UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

# Entwicklung eines Webcam-basierten Gestenerkennungssystems zur konfigurierbaren Steuerung von Applikationen

## Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Marc Lange

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Stefan Müller<br>(Institut für Computervisualistik, AG Computergraphik) |
| Zweitgutachter: | Dipl.-Inform. Matthias Biedermann<br>(Institut für Computervisualistik, AG Computergraphik) |

Koblenz, im Dezember 2005

## Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

| | Ja | Nein |
|---|---|---|
| Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. | ☐ | ☐ |
| Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. | ☐ | ☐ |

Ort, Datum Unterschrift

## *Rechtliches*

Microsoft, ActiveMovie, Windows XP, Windows, Media player, Movie Maker, DirectX, DirectShow, Direct3D, DirectDraw und Visual Studio .Net sind entweder eingetragene Markenzeichen oder Markenzeichen der Microsoft Corporation.
Mac Os X, Quicktime und Quartz Extreme sind entweder eingetragene Markenzeichen oder Markenzeichen von Apple.
Logitech, Sony, Playstation 2 und EyeToy sind entweder eingetragene Markenzeichen oder Markenzeichen der Sony Corporation.
Andere hier namentlich erwähnte Produkte und Gesellschaften sind eingetragene Markenzeichen oder Markenzeichen ihrer jeweiligen Besitzer.

# *Inhaltsverzeichnis*


**1 Einleitung ..... 7**

**1.1 Motivation ..... 7**

**1.2 Ziele ..... 7**

**1.3 Überblick ..... 9**

**2 Grundlagen ..... 10**

**2.1 Möglichkeiten der Eingabe und Steuerung ..... 10**

2.1.1 Bedienelemente ..... 11

2.1.2 Bewertung und Entscheidung ..... 15

**2.2 Application Programming Interface ..... 20**

2.2.1 Grafik-APIs ..... 20

2.2.1.1 Direct3D ..... 20

2.2.1.2 OpenGL ..... 21

2.2.1.3 Quartz ..... 22

2.2.2 Multimedia-APIs ..... 23

2.2.2.1 DirectShow ..... 23

2.2.2.2 SDL ..... 24

2.2.2.3 Quicktime ..... 25

2.2.2.4 GStreamer ..... 25

2.2.3 Begründung der Schnittstellen-Wahl ..... 26

**3 Image Processing ..... 28**

**3.1 Farbräume und Farbmodelle ..... 29**

3.1.1 RGB-Farbraum ..... 30

3.1.2 HSV-Farbraum ..... 31

3.1.3 YUV-Farbmodell ..... 32

**3.2 Filter ..... 33**

**3.3 Bildvorverarbeitung ..... 34**

3.3.1 Grauwert-Bild ..... 34

3.3.2 Glättung ..... 36

3.3.2.1 Lineare Filter ..... 36

3.3.2.2 Rangordnungs-Filter ..... 38

**3.4 Objekterkennung ..... 40**

3.4.1 Schwellwertverfahren ..... 40

3.4.2 Kantendetektion ..... 41

3.4.2.1 Sobel-Operator ..... 47

3.4.2.2 LaPlace-Operator ... 48
3.4.2.3 Kantenverdünnung ... 49
3.4.2.4 Kreis-Erkennung mit der Hough-Transformation ... 50
3.4.3 Regionen-Segmentierungsverfahren ... 52
3.4.4 „Color Indexing“ ... 53
**3.5 Bewegungserkennung ... 55**
3.5.1 Tracking ... 55
3.5.2 Bildverarbeitungsbasierte Methoden ... 55
**4 Systeme der Gestenerkennung ... 58**
**4.1 Definition: Gesten ... 58**
**4.2 Bestehende Systeme und Ansätze ... 59**
4.2.1 Linienfiguren zur Interaktion ... 60
4.2.2 Hidden-Markov-Modell ... 63
4.2.3 Handgesten ... 66
**5 Implementierung ... 68**
**5.1 Der Marker ... 68**
**5.2 Programmablauf ... 72**
**5.3 Prinzip der Gestenerkennung ... 79**
**5.4 Auszüge aus dem Quellcode ... 85**
5.4.1 BitmapArray2D ... 86
5.4.2 Lookup Table ... 87
5.4.3 Non-Maxima-Suppression ... 88
5.4.4 detectCircleCenter() ... 89
5.4.5 findMidpoint() ... 90
5.4.6 regToDir() ... 92
5.4.7 identifyGesture() ... 93
**6 Evaluierung ... 95**
**7 Fazit und Ausblick ... 98**
**8 Literaturverzeichnis ... 101**
**9 Abbildungsverzeichnis ... 105**
**10 Abkürzungsverzeichnis ... 108**

## *Danksagung*

An dieser Stelle möchte ich allen, die mir während der mehrmonatigen Entwicklung, Umsetzung und Evaluierung der vorliegenden Diplomarbeit ihre Unterstützung angeboten haben, ganz herzlich danken. Sie alle standen mir mit Rat und Tat bei Seite und ertrugen mit Geduld meine manchmal stressbedingten Launen. Zuerst danken möchte ich meiner Freundin Sandra und meinen Freunden Karsten, Martin und Daniel, die Korrektur gelesen haben.
Des Weiteren möchte ich meine Eltern, die mir das Studium im Fachbereich Informatik mit Schwerpunkt Computervisualistik ermöglicht haben, und meine Schwester nennen, die für mein leibliches Wohl gesorgt und sonstige Ansprüche gedeckt haben.
Ganz besonderer Dank gilt meinem Betreuer Matthias Biedermann, der zu jeder Tages- und Nachtzeit ein offenes Ohr für meine Fragen aber auch Anregungen hatte und aufgrund seiner Erfahrung hilfreiche Tipps geben konnte.

Weiterhin möchte ich allen danken, die ich vergessen habe zu erwähnen. Danke!

# 1 Einleitung

## 1.1 Motivation

Zu den primären Eingabegeräten eines herkömmlichen Heimrechners oder einer Arbeitsstation gehören unumstritten die Maus und die Tastatur. Sie ermöglichen fast uneingeschränktes Navigieren durch Software-Menüs und Manipulation von Daten, Bildern und Dokumenten.
Doch auch wenn sich der Gebrauch dieser Peripheriegeräte leicht erlernbar, benutzerfreundlich und effizient gestaltet, so sind sie dennoch zur Programmsteuerung in manchen Teilbereichen, beispielsweise in der Medizin, speziell in der Chirurgie, oder in durch Staub und Matsch verschmutzte Produktionsstätten, durchaus ungeeignet. Des Weiteren erschweren in der Hand getragene Gegenstände und Kleidungsstücke, z.B. Arbeitshandschuhe, die Handhabung und Bedienung von Maus und Tastatur oder führen nicht selten zu Fehleingaben. Sterile oder stark verunreinigte Arbeitsplätze erfordern deshalb alternative Eingabe- und Steuerungsgeräte, die möglichst kostengünstig erwerbbar, ebenso effizient wie Maus und Tastatur nutzbar und mindestens genauso flexibel sind. Die zuvor angeführten Beispiele sind jedoch nur exemplarisch und stellen nicht das einzige Einsatzfeld für eine alternative Art der Steuerung durch Befehle dar.

In der vorliegenden Diplomarbeit soll eine Idee für eine kostengünstige und stabile Art der Befehlseingabe und Programmsteuerung, unter Berücksichtigung der soeben entworfenen Szenen, vorgestellt und die Umsetzung dieser dokumentiert werden.
Ferner möchte ich noch meinen persönlichen Wunsch, unterschiedliche Methoden der Bildverarbeitung aufzugreifen, äußern. Die Umsetzung eines Gestenerkennungs-Systems auf Basis eines Live-Video-Streams sehe ich als Herausforderung, der ich mich gerne stellen und zu meiner vollsten Zufriedenheit erfüllen möchte.

## 1.2 Ziele

Gesamtziel ist die Umsetzung einer Applikation zur innovativen Steuerung in Tätigkeitsfeldern, in denen gewöhnliche Eingabegeräte unzureichend, ungeeignet oder nicht effizient erscheinen. Von besonderem Reiz ist folglich die Vorstellung, dem Anwender, statt einer Ergänzung, eine gelungene Alternative zu üblichen Eingabegeräten zu bieten.
Im Rahmen dieser Diplomarbeit soll dementsprechend ein vom Benutzer frei konfigurierbares System zur Gestenerkennung entworfen werden. Sowohl die Bedienung des Programms als auch das Zeichnen der Gesten sollte intuitiv erfolgen und leicht einprägsam sein. Hierbei ist für den Anwender das Festlegen und Neuerstellen von

Gesten in einer einfach strukturierten und übersichtlichen externen Datei ohne viel Vorwissen und mit geringem Aufwand vorgesehen.

Die Applikation soll die Funktion eines Serverdienstes übernehmen, damit sie in absehbarer Zeit mit jeglicher externen Anwendung verknüpft werden kann. Als Client-Demonstrationsprogramm wird eine bescheidene Konsolenanwendung in Form einer einfachen „Chat-Anwendung“ umgesetzt und angebunden, um die Funktionsweise aufzuzeigen und die Evaluierung zu erleichtern. Der Schwerpunkt soll jedoch nicht auf der Client-/Server-Funktion, sondern weitestgehend auf der Entwicklung einer stabilen und konsistenten Gestenerkennung liegen. Das Verfahren zur Erkennung der Gesten wird genauer in den später folgenden Kapiteln erläutert.

Jedoch soll bei all den visionären Ideen nicht die Rechenleistung und der Stand der Technik aus den Augen verloren werden. Zeitaufwändige Berechnungsmethoden, sowie erhebliche Anforderungen an den verwendeten Arbeitscomputer sind vordergründig zu vermeiden. Demzufolge ist der Ablauf der Gestenerkennung in Echtzeit, wenn möglich auch auf älteren Systemen, von besonderer Wichtigkeit, um ein einwandfreies und fehlerfreies Erkennen jederzeit zu gewährleisten, ohne den Anwender zur Neuanschaffung eines High-End-Rechners zu drängen.
Diese Anforderungen decken sich mit dem Anspruch ein kostengünstiges und somit attraktives System zu entwickeln.

Der Verlauf der gesamten Arbeit wurde durch einzelne Teilaufgaben bestimmt. Diese Aufgaben waren im Einzelnen:

1. die Recherche bestehender Systeme zur Gestenerkennung
2. die Recherche geeigneter Methoden zur Umsetzung des entworfenen Konzeptes, sowohl bei der Art der Gestenerkennung als auch bei den benötigten Bildverarbeitungsschritten
3. ein Entwurf auf Software-Ebene als Grundvoraussetzung
4. Konzeptentwicklung zur Bereitstellung geeigneter Steuerbefehle sowie die einfache Erweiterung dieser Befehle
5. Implementierung des Gestenerkennungssystems
6. Evaluierung der Applikation
7. schriftliches Ausarbeiten der Grundlagen und des Konzeptes des Gestenerkennungssystems
8. Zusammentragen und schriftliches Darlegen der Testergebnisse

## 1.3 Überblick

Vorweg soll ein Überblick über den Aufbau der Arbeit dem Leser das Zurechtfinden erleichtern. Nachdem in *Kapitel 1 Einleitung* die Aufgabenstellung vorgestellt und die Ziele abgesteckt wurden, folgen im *Kapitel 2 Grundlagen* Erörterungen wissenswerter Informationen zum Verständnis der gesamten Ausarbeitung und der Implementierung. Das Kapitel lässt sich in folgende Untergruppen aufteilen. Zuerst werden in *Kapitel 2.1.1* handelsübliche und für jedermann zugängliche Eingabegeräte vorgestellt und auf deren Relevanz und Nutzen für die Diplomarbeit analysiert. Anschließend erfolgen in *Kapitel 2.2* Untersuchungen vorhandener Programm-Schnittstellen und eine Bewertung, inwiefern sich diese zur Umsetzung des geplanten Projektes eignen.

Dem folgend schließt sich *Kapitel 3* mit Schwerpunkt Bildverarbeitung an. In diesem werden Begriffe und Methoden der Vor- und Weiterverarbeitung in der Bildanalyse dargeboten.

Nachfolgendes *Kapitel 4.1* versucht eine befriedigende Antwort einer Definition von Gesten zu liefern. Zudem werden in *Kapitel 4.2* Ideen und Möglichkeiten der Eingabe durch Gesten und beispielhaft bereits bestehende Gestenerkennungssysteme vorgestellt, deren Funktionsweisen umrissen und bevorzugte Einsatzgebiete, wenn vorhanden, erläutert.
Die komplette Implementierung sowie die entworfene Gestenerkennung werden ausführlich in *Kapitel 5* präsentiert. Es wird auf wichtige Details der Umsetzung eingegangen und Überlegungen, die sich im Verlauf der Arbeit entwickelten, werden formuliert. Ferner werden signifikante Methoden zur Umsetzung der gestellten Aufgaben aus dem Quellcode entnommen und erklärt.

Gesammelte Erfahrungen während der Implementierung und Ergebnisse der Testphase, die die ausführliche Prüfung der Anwendung zum Zeitpunkt der Abgabe der Diplomarbeit beinhaltet, werden in *Kapitel 6 Evaluierung* wiedergegeben und in einem Fazit (*Kapitel 7*) noch einmal kurz zusammengefasst. Der ebenfalls in *Kapitel 7* enthaltene Ausblick soll nicht mehr umgesetzte Ideen und Verbesserungsvorschläge aufzeigen und zu einer eventuellen Weiterentwicklung anregen.

# 2 Grundlagen

Dieses Kapitel beschäftigt sich mit dem Hardware- und Softwareaspekt der Diplomarbeit. Zuerst erfolgt eine Vorstellung verschiedener momentan erhältlicher Standard-Eingabegeräte, gefolgt von deren Einteilung in Kategorien der Bedienungsart sowie der Entscheidungsbegründung für das gewählte Gerät und der damit verbundenen Methode zur Programmsteuerung. Maßgeblich für die Wahl des Gerätes sind die Erfüllungen bestimmter Kriterien, wie die Handlichkeit des Bedienelementes, die Einschränkungen des Benutzers durch das Steuergerät, aber auch der Kostenfaktor.

Daran anschließend werden verschiedene in Frage kommende Programminterfaces zur Kommunikation mit dem ausgewählten Steuerelement aufgezeigt und deren Funktion näher erläutert. Vorweg gegriffen sei die Entscheidung für zwei APIs[1] zur geplanten Prototyp-Entwicklung der Software, eines zur Ansteuerung des Bedienelements und ein zweites zur Erstellung einer Informationsrückmeldung für den Anwender.

## 2.1 Möglichkeiten der Eingabe und Steuerung

Die Vorstellung der nachfolgenden Eingabe- und Steuergeräte ist nur ein Auszug und erhebt keinen Anspruch auf Vollständigkeit. Es sollen nur die gängigsten Eingabemittel, vor allem unter den Aspekten der günstigen Anschaffung und der flexiblen Belegung der Steuerbefehle, beleuchtet werden.
Des Weiteren werfen wir einen Blick auf die Tauglichkeit in den vorher angesprochenen Szenarien der sterilen oder stark verschmutzten Arbeitsplätze. Hierbei wird bewusst auf eine detaillierte Beschreibung der Technik und des Aufbaus verzichtet. Spezialeingabegeräte zur Bedienung einer Workstation, wie zum Beispiel haptische Geräte, oder zur Interaktion in einer 3D-Welt, wie etwa Datenhandschuhe und Tracking-Systeme (sowohl mit Markern als auch markerlose), werden hier schon aufgrund der Anschaffungskosten und des zusätzlich benötigten Equipments (HMD[2]s) nicht näher untersucht. Dies beruht auf der Überlegung, dass, wenn ein Betrieb oder eine Einrichtung gedenkt, mehrere Arbeitsplätze mit einer optionalen Steuerung auszustatten, die Besorgung dieser den finanziellen Rahmen sprengen oder räumlichen Gegebenheiten übersteigen würde. Abhängig von der Wahl der Steuergeräte ist der einzuplanende Stellplatz ein nicht zu unterschätzender Aspekt. So erfordern zum Beispiel begehbare Visualisierungsräume, beispielsweise Caves, bestehend aus Projektionsleinwänden mit Maßen ab 2 mal 2 Metern, dementsprechend große Räumlichkeiten zur Montage.

[1] s. Abkürzungsverzeichnis
[2] s. Abkürzungsverzeichnis

### 2.1.1 Bedienelemente

Maus:

Die Maus ist wohl das am meisten verbreitete Eingabegerät. In Kombination mit der Tastatur erlaubt sie auf einer grafischen Oberfläche, der so genannten GUI[3], unkomplizierten und schnellen Zugriff auf Menüs, Fenster und Dateien.

Mäuse gibt es mit und ohne Kabel. Letztere, so genannte Funkmäuse, übertragen im Gegensatz zu den kabelgebunden Verwandten zwar ihre Informationen über Infrarot langsamer, erhöhen aber die Bewegungsfreiheit des Users aufgrund ihrer Kabellosigkeit. Mit entsprechender Software können die einzelnen Tasten einer Maus mit Sonderfunktionen ausgestattet werden, um dem Nutzer komplexere Steuerungs-möglichkeiten zu bieten. Auch ist die Maus ein vergleichsweise kostengünstiges Steuerelement. Optische Mäuse sind zusätzlich verunreinigungssicherer als die älteren Mäuse mit Kugelmechanik.

Betrachten wir das Peripheriegerät Maus jedoch unter Berücksichtigung der gestellten Anforderung, dass es auch in medizinischen Bereichen oder in Werkshallen zum Einsatz kommen könnte, geht der Vorteil der leichten und effizienten Bedienung verloren. Ein Chirurg bräuchte beispielsweise immer eine freie Hand zum Bewegen der Maus. Zudem ist die Maus ortsgebunden, müsste also unmittelbar in der Nähe des operierenden Arztes positioniert sein und wären somit stark hinderlich. Dies betrifft auch die Ausführungen mit Funk.

Trackball:

Auf den ersten Blick ähnelt ein Trackball einer auf den Kopf gestellten optomechanischen Maus[4]. Der Cursor oder Mauszeiger wird durch Drehen an der Kugel, die in eine Kuhle eingefasst ist, an die gewünschte Position gebracht. Die seitlich angebrachten Buttons erfüllen die Funktion der linken bzw. rechten Maustaste.

Ein Vorteil des Trackballs ist die Unabhängigkeit von einer Unterlage, z.B. dem Mauspad, da der Ball durch Fingerbewegungen nur in seiner Kuhle rotiert wird, und nicht die gesamte Trackballeinheit bewegt werden muss.
Aufgrund der Ähnlichkeit zur normalen Computermaus ergeben sich nahezu die gleichen Mängel. Auch der Trackball ist ortsgebunden und muss in Armreichweite des Anwenders aufgestellt sein. Zudem hindern Ausführungen mit Kabel den Anwender in seiner Bewegungsfreiheit, wenn dieses Kabel im Weg hängt.

[3] s. Abkürzungsverzeichnis
[4] Kombination aus Optik und Mechanik; Maus mit Kugelmechanik

Tastatur:

Für eine umfangreiche Texteingabe ist die Tastatur unentbehrlich. Zwar könnte das Tastaturfeld gegebenenfalls durch ein Programm, was sich mit der Maus steuern ließe, ersetzt werden, würde allerdings an Effizienz und Geschwindigkeit bei der Textverarbeitung einbüßen. Es ist schwer denkbar, die Tastatur mit ihren vielfältigen Eingabeoptionen abzulösen oder gleichwertig zu ersetzen.

Dennoch treten auch bei der Tastatur ähnliche Probleme der Betätigungsreichweite wie bei der zuvor erwähnten Maus und dem Trackball auf. Das Bedienelement muss notwendigerweise in der Umgebung des Ausführenden installiert und ständig erreichbar sein. Ebenso könnte ein Werksarbeiter mit Arbeitshandschuhen fehlerhafte Eingaben durch ungewolltes Betätigen mehrerer Tasten ausführen oder müsste die Handschuhe bei jeder Nutzung der Tastatur ablegen und wieder anziehen.

Touchpad:

In erster Linie dient das Touchpad als Mausersatz an einem Notebook. Das Panel mit den mindestens zwei darunter angeordneten Tasten bildet die gesamte Steuereinheit und bietet ähnliche Funktionen wie die Maus. Einziger Unterschied ist das gleichzeitige Benutzen des Panels als optionalen primären Mausbutton.
Gesteuert wird der zugeordnete Cursor auf der Desktop-Oberfläche durch Auflegen und Bewegen eines Fingers. Horizontal und vertikal angeordnete Elektroden ermitteln die Position des Fingers anhand von Kapazitätsschwankungen und übertragen die Informationen. Dabei erfüllt der Finger ebenfalls die Funktion einer Elektrode und ist somit mitverantwortlich für die durch das Elektrodengitter gemessene Kapazität.
Entsprechend dicke Handschuhe oder verwendete Zeigehilfen könnten die Bedienung des Touchpad bis zur Nichtfunktionalität beeinflussen. Ansonsten können Vor- und Nachteile der Maus gleichgesetzt werden. Da jedoch das Touchpad in einem Notebook integriert ist, fällt die Kostenbetrachtung weg.

Joystick:

Das früher am häufigsten anzutreffende Steuerinstrument, vor allem im Bereich der Computerspiele und Spielekonsolen, war der Joystick. Mit der Zeit wurde er allerdings von den leistungsstärkeren Gamepads oder Joypads an den Konsolen und durch Maus und Tastatur am PC verdrängt.

Genutzt werden Joysticks heutzutage hauptsächlich noch bei Flugsimulationen, sowohl von Privatanwendern bei spielerischen Anwendungen, als auch von geschultem Personal bei der Steuerung von weiterentwickelten Trainings-Flugsimulatoren, sowie bei der Kontrolle von Kränen und Fernlenkwaffen. Die Präzision der Steuerung ist abhängig von der Bauweise, die analog oder digital sein kann, mehr oder weniger gegeben. Analoge Joysticks können im Gegensatz zu digitalen Joysticks anhand der Druckstärke in eine

Richtung noch eine Geschwindigkeit bzw. Intensität feststellen und reagieren dementsprechend feinfühliger.
Durch die Größe des Steuerknüppels sind denkbar weniger Fehleingaben zu erwarten. Das Neigen des Steuerknüppels in eine Richtung gestaltet sich durch dessen Ausmaße einfacher als zum Beispiel das Bedienen des Steuerkreuzes eines noch nachfolgend vorgestellten Gamepads. Im Gegensatz zur festgelegten Bedienung des Steuerkreuzes durch den Daumen, ist es beim Joystick möglich die Hand im Allgemeinen zu benutzen, was das Ablegen der Handschuhe teilweise erübrigen würde.
Zu den Nachteilen bleibt zu sagen, dass auch bei den Joysticks der Anwender das Gerät in der Hand halten muss, um dies überhaupt bedienen zu können.

Gamepad:

Das Gamepad ist zweifellos dem Computerspiele-Bereich zugeschrieben und löste den damals vorherrschenden Joystick ab. Bis auf wenige Ausnahmen, z.B. das Eyetoy[5] für die Sony Playstation 2, ist das Gamepad das primäre Eingabewerkzeug einer Konsole und für fast alle Konsolenspiele geeignet. Im PC-Bereich werden Spiele häufiger durch Maus und Tastatur als durch Gamepads bedient. Controller der einfachsten Bauweise bestehen aus einem Steuerkreuz auf der linken Seite und aus zwei bis vier Knöpfen, so genannten Buttons, auf der rechten Seite. Weit verbreiteter sind jedoch heutzutage die analogen Versionen der Gamepads die zusätzlich zu den zuvor beschriebenen Steuermöglichkeiten noch über zwei bis vier Buttons auf der Stirnseite und über zwei Analogstick verfügen. Angeboten werden Gamepads sowohl als Ausführung mit Kabel als auch als Infrarotversion. Das Gamepad ist als Steuerelement im Rahmen dieser Diplomarbeit eher ungeeignet, da es üblicherweise für eine einwandfreie Handhabung sogar zweier Hände bedarf und eigentlich für spielerische Anwendungen gedacht ist.

Pen tablet:

Pen tablets, oder zu Deutsch Grafiktabletts, sind Zeigergeräte bestehend aus einer tablettähnlichen Platte und einem Zeiger in Form eines stiftähnlichen Instruments. Der Cursor wird durch das Bewegen des Stiftes auf dem Tablett navigiert. Gleichzeitig wird der Druck des Stifts auf die Platte gemessen, um zusätzliche Informationen, wie etwa Deckkraft einer Farbe oder Druckstärke eines Pinsels für Zeichenprogramme, zu ermitteln. Technisch beschrieben erzeugt der vorwiegend mit Batterie gespeiste Stift ein elektromagnetisches Feld, welches durch das Sensorboard erkannt wird. Somit registriert das System die Position des Stiftes.
Für die Bilderstellung ist diese Methode der Interaktion deutlich präziser, da es stark dem herkömmlichen Malen eines Bildes gleicht. Die präzisere Art der Umsetzung „Bewegung in Linie“, verglichen mit der Maus, scheint in der Nutzung der jeweils unterschiedlichen Muskeln zu liegen. Während die Maus größtenteils ihre Bewegung aus dem Unterarm erfährt, wird der Stift des Tabletts durch die feinmotorischen Finger geführt. Auch

[5] USB-Kamera von Logitech mit manueller Schärfeeinstellung

Grafiktabletts sind in der Anschaffung preiswert, führen jedoch auch den Nachteil der Ortsgebundenheit, gleich der Maus oder Tastatur, mit sich. Ihr Haupteinsatzgebiet ist unter anderem die Design-Nische und die Digitale Bildverarbeitung. Ist für einen Kunden schnell mal ein Entwurf zu skizzieren und per Email zuzuschicken, ist die Methode des Zeichnens über ein Grafiktablett deutlich effizienter als eine auf Papier festgehaltene Skizze, die abschließend noch gescannt werden muss.

Touchscreen:

Touchscreens sind Bildschirme, bei deren Berührung bestimmte Programmabläufe gestartet werden. Grafische Oberflächen definieren den Bereich, der für die jeweilige Anwendung aktiviert bzw. berührt werden muss. Großer Beliebtheit erfreuen sich Touchscreens oder Tastschirme besonders im Infotainment-Bereich. Verwendung finden sie in Messeveranstaltungen, als Bankautomat oder als Infocenter an Bahnhöfen. Die Benutzerführung ist größtenteils selbsterklärend und bedarf keiner großen Einarbeitungszeit. Software kann individuell auf die Recheneinheit eines Touchscreens aufgespielt und so den Bedürfnissen des Auftraggebers und deren Kunden angepasst werden.
Zur Konstruktion von Touchscreens kann derzeit auf mehrere zeitgemäße Bauweisen zurückgriffen werden. Es existieren Konstruktionen nach dem Widerstands-, dem Kapazitäts- und dem Energieabsorptionsprinzip, sowie Umsetzungen durch optische Systeme. Sie alle haben das Erkennen der Koordinaten des Zeigegerätes (Stift, Finger) unter Verwendung unterschiedlicher Techniken gemein.
Wie allerdings auch bei allen anderen bisher vorgestellten Peripherie- und Eingabegeräten ist der Nutzer an den Standort des Gerätes gebunden und somit in seiner Bewegungsfreiheit eingeschränkt. An ein einfaches Einpacken und Mitnehmen sei schon gar nicht gedacht.

Mikrofon/Headset:

Das Mikrofon und das „bidirektionale Mikrofon“, das so genannte Headset, bestehen beide aus einem schallempfindlichen Sensor, der über eine Membran aufgenommenen Schall, im Allgemeinen Laute und Sprache, an den Computer zur Informationsauswertung weiterleitet. Im Gegensatz zum einfachen Mikrofon beinhaltet ein Headset zusätzlich noch einen Kopfhörer über das der Träger Töne empfangen kann. Beide Varianten sind aufgrund ihrer heutigen geringen Größe und ihres Gewichts, insbesondere im Vergleich zu früher, sehr handlich. Sind die Kopfhörer oder Mikrofone obendrein Wireless[6]-Geräte, z.B. wenn die Verbindung über Bluetooth hergestellt wird, ist der User nahezu uneingeschränkt in seiner Handlungsfreiheit. Lediglich die Funkreichweite des jeweiligen Equipments, die abhängig von der Sendeeinheit zwischen 10 - 100 Meter beträgt, legt den Benutzungsradius fest. Verwendung finden Headsets und Mikrofone vorrangig in der Kommunikation in Computerspielen und der Internet-

[6] kabellos

Telefonie (VoIP[7]), sowie in Sprachlernprogrammen. Ein nicht ganz an den Computer angelehntes Einsatzfeld ist der Gebrauch mit Handys.
Die Variante mit Kabel ist im Hinblick auf den Nutzen in dieser Diplomarbeit im Vergleich zu den kabellosen Versionen weniger interessant, da das Kabel wiederum stark einengend und hinderlich ist. Drahtlose Ausführungen könnten jedoch in Anbetracht der Anforderungen an das alternative Bedienelement in die engere Wahl innovativer Lösungen fallen.

Webcam:

Obwohl eine Webcam primär keine Möglichkeit zur Steuerung des Personal Computers aufweist, ist ihre Betrachtung unter Einbezug der Ansprüche dieser Diplomarbeit nicht uninteressant. Ihr favorisiertes Einsatzgebiet ist momentan, aufgrund ihrer, im Vergleich zu normalen und digitalen Camcordern, niedrigen Auflösungen von durchschnittlich 640x480 Pixeln[8], die Kommunikation über das Internet, z.B. in Chats und Messenger-Anwendungen oder bei der Internet-Videotelefonie. Dennoch könnte die Kamera zusammen mit einer angepassten Anwendung, die diverse Bildverarbeitungselemente vereinigt, in einen erweiterten Bewegungserkenner umfunktioniert werden. Angeschlossen an eine Arbeitsstation über den USB[9]-Port oder den zur Zeit weniger üblichen Firewire-Anschluss wäre somit innerhalb des Blickwinkels der Webcam eine Steuerung durch Bewegungen oder Gesten möglich. Für diesen Zweck ist die derzeitige Auflösung einer aktuellen USB-Kamera völlig ausreichend.

### 2.1.2 Bewertung und Entscheidung

Obwohl der Titel dieser Diplomarbeit die Wahl des Eingabegerätes schon ankündigt, soll die Entscheidung an dieser Stelle hinreichend begründet werden. Eine einem Entwicklungs-Prozess ähnelnde Erörterung der unterschiedlichen Bedienungsarten soll die Vor- und Nachteile unter Betrachtung der gestellten Anforderungen aufzeigen.

Tastaturen und tastaturverwandte Geräte sind in ihrer Verwendung äußerst vielseitig. Der Weg der schnellstmöglichen Texteingabe und das angepasste Belegen von Tasten unter Verwendung adäquater Software verleihen ihr ein höchstes Maß an Flexibilität. Auf diese Weise kann beispielsweise eine Tastatur individuell auf den Arbeitsplatz abgestimmt und nach den Wünschen des Benutzers konfiguriert werden, um einen komfortablen Zugriff auf alle Funktionen zu haben. Dennoch besitzen sie unter anderem den Nachteil einer hohen Anfälligkeit für Fehleingaben, gerade bei der Bedienung mit Arbeitshandschuhen, sofern es sich bei einer verwendeten Tastatur nicht um eine speziell angepasste Version mit übergroßen Tasten handelt, bei denen Fehleingaben fast unmöglich erscheinen.

[7] s. Abkürzungsverzeichnis
[8] Zusammensetzung aus picture und element
[9] s. Abkürzungsverzeichnis

Ebenso nachteilig wirkt sich die geringfügige Reichweite kabelgebundener Varianten aus. Für einen reibungslosen Bedienungsablauf ist die Platzierung nahe dem User unabdinglich, was sich störend in dessen Tätigkeitsfeld auswirken kann und somit den erweiterten Komfort einer Funktastatur zunichte macht. Die Schwierigkeit zur einwandfreien Bedienung ohne das Ablegen von Arbeitsutensilien oder vorangehend erwähnter Handkleidung, sowie der beschränkte Bedienungsradius erleichtern hier schnell das Ausschließen bei der Wahl einer geeigneten Bedienungsvariante im Zusammenhang mit den Anforderungen der vorliegenden Diplomarbeit.

Diese Aussagen lassen sich auch auf die vorangehend erläuterten Eingabegeräten, die in erster Linie das Führen in einem zweidimensionalen Koordinatensystem in das der Benutzeroberfläche übersetzen, wie zum Beispiel Maus, Grafiktablett und Touchscreens, übertragen. Das Bewegen in einer Ebene steuert den damit verbundenen Zeiger, Mauszeiger oder Cursor entsprechend am Monitor. Dabei erfordert die einhändige Führung der Geräte mindestens eine freie Hand der aktiven Person. Verwendete Gegenstände mindern in diesem Fall ebenso das Arbeitstempo und vor allem die Benutzungsergonomie außerordentlich.

Ein wesentlicher Aspekt aus der Sicht der Mediziner, speziell der Chirurgen, ist zudem die Anforderung der Sterilität der Instrumente und somit auch der verwendeter Geräte. Die Anwendung eines greifbaren Steuerelements erfordert somit dessen akribische Reinigung vor einer Operation. Des Weiteren sollten in solch sensiblen Tätigkeitsbereichen keine Geräte die Arbeit, durch unüberschaubare Verkabelung oder deren zwischenzeitliches Herumliegen, erschweren.
Aus diesen Gründen scheiden diese Geräte ebenfalls bei dem Entscheidungsprozeß für eine alternative Benutzereingabemöglichkeit im Hinblick dieser Diplomarbeit aus.

Da die bis zu dieser Stelle untersuchten Peripherieeinheiten durch ihre geringe bis mittelmäßige Bedienungsreichweite und hinderliche Positionierung in der Nähe des Benutzers für die Umsetzung innerhalb der gestellten Thematik ungeeignet erscheinen, gilt es nun die weniger durch die genannten Nachteile eingeschränkten Bedienelemente auf anderweitige, negative Eigenschaften zu prüfen.

Interessant in diesem Zusammenhang wäre das Betrachten von Systemen, die auf das Erkennen von Sprache oder Geräuschen spezialisiert sind. Töne in Form von Sprache, sowohl kurze Kommandos als auch längere Sätze, und Lauten bieten eine interessante Alternative zu gewöhnlichen Eingabemethoden. Von den kabelgebundenen Ausfertigungen der Mikrofone und Headsets abgesehen, bieten die drahtlosen Ausführungen schon einen beachtlichen Wirkungsradius, wodurch eine implementierte Sprachsteuerung, unter Verwendung existierender Spracherkennungssysteme, enorme Flexibilität bieten würde. Unter Spracherkennung sei an dieser Stelle das Umwandeln von gesprochenen Wörtern in vom Computer verständliche Zeichenfolgen verstanden.
Ein am Kopf angelegtes oder im Arbeitshelm integriertes Headset, behindert den Anwender beim Gebrauch, wenn überhaupt, nur gering, und ermöglicht uneingeschränktes Arbeiten und gleichzeitiges Bedienen des Computers und das Abrufen von Daten. Überlegungen, die allerdings im Vorfeld getätigt werden sollten, sind die

Fragen nach Zweck und Komplexität. Möchte man ein, im Bezug auf Ergänzungen, äußerst flexibles System, dass sich kontinuierlich ausbauen lässt, umsetzen, so wird man sich in erster Linie für ein sprecherabhängiges System entscheiden. Von Vorteil ist die bestehende Möglichkeit der Erweiterung des Wortschatzes und dem damit verbundenem Aneignen neuer Wörter. Nichtsdestotrotz birgt diese Art einige Nachteile. So besteht die Beschränkung auf einen einzelnen Benutzer, der durch einleitende Trainingsphasen versucht, dem System seine Sprechweise, welche möglicherweise beeinflusst wird durch Dialekte, Akzente oder besondere Betonungen, anzutrainieren. Aufgrund dessen ist in diesem Fall ein häufiger Wechsel zwischen mehreren Nutzern nicht möglich.

Alternativ kann man die sprecherunabhängige Variante anwenden, um die Software einer größeren Gruppe verfügbar zu machen. Trainingseinheiten zum einwandfreien Umgang mit der Software entfallen. Allerdings ist der Umfang des Wortschatzes bedeutend geringer als bei der sprecherabhängigen Methode. Unter Betrachtung der beschriebenen Szenarien wäre diese Art der Bedienmöglichkeit allerdings vollkommen ausreichend und der sprecherabhängigen vorzuziehen. Dies begründet sich mit der Tatsache, dass Kommandos und Befehle nur einen geringen Teil des verfügbaren Gesamtwortschatzes, den es zum Beispiel bei der Verfassung eines Schreibens zu beachten gilt, ausmachen.
Wohl am häufigsten verbreitete Kombinationen von Headsets und Spracherkennungssystemen sind die von verschiedenen Herstellern angebotenen Diktier-Software-Pakete. Nennenswerte Vertreter in diesem Zusammenhang sind IBMs ViaVoice, zum aktuellen Zeitpunkt die 10. Edition, und Dragon Naturally Speaking von Nuance, auf den jeweiligen Websites in verschiedenen Ausführungen käuflich erwerbbar.

