verloren" betrachtet wird, wechselt der Roboter in den nächsten *Lost*-Zustand, oder (im Falle des Zustandes *Lost_3*) in den Zustand *Idle*.

4.7 Digitale Regelung

Im Allgemeinen gibt es zwei Möglichkeiten, um physikalische Größen, wie in diesem Fall den gesamten Antrieb des Roboters, gezielt zu beeinflussen: die *Steuerung* und die *Regelung*. Dabei sind Steuerung und Regelung nach DIN 19226 (Teil 1) wie folgt beschrieben.

Steuerung:

> Das *Steuern*, die *Steuerung*, ist ein Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungskreis.

Der offene Wirkungsweg bedeutet, dass die gesteuerte Größe nach einem Soll-Ist-Vergleich *nicht* mehr auf sich selbst zurückwirkt. Dadurch können intern auftretende Störgrößen nicht berücksichtigt werden.

Regelung:

> Das *Regeln*, die *Regelung*, ist ein Vorgang, bei dem eine Größe, die zu regelnde Größe (Regelgröße), fortlaufend erfasst, mit einer anderen Größe, der *Führungsgröße*[3], verglichen und abhängig vom Ergebnis dieses Vergleichs im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.

Den Wirkungsplan eines Reglers kann man in Abbildung 4.9 betrachten. Da im Fall der Folgefahrt interne Störungen, wie z.B. die Bewegung des zu trackenden Objektes

[3] Der Sollwert ist der Momentanwert der Führungsgröße

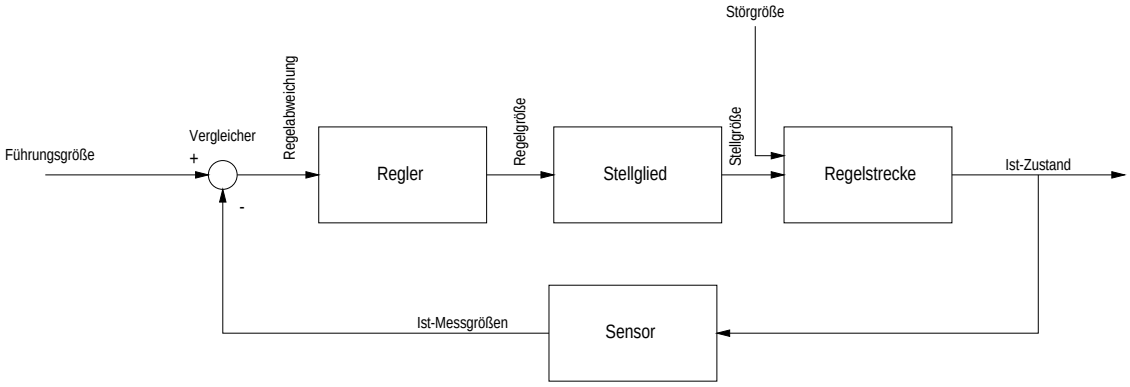

Abbildung 4.9: Wirkungsplan eines Regelkreises

oder zu langsames Abbremsen, auftreten, wurde hier ein Regler verwendet.
Nach [MSF97] lautet das mathematische Modell eines als verzögerungsfrei angenommenen analogen PID-Reglers:

$$y(t) = K_{\mathrm{PR}} \left(e(t) + \frac{1}{T_n} \int_0^t e(\tau) d\tau + T_v \dot{e}(t) \right) \quad \text{mit} \quad e(t) = w(t) - x(t) \tag{4.12}$$

Bei (4.12) sind $w(t)$ die Führungsgröße, $x(t)$ die Regelgröße, $e(t)$ die Regeldifferenz, $y(t)$ die Stellgröße, K_{PR} die Proportionalitätskonstante, T_n die Nachstellzeit und T_v die Vorhaltezeit. Zu sehen ist auch, dass der PID-Regler aus den drei Anteilen, Proportionalanteil, Integralanteil und Differentialanteil, besteht. Dabei kann (4.12) so zerlegt werden, dass für jeden Anteil einen eigener konstanter Faktor existiert. Die Integration wird im digitalen Fall nur approximiert. Hier wurde eine Lösung der numerischen Integration gewählt. Dabei wurde folgende Näherung verwendet:

$$\int_0^{kT} e(\tau) d\tau \approx \sum_{i=0}^{k} \frac{e_i + e_{i-1}}{2} T \tag{4.13}$$

Nach einigen Umformungen lautet die komplette Gleichung für einen digitalen PID-Regler, welcher für die Integration die Trapezregel verwendet (wie auch bei (4.13)), wie folgt:

$$y_k = y_{k-1} + K_{\mathrm{PR}} \left(1 + \frac{T}{2T_n} + \frac{T_v}{T} \right) e_k - K_{\mathrm{PR}} \left(1 - \frac{T}{2T_n} + 2\frac{T_v}{T} \right) e_{k-1} + K_{\mathrm{PR}} \frac{T_v}{T} e_{k-2} \tag{4.14}$$

Für eine genaue Herleitung von (4.14) sei auf [MSF97] verwiesen.