So vorteilhaft diese Art der verbalen Bedienung sich auch in mancherlei Hinsicht gestaltet, umso schwieriger strukturiert sich die Umsetzung. Voraussetzung für eine reibungslose Nutzung der Sprachsteuerung ist die fehlerfreie Eingabe der Kommandos, was sich gerade in einer Umgebung mit höherem Geräuschpegel als äußerst diffizil gestaltet. Vorstellbar wäre die Implementierung diverser Filter, die die Eingaben des Benutzers aus den vorherrschenden Klängen und Nebengeräuschen heraustrennen. Dialoge mit Kollegen könnten sich dennoch als schwierig erweisen, sofern kein Schlüsselwort festgelegt wird, welches dem Computer dessen Verwender eindeutig identifizieren und das Initiieren eines Kommandos durch diesen erkennen lässt. Obendrein ist bei der Erstellung eines festen Wortschatzes stets zu beachten, dass ein und dasselbe Wort bei verschiedenen Betonungen jeweils einen Platz im Wörterbuch belegt. Zusätzlich ist die Größe noch abhängig von der vereinbarten Grundsprache, da durch die Beugung eines Wortes im grammatikalischen Kontext im deutschen 10-mal mehr Wortformen entstehen können. Generell gilt noch die Berücksichtigung von Redepausen zwischen einzelnen Wörtern zu beachten.
Ungeachtet des entwickelten Systems, sprecherabhängig oder –unabhängig, muss sich für eine diskrete oder kontinuierliche Einzelworterkennung entschieden werden. Vorteile diskreter Systeme ist eine zusätzliche Hilfestellung zur Abgrenzung und Erkennung einzelner Wörter. Die Pausen trennen Wörter eindeutiger und unterstützen somit die Verarbeitung der Signale. Allerdings ist dem Menschen die kontinuierliche Sprechweise angenehmer.

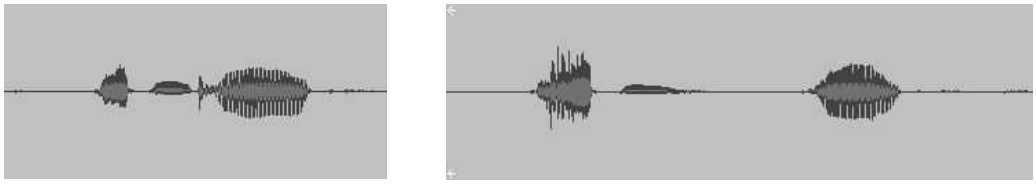


**Abb. 2.1: kontinuierliches (li.) und diskretes (re.) Signal anhand der Wortfolge „Guten Tag“**

Zur Entwicklung einer fehlerfreien Worterkennung wurden schon erste Schritte durch Integration von Mimik in Form des Lippenlesens getätigt. Sprache gekoppelt mit der Stellung der Lippen sollte ein eindeutigeres Identifizieren der gesprochenen Wörter erreichen.

Um solch ergänzende Mimik und Gestik dem Rechner zugänglich zu machen, bedarf es eines optischen Aufnahmegerätes. Hierfür eignet sich unter anderem das zuletzt vorgestellte Bedienelement, die Webcam. Sie präsentiert das vielleicht abstrakteste oder ungewöhnlichste Instrument, das als mögliche Alternative für gebräuchliche Eingabegeräte vorgestellt wurde. Mimik und Gestik bilden hierbei Grundbausteine, um eine herkömmliche Webcam in ein funktionstüchtiges Steuerelement umzuwandeln. Vorteile von Mimik und Gestik liegen zweifelsfrei in der universellen Verständlichkeit und der daraus resultierenden internationalen Unabhängigkeit

In diesem Zusammenhang möchte ich eine sehr interessante Diplomarbeit der Arbeitsgruppe Aktives Sehen an der Universität Koblenz erwähnen. Diplom-Informatiker Fabian Fritzer entwarf Ende des Jahres 2005 ein System zur Texteingabe am Computer durch Augenbewegung. Über eine spezielle aber dennoch gebräuchliche Webcam mit einer niedrigen Lichtempfindlichkeit, können auf vier, am Bildschirm dargestellte, Tasten ausgerichtete Augenbewegungen erkannt werden. Diese vier Tasten dienen der Worteingabe. Das Projekt ist primär eine Softwarelösung für körperlich Behinderte, denen eine standardmäßige Eingabe anhand der Tastatur nicht möglich ist. Für nähere Informationen wird auf die Diplomarbeit „Texteingabe per Augenbewegung“ [Fri05] verwiesen.

Auch wenn es denkbar ist, mit speziellen Anwendungen und Kameras zahlreiche Ideen einer alternativen Bedienung durch Mimik umzusetzen, bieten Gesten unter Umständen einen besseren Ansatz. Je nach Einsatzgebiet, in den angeführten Beispielen sterile oder sehr verdreckte Arbeitsplätze, sind Sicherheitsmaßnahmen gefordert, vor allem im Gesichtsbereich, die ein Erkennen der Mimik stark erschweren. Dazu gehören mitunter Schutzbrillen, Mundschutz und ähnliches. Obendrein wirkt die Vorstellung, dass ein Arzt versucht durch Mimik mit abwechselnd herunterhängenden Mundwinkeln die aktuellen Werte seines auf dem Operationstisch liegenden Patienten abzufragen, eher belustigend. Eindeutig verknüpfte Gesten kombiniert mit einem simplen Marker, zum Beispiel in Form eines farblich herausragenden Merkmals, auf dem Handrücken aufgesetzt oder im Arbeitshandschuh integriert, sollten dem Benutzer die nötige Flexibilität zur Erledigung seiner Aufgaben sowie der parallelen Nutzung des Computers bereitstellen.
Natürlich lässt sich das Verfahren der Steuerung durch Gesten auch in Verbindung mit anderen vorgestellten Eingabegeräten setzen. Denkbar wäre eine Anwendung, die

gezeichnete Gesten auf dem Touchpad erkennt, die selbstverständlich ohne weiteres dann auch auf Mäuse oder Joysticks erweitert werden könnte. Weiterhin vorstellbar ist eine Variante für das Grafiktablett mit gleichbedeutendem Funktionsumfang.

Meines Erachtens bildet die Umsetzung eines Steuergerätes mit Verwendung einer handelsüblichen, und auch günstigen, Webcam in Kombination mit einem Marker zur Unterstützung einer einwandfreien Erkennung vollführter Gesten eine effiziente als auch interessante Variante. Entsprechend umgesetzt, kann die Webcam, vereinigt mit einem Gestenerkennungs-System, ein beachtliches Potenzial zur Interaktion mit dem Computer bieten. Des Weiteren möchte ich einen Schritt in eine innovativere Richtung der Bedienung eines Computers tätigen, die durch Maus und Tastatur eingenommen scheint. Aus diesen genannten Gründen der Flexibilität und Uneingeschränktheit im Bewegungsfreiraum habe ich mich für die Entwicklung eines Gestenerkennungs-Systems unter Einsatz einer Webcam entschieden. Die Wahl des Markers fiel auf einen roten Ball mit diffuser Oberfläche, da er weitere Vorteile, die später näher erarbeitet werden, in sich vereinigt.

## 2.2 Application Programming Interface

Um das gesamte Projekt anforderungsgerecht umzusetzen, wird in jedem Fall mindestens eine Schnittstelle (API) benötigt, die das Kommunizieren mit der Webcam und das extrahieren einzelner Standbilder aus einem Live-Videostream unterstützt und außerdem leicht integrierbar ist. Zudem ist eine Feedback-Anzeige für den User geplant, infolgedessen gegebenenfalls ein zweites Interface zum Tragen kommt. Aus diesem Anlass werden an dieser Stelle unterschiedliche APIs vorgestellt, erläutert und abschließend die Wahl näher begründet.

### 2.2.1 Grafik-APIs

Die nachfolgend vorgestellten Grafik-Schnittstellen besitzen die primäre Aufgabe, dem Entwickler das Verwenden und Einbinden zwei- und dreidimensionaler Bildszenen und Grafiken verschiedenster Eingabe- und Grafikformate zu erleichtern. Dabei wird ihm ein vergleichsweise einfacher Zugriff auf entsprechende Hardwarebefehle ermöglicht.
Grafiken können in diesem Zusammenhang beispielsweise die Aufgabe der Verdeutlichung eines Zusammenhangs oder auch eine Feedback-Funktion in einem Programmablauf erfüllen. Abhängig vom Betriebssystem, ist den Entwicklern der Gebrauch von Grafik-APIs von verschiedenster Art und von unterschiedlichster Qualität möglich.
Der Grad zwischen Grafik und Multimedia ist nicht immer klar getrennt und so fließen die Bereiche nahtlos ineinander über. Ansatzweise kann man das Gebiet der Grafik als Untergruppe der Multimedia-Einheit einordnen.

#### 2.2.1.1 Direct3D

Komponenten zur Erstellung von 2D- und 3D-Grafiken unter dem Betriebssystem Windows sind in Microsofts Direct3D zusammengefasst. Das früher selbständige Projekt DirectDraw, zuständig für die zweidimensionale Bilderstellung wurde im Laufe der Zeit mit Direct3D vermischt und in DirectX integriert. Eine Weiterentwicklung von DirectDraw als eigenständiges Objekt wurde ab Version 8 daher nicht mehr fortgeführt. Zudem existiert noch die auf Direct3D basierende Schnittstelle Direct3DX, die das Entwickeln fortgeschrittener 3D-Anwendungen unterstützt. Für die dem Projekt relevanten Ansprüche würden allerdings die Basisfunktionen von DirectX für zweidimensionale Darstellung in Form der API Direct3D völlig ausreichen.
Direct3D ermöglicht dem Programmier, meistens auf dem Gebiet der Spieletechnik, einen direkten Zugriff auf die Grafik-Hardware. Dabei umgeht Direct3D die Windows eigene Ausgabeschnittstelle Graphic Device Interface (GDI), die aufgrund der Funktionsweise recht langsam arbeitet. Grundoperationen und Zeichenfunktionen müssen

von GDI nachgebildet werden, was sich in einem Geschwindigkeitsverlust bemerkbar macht. Direct3D ist in dieser Hinsicht wesentlich effizienter. Bei nicht unterstützten oder falschen Aufrufen einer Funktion liefert Direct3D eine Fehlermeldung. Dieser Fehler ist dann von der implementierten Anwendung durch eine geeignete Funktion, zum Beispiel durch eine alternative Berechnung, abzufangen.

#### 2.2.1.2 OpenGL

OpenGL, dessen Vorgänger das von Silicon Graphics entwickelte GL[10] darstellt, ist heutzutage größtenteils Bestandteil der Treiber ausgelieferter Grafikkarten und ist definiert als Interface für Grafikhardware. Die aktuellste Spezifikation ist derzeit in Version 2.0 erhältlich. OpenGL hat den großen Vorteil plattformunabhängig zu sein, wodurch es sich enormer Beliebtheit erfreut. Ihre Umsetzung durch eine Programmiersprache ist mehr prozedurenorientiert als beschreibend ausgelegt. Jedoch ist OpenGL eine reine Grafikbibliothek, die keine Möglichkeit für das Zugreifen auf Multimediageräte der Kategorie Audio und Video aufweist. Aus diesem Grund wird sie nicht in die Betrachtung zur Wahl einer geeigneten API für die geplante Anwendung im Bereich Videoerstellung und Bildanalyse mit der Webcam als Ausgangsmedium einbezogen.

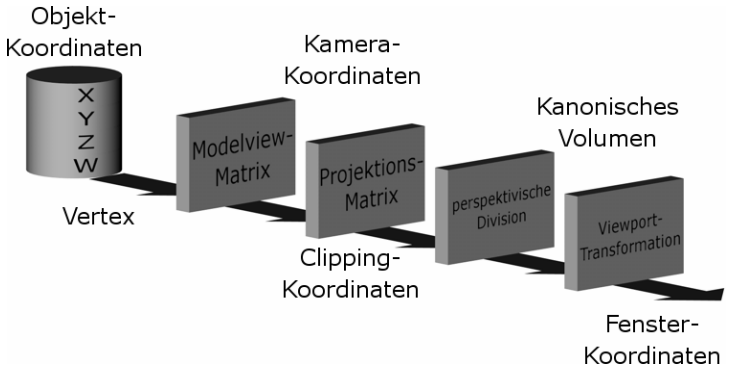



**Abb. 2.2: OpenGL Rendering Pipeline**

Anhand dieser API ist es dagegen möglich komplexe und interaktive 2D- und 3D-Szenarien und Modelle mit den verschiedensten Effekten und Visualisierungsoptionen, zum Beispiel Partikelsysteme und Texture-Mapping, in hoher visueller Qualität zu erstellen. Dies ist auch der primäre Anwendungsbereich von OpenGL. Beschriebene Strukturen und Körper werden intern als Sequenzen von Vertices für geometrische Objekte oder als Pixel für 2D-Bilder im Framebuffer abgelegt. Die in mehreren Stufen ablaufende Umwandlung der Eingaben geometrischer Definitionen in eine fertig gerenderte Szene ist vollends Aufgabe der OpenGL-API.
Fundamental hat OpenGL die Eigenschaften eines Zustandsautomaten. Hintergrund hierfür ist die somit vereinfachte Strukturierung des gesamten Codes. Einmal aktivierte

[10] s. Abkürzungsverzeichnis

Zustände bleiben bis zu ihrer Deaktivierung bestehen und erlauben somit die mehrfache Festlegung von Eigenschaften für eine bestimmte Gruppe, z.B. die Farbe mehrerer Vertices. Vorteilig wirkt sich dies auf die Rechenleistung aus, da die Leistungsbeanspruchende Neuerstellung der Rendering-Pipeline (s. Abb. 2.2) weitestgehend vermieden wird.
Erwähnung finden an dieser Stelle die Grafik-APIs und damit OpenGL aufgrund des Vorhabens, ein simples Feedback für den Nutzer einzurichten, indem gezeichnete Gesten in einem zweiten Fenster durch linienförmige Strukturen dargestellt werden. Hiermit soll ein ergonomischer Aspekt im Hinblick auf ständige Informierung des Benutzers über seine Handlungen berücksichtigt werden.

#### 2.2.1.3 Quartz

Das Pendant zu Windows DirectX im 2D- und 3D-Grafik-Bereich für das Apple-Betriebssystem Mac OS X ist die Grafik-Schnittstelle Quartz Extreme. Diese vereint die vier, von Apple als leistungsstark beschriebene, Komponenten:

- OpenGL,
- Quartz 2D,
- Quicktime und
- Quartz Compositor.

Die Kombination dieser Komponenten gestattet zum einen das Auslagern der Desktop-Oberfläche als 3D-Szene vom Prozessor in den Grafikkarten-Chip, was sich unweigerlich in einer Entlastung des Prozessors und somit einer enormen Leistungssteigerung äußert. Rechenintensive Berechnungen von Effekten, wie Schattenwurf und Transparenz, werden von der Hauptrecheneinheit getrennt und in der Grafikeinheit vollführt. Weiterhin ist die hochwertige Darstellung von Texten und 2D-Grafiken aufgrund der Basis des plattformunabhängigen PDF[11]-Formats als integriertes Element der Quartz 2D-Komponente ohne weiteres gewährt.
Als einzige nur auf das Mac OS–Betriebssystem beschränkte API fällt diese gänzlich aus der Betrachtung auf Tauglichkeit für das angestrebte Vorhaben heraus. Grund hierfür ist die Wahl des Betriebssystem Windows als Entwicklungsplattform. Ungeachtet des Betriebssystems müsste zusätzlich zu Quartz Extreme noch eine Multimedia-API verwendet werden, da es sich bei Quartz Extreme ausschließlich um eine 2D- und 3D-Render-Bibliothek handelt und keine Unterstützung für Audio- oder Videogeräte bietet.

[11] s. Abkürzungsverzeichnis

### 2.2.2 Multimedia-APIs

Multimedia-APIs ermöglichen das Implementieren spielerischer, visueller und auditiver Applikationen. Häufig werden vor allem Kombinationen dieser Bereiche entwickelt, daher auch der Begriff Multimedia („viele Medien“). Solch kombinierte Anwendungen erlauben dem Benutzer verschiedene Arten der Bedienung und Navigation durch virtuelle erstellte Szenen oder das Bearbeiten und Ansehen digitaler Daten. Um dem Programmentwickler einen schnellen Zugriff auf Hardware des multimedialen Bereiches zu ermöglichen, bedarf es effektiver und leicht manipulierbarer Schnittstellen.
Es gibt Schnittstellen als Lösungen für alle Plattformen, aber auch plattformspezifische APIs. Nachfolgend werden einige wichtige Schnittstellen präsentiert und erläutert.

#### 2.2.2.1 DirectShow

DirectShow, früher ActiveMovie, war ehemals Teil des DirectX-SDK[12]-Pakets, ist aber nun im Platform-SDK[13] von Microsoft integriert. Grundsätzlich ist DirectShow eine Programmierschnittstelle zur Erstellung, Umwandlung und zum Abspielen (engl.: capturing, conversion and playback) von Audio- und Videostreams. Dabei basiert diese Schnittstelle auf dem Component Object Model (COM), einer von Microsoft entworfenen Technologie zur Verwendung bereits implementierter Codefragmente in Dynamic Link Libraries (DLL). Mit DirectShow wird dem Programmierer der einfache Zugriff auf Multimedia-Hardware des PCs, unter anderem Webcams, TV-Karten, digitale Camcorder und diverse Audiogeräte, ermöglicht. Dies wird unter anderem durch das Zugreifen auf interne Funktionen der DirectX-Komponenten, wie zum Beispiel DirectSound und Direct3D, realisiert. Des Weiteren ist es fortgeschritteneren Programmierern möglich, eigene Filter zu kreieren und einzubinden.
Entworfen werden können Funktionsabläufe und Aufgaben eines mit DirectShow erweiterten Programms mit Hilfe der im Umfang des Software-Paketes enthalten GraphEdit-Anwendung. Diese gestattet durch einfaches Aneinanderfügen auf dem System vorhandener Filter und installierter Audio- und Video-Codecs (z.B. MPEG, Divx, mp3, asf), sowie der entsprechenden Multimediageräte, das Entwerfen von Prototypen geplanter Software im Stil eines Graphen, so genannten Filter-Graphen (s. Abb. 2.3). Dabei lassen sich die Filter in drei Gruppen aufteilen: Capture-, Transform- und Renderfilter. Die einzelnen Anschlüsse der Filter, besser bezeichnet als „Input-“ und „Output-Pins“, werden durch ein paar Mausklicks verbunden und innerhalb dieses Prozesses auf Korrektheit überprüft. Für diese Art der Erstellung besitzt die Anwendung eine autonome Intelligenz. Es genügt durchaus, den Start- (Capturefilter) und den Endfilter (Renderfilter), z.B. Kamera und Speicheroption, auf der GraphEdit-Arbeitsoberfläche zu platzieren und verbinden. Die erforderlichen Zwischenfilter (Codecs

[12] s. Abkürzungsverzeichnis
[13] Stand 30.01.2006

etc.) fügt das Programm eigenständig ein. Eine weitere Möglichkeit der Graphenerstellung bildet das Öffnen einer Video- oder Audiodatei (Capturefilter) in GraphEdit, mit dem Ergebnis der kompletten Anfertigung des von dieser Datei verwendeten Graphen mit allen Filtern, Codecs und den korrekten Verbindungen der verschiedenen Pins. Allerdings besitzt das ansonsten sehr mächtige Tool zur Entwicklung von Multimedia-Anwendung keinerlei Möglichkeit den ausgearbeiteten Graphen als C++-Datei zu exportieren, wodurch der entstehende Arbeitsaufwand drastisch gesenkt werden könnte.
Zwei der wohl bekanntesten Software-Entwicklungen mit Einbezug von DirectShow sind der Windows Media Player und der Windows Movie Maker, eine Capturing und Editing Softwarelösung.

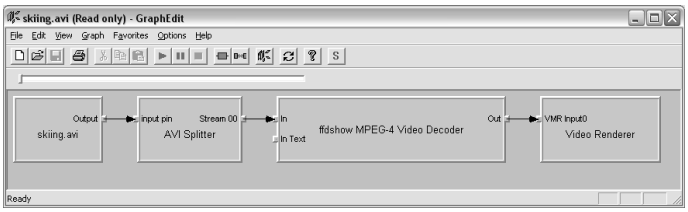



**Abb. 2.3: Beispiel-Graph aus der DirectX-Anwendung Graphedit**

#### 2.2.2.2 SDL

Simple DirectMedia Layer, abgekürzt SDL, ist ähnlich DirectX eine Schnittstelle für Multimedia- und Spieleanwendungen. Jedoch gibt es zwei entscheidende Unterschiede der Multimedia-Bibliothek zur oft zum Vergleich herangezogenen Windows-Multimedia-API. SDL ist plattformunabhängig, läuft dementsprechend auf Betriebssystemen wie Windows, Mac OS X, Solaris, Unix, Linux, und unterliegt ferner der Open Source Lizenz. Im Funktionsumfang steht die SDL-Bibliothek der DirectX-Variante scheinbar in nichts nach. Sie ermöglicht den Zugriff auf Maus, Tastatur, Audiogeräte, Videobuffer und vielem mehr und erlaubt überdies, in Kombination mit der zuvor beschriebenen API OpenGL, Verwendung der 3D-Hardware-Beschleunigung.
Ebenfalls oft anzutreffen, vor allem in der Spieleindustrie, ist die Vereinigung mit der API OpenAL, einer Bibliothek speziell für Audioeffekte. Die Entwicklung und Erweiterung der SDL-Bibliothek erfolgte und erfolgt noch immer in C, ist jedoch in vollstem Umfang mit C++ und vielen anderen Sprachen kompatibel. Für die komplette Liste der unterstützten Sprachen wird auf die offizielle Seite [SDL06] verwiesen. Zu Gute kommt der gesamten Bibliothek der kompakte und leicht verständliche Code, der gerade Einsteigern den ersten Schritt in Richtung Multimedia-Entwicklung erleichtert. Schwer verständliche Abschnitte, die zur Erzeugung von Fensterklassen oder Ähnlichem dienen, werden von dem SDL-Interface im Verborgenen gehalten und fördern so die Übersichtlichkeit eines Programmcodes und richten das Augenmerk auf die Multimedia-Funktionen.
Die Grafik in Abb. 2.4 verdeutlicht die Anbindung von SDL im Gesamtkontext zu anderen Schnittstellen und deren Betriebssystem auf Abstraktionsebene.

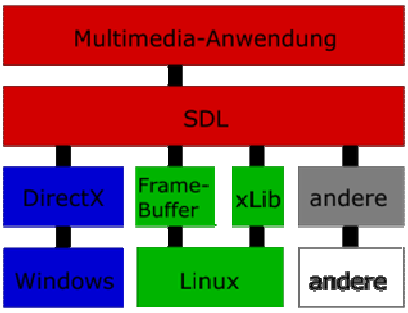



**Abb. 2.4: SDL-Einbindung**

#### 2.2.2.3 Quicktime

Offenkundig wird der Name Quicktime mit der gleichnamigen Videoanwendung in Verbindung gebracht. Jedoch verbirgt sich hinter Quicktime weit mehr. Es vereinigt sowohl einen Framework, bekannt als die Abspielsoftware Quicktime Player, eine multimediale Programmierschnittstelle und das häufig verwendete Dateiformat .mov. Demzufolge ist die von Apple entworfene Quicktime-Schnittstelle ebenso Multimedia-Plattform zur Entwicklung Audio und Video verarbeitender Software. Dabei ist die Nutzung nicht nur auf das Apple-Betriebssystem Mac OS X beschränkt, sondern steht auch für Windows zur Verfügung. Zahlreiche Formate und Darstellungsformen werden problemlos von Quicktime verarbeitet.
Der Verwendungszweck ist dabei nicht nur auf Videos und einfache Audiodateien beschränkt, sondern kann unter anderem auch mit 3D-Grafiken, Streams und vielem mehr umgehen. Eine Besonderheit stellt in diesem Zusammenhang die Erstellung „interaktiver“ Panoramaräume dar. Auch Exporte und Importe diverser gängiger, aber auch zukunftsorientierter Formate werden unterstützt. Dank Beherrschung mannigfaltiger Codecs und Formate ist Quicktime ein Allzweck-Interface. Eine umfangreiche Dokumentation, eine detaillierte Referenz und einfache Einführungsbeispiele sollen dem Programmiereinsteiger den Start in Quicktime erleichtern.

#### 2.2.2.4 GStreamer

Gleichzusetzen mit SDL, handelt es sich bei GStreamer um eine plattformunabhängige und frei nutzbare Open Source Multimedia-API. Dabei ermöglicht GStreamer sowohl Software- als auch Plugin-Programmierung. Der entscheidende Vorteil gegenüber SDL ist der integrierte Graphen-Editor GStream-Editor, der analog zu DirectXs GraphEdit, die Erstellung von Pipelines (DirectX: Filtergraphen) erlaubt. Allerdings scheint dieser nach Angaben der Bereitsteller noch nicht zu hundert Prozent komplettiert zu sein, stelle allerdings schon jetzt ein mächtiges Werkzeug dar. Es werden, wenn gewünscht, die

Verbindungen der einzelnen Elemente bei der Erstellung einer Pipeline selbständig erzeugt (dynamic connection).
Leider war zur Zeit der Implementierung der Diplomarbeit, trotz Unterstützung diverser Programmiersprachen, noch keine Einbindung für .Net oder C++ vorgesehen bzw. noch nicht veröffentlicht.

### 2.2.3 Begründung der Schnittstellen-Wahl

Da sich die Diplomarbeit primär um das Entwickeln der alternativen Steuerung durch Gesten und nicht mit dem Vergleich und der Leistungsmessung bestehender APIs beschäftigt, erfolgten die vorangegangenen Ausführungen über die Schnittstellen ausreichend aber nicht bis ins kleinste Detail.

DirectX und Quartz Extreme sind ohne Zweifel plattformabhängige Schnittstellen. Ihre Basis ist mit dem Betriebssystem Windows bzw. Mac OS X eindeutig definiert. Eine somit verbundene Beschränkung ist ein nicht unbedeutender Aspekt in der Erstellung einer Diplomarbeit im Hinblick auf Portabilität. In diesem Bezug liefern Quicktime, SDL, OpenGL und GStreamer weitaus mehr Möglichkeiten und hätten die Entwicklung einer plattformunabhängigen Software weitestgehend unterstützt.

Dennoch fiel die Entscheidung zugunsten von DirectX, genauer gesagt zugunsten von DirectShow aus. Dies ist unter anderem Ergebnis der während der anfänglichen Recherche gefundenen Tutorials und Beispiele. Um eine möglichst kurze Einarbeitungszeit in die Funktionsweise der Windows-API aufzuwenden und somit letztendlich Zeit für den Schwerpunkt der Arbeit, die Entwicklung der Bildverarbeitungs-Elemente und der Gestenerkennung, zu gewinnen, bot es sich an, bestehende und freie Exemplarbeispiele zusammenzufügen und auszubauen. Hinzu kommt die Wahl von Windows als bevorzugte Umgebung, womit Schnittstellen, die ausschließlich anderen Betriebssystemen vorbehalten waren, von Anfang an ausgeschlossen wurden.

Das Grundgerüst für die gesamte Applikation bilden zum einen die Anwendung *Wincap* aus [Pes03] von Mark D. Pesce, das von ihm ausdrücklich zur Weiterentwicklung und Entwicklung eigener Anwendungen freigegeben wurde, und dem Tutorial *GrabBitmaps* aus der DirectShow C++ Samples Documentation der DirectX SDK. Die an die Anforderungen angepasste Verknüpfung und weiterreichende Erweiterung beider Programmteile dient als Basis der gesamten Software im Rahmen dieser Diplomarbeit. Verwendet wurde DirectShow aus der DirectX 9.0 SDK vom Stand Februar 2003, enthalten auf der CD des Buches von Mark D. Pesce.

Des Weiteren ist das Einbinden der OpenGL-Bibliothek beabsichtigt, um das angesteuerte Feedback der gezeichneten Gesten implementieren zu können. Für dieses Vorhaben reichen einfache Befehle von OpenGL aus. Rechenintensive Methoden und detailreiche Renderfunktionen bleiben deshalb unangetastet. Damit erübrigt sich das Übernehmen einer weiteren multimedia-orientierten Schnittstelle, da die grundlegenden

Kommandos und Funktionen von OpenGL mehr als ausreichen. Die Entscheidung für OpenGL beruht zudem teilweise auf den bereits vorhandenen Kenntnissen der Programmierung in OpenGL, erarbeitet während den Vorlesungen der Computergraphik 1 und Computergraphik 2 bei Prof. Dr. Müller an der Universität Koblenz.

# 3 Image Processing

Im folgenden Kapitel wird der Leser auf den bedeutungsvollen Bildverarbeitungsanteil der Implementierung vorbereitet und es werden notwendige Methoden und Operationen der Bildverarbeitung im Detail vorgestellt. Dies geschieht weitestgehend im Hinblick auf den Nutzen für das zu entwickelnde System.

Generell ist das primäre Ziel der Bildverarbeitung die Signalaufarbeitung. Ziele sind zum Beispiel die Verbesserung der Bildqualität oder das Wiederfinden bekannter oder das Finden unbekannter Objekte in vorliegenden zwei- oder dreidimensionalen Bildszenen. Im Gegensatz dazu steht die Bildbearbeitung, die sich mit der Manipulation und Enddarstellung vorhandener Bilder beschäftigt. Der Verlauf einer Objektextraktion im Rahmen der Bildverarbeitung verfolgt üblicherweise den im Diagramm (s. Abb. 3.1) dargestellten Weg.

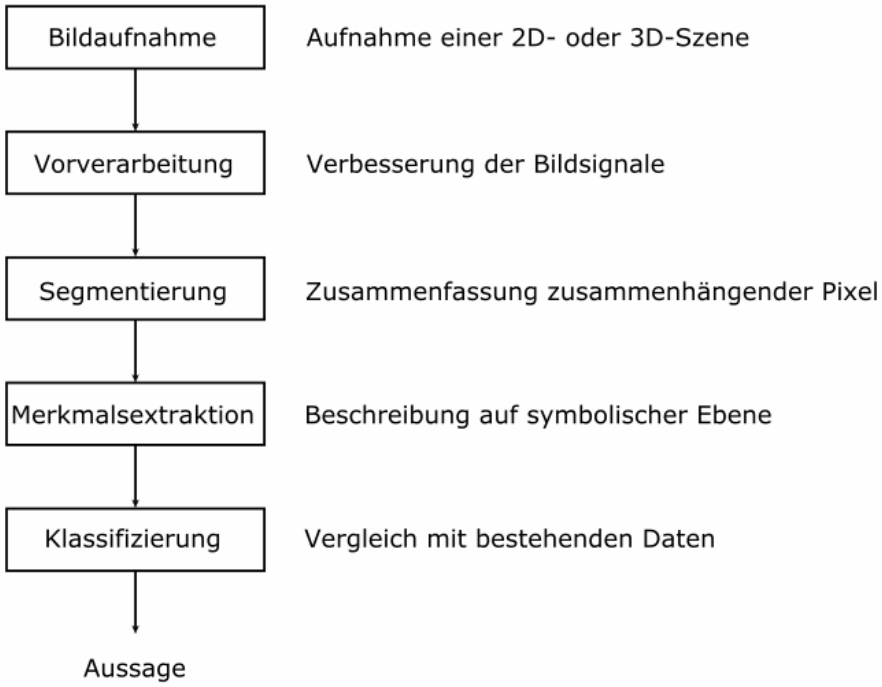



**Abb. 3.1: Stufen eines Bilderkennungssystems**

Ausgehend von einer 2D- oder 3D-Aufnahme einer Szene werden nacheinander bildverarbeitende Schritte ausgeführt. In der Bildvorverarbeitung sollen Störungen des Bildes weitestgehend beseitigt werden. Diese können beispielsweise in Form von, mit ihrer Umgebung verglichen, abweichenden Farbwerten einzelner Pixel vorliegen. *Kapitel 3.3.2 Glättung* beschreibt eine Möglichkeit der Vorverarbeitung eines Bildes hinsichtlich dieser so genannten Ausreißer. Ebenso werden im Allgemeinen Farbwertangleichungen, wie etwa Kontrasterhöhung, des gesamten Bildes als Maßnahme der Vorverarbeitung vorgenommen.

Die Segmentierung, die Merkmalserkennung und die Klassifizierung der Objekte können auf vielerlei Weisen erfolgen und hängen von den zu erreichenden Zielen des Entwicklers und des gewünschten Einsatzgebietes ab. Grundlegend geschehen diese durch die Verarbeitung der Signale durch Extraktion, Auswertung und Sicherung von enthaltenen Informationen. Im Rahmen dieser Diplomarbeit werden zum Beispiel die Kantendetektion und die Kreis-Erkennung mit der Hough-Transformation als Verarbeitungsschritte vorgestellt.

Bildverarbeitung versteht sich als Vorstufe der Bildbearbeitung. Ziel der Bildverarbeitung ist die vorwiegend selbstständige maschinelle Bearbeitung, beispielsweise die automatische Mustererkennung.
In folgenden Abschnitten wird intensiver auf benötigte Methoden der Bildverarbeitung, die zum Erfolg bei der Umsetzung dieses Projektes beitragen, eingegangen. Dabei ist nicht unbedingt deren genaue Reihenfolge innerhalb des Programmkerns eingehalten. Lediglich die Sachbegriffe und die Funktionsweise genannter Operationen werden in diesem Kapitel erläutert. Der genaue Zusammenhang und die Ergebnisse kombinierter Methoden werden erst im nachfolgenden *Kapitel 5 Implementierung* geschildert.

## 3.1 Farbräume und Farbmodelle

Als Farbräume und Farbmodelle werden sämtliche darstellbare Farben eines Peripheriegerätes mit dem Schwerpunkt der Bildeingaben und Bildausgaben oder der menschlichen Farbwahrnehmung zusammengefasst. Die Farben selbst scheinen eng verbunden mit der Wahrnehmung des jeweiligen Betrachters, zum Beispiel Mensch oder Tier. Manche Tiere können sogar Farbtöne außerhalb des wahrnehmbaren Wellenlängen-bereichs des Menschen erkennen. Die Repräsentation eines Farbraumes oder Farbmodells erfolgt üblicherweise in einem Koordinatensystem, dem Farbraumsystem, in dem jeder Koordinate eine eindeutige Farbe zugeordnet ist. Wenn ein Farbraum und sein zugehöriges System nicht differenziert betrachtet werden, so sind sie in einem Modell, dem Farbmodell, zusammengefasst. Der verwendete Farbraum oder das Farbmodell ist abhängig von der Art des Gerätes und der angestrebten Funktionalität desgleichen. Dabei ist es möglich Farbinformationswerte von einem Farbraum in einen anderen oder ein Farbmodell umzurechnen.
Der verwendete Farbraum des eingespeisten Bildes der Webcam beim grabben[14] ist der RGB[15]-Raum, da dieser für das grafische Objekt, ein Bitmap, dessen Abspeicherung im weiteren Programmverlauf erfolgt, gefordert wird. Eigentlich wäre unter diesem Aspekt die Recherche für einen weiteren alternativen Farbraum oder die Analyse eines anderen Farbmodells uninteressant. Nichtsdestotrotz ist die Untersuchung der wichtigsten Farbräume und -modelle unentbehrlich, da eine später beschriebene Idee der Umsetzung der Gestenerkennung ein, dem in dieser Situation ungeeignetem RGB-Farbraum, überlegenes oder vorteiligeres System fordert. Für diesen Fall werden, zusätzlich zum

[14] Herausnehmen eines einzigen Frames aus einer Sequenz von Bildern
[15] s. Abkürzungsverzeichnis

RGB-Raum, zwei der üblichsten Farbräume und Farbmodelle, nämlich der HSV[16]-Raum und das YUV[17]-Modell, betrachtet. Der weit verbreitete CYMK[18]-Farbraum fällt bei den grundlegenden Erläuterungen weg, da er auf den Bereich des Druckens beschränkt ist. Auch auf das YCrBr-Modell wird aufgrund der nur geringfügigen Unterschiede zum YUV-Modell nicht weiter eingegangen.

### 3.1.1 RGB-Farbraum

Der zuerst erwähnte Farbraum ist der RGB-Raum. Hierbei handelt es sich um ein additives Farbsystem, bei dem sich die Grundfarben, Rot, Grün und Blau, aufeinander addiert zu der Farbe Weiß ergänzen. Strikter formuliert ist die Wahl der referierten Grundfarben scheinbar willkürlich, da es genügt, dass sie, vergleichbar mit der Vektorrechnung, mathematisch unabhängig voneinander sind und sich dementsprechend die dritte Farbe nicht durch die beiden anderen konstruieren lässt. Die vielleicht am meisten einleuchtende Begründung der Wahl von Rot, Grün und Blau als Grundfarben beruht auf Untersuchungen der in den Augen befindlichen Sinneszellen, die Rot-, Grün- und Blauzapfen. Nebenbei erwähnt sei die Tatsache, dass z.B. die Zapfen vom Typ B (wie Blau) auch violette Farbtöne registrieren.
Als Angabe für den Anteil der jeweiligen Grundfarbe an einem gemischtem Farbton sind Werte im Bereich von [0%,100%], [0,1] oder [0,255] zulässig. Künftig wird in der Implementierung dieser Diplomarbeit der Wertebereich [0...255] bevorzugt. Detaillierter betrachtet bedeutet dies für die Grundfarbe Rot einen Wert von (255, 0, 0), für Grün (0, 255, 0) und für Blau (0, 0, 255). Zusammen addiert ergibt sich die Kombination (255, 255, 255), was, wie man leicht aus dem RGB-Würfel (s. Abb. 3.2) entnehmen kann, der Farbe Weiß entspricht. Insgesamt sind im RGB-Raum somit 256 x 256 x 256, also über 16 Millionen Farben darstellbar. Für jeden Farbwert ist in der Computerdarstellung jeweils ein Byte vorgesehen.
Diese Art der Darstellung eignet sich jedoch ausschließlich für Bilddaten am Bildschirm. Jeder der schon einmal eine Fernseheroberfläche aus der Nähe betrachtet hat, konnte die farbig separierten Kanäle erkennen, die aus der Ferne den Eindruck einer Einzelfarbe erwecken. Diese Farbe ist jedoch nur ein Zusammenspiel der einzelnen Kanäle, die unterschiedlich intensiv leuchten.
Es ist möglich den Farbraum auf RGBA zu erweitern, das heißt es wird noch ein Alpha-Byte für die Transparenz angelegt. Dieser beschreibt die Durchlässigkeit für unterliegende Farbinformationen des Pixels.
Der RGB-Farbraum wies einige Nachteile im Kontext mancher Anforderungen auf, aufgrund derer zusätzliche Farbsysteme definiert werden mussten. Möchte man zum Beispiel Daten für einen Ausdruck präparieren, so muss auf der subtraktiven Farbmischung (z.B. CYMK) aufgebaut werden.

[16] s. Abkürzungsverzeichnis
[17] s. Abkürzungsverzeichnis
[18] s. Abkürzungsverzeichnis

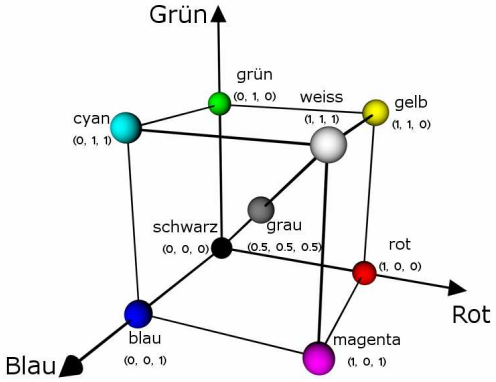



**Abb. 3.2: RGB-Würfel**

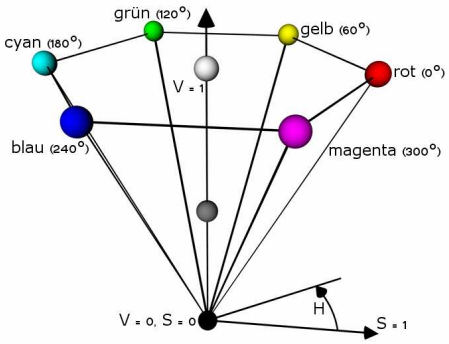



**Abb. 3.3: HSV-Pyramide**

### 3.1.2 HSV-Farbraum

Wenn man sich den HSV-Farbraum als geometrische Figur vorstellen möchte, so wäre eine auf den Kopf gestellte sechseckige Pyramide der typische Formgeber (s. Abb. 3.3). Man kann die Form leicht durch eine geometrische Ableitung des RGB-Würfels begründen und nachvollziehen. Der Würfel wird entlang der Grauwert-Diagonalen gestaucht und anschließend das entstandene Sechseck in eine Pyramide überführt. HSV, oder seltener HSB, teilt Farben nicht in deren Anteile der Grundfarben, sondern in einen Farbwinkel H (Hue), eine Farbsättigung S (Saturation) und eine Helligkeit V oder B (Value, Brightness). Diese Art der Einteilung ist der, der menschlichen Wahrnehmung besser nachempfunden. Ein Farbton wird in diesem Farbraum durch einen Winkel definiert. Vereinbarungsgemäß liegt die Farbe Rot auf 0º, Grün auf 120º und Blau auf 240º. Gegenüberliegende und damit um 180º verschobene Farben bilden jeweils die Komplementärfarben. Die Helligkeit V liegt im Bereich von 0 bis 1, wobei ersteres Schwarz und letzteres Weiß bedeutet, und verläuft entlang der senkrecht nach oben zeigenden Achse. Zwischen der hochragenden V-Achse und den äußeren Farbwinkeln ist die Sättigung S definiert. Sie gibt die Intensität der Farbe mit Angabe des Abstandes zur V-Achse mit einem relativen Wert zwischen 0 und 1 an. Beispielsweise stellt eine Kombination von HSV (120º, 1, 1) die Primärfarbe Grün dar.

Zur Umrechnung eines HSV-Farbwertes in einen RGB-Wert kann folgende Formel (s. Abb. 3.4) zur Hilfe genommen werden. Es muss allerdings beachtet werden, dass die Resultate für RGB im Wertebereich [0,1] liegen. Demnach ist je nach Anforderung noch eine Umrechnung auf den Wertebereich [0,255] mit einzubeziehen.

$$H_i = \left\lfloor \frac{H}{60} \right\rfloor \bmod 6$$

$$f = \frac{H}{60} - H_i$$

$$p = V \cdot (1 - S)$$

$$q = V \cdot (1 - (S \cdot f))$$

$$t = V \cdot (1 - (S \cdot (1 - f)))$$

$$if\ Hi = 0 \rightarrow R = V, G = t, B = p$$

$$f\ Hi = 1 \rightarrow R = q, G = V, B = p$$

$$f\ Hi = 2 \rightarrow R = p, G = V, B = t$$

$$f\ Hi = 3 \rightarrow R = p, G = q, B = V$$

$$f\ Hi = 4 \rightarrow R = t, G = p, B = V$$

$$f\ Hi = 5 \rightarrow R = V, G = p, B = q$$

**Abb. 3.4: Formeln zur Umrechnung von HSV-Pixelwerten in RGB-Farbwerte**

Genauso ist auch eine Berechnung von RGB- in HSV-Werte möglich. Nachstehend befindet sich die entsprechende Formel (s. Abb. 3.5) für die Umrechnung bei bekannten RGB-Vorgaben. Vorab ist eine Prüfung für S = 0 notwendig, da in diesem Fall ein Grauwert vorliegt und sowohl der R-, G- als auch der B-Kanal den Wert V erhalten. Auch hier müssen die vorher bekannten Werte von RGB im Bereich von [0,1] vorliegen.

$$Max = \max(R, G, B)$$

$$Min = \min(R, G, B)$$

$$if\ R = Max \Rightarrow H = \left(0 + \frac{G - B}{Max - Min}\right) \cdot 60$$

$$if\ G = Max \Rightarrow H = \left(2 + \frac{B - R}{Max - Min}\right) \cdot 60$$

$$if\ B = Max \Rightarrow H = \left(4 + \frac{R - G}{Max - Min}\right) \cdot 60$$

$$if\ (H < 0) \Rightarrow H = H + 360$$

$$S = \frac{Max - Min}{Max}$$

$$V = Max$$

**Abb. 3.5: Formel zur Umrechnung vom RGB- in den HSV-Farbraum**

### 3.1.3 YUV-Farbmodell

Entwickelt wurde das YUV-Modells bei der Suche nach Darstellungsmöglichkeiten für den Farbfernseher, die erstens Unterstützung älterer Geräte und zweitens dies ohne Erhöhung der benötigten Bandbreite gewährleisteten. Das Modell ist nunmehr in der Fernseh-Norm PAL[19] verankert. Grundsätzlich besteht YUV aus zwei Bestandteilen, der Luminanz und der Chrominanz, wobei letztere aus zwei Farbdifferenzsignalen der Farben Rot und Blau besteht. Nachstehende Formel (s. Abb. 3.6) soll die Zusammensetzung der einzelnen Werte verdeutlichen.

[19] s. Abkürzungsverzeichnis

$$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$
$$U = B - Y$$
$$V = R - Y$$

**Abb. 3.6: Ermittlung der YUV-Werte aus gegebenen RGB-Werten**

Vor der Berechnung sollte immer eine Gamma-Korrektur mit dem Gamma-Wert des Ausgabegerätes erfolgen.
Aufgrund der Tatsache, dass das menschliche Auge Luminanz besser wahrnimmt als Chrominanz, arbeiten viele auf dem YUV-Modell aufbauende Modelle mit einer Kompression der Chrominanz, um Leistungsvorteile nutzen zu können. Die stärkere Wahrnehmung lässt sich auf die hohe Anzahl der Helligkeit verarbeitenden Zapfen (ca. 120 Millionen) im Auge zu der vergleichsweise geringen Anzahl der Farbzapfen (ca. 7 Millionen) begründen.

## 3.2 Filter

Erst einmal frei definiert sind Filter Operatoren, deren Aufgabe es ist, das aktuelle Pixel im Kontext zu seinen Nachbarpixeln zu betrachten, um das neue Pixel im Ausgabebild zu berechnen. Dabei übernimmt dieser neu bestimmte Bildpunkt die Positionskoordinaten des ursprünglichen.
Filter werden durch Masken umgesetzt, die das gesamte Bild durchlaufen (s. Abb. 3.7) und alle unter ihnen liegenden Pixel nach bestimmten Kriterien mit in die Berechnung einbeziehen. Dabei gibt es verschiedene Verwendungszwecke für Filter. Sie erfüllen unter Anderem folgende Aufgaben:

- Rauschunterdrückung (gleichmäßige verteilte Störungen, „Schnee“)
- Erkennung fehlerhafter Pixel (defekter CCD-Chip)
- Kantendetektion
- Glättung (Minderung der Schärfe)
- Hervorhebung von größeren Details

Allgemein lässt sich sagen, dass es sich bei Filtern um lokale Operatoren handelt, die, je nach Zweck des Filters, unwichtige Informationen verwerfen oder wichtige Informationen intensivieren.
Neben der Gruppe der lokalen Operatoren gibt es noch die Bildpunkt-Operatoren und die globalen Operatoren oder Funktionen. Bildpunkt-Operatoren arbeiten auf Pixelebene und entscheiden nach vereinbarten Kriterien für jedes einzelne Pixel ungeachtet der anderen Bildpunkte. Ein Beispiel für die Bildpunkt-Operationen ist das Schwellwertverfahren, bei dem jedes Pixel mit einem Schwellwert $\theta$ verglichen und dementsprechend neu gesetzt wird. Die Binarisierung eines Bildes ist eine typische Anwendung für das Schwellwert-Verfahren.

Globale Operationen beziehen die Gesamtheit aller Pixel eines Bildes mit ein. Üblicherweise dienen solche Operationen zur Gewinnung von allgemeinen Aussagen über die vorliegende Abbildung. Die im *Kapitel 3.3.1 Grauwert-Bild* beschriebene Helligkeitsbestimmung eines Bildes mit Grautönen ist ein Vertreter der globalen Funktionen. Ergänzend existieren die bereits genannten lokalen Operatoren, die nur definierte Bereiche mit in die Betrachtung einfließen lassen.

Die Form einer Filter-Maske baut auf einer zweidimensionalen M x N Matrix auf, wobei Maskendimensionen von 3x3 und 5x5 üblich sind. Durch diese Maske werden bei der Untersuchung des aktuellen Pixels nur dessen durch die Maskengröße definierten Nachbarpixel mit einbezogen und als das resultierende Ergebnis der Maskenoperation in der neu zu erstellenden Bildmatrix gesetzt. Besondere Aufmerksamkeit bei Filter-Operatoren verdient die Randbehandlung. Verwendete Methoden der Randbehandlung sind zum Beispiel das Übernehmen der Ursprungspixel oder das Benutzen spezieller Randbearbeitungs-Masken.

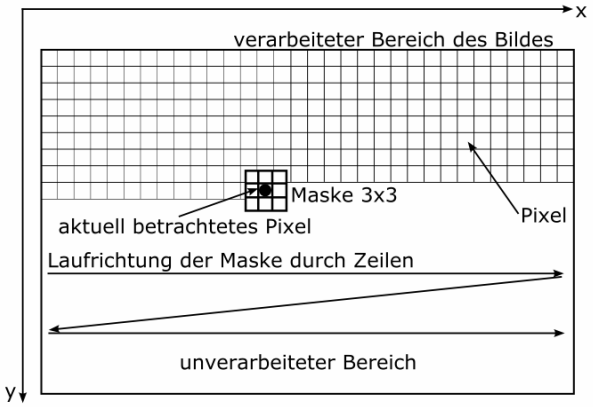



**Abb. 3.7: Maskendurchlauf im Bild**

Ein wichtiges Anwendungsgebiet von Filtern offenbart sich in der Kantendetektion, da durch sie erhebliche Schritte im Bereich der Objekterkennung gemacht werden.
In den nachfolgenden Kapiteln werden einige wichtige Einsatzfelder von Filtern mit Beispielen angeführt.

## 3.3 Bildvorverarbeitung

### 3.3.1 Grauwert-Bild

Vereinfacht betrachtet ist ein Grauwert-Bild ein zweidimensionales Koordinatensystem mit x Spalten und y Zeilen mit der Funktion f(x, y) = Grauwert [0,255]. Der maximale Wert von 255 ist in der Abspeicherung von einem Byte, also $2^8$ Bit, begründet. Alle vorhandenen Grauwerte werden, um sie dem Computer zur Berechnung zur Verfügung zu stellen, diskretisiert und als zweidimensionales Feld, eine Matrix mit N Zeilen und M

Spalten, dargestellt. Vereinbarungsgemäß beginnt die Bezeichnung der Zeilen und Spalten bei Null und verläuft in positiver Richtung. Der Ursprung liegt üblicherweise in der linken oberen Ecke.

Sinn und Zweck der Umwandlung eines Farbbildes in ein Grauwert-Bild ist die eventuell nötige Reduktion der Daten. Sofern Farbinformationen für den weiteren Berechnungsablauf irrelevant sind, ist eine Umrechnung in Grauwerte empfehlenswert. Im Falle bestimmter nachfolgender Bildverarbeitungs-Operationen, z.B. eine Kantendetektion, werden Farbbilder gewöhnlich erst in Grauwert-Bilder umgewandelt. Betrachtet man nochmals den RGB-Würfel (s. Abb. 3.2), so erkennt man eine Verbindung zwischen den Farben Schwarz und Weiß, welche die Grauwert-Diagonale repräsentiert. Auf dieser Gegebenheit beruhend gelangt man zu der Annahme, dass Grauwerte jeweils im R-, G- und B-Kanal denselben Anteil besitzen. Hier sei schon ein Teil der Implementierung vorweg genommen. Bei der Umrechnung der Farb- in Grautöne, deren Wert direkt als Zusatzinformationen in den Objekten der Klasse `RGBPixel` bereitgestellt sind, werden die einzelnen Grauwerte eines Pixels allerdings nach der folgenden Formel in Abb. 3.8 berechnet.

$$Grey = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

**Abb. 3.8: Umrechnung eines RGB-Pixel-Wertes in einen Grauwert**

Die unterschiedliche Gewichtung der Primärfarben ist zurückzuführen auf das Sehverhalten des menschlichen Auges. Farben werden demnach in unterschiedlicher Stärke in der Reihenfolge Grün, Rot, Blau wahrgenommen. Durch die vorangestellte Formel errechnet man nichts anderes als die Helligkeit eines einzigen Pixels anhand vorliegender RGB-Werte, unabhängig von der Farbe. Vergleicht man die Formel mit der Berechnungsvorschrift für die Luminanz Y im YUV-Farbmodell, fällt die Ähnlichkeit auf. Demnach ist eine Matrix der Y-Werte gleichzusetzen mit der Darstellung eines Grauwert-Bildes.

Eine Gesamtaussage über die Helligkeit eines Bildes lässt sich durch nachstehende Formel errechnen. Daraufhin ist erkennbar, ob es sich bei dem Bild um eine relativ helle oder dunkle Vorlage handelt. M und N bezeichnen wie bereits erwähnt die Bilddimensionen. Es werden alle Werte der Grauwert-Pixel aufaddiert und durch die Anzahl dividiert (s. Abb. 3.9).

$$Mittelwert_G = \frac{1}{M \cdot N} \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} g(i, j)$$

**Abb. 3.9: Mittelwertbildung eines Grauwert-Bildes**

### 3.3.2 Glättung

Es kann bei aufgenommenen Bildern immer wieder zu ungewollten zufälligen Störungen innerhalb des Bildes kommen. Hervorgerufen durch schlechte Qualität der Aufnahmen, durch defekte CCD[20]-Chips oder sonstige Einflüsse gilt es dieses Rauschen (engl.: noise) im Bild zu unterdrücken oder noch besser, zu eliminieren. Das Rauschen ist folglich weitestgehend von der Aufnahmetechnik beeinträchtigt. Es existieren diverse Filter mit dem Ziel der Glättung eines Bildes, allerdings mit unterschiedlichen Erfolgsquoten und diversen Vor- und Nachteilen. Bei den zugehörigen Filtern handelt es sich um Masken mit definierten Dimensionen, die je nach Filter variieren können und sowohl Qualität als auch Rechenaufwand erhöhen. Solche Filteroperatoren zur Glättung können in zwei größere Hauptgruppen, nämlich in die der linearen und der nicht-linearen Filter, denen auch die Rangordnungsfilter angehören, unterteilt werden. In verschiedener Literatur findet man des Weiteren noch Aufteilungen in parallel und sequentiell arbeitende Filter. Parallele Filter beziehen nur im Ausgangsbild existierende Pixel mit ein, während bei der sequentiellen Bearbeitung schon die neu ermittelten Werte des Ergebnisbildes mit einfließen.

#### 3.3.2.1 Lineare Filter

Zur Glättung eines Bildes wird unter anderem der lineare Mittelwert-Filter, wie in Abbildung (s. Abb. 3.10) gezeigt, verwendet. Es gibt allerdings auch andere, weniger gebräuchliche Matrizen zur Beschreibung des Mittelwert-Filters, die eine andere Gewichtung aufweisen (s. Abb. 3.11). Formal ausgedrückt berechnet der Mittelwert-Filter das in der Matrix des neuen Bildes zu setzende Pixel $P_{neu}$ aus dem korrespondierenden Pixel $P_{alt}$ und dessen Nachbarpixeln nach der Vorschrift aus Abb. 3.12 (gilt für Filter der Größe 3x3).

$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$$

**Abb. 3.10: Mittelwert-Filter**

$$\frac{1}{10}\begin{pmatrix}1 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 1\end{pmatrix}$$

**Abb. 3.11: modifizierter Mittelwert-Filter**

$$P_{neu}(x,y)=\frac{1}{9}\sum_{m=-1}^{1}\sum_{n=-1}^{1}P_{alt}(x-m,y-n)$$

**Abb. 3.12: Formel des Mittelwert-Filters für die Größe von 3x3 Feldern**

Die entstehende Summe der unter der Maske liegenden Werte wird abschließend durch den Gesamtwert der eingetragenen Einheiten dividiert.

[20] s. Abkürzungsverzeichnis

Der Mittelwert-Filter als lokaler Operator hat zwar den Vorteil vorhandenes Rauschen im Bild zu vermindern, allerdings verwischt er auch für die Objekterkennung wichtige Kanten und Konturen. Die folgenden Bilder (s. Abb. 3.13 bis Abb. 3.16) sollen diese Tatsache verdeutlichen. Eindeutig zu erkennen ist die Wirkung des schwarzen Störungspixels, das seine Umgebung durch Verwendung des Mittelwert-Filters wesentlich beeinflusst.

**Abb. 3.13: Minimal-Bild von 6x5 Pixel mit Rauschen als Beispiel**

**Abb. 3.14: Minimal-Bild nach Verwendung des Mittelwert-Filters mit Übernahme der ursprünglichen Randpixel**

| | | | | | |
|---|---|---|---|---|---|
| 180 | 180 | 0 | 0 | 110 | 110 |
| 180 | 180 | 0 | 255 | 110 | 110 |
| 110 | 180 | 0 | 0 | 110 | 110 |
| 180 | 180 | 180 | 180 | 180 | 180 |
| 180 | 180 | 180 | 180 | 180 | 180 |

**Abb. 3.15: Zahlenrepräsentation des verrauschten Grauwert-Bildes mit Markierung der Ausreißer**

| | | | | | |
|---|---|---|---|---|---|
| 180 | 180 | 0 | 0 | 110 | 110 |
| 180 | 112 | 88 | 65 | 101 | 110 |
| 110 | 132 | 128 | 112 | 137 | 110 |
| 180 | 152 | 140 | 132 | 144 | 180 |
| 180 | 180 | 180 | 180 | 180 | 180 |

**Abb. 3.16: Zahlenrepräsentation des durch den Mittelwert-Filter geglätteten Bildes**

Ein weiterer Filter zum Zweck der Rauschminderung ist der Gauss-Filter, wie er auch in der Programmumsetzung verwendet wurde. Von diesem werden auch die Binominal-Filter abgeleitet. Die relativ komplexe Formel nach Gauss (s. Abb. 3.17) kann zur Erstellung eines M x N-dimensionalen Filters anhand des Pascal´schen Dreiecks (s. Abb. 3.18) angenähert werden.