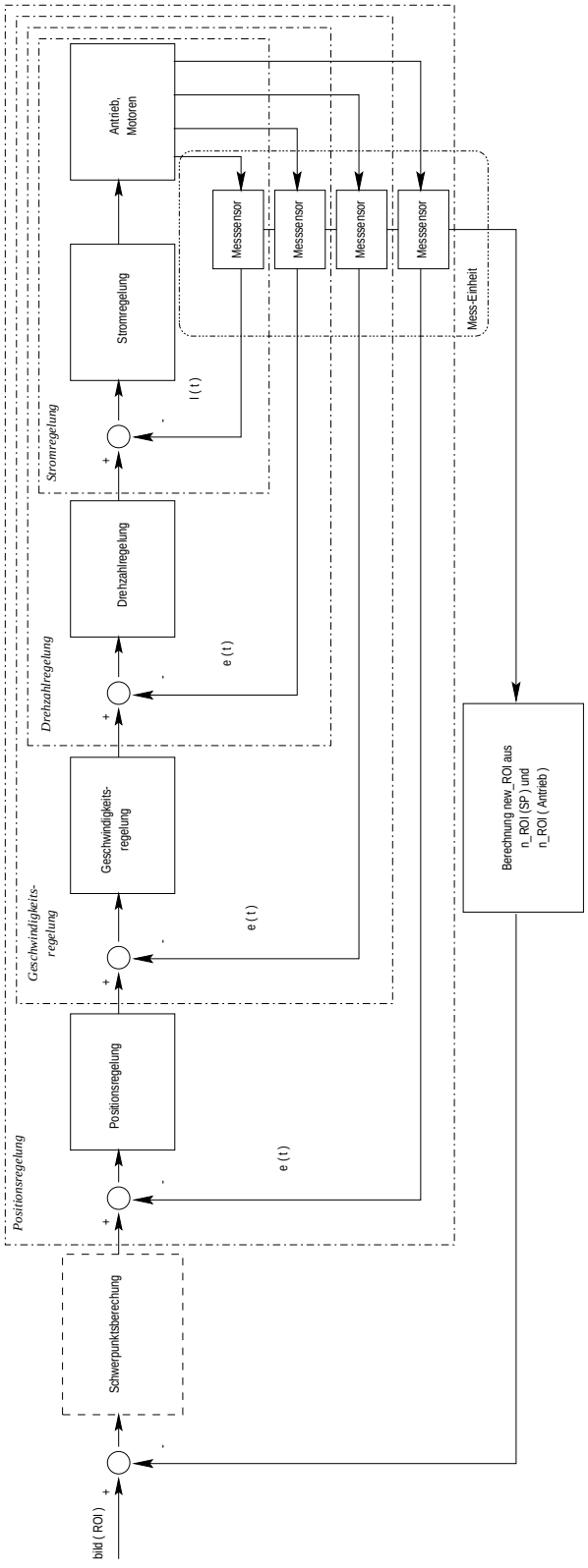

Abbildung 4.10: Regelkreis für Folgefahrt

4.7.1 Fahrbefehle

Die Fahrbefehle für den Roboter sind wie im folgenden Quellcode-Ausschnitt zu sehen, aufgebaut:

```
Robot_Message[0] = 's';
Robot_Message[1] = (char) INPUT_JOINTS;
memcpy ( Robot_Message+2, &mortimer_pos, sizeof ( ROBOT_POSITION ) );
rp_robot_send_cmd ( robsocket, Robot_Message, sizeof ( ROBOT_POSITION ) + 2 );
```

Dabei setzt sich die dem Roboter übergebene Zeichenkette auf mehreren Teilen zusammen. Das erste Byte gibt eine Kategorisierung der Zeichenkette an. Damit ist gemeint, dass aus dem ersten Byte erkennbar ist, an was die Zeichenkette gerichtet ist. So bedeutet z.B. 'b' das die Nachricht für die Bremsen (breaks) bestimmt ist. 's' steht für das "Setzen" eines neuen Wertes. Die interessante Datenstruktur versteckt sich hinter INPUT_JOINTS. In ihr sind alle Gelenkwinkel, Geschwindigkeiten und Beschleunigungen enthalten. Für jedes Gelenk existieren somit Werte für die aktuelle Position und den Gelenkwinkel α. Durch den Sendebefehl wird die gesamte Zeichenkette an den Roboter übermittelt, anschließend werden die in der Datenstruktur INPUT_JOINTS enthaltenen Informationen über die zu erreichende Position auswertet und dementsprechend der Antrieb angesteuert.

4.7.2 Einteilung

Theoretisch würde es reichen, für das gesamte Bild nur eine Geschwindigkeitsstufe der rotatorischen Komponente zuzuordnen. Dementsprechend wäre es dann egal, in welchem Bildbereich sich gerade der Schwerpunkt des segmentierten Bereichs befindet. Allerdings birgt dieser Ansatz auch ein Problem. Das Problem liegt in der Wahl der idealen Geschwindigkeit. Zum genauen Justieren[4] ist eine relativ geringe Geschwindigkeit von Vorteil, um zu große überschwingvorgänge zu vermeiden. Andererseits ist es auch von Vorteil, wenn der Roboter sich relativ schnell dreht, um einen sich gleichmäßig schnell bewegenden Schwerpunkt gut verfolgen zu können. Leider muss bei der langsamen Geschwindigkeit in Kauf genommen werden, dass das zu trackende Objekt verloren wird, sobald es sich mit einer Geschwindigkeit bewegt, welche höher ist, als die zur Feinorientierung Ideale. Die hohe Drehgeschwindigkeit führt unausweichlich zu einem starken und damit schlechten Überschwingverhalten, was eine Feinorientierung quasi unmöglich macht. Diese Betrachtungsweise ist in ähnlicher Weise auch für die translatorische Geschwindigkeitskomponente gültig. Aus diesen Gründen wurden verschiedene Geschwindigkeiten, einerseits verschiedenen Bildbereichen[5] und andererseits verschiedenen Objektgrößen des segmentier-

[4] Wenn sich zum Beispiel der Schwerpunkt in der Nähe des Bildmittelpunktes befindet und sich nur langsam oder gar nicht bewegt

[5] Für die rotatorische Geschwindigkeit

ten und nachbearbeiteten Bereichs[6] zugeordnet. Auf diese Weise ist eine relativ differenzierte Folgefahrt möglich, deren Granularität natürlich von der Anzahl der unterschiedlichen Geschwindigkeiten abhängt.