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

**Abb. 3.17: Gauss-Formel**

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
N       M   M   M   M   M   O
```

**Abb. 3.18: Pascal´sches Dreieck**

Dementsprechend gestaltet sich der Aufbau des Gauss-Filters wie in der Abbildung 3.19 für Spalten M = 3 und Zeilen N = 3 und für M = N = 5 (grafische Darstellung der

Gewichtung des 5x5-Filters in Abb. 3.20). Die Konstruktion von Binominal-Filtern größerer Dimensionen verhält sich analog. Laut verwendeter Literatur erzeugt der Gauss-Filter als einziger Operator keine Artefakte, was unter anderem, dem später erwähnten LaPlacian-of-Gaussian-Operator sehr zu Gute kommt.

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix} \qquad \frac{1}{256}\begin{pmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{pmatrix}$$

**Abb. 3.19: Gauss-Filter der Größe (3x3) und (5x5) (von links nach rechts)**

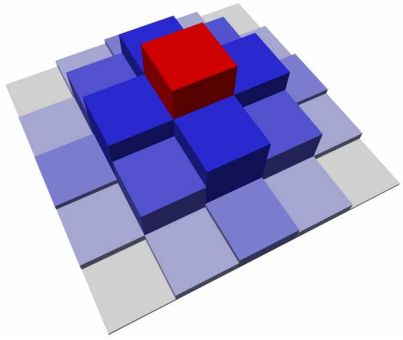


**Abb. 3.20: Grafische Darstellung der Gewichtungsverteilung einer 5x5 Gauss-Maske; Gewichtung des aktuell betrachteten Pixels unter der Maske ist rot hervorgehoben**

Ein nicht unerheblicher Vorteil der linearen Filter ist die Separierbarkeit in einen horizontalen und vertikalen Operator. Alle symmetrischen Filter sind aufgrund ihrer Struktur separierbar. Eine Separation von Filtern resultiert in einer Geschwindigkeits-steigerung von implementierten Programmen.

#### 3.3.2.2 Rangordnungs-Filter

Weiterentwickelte Operatoren schwächen den Nachteil der Kantenverwischung ab. Hierzu gehören die Rangordnungsfilter. Sie besitzen kantenerhaltenden Charakter. Rangordnungs-Filter bringen die unter ihrer Maske befindlichen Werte durch Sortierung in eine Reihenfolge. Dabei besitzen die Masken stets eine ungerade Zeilen- und Spalten-Anzahl.
Der wohl am meisten verwendete Rangordnungsfilter ist der Medianfilter. Nach Anordnung der Werte des betrachteten Pixels und seiner, durch die Maskendimensionen definierten, Nachbarpixel (s. Abb. 3.21), nach einer „<“-Relation (s. Abb. 3.25 und Abb. 3.26), wird der, lokal gesehen, mittlere Wert herausgenommen und in der neuen Bildmatrix gesetzt. Demzufolge entstehen keine neuen Grauwerte, sondern es werden

Werte aus der alten Bildmatrix übernommen. Abb. 3.21 bis Abb. 3.26 soll die Vorgehensweise näher erläutern.

**Abb. 3.21: Minimal-Bild von 6x5 Pixel mit Rauschen als Beispiel**

**Abb. 3.22: Minimal-Bild nach Verwendung des Medianfilters mit Übernahme der ursprünglichen Randpixel**

| | | | | | |
|---|---|---|---|---|---|
| 180 | 180 | **0** | **0** | **110** | 110 |
| 180 | 180 | **0** | **255** | **110** | 110 |
| 110 | 180 | **0** | **0** | **110** | 110 |
| 180 | 180 | 180 | 180 | 180 | 180 |
| 180 | 180 | 180 | 180 | 180 | 180 |

**Abb. 3.23: Zahlenrepräsentation des verrauschten Grauwert-Bildes mit markierter 3x3-Maske**

| | | | | | |
|---|---|---|---|---|---|
| 180 | 180 | 0 | 0 | 110 | 110 |
| 180 | 180 | 0 | **0** | 110 | 110 |
| 110 | 180 | 0 | 0 | 110 | 110 |
| 180 | 180 | 180 | 180 | 180 | 180 |
| 180 | 180 | 180 | 180 | 180 | 180 |

**Abb. 3.24: Zahlenrepräsentation des durch den Medianfilter geglätteten Bildes**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 110 | 0 | 255 | 110 | 0 | 0 | 110 |

**Abb. 3.25: ungeordnete Werte unter der Maske**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** | 110 | 110 | 110 | **255** |

**Abb. 3.26: sortierte "Rangordnung" mit markiertem Übernahmewert**

Ferner sollen weitere Rangordnungs-Filter nicht unerwähnt bleiben. Der k-trimmed-mean-Filter (s. Abb. 3.27) und der Mid-Range-Filter (s. Abb. 3.28) gehören ebenso in diese Kategorie. Auch bei ihrer Nutzung erfolgen eine Sortierung des Center-Pixels und seiner Nachbarn, sowie die anschließende Auswahl oder Berechnung der Werte nach Vereinbarung. Beim k-trimmed-mean-Filter werden sowohl am Anfang als auch am Ende *k* Elemente ignoriert. Sie fließen bei der Bildung des Durchschnitts nicht mit ein. Dem gegenüber berechnet der Mid-Range-Filter den Durchschnitt des ersten und letzten Elementes der sortierten Wertereihe. Beide Filter ermöglichen jedoch das Entstehen neuer Grauwerte in der neuen Bildmatrix.

$$value_{trimmed} = \frac{1}{N-2k} \cdot \sum_{i=1+k}^{N-k} x_i$$

**Abb. 3.27: Formel k-trimmed-mean-Filter**

$$value_{mid-range} = \frac{1}{2}(x_1 + x_N)$$

**Abb. 3.28: Formel Mid-Range-Filter**

## 3.4 Objekterkennung

In diesem Kapitel werden einige Bildverarbeitungs-Methoden zur Erkennung von Objekten behandelt. Man könnte diese Methoden in zwei Kategorien einteilen. Sowohl die modellbasierte als auch die eigenschaftsbasierte Methoden durchsuchen das Bild nach Objekten der jeweilig vereinbarten Kriterien. Modellbeschreibende Kriterien sind geometrische Figuren, Kanten und Konturen, die durch nachfolgend beschriebene Verfahren der Kantendetektion und der Kreis-Erkennung nach Hough aus dem Bild extrahiert werden können. Eigenschaftsbeschreibende Merkmale eines Körpers sind vordergründig die Farbe und weiterhin möglicherweise Reflektionsverhalten des Körpers. Stets zu beachten ist, dass eine Objekterkennung auf Farbbildern größere Erfolge verspricht, als die Erkennung von Objekten in Grauwert-Bildern. Dies begründet sich in der zusätzlich enthaltenen Information durch Auswertung der Farbwerte, anhand der sich Objekte eindeutiger identifizieren lassen. Allerdings erfolgt die Verarbeitung von Farbbildern auf Kosten der Leistung, wodurch abzuwägen gilt, ob eine auf Grauwert-Bilder angewandte modellbasierte Suche zur Erkennung oder Identifikation von Objekten nicht ausreichend operiert.

### 3.4.1 Schwellwertverfahren

Das wohl einfachste Verfahren zur Segmentierung und Erkennung von Objekten in einem Bild ist die Schwellwert-Segmentierung. Diese kann mit einer Binarisierung eines Bildes gleichgesetzt werden. Lassen sich Objekte schon aufgrund ihres Grauwertes im Bild beschreiben ist das Verfahren zumeist ausreichend. Ausgeführt wird diese Operation auf das ganze Bild, also global.

$$g_{n(x,y)} = \begin{cases} schwarz, wenn\ 0 \le g_{p(x,y)} < S \\ weiss, wenn\ S \le g_{p(x,y)} \le 255 \end{cases}$$

**Abb. 3.29: Schwellwert**

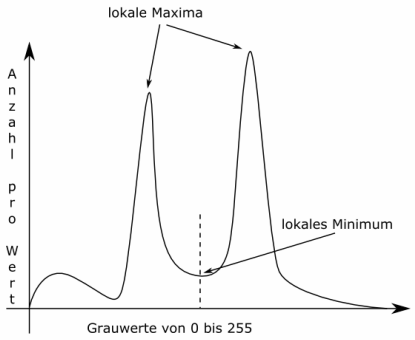



**Abb. 3.30: Histogramm**

Alle Pixel des Ursprungsbildes werden dabei mit einem Schwellwert *S* verglichen und dementsprechend die neuen Pixel *n* auf Schwarz oder Weiß gesetzt (s. Abb. 3.29). Es entsteht ein Binärbild. Zudem besteht auch die Möglichkeit den Schwellwert dynamisch, an das Bild angepasst, zu ermitteln. Dazu wird jedes einzelne Pixel des Ausgangsbildes statistisch in einem Histogramm (s. Abb. 3.30) festgehalten. Für den Grauwert *g* eines Pixels *p* mit den Koordinaten *(x, y)* wird an der Stelle *g* im Histogramm der Zählwert um

„1“ inkrementiert. Dem folgend wird das lokale Minimum zwischen den zwei lokalen Maxima als Schwelle eingesetzt. Im neu entstandenen Ausgangsbild sollten sich dann, je nach Einsatz der Schwelle, Bildpunkte des gesuchten Objekts hervorhebend darstellen.

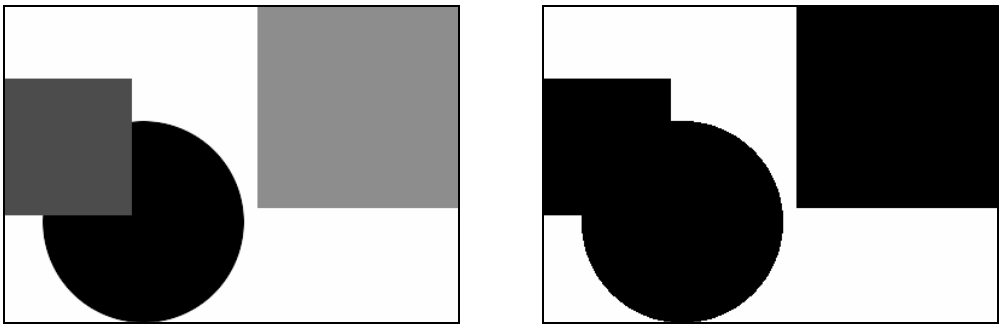


**Abb. 3.31: links: Beispielbild mit mehreren Objekten; rechts: nach Schwellwertverfahren mit Schwelle zwischen weißem Hintergrund (1. Maximum) und hellerem Quadrat (2. Maximum)**

### 3.4.2 Kantendetektion

Ein Mensch ist im Normalfall in der Lage, Objekte und Hintergrund klar voneinander zu trennen, wenn er ein Foto oder Bild betrachtet. Enthält die unter Augenschein genommene Abbildung zum Beispiel Würfel und Bälle werden die Grundformen Kugel und Quader ohne Probleme von den Augen und dem Gehirn segmentiert. Tiefensehen oder die Erkennung zusammengehöriger Flächen und Objekte sind jedoch Fähigkeiten, an denen es einem Computer mangelt. Bilder sind für einen Rechner nur zweidimensionale Matrizen aus aneinander gereihten, je nach verwendetem Farbraum oder Farbmodell definierten, Werten, die die Farben und Helligkeit der Bildpunkte vertreten. Demnach sucht man im Bereich der Computergrafik und der Bildverarbeitung Mittel und Wege, dem Rechner durch geeignete Methoden Objekterkennung und –wiederfindung zugänglich zu machen.

Ein typischer und wichtiger Vorgang der Bildsegmentierung ist die Detektion von Kanten. Anhand dieser kann ein Objekt besser „eingegrenzt“ werden und somit sind Kanten dementsprechend bedeutende Informationsträger. Kanten liegen vor, wenn es zu einem plötzlichen Grauwert-Wechsel benachbarter Pixel, oder weiträumiger betrachtet von homogenen Flächen, kommt. Demnach deuten auch Wechsel von Farben oder Texturen auf Kanten hin. Im Folgenden werden jedoch die Methoden auf Grauwert-Bilder bezogen.

Explizite Funktionen zur Verarbeitung von Farbbildern arbeiten ähnlich und betrachten zur Kantendetektion jeden der drei Farbkanäle Rot, Grün und Blau separat. Anschließend werden Kombinationsoperatoren verwendet, um die Einzelresultate der Farbebenen zusammenzufassen und auszuwerten.

Solche Kombinationsoperatoren sind Normen zur Bestimmung des Farbabstands, wie zum Beispiel folgende:

- $L_1$–Norm (Summe der Beträge)
- $L_2$–Norm (Euklidischer Abstand)
- $L_\infty$–Norm (Maximum der Beträge)

Erwähnt seien die Untersuchungen von Novak und Shafer [NoS87], die den streng mathematischen Canny-Operator, einen Kantendetektor, mit Kombinationsoperatoren in verschiedenen Varianten und demzufolge mit unterschiedlichen Ergebnissen getestet haben.

Allgemein ausgedrückt erfolgt die Kantendetektion in zwei ineinander übergehenden Schritten. Diese sind:

1. Kantenerkennung
2. Erstellung eines Kantenbildes zur weiteren Verarbeitung

Zudem ist man bei sämtlichen Typen der Kantendetektion darauf bedacht, möglichst fehlerfrei zu arbeiten. Dies bedeutet im Hinblick auf die Erkennung, dass tatsächliche Kantenpunkte auch als solche erkannt und Rauschen in einem Bild nicht fälschlicherweise als Kantenpunkte interpretiert werden. Außerdem sollte versucht werden, Kanten an ihrem ursprünglichen Ort zu erhalten und nicht durch die Erkennungsverfahren zu verschieben.
Zusätzlich ist es zur Gewinnung weiterer Informationen ausreichend, wenn nicht sogar erwünscht, bestehende Kanten nur einmal zu registrieren. Mehrmalige Erkennung äußert sich in Kanten mit der Breite mehrerer Pixel. Demnach sollten Kanten durch verwendete Verfahren nicht unbedingt verbreitert, sondern möglichst dünn gehalten werden.
Abschließend wird versucht, lückenhafte Konturzüge, wenn möglich, zu einer geschlossenen Form zu vervollständigen.
Es haben sich für die Detektion von Kanten vier Hauptschritte herauskristallisiert, die in folgender Reihenfolge mit Auswahl geeigneter Methoden ausgeführt werden sollten:

1. Glättung
2. Kantenerkennung
3. Konturaufbesserung
4. Konturpunktverkettung

Da sich für eine weitere Verarbeitung und Analyse, wie bereits erwähnt, Kantenbilder mit mehrmals registrierten Grauwert-Wechseln in der Gestalt mehrerer Pixel breiter Linien als ungeeignet erweisen, erfolgt meistens nach dem Algorithmus der Kantendetektion eine Konturaufbesserung. Die später beschriebene Non-Maxima-Suppression ist eine solche Methode zur Aufbesserung. Als weiteren Schritt plant man üblicherweise Methoden zur Linienergänzung, Konturschließung sowie der Eliminierung kurzer Kanten ein, die eng beieinander liegende Linienzüge, wenn möglich, aneinander knüpfen und somit die Gesamtkontur erstellen. Abb. 3.32 bis Abb. 3.36 dokumentieren den ganzen

Vorgang zur Verdeutlichung der einzelnen Schritte anhand eines einfachen Grauwert-Bildes mit leichtem Rauschen. Dies soll die gewünschten Resultate der einzelnen Methoden verdeutlichen.

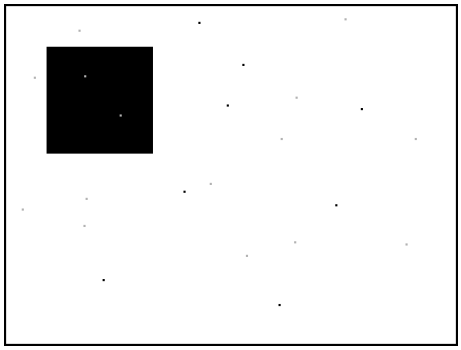


**Abb. 3.32: Originalbild mit leichtem Rauschen**

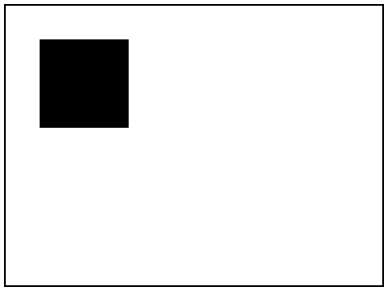


**Abb. 3.33: Glättung des Bildes; Rauschunterdrückung**

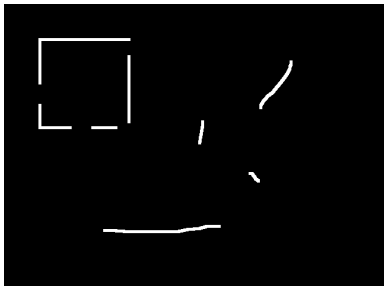


**Abb. 3.34: Kantendetektion**

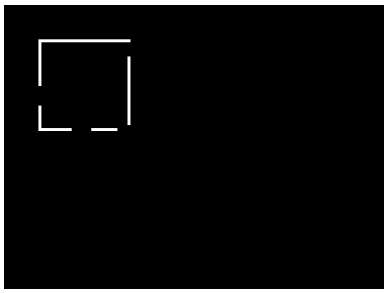


**Abb. 3.35: Konturaufbesserung**

**Abb. 3.36: Konturpunktverkettung**

Bereits vorgestellte Filter zur Rauschunterdrückung bewirken gleichzeitig, je nach Wahl des Filters, eine mehr oder weniger effiziente Glättung des Ausgangsbildes. Diese Glättung begründet sich in der Einbeziehung der Nachbarpixel bei der Berechnung, wodurch, zusätzlich zur Rauschunterdrückung, weiche Übergänge entstehen. Vorherig präsentierte Abbildungen der Rauschunterdrückung haben dies bereits gezeigt.
Auch nachfolgend erörterte Operatoren zur Ermittlung von Kanten bewirken ab einer Größe von drei mal drei Feldern zusätzlich eine simultane Glättung des Bildsignals.

In der Literatur kann man verschiedene Typen der Kantendetektion wieder finden. Dies sind vornehmlich:

- diskrete Ableitungen der Intensitätsfunktionen
- Kantendetektor-Filter (Masken)
- Anpassung von parametrischen Modellen der Kanten an Bildfunktionen
- oder aber auch Kombinationen aller vorherig genannten Methoden.

Die Grundidee zur Ermittlung von Kanten ist die Findung von Diskontinuitäten innerhalb einer Intensitätsfunktion. Diese Funktion repräsentiert dabei die Auffassung eines zweidimensionalen Bildes als mathematische Funktion im Wertebereich *R* (s. Abb. 3.37). Abbildungen 3.40 und 3.41 stellen einen solchen Verlauf der Intensitätsfunktion dar. Die erste Grafik deutet dabei auf eine perfekte Kante hin, wie sie so höchstwahrscheinlich selten in einem digitalen Bild vorkommen wird. Dazu ist die Stelle farblich markiert, an der ein idealer Kantendetektor wirksam werden sollte. Die zweite Abbildung zeigt eine weitaus realistischere Kantendarstellung.
Über die Berechnung der Gradienten ist eine Bestimmung der Kanten, welche sich in starken Grauwert-Wechseln ausdrücken, möglich. Der durch die Formel in Abb. 3.39 errechnete Gradienten-Vektor zeigt dabei in Richtung der stärksten Grauwert-Änderung und gibt zudem die zu der Größe des Vektorbetrags proportionale Stärke der Kante wieder.

$$f(x,y): R^2 \rightarrow R$$

**Abb. 3.37: Abbildungsbereich**

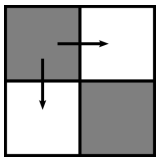


**Abb. 3.38: Richtung des Gradienten**

$$\nabla f(x,y) = \left(f_x(x,y), f_y(x,y)\right)^T = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right)^T$$

**Abb. 3.39: Berechnungsvorschrift des Gradienten-Vektors**

Das Augenmerk soll in dieser Arbeit primär auf die Funktionsweise der Filter-Masken als Kantendetektoren gerichtet werden. Aus diesem Grund werden vorangehend die Bedeutungen der diskreten Ableitungen nur kurz umrissen.
Grauwert-Kanten können wie bereits erwähnt in einer Intensitätsfunktion durch ihre diskreten Ableitungen erkannt werden. Die abrupten Wechsel der Grauwerte und die Diskontinuitäten äußern sich zum einen in den Maxima der ersten Funktionsableitung und in den Nullpunkten der zweiten Ableitung (Abb. 3.42). Den Zusammenhang sollen die Grafiken in Abbildung 3.43 nochmals verdeutlichen, in deren Bildern die Funktion und Ableitungsfunktionen 1. Grades eingezeichnet sind.

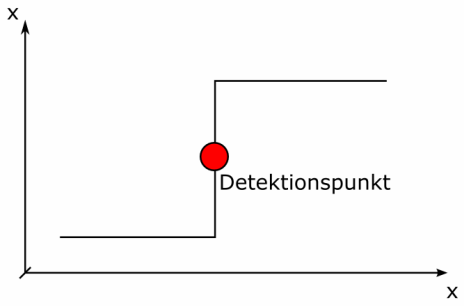



**Abb. 3.40: perfekte Kante mit optimalem Einstiegspunkt der Kantenerkennung**

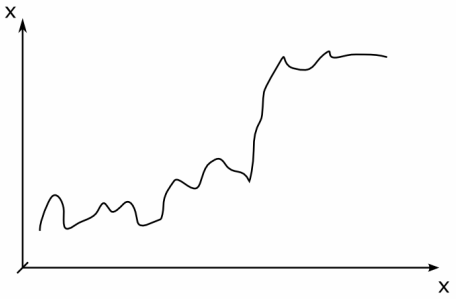



**Abb. 3.41: realistische Kante im Bild**

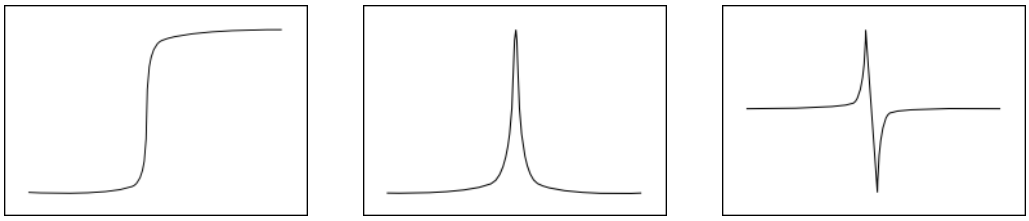


**Abb. 3.42: Grauwert-Intensitätsfunktion, Ableitung 1. Grades und Ableitung 2. Grades (von links nach rechts)**

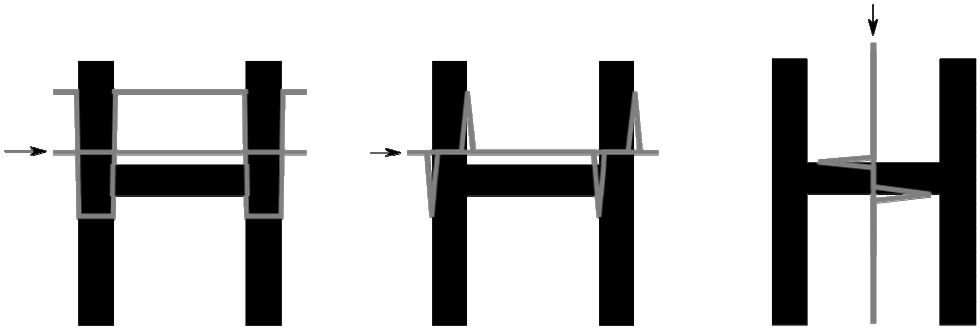


**Abb. 3.43: Intensitätsfunktion, 1. Ableitung horizontal und 1. Ableitung vertikal des H-förmigen Objektes an markierter Stelle (von links nach rechts)**

Üblicherweise beschreiben Differenzengleichungen die Ableitungen einer Funktion. Aufgrund der zwei Dimensionen eines Bildes kann man schließlich zwei Differenzen für jeden einzelnen Bildpunkt berechnen. Dabei sind die Differenzen $g_x$ (s. Abb. 3.44) und $g_y$ (s. Abb. 3.45) die diskrete Ableitung in x- bzw. y-Richtung. Zusammengefasst ergeben sie den Gradientenbetrag *G(x,y)* (s. Abb. 3.46) eines Pixels. Zusätzlich lässt sich die Richtung *R* für die stärkste Wert-Änderung mit Hilfe der entsprechenden Formel, dargestellt in Abb. 3.47, berechnen. Diese Werte werden für jedes Pixel ermittelt und zugeordnet.

$$\frac{\partial f(x,y)}{\partial x} \approx g_x = \frac{f(x,y) - f(x-\Delta x, y)}{\Delta x}$$

**Abb. 3.44: horizontale Differenz**

$$\frac{\partial f(x,y)}{\partial y} \approx g_y = \frac{f(x,y) - f(x,y-\Delta y)}{\Delta y}$$

**Abb. 3.45: vertikale Differenz**

$$G(x,y)=\sqrt{g_x^2+g_y^2}$$

**Abb. 3.46: Gradientenbetrag**

$$R=\Phi(x,y)=\arctan\frac{g_x}{g_y}$$

**Abb. 3.47: Gradientenrichtung**

Bekannte und verbreitete Kantendetektoren sind:

- Sobel-Filter
- Prewitt-Filter
- LaPlace-Filter
- Roberts-Filter
- Kirsch-Filter

Beispielhaft für einen Differenzenoperator erster Ableitung wird der Sobel-Filter und als Beispiel zweiter Ableitung der LaPlace-Filter ausführlich vorgestellt. Aus Gründen der Vollständigkeit sollen jedoch von allen aufgelisteten Kantendetektions-Filtern die Masken grafisch dokumentiert werden. Weiterführende Literatur ist im *Kapitel 8 Literaturverzeichnis* zu finden.

$$\begin{pmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{pmatrix} \qquad \begin{pmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{pmatrix}$$

**Abb. 3.48: Prewitt-Operator; links Gradientenermittlung in horizontaler Richtung, rechts in vertikaler Richtung**

$$\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix} \qquad \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$$

**Abb. 3.49: Roberts-Operator; Differenzen $g_x$ (li.) und $g_y$ (re.) werden über Kreuz berechnet**

$$\begin{pmatrix}5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{pmatrix} \quad \begin{pmatrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{pmatrix} \quad \begin{pmatrix}5 & -3 & -3\\ 5 & 0 & -3\\ 5 & -3 & -3\end{pmatrix} \quad \begin{pmatrix}-3 & -3 & -3\\ 5 & 0 & -3\\ 5 & 5 & -3\end{pmatrix}$$

**Abb. 3.50: Kirsch-Operator; 8 Gradientenberechnungen im Winkelabstand von 45°; abgebildet sind nur 4 der 8 Masken; weitere Masken analog**

#### 3.4.2.1 Sobel-Operator

Bei dem Sobel-Operator handelt es sich um einen weit verbreiteten, einfach zu implementierenden Kantendetektions-Algorithmus, der sehr gute Ergebnisse liefert. Der Sobel-Operator approximiert die erste Ableitung der Funktion eines Bildes. Bei Anwendung des Sobel-Filters erfolgen eine gleichzeitige Glättung und die Unterdrückung von Ausreißern. Diese Glättung wird dabei rechtwinklig zur Bildung der Differenzen ausgeführt. Definiert als Maske besitzt der Sobel-Filter den in Abbildung 3.51 präsentierten Aufbau. Abhängig von der Literatur kann man auch um 180° -gedrehte Darstellungen der Filter antreffen.
Anhand der ersten Filter-Maske wird die zentrierte Differenz $g_x$ in x-Richtung, mit Hilfe der zweiten Maske demzufolge die Differenz $g_y$ in y-Richtung errechnet, mit denen sich nach den Berechnungsvorschriften im *Kapitel 3.4.2 Kantendetektion* der Gradientenbetrag und dessen Richtung ermitteln lassen.

Das zu untersuchende Bild wird demnach von zwei Filtern durchlaufen und es werden für jeden Bildpunkt zwei Koeffizienten berechnet. Durch diese Koeffizienten können schließlich Gradientenrichtung und Stärke ermittelt werden. Vorteilhaft beim Sobel ist die schnelle Umrechnung der Koeffizienten in den Betrag (s. Abb. 3.52) und die Richtung (s. Abb. 3.53) des Gradienten. Um weniger wichtige Informationsträger, wie etwa sehr feine und dezente Kanten, zu entfernen, folgt häufig ein Schwellwertverfahren, das die fein strukturierten Kanten ausfiltert und verwirft.

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

**Abb. 3.51: horizontaler (li.) und vertikaler (re.) Sobel-Operator**

$$G(x,y) = \sqrt{S_x^2 + S_y^2}$$

**Abb. 3.52: Gradientenbetrag des Sobel-Operators**

$$R = \Phi(x,y) = \arctan 2(S_y, S_x)$$

**Abb. 3.53: Gradientenrichtung des Sobel-Operators**

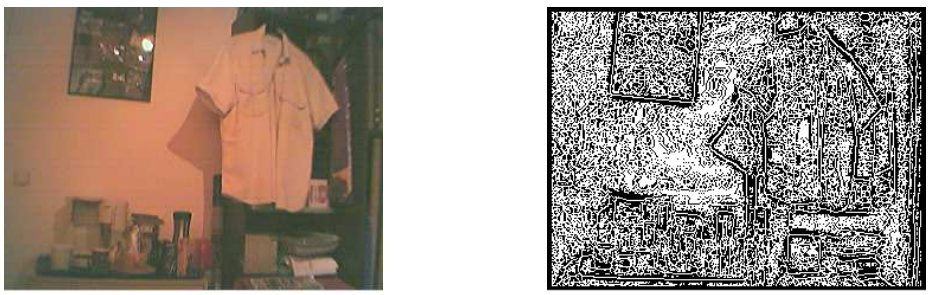


**Abb. 3.54: Originalbild (li.) und Sobel-Operator ohne Schwellwert (re.)**

Viele Operatoren wurden aufgrund der Richtungsabhängigkeit des Sobel-Operators entwickelt. Da er die Differenzen in horizontaler und vertikaler Richtung berechnet, erkennt er dementsprechend nur solch verlaufende Kanten. Um auch mögliche diagonale Kantenzüge zu erkennen, wurde zum Beispiel der Kirsch-Operator entworfen. Die zugehörigen Masken des Kirsch-Operators sind in Abb. 3.50 zu sehen.
Zur Erweiterung des Sobel-Filters auf Farbbilder sind im Skript von Prof. Dr. Dietrich Paulus zur Vorlesung „Bildverarbeitung“ im Wintersemester 2002/2003 Lösungsansätze vorzufinden.

#### 3.4.2.2 LaPlace-Operator

Der LaPlace-Operator, dessen gebräuchlichste Filter-Masken in Abbildung 3.55 dargestellt sind, bewirkt eine Unterdrückung strukturarmer und eine Hervorhebung strukturreicher Bereiche eines Bildes. Der Filter von LaPlace ist wie auf den Abbildungen zu erkennen symmetrischen Aufbaus. Demzufolge ist er separierbar und rotationsstabil. Seine Funktionsweise basiert auf der zweiten Ableitung, welche sich auch aufgrund der Separierbarkeit als zweimalige Ableitung erster Ordnung auffassen lässt. Entsprechende Funktionen zur Ableitung, sowohl in x-Richtung als auch in y-Richtung, können aus den Abbildungen 3.56 und 3.57 entnommen werden. Die resultierende Summe beider diskreten Ableitungen lassen sich zu einer digitalen Umsetzung des LaPlace-Operators (s. Abb. 3.58) zusammenfassen. Demzufolge ergibt sich der bereits angeführte Aufbau der LaPlace Filter-Maske in der linken Grafik der Abbildung3.55.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \qquad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

**Abb. 3.55: Varianten des LaPlace-Operators**

$$g_{xx} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

**Abb. 3.56: horizontale LaPlace-Ableitung**

$$g_{yy} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

**Abb. 3.57: vertikale LaPlace-Ableitung**

$$\begin{aligned} \nabla^2 f(x,y) &= g_{xx} + g_{yy} \\ &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \end{aligned}$$

**Abb. 3.58: Summe der diskreten LaPlace-Ableitungen**

Der Nachteil aller auf der zweiten Ableitung basierenden Faltungsoperatoren und Methoden ist deren Anfälligkeit für Rauschen. Um dem entgegenzuwirken, glättet man das Bild in einem ersten Durchgang mit einem Filter zur Rauschunterdrückung. Man verwendet meistens den Gauss-Filter und findet sogar eine häufig verwendete Kombination dessen mit dem LaPlace-Filter im LaPlacian-of-Gaussian-Operator, auch als Marr-Hilldreth-Operator bekannt. Der theoretische Ansatz basiert auf den ersten Bildverarbeitungsschritten beim Sehen.

### 3.4.2.3 Kantenverdünnung

Ein bekanntes und weit verbreitetes Verfahren zur Verdünnung von Kanten ist die Nicht-Maxima-Unterdrückung, besser unter den englischen Bezeichnungen Non-Maxima-Suppression (NMS) oder Non-Maxima-Absorption (NMA) bekannt. Das Prinzip bildet dabei der Vergleich der quer zum aktuell betrachteten Pixel liegenden Konturpunkte. Genauer formuliert wird die Stärke der Pixel, ausgedrückt durch den errechneten Gradientenbetrag (s. Abb. 3.46), verglichen. Der Maximal-Wert dieser drei Punkte wird übernommen, die restlichen zwei werden durch Setzen der Stärke auf Null unterdrückt. Daher der Begriff Non-Maxima-Unterdrückung.
Damit eindeutig ist, welche zwei Pixel der insgesamt aus acht Pixeln bestehenden Nachbarschaft zum Vergleich mit einbezogen werden, wird die Gradientenrichtung betrachtet. Anhand nachfolgender Tabelle (s. Abb. 3.59), dem Bereichsdiagramm (s. Abb. 3.60) und der entsprechend zugewiesenen Masken in Abbildung 3.61 kann man die Kriterien der Auswahl, welche Pixel zum Vergleich in Frage kommen, nachvollziehen.

| **Maske** | **1. Bereich** | **2. Bereich** |
|---|---|---|
| A | $337{,}5° \le \alpha \le 360°$ *und* $0° \le \alpha < 22{,}5°$ | $157{,}5° \le \alpha < 202{,}5°$ |
| B | $22{,}5° \le \alpha < 67{,}5°$ | $202{,}5° \le \alpha < 247{,}5°$ |
| C | $67{,}5° \le \alpha < 112{,}5°$ | $247{,}5° \le \alpha < 292{,}5°$ |
| D | $112{,}5° \le \alpha < 157{,}5°$ | $292{,}5° \le \alpha < 337{,}5°$ |

**Abb. 3.59: Winkeltabelle der Non-Maxima-Suppression**

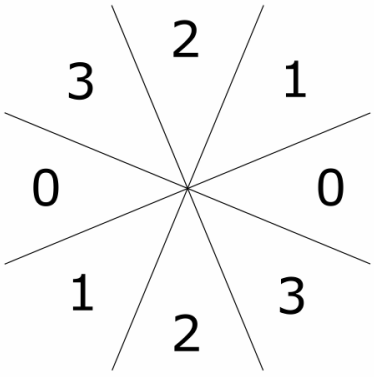



**Abb. 3.60: Bereichsdiagramm der NMS**

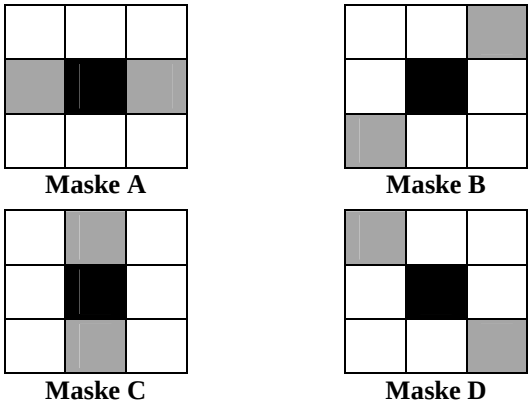



**Abb. 3.61: Masken der zu vergleichenden Konturpunkte; aktuelles Pixel ist schwarz, die zugehörigen Nachbarn sind grau dargestellt**

Sind die entsprechenden Pixel ermittelt, wird die Gradientenrichtung der involvierten benachbarten Bildpunkte untersucht. Diese müssen die gleiche Richtung wie das Ursprungspixel aufweisen, da sie sonst auf eine überschneidende Kante oder eine Biegung der Kante hindeuten. Handelt es sich um ungefähr gleichgerichtete Pixelorientierungen, erfolgt die Endprozedur der Non-Maxima-Suppression, das Setzen der Nicht-Maxima auf Null.

Sollten die gelieferten Ergebnisse nicht gewünschte Resultate ergeben, kann optional noch ein Schwellwert-Verfahren zwischengeschaltet werden, bei dem das Maximum der drei Pixel nochmals mit der Grenze des Schwellwert-Verfahrens verglichen und bei Unterschreitung des Wertes gegebenenfalls auch auf Null gesetzt wird.

#### 3.4.2.4 Kreis-Erkennung mit der Hough-Transformation

Ein nicht selten angewandtes Verfahren zur Findung eines Kreises oder kreisähnlicher Objekte und deren Mittelpunkte ist eine auf der Transformation nach Hough aufbauende Methode. Daher ist die Hough-Transformation in die Kategorie der Objekterkennung einzuordnen.
Die Hough-Transformation wurde ursprünglich zur Linienerkennung eingesetzt, lässt sich aber auf weitere Linienfiguren, zum Beispiel Kreise, erweitern und setzt eine Kantendetektion voraus. Das Prinzip ist die optimale Darstellung von Linienfiguren in einem parametrisierten Raum, auch Akkumulator genannt.
Für Linien definiert sich das Verfahren folgendermaßen: Es wird ein zweidimensionaler Akkumulator (s. Abb. 3.63) mit M Spalten für die Abstände einer Linie zum Ursprung und mit N Zeilen für die Winkel, gemessen im Uhrzeigersinn von der positiven x-Achse (s. Abb. 3.62), angelegt. Die einzelnen Felder des Akkumulators werden mit dem Wert Null initialisiert. Statt Nutzung der üblichen Geradengleichung (s. Abb. 3.64), greift man auf die Hess´sche Normalform (s. Abb. 3.65) zurück. Demzufolge repräsentiert jedes einzelne Feld des Akkumulators eine Linie im Bildraum.

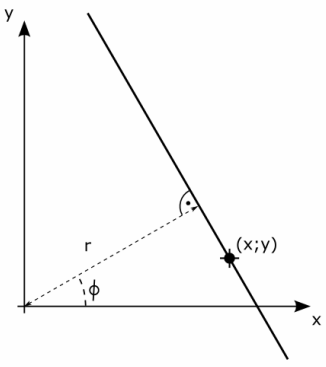



**Abb. 3.62: Linie im Bildraum**

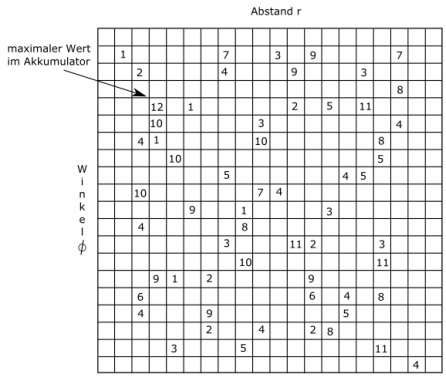



**Abb. 3.63: Auszug aus einem Akkumulator mit Beispieleinträgen zur Verdeutlichung des Aufbaus**

$$y = m \cdot x + b$$

**Abb. 3.64: Geradengleichung**

$$r = x \cdot \cos(\phi) + y \cdot \sin(\phi)$$

**Abb. 3.65: Hess´sche Normalform**

Dann werden die Koordinaten x und y der als Kantenpunkte erkannten und im Kantenbild gesicherten Bildpunkte durch Einsetzen in die Hess´sche Normalform in den Abstand *r* und den Winkel *Φ* umgerechnet. Das den ermittelten Werten entsprechende Feld im Akkumulator wird um „1“ inkrementiert. Die Felder mit den höchsten Werten im

Akkumulator, nach Behandlung aller zu berücksichtigenden Pixel, sind dann markante Linien im Kantenbild.

$$(x-a)^2 + (y-b)^2 = R^2$$

**Abb. 3.66: Kreisformel**

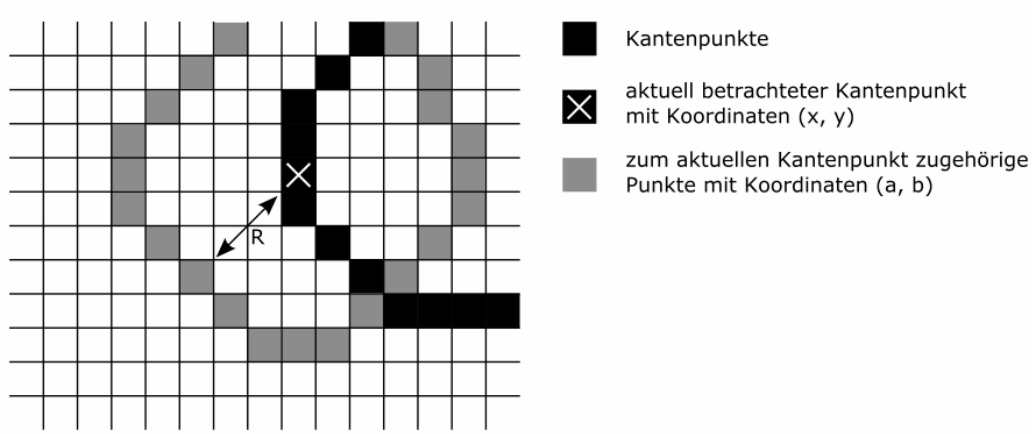



**Abb. 3.67: Bildausschnitt zur Verdeutlichung der Berechnung zugehöriger Pixel eines Kantenpunktes anhand der Kreisformel auf Pixelebene**

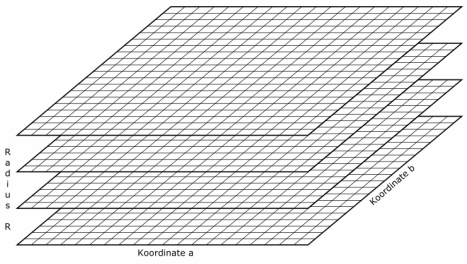



**Abb. 3.68: Parameterraum für Kreisdetektion mit unbekanntem Radius**

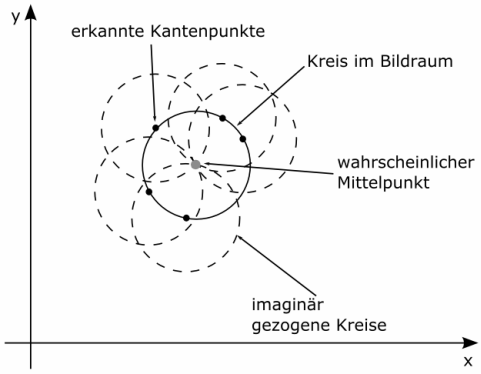



**Abb. 3.69: Illustration zur Ermittlung des Kreismittelpunktes**

Die Erweiterung zur Erkennung von komplexeren Linienfiguren erfolgt analog. Es wird jeweils ein Akkumulatorraum angelegt und eine geeignete Parameterdarstellung ausgewählt. Zur Detektion von Kreisen benötigt man je nach Anforderung zwei oder drei Dimensionen im Akkumulator. Ist der Radius des Kreises bekannt, ist ein zweidimensionaler Parameterraum für die Punktkoordinaten *a* und *b* ausreichend. Andernfalls benötigt man noch eine dritte Dimension für variable Kreisradien *R* (s. Abb. 3.68). Ausgangspunkt ist selbstverständlich das Kantenbild. Um jeden einzelnen Kanten-Bildpunkt mit den Koordinaten (*x*, *y*) des Kantenbildes wird ein imaginärer Kreis mit Radius *R* gezogen. Bei unbekanntem Radius sind dies mehrere Kreise mit Radien von $R_{Min}$ bis $R_{Max}$, näheres dazu im *Kapitel 5 Implementierung*. Nach der Kreis-Formel (s. Abb. 3.66) lassen sich die Punkte mit den Koordinaten (*a*, *b*) auf diesem, um den Kantenpunkt gezogenen Kreis berechnen (s. Abb. 3.67). Die zugehörigen Felder im Akkumulator werden dementsprechend um „1“ inkrementiert. Um Kantenpunkte herum

aufgespannte Kreise mit dem gleichen Radius des gesuchten Kreises schneiden sich folglich genau in dessen Mittelpunkt (s. Abb. 3.69). Diese Tatsache setzt die Hough-Transformation für Kreise um. Das Feld mit dem höchsten Eintrag im Akkumulator ist mit größter Wahrscheinlichkeit der Mittelpunkt des gesuchten Kreises.

### 3.4.3 Regionen-Segmentierungsverfahren

Eine Möglichkeit auf höherer Ebene eine Objekterkennung auszuführen, ist die Suche nach definierten Regionen. Segmente von zusammenhängenden Pixeln mit ähnlichen Eigenschaften können in Regionen, manchmal auch Blobs[21] genannt, zusammengefasst werden. Die Begriffe Blobs und Regionen sind nicht klar definiert und unterscheiden sich je nach Literatur. Eingebürgert hat sich als Definition von Blobs, die Zusammenfassung mehrerer gleichartiger Pixel in beliebiger Form, während Regionen mit Bounding Boxen verglichen werden und auch mehrere Blobs umschließen können. Kriterien zur Gruppierung mehrerer Pixel zu einer Region oder Blobs und deren Suche sind zum Beispiel Farbton oder Grauwerte (Helligkeit). Auch die Anzahl der Pixel eines Segments kann als Suchkriterium dienen. Erkannte Blobs und Regionen können hinterher mit einer Datenbank von bereits erfassten Objekten verglichen und gegebenenfalls als solche identifiziert werden.

Ein sehr einfaches Verfahren zur Zusammenfassung von mehreren Pixeln zu Regionen ist das Vereinen der Pixel anhand ihrer Werte (s. Abb. 3.70 und Abb. 3.71 ). Man legt eine allgemeingültige Maximal-Differenz der benachbarten Pixel an und vergleicht diese. Ist die Differenz der Pixel kleiner als die Maximal-Differenz, werden sie zu einer Region verbunden. Diese Vorgehensweise wird auch einfache Verkettung genannt. Ein beständiger Nachteil ist die schnelle Verbindung zweier Regionen, die zum Beispiel über einen Farbverlauf langsam ineinander übergehen und so die Region eine große Fläche des Bildes einschließt. Obwohl das Verfahren schnell und einfach ist, ist vom Gebrauch für komplexe Szenen abzuraten.

| | | | |
|---|---|---|---|
| 10 | 11 | 12 | 11 |
| 111 | 112 | 30 | 34 |
| 109 | 115 | 37 | 35 |
| 109 | 111 | 36 | 35 |

**Abb. 3.70: Pixelwerte eines kleinen Grauwertbildes**

| | | | |
|---|---|---|---|
| 10 | 11 | 12 | 11 |
| 111 | 112 | 30 | 34 |
| 109 | 115 | 37 | 35 |
| 109 | 111 | 36 | 35 |

**Abb. 3.71: zu Regionen zusammengefasste Pixel**

Eine weiterführende Methode bietet das Region-Growing-Verfahren. Das Bild wird in kleine Zellen der Größe 1x1 bis 4x4 Pixel aufgeteilt und für jede Zelle die durchschnittliche Intensität oder Helligkeit bestimmt. Standardgemäß beginnt man in der

[21] dt.: Klumpen

linken oberen Ecke des Bildes mit der ersten Zelle und vergleicht deren Wert mit den Durchschnittswerten der Nachbarn. Liegen die Differenzen der Nachbarn im akzeptablen Bereich, werden die entsprechenden Zellen vereinigt. Die Zusammenfügung abschließend, wird der neue Durchschnittswert berechnet und man verarbeitet die nächsten in Frage kommenden Nachbarn. Dies geschieht, bis keine Zellen aufgrund großer Differenzen mehr angefügt werden können und man beginnt die Prozedur mit einer noch nicht behandelten Zelle von vorne.

### 3.4.4 „Color Indexing“

Zu den ebenfalls häufig eingesetzten Methoden zur Findung eines Objektes im Bild gehören die histogrammbasierten Verfahren. 1991 wurde das Verfahren mit dem Namen „Color Indexing“ von Michael .J. Swain und Dana H. Ballard entworfen. Wie der Name schon verrät, handelt es sich bei „Color Indexing“ um ein Erkennungsverfahren auf Basis von Farbwerten. Aufgrund der Tatsache, dass diese Art der Erkennung unabhängig von Kanten und Formen ist, gestaltet sich diese Methode als rotations- und translationsrobust. Auch Verdeckungen durch andere Objekte lassen sich größtenteils abfangen. Aufgrund seiner Einfachheit und der unkomplizierte Implementierung hat dieses Verfahren sich zu einer Standardmethode der Objekterkennung etabliert.

Die „Histogram Intersection“, ein zum „Color Indexing“ zuzuordnender Algorithmus, bildet den Ausgangspunkt einiger weiterentwickelter Ansätze histogrammbasierter Verfahren.
Prinzipiell wird ein Objekt bei dieser Art der Objekterkennung anhand seines Histogramms, gespeichert in einer Objekt-Datenbank, identifiziert.
Eine Erstellung des zu vergleichenden Objekt-Histogramms erfolgt unter Annahme der ungefähren Position des zu identifizierenden Objektes im Bild. Bei der Erstellung des Histogramms innerhalb der bekannten Position fließt zwar der Umgebungshintergrund geringfügig mit ein, spielt aber keine sonderlich beeinflussende Rolle. Dieses neu erstellte Histogramm wird anschließend mit einer Datenbank bestehender Histogramme von Objekten verglichen. Als Maß der Ähnlichkeit dient der Wert $V$ (s. Abb. 3.72). Die Berechnung der Maßzahl $V$ erfolgt über das Histogramm $H_O$ des gefundenen Objektes im Bild und anhand des Histogramms $H_D$ des Objektes in der Datenbank.