4.7.3 Einbindung weiterer Sensoren

Es bleibt noch aus Sicherheitsgründen zu gewährleisten, dass der Roboter während der Folgefahrt nicht mit anderen Gegenständen kollidiert. Für diese Gewährleistung eignen sich die hier benutzen Kamerabilder nicht, so dass für die Lösung dieses Problems auf Informationen anderer Sensoren zurückgegriffen werden muss. Im Falle der Folgefahrt hat sich der Laserscanner (siehe Kapitel 2.2.1) als Hilfe gezeigt. Da schon auf den Laserscanner eingegangen wurde, werden hier nur die für die Lösung dieses Problems wesentlichen Aspekte betrachtet. Es werden (ähnlich, wie bei der Zutei-

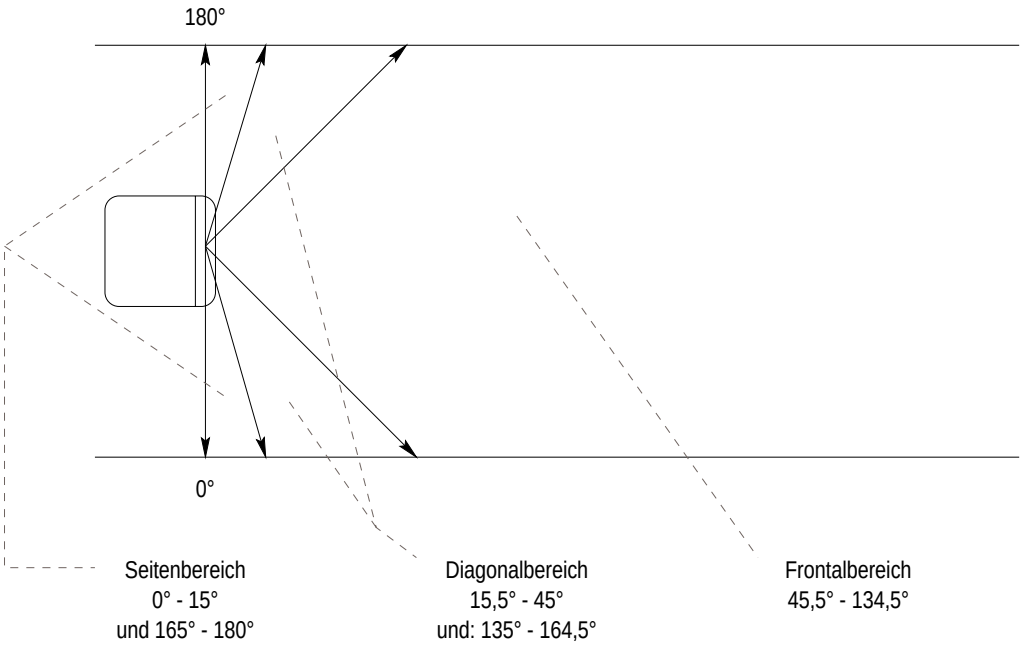

Abbildung 4.11: Sensitivitätsbereiche des Laserscanners zur differenzierten, bereichsabhängigen Kollisionsvermeidung

lung verschiedener Geschwindigkeiten) verschiedene Bereiche unterschiedlich bewertet (siehe Abbildung 4.11). Dabei handelt es sich um einen Seitenbereich, Diagonalbereich und Frontalbereich. Dabei teilen sich Seitenbereich und Diagonalbereich in jeweils zwei Unterbereiche auf (links und rechts). In jeder Iteration des Trackingprozesses wird der gesamte Bereich des Laserscanners abgefragt und das Minimum

[6] Für die translatorische Geschwindigkeit

$\zeta_{\min}$ der Messwerte ζ für jeden Bereich berechnet. Der Roboter darf manövrieren, wenn die Funktion $F_{\text{koll}} > 0$ ist. Dabei sieht F_{koll} wie folgt aus:

$$F_{\text{koll}} = \begin{cases} \zeta_{\min} - 40 & \text{für Seitenbereich} \\ \zeta_{\min} - 57 & \text{für Diagonalbereich} \\ \zeta_{\min} - 80 & \text{für Frontalbereich} \end{cases} \tag{4.15}$$

Dabei entsprechen die Zahlenwerte den Minimalabständen der verschiedenen Bereiche, gemessen in Zentimetern.

4.8 Vorhersage im Kamerabild

Die Vorhersage im Kamerabild beschäftigt sich mit der Approximation der Position des Schwerpunktes im in der nächsten Iteration eingelesenen Bild. Diese Approximation soll dem Roboter ermöglichen, auch bei relativ (gleichmäßig) hohen Geschwindigkeiten, das zu trackende Objekt nicht zu verlieren. Dabei gehen Parameter wie die aktuelle Position x und y, die Geschwindigkeiten v_x und v_y, sowie die Beschleunigungen a_x und a_y des Schwerpunktes ein. Es werden die üblichen Formeln zur Berechnung der Geschwindigkeit und Beschleunigung verwendet. Die Zeitdifferenz wurde in der Entwicklungsphase unter Linux mit einer Funktion gettime(); bestimmt, welches sich unter VxWorks bisher nicht realisieren ließ. Deshalb wird die Zeit nicht in Sekunden gemessen, sondern in Iterationsschritten, mit der Annahme, dass die Dauer der Iterationen nur unwesentlich schwankt. Deshalb wurde auch folgende zeitunabhängige Umformung verwendet, um die neue Position des Schwerpunktes zu berechnen:

$$\Delta x = \left(\frac{v_{x_i}^2 - v_{x_{i-1}}^2}{2a_x} \right) \quad \text{und} \quad \Delta y = \left(\frac{v_{y_i}^2 - v_{y_{i-1}}^2}{2a_y} \right) \tag{4.16}$$