Um die Größe an Daten eines Histogramms gering zu halten, werden nebeneinander liegende Farben meist zu einer zusammengefasst. Dies reduziert die Speichergröße der Histogramme von $n = 256^3 = 16.777.216$ Einträgen bei 8 Bit Tiefe für den RGB-Farbraum auf $n = (256/u)^3$ Einträge, wobei $u$ die Anzahl der angestrebten Einteilung verkörpert. Bildlich kann man sich die Unterteilung des RGB-Würfels aus Abbildung 3.2 zum Beispiel für $u = 4$ in 4 x 4 x 4 Quader, im Englischen vornehmlich „Bins[22]“ genannt, vorstellen. Formal werden mehrere Farben zu einem Index zusammengefasst. Daher die Herleitung des Begriffes „Color Indexing“.

[22] dt.: Behälter

$$V(H_D, H_O) = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\sum_{k=0}^{m-1} \min(H_D(i,j,k), H_O(i,j,k))}{\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\sum_{k=0}^{m-1} H_O(i,j,k)}$$

**Abb. 3.72: Berechnung der Maßzahl V mit *m = n/3* Einträgen pro RGB-Kanal**

Des Weiteren bietet sich die Möglichkeit durch eine Histogramm-Rückprojektion Objekte in einem Bild zu detektieren. Ziel der Histogramm-Rückprojektion ist die Unterdrückung der nicht im Objekt vorhandenen Werte und das Hervorheben des gesuchten Objektes. Zuerst wird ein Verhältnis-Histogramm $H_R$, in der Literatur „Ratio Histogram“ genannt, nach Vorschrift in Abbildung 3.73 aus dem Histogramm des zu analysierenden Gesamtbildes $H_B$ und dem Histogramm $H_O$ des möglichst isoliert vorhandenen Objektes erstellt. Die Darstellung der Szene und die des Objektes müssen zwar nicht die gleiche Bildabmessung, aber eine einheitliche Quantisierung des Farbraumes aufweisen. Das Verhältnis-Histogramm wird anschließend rückprojiziert, was zur Ausschließung nicht im Objekt vorhandener Farben führt.

$$H_R = \frac{H_O}{H_B} \qquad \text{mit} \qquad \text{für } 1 \le l \le N$$

$$H_R = [H_{Rl}]$$
$$H_O = [H_{Ol}] \qquad \text{wobei } N \text{ Gesamtanzahl der „Bins“ des Farbraums}$$
$$H_B = [H_{Bl}]$$

$$H_{Rl} = \begin{cases} 0, wenn\ H_{Bl} = 0 \\ \min\left(\frac{H_{Ol}}{H_{Bl}}, 1\right) sonst \end{cases}$$

**Abb. 3.73: Berechnungsvorschrift der Histogramm-Rückprojektion**

Größter Nachteil des Verfahrens „Color Indexing“ nach Swain und Ballard ist seine Anfälligkeit für Helligkeits- und daraus resultierenden Farbänderungen, die sich global auswirken. Des Weiteren sollte für eine einwandfreie Objekterkennung die Farbe des zu verfolgenden Objektes nicht als sonderlich große homogene Fläche im Hintergrund vorkommen. Zudem stellt die Wahl der richtigen Anzahl an „Bins“ ein häufiges Problem dar, zumal eine zu niedrige Anzahl ein großes Farbspektrum glättet, während eine relativ hohe Einteilung sich in einer Art Rauschen äußert [RMG98].

## 3.5 Bewegungserkennung

Man ist bei der Bewegungserkennung stets bemüht, so genannte Trajektoren, „Spuren“ oder „Bahnen“ von Bewegung im zwei- und dreidimensionalen Raum, zu berechnen. Demnach wird versucht, die Position und Orientierung des bewegten Körpers zu jeder Zeit zu kennen. Erstrebenswert ist dabei in verschiedenen Einsatzgebieten die Echtzeit-Fähigkeit und Robustheit entwickelter Systeme. Dies gilt sowohl für den Bereich der Virtuellen Realität wie auch der Bildverarbeitung.

### 3.5.1 Tracking

Durchaus brauchbarere Methoden der Bewegungserkennung werden durch Tracking-Systeme realisiert. Diese besitzen stationäre Sensoren verschiedenster Bauweisen und mit unterschiedlichen Übertragungsmethoden. Es existieren unter anderem mechanische, optische, akustische und elektro-magnetische Trackingsysteme, die ihre bevorzugte Verwendung im Gebiet der „Virtuellen und der Erweiterten Realität“ finden. Mithilfe der stationären Sensoren können die aktuelle Orientierung und Position des Benutzers in Echtzeit ermittelt werden.
Obwohl es sicherlich geeignete Trackingsysteme zur Umsetzung des angestrebten Systems gibt, sprechen, wie bereits im vorherigen *Kapitel 2.1* angeführt, die Kosten, die Handhabung und die Größe der Tracking-Geräte gegen deren Einsatz innerhalb der formulierten Aufgabenstellung. Nachteilig wirken sich zudem behindernde Kabel oder unbequeme und somit leistungshemmende Konstruktionen aus.

### 3.5.2 Bildverarbeitungsbasierte Methoden

Existierende Verfahren der Bewegungserkennung, basierend auf Bildverarbeitungs-methoden, bauen primär auf der Objekterkennung auf. Um eine Bewegung (Rotation, Translation oder Skalierung) zu registrieren, muss zuerst das in Bewegung befindliche Objekt oder Hinweise auf dessen Bewegung gefunden beziehungsweise erkannt werden.

Durch Auswertung der Analysen bewegter Objekte in einer Bildfolge können zusätzlich Informationen über Position, Geschwindigkeit und Beschleunigung geliefert werden. Obwohl nach [Pau01] (S.164) Objekte in Bildfolgen stabiler zu erkennen sind, sofern die Positionsänderung zwischen zwei Bildern nur gering verläuft, ist die Erkennung aufwändiger. Dieser Aufwand ergibt sich aus der Verarbeitung und Analyse einer Bilderfolge statt einer Einzelaufnahme. Eine besondere Herausforderung bieten zudem elastische Gegenstände, die im zeitlichen Ablauf ihre Form ändern.

Die Medien zur Bewegungserkennung bilden zweifelsfrei über Kamera aufgenommene Bildfolgen. Dabei kann es sich um Live-Streams oder bereits aufgezeichnete und vorgefertigte Videos handeln. Gegenstände oder Personen, die während dieser Bildfolge ihre Position ändern, sind bewegte Objekte innerhalb einer Szene. Um diese zu erkennen, gibt es verschiedene Ansätze, die je nach Aufgabenstellung unterschiedliche Anforderungen zu erfüllen haben und demnach ihren Einsatz finden. Abhängig von der Komplexität und der Struktur des zu registrierenden Objektes, werden entsprechend aufwändigere Methoden zur Erkennung benötigt. Auch die Frage nach Echtzeitfähigkeit spielt eine wichtige Rolle bei der Konzipierung eines zu entwickelnden Systems. Vorrangige Kriterien für eine Suche erkennbarer Objekte sind Kontur, Farbe oder Textur, wie bereits in *Kapitel 3.4 Objekterkennung* erläutert.

Eine oftmals angewandte Methode zur Bewegungserkennung ist das Vergleichen des aktuellen Bildes mit einem Referenzbild. In diesem Sinne handelt es sich bei Referenzbildern um Aufnahmen des Hintergrundes, wenn gefordert ohne die zu erkennenden Objekte. Das momentan bearbeitete Bild wird mit dem Referenzbild unter Verwendung einer mathematischen Operation, bevorzugt Subtraktion und Division, verrechnet. Sollte im aktuellen Bild die Anzahl der veränderten Pixel, zum Beispiel durch Differenzen der Farbwerte oder der Intensität, eine vereinbarte Schwelle überschreiten, kann von einer Bewegung ausgegangen werden. Der Vergleich verläuft folglich auf Pixelebene.
Diese Art der Bewegungsdetektion kann in verschiedenen Varianten implementiert werden. Zum einen ist es möglich Bewegungen im Blickwinkel der Kamera allgemein zu registrieren. Sobald eine Veränderung im Bild bemerkt wird, gilt die Bewegung als initiiert. Das Prinzip wird häufig für generelle Überwachungsmechanismen verwendet, in der eine Bewegung einen Prozess, beispielsweise das Aufnehmen des folgenden Live-Streams auf Festplatte, starten soll. Zum anderen kann man das Bild in Regionen einteilen, um die Bewegung genauer zu verfolgen. So ist es möglich eine Bewegungstendenz über mehrere Bilder zu beobachten und auszuwerten. Bei diesem Prinzip wird deutlich, dass nicht primär nach einem Objekt, sondern eher nach der Spur der Bewegung gesucht wird.
Problematisch bei dieser Art der Bewegungserkennung ist die Anfälligkeit für Änderungen der Lichtverhältnisse oder Selbstkalibrierungen von Kameras. Aufgrund der gleichzeitigen Helligkeitsänderung mehrerer Pixel wird häufig eine nicht vorhandene Bewegung erkannt. Die daraus resultierende Fehleranfälligkeit spricht gegen die Verwendung im Rahmen dieser Diplomarbeit. Eine Verwendung in Räumen mit stetig wechselnden Lichtverhältnissen kann nicht ausgeschlossen und muss deshalb berücksichtigt werden.

**Abb. 3.74: Beispiel Differenzbild durch Division (Null-Division wird abgefangen)**

Eine weitere Möglichkeit der Verfolgung eines Objektes im Bild setzt die Segmentierung des Bildes in Regionen voraus. Positionen zusammengefasster Regionen gleicher Farbe oder Intensität innerhalb eines Bildes können festgelegt und abgespeichert werden. Der Referenzpunkt zur Positionsbestimmung sei beispielsweise der angenommene Schwerpunkt der Region. Anschließend versucht man im Folgebild die Regionen mit gleichen Informationsschemata wieder zu finden und die entsprechende Positionsänderung des Referenzpunktes zu ermitteln.

Die in dem entwickelten System verwendete Erkennung von Positionsänderungen wird mit Hilfe zweier Schritte mit Einsatz von Bildverarbeitungsmethoden bewerkstelligt. Der erste Schritt besteht in der vorgestellten Methode der Kreis-Erkennung durch Anwendung der Hough-Transformation. Die zweite Methode wertet die im zeitlichen Abstand gewonnenen Mittelpunkte geeignet aus. Vorteilig wirkt sich die konstante Form des verwendeten und zu verfolgenden Markers aus, die die Objektverfolgung durch komplexe Farboperationen nicht notwendig werden lässt. Die einfache Kontur des Markers, näher in *Kapitel 5.1 Der Marker* beschrieben, lässt auf gute Erkennungswerte hoffen. In der Evaluierung wird diese Annahme näher beleuchtet. Des Weiteren erfolgt in späteren Kapiteln eine detaillierte Beschreibung der einzelnen Vorbereitungs- und Verarbeitungsschritte.

# 4 Systeme der Gestenerkennung

Auf dem Titel der Diplomarbeit aufbauend, erfolgt eine kurze Definition von Gesten. Was genau sind Gesten und welche Art von Gesten eignet sich vorrangig für das angestrebte Projekt?
Dem folgend werden beispielhaft grundlegende Ideen und eventuelle Umsetzungen dieser in bereits vorhandenen Gestenerkennungssystemen aufgezeigt. Die Art der Gesten und der verwendeten Geräte spielt dabei keine besondere Rolle. Es soll vielmehr ein Überblick über Möglichkeiten, Ergebnisse und Probleme der Gestenerkennung im Allgemeinen gegeben werden.

## 4.1 Definition: Gesten

Der Begriff der Geste bezeichnet in seinem häufigsten Gebrauch ein großes Spektrum von Körperbewegungen und Abläufen zur Übermittlung von Gesinnung und Informationen, je nach Geste auch eine momentane Gefühlslage. Sie sind ein Mittel der Kommunikation. Das freundliche Winken zu seinem Gegenüber oder das weniger höfliche „Vogelzeigen“ demonstrieren ohne Worte die momentane Einstellung des Akteurs zu seinem Mitmenschen. Das Zunicken oder Zeigen in eine bestimmte Richtung mit Hilfe des Zeigefingers sind Beispiele schneller nonverbaler Kommunikation unter Verwendung einfachster Gesten zur Informationsübermittlung.
Die Anwendung von Gesten ist fast überall auf der Welt bekannt, weist jedoch kulturell unterschiedliche Bedeutungen auf. Ebenso gelten Unterscheidungen im soziologischen und biologischen Bereich.

Häufig werden Gesten als Steuerzeichen organisierter Einrichtungen verwendet. Zum Verwendungsbereich gehören unter anderem Flughäfen, Militär und Polizei. Kurzgehaltene Kommandos in Form bestimmter Handstellungen oder Bewegungen verhelfen Befehle effizient weiterzuleiten.
In der Schauspielerei werden Gesten oftmals als Untermalung wichtiger Kernaussagen oder generell als Betonung bedeutungsvoller Punkte verwendet. Dabei kann es sich um einen erhobenen Zeigefinger zur Demonstration von Härte oder auch um das Legen beider Hände auf die Brust zur Signalisierung einer romantischen Gefühlsregung handeln. Diese Darstellung kann man unter Anbetracht der Anforderungen der Diplomarbeit getrost außer Acht lassen.
Im Bereich der Computergraphik definieren sich Gesten allgemein im Vollziehen einer Bewegung, meistens der Hände oder Finger, häufig unter Zuhilfenahme von Zeigegeräten (Pen tablet), um beliebige Informationen zu übermitteln. Die Auswertung der Informationen und die daraus resultierenden Ergebnisse sind abhängig von der Arbeitsweise und Interpretation der entwickelten Software. Zeichen-Software erkennt mit dem Stift vollführte Gesten auf einem Grafiktablett beispielsweise als geometrische Figuren und diese werden dementsprechend als Objekt auf dem Monitor dargestellt. Der

Internet-Browser Firefox kann mit einem Plugin durch GEsten bedient werden. So öffnet beispielsweise eine Mausgeste der Folge *gedrückte Maustaste, Mausbewegung nach oben, Maustaste loslassen* ein neues Browser-Fenster.

Es gibt verschiedene Ansätze zur Klassifizierung von Gesten. Zur Intensivierung der einzelnen kategorischen Ansätze sei auf [Nes86] verwiesen. Die Ausführungen von Jean-Luc Nespoulous, Paul Perron und André Roch Lecours fassen Gesten in blickwinkelabhängige Modelle zusammen.

Die Idee der Gesten auf Basis geometrischer Formen oder Linienzüge bietet oftmals die Grundlage für weitere Überlegungen. Im Verlauf der Diplomarbeit kristallisierte sich eine Tendenz zu einfachen Linienfiguren heraus. Allerdings wird auf die Erkennung einer Geste durch das entwickelte System erst in *Kapitel 5.3 Prinzip der Gestenerkennung* eingegangen.

Vorteil aller Gesten ist die akustische Unabhängigkeit und die schnelle Informationsübermittlung. Ein Geräuschpegel, der die verbale Kommunikation einschränkt oder unmöglich macht, behindert den Informationsaustausch per Finger- und Handbewegungen in keinster Weise. Dies betont noch einmal die erarbeiteten Anstöße und Denkansätze, die sich bei dem Vergleich der unterschiedlichen Eingabegeräte und deren Vor- und Nachteile ergaben.
Dem entgegen stellt sich die Notwendigkeit des Senders im Sichtbereich des Empfängers positioniert zu sein.

## 4.2 Bestehende Systeme und Ansätze

Ziel der Bemühungen hinsichtlich einer optimalen Steuerung durch Gesten oder durch andere, zum Beispiel in *Kapitel 2.1 Möglichkeiten der Eingabe und Steuerung* erörterte Verfahren ist eine effiziente oder, überspitzt ausgedrückt, perfekte Verbindung des Menschen zum Computer. Human-Computer-Interfaces ist sicherlich ein treffendes Schlagwort. Doch was wie Zukunftsmusik klingt, ist in manchen Ideen schon ansatzweise vorhanden.
Als Beispiel sei das vom Fraunhofer Institut ins Leben gerufene Projekt „inhaus“ genannt. Hierbei handelt es sich um ein Bauprojekt mit dem Gesamtkonzept des „vernetzten Lebens“. Unter anderem sind die elektronischen Einrichtungsgegenstände miteinander verbunden und erlauben das Automatisieren vielerlei Aufgaben im Haushalt. Das Wohlfühlen und die Behaglichkeit des Hausbesitzers sind die vordergründigen Prinzipien des softwaretechnischen Highlights.

Zurzeit oftmals angesteuerte Ziele sind die Interaktion und Manipulation auf 2D-Bildschirmen oder in 3D-Umgebungen. Dies kann die Bearbeitung von Daten, zum Beispiel Videoschnitt oder das Navigieren durch 3D-Welten beinhalten.
Die Möglichkeiten der Gestenzeichnung und der dazugehörigen Erkennung sind vielfältig. Einen Überblick über gestartete und entwickelte Projekte bietet die Seite „The

Gesture Recognition Home Page“ [Coh06a], die jedem Interessierten einen Einstiegs- und Vertiefungspunkt in den Bereich Gesture Recognition offeriert.

Generell kann zwischen dynamischen und statischen Gesten unterschieden werden. Statische Gesten werden anhand eines einzigen Bildes analysiert und interpretiert. Beispiel hierfür wäre die Untersuchung der Handstellung oder der variablen Anzahl hochgehaltener Finger im Bild als Interaktions- oder Manipulationselement. Im Gegensatz dazu verlaufen dynamische Gesten über einen längeren Zeitraum und somit über mehrere Bilder. Dabei kann es sich bei dynamischen Gesten um Körperbewegungen oder die Zeichnung imaginärer Linien und geometrischer Figuren handeln. Die Eingabe einer dynamischen Geste kann über eine Maus, einen Datenhandschuh oder zum Beispiel über visuell gestützte Eingabe, wie die in dieser Diplomarbeit konzipierte Möglichkeit, erfolgen.
Einen detaillierten Überblick über bis dato bekannte Methoden liefert [WH99]

Um einen Auszug der diversen Arten der Gestenerkennung darzulegen, werden folgend drei Verfahren vorgestellt. Diese behandeln vornehmlich die Erkennung dynamischer Gesten.

### 4.2.1 Linienfiguren zur Interaktion

Mit „Specifying Gestures by Example“ hat Dean Rubine [Rub91] einen oft zitierten Beitrag zur Entwicklung interaktiver Gestensteuersysteme geschaffen. In seiner Veröffentlichung geht er dabei auf das Toolkit GRANDMA[23], das eine schnelle und einfache Einbindung von in einem Zug vollführbarer Gesten in entwickelte Anwendungen ermöglicht, detaillierter ein. Von besonderem Interesse ist der integrierte Gestenerkenner, der mit Beispielgesten operiert. Im Gegensatz dazu existiert noch die Möglichkeit der Verwendung beschriebener Gesten, die ohne Trainingsphasen angewandt werden können.
Anhand der gestenbasierten Zeichenanwendung GDP, die lediglich kurz erwähnt sei, wird die Verknüpfung des Toolkits mit einer Applikation verdeutlicht.
Das grundlegende Verfahren von GRANDMA ist für die Handhabung mit einer Maus oder dem Stift eines Grafiktabletts gedacht.

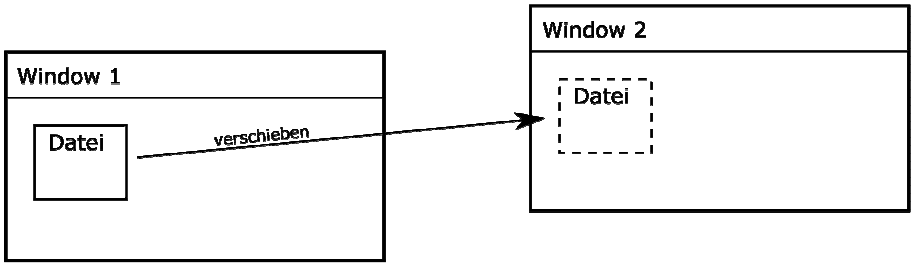



**Abb. 4.1: drei Parameter in einer Geste: die Datei (Objekt) nach Window 2 (Ziel) verschieben (Befehl)**

[23] s. Abkürzungsverzeichnis

Betont wird die Mächtigkeit einer einzigen Geste zur Beschreibung mehrerer Parameter. Zur Veranschaulichung dieser wird das Verschieben eines Textes, vertreten durch drei zu verarbeitende Informationen, angeführt (s. Abb. 4.1), das durch eine ununterbrochene Geste vollzogen wird.
In diesem Konzept der Steuerung werden folglich Gesten mit dem Aspekt der direkten Manipulation von Objekten gekoppelt.

Das Prinzip der Erkennung und Identifikation von Gesten bildet die Berechnung spezieller Merkmale, im weiteren Verlauf Features genannt. Eine zweidimensionale gezeichnete Geste wird wie ein Array *g* von *P* zeitlich markierten Punkten von *0* bis *P-1* interpretiert. Dabei wird bei dicht beieinander liegenden Punkten der später markierte verworfen. Ziel ist es, die gezeichnete Geste in zwei Schritten zu klassifizieren. Erster Schritt beinhaltet das extrahieren von Features und das Zusammenfassen in einem Feature-Vektor $f = [f_1,...,f_F]$, zweiter Schritt ist der Vergleich mit *C* bestehenden definierten Gestenklassen nummeriert von *0* bis $C - 1$. Pro Klasse existieren mehrere Ausführungen einer Geste, um variable Zeichenschwankungen abzudecken (s. Abb. 4.2).

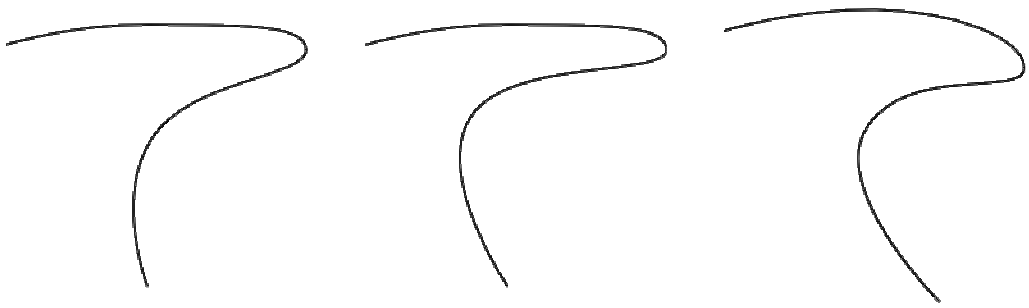
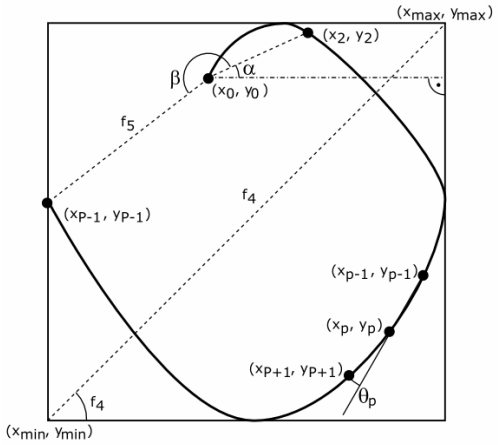


**Abb. 4.2: drei Ausführungen einer Geste**

Die als Kriterien herangezogenen Features sind jederzeit in jedem Punkt berechenbar, um sowohl für einfache als auch für komplexere Gesten den Feature-Vektor schnell und effizient ermitteln zu können. Abbildung 4.3 zeigt die gewählten Features im Zusammenhang einer gezeichneten Geste. Der Ausgangspunkt wird durch Betätigen der Maustaste, der Endpunkt durch Loslassen der Maustaste oder durch Bewegungslosigkeit der Maus über einen längeren Zeitraum bestimmt.

Ist der Feature-Vektor berechnet worden, erfolgt die Klassifizierung. Alle errechneten Features werden mit einer jeweilig zugeordneten Gewichtung $w_{\hat{c}\hat{i}}$, die über eine spezielle Funktion unter Einbezug aller Trainingsgesten einer Klasse berechnet wird, aufaddiert (s. Abb. 4.4). Jede Klasse *c* einer Geste besitzt unterschiedliche Faktoren $w_{\hat{c}\hat{i}}$ für $0 \le i \le F$, wobei *F* die momentane Anzahl benutzter Features bezeichnet. Eine Übereinstimmung der Geste mit einer Klasse *c* liegt für ein maximales $v_{\hat{c}}$ vor.




Cosinus zu Beginn der Geste $f_1 = \cos\alpha$

Sinus zu Beginn der Geste $f_2 = \sin\alpha$

Länge der Bounding Box $f_3 = \sqrt{(x_{\max} - x_{\min})^2 + (y_{\max} - y_{\min})^2}$

Winkel der Bounding Box $f_4 = \arctan\frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}}$

Distanz zwischen Anfangs- und Endpunkt $f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$

Cosinus zwischen Anfangs- und Endpunkt $f_6 = \cos\beta$

Sinus zwischen Anfangs- und Endpunkt $f_7 = \sin\beta$

Sei $\Delta x_p = x_{p+1} - x_p$ und $\Delta y_p = y_{p+1} - y_p$

gesamte Gestenlänge $f_8 = \sum_{p=0}^{P-2}\sqrt{\Delta x_p^2 + \Delta y_p^2}$

Sei $\theta_p = \arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1}\Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$

gesamter Winkel $f_9 = \sum_{p=0}^{P-2}\theta_p$

Summe der absoluten Winkel in jedem Punkt $f_{10} = \sum_{p=0}^{P-2}|\theta_p|$

Summe der quadratischen Werte dieser Winkel $f_{11} = \sum_{p=0}^{P-2} \theta_p^2$

Sei $\Delta t_p = t_{p+1} - t_p$

maximale Geschwindigkeit der Geste $f_{12} = \max_{p=0}^{P-2} \frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$

Gesamtdauer der Geste $f_{13} = t_{P-1} - t_0$

**Abb. 4.3: Berechnungsvorschriften der Features im Zusammenhang mit einer einfachen zweidimensionalen Geste**

$$v_{\hat{c}} = w_{\hat{c}0} + \sum_{i=1}^{F} w_{\hat{c}i} f_i \qquad 0 \le c < C$$

**Abb. 4.4: Errechnen des maximalen Wertes $v_{\hat{c}}$ zur Klassifizierung der Geste**

Zusammenfassend sei gesagt, dass Dean Rubine während seiner Evaluierungsphase sehr gute Erkennungsquoten erreichen konnte. Korrekte Klassifizierungen mit Ergebnissen von bis zu 98% sind aussagekräftige Belege. Diese Ergebnisse schwanken minimal bei unterschiedlicher Anzahl der Trainingsgesten pro Klasse. 15-30 Trainingsgesten für jede Gestenklasse waren dabei keine Seltenheit.
Möglicherweise ist dies auch gleichzeitig der einzige Nachteil in Bezug auf das entwickelte System GRANDMA. Gesten müssen antrainiert werden, um eine ausreichend qualifizierte Feature-Gewichtung erstellen zu können, damit gleiche Gesten unterschiedlichster Zeichenart noch identifiziert und eindeutig zuordnet werden können. Dies muss für jeden neu erstellten Satz von Gesten erneut geschehen. Allerdings, einmal die Liste der Trainingsgesten erstellt, können die ermittelten Gewichtungen dauerhaft benutzt werden.

### 4.2.2 Hidden-Markov-Modell

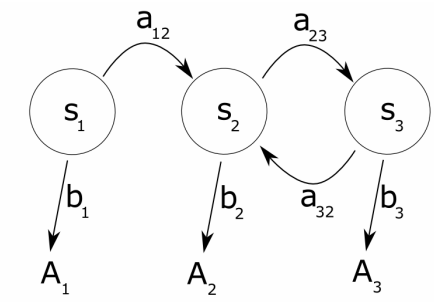
In [YX94] wird ein Verfahren der Gestenerkennung basierend auf einem Hidden-Markov-Modell, kurz HMM, vorgestellt. Eine erfolgreiche Funktionsweise des HMM in der Spracherkennung wurde bereits mehrmals bewiesen (s. z.B. [HAJ90]**)**. Der Ansatz der beschriebenen Methode ist die Frage nach der Übertragbarkeit von Sprach- auf Gestenerkennung.

Das HMM ist ein stochastisches Modell und repräsentiert zwei Zufallsprozesse. Erster beruht auf einer Markov-Kette, eine Verknüpfung von endlichen Zuständen durch Übergangswahrscheinlichkeiten. Zweiter Prozess verbindet jeden Zustand zu jedem Zeitpunkt mit einem entsprechenden Ausgangssymbol. Formal beschrieben wird ein

HMM durch ein Fünfertupel. Abbildungen 4.5 und 4.6Abb. 4.6 sollen den Aufbau eines HMM verdeutlichen.

| | |
|---|---|
| $S = \{s_1, ..., s_n\}$ | Menge aller Zustände |
| $A = \{a_{ij}\}$ | Zustandsübergangsmatrix; legt fest mit welcher Wahrscheinlichkeit $s_j$ auf $s_i$ folgt |
| $B = \{b_1, ..., bn\}$ | Menge der Wahrscheinlichkeiten der Ausgangssymbole (Emissionswahrscheinlichkeit) |
| $\Pi$ | Wahrscheinlichkeitsverteilung, dass $s_i$ Startzustand ist |
| $V$ | Definitionsbereich von $B$ |

**Abb. 4.5: Definition des HMM-Fünfertupels (S,A,B, Π , V)**




| | |
|---|---|
| *s* | „hidden[24]“ Zustände |
| *a* | Übergangswahrscheinlichkeiten |
| *A* | Ausgangssymbole |
| *b* | Emissionswahrscheinlichkeiten |

**Abb. 4.6: Zustandsdiagramm des HMM**

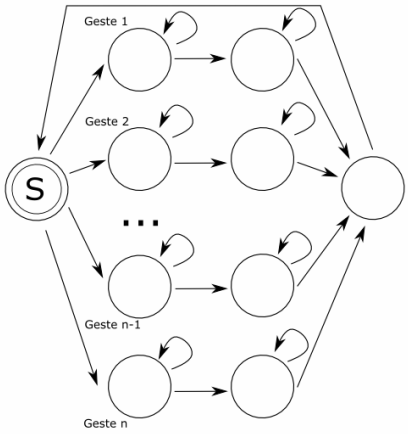



**Abb. 4.7: multi-dimensionales Hidden-Markov-Modell für kontinuierliche Gestenerkennung mit Startzustand *S***

Die Auffassung einer Geste als sequentielles Modell begünstigte laut Autoren die Wahl des HMM als Basis eines Gestenerkennungssystems. Jede einzelne Geste wurde demnach durch ein zweidimensionales HMM repräsentiert. Verwendet wurde darüber hinaus ein multi-dimensionales HMM (s. Abb. 4.7) aller Gesten, um eine größere Erfolgsquote bei der Erkennung einer Geste zu gewährleisten.

[24] dt.: verborgen

Folgende fünf Schritte repräsentieren den Ablauf des entwickelten Gestenerkennungssystems:

1. Festlegung aussagekräftiger Gesten
2. Umformung der Gesten in multi-dimensionale HMM-Terme
3. Sammlung Übungsdaten
4. Training des HMM anhand der gesammelten Übungsdaten
5. Erkennung der Geste mit Hilfe des trainierten HMM z.B. unter Verwendung des Viterbi- oder des Forward-Backward-Algorithmus

Zur Erkennung einer Geste dienen oftmals deren Hauptmerkmale in Form von Schablonen, globalen Transformationen, Zonen und geometrischen Daten. Einleitend wurde als Vorverarbeitung einer Geste die Fourier-Transformation STFT[25] als Verfahren eingesetzt. Die Begründung liegt in der effizienten Umsetzung der FFT[26]-Algorithmen und der Möglichkeit der identischen Abbildung vom Zeit- in den Frequenzbereich ([YX94] S.14).

Eine Geste wird im weiteren Zusammenhang als Bewegungspfad und folglich als dimensionales Signal aufgefasst. Über dieses Signal werden Fenster (Hamming-Windows) gelegt, die das Signal in Blöcke unterteilt, und für jedes einzelne Fenster wird die FFT durchgeführt. Die Merkmale des Signals werden dann aus den Amplituden der FFT-Koeffizienten entnommen und anschließend durch Quantisierung (Vektor-Quantisierung), aufgrund der Verwendung eines diskreten HMMs, in endliche Symbole umgewandelt.
Ziel ist es, das HMM mit der höchsten Wahrscheinlichkeit $P$ für die als Sequenz $\mathrm{O}$ verstandene Geste zu ermitteln (s. Abb. 4.8). Dies kann beispielsweise unter Verwendung des bereits erwähnten Forward-Backward-Algorithmus[27] geschehen.

$$g^* = \arg\max_{alle\lambda} P(\lambda / \mathrm{O})$$

**Abb. 4.8: Bedingung zur Identifikation der Geste**

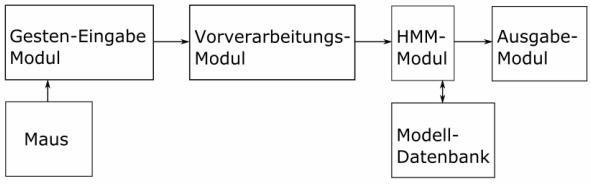



**Abb. 4.9: Aufbau des HMM-basierten Gestenerkenners**

Interaktionselement des entwickelten Prototyps während der Evaluation war eine Maus. In Abb. 4.9 wird die modulare Struktur des fertigen Prototyps grafisch dargestellt. Die Ergebnisse der optimalen Erkennung isoliert behandelter Gesten lagen, abhängig von der

[25] s. Abkürzungsverzeichnis
[26] s. Abkürzungsverzeichnis
[27] Verfahren zur Berechnung von Wahrscheinlichkeiten nach S. L. Lauritzen und D.J. Spiegelhalter

Anzahl der Trainingsgesten, zwischen ca. 92% bis fast 100%. Erfolgsquoten kontinuierlicher Gesten, die sowohl zusammen als auch getrennt untersucht wurden, waren laut Autoren nahezu perfekt. Beispiel für eine kontinuierliche Geste wäre eine zweistellige Zahl, die als ganzes oder bei der jede Stelle einzeln als Geste betrachtet wird (z.B. die Zahl „11“ als „11“ oder „1“ und „1“).

Eine weitere Literatur zur Umsetzung von Gestenerkennungssystemen durch HMM mit ähnlichem Vorgehen ist beispielsweise [LX96].

### 4.2.3 Handgesten

Eine andere Form der Gestenerkennung ist die Verfolgung von Körperteilen wie zum Beispiel dem Kopf oder der Hände. Besonders die Hände können Initiatoren sowohl von dynamischen als auch statischen Gesten sein. Anwendungsgebiete mit Anforderung einer Technik der Handerkennung sind vor allem mit voranschreitender Entwicklung schnellerer Bildverarbeitungsmethoden und entsprechender Hardware überall gegeben, sei es der Bereich der audiovisuellen Kommunikation via Videokonferenz, Überwachungssysteme, die Interaktion in virtuellen Welten oder gar der medizinische Analysebereich.

Primäres Kriterium zur Wiedererkennung der sichtbaren Körperteile ist vielfach die Farbe der Haut. Man ist größtenteils der Annahme, dass die unterschiedlichen Hautfarben der Menschen stark voneinander variieren. Selbst wenn die Hautfarben entfernt miteinander in irgendeiner Weise in einem Farbraum oder Farbmodell verbunden sind, so scheinen Lichtverhältnisse und Helligkeitsschwankungen der Umgebung den eben erworbenen Vorteil in Richtung Hautsegmentierung zunichte zu machen. Ebenso kann ein und dieselbe Hautfarbe unter Verwendung zweier unterschiedlicher Kameras trotz gleich bleibender Lichtverhältnisse zu verschiedenen Farbinformationen führen.

In [YLW98] wird jedoch gezeigt, dass sich dieser Irrtum weitestgehend auf die Intensität der Haut begrenzen lässt und die Farbe an sich eine herausragende Komponente zur Erkennung liefert. Zudem lässt sich der Farbunterschied verschiedener Hauttypen mit Hilfe einer Farbnormalisierung reduzieren. Eine Untersuchung der Hautfarbe von 48 zufällig aus einer Datenbank gewählten Gesichtern lieferte die grafische Informationsdarstellung in [YLW98], S.3 Fig.1(b). Deutlich ersichtlich ist die Tendenz einer Gruppierung im RGB-Farbraum. Nach einer Übertragung in den normalisierten Farbraum ergaben sich zudem noch weitaus bessere Ergebnisse in der Beschränkung der Farbverteilung ([YLW98], S.3 Fig.1(c)).
Häufig erfolgt der Einsatz derartiger Methoden zur Erkennung der Extremitäten aufgrund der schon bereits beschriebenen Robustheit gegenüber Rotation, Skalierung und einer teilweisen Verdeckung. Zudem ließ sich die effiziente Umsetzung farbbasierter Systeme auf weniger leistungsstarken Rechnern in Echtzeit realisieren ([RMG98], S.2).

Diesen Ansatz der Farbnormalisierung und Adaption der Hautfarbe verwendend, können einzelne freiliegende Körperteile erkannt und weitere Verfahren darauf aufbauend angewandt werden. So können bei Methoden zur Erkennung statischer Gesten unter anderem die Stellungen der Extremitäten, vorwiegend Kopf und Hände, detektiert und analysiert werden.

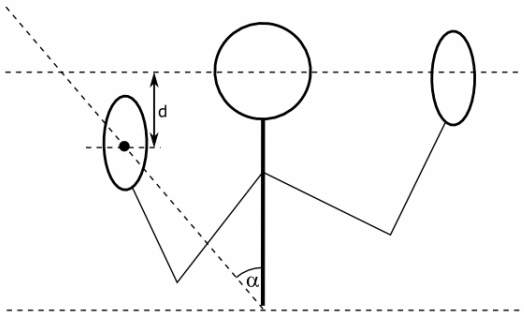



**Abb. 4.10: Denkbare Methode zur Detektion statischer Gesten über mögliche Informationen des Abstandes d zwischen Kopf- und Handhöhe und dem Winkel α**

Auch dynamische Gesten können beispielsweise anhand einer Kombination mit dem vorgestellten Verfahren aus [Rub91] und den Verarbeitungsmethoden des HMM aus [YLW98] kombiniert werden. Als Annäherung an die verbesserte und intuitive Mensch-Maschine-Kommunikation könnte somit ein Gestenerkennungssystem zur Identifikation mit der Hand in die Luft gezeichneter Gesten erfolgen.

Ein Aspekt der hier vorgestellten Methode wird in dem entwickelten System annähernd umgesetzt. Dabei handelt es sich um die Betrachtung des Bildes unter Verwendung eines vom RGB-Raum abweichenden Farbraumes. Vorweg wurde in diesem Zusammenhang der HSV-Farbraum angeführt. Der Vorteil einer Umwandlung liegt im Bereich der Identifikation oder Einordnung einer Farbe trotz schwankender Helligkeit. Mehr dazu im *Kapitel 5 Implementierung*.
Auch wenn die Erkennung von Körperteilen die Anforderungen der Diplomarbeit klar unterstützen würden, wird der konzipierte Gesture Recognizer nicht mit diesen Mitteln umgesetzt. Weitere Erläuterungen dazu im *Kapitel 5.1 Der Marker*.

# 5 Implementierung

Grundsätzlich gab es zu Beginn der Diplomarbeit zwei Ideen, nach welchem Modell man die Implementierung konzipiert. Als Möglichkeiten kamen eine Client-/Server-Anwendung oder eine Anwendungs-Bibliothek in Frage. Eine Bibliothek oder auch ein Toolkit haben den Vorteil, von jedem Programmierer in sein Programm eingebunden und weiterverwendet werden zu können. Dies hätte eine Vereinheitlichung entstehender Software und ständige Verbesserung eines einzigen Toolkits zur Folge, was sich in kontinuierlich steigender Qualität äußern würde. Dagegen stellt eine Client-/Server-Anwendung weniger Ansprüche bei der Umsetzung des Gesamt-Projektes. Zudem handelt es sich bei der Server-Anwendung um ein Endprodukt, das problemlos mit Clientanwendungen verbunden werden könnte, ohne erst ein neues Server-Programm implementieren zu müssen.
Die Entscheidung fiel zugunsten einer Umsetzung nach dem Client/Server-Prinzip aus.

## 5.1 Der Marker

Im Verlauf dieser Ausarbeitung erschien häufiger der Begriff „Marker“, ein Hilfestellungs-Objekt zur Gestenerkennung über die Webcam. In diesem Kapitel soll nun die Begründung der Wahl eines roten Balls als Marker für das umgesetzte Erkennungssystem folgen.

Während der anfänglichen Recherche-Phase wurden Überlegungen über die effiziente Ausarbeitung einer Geste aus einer laufenden Videoeingabe getätigt. Ergebnis war die Einbeziehung eines Markers, anhand dessen in bestimmten Zeitintervallen extrahierte Verlaufspunkte einer Geste deutlicher hervorgehoben werden sollen. Das Aussehen und die Beschaffenheit des Markers waren zu Beginn noch nicht genauer bekannt, sollten aber folgende vorüberlegte Kriterien erfüllen:

- homogene Fläche
- unelastisch
- kantenintensiv
- diffuse Oberfläche
- einfache Form und Kontur
- angemessene Größe

Zwar hätte man auch die Hand, wie im vorgestellten Beispiel in *Kapitel 4.2.3 Handgesten* der bestehenden Gestenerkennungssysteme tracken können, allerdings schränken angestrebte Einsatzgebiete diese Methode wieder ein. Statt der Hautfarbe müssten Arbeitshandschuhe unterschiedlicher Farbe verfolgt werden. Zudem ergibt sich das Problem des elastischen Körpers, da eine Hand in Bilderfolgen nicht zwangsläufig

dieselbe Form aufweist. Demnach könnte man in erster Linie nur auf eigenschaftsbasierte Objekterkennung zurückgreifen. Eine feste Kontur würde jedoch die Wahrscheinlichkeit einer effizienten Erkennung erhöhen.
Folglich bedarf es eines Objektes, das seine Gestalt und Form auch über einen längeren Zeitraum konstant beibehält. Ein geometrischer Körper oder eine geometrische Form mit markanten Konturen sollte Abhilfe schaffen. Das Problem der Abbildung dreidimensionaler Objekte auf zweidimensionale Formen ist eine eventuelle Skalierung, Scherung oder Veränderung der Kontur durch Wechsel des Betrachtungswinkels. Es gilt stets zu bedenken, dass sowohl eine Kamera als auch ein Computer im Allgemeinen keine räumliche Wahrnehmung besitzen. Auch bei geometrischen Objekten wie Würfel oder Pyramiden entsteht eine solche Verzerrung aus verschiedenen Blickwinkeln (s. Abb. 5.1).
Bei einer relativ runden Kugel verfällt dieser Nachteil, da sie aus jeder Blickrichtung wie ein Kreis erscheint. Zudem ist sie rotationsstabil (s. Abb. 5.2). Leistungsintensive Berechnungen rotierter Körpers in eine dem System bekannte Position oder Abbildung sind nicht von Nöten.