Die neue Position errechnet sich dann aus $x_{\text{neu}} = x_{\text{akt}} + \Delta x$ und $y_{\text{neu}} = y_{\text{akt}} + \Delta y$. Dabei unterscheiden sich die ersten Iterationen von den Restlichen. In der ersten Iteration wird die Position des aktuellen Schwerpunktes bestimmt und mit ihr die erste Geschwindigkeit[7] berechnet. Ab der zweiten Iteration erfolgt dann auch noch zusätzlich die Berechnung der Beschleunigung, wofür man ja die letzten beiden Geschwindigkeiten benötigt.
Einen Sonderfall der Approximation stellt der Übergang vom Status *Lost_3* zum Status *Idle* dar. Da es schlecht von dem Getrackten zu erwarten ist, dass er sich jedes Mal, wenn der Roboter das Objekt verliert, die Position merkt, an der dieses geschehen ist[8], wird in diesem Fall der approximierte Schwerpunkt wieder auf

[7] Die Position des Schwerpunktes wird in der nullten Iteration im Bildmittelpunkt angenommen

[8] Zumal das Objekt ja laut Kapitel 4.6.2 schon theoretisch vier Iterationen vorher verloren gegangen sein kann.

den Bildmittelpunkt gesetzt. Hier tritt das Problem auf, dass der Tracking-Prozess nun glauben könnte, dass sich der Schwerpunkt innerhalb einer Iteration von seiner vorherigen Position in den Bildmittelpunkt bewegt hätte. Die Entstehung dieses Fehlers wird durch das Zurücksetzen des gesamten Prozesses auf seinen Anfangszustand vermieden.

4.9 Hardwareanpassung

Da es zu mühselig ist, vor jedem Tracking das entsprechende Modul entweder über eine feste Netzwerkverbindung oder Funkmodem zu starten, ist das Tracking-Modul in den Roboter-Kernel eingebaut. Ein Kippschalter (siehe Abbildung 4.12) wurde an der Front des Roboters angebracht, mit dem es möglich ist, den Trackingprozess zu starten und auch wieder zu beenden. Dazu wird beim Hochfahren des Roboters ein

Abbildung 4.12: Schalter zum Ein- und Ausschalten des Tracking-Prozesses

paralleler Prozess gestartet, welcher fortlaufend, alle 10ms, die parallele Schnittstelle, mit welcher auch die Kameras verbunden sind, abfragt. An dieser Schnittstelle wurde der Schalter mit den Pins 13 und 25 verbunden. Dabei repräsentiert der Pin 13 den Status "SELECTED" und Pin 25 die Masse. Bei dem Übergang $1 \longrightarrow 0$ zwischen den Pins dieser Schnittstelle wird der Trackingprozess gestartet, welcher ständig eine globale Variable abfragt. Der Schalter steht jetzt auf "ON". Wenn der Schalter wieder auf "OFF" gesetzt wird, was dem Übergang $0 \longrightarrow 1$ entspricht, wird diese globale Variable so gesetzt, dass keine weitere Iteration des Trackingprozesses mehr erfolgt und dieser beendet wird. Der ursprüngliche Zustand ist wieder hergestellt. Dieser Prozess lässt sich beliebig oft wiederholen. Abbildung 4.13 gibt einen Überblick, wie der Schalter an die DCE25-Schnittstelle angeschlossen ist.

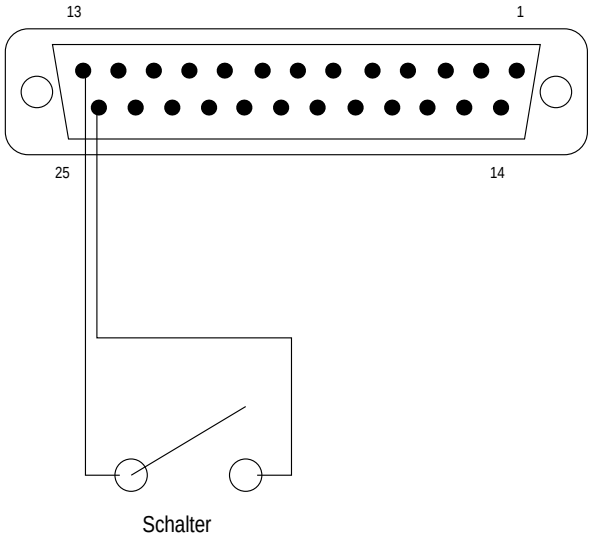

Abbildung 4.13: Schaltung zur Ansteuerung des DCE25

Kapitel 5

Experimente

Der Sinn eines Experimentes ist es, die entwickelte Software in einer definierten Umgebung zu testen, um dann aus dem festgestellten Verhalten und den sich daraus ergebenden Differenzen zum approximierten Verhalten Rückschlüsse auf die Qualität der Methoden zu ziehen.

5.1 Konzeptionierung

Da es sich bei dieser Arbeit um die Realisierung einer Folgefahrt handelt, wurden Experimente durchgeführt in welchen alle für eine Folgefahrt notwendigen Funktionen zum Einsatz kommen. Ein Experiment wird an dieser Stelle skizziert.
Um in den Screenshots, die aus dem Umgebungsmodell von MEPHISTO[1] gewonnen werden, sehen zu können, wo sich gerade das getrackte Objekt[2] im Moment der Aufnahme befindet, wurde die Konvention getroffen, dass die getrackte Person eine vordefinierte Haltung einnimmt (siehe Abbildung 5.1). Somit ist auf dem Laserscan die Position der Hand durch die zu erkennende Position der Beine festgemacht.