Darauf aufbauend erfolgen im Gestenerkennungs-System nacheinander eine Kantendetektion und eine Hough-Transformation für den Kreis, um den Mittelpunkt der Kugel zu ermitteln. Anhand dieser Methoden sollte obendrein das Verdeckungsproblem theoretisch größtenteils umgangen werden, da auch Segmente des Kreises genügen, um Informationen über den Mittelpunkt anhand des Akkumulators zufriedenstellend auswerten zu können.

**Abb. 5.1: Würfel frontal (li.) und um zwei Achsen mit 45° gedreht (re.); vordere Fläche ist hell dargestellt; die Kontur hat sich verändert**

**Abb. 5.2: Kugel frontal (li.) und um zwei Achsen mit 45° gedreht (re.); vordere Halbkugel ist schwarz dargestellt; die Kontur ist unverändert**

In *Kapitel 3.1 Farbräume und Farbmodelle* wurden verschiedene Farbmodelle vorgestellt. Doch was ist der Zweck? Schließlich ist das Ziel eine Gestenerkennung und Gesten besitzen primär keine Farbinformationen und eine reine Objekterkennung einzig basierend auf den Farbmodellen wurde verworfen. Diese Frage ist schnell zu beantworten. Befinden wir uns in einem Raum mit einem monotonen und bestenfalls homogenen Hintergrund, wird die Objekterkennung des kugelförmigen Markers in einem Grauwert-Bild, bis auf einige ungünstige Ausnahmen, soweit reibungslos verlaufen. Da solche idealen Rahmenbedingungen allerdings selten vorherrschen und in einer räumlichen Szene mehrere kugelähnliche Objekte vorkommen können, bedarf es eines zweiten Merkmals zur Extraktion des Markers. Um die Fehlerkennung einzudämmen, wird die Farbinformationen des Bildes genutzt.
Die Wahl einer weniger im Hintergrund vertretenden Farbe sollte als zweite Eigenschaft genügen. Ein intensives Rot war die erste Wahl. Dabei lag die Überlegung, dass die Farbe Rot mutmaßlich deutlich weniger in Arbeitsräumen auftritt als andere Farben, der Wahl zu Grunde.
Das Prinzip bildet der Vergleich möglicher Kugelzentren mit den Farbwerten der Bildpunkte in der Umgebung. Diese umgebenden Pixel sind in einem separaten Bild gespeichert. Unterstützend würde sich demnach eine homogene, diffuse Oberfläche des Balls auswirken. Farbverfälschende Glanzlichter haben durch solch eine Oberflächenstruktur zudem weniger Einfluss auf das Ergebnis des Farbvergleichs.

Noch ein Wort zur Größe des Markers. Der Marker sollte so gewählt sein, dass bei der Kreiserkennung genügend Punkte auf dem Umfang für eine Berechnung vorhanden sind. Wäre die Darstellung des Markers im Bild beispielsweise nur bis zehn Pixel im Radius groß, würden adäquate Mindest- und Maximalradien klein ausfallen. Folglich könnten Berechnungen mit verschiedenen Winkeln dasselbe Akkumulatorfeld füllen. So würden auch dicht beieinander liegende Kanten einen hohen Akkumulatoreintrag an falscher Stelle begünstigen. Dies hätte eine irrtümliche Identifikation des Mittelpunktes zur Folge. Die Grafik in Abbildung 5.3 soll das Problem nochmals verdeutlichen.

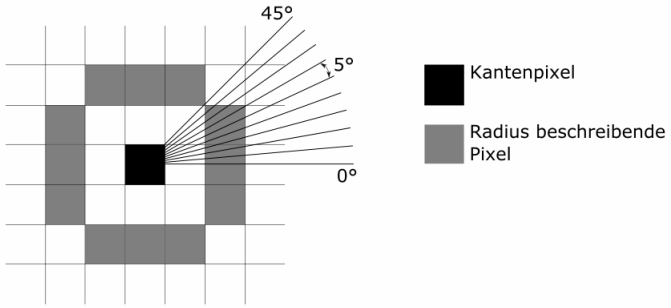



**Abb. 5.3: graue Pixel stellen den Radius von 2 Pixeln dar; ein Eintrag im gleichen Akkumulatorfeld erfolgt für mehrere Winkel (hier zwischen 15° und 40°)**

**Abb. 5.4: alter Marker; Bastelei aus einem Flaschenstopfen, einem Korken, beklebter Pappe und einem Nagel**

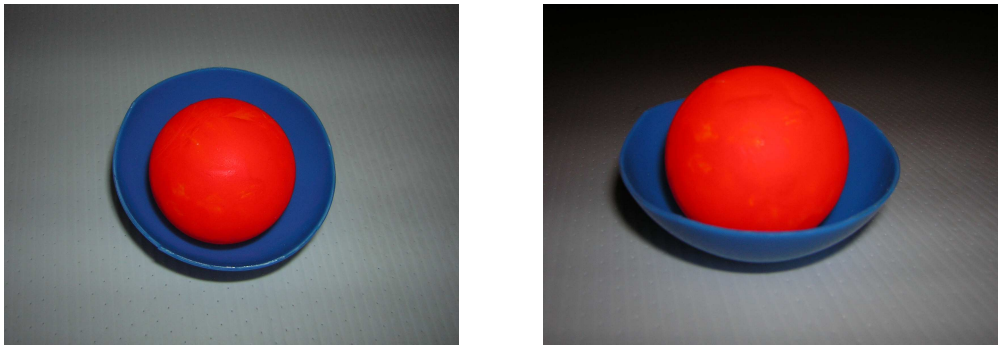


**Abb. 5.5: neuer Marker; mit leuchtroter Farbe bemalter Tischtennisball (RAL 3026 = RGB[255 60 15] = HSV(11°, 0.94, 1.00); ummantelt mit aufgeschnittenem blauen Spielball**

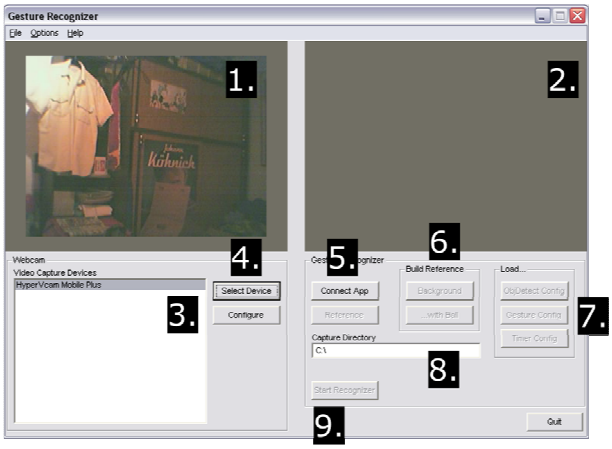
Während den zwischenzeitlichen Testphasen fiel, zwar selten aber nicht unbedeutend, ein Zusammenschmelzen vom Marker mit dem Hintergrund auf. Dies übte einen großen Einfluss auf die Kantendetektion aus. Aus diesem Grund wurde der Marker mit einer Halbkugel ummantelt. Der Zuschnitt der Kugel fiel ein bisschen geringer als die Hälfte aus, um eine gewisse Rotationsstabilität beizubehalten. Die Einfärbung der Halbkugel sollte einer im Farbraum möglichst dem Rot des Markers gegenüberliegenden Farbe oder einer Farbe mit geringer Helligkeit entsprechen. In Frage kamen demnach Farben mit hoher Intensität zwischen Grün und Blau oder Farben im unteren Bereich der HSV-Pyramide. Diese umgebende Halbkugel sichert ein deutlicheres Erkennen der Markerkontur.

## 5.2 Programmablauf

Im Verlaufe dieses Kapitels soll die Funktion und Bedienung des entwickelten Systems näher erläutert werden. Dabei wird vorerst auf eine Demonstration von Quellcode verzichtet. In *Kapitel 5.4* werden die wichtigsten Methoden durch Erklärung der implementierten Funktionen genauer beleuchtet.
Das Programm bzw. dessen Ablauf lässt sich grob in drei Phasen unterteilen:

- Initialisierung
- Konfiguration
- Gestenerkennung

Während der Initialisierungsphase werden alle am Computer angeschlossenen und gleichzeitig für die Anwendung relevanten Geräte aufgelistet. Anschließend erfolgt eine Überprüfung des Vorhandenseins aller drei benötigten Konfigurationsdateien. Beim erstmaligen Aufruf oder nach versehentlicher Löschung dieser Dateien werden sie beim Programmstart mit den vordefinierten Werten der zugehörigen Member-Variablen im Standardverzeichnis angelegt.




1. Vorschaufenster des Live-Streams
2. OpenGL-Feedback
3. Auflistung erkannter Peripheriegeräte
4. *Select Device*: Bestätigungsbutton für das Peripheriegerät
5. *Connect App*: Socket zur Verbindung mit dem Client erstellen
6. Buttons zur Kontrollbilderstellung
7. Buttons zum Laden der Konfigurationsdateien
8. Auswahl des Speicherortes
9. Button zum Starten und Stoppen der Gestenerkennung

**Abb. 5.6: Programmoberfläche**

Abbildung 5.7Abb. 5.7 verdeutlicht den Aufbau des erstellten DirectShow-Graphen nach Auswahl des Aufnahmegerätes und anschließender Bestätigung durch den „Select Device“-Button. Der Capture-Filter repräsentiert das Aufnahmegerät, in diesem Fall die verwendete Webcam. Es folgt ein „Smart Tee“ zur Verteilung des empfangenen Signals auf einen Aufzeichnungs- und einen Vorschau-Pin. Am Aufzeichnungs-Pin befindet sich der „Frame Grabber“, der die einzelnen Frames zur Analyse und Speicherung aus der Bilderfolge extrahiert. Da kein digitales Video aufgezeichnet werden soll, sondern die Entnahme einzelner Bilder genügt, wird der Vollständigkeit entsprechend nur ein „Null-Renderer“ angefügt. Ein an den „Preview“-Pin angeschlossener „Video-Renderer“ liefert die im linken Fenster der Applikation gezeigte Vorschau. Der Graph wird zur Ansicht im Standardverzeichnis gespeichert.

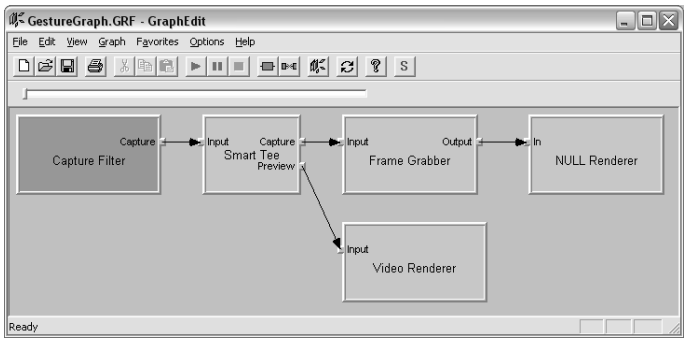



**Abb. 5.7: Filter-Graph der implementierten Anwendung**

Der nächste Schritt besteht in der Herstellung einer Socket-Verbindung mit der Client-Anwendung. Das konzipierte System fungiert hierbei als Server und befindet sich bis zum Zustandekommen der Verbindung im listen()-Modus. Aktiviert wird die Initialisierung der Verbindung durch Betätigen des „Connect App“-Buttons. Dieser Button wird erst nach Festlegung des Eingabegerätes freigeschaltet. Wird die Client-Applikation gestartet, soll sich diese mit dem Port des Servers verbinden. Bei Beendung der Client-Anwendung wird die Verbindung automatisch getrennt und muss bei Bedarf erneut durch Betätigung des Verbindungsbuttons aufgebaut werden. Die verwendete Client-Applikation besteht aus einer einfachen Konsolenanwendung, der mit Gestenbefehlen verknüpfte Strings übermittelt und anschließend in der Konsole ausgegeben werden.

Bevor der Benutzer die Gestenerkennung starten kann, ist die Erstellung von Referenzbildern vorgeschaltet. Durch Anklicken der entsprechenden Buttons werden diese Bilder aus dem Live-Stream gegrabbt und nach Verarbeitung als Bitmaps im Standardverzeichnis gespeichert. Dieser Vorgang soll der manuellen Kontrolle der eingestellten Werte dienen. Idealerweise wird das erste Referenzbild ohne Marker, die folgenden Bilder mit Marker erstellt. Ersteres wird ohne weitere Verarbeitungsschritte direkt als Bitmap auf der Festplatte gespeichert. Die Anfertigung der Referenzbilder ohne und mit Marker ist noch an eine anfängliche Idee, eine Bewegungserkennung durch die mathematische Division umzusetzen, angelehnt.
Nach der Überprüfung der Bilder durch den Anwender kann der Gesture Recognizer durch Anklicken des Buttons „Start Recognizer“ gestartet werden. Durch erneutes Betätigen des Buttons wird die Gestenerkennung wieder ausgeschaltet.

Zur Erstellung der Referenzbilder und zur Erkennung der Gesteninitialisierung und des Verlaufs der Geste werden einige der in *Kapitel 3 Image Processing* beschriebenen Methoden und Verfahren verwendet. Im Ansatz ist die Verarbeitung der Bilder für beide Zwecke ähnlich. Jedoch endet die Bildverarbeitung bei der Erstellung der Kontrollbilder nach der Detektion des Kreismittelpunktes. Diese Bilder sind für den eigentlichen Prozess der Gestenerkennung nicht von Bedeutung, sondern dienen lediglich der manuellen Anpassung der verwendeten Einstellungen durch den Benutzer.
Aus den Informationen des gegrabbten Bildes, abgelegt im Zwischenspeicher, wird in einem ersten Schritt ein zweidimensionales Bitmap-Array ($BA_{Grab}$) erzeugt und in einer STL-Map abgelegt. Ausgehend von diesem Array werden Schritt für Schritt Bildverarbeitungsmethoden darauf angewandt und die jeweils neu entstandenen Bitmap-Arrays gespeichert.
Zuerst erfolgt eine Binarisierung roter Pixel unter Einbezug des HSV-Farbmodelles. Vorteil des Farbmodells ist die Darstellung einer Farbe in Abhängigkeit einer Helligkeitsinformation. Den Hintergrund der Umwandlung in den HSV-Raum bildet im Endeffekt eine eindeutigere Identifizierung des Markers bei schwankender Helligkeit. Jeder einzelne Bildpunkt des RGB-Ursprungsbildes $BA_{Grab}$ wird anhand der Formel in einen adäquaten HSV-Pixel bzw. in zugehörige Werte umgerechnet und mit den einzelnen Schwellwerten $H_{Sch}$, $S_{Sch}$ und $V_{Sch}$ verglichen (vgl. Abb. 3.3: HSV-Pyramide). Handelt es sich um ein tendenziell rotes Pixel, wird der RGB-Pixel im neuen Array $BA_{HSV}$ auf Rot gesetzt, ansonsten auf Schwarz. Das neu entstandene Bitmap-Array wird zur späteren Verwendung ebenfalls in der Map abgespeichert.

Es folgt die Erstellung eines Grauwert-Arrays. Die Umrechnung geschieht anhand der in *Kapitel 3* Abbildung 3.8 vorgestellten Formel. Ziel ist die Reduktion des Datenbestandes. Auch für die noch folgende Kantenerkennung ist ein Grauwert-Array ausreichend. Das errechnete Array $BA_{Grey}$ wird anschließend mit dem Gauss-Filter geglättet, um Fehler bei der Kantendetektion zu vermeiden. In der Implementierung wurden die Gauss-Masken der Größe 3x3 und 5x5 umgesetzt, die jedoch keinen bemerkbaren Leistungsgewinn oder –verlust hervorrufen. Die Randbildpunkte sind durch Aufruf des Konstruktors der Bitmap-Arrays standardmäßig auf Schwarz gesetzt.

Die Kantendetektion wird unter Verwendung des Sobel-Filters getätigt, der eine gleichzeitige Berechnung des Gradientenbetrags sowie der Gradientenrichtung begünstigt (s. *Kapitel 3.4.2.1 Sobel-Operator*). In der programmierten Methode des Sobel-Filters erfolgt direkt anschließend die Kantenverdünnung durch die Non-Maxima-Suppression.
Nach der Kantenverdünnung wird die Kreis-Erkennung nach dem Prinzip der Hough-Transformation angewandt. Als kritisch offenbart sich die Tatsache, dass der Radius des Markers nicht präzise vorliegt. Demnach muss ein dreidimensionaler Akkumulator, wie in *Kapitel 3.4.2.4* präsentiert, angelegt werden. Die Radien sind manuell in der *Config_ObjDet.txt* konfigurierbar und bestimmen durch ihren Abstand die Größe der dritten Dimension des Akkumulators. Durch diese Funktionsweise ergibt sich bei guter Konfiguration ein Leistungsgewinn, da man keine riesige Spanne eventueller Radien schon zur Implementierung einplanen muss, sondern der Benutzer an die Distanz zur Kamera oder an die Größe des Markers angepasste Radien einstellen kann.

Nachdem der Akkumulator durch die Methode der Kreiserkennung gefüllt wurde, wird der Eintrag mit dem höchsten Wert je Radius, folglich pro Akkumulator-Ebene, ermittelt und mit seiner Umgebung verglichen. Für diesen Vergleich wird das Bitmap-Array $BA_{HSV}$ zur Hilfe genommen. Es wird für jeden potentiellen Mittelpunkt die Anzahl der roten Bildpunkte unter einer quadratischen Maske mit dem Mittelpunkt als Zentrum dieser gezählt. Die höchste Summe der Rot-Pixel unter der Maske kennzeichnet im Idealfall den Mittelpunkt des Markers. Zur Visualisierung wird dieser Punkt im HSV-Array durch grüne Pixel gekennzeichnet.

**Abb. 5.8: Originalbild aus Live-Stream gegrabbt**

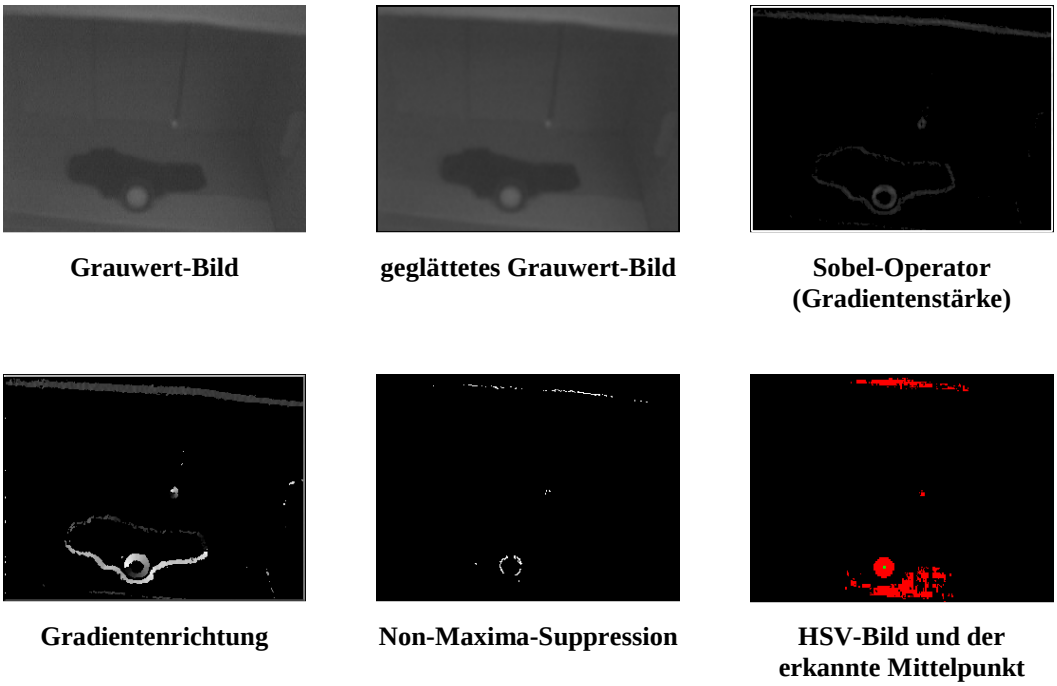



**Abb. 5.9: Überblick der Kontrollbilder**

Bis hierhin unterscheiden sich die Referenzbilderstellung und ein einzelner Schritt der Gestenerkennung nicht. Der abschließende Schritt der Referenzen ist jedoch nur das Zurückschreiben der entstandenen Bitmap-Arrays in einen Zwischenspeicher und die daran anknüpfende Sicherung der einzelnen Arrays als Bitmap im Standardverzeichnis.

Bei der Gestenerkennung hingegen folgen noch die in *Kapitel 5.3 Prinzip der Gestenerkennung* erläuterten Methoden.

Nach der anfänglich beschriebenen Wahl des bevorzugten Gerätes aus der Liste kann, wenn gewünscht oder erforderlich, jederzeit eine Konfigurationsphase begonnen werden. Erst nach Auswahl des Eingabegerätes werden die Konfigurationsbuttons aktiviert. In dieser Konfigurationsphase können Kameraeinstellungen und Veränderungen an den Konfigurationsdateien neu geladen werden. Hiermit sollen eine Feinabstimmung und somit bessere Ergebnisse bei der Markererkennung erreicht werden. Die einzelnen extern bearbeitungsfähigen Dateien können somit fortan durch Betätigen des entsprechenden Buttons stets neu eingelesen werden. Es werden folgende drei Dateien, wie bereits erwähnt, angelegt:

- *Config_ObjDet.txt*: beinhaltet Bildverarbeitungs-relevante Werte
- *Config_Gestures.txt*: Datei zur Vereinbarung der Gesten und deren Funktion
- *Config_Timer.txt*: Festlegung von Zeitintervallen der Gestenerkennung

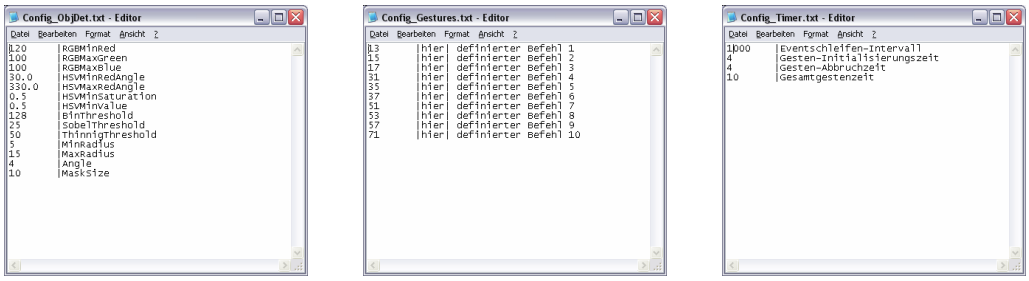



**Abb. 5.10: die Dateien Config_ObjDet, Config_Gestures und Config_TimerGW (von li. nach re.)**

Mit einem einfachen Texteditor werden diese Dateien manipuliert. Die Konfiguration und das erneute Laden können jederzeit in der Konfigurationsphase geschehen. Das Einlesen ist jedoch nicht vor der Auswahl des Eingabegerätes, während dem Warten auf die Client-Anwendung (listen()-Modus) und im Anlegen der Kontrollbilder möglich. Eine kurzweilige Deaktivierung der Buttons signalisiert dies.

Der Aufbau der *Config_ObjDet.txt* und *Config_Timer.txt* erfolgen nach dem gleichen Prinzip. In jeder Zeile steht ein Wert und dessen Bezeichnung. Beim Neuladen werden nur die am Anfang einer Zeile befindlichen Werte eingelesen und als Variablen gesetzt.
Bezüglich der Objekterkennung können mehrere Werte eingestellt werden. An oberster Stelle können die Grenzen der Farbräume festgelegt werden. Diese definieren den Akzeptanzbereich für die Binarisierung des Rot-Wert-Bildes (HSV-Werte). Die RGB-Werte sind allerdings noch ein Überbleibsel, die nur dem Zweck der Visualisierung während der Entwicklung dienen. Wichtig sind die Grenzen für den HSV-Farbraum. Dies sind der einschließende Winkel im Bereich der Rottöne sowie die Mindestsättigung und –helligkeit. Da der Winkel 0° die Farbe Rot definiert, gelten zwei einschließende Winkel. Zum einen der Winkel von 0° bis zum Mindestwinkel und vom Maximumwinkel bis 360°. Beide Werte können in der Textdatei manipuliert werden. Dies sichert eine manuelle Anpassung der Toleranz bei der Farberkennung. Die Standardeinstellungen sind

an die Farbwerte des zu diesem Zeitpunkt aktuell verwendeten Markers angepasst. Allerdings sind die Grenzen recht großzügig festgelegt, um Helligkeitsänderungen oder leichte Verdeckungen durch Schatten abzufangen.
Nächste zwei Faktoren sind nur zu Evaluationszwecken eingebunden und dienen der Informationsdarstellung von Binarisierungsmethoden während der Implementierung. Diese Art der Binarisierung erübrigt sich, da während der Non-Maxima-Suppression ebenfalls ein Schwellwert-Verfahren zum Einsatz kommt, das Kanten geringer Gradientenstärke ausschließt. Dieser Schwellwert lässt sich ebenfalls in der Textdatei verändern.
Nachfolgende Werte beeinflussen die Kreiserkennung. Der Minimal- und Maximalradius grenzen die Suchradien für die Hough-Transformation ein und legen demzufolge die Größe der dritten Dimension des Akkumulators fest. Diese beiden Angaben sollten so gewählt werden, dass sie möglichst nahe an den Radius des Markers auf einem Kontrollbild kommen. Eine Differenz von höchstens fünf Pixeln sollte hierbei genügen. Besitzt der Marker beispielsweise den Radius von zehn Pixeln in einem Kontrollbild, sollte idealerweise der Mindestradius zwischen fünf und zehn und der Maximalradius zwischen 10 und 15 Pixeln betragen. Damit wird eine Toleranz bei einer geringfügigen Verschiebung des Markers in der Tiefe zugesichert und gleichzeitig die Größe des Akkumulators so gering wie möglich gehalten. Ein zusätzlich einstellbarer Abstandswinkel bestimmt die Anzahl der zu ermittelnden Punkte auf den Kreisen um einen Kantenpunkt. Bei einem Winkel von beispielsweise 4° werden 90 Punkte (360°/4° = 90) für jeden Kantenpunkt in den Akkumulator eingetragen.
Ein letzter Wert legt die Größe der Bildregionen fest. Deren genaue Bedeutung wird im nächsten Kapitel erläutert. Wichtig ist, dass die Bildauflösung, sowohl Höhe als auch Breite, ein vielfaches der Regionengröße sein muss, um korrekt zu arbeiten. Für die verwendete Auflösung von 320x240 Pixeln wären folglich Regionengrößen von 10x10, 20x20, 40x40 oder 80x80 zulässig.

Zu beachten gilt bei der Datei zur Konfiguration zeitrelevanter Werte, dass es sich ab dem zweiten Wert um Faktoren handelt. Diese jeweils multipliziert mit dem als erstes angeführten Intervall (Angabe in Millisekunden), ergeben die tatsächliche Zeit. Vier Werte können in dieser Datei bearbeitet werden. Das zuerst angeführte Intervall bestimmt den zeitlichen Abstand in Millisekunden, in dem während der Gestenerkennung die Berechnungen der Regionen gestartet werden sollen. Zweiter Faktor definiert, mit dem Intervall multipliziert, die Initialisierungszeit einer Geste (näheres im Folgekapitel). Der dritte Wert ist der Abbruchbedingung der Geste zugeordnet. Letzter Wert repräsentiert die vorgegebene Gesamtdauer einer Geste. Sollte diese Zeit überschritten werden, wird die Erkennung der momentanen Geste beendet und die Gestenidentifikation automatisch gestartet. Damit sollen zeitlich ins Unendliche gedehnte Gesten vermieden werden.

Im Gegensatz zu vorherig beschriebenen Dateien beinhaltet die Datei *Config_Gestures.txt* zwei Informationen pro Zeile. Erste Information ist eine Zahlenfolge von einer Länge bis maximal 25 Zeichen. Dabei ist darauf zu achten, dass keine gleiche Zahl direkt aufeinander folgend auftaucht. Diese Vereinbarung steht im Zusammenhang mit dem Prinzip der Gestenerkennung (s. *Kapitel 5.3*). Zulässige Zahlenfolgen sind demnach „121234“, „4567“ oder auch „989798“. Folgen wie „1121“ oder „453336“ sind

unzulässig. Diese Zahlenfolgen definieren Gesten nach einem Code vergleichbar mit dem Freeman-Code. Zweite Information, deutlich mit zwei Strichen in der Textdatei umschlossen, ist der mit der Zahlenfolge und dementsprechend mit der definierten Geste verbundene Befehl. Dieser wird nach erfolgreicher Identifikation der Geste an die Client-Anwendung gesendet.

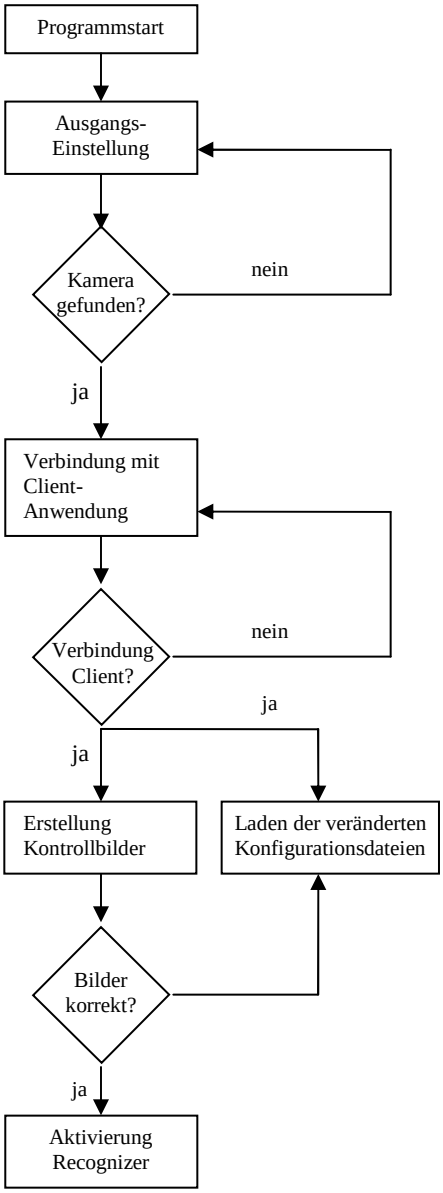



**Abb. 5.11: Standard-Programmablauf**

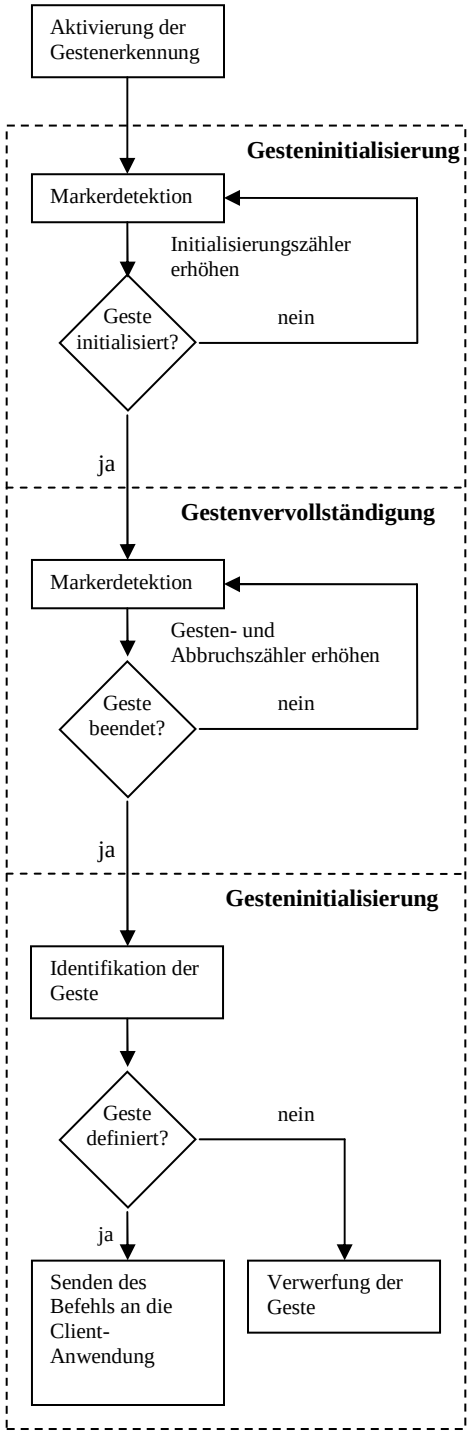



**Abb. 5.12: Vorgang der Gesteninitialisierung und -erkennung**

Das Diagramm in Abb. 5.11 demonstriert einen standardisierten Ablauf bei der Bedienung des implementierten Systems. Abbildung 5.12 stellt zusätzlich den Vorgang der Gestenerkennung nach Aktivierung dar.

## 5.3 Prinzip der Gestenerkennung

In diesem Kapitel soll nun intensiver auf den Grundgedanken der implementierten Gestenerkennung eingegangen werden. Diese Umsetzung wurde im Hinblick auf Einfachheit, Performanz und Benutzerfreundlichkeit entworfen. Das Prinzip und die Funktionsweise ergaben und ergänzten sich im Verlauf der vorliegenden Diplomarbeit und es wird im Folgenden der Stand bei Abgabe vorgestellt. Probleme und Vorschläge zur Verbesserung der Stabilität und Performanz werden im *Kapitel 6 Evaluierung* aufgezeigt und abschließend in *Kapitel 7 Fazit und Ausblick* noch einmal kurz zusammengefasst.
Im Detail funktioniert das Ganze wie folgt. Das gegrabbte Bild wird theoretisch in verschiedene Bereiche, in der Implementierung „Regions“ genannt, unterteilt. Zur Initialisierung einer Geste muss der zur Anwendung gehörende Marker über ein bestimmtes Zeitintervall hinweg in einer Region verharren.

Die Methoden für die Berechnung der Initialisierung sind in der Funktion `checkInitializing()` vereinigt, die durch die Timer-Funktion nach Ablauf eines vereinbarten Intervalls wiederholt aufgerufen wird. Standardmäßig ist die dazu benötigte Regionengröße auf 10x10 Pixel eingestellt. Nach Ermittlung der Mittelpunkt-Koordinaten des Markers werden die x- und die y-Koordinate des Mittelpunktes durch die Regionengröße dividiert und die Nachkommastellen verworfen. Auf diese Weise erhält man zum Beispiel für ermittelte x-Koordinaten des Kreismittelpunkts von 120 bis 129 immer den Regionenwert 12. Berechnete Regionenwerte werden temporär in einer Variable abgelegt und im nächsten Durchlauf zum Vergleich mit der aktuellen herangezogen. Standardmäßig besitzt beim ersten Durchgang die Vergleichsregion den Wert 100 sowohl für die Regionengröße der x- als auch y-Richtung.
Stimmen sowohl der x-Wert als auch der y-Wert der vorherigen mit denen der aktuellen Region überein, wird ein Zeitzähler inkrementiert oder andernfalls auf Null zurückgesetzt. Überschreitet der Wert des Zeitzählers einen weiteren Wert, der die Gesamtwartezeit der Initialisierung repräsentiert (vgl. *Config_Timer.txt*), wird der Beginn der Geste auf „wahr“ gesetzt. Vereinfacht ausgedrückt bedeutet dies, dass der Nutzer den Marker über einen längeren Zeitraum an der gleichen Position halten muss. Mit der Umsetzung durch Regionen ist dabei eine Toleranz bezüglich leichter Bewegungen während des Ruhighaltens gegeben.

Nachdem die Geste als initialisiert registriert wurde, erfolgt der Aufruf der Methode `checkGesture()`, in welcher nachfolgende Regionen gleichermaßen ermittelt und anschließend, im Gegensatz zur Initialisierung, nacheinander in einem Speichercontainer abgelegt werden. Dies geschieht bis zum eindeutigen Abbruch der Geste durch den

Benutzer, simultan zur Art der Initialisierung, oder bis zum Ablauf der erlaubten Ausführungszeit einer Geste.
Der Beginn des Gestenzeichnens soll durch ein geeignetes Symbol im OpenGL-Feedback-Fenster signalisiert werden. Man könnte diese Informationsdarstellung mit dem Recording-Zeichen eines gängigen Videorecorders oder einer Videokamera vergleichen. Das Ende des Zeichnens wird analog durch Deaktivierung dieses Symbols gekennzeichnet. Eine ähnliche Rückmeldung soll auch für den Start des gesamten Gestenerkennungsprozesses umgesetzt werden.

Die vereinbarten und zu zeichnenden Gesten bestehen nicht aus geometrischen Grundkörpern, zum Beispiel aus Kreisen und Quadraten oder buchstabenähnlichen Objekten, sondern aus einer Kombination aus Zahlen umgerechneter Regionen. Herleiten lassen sich diese berechneten Zahlenkombinationen aus einem Freeman-Code-ähnlichem Raster, besser bezeichnet als Gesten-Code. In Abbildung 5.13 der Aufbau des im Uhrzeigersinn angeordneten Gesten-Codes verdeutlicht.

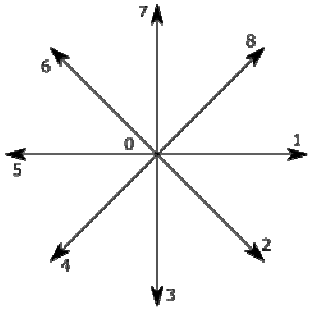



**Abb. 5.13: Gestencode; 0 bedeutet keine Richtungsänderung**

| | *-1* | *0* | *1* |
|---|---|---|---|
| *-1* | **6** | **7** | **8** |
| *0* | **5** | **0** | **1** |
| *1* | **4** | **3** | **2** |

**Abb. 5.14: Gesten-Code-Matrix**

Die Initialisierung einer Geste wurde im vorherigen Abschnitt grundlegend erläutert. Aber wie funktioniert nun das Erkennen der in die Luft gezeichneten Geste im Detail? Wie bereits angesprochen, erfolgt nach dem Setzen der Variablen zur Kennzeichnung der Gesteninitialisierung auf „wahr“ der Aufruf der Funktion `checkGesture()` nach jedem Ablauf des Timer-Intervalls. Nacheinander werden Funktionen zum Ermitteln des Kreismittelpunktes und der zugehörigen Region und zur Berechnung der vergangenen Zeit aufgerufen. Die Vorgehensweise der einzelnen bisher genannten Funktionen sollte soweit geklärt sein, weswegen an dieser Stelle nicht weiter auf diese eingegangen wird.

Bedeutsam sind noch die Funktionen `regToDir()` sowie `identifyGesture()`. In erstgenannter werden die ermittelten Regionen anhand des Gesten-Codes in Richtungswerte zwischen Null und Acht überführt und in einem Speichercontainer abgelegt. Die Überführung erfolgt durch das Teilen der Differenz der aktuellen zur vorherigen Region, sowohl in x- als auch in y-Richtung, durch den Betrag der Differenz. Ergebnisse sind jeweils ein x- und ein y-Wert im ganzzahligen Wertebereich von -1 bis 1, die die Koordinaten für die Gesten-Code-Matrix repräsentieren (s. Abb. 5.14). Der sich an dieser Stelle der Matrix befindliche Eintrag wird schließlich in einem STL-Vector gespeichert.

Vor dem Abspeichern erfolgt allerdings innerhalb der Methode `regToDir()` eine Prüfung der Regionen, um unbeabsichtigte Verwacklungen, im Folgenden Ausreißer genannt, während des Zeichnens zu eliminieren. Hierbei wurden zwei Behandlungen entworfen und implementiert, die für das Aufspüren dieser Ausreißer notwendig sind. Zuerst wird auf die allgemeine Ausnahmebehandlung eingegangen, anschließend wird die spezielle Behandlung für die ersten Einträge im Speichercontainer erläutert. Die Notwendigkeit der Spezialbehandlung ergibt sich aus der Beschreibung der allgemeinen Ausreißerbehandlung.
Prinzipiell geschieht nach der Ermittlung einer neuen Region und deren Zuordnung des Gesten-Codes folgendes: Die zuletzt in den Speichercontainer (STL-Vector) eingefügte Zahl wird mit ihrem Vorgänger und der aktuell ermittelten Codezahl, gleichbedeutend dem Nachfolger des untersuchten Elementes, verglichen (s. Abb. 5.15). Sollte diese ungleich dem Vorgänger oder dem Nachfolger sein, wird sie aus dem Vector gelöscht. Nicht beabsichtigte Ausrutscher während des Gestenzeichnens sollen somit reduziert werden.
Allerdings funktioniert diese Art der Überprüfung zur Ausschließung von Fehlern erst ab dem zweiten Eintrag im Vector und einer dritten aktuell berechneten Richtung. Aus diesem Grund wird eine zweite Überprüfung für den ersten Wert im Vector, der ebenfalls ein Ausreißer sein könnte, benötigt. Diese Methode wird jedes Mal dann aufgerufen, wenn der Vector genau zwei Elemente beinhaltet. Falls der erste Eintrag im Vector ungleich seinem Nachfolger und ungleich dem derzeitig berechneten Gesten-Code-Wert seien sollte, so wird der erste Eintrag gelöscht. In beiden Behandlungsfällen für Ausreißer wird nach der Prüfung der momentan aktuelle Wert, errechnet aus dem Gesten-Code, dem Vector beigefügt
Generell lässt sich das Eliminieren von Ausreißern auch als Ignorieren von Regionenänderungen von nur einem Feld bezeichnen. Die Gestenbeispiele in Abbildung 5.15 sollen die Verwerfung von Ausreißern noch einmal verdeutlichen.
Eine zusätzliche Behandlung für das Ende der Geste wird erst in der noch zu erläuternden Methode der Gestenidentifikation `identifyGesture()` ausgeführt. Bei dieser Behandlung wird die letzte Gestencodezahl mit den zwei vorherigen verglichen und bei Ungleichheit aus dem Vector gelöscht. In diesem speziellen Fall wird auch anschließend kein weiteres Element in den Vector eingefügt.
Eine korrekte Funktionsweise dieser Art der Gestenerkennung setzt ein geringstmögliches Zeitintervall zur Erfassung des Markers voraus. Diese Tatsache stellt sich als Knackpunkt der ganzen Implementierung heraus.

Das Ende der Geste wird nach Vollendung der Zeichenprozedur durch Setzen auf „false“ eingeleitet. Dies ist bei Abbruch des Users durch aktives Beenden, also innehalten in einer Region über eine gewisse Zeitspanne, oder durch Überschreiten der eingeräumten Zeit des Gestenzeichnens der Fall.

Anschließend wird ein Aufruf der Funktion `identifyGesture()` ausgelöst, um aus dem gefüllten Vector die benötigten Informationen über die gerade vollzogene Geste zu erhalten. Hierzu ist es erforderlich, Zahlenfolgen gleichen Wertes auf eine Zahl pro Abschnitt zu reduzieren, sowie Nullen, die eine kurzzeitige Unbeweglichkeit darstellen, zu löschen. Zu guter letzt bleibt eine eindeutige Zahlenfolge übrig, die dann der

zugehörigen Aktion oder dem verknüpften Befehl zugeordnet werden kann. Die hierfür benötigte Funktion besteht aus einfachen `if-else`-Schleifen, und löscht benachbarte Doppelgänger aus dem Vector.

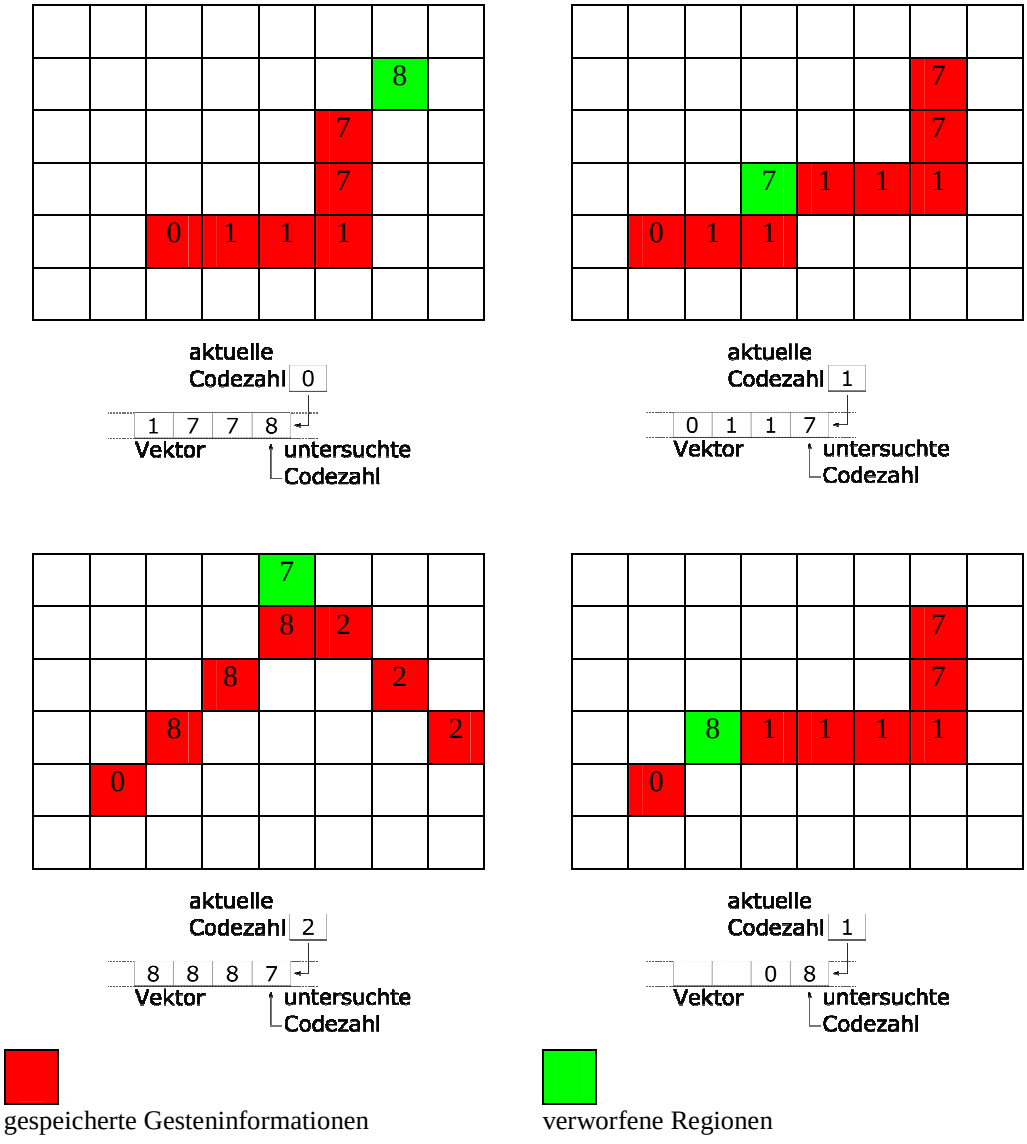



**Abb. 5.15: Beispiele zur Prüfung von Ausreißern. Tabellen symbolisieren ein Bild mit Regionenaufteilung (bei einer Auflösung von 320x240 wäre die Regionengröße 40x40); die Zahlen in den Kästchen der Tabelle geben den berechneten Gesten-Code an.**

Ursprünglich war das Vorhaben ein paralleles Gestenerkennen zuzulassen, geplant. Allerdings gäbe es sehr schnell Überschneidungen bezüglich der Gestendefinitionen. So könnte die Gestendefinition „1-3-4“ vorzeitig und irrtümlich als „1-3“, sofern definiert, erkannt und womöglich ein falscher Befehl ausgeführt werden. Die jetzt implementierte Variante wartet auf ein eindeutiges Signal, in diesem Fall das Innehalten des Markers, und analysiert anschließend die vervollständigte Geste.

Um das Verständnis für die Vorgehensweise der entworfenen Gestenerkennung zu vertiefen, dient das folgende, zahlenorientierte Beispiel, welches den Initialisierungs-

prozess einer Geste darstellt. In *Kapitel 5.4 Auszüge aus dem Quellcode* werden einige wichtige Methoden und Funktionen der gesamten Anwendung detaillierter geschildert.

---

*Vereinfachtes Beispiel zur Demonstration der Gesteninitialisierung*

Beispielhafte vordefinierte Werte:

- Bildauflösung 320x240, Ursprung oberer linker Eckpunkt
- Regionengröße 10x10 Pixel
- Initialisierungszähler = 1
- Abbruchzähler = 2
- Gestenausführungszähler = 4

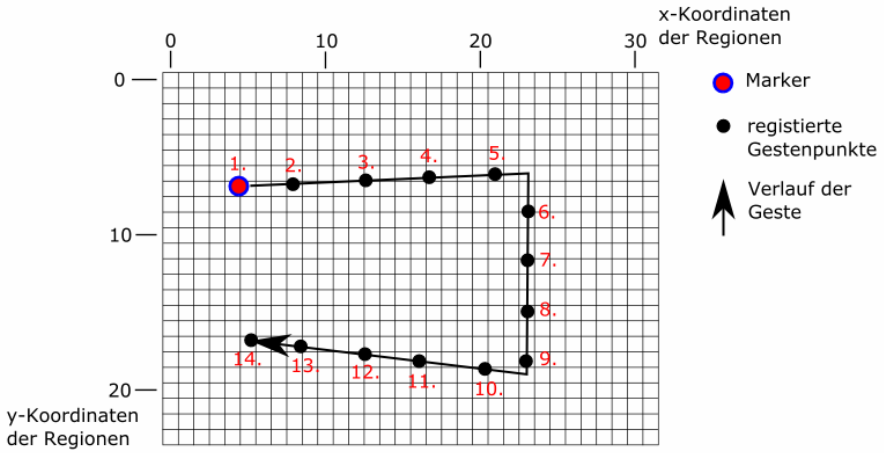
erster Aufruf von `checkInitializing()`:

1. ermittelter Mittelpunkt(x, y) des Markers im Bild: (91, 112)
2. Regionenberechnung:
   a. $\frac{91}{10} = 9$ (x geteilt durch Maskengröße = x-Region)
   b. $\frac{112}{10} = 11$ (y geteilt durch Maskengröße = y-Region)
3. Vergleich mit vorangegangener Region (erster Aufruf: $Region_0$ (x, y) = 100,100)
4. aktuelle $Region_1$(9, 11) ungleich alte $Region_0$(100, 100)
5. setzen des Timers auf Null, $Region_0$ mit aktueller $Region_1$ überschreiben
6. Timer ist kleiner als der Initialisierungszähler ($0 < 1$) → Geste nicht gestartet

zweiter Aufruf von `checkInitializing()`:

1. errechneter Mittelpunkt: (95, 115)
2. Regionenberechnung:
   a. $\frac{95}{115} = 9$
   b. $\frac{115}{10} = 11$
3. Vergleich mit vorangegangener $Region_1$(9,11)
4. $Region_2$ entspricht $Region_1$
5. Inkrementieren des Timers um 1
6. Timervergleich ($1 \geq 1$) → Geste gestartet
7. ...

---




| Punkt | Region | | Vergleichsregion | | Differenz | | Richtung |
|---|---|---|---|---|---|---|---|
| | x | y | x | y | x | y | |
| 1. | 4 | 7 | ... | | ... | ... | ... |
| 2. | 8 | 7 | 4 | 7 | 1 | 0 | 1 |
| 3. | 13 | 6 | 8 | 7 | 1 | -1 | **8** |
| 4. | 17 | 6 | 13 | 6 | 1 | 0 | 1 |
| 5. | 21 | 6 | 17 | 6 | 1 | 0 | 1 |
| 6. | 23 | 8 | 21 | 6 | 1 | 1 | **2** |
| 7. | 23 | 12 | 23 | 8 | 0 | 1 | 3 |
| 8. | 23 | 15 | 23 | 12 | 0 | 1 | 3 |
| 9. | 23 | 18 | 23 | 15 | 0 | 1 | 3 |
| 10. | 20 | 19 | 23 | 18 | -1 | 1 | **4** |
| 11. | 16 | 18 | 20 | 19 | -1 | -1 | 6 |
| 12. | 13 | 17 | 16 | 18 | -1 | -1 | 6 |
| 13. | 8 | 17 | 13 | 18 | -1 | -1 | 6 |
| 14. | 5 | 17 | 8 | 17 | -1 | 0 | **5** |

**Abb. 5.16: Beispiel einer Geste mit Regionenermittlung und Richtungsbestimmung; die in der letzten Spalte fett dargestellten Zahlen werden durch die Vergleichsbehandlung gelöscht; nach Reduktion doppelter Zahlen bleibt die Zahlenfolge 1-3-6**

Damit die Erkennung des Markers und die Fehlerbehandlung einwandfrei erfolgt, ist als Grundvoraussetzung ein sehr geringes Zeitintervall zu wählen. So kann durch das Erkennen mehrerer Positionen ein eindeutigerer Verlauf der Geste ermittelt werden. Hiermit wird noch einmal die Echtzeitfähigkeit der implementierten Methoden angefordert.

## 5.4 Auszüge aus dem Quellcode

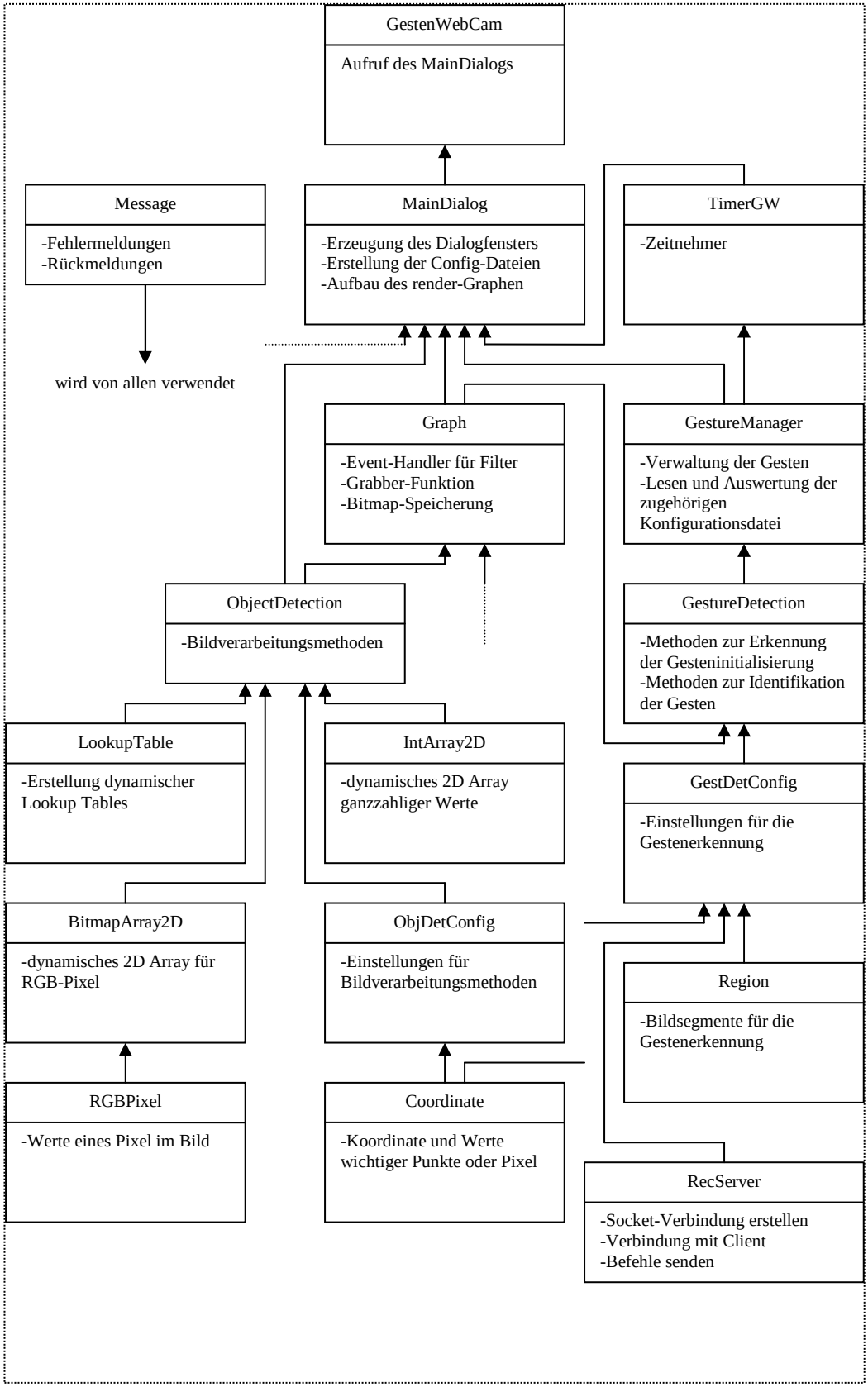



**Abb. 5.17: vereinfachtes "Klassendiagramm" mit Include-Beziehungen**

Sicherlich ist der Programmcode als Anhang nur bedingt förderlich für die Lesbarkeit und Übersichtlichkeit einer Diplomarbeit. Deswegen soll an dieser Stelle nur auf einige der wichtigsten Programmabschnitte eingegangen werden. Kriterien für die Auswahl der präsentierten Methoden sind gesteigerte Flexibilität und Effizienz. Vorweg soll ein alltagssprachlich formuliertes Klassendiagramm einen Überblick über den Aufbau der entwickelten Applikation und der Hauptfunktionen der programmeriten Klassen vermitteln. Eine Pfeilverbindung symbolisiert die „Ist-inkludiert-von“-Beziehung.

### 5.4.1 BitmapArray2D

Zweidimensionale Arrays bieten sich an, um die im Zwischenspeicher enthaltenen Informationen des gegrabbten Bildes für die weiteren Zugriffe geeignet abzulegen. Der Aufbau eines Arrays mit zwei Dimensionen vermittelt einen passenden Bezug zu einer Bildstruktur. Arrays benötigen jedoch bekanntlich konstante Größen als Angaben während der Deklaration. Da aber eine Veränderung der Bildauflösung, abhängig vom Gerät, während der Programmlaufzeit möglich ist, sind vordefinierte Dimensionen wenig sinnvoll. Aus diesem Grund wurde eine Idee beruhend auf den *ByteArrays* aus [Pau03a] in der Klasse `BitmapArray2D` implementiert, mit denen Arrays variabler Größe zur Laufzeit im Speicher angelegt werden können.