5.2 Durchführung

In Abbildung 5.2 ist die Konzeptionierung und Durchführung des Experimentes aufgezeigt. Dabei besteht die Aufgabe, durch Tracking den Roboter zwischen den beiden Hindernissen (hier als Kreise dargestellt) hindurchzumanövrieren. Das Experiment endet, wenn der Getrackte in der Raumecke steht (hier: unten links). Dazu wird der Roboter zuerst links herum gedreht, dann geht man etwas schneller um das erste Hindernis herum, um den Roboter auf dieses auffahren zu lassen. Hier soll die Funktion der Laserscannerabfragen überprüft werden. Das approximierte Verhalten

[1] Modular and Extensible Path Planning System using Observation
[2] Im allgemeinen die Hand

Abbildung 5.1: Körperhaltung während des Experimentes. Die Position der Hand befindet sich im Laserscan zwischen den Beinpeaks.

sieht so aus, dass der Roboter vor dem Hindernis zum Stehen kommt und sich nur noch drehen lässt. Durch eine weitere Linksdrehung (in Abbildung 5.2 durch das retrograde Durchschreiten der Hindernisse angedeutet) positioniert man den Roboter so, dass er sich vor der Öffnung zwischen beiden Hindernissen befindet. Nun wiederholt man das Umlaufen (diesmal des zweiten Hindernisses), um den Roboter passend zum Passieren der Hindernisse zu orientieren. Wenn alle Abstandsparameter ein Weiterfahren erlauben, fährt der Roboter dem Getrackten nach, welcher sich schon ein wenig vom Roboter entfernt hat. Wenn der Getrackte sich am "Ziel" befindet, bewegt er sich nicht mehr. Der Roboter fährt nun das getrackte Objekt solange an, bis die minimal erlaubten Abstandswerte, oder die zum Fahren maximale Größe des getrackten Objektes im Bild erreicht werden.

5.3 Ergebnisse

Zum Lesen der Screenshots sind noch ein paar grundsätzliche Bemerkungen zu machen. Der Roboter ist durch ein Achteck simuliert. Die Linie in dem Achteck zeigt, wo sich die Frontseite des Roboters befindet. Vor dem Roboter ist der aktuelle Laserscan eingezeichnet. Das statische Umgebungsmodell besteht aus indifferenten Objekten, wie Tischen, Wänden und Schränken, welche alle als Rechtecke im Umgebungsmodell eingetragen sind. Die Aussparungen in den Wänden stellen Türöffnungen dar.

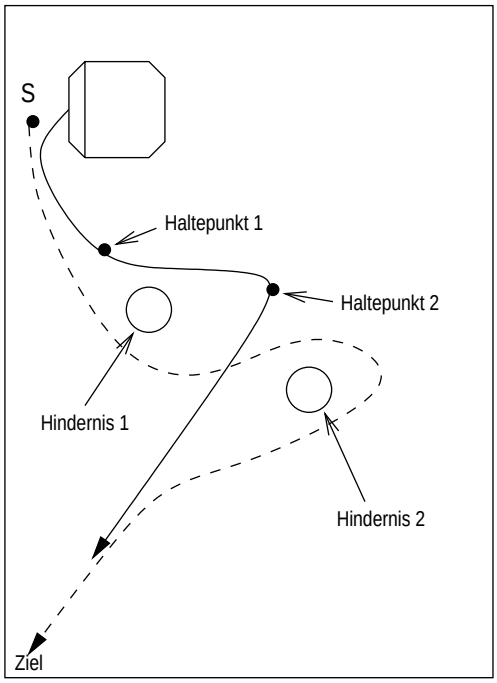

Abbildung 5.2: Konzeptionierung und Durchführung des Experimentes zur Durchführung einer Folgefahrt, Gestrichelte Linie: Weg der getrackten Person, durchgezogene Linie: Roboterbahn

Die Markierungen der Hindernisse[3] und der getrackten Person[4] sind nachträglich zur besseren Anschauung eingefügt worden. In Abbildung 5.3 ist die Anfangssituation des Experimentes zu sehen. Die beiden "Zacken" im Laserscan vor dem Roboter stellen Beine der getrackten Person dar. Ausserdem ist zu sehen, dass die Tür in der linken Wand geöffnet ist. Abbildung 5.4 zeigt den Roboter während der Linksdrehung. Der Laserscan gibt Auskunft über die Position der Beine des Getrackten. Die beiden Hindernisse sind ebenfalls schon zu erkennen. Beim ersten hindernisbedingten Halt des Roboters ist das linke Bein des Getrackten durch das davor stehende Hindernis verdeckt und daher nicht zu sehen. Das getrackte Objekt befindet sich direkt hinter dem ersten Hindernis. Der Roboter steht. Das Ergebnis der folgenden Bildauswertung löst die Übergabe einer positiven translatorischen Geschwindigkeit an die Motorsteuerung des Roboters aus. Diese wird jedoch ignoriert, da der vordefinierte Minimalabstand zu einem Objekt im Frontalbereich erreicht ist. Abbildung 5.5 verdeutlicht dies. Wie in Abbildung 5.6 zu sehen ist, wurde der Roboter durch das getrackte Objekt nach links gedreht und kann nun am ersten Hindernis vorbei aus

[3] durch Kreise, jeweils mit "H1" und "H2" gekennzeichnet
[4] durch eine Ellipse, mit "P" gekennzeichnet

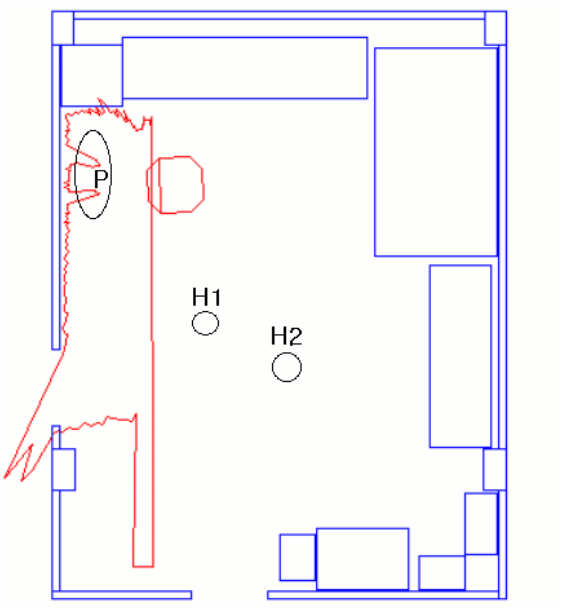

Abbildung 5.3: Startsituation des Experimentes. Im Laserscan zu erkennen: Die Beine des Getrackten (**P**), welcher sich vor dem Roboter stehend befindet.

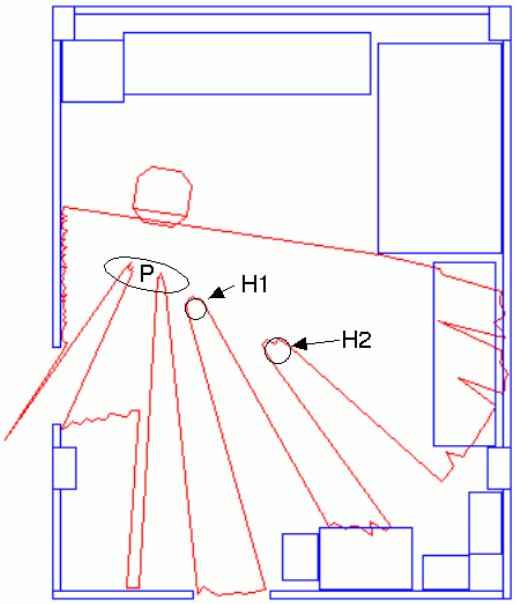

Abbildung 5.4: 2. Screenshot. Roboter während der Linksdrehung. Zu erkennen im Laserscan (von rechts nach links (vom Roboter aus)): Beine des Getrackten (**P**), dazwischen die geöffnete Tür, Hindernis 1 (**H1**) und Hindernis 2 (**H2**).

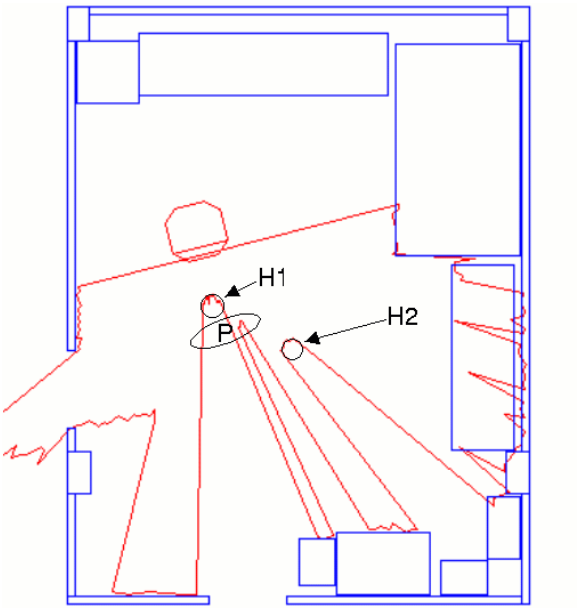

Abbildung 5.5: Situation beim ersten hindernisbedingten Halt. Beine des Getrackten teilweise hinter erstem Hindernis versteckt.

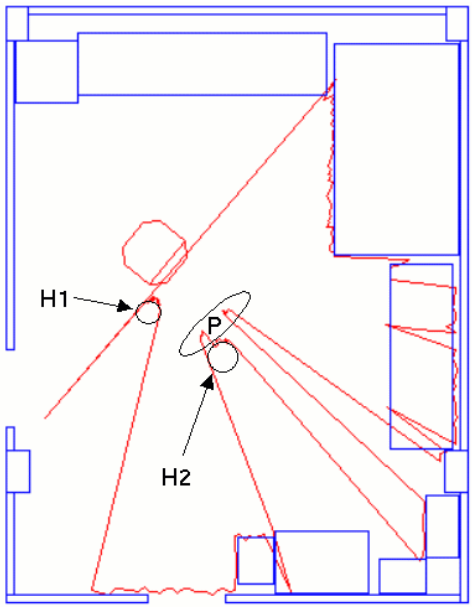

Abbildung 5.6: Vorbeifahren am ersten Hindernis. Annähern auf zweites Hindernis.