```
...
//Konstruktor
BitmapArray2D::BitmapArray2D(int width, int height)
{
  m_Width = width;
  m_Height = height;

  m_Array = new RGBPixel*[width];
  for(int i = 0; i < width; i++)
  {
    m_Array[i] = new RGBPixel[height];
  }
}
```

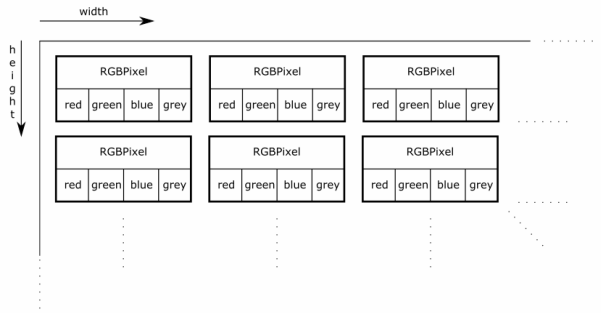



**Abb. 5.18: Aufbau eines dynamischen Objektes der Klasse BitmapArray2D**

Die Klasse `BitmapArray2D` legt Objekte zur Speicherung von `RGBPixel` an. Für ganzzahlige Werte existiert hingegen eine ähnliche Klasse namens `IntArray2D`, die

beispielsweise in der Methode des Sobel-Operators zum Abspeichern von Gradientenbeträgen Verwendung finden.

### 5.4.2 Lookup Table

Der Konstruktor der Klasse LookupTable ist an die des BitmapArrays angelehnt. Um das Inkrementieren im Akkumulator zu beschleunigen, wird ein LUT[28] beim Programmstart für die definierten Radien und Winkeln angelegt, um an entsprechender Stelle die Sinus- und Cosinus-Werte effizient abfragen zu können. Auch hier ist während der Laufzeit eine Manipulation der Radien und Winkel möglich. Aus diesem Grund sind Lookup Tables ebenfalls dynamische, zweidimensionale Arrays.

```
...
//Konstruktor
LookupTable::LookupTable(int dimR, int dimA)
{
  m_DimRadii = dimR;
  m_DimAngle = dimA;

  m_Table = new double*[m_DimRadii];
  for(int i = 0; i < m_DimRadii; i++)
  {
    m_Table[i] = new double[m_DimAngle];
  }
}
```

```
//---------------------------------------------------------------
// Name:   detectCircleCenter()
// Desc:   draws circles to find center of the most significant
//       circle
//---------------------------------------------------------------
...
for(int radius = minRadius; radius <= maxRadius; radius++)
        {
          for(int actualAngle = 0; actualAngle < 360;
              actualAngle += angle)
          {
            cX = i + cosLookup[radius - minRadius][actualAngle];
            cY = j + sinLookup[radius - minRadius][actualAngle];

...
              //increment all points of the circle
              accumulator[cX,cY,(radius - minRadius)]++;
...
          }
        }
...
```

Beim Füllen des Akkumulators werden ein Sinus- und ein Cosinus-LUT zur Kreiserkennung verwendet. Abbildung 3.67 verdeutlichte das Prinzip. Um jeden Kantenpunkt wird ein Kreis mit Radius *R* gezogen. Zur Ermittlung der entsprechenden

[28] s. Abkürzungsverzeichnis

Koordinaten im Akkumulator des Winkels $\alpha$ wird der Eintrag in den entsprechenden LUTs (im Codefragment `cosLookup` und `sinLookup`) abgefragt.

### 5.4.3 Non-Maxima-Suppression

An dieser Stelle folgt ein Auszug aus der Funktion der Non-Maxima-Suppression. Es wird der Ausschnitt zur Kantenverdünnung in einem Pixelpunkt *sF[i][j]* nach *Maske A* aus Abb. 3.61 gezeigt. Die zugehörige Gradientenorientierung (vgl. Abb. 3.60) des Pixels ist im entsprechenden Array *sectorArray[i][j]* gespeichert. Besitzen die Nachbarn des aktuellen Punktes die gleiche Gradientenrichtung, so werden die zugehörigen Werte in den Vergleich der Maxima mit einbezogen. Handelt es sich bei dem betrachteten Pixel um das Maximum der höchstens drei Werte, wird er im Array auf Weiß, ansonsten auf Schwarz gesetzt.

```
...
//Non-Maxima-Suppression-----------------------------------------
  //ignore the border
  for(j = 4; j < (height - 4); j++)
  {
    for(i = 4; i < (width - 4); i++)
    {
      //abolish "minimal maxima"
      if(sF[i][j] > threshold)
      {
        sectorValue1 = 0;
        sectorValue2 = 0;

        //test Sector-value
        if(sectorArray[i][j] == 0)
        {
          //neighbors same gradient-direction?
          if(sectorArray[i-1][j] == 0)
            sectorValue1 = sF[i-1][j];
          if(sectorArray[i+1][j] == 0)
            sectorValue2 = sF[i+1][j];

          //suppress non-maxima
          if(sF[i][j] ==
              max(sF[i][j],max(sectorValue1, sectorValue2)))
          {
            sobelArray[i][j] = white;
          }
          else
            sobelArray[i][j] = black;
        }
        ...
      }
      else
        sobelArray[i][j] = black;
    }
  }
...
```

Zu beachten ist die Schwellwertprüfung (`threshold`). Ein betrachteter Bildpunkt wird schon ohne weitere Behandlung auf Schwarz gesetzt, wenn der Schwellwert unterschritten ist. Die Abbildung 5.19 zeigt links eine Kantenverdünnung ohne Schwellwertabfrage, rechts mit geeigneter Schwellwertprüfung. Das Ergebnis ist eindeutig, denn minimal ausgeprägte Kanten, die einen ansonsten großen Einfluss auf das Bild mit Kantenverdünnung, werden unterdrückt.

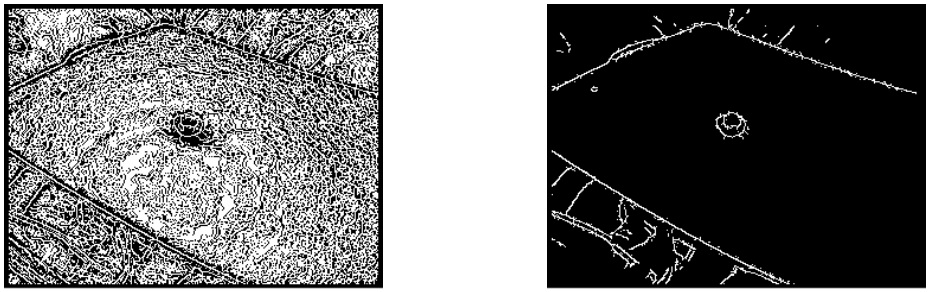


**Abb. 5.19: Bilder nach NMS; links ohne, rechts mit vorhergehender Schwellwertprüfung der Kantenstärke pro Pixel**

### 5.4.4 detectCircleCenter()

In dieser Funktion ist die Kreiserkennung nach Grundlage der Hough-Transformation umgesetzt. Der beschriebene Akkumulator wird durch ein System-Array umgesetzt. Das Verfahren wird auf das nach dem Sobel-Filter und nach der NMS entstandene Kantenbild angewandt. Für jeden Kantenpunkt werden im Winkelabstand `angle` von 0° bis 360° alle Werte der beigeordneten Felder im definierten Radius um „1“ erhöht. Die Feld-Koordinaten werden errechnet aus den Koordinaten des aktuellen Kantenpixels zuzüglich eines entsprechenden Cosinuswertes für die Breite und eines Sinuswertes für die Höhe. Diese Werte können aus einem vorher angelegten Lookup Table abgelesen werden. Nachdem für alle Kantenpunkte die zugehörigen Felder im Akkumulator erhöht wurden, erfolgt der Aufruf der noch erläuterten Funktion `findMidpoint()`, in der aus allen potentiellen Mittelpunkten das endgültige Kreiszentrum des Markers ermittelt und zurückgegeben wird.

```
//---------------------------------------------------------------
// Name:   detectCircleCenter()
// Desc:   draws circles to find center of the most significant
//       circle;
//       version with Managed Extension
//---------------------------------------------------------------

...

    //three-dimensional accumulator
  Int32 accumulator[ , , ] = __gc new Int32[width, height, dim];

  ...
```

```
  //coordinates of the lookup-table
  int cX = 0;
  int cY = 0;

  ...

  //alternative with lookup-table
  LookupTable* sinTable = getSinLookup();
  LookupTable* cosTable = getCosLookup();
  double** sinLookup = sinTable->getTable();
  double** cosLookup = cosTable->getTable();

  //search in the image for edge-pixel and increment
  //in every dimension (radius) of the accumulator
  //for every angle-step the corresponding field
  for(int j = 0; j < height; j++)
  {
    for(int i = 0; i < width; i++)
    {
      if(bitmapArray[i][j] == white)
      {
        for(int radius = minRadius; radius <= maxRadius; radius++)
        {
          for(int actualAngle = 0; actualAngle < 360;
              actualAngle += angle)
          {
            cX = i + cosLookup[radius - minRadius][actualAngle];
            cY = j + sinLookup[radius - minRadius][actualAngle];

            if(cX >= 0 && cX < width && cY >= 0 && cY < height)
            {
              //increment all points of the circle
              accumulator[cX,cY,(radius - minRadius)]++;
            }
          }
        }
      }
    }
  }

  Coordinate* coord =
              findMidpoint(accumulator, width, height, hsvBmp);

  return coord;
}
```

### 5.4.5 findMidpoint()

Die endgültige Suche des Marker-Zentrums verläuft in drei recht einfachen Schritten. Zuerst wird aus jeder Radius-Dimension des Akkumulators das Maximum als (x, y)-Koordinate in einem STL-Vector eingetragen. Hierbei handelt es sich im Endeffekt um potentielle Mittelpunkte. Dann wird für jeden Eintrag im Vector die Summe der roten Pixel unter einer Maske im HSV-Array gebildet. Die Größe der Maske ist dabei in der zugehörigen Textdatei manuell veränderbar. Abschließend wird das Pixel mit der größten Summe als erkannter Mittelpunkt zurückgegeben.

```
//---------------------------------------------------------------
// Name:   findMidpoint()
// Desc:   detects the most significant centerpoint with
//       the help of a mask and the hsv-image
//---------------------------------------------------------------

...

  for(int dim = 0; dim < dimension; dim++)
  {
    max = 0;

    for(int y = 0; y < height; y++)
    {
      for(int x = 0; x < width; x++)
      {
        if(accumulator[x, y, dim] > max)
        {
          max = accumulator[x, y, dim];
          xCenter = x;
          yCenter = y;
        }
      }
    }
    coord = new Coordinate(xCenter, yCenter);
    coordVector.push_back(coord);
  }


  //count all red pixel of the hsv-image under the mask
  for(coordinate = coordVector.begin();
      coordinate!= coordVector.end(); ++coordinate)
  {

    int x = (*coordinate)->getX();
    int y = (*coordinate)->getY();
    value = 0;

    for(int i = (x - size); i <= (x + size); i++)
    {
      for(int j = (y - size); j <= (y + size); j++)
      {
        if((i >= 0) && (j >= 0) && (i < width) && (j < height))
        {
          if(hsvArray[i][j] == red)
            value++;
        }
      }
    }
    (*coordinate)->setValue(value);

  }

  //search the coordinate with the highest red-share in the hsv-image
  redValueMax = 0;

  for(coordinate = coordVector.begin();
      coordinate!= coordVector.end(); ++coordinate)
  {
    if((*coordinate)->getValue() > redValueMax)
    {
      redValueMax = (*coordinate)->getValue();
      center = *coordinate;
    }
```

```
  }

  ...

  return center;
}
```

### 5.4.6 regToDir()

In dieser Funktion werden die bereits in *Kapitel 5.2 Programmablauf* beschriebenen Methoden zur Umrechnung der ermittelten Regionen des Mittelpunktes in Richtungscodes (s. Abb. 5.13 und Abb. 5.14) sowie zwei von drei Behandlungen zur Eliminierung von unwichtigen Gestenschritten ausgeführt. Um einen Zugriff auf die Maske des Gestencodes mit negativen Indizes von „-1“ zu vermeiden, erfolgt noch eine Verschiebung der Indizes in den positiven Bereich.

Damit Ausreißer abgefangen werden, wird die letzte im Vector stehende Region $R_L$ mit ihrem Vorgänger $R_M$ und der aktuellen Region $R_N$, die sich zu diesem Zeitpunkt noch nicht im Vector befindet, verglichen. Ist $R_L$ ungleich $R_M$ und ungleich $R_N$ wird sie gelöscht. Um auch den ersten Eintrag im Vector prüfen zu können, wird bei einer Anzahl von zwei Vectorelementen und nach Berechnung der dritten Region, die erste Region im Vector mit ihren beiden Nachfolgern verglichen und ergebnisabhängig behandelt. Zum Schluss wird die aktuelle Region im Vector abgelegt.