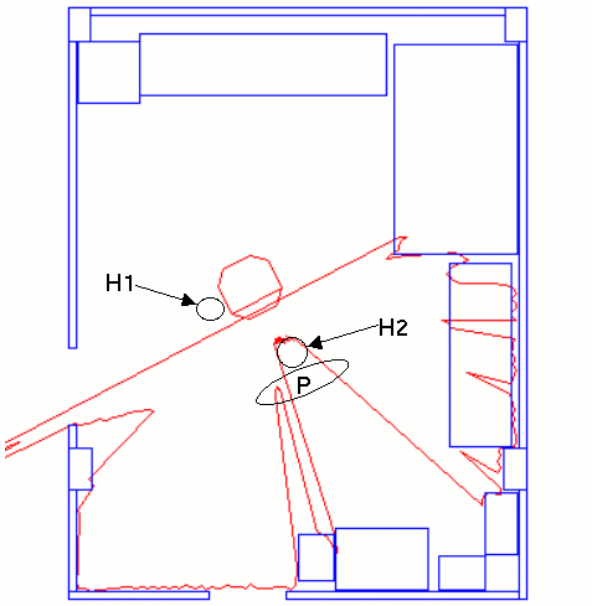

Abbildung 5.7: Zweiter hindernisbedingter Halt. Getrackte Person steht hinter dem Hindernis.

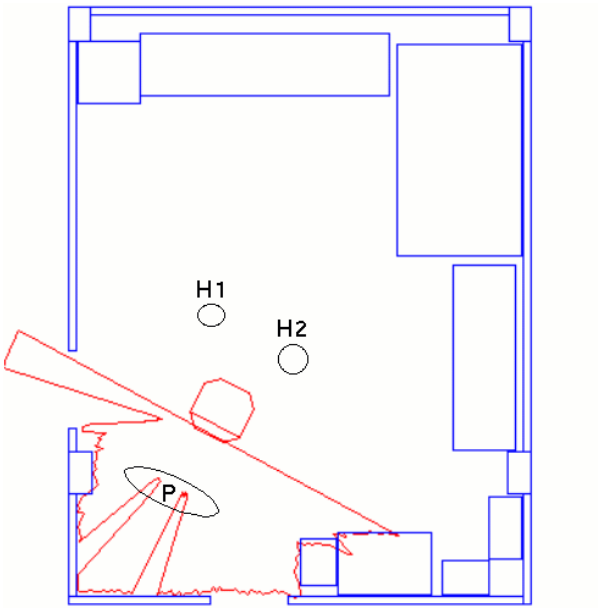

Abbildung 5.8: Situation am Ende des Experimentes. Der Roboter steht vor dem zu trackenden Objekt.

den Getrackten zu fahren. Dieser steht vor dem zweiten Hindernis. Die Situation, welche Abbildung 5.7 zeigt, ist vergleichbar mit derjenigen aus Abbildung 5.5. Im Laserscan ist hier das linke Bein der getrackten Person zu erkennen. Schließlich zeigt Abbildung 5.8 die Situation am Ende des Experimentes. Der Roboter ist zwischen den beiden Hindernissen durchgefahren und auf das zu trackende Objekt zu.

5.4 Analyse

Der Roboter reagiert auf die Bewegungen des zu trackenden Objektes. Bei relativ geringem Abstand zum Objektiv können leichte Pendeleffekte auftreten, welche aber mit einer leichten Handbewegung eliminiert werden können. Diese Effekte treten jedoch nicht auf, wenn das zu trackende Objekt mehr als 70 Zentimeter vom Roboter entfernt ist. Wenn man etwas zu weit vom Roboter entfernt ist, besteht die Möglichkeit, dass der Roboter den Kontakt verliert. Eine Bewegung der Hand durch die Bildmittelachse der Kamera löst aber dieses Problem. Es gibt mehrere Einstellungsmöglichkeiten, um dieses Problem stark einzuschränken:

- Veränderung der Einstellung der Blende
- Erhöhung der Bildverarbeitungsgeschwindigkeit durch Verzicht auf morphologische Bildnachbearbeitung
- Vergrößerung der Region of Interest (ROI)

Um optimale Ergebnisse zu bekommen, ist es notwendig, die Blende des Objektives an die gegebenen Lichtverhältnisse anzupassen.Die Verbesserung der Bildverarbeitungsgeschwindigkeit durch das Auslassen der morphologischen Operationen hat den Nachteil, dass die Qualität der Erkennung nachlässt und somit die Wahrscheinlichkeit auf Fehlerkennungen erhöht. Schließlich bietet das Vergrößern der Region of Interest eine bessere optische "Haftung" am zu trackenden Objekt, da es dadurch unwahrscheinlicher wird, dass das Objekt durch zu schnelle Bewegungen die ROI verlässt. Dieser Effekt wird durch den mit der Vergrößerung verbundenen quadratisch erhöhten Aufwand bei der Bildverarbeitung und der somit resultierenden langsameren Bildverarbeitungsgeschwindigkeit eliminiert.
Zu diesen Überlegungen sind noch zwei Fehlerquellen erwähnt: Fehlinterpretationen bei der Hautsegmentierung und Tracken bei Tageslicht. Bei der Hautsegmentierung kann es zu Fehlinterpretationen kommen, wenn sich Objekte in der ROI befinden, welche in das zu erkennende Farbschema passen. Dies trifft für bestimmte Rottöne aber auch in besonderem Maße für Kartonagen zu. Letztere werden unbefriedigend zuverlässig als Haut fehlinterpretiert. Eine zweite Fehlerquelle bieten die herrschenden Lichtverhältnisse. Dies macht sich dadurch bemerkbar, dass der Roboter den

Kontakt zum trackenden Objekt relativ früh verliert. Bei bestimmten Lichtverhältnissen ist sogar eine Folgefahrt nicht möglich. Gute Ergebnisse ließen sich bei normaler künstlicher Beleuchtung durch Leuchtstoffröhren erzielen.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel erfolgt eine Zusammenfassung der geleisteten Arbeit und ein Ausblick auf mögliche Weiterentwicklungen.