```
//---------------------------------------------------------------
// Name:   regToDir()
// Desc:   Converts region into an freeman-code-like integer
//       and deletes sequels of double values
//---------------------------------------------------------------

  ...
  Region* regionOld = getRegionOld();

  int x = region->getRegionX() - regionOld->getRegionX();
  int y = region->getRegionY() - regionOld->getRegionY();

  //divide through abs to get coordinates for the code-mask
  //values between -1 and 1
  if(x != 0)
    x = x/(abs(x));
  if(y != 0)
    y = y/(abs(y));

  //shift values to the range 0 to 2
  dir = m_CodeMask[x+1][y+1];

  //special-treatment at the beginning of the gesture
  if(m_GestVec.size() == 2)
  {
    if((m_GestVec.front() != dir) &&
        (m_GestVec.front() != m_GestVec[1]))
    {
      m_GestVec.erase(m_GestVec.begin());
```

```
    }
  }

  //normal-treatment for the gesture
  if(m_GestVec.size() > 2)
  {
    //compare the last three regions
    if((m_GestVec.back() != dir) &&
        (m_GestVec.back() != m_GestVec[m_GestVec.size() - 2]))
    {
      //erase the second last element/region if it is an outlier
      m_GestVec.pop_back();
    }
  }
  //push actual direction in the vector
  setDirection(dir);
}
```

### 5.4.7 identifyGesture()

Diese Methode ist letztendlich für die Behandlung von Ausreißern am Ende einer Geste und, viel wichtiger, für die Reduktion der Gestencodezahlen im Vector auf die wichtigsten Informationen zuständig. Eine Erkennung von Ausreißern am Ende kann durch den Vergleich des letzten Eintrages mit den zwei vorherigen im Vector bewerkstelligt werden. Sollte der letzte Eintrag weder gleich seinem Vorgänger, noch dessen Vorgängers sein, wird er gelöscht.

Die Reduktion der Daten im Code auf akzeptable Zahlenfolgen ist eine Verkettung mehrerer `if`-Abfragen. Ziel ist die Reduzierung auf Zahlencodes mit nicht aufeinander folgenden gleichen Zahlen. Zudem werden aus dem Vector sämtliche Nullen, die einen kurzzeitigen Stillstand des Markers symbolisieren, gelöscht.
Vorgegangen wird dabei wie folgt: Der Iterator läuft über den gesamten Vector und überprüft den Inhalt des aktuellen Vectorfeldes. Ist in diesem Vectorfeld eine Null eingetragen, wird diese sofort gelöscht. Handelt es sich bei dem Eintrag um eine Zahl größer Null, wird diese mit einer temporären Variablen mit dem Wert des vorherigen Feldes, bzw. beim ersten Durchlauf mit Null, verglichen. Bei Gleichheit mit der temporären Variablen wird der aktuell untersuchte Eintrag ebenfalls gelöscht. Tritt keiner der beiden Fälle ein, wird der Wert im aktuell untersuchten Vectorfeld der Vergleichsvariablen zugewiesen.
Nach Durchlaufen des gesamten Vectors werden abschließend die restlichen Einträge des Vectors in einen Zwischenspeicher ausgelesen, um die Information mit den vereinbarten Gestenzahlenfolgen aus der Textdatei vergleichen zu können. Danach wird der zugehörige Befehl an die Client-Applikation gesendet.

```
//---------------------------------------------------------------------
// Name: identifyGesture()
// Desc:   Reduces important features and identifies the "drawn" //
    gesture
//---------------------------------------------------------------------
void GestureDetection::identifyGesture()
{
  int temp = 0;
  char gestureBuf[25] = "";
  int bufCounter = 0;

  std::vector<int>::iterator reg_Iter;

  //special-treatment at the end of the gesture
  if((m_GestVec.back() != m_GestVec[m_GestVec.size() - 3]) &&
(m_GestVec.back() != m_GestVec[m_GestVec.size() - 2]))
  {
    m_GestVec.pop_back();
  }

  //for all regions in the vector
  for(reg_Iter = m_GestVec.begin(); reg_Iter != m_GestVec.end();
++reg_Iter)
  {
    //delete all regions without movement-direction
    if((*reg_Iter) == 0)
    {
      m_GestVec.erase(reg_Iter);
    }
    else if((*reg_Iter) == temp)
    {
      m_GestVec.erase(reg_Iter);
    }
    else
    {
      temp = (*reg_Iter);
    }
  }

  //read every region into a buffer
  for(reg_Iter = m_GestVec.begin(); reg_Iter != m_GestVec.end();
++reg_Iter)
  {
    gestureBuf[bufCounter] = (*reg_Iter);
    bufCounter++;
  }
  ...
}
```

# 6 Evaluierung

Eingangs erwähnt wurde das Anstreben, die Anwendung, so fern durchführbar, auch minimal ausgestatteten Rechner zugänglich zu machen. Die bis zur Evaluierung geringste leistungsfähige Konfiguration bestand aus einem Notebook der Marke Acer, Typbezeichnung Aspire 1362LMi. Angefügt kurz die technischen Daten:

| Art | Leistung |
|---|---|
| Prozessor: | AMD Sempron 2800+ (1,6 GHz) |
| Grafikkarte: | NVIDIA GeForce FX Go, 64MB |
| Speicher: | 512MB DDR RAM |
| Betriebssystem: | Windows XP Home Edition |

Bei der verwendeten Webcam handelte es sich um eine Hypercam Mobile Plus aus dem Hause Aiptek. Auch von dieser seien die technischen Spezifikationen kurz angeführt.

| Art | Leistung |
|---|---|
| Sensor: | CMOS VGA 640x480 Pixel |
| Auflösung: | min. 160x120 – max. 640x480 |
| Bildwiederholrate: | max. 30 Bilder pro Sekunde |
| Farbtiefe: | 16/24 RGB |
| Schnittstelle: | USB-Port |
| Blickwinkel: | 56° |

Nun folgend sollen Testergebnisse und Erfahrungen mit dem System präsentiert werden. Schon vorweg genommen sei, dass die Gestenerkennung selbst nicht getestet werden konnte, der Grund wird später in diesem Kapitel genauer beschrieben.

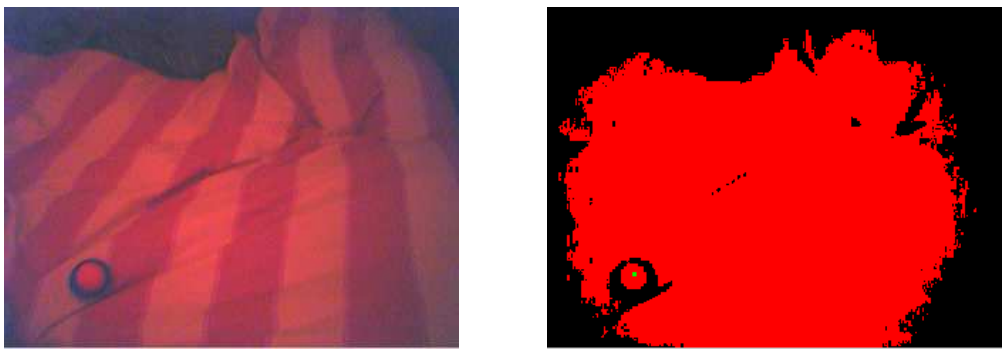


**Abb. 6.1: zusätzliche Kanteninformationen ermöglichen das einwandfreie Erkennen des Markers in farbähnlichen Hintergründen**

Ein nicht zu unterschätzendes Problem, das sich zu Beginn ergab, war die Kombination der Anwendung *WinCap* und dem DirectX-Sample *GrabBitmaps*. Anfänglich schien die Verknüpfung beider Basis-Bausteine weitaus schwieriger und zeitaufwendiger als eine komplette Neuerstellung eines eigenen Frameworks. Der Code der einzelnen Anwendungen musste erst einmal verstanden werden. Anschließend wurde ein Verknüpfen beider, separat vorliegender Anwendungen in Angriff genommen. Aufgrund des manchmal unübersichtlichen Codes, dauerte die Kombination der Applikationen länger als geplant. Nach erfolgreicher Verbindung beider Komponenten liefern diese jedoch manche Vorteile, zum Beispiel die Erkennung sämtlicher nutzbarer Video-Hardware.
Des Weiteren gibt es häufiger Probleme mit der Verbindung der WebCam. Eine Fehlermeldung „Error 1: Cannot create video capture filter“ scheint auf eine vorhergehende unsachgemäße Verbindungstrennung von der Webcam hinzuweisen. Diese tritt vor allem nach einem StandBy des Laptops ein. Eine Auswertung der Fehlermeldung wäre sicherlich einfacher gewesen, wenn der verwendete Code zur Verbindung mit einer Webcam Schritt für Schritt von eigener Hand erstellt worden wäre.

Detailreiche Hintergründe einer Bildszene stellen ein weiteres Problem bei der einwandfreien Erkennung des Markers dar. Diese fallen besonders negativ bei der Kantenerkennung auf. Trotz Anwendung von Schwellwertverfahren zur Dezimierung informationsarmer Kantenpunkte und der direkt anschließenden Kantenverdünnung durch die Non-Maxima-Suppression, hinterlassen kleine Bereiche mit vielen Objektkanten störende Pixel, die bei der Kreiserkennung, vor allem bei niedrigem Radius, schwerwiegend ins Gewicht fallen. Auch wenn diese Konturen nicht zwangsläufig kreisförmig auftreten, wird der Akkumulator mit den Analyseergebnissen dieser Konturen gefüllt. Somit können diese zu Fehlauswertungen führen. Die Folgen wären die Erkennung des Mittelpunktes in einem Bereich mit vielen kleinen Objektkanten. Jedoch kann man diesen Fall teilweise abfangen. Wählt man den Mindestradius groß genug, werden feinere Strukturen weniger beeinflussend ausgewertet. Der Nachteil ist allerdings, dass man, abhängig vom gewählten Mindestradius, entsprechend näher an die Kamera treten muss oder einen größeren Marker braucht.

Ein weiteres anfängliches Problem äußerte sich in der Arbeitsgeschwindigkeit der Methode `detectCircleCenter()`.Anfänglich programmierte Berechnungen von Sinus und Cosinus zur Laufzeit innerhalb einer Schleife, die merklich die Geschwindigkeit senkten, wurden durch dynamische Lookup Tables ersetzt. Die Implementierung von LUTs hatte einen enormen Leistungsgewinn zur Folge. Sowohl der Sinus- als auch der Cosinus-LUT wird beim Programmstart und beim Einlesen der *Config_ObjDet.txt* neu erstellt und erlaubt eine effizientere Bestimmung des Sinus und des Cosinus als eine zur Laufzeit durchgeführte mathematische Berechnung dieser Werte.

Das Hauptproblem allerdings während der gesamten Entwicklungsphase bestand im extrem hohen Speicherbedarf. Maßgeblich für diesen Speicherbedarf ist die Erstellung und Handhabung des Akkumulators. Durchschnittlich liegt die Prozessorauslastung nach dem Start der Applikation und der Auswahl des Peripheriegerätes bei ca. 8% - 20%. Der

angeforderte Speicher beansprucht parallel rund 20MB. Dieser angeforderte Speicher wächst bei mehrmalig ausgeführter Kontrollbilderstellung oder beim Starten des Gesture Recognizers zusehends an. Damit das laufende Programm keinen Systemabsturz zur Folge hatte, wurde die Anwendung bei einem maximalen Speicherbedarf von 1GB inklusive Auslagerungsdatei entweder in der Anwendung oder im Taskmanager manuell beendet.
Die Umsetzung eines dreidimensionalen, leistungsstarken und zudem echtzeitfähigen Arrays erwies sich als äußerst diffizil. Abgewandelte *ByteArrays* aus [Pau03a] zur Erzeugung dreidimensionaler Arrays hatten Programmabstürze durch Zugriffsverletzungen zur Folge.
Außerdem führte die Verwendung des zur Verfügung stehenden System-Arrays der Managed Extensions aus dem .NET Framework zur Beanspruchung von über 3MB Arbeitsspeicher für jeden Durchlauf eines einzelnen Erkennungsschrittes. Zwar gab der interne Garbage Collector zwischendurch nicht benötigte Speicherreserven frei, verlangsamte aber im Endeffekt nur den Anstieg des Speicherbedarfs. Auch eine gezwungene Ausführung des Garbage Collectors führte keinen sichtbaren Erfolg herbei.
Als dritte Alternative wurde die Verwendung der Multi-Arrays aus der boost-C++-Library , herunterladbar von [BCL06], getestet, allerdings mit ebenso geringem Erfolg.

Dieses Problem ließ sich bis zur Fertigstellung der Diplomarbeit nicht beseitigen. Das entwickelte System war bei Abgabe der Diplomarbeit deshalb leider nicht funktionstüchtig. Eine Testphase zur Erkennung der Gesten konnte nicht begonnen werden. Infolgedessen wurde auch keine Rückmeldungsfunktion mit OpenGL umgesetzt, da die Priorität auf der Lösung des vorher beschriebenen Problems lag.

# 7 Fazit und Ausblick

In Filmen wie Steven Spielbergs Minority Report werden uns fiktive Steuergeräte der Zukunft präsentiert. Tom Cruise bedient eine Art hochmoderner Polizeicomputer mithilfe von in Handschuhen integrierten Leuchtdioden. Aber was wie eine Fiktion aussieht, basiert auf einem Bedarf nach ergonomischen und effizienten Steuerungsmöglichkeiten. Viele Publikationen (s. z.B. [Fri05] oder [Kon03]) beweisen, dass die Nachfrage nach alternativen Bedienelementen steigt. [Coh06a] belegt zudem, dass sich im Bereich der Entwicklung innovativer und effizienter Bedienung eines Computers oder der Navigation in virtuellen Welten, vor allem in der Steuerung durch Gesten einiges getan hat und noch tun wird.

Im Rahmen dieser Arbeit wurde eine mögliche Alternative oder Idee eines Gestenerkennungssystems konzipiert, implementiert und vorgestellt. Theoretisch wurde gezeigt, dass das entwickelte Konzept durchaus umgesetzt werden kann. Geeignete Bildverarbeitungsmethoden, ressourcenfreundlich implementiert, ermöglichen das Erkennen des Markers und die Ermittlung seiner Position zur Laufzeit. Unter Verwendung eines geringen Zeitintervalls könnten gezeichnete Gesten mutmaßlich fehlerfrei erkannt werden. Praktisch konnte ein Ablauf der Gestenerkennung aufgrund der in *Kapitel 6* beschriebenen Probleme leider nicht vollständig implementiert werden.

Ausgehend von einer handelsüblichen Webcam können einzelne Frames aus einem Live-Stream extrahiert und mit verschiedenen Bildbearbeitungsmethoden zur Analyse vorbereitet werden. Das konzipierte System besitzt die grundsätzlichen Eigenschaften eines Serversystems und reagiert auf die Verbindungsversuche der einfachen Client-Anwendung.
Geeignete Methoden untersuchen zeitliche Verläufe von mit einem Marker in der Luft gezeichneten Gesten. Anschließend werden diese Gesten mit vereinbarten und in einer externen Datei konfigurierbaren Gestenkombinationen verglichen und verknüpfte Befehle ausgeführt. Diese Befehle liegen in Form von Wörtern oder Sätzen vor, die an die Client-Applikation als Bestätigung der identifizierten Geste dienen. Für eine bessere Übersicht sind in der entsprechenden Textdatei nur zehn Gesteneinträge vorgesehen. Mehr Einträge werden auch beim Einlesen nicht berücksichtigt.
Vorteil dieser Art der Gestendefinition ist sicherlich die im Gegensatz zu [Rub91] stehende, nicht benötigte Trainingsphase und die folgende Datenermittlung.

Auch wenn das System nur teilweise einwandfrei funktioniert und noch Mängel aufweist, bin ich mit den erreichten Erfolgen sehr zufrieden. Nachträglich möchte ich noch einige Verbesserungsvorschläge anführen, die in einer eventuellen Weiterentwicklung der Applikation mit einbezogen werden könnten.
Vordergründig sollte das Speichermanagment optimiert werden. Demzufolge ist eine praktische Testphase der Gestenerkennung möglich und lässt weitere Untersuchungen bezüglich der Erkennungsquote und Fehlertoleranz zu.

Eine weitere denkbare Verbesserung wäre eine lokale statt globale Suche des Markers. Wurde der Marker beim ersten Mal einwandfrei detektiert, könnte man ab dem zweiten gegrabbten Frame den Marker in der Umgebung seiner vorhergehenden Position suchen. Dies würde sicherlich eine beachtliche Geschwindigkeitsverbesserung herbeiführen Zudem könnte möglicherweise das in der Evaluierung beschriebene Problem der Beeinflussung detailreicher Bildbereiche abgefangen werden. Allerdings setzt dies eine einwandfreie Erkennung der ersten Markerposition voraus.
Des Weiteren wäre eine Plug & Play ähnliche Funktion wünschenswert. Beim erneuten Anschließen nach dem Entfernen der Webcam, wird diese vom Gesture Recognizer zur Programmlaufzeit nicht wieder erkannt. Gerade bei versehentlichem Herausziehen des USB-Kabels kann dies sehr ärgerlich sein.

Die im Test verwendete Webcam erwies sich stellenweise als ungeeignet. Bei Vorführungen beim Betreuer dieser Diplomarbeit erkannte das System, obwohl zuvor keine Änderungen am bis dahin funktionierenden Code vorgenommen wurden, eine blaue Stuhllehne fälschlicherweise als rotes Element. Außerdem reagierte die Webcam teilweise extrem lichtempfindlich.
In diesem Zusammenhang muss noch erwähnt werden, dass äußere Lichteinflüsse nicht berücksichtigt wurden. Erfahrungsgemäß können eingefärbte Lichtquellen, beispielsweise die Lampen in einem Fotolabor, ein anderes Farbempfinden hervorrufen und demnach die Farbwerte eines Objektes fälschen. Ein anderes Beispiel wären rote Leuchtkörper zur Alarmsignalisierung. Solche und ähnliche Lichtquellen beeinflussen das farbliche Erscheinungsbild erheblich.

Um die Eingabe falscher Regionengrößen abzufangen, bedarf es sicherlich noch einer geeigneten Prüfung der Werte in Abhängigkeit der eingestellten Auflösung der Kamera. Momentan ist noch die Eingabe nicht kompatibler Regionen möglich. Das Verhalten bei solch einer Eingabe konnte jedoch aufgrund der fehlenden Testphase durch vorherig beschriebene Probleme nicht untersucht werden.

Eine die Gesten betreffende Überlegung ist sicherlich die Frage des Umfangs. Zum einen wäre ein Algorithmus zur Erkennung der Anzahl, der in der Textdatei definierten Gesten außerordentlich vorteilhaft. Zum anderen könnte die Verwendung einer suchbaumähnlichen Struktur das Finden der Gesten beschleunigen
Eine weitere Betrachtung unter dem Aspekt des Umfangs wäre, wie viele Gesten kann sich ein durchschnittlicher Mensch, oder vielmehr ein durchschnittlicher Anwender, mit entsprechenden Funktionen merken? Zwar ermöglicht das entwickelte System die Vereinbarung eigener Gesten, allerdings benötigt man auch auf diesem Weg eine gewisse Einarbeitungszeit, bis die Gesten nicht mehr verwechselt und fehlerlos beherrscht werden.
Zusätzliche Überlegungen können hinsichtlich einer parallelen Erkennung der Gesten getätigt werden. Natürlich könnte man dies von vorneherein ermöglichen, indem man eine weitere Funktion implementiert, die zur Laufzeit die erkannten Richtungen mit bestehenden Gesten vergleicht. Dazu muss allerdings darauf geachtet werden, dass Zahlenkombinationen des Gestencodes keine gleichen Zahlenreihenfolgen besitzen. Diese Vorgehensweise wird jedoch als unakzeptabel erachtet.

Wird beim Entwurf einer Anwendung darauf geachtet, diese mit Client-Funktionen zu versehen, sollte die Einbindung ins konzipierte System kein Problem darstellen

Betrachtet man das konzipierte System aus medizinischer Sicht, könnte sich der involvierte Ball als Marker nachteilig auswirken. Während eines chirurgischen Eingriffs bietet der Marker zusätzliche, nicht immer vorteilihafte Eigenschaften. Sofern der Marker nicht sicher montiert ist, kann er sich vom Handschuh oder ähnlichem lösen und könnte schlimmstenfalls in die zu behandelnde Wunde geraten. Auch eine immer wieder erfolgende Desinfektion des Markers könnte als störend empfunden werden und die eventuell gewonnene Bewegungsfreiheit durch neu entstandene Abläufe nachteilig beeinflussen.

# 8 Literaturverzeichnis

**Bücher und Publikationen:**


[Fri05] Fritzer, Fabian <*Texteingabe per Augenbewegung*> [Diplomarbeit an der Universität Koblenz], Koblenz, November 2005

[HAJ90] Huang, X. D.; Ariki, Y.; Jack, M. A. <*Hidden Markov Models for Speech Recognition*>, Edinburgh, Edinburgh University Press, 1990.

[Kon03] Kondratyuk, Yevgeniya <*Verfolgung und Segmentierung von Händen in Bildsequenzen zur Unterstützung der 3D-Analyse in einem Echtzeit-Videokonferenzsystem*> [Diplomarbeit an der Technischen Universität Berlin], Januar 2003

[Lib99] Liberty, Jesse <*C++ in 21 Tagen*>, München, Markt und Technik Verlag, 1999, ISBN 3-8272-5624-0

[LX96] Lee, Christopher; Xu, Yangsheng <*Online, Interactive Learning of Gestures for Human/Robot Interfaces*>, Pittsburgh, 1996

[Nes86] Nespoulous, Jean-Luc; Perron, Paul; Lecours, André Roch <*The Biological Foundations of Gestures: Motor and Semiotic Aspects*>, Hillsdale, Lawrence Erlbaum Associates, 1986, ISBN 0898596459

[NoS87] Novak, C.L.; Shafer, S.A. <*Color Edge Detection*> in *Proceedings DARPA Image Understanding Workshop Volume 1* [S. 35 – 37], Los Angeles, Februar 1987

[Pau03a] Paulus, Dietrich W.R.; Hornegger, Joachim <*Applied Pattern Recognition: Algorithms and implementation in C++*>, [4. Edition], Wiesbaden/Braunschweig, Vieweg, 2003, ISBN 3-528-35558-1

[Paz03] Pazera, Ernest <*Focus on SDL*> [Ebook], Premier Press, August 2003, ISBN 1-59200-030-4

[Pes03] Pesce, Mark D. <*Programming Microsoft DirectShow for digital video and television*>, Redmond/Washington, Microsoft Press [Hrsg.], Februar 2003, ISBN 0-7356-1821-6

[RMG98] Raja, Yogesh; McKenna, Stephen J.; Gong, Shaogang <*Tracking and Segmenting People in Varying Lighting Conditions using Color*> in *Proceedings of FG`98*, Nara, 1998

[Rog92] Rogelberg, David <*OpenGL Reference Manual*>, Addison-Wesley Publishing Company [Hrsg.], November 1992, ISBN 0-201-63276-4

[Rub91] Rubine, Dean <*Specifying Gestures by Example*> in *International Conference on Computer Graphics and Interactive Techniques* [S. 329-337], Pittsburgh, ACM Press, 1991, ISSN 0097-8930

[Ste93] Steinbrecher, Rainer <*Bildverarbeitung in der Praxis*> [2. Auflage], München [u.a.], Oldenbourg Verlag, 1993, ISBN 3-486-22372-0
kostenloser Download: http://www.rst-software.de/dbv/download.html
Stand 01.03.06

[WH99] Wu, Ying; Huang, Thomas S. <*Vision-Based Gesture Recognition: A Review*>, Urbana, 1999

[WS00] Wright, Richard S.; Sweet, Michael <*OpenGL Super Bible*>, [2. Edition], Indianapolis, Waite Group Press, 2000, ISBN 1-57169-164-2

[YLW98] Yang, Lie; Lu, Weier; Waibel, Alex <*Skin-Color Modeling and Adaption*> in *Proceedings of ACCV`98 Volume 1* [S. 687-694], Pittsburgh, 1998

[YX94] Yang, Jie; Xu, Yangsheng <*Hidden Markov Model for Gesture Recognition*>, Pittsburgh, Mai 1994

**Skripte:**

[Ehr03] Ehrich, Sven <*Mathematische Methoden in der Bildverarbeitung*> [Skript zur Vorlesung Wintersemester 2003/2004], Universität München, 2003

[Fre05] Frese, Udo <*Echtzeit Bildverarbeitung*> [Skript zur Vorlesung], Universität Bremen, 2005

[Mül05] Müller, Stefan <*Computergraphik 1*> [Skript zur Vorlesung Sommersemester 2005], Universität Koblenz, 2005

[Pau01] Paulus, Dietrich W.R. <*Aktives Bildverstehen*> [Skript], Universität Erlangen, Oktober 2001

[Pau03b] Paulus, Dietrich W.R. <*Bildverarbeitung*> [Skript zur Vorlesung], Universität Koblenz, Februar 2003

[Reh99] Rehrmann, Volker <*Digitale Bildverarbeitung*> [Skript zur Vorlesung], Universität Koblenz, 1999/2000

**Weblinks:**


[App06] Apple Developer Connection: Quicktime
http://developer.apple.com/quicktime/
Stand 02.02.2006

[Boo06] boost C++ Libraries
http://www.boost.org
Stand 11.03.2006

[Coh06a] Cohen, Charles [Maintainer] <*The Gesture Recognition Homepage*>
http://www.cybernet.com/~ccohen/
Stand 17.02.2006

[Coh06b] Cohen, Charles [Maintainer] <*A brief overview of Gesture Recognition*>
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html
Stand 17.02.2006

[GSt06] GStreamer: Open Source Media Framework (offizielle Homepage)
http://gstreamer.freedesktop.org
Stand 02.02.2006

[MSD06] Microsoft MSDN Library: <*Introduction to DirectShow*>
http://msdn.microsoft.com/library/default.asp
Stand 25.01.2006

[NeJ05] Nerche, Jan; Reis, Andreas; Vortrag <*Linux und Multimedia*>, Dresden 2005
http://wwwbs.informatik.htw-dresden.de/svortrag/i01/Reis/vortrag/linuxmm.html
Stand 02.02.2006

[Phy06] Physik für Kids; Projekt der Universität Oldenburg
http://www.physikfuerkids.de/wiewas/touch/
Stand 01.02.2006

[SDL06] Kraus, Marco [Maintainer] <*Offizielle deutsche Webseite des "Simple DirectMedia Layer"*>
http://www.libsdl.de/
Stand 25.01.06

[VBa06] Otter, Dieter [Gründer]; VB@rchiv
http://www.vbarchiv.net
Stand 09.02.2006

[WA05] Willemer, Arnold; Galileo Computing [Hrsg.] <*Einstieg in C++*>
[Online-Ausgabe], April 2005,
ISBN 3-89842-649-1
http://www.willemer.de/informatik/cpp/index.htm
Stand: 01.03.2006

[WAC06] Internet-Seite der Wacom Europe GmbH
http://www.wacom-europe.com/
Stand 01.02.2006

[Wik06] Wikipedia: Die freie Enzyklopädie
http://de.wikipedia.org/wiki/
Stand 18.01.2006

[WJ06] Williamson, John <*Hidden Markov Model Gesture Recognition*>, Glasgow
http://www.dcs.gla.ac.uk/~jhw/gesturehmm.html
Stand 22.02.2006

# 9 Abbildungsverzeichnis

Abb. 2.1: kontinuierliches und diskretes Signal der Wortfolge „Guten Tag“ ..... 18
Abb. 2.2: OpenGL Rendering Pipeline..... 21
Abb. 2.3: Beispiel-Graph aus der DirectX-Anwendung Graphedit..... 24
Abb. 2.4: SDL-Einbindung..... 25
Abb. 3.1: Stufen eines Bilderkennungssystems..... 28
Abb. 3.2: RGB-Würfel ..... 31
Abb. 3.3: HSV-Pyramide..... 31
Abb. 3.4: Formeln zur Umrechnung von HSV-Pixelwerten in RGB-Farbwerte..... 32
Abb. 3.5: Formel zur Umrechnung vom RGB- in den HSV-Farbraum..... 32
Abb. 3.6: Ermittlung der YUV-Werte aus gegebenen RGB-Werten..... 33
Abb. 3.7: Maskendurchlauf im Bild ..... 34
Abb. 3.8: Umrechnung eines RGB-Pixel-Wertes in einen Grauwert ..... 35
Abb. 3.9: Mittelwertbildung eines Grauwert-Bildes ..... 35
Abb. 3.10: Mittelwert-Filter ..... 36
Abb. 3.11: modifizierter Mittelwert-Filter..... 36
Abb. 3.12: Formel des Mittelwert-Filters für die Größe von 3x3 Feldern ..... 36
Abb. 3.13: Minimal-Bild mit Rauschen..... 37
Abb. 3.14: Minimal-Bild nach Verwendung des Mittelwert-Filters..... 37
Abb. 3.15: Zahlenrepräsentation eines verrauschten Grauwert-Bildes..... 37
Abb. 3.16: Zahlenrepräsentation eines durch den Mittelwert-Filter geglätteten Bildes ... 37
Abb. 3.17: Gauss-Formel..... 37
Abb. 3.18: Pascal´sches Dreieck..... 37
Abb. 3.19: Gauss-Filter der Größe (3x3) und (5x5) ..... 38
Abb. 3.20: Grafische Darstellung der Gewichtungsverteilung einer 5x5 Gauss-Maske .. 38
Abb. 3.21: Minimal-Bild mit Rauschen..... 39
Abb. 3.22: Minimal-Bild nach Verwendung des Medianfilters ..... 39
Abb. 3.23: Zahlenrepräsentation eines verrauschten Grauwert-Bildes..... 39
Abb. 3.24: Zahlenrepräsentation eines durch den Medianfilter geglätteten Bildes ..... 39
Abb. 3.25: ungeordnete Werte unter der Maske..... 39
Abb. 3.26: sortierte "Rangordnung" ..... 39
Abb. 3.27: Formel k-trimmed-mean-Filter..... 39
Abb. 3.28: Formel Mid-Range-Filter..... 39
Abb. 3.29: Schwellwert ..... 40
Abb. 3.30: Histogramm ..... 40
Abb. 3.31: Beispielbild mit mehreren Objekten für ein Schwellwertverfahren ..... 41
Abb. 3.32: Originalbild mit leichtem Rauschen ..... 43
Abb. 3.33: Glättung eines Bildes..... 43
Abb. 3.34: Kantendetektion..... 43
Abb. 3.35: Konturaufbesserung..... 43
Abb. 3.36: Konturpunktverkettung..... 43
Abb. 3.37: Abbildungsbereich..... 44
Abb. 3.38: Richtung des Gradienten..... 44
Abb. 3.39: Berechnungsvorschrift des Gradienten-Vektors..... 44
Abb. 3.40: perfekte Kante mit optimalem Einstiegspunkt der Kantenerkennung ..... 45
Abb. 3.41: realistische Kante im Bild..... 45
Abb. 3.42: Grauwert-Intensitätsfunktion..... 45
Abb. 3.43: Intensitätsfunktion eines H-förmigen Objektes ..... 45
Abb. 3.44: horizontale Differenz..... 45
Abb. 3.45: vertikale Differenz..... 45

Abb. 3.46: Gradientenbetrag ... 46
Abb. 3.47: Gradientenrichtung ... 46
Abb. 3.48: Prewitt-Operator ... 46
Abb. 3.49: Roberts-Operator ... 46
Abb. 3.50: Kirsch-Operator ... 46
Abb. 3.51: horizontaler und vertikaler Sobel-Operator ... 47
Abb. 3.52: Gradientenbetrag des Sobel-Operators ... 47
Abb. 3.53: Gradientenrichtung des Sobel-Operators ... 47
Abb. 3.54: Originalbild und Sobel-Operator ohne Schwellwert ... 47
Abb. 3.55: Varianten des LaPlace-Operators ... 48
Abb. 3.56: horizontale LaPlace-Ableitung ... 48
Abb. 3.57: vertikale LaPlace-Ableitung ... 48
Abb. 3.58: Summe der diskreten LaPlace-Ableitungen ... 48
Abb. 3.59: Winkeltabelle der Non-Maxima-Suppression ... 49
Abb. 3.60: Bereichsdiagramm der NMS ... 49
Abb. 3.61: Masken zu vergleichender Konturpunkte ... 49
Abb. 3.62: Linie im Bildraum ... 50
Abb. 3.63: Auszug aus einem Akkumulator ... 50
Abb. 3.64: Geradengleichung ... 50
Abb. 3.65: Hess´sche Normalform ... 50
Abb. 3.66: Kreisformel ... 51
Abb. 3.67: Verdeutlichung der Berechnung zugehöriger Pixel eines Kantenpunktes ... 51
Abb. 3.68: Parameterraum für Kreisdetektion mit unbekanntem Radius ... 51
Abb. 3.69: Illustration zur Ermittlung des Kreismittelpunktes ... 51
Abb. 3.70: Pixelwerte eines kleinen Grauwertbildes ... 52
Abb. 3.71: zu Regionen zusammengefasste Pixel ... 52
Abb. 3.72: Berechnung der Maßzahl V ... 54
Abb. 3.73: Berechnungsvorschrift der Histogramm-Rückprojektion ... 54
Abb. 3.74: Beispiel Differenzbild durch Division ... 56
Abb. 4.1: drei Parameter in einer Geste ... 60
Abb. 4.2: drei Ausführungen einer Geste ... 61
Abb. 4.3: Berechnungsvorschriften der Features einer Geste nach Rubine ... 63
Abb. 4.4: Errechnen des maximalen Wertes zur Klassifizierung einer Geste ... 63
Abb. 4.5: Definition des HMM-Fünfertupels $(S,A,B,\Pi,V)$ ... 64
Abb. 4.6: Zustandsdiagramm des HMM ... 64
Abb. 4.7: multi-dimensionales Hidden-Markov-Modell ... 64
Abb. 4.8: Bedingung zur Identifikation der Geste ... 65
Abb. 4.9: Aufbau des HMM-basierten Gestenerkenners ... 65
Abb. 4.10: Methode zur Detektion statischer Gesten ... 67
Abb. 5.1: Kontur Würfel ... 69
Abb. 5.2: Kontur Kugel ... 69
Abb. 5.3: Eintrag im gleichen Akkumulatorfeld für mehrere Winkel ... 70
Abb. 5.4: alter Marker ... 71
Abb. 5.5: neuer Marker ... 71
Abb. 5.6: Programmoberfläche ... 72
Abb. 5.7: Filter-Graph der implementierten Anwendung ... 73
Abb. 5.8: Originalbild aus Live-Stream ... 75
Abb. 5.9: Überblick der Kontrollbilder ... 75
Abb. 5.10: *Config_ObjDet.txt, Config_Gestures.txt und Config_TimerGW.txt* ... 76
Abb. 5.11: Standard-Programmablauf ... 78
Abb. 5.12: Vorgang der Gesteninitialisierung und -erkennung ... 78
Abb. 5.13: Gestencode ... 80
Abb. 5.14: Gesten-Code-Matrix ... 80
Abb. 5.15: Beispiele zur Prüfung von Ausreißern. ... 82
Abb. 5.16: Beispiel einer Geste mit Regionenermittlung und Richtungsbestimmung ... 84

Abb. 5.17: vereinfachtes "Klassendiagramm" mit Include-Beziehungen..... 85
Abb. 5.18: Aufbau eines dynamischen Objektes der Klasse BitmapArray2D ..... 86
Abb. 5.19: Bilder nach NMS ..... 89
Abb. 6.1: Beispiel Markererkennung..... 95

# 10 Abkürzungsverzeichnis

- API: Application Programming Interface
- CCD: Charged Coupled Device
- CYMK: Cyan Yellow Magenta Black
- DLL: Dynamic Link Library
- FFT: Fast Fourier Transformation
- GDI: Graphics Device Interface
- GL: Graphics Library
- GRANDMA: Gesture Recognizers Automated in a Novel Direct Manipulation Architecture
- GUI: Graphical User Interface
- HMD: Head Mounted Display
- HSV: Hue Saturation Value
- LUT: Lookup Table
- NMS/A: Non Maxima Suppression/Absorption
- OpenAL: Open Source Audio Library
- OpenGL: Open Source Graphics Library
- PAL: Phase Alternating Line
- PDF: Portable Document Format
- RGB: Rot Grün Blau
- SDK: Software Development Kit
- SDL: Simple DirectMedia Layer
- STFT: Short-time Fourier Transformation
- STL: Standard Template Library
- USB: Universeller Serieller Bus
- VoIP: Voice over IP
- YUV: entwickeltes Farbmodell für das Farbfernsehen