6.1 Zusammenfassung

Am Anfang der Konzeption eines Lösungsweges für diese Arbeit bestand die Frage, welches Objekt sich am besten eignet, um getrackt zu werden. Dabei waren mehrere Faktoren wie zum Beispiel Lichtunabhängigkeit, Form, Farbe, Verfügbarkeit etc. zu berücksichtigen. Im Zuge dieser Überlegungen kam man zum Entschluss, dass sich Haut für ein solches Vorhaben am besten eignet. Vorhandene Erkenntnisse aus dem Gesichts-Tracking wurden mit in die Realisierung der Hautsegmentierung einbezogen und noch verbliebene Probleme durch eigene Ansätze reduziert. Die so entstandenen segmentierten Bilder wurden binarisiert und durch morphologische Operatoren optimiert. Eine Bestimmung des Schwerpunktes und der Schwerpunktdynamik ermöglichten eine hinreichend genaue Vorhersage der zu erwartenden Position des Schwerpunktes im nächsten Kamerabild. Mit diesen Informationen konnten nun gezielt Fahrbefehle an den Roboter geschickt werden, um den frisch segmentierten Schwerpunkt in der Bildmitte zu halten. Um das Pendeln der Roboter-Plattform zu minimieren wurde als Regler ein PI-Regler gewählt. Zusätzlich zum Tracking selbst wurde eine geeignete Lösung gesucht und gefunden, um den Trackingvorgang ein- und auszuschalten. Ein Kippschalter wurde an der Front der Roboter-Plattform angebracht, mit welchem sich das Tracking bequem durch den Benutzer an- und abschalten lässt. Das somit entstandene System wurde Experimenten unterzogen, welche Aufschluss über die Einsatzfähigkeit des implementierten Trackings geben sollten. Anhand der Ergebnisse dieser Experimente ist zu sehen, dass das entwickelte System eine Folgefahrt unter geringfügigen Einschränkungen ermöglicht.

6.2 Ausblick

Mit dem entwickelten Trackingalgorithmus wird die Grundlage zu weiteren Entwicklungen gelegt. So ist es zum Beispiel vorstellbar, dass der Algorithmus so erweitert wird, dass eine Handerkennung möglich ist. Am Lösungsweg könnte ein am IPR entwickelter blob-detection-Algorithmus beteiligt sein, welcher den größten segmentierten Bereich als Hand annimmt. Auf diese Handerkennung kann dann schließlich eine Stufe aufsetzen: Die Gestenerkennung. Ziel bei der Gestenerkennung ist eine Erweiterung der multimodalen Steuerung eines Roboters. So soll der Roboter in der Lage sein, verschiedene Gesten zu erkennen und verschiedene Aktionsabläufe mit ihnen zu verbinden. So ist es zum Beispiel denkbar, dass der Benutzer sich vor den Roboter stellt und ihm durch eine eindeutige Handgeste[1] zu erkennen gibt, ihm zu folgen. Die Folgefahrt kann dann durch eine weitere geeignete Handgeste[2] wieder beendet werden. Aber auch komplexere Arbeitsschritte könnten auf diese Weise mit Hilfe einer schon fast intuitiven menschenfreundlichen Bedienung erledigt werden. Die Anwendungsgebiete hierbei sind zahlreich, denkt man zum Beispiel an Gepäckträgersysteme, welche durch Gesten veranlasst werden, das Gepäck des Kunden zu einem bestimmten Ort zu bringen, autonome Roboter in einer Lagerhalle, welche beladen werden und durch bestimmte Gesten zu einem bestimmten Regal fahren, an dem sie dann durch einen anderen Archivierungsroboter entladen werden oder sogar an Serviceroboter, welche in Wohnungen von bedürftigen Menschen durch Gesten Arbeiten erledigen, welche diese nicht mehr selbst ausführen können. Die Wissenschaft hat auf diesen Gebieten noch großes Potential.

[1] zum Beispiel das Herbeiwinken

[2] zum Beispiel die zur Blickrichtung orthogonale, ausgestreckte flache Hand

Literaturverzeichnis

[Gra96] Graf, René: *MORTIMER — Projektbericht 2/1996.* Universität Karlsruhe – Projektbericht, Februar 1996. http://wwwipr.ira.uka.de/~graf/Robots/mortimer_projdescr.html.

[Hub98] Huber, Mathias: *Entwurf und Aufbau eines mobilen Roboters zur dreidimensionalen Datenerfassung.* Diplomarbeit, Universität Karlsruhe (TH), Juni 1998. S. 68–71.

[Jäh97] Jähne, Bernd: *Digitale Bildverarbeitung*, Kapitel 15 Formanalyse, Seiten 519–527. Springer Verlag, Berlin Heidelberg, 4 Auflage, 1997. ISBN 3-540-61379-X.

[Mes99] Messmer, Hans-Peter: *PC-Hardwarebuch*, Kapitel 29 Andere Peripheriechips und Komponenten, Seiten 737–742. Nummer32. Addison-Wesley, Longman, 1999. ISBN 3-8273-1461-5.

[MSF97] Mann, Heinz, Horst Schiffelgen und Rainer Frofriep: *Einführung in die Regelungstechnik*, Kapitel 6 Digitale Reglerrealisierung (DDC), Seiten 224–235. Hanser Lehrbuch. Carl Hanser Verlag, München, Wien, 7 Auflage, 1997. ISBN 3-446-17672-1.

[Ste98] Steinhaus, Peter: *Mobile Vision System (MobVis).* Universität Karlsruhe – Projektbeschreibung, September 1998. http://wwwipr.ira.uka.de/~steinhau/MobVis.html.

[YLW98] Yang, Jie, Weier Lu und Alex Waibel: *Skin-Color Modeling and Adaption.* In: *3rd Asian Conference on Computer Vision*, Seiten 687–694, Hong Kong, Jan 1998. Hong Kong University of Science and Technology.